Deep Reinforcement Learning - A short survey

Adrian Millea Abbas Edalat
$\{a.millea14, a.edalat\}$@imperial.ac.uk
Department of Computing, Imperial College London, SW7 2AZ, London

Abstract

Deep reinforcement learning has achieved significant results on Atari 2600 games. With no parameter tuning and no hand engineered features it learned how to play numerous games with almost human or, for some, better than human performance. However, each game was learnt from scratch and for some games, where temporal abstraction was needed, the agent performed poorly. We strive in this short survey paper to state the limitations and possible future direction for enhancing such agents. While still trying to provide a complete picture of the current research, we will investigate in more detail four important aspects, one is the *learning* mechanism, which is of critical importance for any DRL agent, the second one is existing approaches to *transfer learning* in the context of control, the thirs one is the *hierarchy* of representations (here we focus on the options framework), and the last one is the *exploration* policy or intrinsic motivation of the agent, which for large state spaces needs to be highly complex and adaptive, to enable the agent to visit unseen and meaningful states.

Keywords: **deep reinforcement learning, hierarchy, options, curiosity, transfer learning, intrinsic reward, successor representation, policy gradient, temporal abstraction, model-based RL, model-free RL**

1 Introduction

Deep reinforcement learning is a recent field of Machine Learning that combines two powerful techniques, the data-hungry deep learning and the older process oriented reinforcement learning (RL). Reinforcement learning took birth as a heuristic approach, descendant of dynamic programming. Sutton and Barto in their seminal work [Sutton and Barto, 1998] put the basis for a whole new field, which would turn out to be highly influential throughout the years, both in neuroscience and machine learning. In general, reinforcement learning was employed where the data was little and the behaviour was complex. However, recently, because of the advent of deep networks, reinforcement learning has received increased horsepower to tackle more challenging problems. The general reinforcement learning problem is defined by an agent acting, or making decisions in an environment, where the process is modelled as a Markov Decision Process (MDP). The MDP is represented by a state space $\mathcal{S}$ comprising the states the agent can be in, defined by the environment, the action space $\mathcal{A}$, which is the set of actions an agent can take, a transition dynamics which gives the probabilities that the agent has of being in a state at time t, taking an action and being in another state, at time $t+1$ i.e. $p(s_{t+1}|s_t, a_t)$, and a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. An MDP dynamics is assumed to satisfy the Markov property $p(s_{t+1}|s_1, a_1, ..., s_t, a_t) = p(s_{t+1}|s_t, a_t)$, that is the next state is dependent only on the previous state and action for any state-action space trajectory. The general goal of the agent is to *find a policy* which maximizes the discounted future reward, given by $R = \sum_{k=t}^{T} \gamma^{k-t} r_{s_k, a_k}$, with γ a real number called discount factor, with $0 < \gamma < 1$. The policy is modelled to be stochastic in general and is parameterized by a set of parameters θ, i.e. $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ is the set of probability measures on the action space $\mathcal{A}$. The reward is assumed to be given by the environment, however, we will see later that auxiliary reward functions (not given by the environment) can significantly improve an agent's performance. We will state briefly the main paradigms used in finding a good policy and then we will delve directly into the acclaimed DQN, the first DRL agent introduced in the machine learning community. We consider the partitioning of the RL algorithms following Silver [1]. The three approaches are: **value-based RL**, where the value in each state (and action) is a kind of prediction of the cumulative future reward. In short, each state has an associated

[1] `http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf`

real value number which defines how good it is to be in one state (or to take a specific action being in one state). This is given by the state-value function, usually denoted by $V^{\pi}(s)$, where π is the policy which governs the agent, and which has one identifier, namely the state, or the state-action value function given by $Q^{\pi}(s,a)$ which has two identifiers namely the state and action. We will see later that new flavours of the value function exist, where additional identifiers are used, for example the current goal of the agent (in the case where there are multiple goals). In DRL, the value functions are outputs of deep networks. The second approach is to represent the actual policy as a deep network, and this is referred to as **policy-based RL**. In general, the policy networks can be directly optimized with stochastic gradient descent (SGD). The third approach is **model-based RL**, where the agents specifically construct a transition model of the environment, which is generally modelled by a deep network. This is sometimes hard, depending on the complexity of the environment but offers some advantages over the model-free approaches, for example, the model is generative, meaning that the agent can generate samples from its model of the environment and thus can avoid actually interacting with the environment, and this can be highly beneficial when this interaction is expensive or risky. Having stated the main approaches we proceed to the description of the first DRL agent, often referred to as DQN (Deep Q Network) [Mnih et al., 2015]. In DQN the powerful deep networks were employed for approximating the optimal action-value function, i.e.:

$$Q^*(s,a) = \underset{\pi}{max} \mathbb{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... | s_t = s, a_t = a, \pi\right]$$

where r_t, a_t are the reward and action at timestep t, each future reward is discounted by a factor of γ and π is the policy that governs the agent's behaviour. For the rest of this document, if we do not specify with respect to what variables the expectation operator is considered, then it means it is with respect to the state variable. The iterative version is the given by:

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma \underset{a'}{max} Q^*(s_{t+1}, a') | s_t = s, a_t = a\right]$$

In general, in reinforcement learning, the optimal Bellman value or action-value function is denoted with an upper *. DQN was able to play a few tens of Atari games with no special tuning of hyper-parameters or engineered features. Learning just from pixel data, the agent was capable of learning to play all games with human or almost human performance. This is remarkable for an artificial agent. For the first time, an agent combined the representational power of deep nets with the RL's complex control, with two simple tricks. The first one is *replay memory*, which is a type of memory that stores that last million frames from the experience of the agent and avoids the highly correlated consecutive samples arising from direct experience. The second one was brought by replacing the supervised signal, i.e. the target values of the Q-network, or the optimal Q function from the Bellman equation with an approximation consisting of a previously saved version of the network, this is referred to as the *target network*. Using this approach defines a well-posed problem, avoiding the trouble of estimating the optimal value function. Then the loss function is given by (we use the notation used in the original DQN paper [Mnih et al., 2015]):

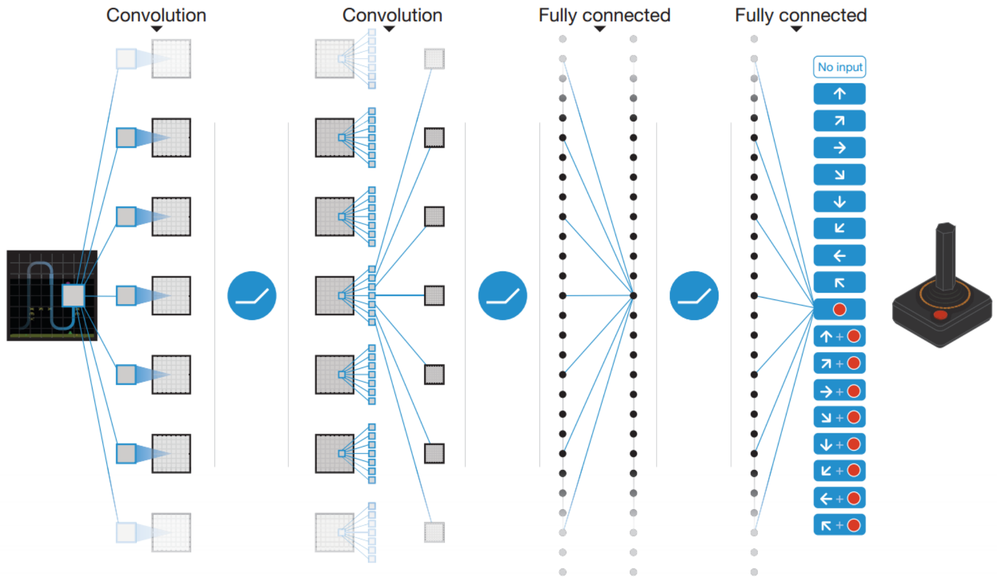
$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \underset{a'}{max} Q(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

where $U(D)$ is the uniform distribution giving the sample from replay memory, γ is the usual discount factor, and θ_i^- is the set of parameters from a previous iteration (this is the second enhancement we mentioned above). This gives the following gradient for the loss:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \underset{a'}{max} Q(s', a'; \theta_i^-) - Q(s,a;\theta_i)\right) \nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

These two main additions enabled unprecedented performance and adaptability of the deep net enabled agent to deal with a relatively diverse set of task. Different games, but still the same domain, Atari

Figure 1: The original architecture of the Deep Q Network that learnt to play Atari games from pixel data.



2600. The work that followed was immense, in less than two years the paper has accumulated over 700 citations. We will talk about some of the ideas that extend the current approach, but functionally and theoretically. We will not be discussing practical concerns, even though we are aware of the critical importance of such works and the fact that they effectively enable these algorithms to run in an acceptable time. We intend to investigate theoretical and algorithmic frameworks which enable higher level reasoning, or better adaptability, i.e. frameworks that add more abstraction and functionality to the DRL framework (e.g. adding some sort of hierarchy or memory, temporal abstraction, curiosity) or frameworks that enable transfer learning (i.e. learning a wider set of tasks without losing the old knowledge, moreover making use of it). We will also describe the different challenges DRL agents currently face and state briefly what other tasks they might face in the future. The goal of this work is to attract more people into DRL, which does not need to be neural network based, but it can make use of any underlying representation. We strongly believe that combining deep architectures with reinforcement learning has immense potential for general behavioural agents and in this work we investigate some of these approaches that tackle more datasets and try to implement scalable transfer learning. Deep learning is a relatively small subset of the whole machine learning community, even though it is progressing fast, at an unprecedented rate, due to the relative ease of use and ease of understanding but also due to the higher computational resources available. Being a visual task, CNNs perform best at playing Atari games, but other types of networks have been used, for example, for text based reinforcement learning as well [Narasimhan et al., 2015, Foerster et al., 2016]. More and more tasks will be amenable to DRL as the field embraces the new paradigm and the various existing approaches are adapted to fit these new understandings of what can be achieved.

We now proceed to describe the main different learning approaches used in DRL and some of their advantages and disadvantages. Following DQN many approaches have tried to improve various aspects of it, however, the general reinforcement learning community realized that any type of previously used method can be used with deep networks by enhancing it with the two main contributions of DQN (replay memory and target network). We show in 1 the much acclaimed architecture of the original DQN.

1.1 Overview

After stating the basics the original DQN architecture in the introduction, we proceed in Section 2 by describing the main *learning approaches* for RL but focusing on the works which find themselves in the Deep RL context. We state briefly the types of learning used in in the bigger context of RL with their principal characteristics and first describe the main notions involved in value-based RL with a couple of summaries describing some interesting papers exemplifying these notions. Then we discuss what we consider to be a few of the main approaches to policy-based DRL and then we give again a couple of summaries of papers exemplifying the model-based approach. We continue in Section 3 with what we considered to be a few of the most interesting approaches to *transfer learning* and then in Section 4 we discuss *hierarchical reinforcement learning*, having a brief overview of findings in neuroscience and then focusing on the options framework and their most important characteristic: temporal abstraction. In the Section 5 we focus on what it means to structurally decompose the learning algorithm as a function of states, actions and rewards and what approaches exist for such decompositions. The bulk of the section consists of various directions concerning *intrinsic rewards*, rewards not given explicitly by the environment.

We need to mention here that throughout this document we will focus more on the algorithmic and theoretical aspects of the works presented and less on the practical aspects (e.g. performance results). We do this because we consider the main purpose of the document to be the presentation of some of the most interesting and promising approaches to the different problems tackled and not the actual performance of each one. Quite often performance is not correlated with how fundamental a research paper is, and how many new and important concepts it brings into the field.

Most of the time we will summarize individual papers in their own subsection, however sometimes we combine just a few ideas or conclusions from many papers into one or more paragraphs as the topic investigated requests it.

2 Learning

2.1 Value-based RL

As mentioned briefly in the introduction, value-based RL is used in the DQN agent. It is a model-free architecture, where pairs of states and actions are associated with real valued numbers. This association is a consequence of learning from experience, i.e. interacting with the environment. However, another flavour of the value-based representation is using just the states as identifiers, i.e. state-value function.

2.1.1 State value function

Generally the state value function is denoted by V^{π}, where, as before, π is the policy that governs the agent. It has a set of parameters θ which are learned during training to maximize future reward by mapping states (the states are usually very high-dimensional, that is where the power of deep networks comes into play) to real values, independent of the action chosen. Thus the agent will try to get to the states which have the maximum value of V^{π}. Generally this approach by itself is somehow obsolete now with the increasing computational capabilities, as using the V function instead of the Q function would provide a somewhat coarser internal representation for the agent. However, we will see that it is still useful to model the V-function as this can be considered to provide a measure of the contribution of the state to the final Q-value. Another use for the V-function is when considering successor representations. We will see later, that it is very useful to decompose the outcome of a state-action pair into specific state-contribution and action-contribution. Some newer approaches output the policy and the V function from the same network [Wang et al., 2016]. The general form of the optimal Bellman equation using the state-value function looks like:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left\{R_t | s_t = s\right\} \\ &= \underset{a}{max}\mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right] \end{aligned}$$

The relation between the optimal state value function V and the optimal state-action value function Q is then given by:

$$Q^*(s,a) = \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\right]$$

2.1.2 State-action value function

We mentioned already the Bellman equation for the state-action value function used in DQN. We saw that one of the main contributions of the DQN is the fact that the target value (the approximation of the optimal Q-function) for the Q-network is actually an older version of it. This removes some stability issues and improves convergence. We show next two very simple ways to improve over the original DQN.

Deep Reinforcement Learning with Double Q-Learning [Van Hasselt et al., 2015a]

In [Van Hasselt et al., 2015a] the authors show that this old version of the network can be updated as well, to remove the bias introduced by the overoptimism of the Q-learning. The main idea used is to decompose the maximum operation of the target value into two parts: action selection and action evaluation. Because the target network is different than the online network, we can use the former for evaluation. Then the update for the target becomes:

$$Y_t^{DDQN} = r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{argmax} Q(s_{t+1}, a; \theta_t), \theta_t^-)$$

The rest of the algorithm remains the same. This simple modification significantly improves performance of the DDQN on the Atari domain. There are quite a few works that improve on the original DQN performance, however we are interested to outline a comprehensive list of algorithmical and theoretical approaches, without going too much into the small details, thus we focus on works that bring forward new notions or perspectives.

Learning to play in a day: Faster deep reinforcement learning by optimality tightening [He et al., 2016]

A very powerful and relatively simple approach which improves training time significantly as well as accuracy considers a simple set of inequalities which is used to upper and lower bound the Q function. Through some quadratic penalty added to the loss function, the bounds improve training speed significantly by faster reward propagation. These are based on the following observation:

$$Q^*(s_j, a_j) = r_j + \gamma \underset{a}{max} Q^*(s_{j+1}, a) \geq ... \geq \sum_{i=0}^{k} r_{j+i} + \gamma^{k+1} \underset{a}{max} Q^*(s_{j+k+1}, a) = L_{j,k}^*$$

This is referred to as the lower bound for sample j and time horizon k. Similar for the upper bounds, changing variables as $j = j - k - 1$ and dropping the max operator. The largest lower bound and smallest upper bound are considered, where instead of the optimal Q-function, the standard target used in DQN is used Q_{θ^-}. The new loss function then becomes:

$$\underset{\theta}{min} \sum_{s_j, a_j, r_j, s_{j+1} \in \mathcal{B}} \left[(Q_\theta(s_j, a_j) - y_j)^2 + \lambda(L_j^{max} - Q_\theta(s_j, a_j))_+^2 + \lambda(Q_\theta(s_j, a_j) - U_j^{min})_+^2\right]$$

where λ is the penalty coefficient and $(x)_+ = max(0, x)$ is the ReLU function. The authors also augment the replay memory with the real cumulative discounted return over the episode, which is added to memory after the episode has ended, for efficiently computing the discounted reward over multiple time steps. The results of this loss procedure are highly significant. Performance is state of the art, while the frames needed are improved with a factor of 20 compared to previous approaches (DQN, DDQN).

2.1.3 Advantage function

Another flavour of value-functions is modelling the difference between the Q-function and the V-function, basically splitting the network into two components, the action-independent V-function and action dependent Q-function and then looking at the so called advantage function, introduced in [Baird III, 1993], given by:

$$Q^{\pi}(s,a) = V^{\pi}(s) + A^{\pi}(s,a)$$

Advantage and state updates have been shown to converge faster than the usual state-action updates. However from a unique Q-value associated with some action, there are multiple V and A values that satisfie this equality, thus the problem is unidentifiable. A solution to this, proposed in [Wang et al., 2015] is to output V and A from (almost) the same neural network and then force zero advantage at the chosen action. Denote by $V(s;\theta;\beta)$ the scalar (repeated to form a vector the same size as $|\mathcal{A}|$) output of the V-function parameterized by common (with A) parameters θ and particular (just to V) parameters β, and by $A(s,a,\theta,\alpha)$ the vector valued output with α its particular parameters. Thus the new equation for Q becomes:

$$Q(s,a,\theta,\alpha,\beta) = V(s,\theta,\beta) + (A(s,a,\theta,\alpha) - \underset{a' \in |\mathcal{A}|}{max} A(s,a',\theta,\alpha))$$

The authors replace the max operator with an average and they report improved stability of the optimization and better performance. Another important notion related to the advantage function is the response function given by:

$$\chi(l;,s_t,a_t) = \mathbb{E}\left[r_{t+l}|s_t,a_t\right] - \mathbb{E}\left[r_{t+l}|s_t\right]$$

The response function decomposes the advantage function over timesteps. We will show in the next section a context in which this is used, that uses both advantage estimation and policy gradients [Schulman et al., 2015b]. Having gone through the main concepts in value-based RL we now proceed to the more popular approach (especially in the control community) of policy-based RL.

2.2 Policy-based RL

In policy-based techniques, the policy function $\pi(a|s,\theta)$ is parameterized by a deep network with weights θ, which tries to maximize the the total future discounted reward, as before. The optimization procedure is generally using the ubiquitous SGD. When considering stochastic policies, the gradient of the total discounted reward with respect to the parameters is given by:

$$\frac{\partial L}{\partial \theta} = \mathbb{E}\left[\frac{\partial \log \pi(a|s,\theta)}{\partial \theta} Q^{\pi}(s,a)\right]$$

while for deterministic policies the gradient is given by:

$$\frac{\partial L}{\partial \theta} = \mathbb{E}\left[\frac{\partial Q^{\pi}(s,a)}{\partial a}\frac{\partial a}{\partial \theta}\right]$$

Deterministic policy-based methods can be used off-the-shelf when the set of actions is continuous and the Q-function is differentiable. Moreover, general policy based methods have local convergence guarantees and are effective in high-dimensional states and actions. We mention some of the approaches used in the DRL literature, but we note that the general policy search literature is much wider [Deisenroth et al., 2013]. For the general context and the applicability to DRL see the recent presentation from NIPS 2016[2]. Some of the disadvantages are: when evaluating the policy there is high variance (this is applicable for all policy-based RL, but is more pronounced for stochastic policies;

[2]`https://people.eecs.berkeley.edu/~pabbeel/nips-tutorial-policy-optimization-Schulman-Abbeel.pdf`

however we will see later there are a number of techniques for variance reduction, this is still an open problem) and they usually converge to a local minimum rather than global. Policy gradients are based on a fundamental result by [Sutton et al., 1999a], called *the policy gradient theorem*, which says that:

$$\begin{aligned}\nabla_\theta L(\pi_\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a) da ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\right]\end{aligned}$$

We can note here that the policy gradient does not depend on the gradient of the state distribution ($\rho^\pi(s)$), even though the state distribution does depend on the policy. This means that the gradient computation reduces to the estimation of this expectation, which can easily employ sample based techniques. An important aspect of policy gradient is the estimation of the Q-function, which, if one uses the sample return, reduces to the REINFORCE algorithm [Williams, 1992]. An often used architecture using the above theorem is the actor-critic. In short this uses two agents, one that adjusts the parameters of the policy (actor) and one that estimates the action-value function Q using policy evaluation (critic). If one uses a deep network to approximate the Q-function, with parameters w, this introduces bias, however there are some conditions to avoid this. To avoid bias one needs a compatible function approximation i.e. $Q^w(s,a) = \nabla_\theta \log \pi_\theta(a,s)^T w$ and the parameters w of the network should be chosen such that they minimize the mean-squared error between the Q function estimation and the true Q function (more details in the next section). Stochastic policy gradient is more often encountered in the RL literature and there are many flavours of it. To show this more explicitly, following [Schulman et al., 2015b] we show an interpretation which uses a general variable ψ and the different values that ψ can take:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

where there the following are some choices for ψ_t

- $\sum_{t=0}^{\infty} r_t$ total reward of a trajectory
- $\sum_{t=t'}^{\infty} r'_t$ total reward following a specific action a'
- $\sum_{t=t'}^{\infty} r'_t - b(s_t)$ the same as above but with a baseline $b(s_t)$ subtracted from the reward. The baseline is often chosen to be a value function estimate
- $Q^\pi(s_t, a_t)$ state-action value function
- $A^\pi(s_t, a_t)$ advantage function
- $r_t + V^\pi(s_{t+1} - V^\pi(s_t))$ temporal difference residual

In [Schulman et al., 2015b] they use a version of the advantage function for ψ_t for the policy gradient. This is given by:

$$\widehat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

where δ_t^V is the discounted temporal difference residual as above: $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. Another important aspect put forward by this work, is to consider γ as an algorithm parameter that adjusts the bias-variance trade-off and not as is usually considered, i.e. the discount factor. The λ parameter defines the exponentially weighted average of $\widehat{A}_t^{GAE(\gamma,\lambda)}$ as:

$$\widehat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)\left(\delta_t^V(\frac{1}{1-\lambda}) + \delta_{t+1}^V(\frac{\lambda}{1-\lambda}) + \delta_{t+2}^V(\frac{\lambda^2}{1-\lambda}) + ...\right)$$

This method is then combined with trust region policy optimization [Schulman et al., 2015a] and applied successfully to a set of control tasks not using any hand crafted policy features (as was done before), but directly from the kinematics. Other significant contributions of this work are a way of controlling the bias-variance through the newly introduced parameters λ and the significant reduction in training samples needed for good performance.

2.2.1 Deterministic Policy Gradient

As opposed to the more widely used stochastic policy gradient, deterministic policy gradient (DPG) [Silver et al., 2014] considers a deterministic parameterized policy $\mu_\theta(s) = a$. It can be shown that the standard stochastic policy gradient converges in the limit to the deterministic one. One of the main advantages of DPG is that it has been shown to converge much faster to a good policy for high-dimensional action sets. The stochastic policy gradient theorem given above is extended to deterministic policies. DPG is actor-critic as one cannot explore if the behavioural policy is the target policy, the latter being deterministic as we said. We give next one of the main results for DPG (the analogous of the stochastic policy theorem), for a policy μ:

$$\begin{aligned}\nabla_\theta L(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right]\end{aligned}$$

We note that there are some conditions which the MDP needs to satisfy, given in the appendix of the original paper, which imply that the two gradients exist. The compatible function approximation is of high importance for preserving the true gradient. The idea is extended to the deterministic policies as follows:
Theorem (Theorem 3 in [Silver et al., 2014])
A function approximator $Q^w(s,a)$ is compatible with a deterministic policy $\mu_\theta(s)$, with $\nabla_\theta L_\beta(\theta) = \mathbb{E}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^w(s,a) | a = \mu_\theta(s)\right]$ if

$$\begin{aligned}&1. \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} = \nabla_\theta \mu_\theta(s)^T w \text{ and} \\ &2. w \text{ minimizes the mean squared error, } MSE(\theta,w) = \mathbb{E}\left[\epsilon(s;\theta,w)^T \epsilon(s;\theta,w)\right] \\ &\quad \text{where } \epsilon(s;\theta,w) = \nabla_a Q^w(s,a)|_{a=\mu_\theta(s)} - \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\end{aligned}$$

where β is the behavioural policy and Q^μ is the true state-action value function. Now, if $V^v(s)$ is the action independent differentiable value function parameterized by v, with, for example, $V^v(s) = v^t\phi(s)$ (with $\phi(s)$ are state features) then there always exists a compatible function approximator:

$$Q^w(s,a) = (a - \mu_\theta(s))^T \nabla_\theta \mu_\theta(s)^T w + V^v(s)$$

The first term in the above equation can be seen as the advantage of taking action a over the action given by the policy $\mu_\theta(s)$. If we consider the state-action feature as $\phi(s,a) = \nabla_\theta \mu_\theta(s)(a - \mu_\theta(s))$ then the advantage function can be written as $A^w(s,a) = \phi(s,a)^T w$. An advantage function linear in the features and parameters is useful just as a local critic, i.e. considering a small deviation (δ) from the policy $A^w(s, \mu_\theta(s) + \delta) = \delta^T \nabla_\theta \mu_\theta(s)^T w$ we can then know the direction of changing the policy parameters. Condition 2 of the theorem can be seen as a linear regression problem, for which the targets are $\nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}$ and the features are $\phi(s,a)$. In practice a linear function approximator is used ($Q^w(s,a) = \phi(s,a)^T w$) for condition 1. but condition 2. is not generally satisfied using Sarsa or Q-learning to learn w. Many versions of the deterministic policy gradient algorithm are provided in the original research, however we focus on the last one (the most complex in terms of number of equations used, and the last one presented in the original paper). We state shortly the general outline. This last algorithm is called COPDAC-GQ (Compatible Off-Policy Deterministic Actor Critic Gradient Q-learning) and has two components: the critic, which is a linear function approximator of state-action features ($\phi(s,a) = a^T \nabla_\theta \mu_\theta(s)$) and estimates the state-action values. This is learnt off-policy from the samples of another behavioural policy $\beta(a|s)$. The second component, the actor, updates its parameters in the direction given by the critic. The full COPDAC-GQ is given below.

$$\begin{aligned}
\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\
\theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t)(\nabla_\theta \mu_\theta(s_t)^T w_t) \\
w_{t+1} &= w_t + \alpha_w \delta_t \phi(s_t, a_t) - \alpha_w \gamma \phi(s_{t+1}, \mu_\theta(s_{t+1}))(\phi(s_t, a_t)^T u_t) \\
v_{t+1} &= v_t + \alpha_v \delta_t \phi(s_t) - \alpha_v \gamma \phi(s_{t+1})(\phi(s_t, a_t)^T u_t) \\
u_{t+1} &= u_t + \alpha_u (\delta_t - \phi(s_t, a_t)^T u_t)\phi(s_t, a_t)
\end{aligned}$$

It is very interesting to see that the natural policy gradient [Kakade, 2002, Peters et al., 2005] can be extended to deterministic policies. The natural gradient is given by $M(\theta)^{-1}\nabla_\theta L(\mu_\theta)$, where M is the metric chosen, here M^{-1} is the inverse of the Fisher metric (which is a matrix) given by:

$$M(\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi^\theta}\left[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T\right]$$

The metric used here is $\mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_\theta \mu_\theta(s)^T\right]$, which is the limiting case of the Fisher metric when variance is reduced to 0. The gradient policy theorem with compatible function approximation gives $\nabla_\theta L(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_\theta \mu_\theta(s)^T w\right]$. This finally gives: $M(\theta)^{-1}\nabla_\theta L(\mu_\theta) = w$. In the original work policies with up to 50 action dimensions were learnt from pixel data and also 50 state dimension and 20 action dimensions. This is very promising work and avoids the hard integral over the action space particular to stochastic policies making the gradient estimation much simpler. Another flavour of the DPG is a recurrent version, where memory is integrated into the system as an RNN. This was chosen with the goal of solving Partially Observable MDPs (POMDPs) and assumes the dependency of the current state upon many (or all) of the previous states which are summarized in the activations of the RNN. Thus, the gradient of the cost function with respect to the parameters becomes:

$$\frac{\partial L}{\partial \theta} = \mathbb{E}_\tau\left[\sum_t \gamma^{t-1} \frac{\partial Q^\mu(h_t, a)}{\partial a}|_{a=\mu_\theta(h_t)} \frac{\partial \mu_\theta(h_t)}{\partial \theta}\right]$$

where h_t are the states of the RNN that summarize the previous observed states and actions given by τ. This new version of DPG is referred to as RDPG and among the described difference it includes soft updates of the target network, uses the Adam optimizer for updating the actor and critic and employs backpropagation through time (BPTT) for computing the gradients for the RNN. This version of DPG is shown to perform much better for partially observable tasks than its purely feedforward counterpart. In general the memory is considered to be particular to the policy, however here is considered as extra state dimensions which can be used by the policy. This avoids explicit trajectory optimization and access to a well-defined latent space. See the original work for more details [Heess et al., 2015]. Naturally, there is also a deep version of the deterministic policy gradient (DDPG) [Lillicrap et al., 2015].

Following a somewhat different direction to policy optimization, the next section describes a procedure which is very similar to the natural gradient methods.

2.2.2 Trust region policy optimization (TRPO) [Schulman et al., 2015a]

This research describes an iterative procedure which applies to neural networks policy representation and guarantees monotonic improvement. Even though many practical approximations are employed this method has robust performance in a wide variety of control tasks, also necessitating little tuning of hyperparameters. Considering a stochastic policy π, its expected discounted reward is given by:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right] \text{ with}$$

$$s_0 \sim \rho_0(s_0), a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$

The definitions for the state-value function, state-action-value function are given by:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right]$$
$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \ldots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right]$$

and with the advantage function given by Equation **??**. Now, according to [Kakade and Langford, 2002] the expected return of a policy $\tilde{\pi}$ can be expressed in terms of the advantage of a policy π, over timesteps:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \ldots, \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

It is interesting to see that this construction is used in the derivation of a theoretical algorithm which then is discarded when considering the practical one. We see that the sum over timesteps in the above equation can be replaced with a sum over states as:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

where $\rho(\pi)$ is the state visitation frequency (normalized and discounted). Then, introducing the local approximation to η by ignoring the changes induced by the new policy in the state visitation frequency gives:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

It can be shown that $L(\pi)$ and $\eta(\pi)$ match up to the first order for any differentiable (with respect to the parameters) policy π. This reasoning lead to specific bounds derived in [Kakade and Langford, 2002] for the improvement of mixture policies depending on a constant $\alpha << 1$, which were extended by the current work to general stochastic policies. The method used to extend the bounds replaced the constant α with a distance measure between two policies, called *the total variation divergence*, given by:

$$D_{TV}(p||q) = \frac{1}{2} \sum_i |p_i - q_i|$$

Taking the maximum over the states from this divergence, and replacing α with the resulting quantity, gives the same bound as the original work, but for any stochastic policy. There are given two proofs for this, one using perturbation theory which improves slightly on the original bound. The divergence can be further bounded by [Pollard, 2000]: $D_{TV}(p||q)^2 \leq D_{KL}(p||q)$ to give rise to an approximate policy iteration algorithm. It is guaranteed that such an algorithm will be non-decreasing with respect to the true objective. We are not focusing on the theoretical details in this work, but we mention that the exposition is quite interesting, and direct the reader to the original work [Schulman et al., 2015a]. Even if the theory provides some specific steps to surely improve on the old value, in practice these steps are to small and the authors consider a practical alternative. Instead of taking small steps they use a constraint onto the KL divergence between the two policies, or a so called *trust region* constraint i.e.:

$$\underset{\theta}{\text{maximize}} L_{\theta_{old}}(\theta)$$
$$\text{such that } D_{KL}^{max}(\theta_{old}, \theta) \leq \delta$$

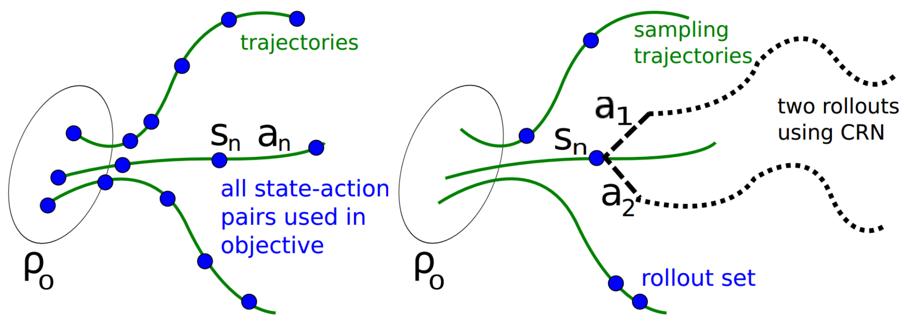


Figure 2: Single path sampling procedure (left) and Vine sampling (right). See text for details. Figure from [Schulman et al., 2015a]

.

which is impractical, thus the authors choose a heuristic approach instead, minimize the average KL-divergence: $\tilde{D}^{\rho}_{KL}(\theta_1,\theta_2) = \mathbb{E}_{s\sim\rho}\left[D_{KL}(\pi_{\theta_1}(\cdot|s)||\pi_{\theta_2}(\cdot|s))\right]$. For clarity we unfold the loss used and then state the practical considerations stated in the paper that modify each term:

$$\underset{\theta}{\text{maximize}} \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s|a)$$
$$\text{such that } \tilde{D}^{\rho}_{KL}(\theta_{old},\theta) \le \delta$$

The individual terms are replaced as follows: the sum over states $\sum_s \rho_{\theta_{old}}(s)[...]$ is replaced with $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{old}}}[...]$, the advantage values are replaced by the Q-values, noticing that the objective just changes by a constant by doing this, while the sum over action is replaced with an importance sampler, governed by distribution q. Then a single state s_n contributes to the objective the following term:

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{old}}(s_n,a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n,a)\right]$$

As we said, replacing the advantage function with the Q-function (actually an empirical estimate), and replacing all expectations with sample averages gives the final formulation of the optimization problem. Two sampling schemes are used, shown in Figure 2.2.2 previously used in the context of policy gradient (the first one) and policy iteration (the second one). In short, the first one, referred to as *single path*, samples $s_0 \sim \rho_0$ and then simulates the policy $\pi_{\theta_{old}}$ for a number of timesteps to generate a trajectory and the Q-function is computed for each state-action pair along the trajectory. The second one, referred to as *vine* generates some trajectories then chooses a subset of states along these (rollout set), and then samples a number of actions according to q. The Q-function is then estimated based on these samples and its variance is reduced between samples by using the *common random numbers* technique [Ng and Jordan, 2000]. The final optimization uses the conjugate gradient followed by a line search. The generality of this method is shown in the experimental section through learning on classic control tasks as well as Atari games. This work uses quite a lot of different techniques and connections to prior work and is in the authors' opinion one of the most interesting approaches to policy optimization. Having mentioned some of the most interesting and fruitful approaches to policy optimization, which is one of the most popular optimization strategies for DRL, either in combination with value-based RL or by itself, we proceed now to one of the less used methods for DRL, i.e. model-based DRL.

2.3 Model-based RL

In general, model-based methods are more data efficient, or have lower sample complexity, but are more computationally intensive. There are many approaches in the general RL literature, both parametric and non-parametric, most notably [Deisenroth and Rasmussen, 2011, Abdolmaleki et al., 2015, Levine and Koltun, 2013] but we are interested to describe in this work just the few approaches used in deep RL. Both model-free and model-based methods are known to exist in the brain [Littman, 2015], however the cognitive sciences differentiate between the type of task which employs each. Model-based decision making is shown to be used when the task is outcome-based, so one is interested in the unfolding in time of the sequence of states and actions (and sometimes rewards), while model-free methods are used when the decision making is mostly concerned with the value of an action (e.g. moral judgements, this action is good because it is moral). For DRL however, model-based approaches are quite scarce, probably due to the high complexity of models that need to be learned and the variety of models needed for the tasks that DRL is usually applied to. For example, TRPO deals with a number of classical control tasks and Atari games as well, it would be quite hard to derive a model building technique easily applicable to both of these domains. Even the Atari games are so different from one another, that a general model-building algorithm for all of them (the 50 games used more often in the DRL literature) would not be trivial. We proceed next to summarize some of the most interesting approaches to model-based DRL, that even though do not deal with the Atari domain, they use pixel information for learning.

2.3.1 Continuous Deep Q-Learning with Model-based Acceleration [Gu et al., 2016]

This work is generated by the need to speed up model-free reinforcement learning considering the significant higher needs of samples (also known as sample complexity) compared to model-based approaches. In high dimensional physical systems, the sample complexity needs to be low to reach good policies in reasonable time. Thus, the authors combine (continuous) Q-learning with learned local models (around samples) to speed up learning, while also making use of the model-free benefits. Another interesting aspect brought forward by this work is the avoidance of two estimators (as is the case with actor-critic methods), the one for the value function and the one for the policy. This is replaced with a single neural network that outputs both the value function and the policy function (as we have seen previously, this is quite an efficient way of combining value-based and policy-based RL). The main idea of combining model-based and model-free methods is to use local linear models that make use of imagination rollouts, that is sampling the model around samples, which can be seen as pretraining the value-function. The method used for iteratively locally linearizing is the iLQG, which assumes the following dynamics, with quadratic rewards:

$$\widehat{p}(x_{t+1}|x_t, u_t) = \mathcal{N}(f_{x_t}x_t + f_{u_t}u_t, F_t)$$

where x are states, u are actions and $\widehat{x}$ denotes an approximation. This makes the action-state value function and state value function quadratic and computable with dynamic programming. The optimal policy can then be obtained analytically and is given by:

$$g(x_t) = \widehat{u_t} + k_t + K_t(x_t - \widehat{x_t})$$

where k_t is the open-loop term, K_t is the closed-loop feedback matrix, while $\widehat{x_t}$ and $\widehat{u_t}$ are states and actions of the average trajectory of the controller. A very important idea put forward next, allows for analytic maximization, instead of the standard expensive *max* operator used in the greedy selection of actions. The network outputs two terms, a value function term $V(x)$ and an advantage term $A(x,u)$ that is modelled as a quadratic function of the state features:

$$Q(x,u|\theta^Q) = A(x,u|\theta^A) + V(x|\theta^V)A(x,u|\theta^A) = -\frac{1}{2}(u - \mu(x|\theta^\mu))P(x|\theta^P)(u - \mu(x|\theta^\mu))$$

where $P(x|\theta^P)$ is a positive-definite square matrix, which is state dependent and is given by $P(x|\theta^P) = L(x|\theta^P)L(x|\theta^P)^T$, with L being a lower-triangular matrix with entries from the linear output layer of a neural network and diagonal terms exponentiated. Even though is is more restrictive than a standard neural network, the maximum value of the Q-function is always given by $\mu(x|\theta^\mu)$. The learned model can then be used to guide exploration and the trajectories are mixed with on-policy samples by adding them to the replay memory. Even if planning under the exact model, the improvements are small, because the agent needs also bad samples to make the difference between good and bad samples. The model-based samples are combined with on-policy rollouts, with mixing coefficients such that additional synthetic rollouts are generated from each state along the real-world rollouts. Even though this is a powerful approach, which augments significantly the speed at which the agents learns starting from the random policy, when reaching a certain performance, the imagination rollouts become useless, thus, the authors switch off the imagination rollouts after a number of iterations. This makes this process very similar to pretraining in neural networks, where fine-tuning comes as learning from real-world experience with model-free learning. The local models are learned around the latest set of samples, and are given by:

$$p_t(x_{t+1}|x_t, u_t) = \mathcal{N}(F_t[x_t; u_t] + f_t, N_t)$$

The parameters of the models F_t, f_t and N_t are refitted every n episodes by a Gaussian distribution at each timestep to $[x_t^i; u_t^i; x_{t+1}^i]$, where i is the index of the sample. This approach yields significant performance gains in terms of sample efficiency. As we said, this is not used on the Atari domain, but nevertheless uses pixel data for learning to control using the MuJoCo simulator [Todorov et al., 2012], an often used simulator in the DRL literature. We now present a very interesting line of research, where the problem is formulated from an optimal control problem in the latent space, constrained to be locally linear and that supports long term prediction of sequences of images.

2.3.2 Embed to control: A locally linear latent dynamics model for control from Raw Images [Watter et al., 2015]

This work is in the same line of research as the previous one but spawned by the control and robotics literature and not the DQN (this can be seen usually through the experiments performed) literature, however the input is still very high dimensional, i.e. raw pixels, as the title suggests. However, in contrast to the previous work, and in general the DQN literature, this research does not use reinforcement learning in any way, but formulates the problem as is done in the control literature, assuming a latent space $\mathbf{z}$ and a controller $\mathbf{u}_{1:T}$ which minimizes the cost given by:

$$J(\mathbf{z}_{1:T}, \mathbf{u}_{1:T}) = \mathbb{E}_{\mathbf{z}}\left[c_T(\mathbf{z}_T, \mathbf{u}_T + \sum_{t_0}^{T-1} c(\mathbf{z}_t, \mathbf{u}_t))\right]$$

where $c(\mathbf{z}_t, \mathbf{u}_t))$ are the instantaneous costs, with T being the final timestep (c_T associated final cost). The observation model (relation between $\mathbf{s}$ and $\mathbf{z}$) is assumed to be smooth and arbitrary, and this is also to be discovered or learned, which makes the problem also a system identification task as well. The latent space is assumed to come from a linear dynamics:

$$\mathbf{z}_{t+1} = A(\widehat{\mathbf{z}}_t)\mathbf{z}_t + B(\widehat{\mathbf{z}}_t)\mathbf{u}_t + \mathbf{o}(\widehat{\mathbf{z}}_t) + \omega, \text{ with } \omega \sim \mathcal{N}(0, \sigma_\omega)$$

with $A(\widehat{\mathbf{z}}_t) = \frac{\partial f^{lat}(\widehat{\mathbf{z}}_t, \widehat{\mathbf{u}}_t)}{\partial \widehat{\mathbf{z}}_t}$ and $B(\widehat{\mathbf{z}}_t) = \frac{\partial f^{lat}(\widehat{\mathbf{z}}_t, \widehat{\mathbf{u}}_t)}{\partial \widehat{\mathbf{u}}_t}$ are local Jacobians and $\mathbf{o}(\widehat{\mathbf{z}}_t)$ being an offset. The function f^{lat} gives the transition dynamics in the latent space: $\mathbf{z_{t+1}} = f^{lat}(\mathbf{z}_t, \mathbf{u}_t)$. This is usually done in the control literature, assuming a linear local model is quite often very efficient. Another important assumption made is that the cost is a quadratic function of the latents:

$$c(\mathbf{z}_t, \mathbf{u}_t) = (\mathbf{z} - \mathbf{z}_{goal})^T R_z(\mathbf{z} - \mathbf{z}_{goal}) + \mathbf{u}_t^T R_u \mathbf{u}_t$$

where the R matrices are cost weighting matrices (dimensions given such that the respective multiplications work) and $\mathbf{z}_{goal}$ is the representation of the goal state in latent space, which is usually inferred. This gives a linear quadratic Gaussian control problem which can be solved with the iLQR controller (the same one used in the previous section). We need to mention that the observations are

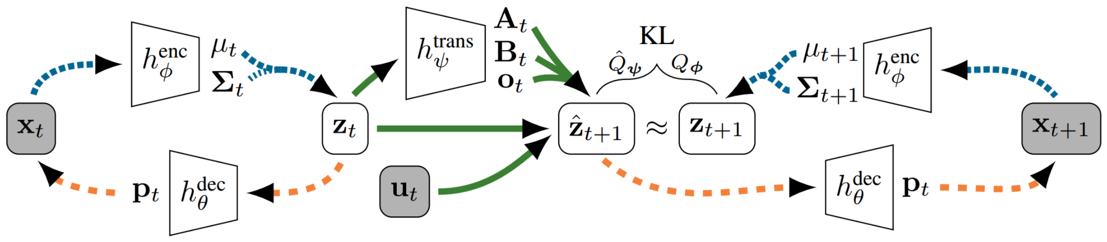


Figure 3: The encoder and decoder are neural networks which output a Gaussian described by mean μ_t and Σ_t. The latent code $\mathbf{z}_t$ is used to model the transition. Learning $\mathbf{A}, \mathbf{B}$ and $\mathbf{o}$ enables prediction $\widehat{\mathbf{z}}_{t+1}$. Similarity to a true encoding $\mathbf{z}_{t+1}$ is enforced by the KL divergence. Figure from [Watter et al., 2015].

encoded in the latent space through neural networks (actually Gaussian distributions coming from neural networks) and the full generative model can denoted in the following way (note that $\mathbf{s}_t$ is assumed to be unknown and we observe just a different dimensional representation $\mathbf{x}_t$):

$$\mathbf{z}_t \sim Q_\phi(Z|X) = \mathcal{N}(\mu_t, \Sigma_t)$$
$$\mathbf{z}_{t+1} \sim \widehat{Q}_\psi(\widehat{Z}|Z, \mathbf{u}) = \mathcal{N}(\mathbf{A}_t\mu_t + \mathbf{B}_t\mathbf{u}_t + \mathbf{o}_t, \mathbf{C}_t)$$
$$\tilde{\mathbf{x}}, \tilde{\mathbf{x}}_{t+1} \sim P_\theta(X|Z) = Bernoulli(\mathbf{p}_t)$$

where $\widehat{Q}$ is the posterior of the next latent state following the linear dynamics described above. The linearized model (i.e. the parameters A, B and $\mathbf{o}$) are given by other neural networks parameterized by ψ and learned through gradient descent. Also because predicting in the latent space might give a sample which when projected back into the input space does not really make sense (i.e. there is no input for which such a latent space exists) then a KL divergence penalty term is added to the standard cost function, which is the negative data log likelihood:

$$\mathcal{L}(\mathcal{D}) = \sum_{(\mathbf{x}_t,\mathbf{u}_t,\mathbf{x}_{t+1})\in\mathcal{D}} \mathcal{L}_{bound}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}) + \lambda KL(\widehat{Q}_\psi(\widehat{Z}|\mu_t, \mathbf{u}_t))||Q_\phi(Z|\mathbf{x}_{t+1})$$

where the variational bound for each example is given by:

$$\mathcal{L}_{bound}(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}) = \mathbb{E}[-\log P_\theta(\mathbf{x}_t|\mathbf{z}_t) - \log P_\theta(\mathbf{x}_{t+1}|\widehat{\mathbf{z}}_{t+1})] + KL(Q_\phi||P(Z))$$

where $P(Z)$ is an isotropic Gaussian prior for the approximate posterior Q_ϕ with mean zero and unit variance. The overall architecture can be seen in Figure 2.3.2. In the experimental section, the system is shown to find stochastic optimal controllers from high dimensional pixel data. The latent linear dynamics is able to provide enough richness to the system such that these controllers perform very close to the optimal ones in the real world.

After seeing a couple of approaches to model-based DRL, we see that many simplifying assumptions are made for practical algorithms dealing with high dimensional pixel data, but these work quite well in practice, so we expect further proliferation of such methods. Having detailed and exemplified the three main paradigms of learning in DRL systems, we now turn to the more general class of learning, called transfer learning, where an agent deals with multiple tasks or (rarely) domains and tries to use knowledge learned from one task, when dealing with another task, without forgetting the initial one. This is the name that has caught on for this important direction to general AI.

3 Transfer learning

Transfer learning is the name given to any machine learning methodology that can deal with multiple datasets by making use of existent learnt knowledge. In this way it is transferring the learning from one task to another, which might be very similar, or quite different. In general, if the task is too different we can see negative transfer, in the sense that when trying to use existing knowledge we see a decrease in performance. Then positive transfer is the opposite effect. The task from which we transfer knowledge is called the source task, while the task to which we transfer it is called the target task. Most transfer learning approaches until now dealt with tasks as classification, however we are interested in transfer learning in the DRL context. There are a few approaches which tackle this problem. We will briefly state next these approaches and their achievements. We need to mention here, that the literature on transfer reinforcement learning (i.e. not deep) is significantly more vast than transfer DRL. We will touch upon some of the methods used in TRL in the following exposition. For a comprehensive overview the reader is referred to [Taylor and Stone, 2009]. Techniques used in TL for classification tasks can of course be extended to DRL case but we are more interested in RL specific techniques that take into account the sequential nature of the tasks that DRL agents generally deal with. We will shortly describe next a few approaches that look very promising and that deal with TL in the DRL context.

3.1 Actor-Mimic [Parisotto et al., 2015]

In this paper the authors use a set of expert networks to teach a student network their skills, so to say; they call this the Actor-Mimic Network (AMN). The student effectively mimics the internal representation of the expert network through an additional loss term on the last layer of both networks, effectively trying to match the features of the AMN to the expert's. More specifically, the term is given by:

$$\mathcal{L}^i_{FeatureRegression}(\theta, \theta_{f_i}) = ||f_i(h_{AMN}(s;\theta);\theta_{f_i}) - h_{E_i}(s)||^2_2 \tag{1}$$

where the mapping f_i is the feature regression network that strives to predict the features of the expert network from the features of the AMN network. h_{E_i} and h_{AMN} are the last features of the networks (the pre-output layer activations), E_i is the expert network that has learnt to deal with a task, indexed by i and θ are parameters. To note here is that this mapping can have a different dimensionality for the input and output as the AMN and expert networks might have different sizes in the pre-output layers. Then the full objective is just a sum of the feature regression loss shown in Equation 1 and the ubiquitous cross-entropy loss between policies of the two networks, given by:

$$\mathcal{L}^i_{policy}(\theta) = \sum_{a \in \mathcal{A}_{E_i}} \pi_{E_i}(a|s) \log \pi_{AMN}(a|s;\theta)$$

where $\mathcal{A}_{E_i}$ is the set of actions for expert E_i and π are the policies associated with the networks identified by their subscripts. The transfer of knowledge takes place when a new expert network is initialized from the weights of the AMN network. The last softmax layer is removed and then the new DQN expert network is initialized with its weights. If the new task is similar to one of the tasks that the AMN learnt, then with slight fine tuning the new expert DQN will learn much faster and have good performance on the new task. The authors provide also convergence and performance guarantees. For more details the reader is referred to the original work [Parisotto et al., 2015]. The performance of the AMN is comparable to that of the expert DQNs, however, for some games the AMN performs worse, in other words, we see the negative transfer mentioned above. We hypothesize that this is happening because of the monolithic approach used, as the original DQN, the state representation and action representation are coupled, so that it is hard to have good performance on multiple different state representations and different policies. We will show in Section 5 how separating the two can help with this problem.

Next, we present a similar architecture, where different skills are learned independently, however, in this following work the different skills are combined using another deep network.

3.2 Reusable skills - Lifelong learning in Minecraft [Tessler et al., 2016]

Minecraft is a relatively new lifelong learning game where complex skills are needed to perform high level tasks. For example, to build a house, one needs to chop wood, then sand the wood, cut the wood into pieces, and finally nail the pieces together. The authors build individual networks (they call them Deep Skill Networks - DSN) to deal with such skills and then use a hierarchical deep reinforcement learning network (H-DRLN) to combine and reuse the skills learnt with the DSNs. The skills are exactly the options described in Section 4.4. We describe next some details of the architecture used, relevant to the current paper. For a detailed description and a pictorial representation of the architecture, the reader is referred to the original work [Tessler et al., 2016].

The highest level controller is a deep network that can output either a single action or a skill, which is a set of actions. The skills implement individual policies with deep networks trained a-priori on individual tasks and combined into a Deep Skill Module. This module was implemented as a policy distillation technique, where all skills are learnt with a single network, or as an array of different DSNs, where each network was specialized on one individual skill. The distillation technique was used as a scalable version of the array, as having an array of deep networks is quite expensive in terms of memory as well as computation. The deep skill module outputs a policy given a state and a skill index. It is not clear how the index is chosen, but we assume it is done by the higher level controller, the HDRLN. This relatively simple architecture enables reuse of the already learnt skills for new tasks, with no additional training. It also scales through the use of a variation of policy distillation, and finally can combine individual skills to solve complex tasks needed in Minecraft. We need to mention here the modifications brought by the authors in the whole system. They use Double DQN [Van Hasselt et al., 2015b] instead of the Vanilla DQN for faster convergence, and Skill Experience Replay, which is as the name implies the replay memory for the skill networks, which does not store every transition from t to $t+1$ but uses the length of the skill, so from t to $t+k$, where k is the number of timesteps the skill is active for. The main points relevant for us in this work are the use of the higher level controller, we can call it a *meta-controller*, and the use of skill distillation, a variant of policy distillation [Rusu et al., 2015] for scaling the architecture to multiple skills more efficiently. We will see that the notion of a meta-controller in more approaches, as this seems to be quite a straightforward way of modelling control at different levels of abstraction.

Even though this work was not applied to the Atari domain, its applicability seems to be quite straightforward, considering the relatively low complexity of the whole architecture. Remains to be seen if such an architecture will catch on or not.

We proceed further with a very interesting line of research, where a second learning algorithm (besides the original RL used to train the network for each task) is learned based on learning a series of tasks. This is called meta DRL and the main idea behind it is to use a recurrent network (RNN) to act like the policy which takes into account multiple tasks when adapting its weights. A second learning algorithm is developed in the weights of the RNN, due to its memory, such that even after its weights are fixed, the RNN is able to learn to adapt to new tasks. Thus we can say that the RNN is learning how to learn. We present next two such approaches to meta-DRL.

3.3 RL^2: Fast reinforcement learning via slow reinforcement learning [Duan et al., 2016]

A very recent work on learning general policies, much faster than the original DQN and subsequent works is using a Recurrent Neural Network (RNN), more specifically a GRU (Gated Recurrent Unit) to represent the policy. The reinforcement learning algorithm is encoded in the weights of the RNN and is learned from data, through a slow RL algorithm. The system is tested on multiple different MDPs and in general the interest lies in how the agent deals with unseen MDPs based on the learned and seen MDPs. The full interaction with an MDP is referred to as a trial, which consists of episodes, with different trajectories in the particular MDP. The RNN receives as inputs the action a_t, the state s_{t+1}, reward r_t and termination flag d_t (which is normally 0, unless an episode has ended, when it is 1). These are concatenated into a long vector and then fed to the policy, which conditioned on the inner state h_{t+1} generated the next inner state and action h_{t+2} and a_{t+1}. The inner states of the network are preserved for the same MDP (so between episodes) but are reset when the MDP changes (so between trials). The objective is changed such that the agent is trying to maximize discounted reward in a trial, so for multiple episodes. This is equivalent to minimizing the cumulative pseudo-regret [?]. When faced with a new MDP the agent needs to continually adapt its strategy integrating information about past actions, rewards and terminations. We need to mention that the GRU is connected to a fully connected layer, which is then connected to a softmax that gives the distribution over actions. The policy optimization used is the first order TRPO [Schulman et al., 2015a]. The authors use a further baseline, or V-function, modelled by a RNN as well. In the arm-bandit case, finite horizon, the performance is on par with the theoretically justified algorithms, with minor differences in the large horizon case. This work shows a very important concept which discovers a learning algorithm which is optimal in a sense to the given domain. In the next work a similar approach shows how this type of learned algorithm is able to have an adaptive learning rate as well as variable, data-driven exploration-exploitation trade-off.

3.4 Learning to reinforcement learn [Wang et al., 2016]

This work is very similar with the work described in the previous section. The learning algorithm is modelled as an RNN, more specifically a LSTM, such that it can adapt without changing the weights to different tasks. In the training phase the algorithm is presented with a set of different MDPs and it is expected to fit the weights such that it can easily switch between the tasks when presented with slightly different versions of the MDP during testing time. The architecture is similar to the one above, there is a slower system which adapts individually to each task and a faster learning system which learn to adapt between tasks, in a sense learn the task structure or domain knowledge. The input to both the low level and this higher level learning system are actions performed in the previous timestep and (critically) the rewards received as a consequence of these actions. This system is referred to as deep meta-RL and is tested on a series of tasks, some from the psychology literature, showing very interesting behaviour, like balancing exploration-exploitation in a task-dependent way, adapting the learning rate again as a function of the task and the ability to improve on some tasks even though the weights of the RNN remain fixed. The authors notice that this is building domain-specific biases and thus the learning algorithm discovers and leverages the tasks' covariance structure, which can lead to one-shot learning in some cases where this is to be expected (in the sense that the original experiments performed on monkeys exhibited one-shot learning). We believe this is a very promising line of research and we expect to see continuations of this work very soon. This relatively simple formulation of the optimization over multiple different MDPs enhances the agent such that is able to infer a general pattern in learning, basically being able to adapt to yet unseen MDPs. So we could say this is a process driven learning, as opposed to the classical data driven learning.

A yet different approach, presented in the next section, for transfer learning is using task similarity as a measure of transfer. Also the ability to combine multiple known policies is quite an important desirable behaviour which is presented in this work.

3.5 A^2T : Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources [Rajendran et al., 2015]

One of the major problems in transfer learning is how to avoid negative transfer. A mechanism that allows for positive transfer and elegantly avoids negative transfer is proposed in this work. A set of source policies is learnt with vanilla DQN and the goal is to be able to learn existing good behaviour for a new related task from these existing policies. Quite a few contributions are put forward by this work. One is the ability to avoid negative transfer, by having an attention mechanism which can assign a weight to the existing policies as a function of the state, so if some existing states and the current state are very similar then the policies associated with the existing states will be used in the current decision making. Another important contribution is the ability to combine existing policies to this soft attention mechanism which assigns real values to these weights of existing policies when building the new policy. This combination can be done for each state, this makes the new policy extremely adaptive. Moreover, the transfer mechanism works for policies as well as for value functions. However, a major drawback is the fact that this whole mechanism works for source and target tasks which have the same state-action space. So, in effect, the transfer is done in the same domain, albeit the tasks can be quite different. Stating the main points of the paper we proceed with the main formal description. We show in Figure 3.5 the architecture of the agent. The behaviour of the agent on a new task is given by:

$$K_T(s) = w_{N+1,s}K_B(s) + \sum_{i=1}^{N} w_{i,s}K_i(s) \text{ with } \sum_{i=1}^{N+1} w_{i,s} = 1, w_{i,s} \in [0, 1]$$

where N is the number of existing source policies, which remain fixed. K_B is the base policy, which will adapt specific to this task and K_T is the actual behaviour of the agent. As we said this transfer process can be used with policy-based learning as well as value-based learning. The weights $w_{i,s}$ are learned by the attention network and can change with every state, which enables high adaptivity of the current behavioural policy. Both the base policy (parameterized by θ_b) and the attention policy (parameterized by θ_a) are updated with REINFORCE [Williams, 1992] in the following manner:

$$\theta_b \leftarrow \theta_b + \alpha_{\theta_b}(r - b)\frac{\partial \sum_{t=1}^{M} \log(\pi_B(s_t, a_t))}{\partial \theta_b}\theta_a \leftarrow \theta_a + \alpha_{\theta_a}(r - b)\frac{\partial \sum_{t=1}^{M} \log(\pi_T(s_t, a_t))}{\partial \theta_a}$$

where the α are learning rates, b is the baseline, r is the return per episode and M is the time length of the episode. We note that even though the behaviour is generated by π_T the update to the base policy is done by looking at the gradient of π_B. The central network, the attention network, learns the weights of the existing policies for the current task, given the input state. This is just to show an example of the learning rules, however, the authors use also an actor-critic architecture with TD-learning as well as Q learning. The updates are as expected, for more details see the original work. The authors show a series of experiments where the negative transfer is avoided, and they also show significant speed-up achieved by transferring from a related task. Even though it has some limitations for now (fixed state-action space) this architecture is quite elegant and simple and shows a lot of promise for use in transfer deep reinforcement learning.

The limitation of having the same state-action space for the source and target domains could easily be overcome by using some type of dimensionality reduction, e.g. an autoencoder and then look just at the bottleneck layer which ought to have the same dimension when computing similarity. Or an even simpler alternative could be a hash function encoding of the states.

3.6 Overview and discussion

We presented a series of approaches to transfer learning for DRL. This is still a very young field, but we can already see some intuitive trends. Hierarchy seems to play a major role in dealing with multiple tasks, be it in a form of a meta-controller or the meta-DRL. Most of the approaches use some type of memory, be it in the RNN or the storing states for comparisons. We imagine that as the number of tasks increases, there will be need for an efficient representation and searching mechanism. Also it is to be expected, giving the nature of transfer learning, to have to deal with multi-objective optimization. We expect more and more methods from this field to be used in the following works in transfer DRL.

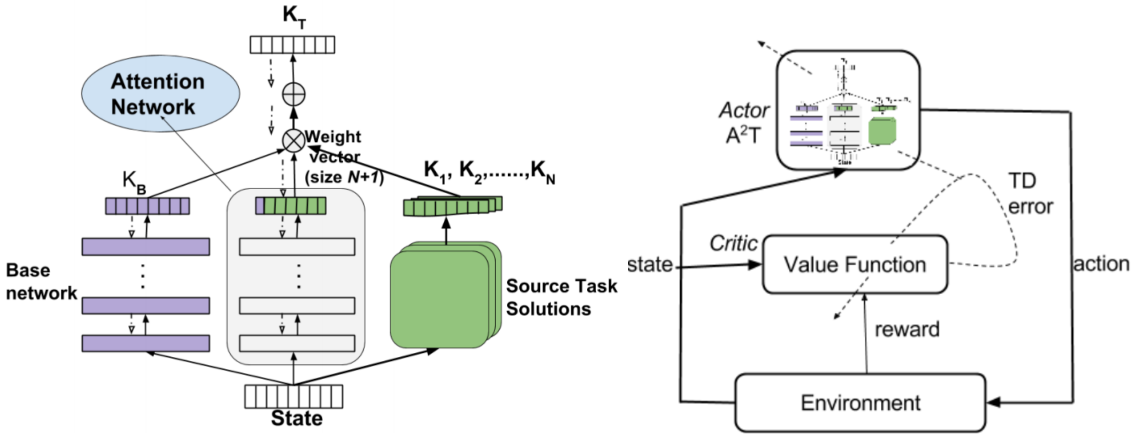


Figure 4: A2T agent architecture. Figure from [Rajendran et al., 2015].

4 Hierarchical Reinforcement Learning

4.1 In humans

The problem of constructing modular subroutines of actions, that can be easily and flexibly assembled into complex behaviour comes originally from developmental psychology [Fischer, 1980]. Evidence from neuroscience suggests that this is indeed how the brain represents action mapping as well. Areas in the frontal cortex represent actions mapping at multiple time scales, i.e. they are organized at multiple levels of temporal abstraction [Badre, 2008]. For example, a single pattern in the dorsolateral prefrontal cortex (DLPFC) corresponds to an entire mapping from stimulus to response [Miller and Cohen, 2001]. Moreover, in [Diuk et al., 2013] they show through three experiments how humans can detect states with high centrality, in a graph-theoretic sense, and then use them as subgoals for temporally abstract representations. For more information the reader is referred to the original work [Diuk et al., 2013], which is a good overview of the parallels between current hierarchical reinforcement learning approaches through options and the analogous processes in the human brain and that strongly advocates for such hierarchical temporally abstract representations. One critical question stated by the authors, and relevant to our current work, is how are partially similar states or action represented or learned by such mechanisms. Another important point the authors make in the cited work is the existence of two reward signals, at different hierarchical levels of representation, or time abstraction. We will get back to this point in Section 5.3. More evidence for this kind of hierarchical abstraction comes from [Botvinick and Weinstein, 2014] where they show and cite significant work from neuroscience and psychology that supports the options framework. Moreover, they argue for a synthesis of model-based and model-free HRL in the brain, raising many interesting questions with respect to their combined functioning. They also argue for the superiority of model-based HRL, when it comes to planning efficiency and lower computational load (in terms of memory), especially when enabling this type of agent to assign expected rewards to the actual options. They call this saltatory MB-HRL and implement it through an actor-critic approach, whose details are given in the original work. Next, we will go into more details about the model-free and model-based approaches to RL and show how these approaches develop with respect to the hierarchical context.

4.2 HRL with Options

One of the most interesting aspects in the DRL agents' behaviour is the ability to pursue higher level tasks built from lower level, simple, atomic behaviours. We will go briefly through the main idea used in the this section: the option framework [Sutton et al., 1999b]. Options are tuples of the form: $o = \langle I, \pi, \beta \rangle$ where I is the set of possible starting states for the option, π is the sub-policy that governs this option and β is a stochastic function that defines when the option terminates. This defines a semi-Markov decision process (SMDP) that gives the following update rule, using Q-learning:

$$Q(s,o) \leftarrow Q(s,o) + \alpha[r + \gamma^k \underset{a \in O}{max} Q(s,a) - Q(s,o)]$$

where O is the set of all options, α is the learning rate, and k is the number of steps for which the option is active. Q is the action and option value function respectively. The option framework enables temporal abstraction over the state-action space, such that options that operate at different time scales are possible which then enable a modular approach to state-action representation.

4.3 Model-free HRL

In model-free approaches, the state representation is not different than the action representation. The agent learns to associate high-value states with actions through the reward signal by employing what is known as the *value function* or *action-value* function. By the reward signal we mean the long-term cumulative discounted reward. In the HRL version, this value function now associates sequences of actions, or subpolicies with rewards, such that the behaviour of the agent takes into account higher-level abstraction of state-action pairs, the options described in Section 4.4.

4.4 Model-based HRL

As we said before, in model-based RL, the agent forms a model of the state-action space, it learns the dynamics of transitioning from one state to another. This is a type of causal model, where actions have, in general, probabilistic outcomes. Moreover, the agent learns also a reward model, where actions are associated with rewards, enabling the agent to plan accordingly. Model-based learning is generative, meaning one can sample from the model for unseen inputs (in this case states) to obtain a somewhat meaningful representation (in this case meaningful actions).

In the HRL version, this paradigm enhances the behaviour of the agent significantly, enabling the agent to learn a model over the option space, basically over abstract state-action representations. This increases the efficiency of the agent significantly, in terms of computation (as now high-level learning takes place in this abstract space, over abstract state-actions, but whose number is much less than the primitive states), but also in terms of behavioural capabilities, enabling it to have meaningful new behaviours for unseen circumstances. In the saltatory MB-HRL mentioned above, the agent now defines an option as a joint probability over option duration and termination state and also the expected cumulative discounted reward while executing the particular option. This enables high-level predictions in a sense, where an agent uses the options to reason in a more efficient (high-level) way about the possible outcomes. A number of very interesting aspects of model-based HRL are mentioned in [Botvinick and Weinstein, 2014]. First of all, they show how having an option model can aid in determining the bottleneck states, or the subgoals on which options are then learnt. Based on the transition model, the agent can remember which states enable the agent to reach a wider variety of different states for example, defining causal dependencies in the transition model, which can then help into decomposing the action space into subgoals. This theory of building a transition model that can help option discovery is put forward also in [Schapiro et al., 2013]. Representing the desired outcomes through options can help in identifying existing candidate options through these specific outcomes, effectively employing options by means of the subgoal search. The critical point of employing options in HRL is the ability of the agent to skip low-level state-action evaluation, and considering that sometimes these low-level models are not accessible yet or unknown, this still enables to agent to reason about high-level temporally extended state-action sequences.

4.5 Subgoals

4.5.1 Option discovery

The option discovery problem is of critical importance to HRL. We saw in previous sections how option models can significantly improve efficiency and performance of RL agents. However, the agent needs to first discover these options. This can be considered a subset of the intrinsic reward problem described in Section 5.3. We state here briefly some of the approaches existent in the literature for option discovery. The problem of discovering options is tightly related to the exploration-exploitation dilemma in reinforcement learning. We need a exploration mechanism that enables us to efficiently go through the state-action space such that we can identify candidate options. A question that naturally arises in this context is what particular characteristics does an option need to posses to be considered useful for the agent. An option is considered useful if it can implement some behavioural pattern that can be reused by the agent in different contexts, so in effect we are interested in what subtasks an option implements, so we can look at the final subgoal of an option to figure out some of its characteristics. One perspective, is to consider as subgoals particular states or state-sequences which enable access to a wider variety of next states, so effectively maximizing the state-space subset accessible from this subgoal state, and thus maximizing information accessible to the agent. This is a widely used concept in curiosity and efficient exploration, it is often referred to as the information-seeking principle. So an option could enable the agent to learn to transition efficiently to these subgoals. These are often called bottleneck states. We give next some of the approaches existent in the literature for finding such states:

- [McGovern and Barto, 2001] look at rare transitions between subsets of state-space, and consider these points of transition as bottleneck states
- [Hengst, 2004] looks at the sequences of actions that cause rare changes in the state representation. By employing a factored state representation, rare transitions can be identified
- In [Şimşek and Barreto, 2009] they derive a graphical representation of the state-action space and then use node centrality, a graph theoretic measure to identify nodes with high centrality
- In [Menache et al., 2002] they use clustering methods to identify strongly connected components of the underlying MDP and then consider as bottleneck states those states that connect two or more clusters.
- In [Brunskill and Li, 2014] they use a greedy approach to discovering options inspired by PAC-learning, by making use of sample complexity, a measure of learning speed, counting the number of non-optimal actions [Kakade et al., 2003].
- Search for more approaches

As we can see there is an overarching principle, we are searching for peculiar states, states that are different and that enable the agent to gather more information about the state-action space or get access to a new part of the state space. In the context of machine learning we could even see them as some type of outliers in the state-action space.

4.6 Temporal abstraction

Hierarchical structuring plays an important role in deep networks, as well as in human representation, knowledge and behaviour. Actually, as argued in [Lin and Tegmark, 2016], it is fundamental to our existence. So we would expect it to play a major role in DRL as well. And indeed it does. As investigated in [Baram et al., 2016] the underlying representation which DQN builds when learning to play Atari games is indeed hierarchical. They cluster states in a low dimensional embedding, done through t-SNE, based on temporal information. The clusters then define a SMDP, which also minimizes entropy, through a regularization term. So in effect, this discovers options governed by particular policies. In light of the idea described next, we emphasize the use of an additional identifier (the Q function depends now on the state and the skill, or option). A very similar approach is in [Kulkarni et al., 2016a], which makes use of an additional goal that plays the role of the identifier above. The Q function now depends on the goal, that, the same as above, persists for a number of time-steps. In this case the goals are selected by a meta-controller, which learns a policy over the them, and are fed to the controller to learn policies over actions. The meta-controller gets rewards from the environment, while the controller gets an intrinsic reward from a critic which checks if the goal is reached. So the options discovered are built such that they reach the goals that the meta-controller chooses to feed to the controller. The authors use as a benchmark the Atari game Montezuma's revenge and show superior performance compared to the most of the previous approaches. The agent makes use of two replay memories, one for each controller. We will describe in a later section the intrinsic reward which the critic is giving to the controller to reach the inner goals. Another hierarchical approach to discovering options is through spatio-temporal clustering through PCCA+ (Perron Clustering) [Lakshminarayanan et al., 2016]. In this work the authors assign states to abstract states in a fuzzy manner, where each state has a membership function which governs how much a particular state is part of an abstract state. Then they find the policy over abstract states through hill-climbing on the membership functions. This is quite elegant and moreover it is enhanced by the use of the powerful spectral clustering technique (PCCA+) and by a action conditional convolutional network [Oh et al., 2015], which extracts high level features that also take time into account. The authors show the advantage of discovering the different structural and functional abstractions, and especially differentiating between them, as we are also advocating throughout this paper.

4.6.1 Spatio-temporal clustering

As we mentioned briefly, to enable meaningful state representation for the agent to map policies on, we need to take into account the temporal nature of the task at hand. There are a number of ways to do this, and we mentioned briefly some of them above. A simple concrete example is given next. When clustering states with K-means, the authors in [Baram et al., 2016] use a distance that takes into account temporal neighbouring states when assigning points to clusters, e.g. for any observation x_p and any center of a cluster μ_i the assignment is:

$$C_i = \left\{ x_p : ||X_{p-w:p+w} - \mu_i||^2 \leq ||X_{p-w:p+w} - \mu_j||^2, \forall j, 1 \leq j \leq k \right\}$$

where p is the time index and $X_{p-w:p+w}$ is a set of $2w$ points before and after x_p along the trajectory. We have to note here, that the authors use an entropy regularization term as well with the purpose of finding simpler models. In general, clustering techniques that take into account global information are more desirable, but also more expensive. Ideally one would reduce dimensionality and then perform some type of spectral clustering which takes into account the global structure of the data.

4.6.2 Bottleneck states

One often used method for finding subgoals, or options, or states that have interesting properties is to use the minimum normalized cut. This has been used for discovering options [Kulkarni et al., 2016b, Şimşek et al., 2005] but also for discovering other types of structures, for example, objects in an image [Shi and Malik, 2000]. The main idea used in DRL [Kulkarni et al., 2016b] is to first collect a large number of states T $= \{m_{s_1,a_1}, m_{s_2,a_2}, ..., m_{s_n,a_n}\}$, following a random policy, and then generate an affinity matrix W by applying a radial basis function (with Euclidean metric) to every pair $(m_{s_i,a_i}, m_{s_j,a_j})$ in T, to generate for each such pair a similarity w_{ij}. Now, form the diagonal matrix $D = \sum_j w_{ij}$ and take the second largest eigenvalue of the matrix $D^{-1}(D - W)$ which will give an approximation of the minimum normalized cut of partitioning T. Then the subgoals are the points, in this case these are states, which lie on the endpoints of the cut. After randomly sampling T, statistics of how many times a state lies along the cut are collected and then the top-k such states are selected as subgoals. These prove to be useful subgoals for speeding up learning. A very similar idea, a bit simpler, is presented in [Kulkarni et al., 2016a] where the authors perform image segmentation (in Montezuma's revenge) to localize individual objects in the image and then use those as subgoals. Ideally one would like a relatively efficient way of devising subgoals but also comprehensive. This is an active area of research, however there is an alternative to this which tends to be more efficient, bypassing the need for collecting and processing states in some specific manner. The alternative can be referred to as the auxiliary reward addition and is presented in Sections **??**.

5 Decomposition

Decomposition of the state-action-reward function offers a number of advantages over the monolithic approach. This section investigates how we can uncouple the states from actions and rewards and what kind of benefits we can expect from such a decomposition. We discuss shortly the advantages of decomposing the state-action-reward function.

5.1 States

In the original DQN paper, the states and actions are coupled, in the sense that the cost function influences both the state representation, as well as the action representation or mapping. In subsequent papers, these have been separated. For example in [Kulkarni et al., 2016b] the representation is learned through unsupervised learning, using an autoencoder to obtain a higher level representation for states. In [Mnih et al., 2016], the fastest instance of a DRL agent from DeepMind, a very similar idea is used where the state contribution is subtracted from the estimate of the so-called advantage function. The main idea is to isolate, or decouple the state contribution from the behavioural policy. We emphasize this idea, as we consider it to be critical for successful developments of future DRL agents. We can think about this in the following way. It shouldn't matter on which representation is the agent acting upon, as long as the task is identical or similar, the fact that we are using DBMs, DNNs or CNNs, as long as the representation catches the characteristics, or features of the state space, the policy should not be affected by the underlying representation. Obviously some will be better than others, in the sense that they will more separated or have lower intrinsic dimensionality, which should help the mapping from the representation to action, but other than that there is no reason to couple the two.

5.1.1 Deep Successor Reinforcement Learning [Kulkarni et al., 2016b]

One of the most fruitful approaches that focuses on state representation is called the *successor representation* and is based on the idea that the action-value function can be decomposed, or expressed as a dot product between the expected state occupancy and the reward associated with each of those states. In effect this gives a hybrid approach between model-free and model-based approaches, where the successor representation can be seen as a predictive model, while the reward learning is still model-free. The two systems are trained alternatively in [Kulkarni et al., 2016b]. Formally the successor representation (SR) is defined as:

$$M(s, s', a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathbb{1}[s_t = s'] | s_0 = s, a_0 = a\right],$$

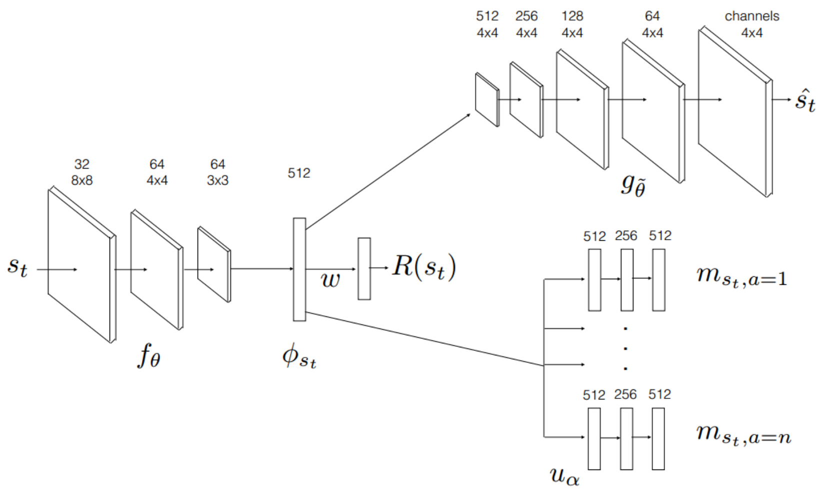


Figure 5: Architecture of the Deep Successor Reinforcement Learning agent. Figure from [Kulkarni et al., 2016b]
.

where $\nVdash[] = 1$ if the argument is true or 0 otherwise, which enables the SR to also keep track of the states visitation. The iterative version of this equation is:

$$M(s, s', a) = \nVdash[s_t = s] + \gamma\mathbb{E}[M(s_{t+1}, s', a_{t+1})]$$

Now we the action-value function can be expressed as:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} M(s, s', a)R(s')$$

Each state is represented as a D-dimensional vector ϕ_s which is the output of a deep neural network with parameters θ. The SR is denoted by m_{sa} and is approximated by another neural network $u_\alpha(\phi_s, a) \approx m_{sa}$ with parameters α. The reward function is approximated by a linear function of the state features ϕ_s:

$$R(s) \approx \phi_s \cdot w \text{ with } w \in \mathbb{R}^D$$

which then gives $Q^\pi(s, a) \approx m_{sa} \cdot w$. A version of the Bellman equation with state features for the SR is given by:

$$m_{sa} = \phi_s + \gamma\mathbb{E}[m_{s_{t+1}a'}] \text{ with } a' = \underset{a}{argmax}(m_{s_{t+1}a} \cdot w)$$

An interesting addition is the intrinsic reward function given by $R_i(s) = g_{\tilde{\theta}}(\phi_s)$ which provides a dense reward signal trying to reconstruct the input through an autoencoder. The entire architecture is presented in Figure 5.1.1. The combined loss function is then elegantly given by:

$$\text{Ł}(\theta, \alpha, w, \tilde{\theta}) = L^m(\alpha, \theta) + L^r(w, \theta) + L^a(\tilde{\theta}, \theta)$$

where $L^m(\alpha, \theta)$ is given by the Bellman equation for SR given above, i.e.:

$$L^m(\alpha, \theta) = \mathbb{E}[(\phi(s_t) + \gamma u_{\alpha_{prev}}(\phi_{s_{t+1}}, a') - u_\alpha(\phi_{s_t}, a))^2] \text{ with } a' = \underset{a}{argmax} u_\alpha(\phi_{s_{t+1}}, a) \cdot w$$

where α_{prev} are parameters of the target network (θ^{-} in the original DQN paper). The learning the reward parameters,w, the following loss is used:

$$L^r(w, \theta) = (R(s_t) - \phi_{s_t} \cdot w)^2$$

Because the features ϕ should be good state discriminators, this can be achieved by letting ϕ be the bottleneck layer of an autoencoder, so adding the autoencoder loss is added to the loss function:

$$L^a(\tilde{\theta}, \theta) = (g_{\tilde{\theta}}(\phi_{s_t}) - s_t)^2$$

We see here quite an interesting approach, where three losses are combined and the successor representation is successfully used to increase performance of a DRL agent compared to the original DQN. A interesting aspect is the greater sensitivity exhibited by the agent to the reward magnitude compared to the DQN agent.

5.1.2 Towards Deep Symbolic Reinforcement Learning [Garnelo et al., 2016]

Another very interesting alternate direction of deep reinforcement learning is to combine it with symbolic representations, that is high level discrete, human-interpretable representations. In this work, a low-level convolutional representation is built which is then connected to a higher lever symbolic system that learns to take actions as to maximize the discounted future reward. Even though the system described is highly tailored to the problem at hand (which is a simple grid world with 3 types of symbols, one type which gives positive rewards, one type which gives negative and one which is the actual agent), the approach seems promising in terms of problems which it could tackle and classical DRL could not. The pipeline starts as always with a convolutional network which extracts features from pixels. After this, an ad-hoc thresholding procedure is applied which outputs the most relevant pixels in the convoluted images and classifies existing objects. Then a temporally extended representation is built to track existing objects by comparing subsequent frames, considering also the notion of moving continuity (if some objects move the movement should be continuous) and the neighbours of each object. To model the interactions again subsequent frames are considered and the positions of objects are considered relative to each other, so locally and not globally. Another interesting modification is the existence of multiple Q functions (tabular) which describe different interactions and for each action selection, just the ones that are relevant for the interaction are queried and then summed up to select the action. ϵ-greedy is used for building the convolutional representation and exploring the state-action space, with $\epsilon = 0.1$. This representation turns out to be quite powerful for this relatively simple grid world, at least compared to the original DQN, even though in a simpler version of the game DQN learns to maximize rewards while the symbolic agent cannot do this.

symbolic DRL
deep successor - mit
successor deepmind

5.2 Actions

In this section we will advocate for the use of history or memory enhancements of the policy. A simple reason is the following: when in complex environments, the current optimal action selection policy is not purely dependent on the state the agent is in, but also on the previous states as well as previous actions. Consider the (in)famous Montezuma's revenge game, where the action sequence of going to the door is completely unrewarding unless the agent picked up the key first. So being in the same state is one time rewarding and one time not, depending on the previous sequence of actions and states. As we saw earlier actions have their own value function called advantage function (Section 2.1.3), they can be the labels for a classification problem (Section 5.3.5) or can be fed as in input to a recurrent neural network (Section 3.4) in the context of meta-DRL. We show next a technique which conditions on actions to predict future frames. Impressively, accurate 100 steps prediction is possible using this approach.

5.2.1 Action-conditional video prediction using deep networks in Atari games [Oh et al., 2015]

The major contribution of the paper in terms of practical achievements, besides the interesting approach used, is the ability to make long term predictions of high dimensional images conditioned on control input. The process is very similar to an autoencoding process but with the action added to the encoding layer and an LSTM which encodes the temporal features (we note that there is also a feedforward version without the LSTM). The encoding is given by:

$$[h_t^{enc}, c_t] = LSTM(CNN(x_t), h_{t-1}^{enc}, c_{t-1})$$

where c_t is a memory cell which encodes a long history of the inputs. The LSTM can be interpreted as an temporal encoder of the high level features given by the CNN. Then, multiplicative interactions between the action variable and the features are used:

$$h_{t,i}^{dec} = \sum_{j,l} W_{ijl} h_{t,j}^{enc} a_{t,l} + b_i$$

with h_t^{enc} is an encoded feature and h_t^{dec} is the encoded feature after the multiplicative action interaction. The weights are shared for different actions when can be useful when there exist common (across actions) temporal dynamics. The image is then constructed by deconvolution:

$$\widehat{x}_{t+1} = Deconv(Reshape(h^{dec}))$$

where Reshape is a hidden layer of a 3D feature map and Deconv is a set of deconvolution layers followed by nonlinearity, except the last one. The loss function is given by:

$$\mathcal{L}_K(\theta) = \frac{1}{2K} \sum_i \sum_t \sum_{k=1} K ||\widehat{x}_{t+k}^i - x_{t+k}^i||^2$$

where $\widehat{x}_{t+k}^i$ is a k-step prediction, where the training data is given by $\left\{(x_1^i, a_1^i), ..., (x_t^i, a_t^i)\right\}_{i=1}^N$. The model is trained in multiple phases as suggested in [Michalski et al., 2014].

5.3 Rewards

As one can imagine, rewards are critical for successful learning of reinforcement learning agents. As we mentioned previously, in complex environments and tasks, rewards are highly sparse, thus the behaviour of the agent cannot be driven only by external rewards. By external, or extrinsic rewards, we mean the actual reward signal given by the environment when the agent reaches a certain state (to be more pedantic, in some complex environments, rewards can be associated with a particular sequence of reached states). So the behaviour of the agent should be driven by other mechanisms than the extrinsic rewards. We refer here to meaningful behaviour, not some random generated policy following an ϵ-greedy policy for example. By meaningful behaviour we mean some sequence of actions that provides more information to the agent, either about the state space, the action space, or their coupling. The final purpose is, as expected, to reach the reward faster, with less random exploration. In the literature this type of reward is called pseudo-reward or intrinsic reward or motivation, or even curiosity. The first formalization of curiosity is given in [Schmidhuber, 1991] and describes how an agent should be driven by improving its world model. If its predictions about the world are consistent with the world then there is no reward given, however if the agent almost predicts the world then high rewards are given, whilst if the agent is far from predicting the world, then low rewards are given. This idea has caught up in literature as model-based reinforcement learning, introduced for the first time in [Sutton, 1990], where the agent learns also the dynamics of the environment which can then be used to augment learning from experience. As we mentioned earlier, learning the state transition function can help identify particular states that have some special properties, for example, they enable access to a wider set of additional states. These states are sometimes called bottleneck states. Another perspective on the curiosity problem, especially from a human point of view is to see it as the agent that is trying to minimize uncertainty. This problem is investigated in more detail in [Gottlieb et al., 2013], where comparisons are made between the biological and the computational approaches.

5.3.1 Variational information maximisation for intrinsically motivated reinforcement learning [Mohamed and Rezende, 2015]

In a different direction is the work of [Mohamed and Rezende, 2015] where they use a measure based on mutual information to reason about the information maximizing behaviour of the agent, called empowerment [Klyubin et al., 2005], given by:

$$\mathcal{E}(\mathbf{s}) = \underset{\omega}{max} \mathcal{I}^{\omega}(\mathbf{a}, \mathbf{s}'|\mathbf{s}) = \underset{\omega}{max} \mathbb{E}_{p(s'|a,s)\omega(a|s)} \left[\log \left(\frac{p(\mathbf{a}, \mathbf{s}'|\mathbf{s})}{\omega(\mathbf{a}|\mathbf{s}) p(\mathbf{s}'|\mathbf{s})} \right) \right]$$

where $\mathbf{a} = \{a_1, ..., a_K\}$ is the sequence of actions leading to $\mathbf{s}'$, while $p(\mathbf{s}'|\mathbf{a}, \mathbf{s})$ is the transition probability particular to the environment. $p(\mathbf{s}', \mathbf{a}|\mathbf{s})$ is the joint distribution of the actions and the final state conditioned on the current state. $\omega(\mathbf{a}|\mathbf{s}$ is the conditional over K-step sequences of actions, while $p(\mathbf{s}'|\mathbf{s})$ is the joint marginalized over the actions. Mutual information is given by $\mathcal{I}^{\omega}$ and it has the form:

$$\mathcal{I}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{p(y|x)p(x)} \left[\log \left(\frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x}) p(\mathbf{y})} \right) \right]$$

Maximizing the empowerment leads to states from which more states are accessible within the next K steps. In this research, the authors derive an efficient approximation of the mutual information, which is usually intractable for even medium sized state spaces, employing a variational approximation which makes use of deep networks to parametrize the different quantities of interest. The respective approximation can be used in many contexts where an efficient approximation of mutual information is needed. Before this approximation, the Blahut-Arimoto algorithm was used, which is exact, but is exponential in the size of the state space and action horizon K.

5.3.2 Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning [Tang et al., 2016]

.

Yet another approach for exploration comes from the tabular Q-learning literature, which proposes counting state-action pairs that are visited and give to the agent a reward proportional to this count [Lai and Robbins, 1985, Strehl and Littman, 2008b]. The deep version extends this approach for continuous and high dimensional state spaces through a simple mechanism, i.e. it replaces the counting method with a hash function that discretizes the space such that the states in this space can also be differentiated when necessary, but also generalizes well when two states are close to one another. Then the usual counting is employed in this new hashed space. One modification to the original approach is the fact that the authors in [Tang et al., 2016] use a state count and not state-action pairs. The hash function used needs to be part of the set of hash functions which are referred to as locality-sensitive hashing. The authors use SimHash [Charikar, 2002], a hash function which assigns a binary code to a state and that measure similarity between states using angular distance.

5.3.3 Deep Exploration via Bootstrapped DQN [Osband et al., 2016]

Yet another interesting approach is [Osband et al., 2016] which makes use of randomized value functions to explore in a more meaningful manner. The bootstrapping principle used is to try to approximate a population by a sample. Having multiple different approximations, which they call bootstrapped heads, and which essentially are randomized value functions used in exploration, enables the diversity needed for deep exploration. Deep exploration is far-sighted, extended to multiple time steps. All the heads are connected to the same shared representation and are fed an individual subset of the whole dataset. The random initialization plus the use of stochastic minibatches unique to each head are enough to provide sufficient diversity for deep exploration through sampling from the approximate posteriors. This approach provides uncertainty estimates for the bootstrapped network, and is similar to dropout, where the dropout for each head is fixed for each data point.

5.3.4 Reinforcement learning with unsupervised auxiliary tasks [Jaderberg et al., 2016]

This works build upon the state-of-the-art in Deep Reinforcement Learning, the Asynchronous Advantage Actor-Critic (A3C), which shares a subset of parameters between the policy $\pi(a|s,\theta)$ and the state-value function $V(s,\theta)$. The update includes a regularisation term, and uses n-steps lookahead (using notation in [Jaderberg et al., 2016]):

$$L_{A3C} \approx L_{VR} + L_{\pi} - \mathbb{E}_{s\sim\pi}[\alpha H(\pi(s,\cdot,\theta)] \text{ with}$$
$$L_{VR} = \mathbb{E}_{s\sim\pi}\left[(Rt : t+n+\gamma^n V(s+n+1,\theta^-) - V(s_t,\theta))^2\right]$$

The instance of the A3C used in this work uses a LSTM to output both policy and value function, having as input the entire history of its experience.

Auxiliary control and rewards

An auxiliary control task c is defined to have a reward function $r^c : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{S}$ is an extended state space, which includes besides the history of observations and rewards, the activations of the hidden units of the network. The overall objective is the given by:

$$\underset{\theta}{argmax} \mathbb{E}_{\pi}[R_{1:\infty}] + \lambda_c \sum_{c\in\mathcal{C}} \mathbb{E}_{\pi_c}[R^c_{1:\infty}]$$

with $R^c_{t:t+n} = \sum_{k=1}^{n} \gamma^k r^c_t$, being the discounted return for control task c, and θ being the parameters of the policies (base + auxiliary). The base policy shares some parameters with the auxiliary policies, thus the agent must balance the performance on the base task as well as the auxiliary ones. Any RL method can be used for optimizing this function, and the authors choose the n-step Q-learning. The auxiliary tasks are defined as:

- Pixel changes, i.e. the agent tries to find policies which maximally change the pixels in the input, by partitioning the input into a grid of $n \times n$ input cells. This loss is denoted as L_{PC}.
- Network features, i.e. maximally activating each unit of the network layers.

Besides these auxiliary control tasks, an auxiliary reward is used, which is to predict the reward (if positive or negative) for the next frame given a certain history of observations. This loss is denoted as L_{RP}. Here an additional simple feedforward network which stacks encoded (by the CNN) states outputs this prediction. This simplifies the temporal aspects of the reward prediction task. Another interesting contribution is the value function replay (besides a simplified version of prioritised replay [Schaul et al., 2015]) which resamples recent sequences with random varying temporal position of the truncation window of the n-step return, and can be seen as value iteration. This has the loss denoted by L_{VR}. This makes use of the features discovered by the reward prediction. The final agent is referred to as the UNREAL agent (Unsupervised Reinforcement and Auxiliary Learning) and its overall loss

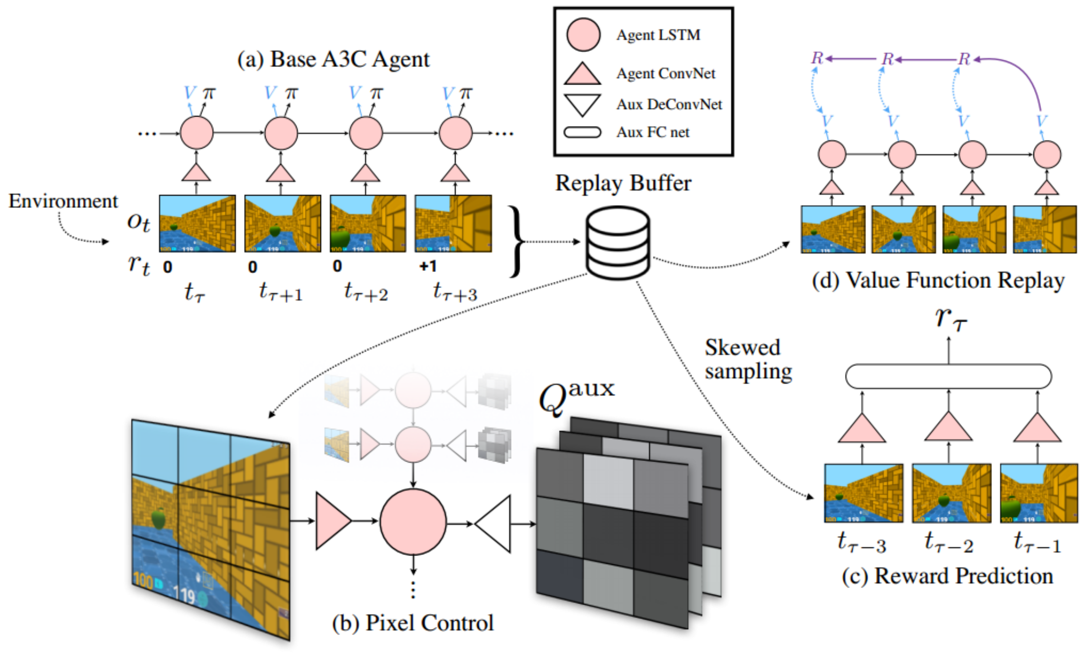


Figure 6: Depiction of the UNREAL agent. Figure from [Jaderberg et al., 2016]

.

is given by:

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

where the λs are the loss weighting terms. We note that the auxiliary tasks were trained on the very recent history of experiences. We observe here a very interesting principle at work. The agent is biased towards having some kind of internal regularization forced by these auxiliary tasks. This significantly speeds up the learning process through narrowing of the policy and value function. These auxiliary losses can be seen as some kind of proxies of the true reward function. But as the reward function gives very sparse rewards, the agent needs to have some type of feedback, even if it does not completely describe the desired objective. The processes described above are depicted in Figure **??** We will see in the next research paper a similar approach, where some additional experiments show the fact that the performance of the final policy learned does not transfer back to the original auxiliary losses, which means that indeed, the set of good policies for the auxiliary losses is a superset of the set of good policies adapted to the environment.

5.3.5 Loss is its own Reward: Self-Supervision for Reinforcement Learning [Shelhamer et al., 2016]

Very similar to the previous work, the authors of this paper consider different losses as auxiliary rewards. The main difference is the fact that, while the previous paper added the loss terms to the original loss function, in this work, the auxiliary losses are used before the original objective, which means that they function as a pre-training mechanism. Similar as above, reward binning is used, but also new auxiliary losses are introduced. First of all, a very important concept is the fact that the dynamics can be learned (at least approximately), without the need for specific model-based reinforcement learning. The problem is cast as a classification problem where the two classes are plausible transitions, i.e. pairs of (s, s') which can be found in the environment, meaning such a transition is possible and pairs of (s, s') for which there is no such transition, so they do not belong to the environment. So the agent will try to differentiate the two, learning the dynamics (or a surrogate of it) in its internal representation, without the need for a specific model. The action value can also be added to this pair. Another interesting addition is figuring out the inverse dynamics, i.e. what actions are plausible given a pair of (s, s'). Again this is cast as a classification problem for discrete actions and regression problem for continuous actions.

5.3.6 Incentivizing exploration in reinforcement learning with deep predictive models [Stadie et al., 2015]

A very simple and elegant idea which has been used before in one form or another (MBIE-EB [Strehl and Littman, 2008a], BEB[Kolter and Ng, 2009]) is to assign some reward for discovering new states in the environment. This work shows a very simple way of assigning rewards to novel states by having a model of the environment and then computing a distance (Euclidean) between the seen states and the model-predicted state. if this difference is big enough then assign a significant reward. Formally, the reward function is modified as:

$$R_{mod}(s,a) = R(s,a) + \beta\mathcal{N}(s,a)$$

where $\mathcal{N}$ is a novelty function. We will see in a moment what form this has. Furthermore, assume a model of the environment is given by $\mathcal{M}$ parameterized by ϕ, with $\sigma(s)$ being the encoding of a state s. $\mathcal{M}$ predicts the next state from the current state and the action. The distance between the prediction and the actual state is computed as:

$$e(s_t, a_t) = ||\sigma(s_{t+1} - \mathcal{M}_\phi(\sigma(s_t), a_t))||_2^2$$

The novelty function $\mathcal{N}$ is defined as:

$$\mathcal{N}(s_t, a_t) = \frac{\tilde{e}_t(s_t, a_t)}{t \times C}$$

where $\tilde{e}$ is a normalized version of the above e and $C > 0$ is a decaying variable. This is a very general framework for using predictive modelling to explore, in the sense that any function can take the place of σ or $\mathcal{M}$. The authors choose for σ an 8-layer autoencoder is used, retrained every 5 epochs and for $\mathcal{M}$ a simpler two layer neural network is used and is retrained every epoch (50k observations). In the experimental section we then see how this approach improves especially where the human performance was much better than the original DQN.

5.3.7 Variational Information Maximizing Exploration [Houthooft et al., 2016]

One other very interesting exploration strategy is using the same notion of surprise or curiosity we have mentioned multiple times until now, as well as model based reinforcement learning. In short, the concept of surprise is quantified with respect to the model, the agent is encouraged to find states that induce large changes in the model dynamics after being discovered. The approach is based on variational inference, while for the model dynamics Bayesian Neural Networks (BNN) are used. We proceed with a more formal description. The dynamics of the environment is denoted by $p(s_{t+1}|s_t, a_t; \theta)$ where θ are the parameters of the model, in this case of the BNN, which are set to be the parameters of a fully factorized Gaussian distribution. Even though this is quite simplifying, it is still rich enough and the KL divergence between the posterior and the prior has an analytical simple form. The history of the states and actions until timestep t is denoted by $\xi_t = \{s_1, a_1, ..., s_t\}$. The parameters θ being updated in a Bayesian manner, we can talk about uncertainty of the parameters, and thus the dynamics, and so an agent would like to *maximize the reduction in uncertainty about the dynamics* or maximizing the following quantity:

$$\sum_t (H(\Theta|\xi_t, a_t) - H(\Theta|S_{t+1}, \xi_t, a_t))$$

for a sequence of actions. S_{t+1} is the next state distribution and Θ is the random variable associated with the above mentioned value θ. The important starting observation is that these individual terms are equal to the mutual information between the next state distribution and the model parameters, which is given by:

$$I(S_{t+1};\Theta|\xi_t,a_t) = \mathbb{E}_{s_{t+1}\sim p(\cdot|\xi_t,a_t)[D_{KL}[p(\theta|\xi_t,a_t,s_{t+1})]]}$$

We can see that the agent will take actions to increase this information about the dynamics, thus this can be seen as *information gain*. This term is added as intrinsic reward to the overall reward function, weighted by a parameter η which controls how much weight the need for exploration has for the overall behaviour of the agent. However the quantity added is usually intractable so an approximate variational distribution is considered and optimized by minimizing the well known variational lower bound. The authors show also a very interesting connection with compression improvement [Schmidhuber, 2009] which gives the reverse form of the KL penalty mentioned above. Training of the BNN used in the model uses many practical considerations, like sampling the weights θ, the reparametrization trick and sampling neuron pre-activations which is cheaper and reduces the variance of the gradient than the actual activations. Experience replay is used as well. The experiments used to test this approach are all control tasks with a relatively small state space dimensionality (maximum 33) and action space dimensionality (maximum 6), albeit both are continuous. The performance improvement is quite significant on some problems where the rewards are highly sparse. This is a very promising approach, theoretically justified, and is highly related to the work summarized previously [Mohamed and Rezende, 2015].

References

[Abdolmaleki et al., 2015] Abdolmaleki, A., Lioutikov, R., Peters, J. R., Lau, N., Reis, L. P., and Neumann, G. (2015). Model-based relative entropy stochastic search. In *Advances in Neural Information Processing Systems*, pages 3537–3545.

[Badre, 2008] Badre, D. (2008). Cognitive control, hierarchy, and the rostro–caudal organization of the frontal lobes. *Trends in cognitive sciences*, 12(5):193–200.

[Baird III, 1993] Baird III, L. C. (1993). Advantage updating. Technical report, DTIC Document.

[Baram et al., 2016] Baram, N., Zahavy, T., and Mannor, S. (2016). Deep reinforcement learning discovers internal models. *arXiv preprint arXiv:1606.05174*.

[Botvinick and Weinstein, 2014] Botvinick, M. and Weinstein, A. (2014). Model-based hierarchical reinforcement learning and human action control. *Phil. Trans. R. Soc. B*, 369(1655):20130480.

[Brunskill and Li, 2014] Brunskill, E. and Li, L. (2014). Pac-inspired option discovery in lifelong reinforcement learning. In *ICML*, pages 316–324.

[Charikar, 2002] Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM.

[Deisenroth and Rasmussen, 2011] Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

[Deisenroth et al., 2013] Deisenroth, M. P., Neumann, G., Peters, J., et al. (2013). A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142.

[Diuk et al., 2013] Diuk, C., Schapiro, A., Córdova, N., Ribas-Fernandes, J., Niv, Y., and Botvinick, M. (2013). Divide and conquer: hierarchical reinforcement learning and task decomposition in humans. In *Computational and robotic models of the hierarchical organization of behavior*, pages 271–291. Springer.

[Duan et al., 2016] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

[Fischer, 1980] Fischer, K. W. (1980). A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological review*, 87(6):477.

[Foerster et al., 2016] Foerster, J. N., Assael, Y. M., de Freitas, N., and Whiteson, S. (2016). Learning to communicate to solve riddles with deep distributed recurrent q-networks. *arXiv preprint arXiv:1602.02672*.

[Garnelo et al., 2016] Garnelo, M., Arulkumaran, K., and Shanahan, M. (2016). Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*.

[Gottlieb et al., 2013] Gottlieb, J., Oudeyer, P.-Y., Lopes, M., and Baranes, A. (2013). Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*, 17(11):585–593.

[Gu et al., 2016] Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). Continuous deep q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*.

[He et al., 2016] He, F. S., Liu, Y., Schwing, A. G., and Peng, J. (2016). Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*.

[Heess et al., 2015] Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. (2015). Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*.

[Hengst, 2004] Hengst, B. (2004). Model approximation for hexq hierarchical reinforcement learning. In *European Conference on Machine Learning*, pages 144–155. Springer.

[Houthooft et al., 2016] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117.

[Jaderberg et al., 2016] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

[Kakade, 2002] Kakade, S. (2002). A natural policy gradient. pages 1531–1538.

[Kakade and Langford, 2002] Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274.

[Kakade et al., 2003] Kakade, S. M. et al. (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University of London.

[Klyubin et al., 2005] Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). Empowerment: A universal agent-centric measure of control. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE.

[Kolter and Ng, 2009] Kolter, J. Z. and Ng, A. Y. (2009). Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 513–520. ACM.

[Kulkarni et al., 2016a] Kulkarni, T. D., Narasimhan, K. R., Saeedi, A., and Tenenbaum, J. B. (2016a). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *arXiv preprint arXiv:1604.06057.*

[Kulkarni et al., 2016b] Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016b). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396.*

[Lai and Robbins, 1985] Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22.

[Lakshminarayanan et al., 2016] Lakshminarayanan, A. S., Krishnamurthy, R., Kumar, P., and Ravindran, B. (2016). Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359.*

[Levine and Koltun, 2013] Levine, S. and Koltun, V. (2013). Guided policy search. In *ICML (3)*, pages 1–9.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971.*

[Lin and Tegmark, 2016] Lin, H. W. and Tegmark, M. (2016). Why does deep and cheap learning work so well? *arXiv preprint arXiv:1608.08225.*

[Littman, 2015] Littman, M. L. (2015). Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553):445–451.

[McGovern and Barto, 2001] McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density.

[Menache et al., 2002] Menache, I., Mannor, S., and Shimkin, N. (2002). Q-cutdynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*, pages 295–306. Springer.

[Michalski et al., 2014] Michalski, V., Memisevic, R., and Konda, K. (2014). Modeling deep temporal dependencies with recurrent grammar cells"". In *Advances in neural information processing systems*, pages 1925–1933.

[Miller and Cohen, 2001] Miller, E. K. and Cohen, J. D. (2001). An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, 24(1):167–202.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783.*

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[Mohamed and Rezende, 2015] Mohamed, S. and Rezende, D. J. (2015). Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2125–2133.

[Narasimhan et al., 2015] Narasimhan, K., Kulkarni, T., and Barzilay, R. (2015). Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941.*

[Ng and Jordan, 2000] Ng, A. Y. and Jordan, M. (2000). Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc.

[Oh et al., 2015] Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. (2015). Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871.

[Osband et al., 2016] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. *arXiv preprint arXiv:1602.04621*.

[Parisotto et al., 2015] Parisotto, E., Ba, J. L., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*.

[Peters et al., 2005] Peters, J., Vijayakumar, S., and Schaal, S. (2005). Natural actor-critic. In *European Conference on Machine Learning*, pages 280–291. Springer.

[Pollard, 2000] Pollard, D. (2000). *Asymptopia: An exposition of statistical asymptotic theory*.

[Rajendran et al., 2015] Rajendran, J., Lakshminarayanan, A., Khapra, M. M., Ravindran, B., et al. (2015). A^2T: Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources. *arXiv preprint arXiv:1510.02879*.

[Rusu et al., 2015] Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*.

[Schapiro et al., 2013] Schapiro, A. C., Rogers, T. T., Cordova, N. I., Turk-Browne, N. B., and Botvinick, M. M. (2013). Neural representations of events arise from temporal community structure. *Nature neuroscience*, 16(4):486–492.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[Schmidhuber, 1991] Schmidhuber, J. (1991). Curious model-building control systems. In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pages 1458–1463. IEEE.

[Schmidhuber, 2009] Schmidhuber, J. (2009). Simple algorithmic theory of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes.

[Schulman et al., 2015a] Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015a). Trust region policy optimization. In *ICML*, pages 1889–1897.

[Schulman et al., 2015b] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[Shelhamer et al., 2016] Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. (2016). Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*.

[Shi and Malik, 2000] Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905.

[Silver et al., 2014] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395.

[Şimşek and Barreto, 2009] Şimşek, Ö. and Barreto, A. S. (2009). Skill characterization based on betweenness. In *Advances in neural information processing systems*, pages 1497–1504.

[Şimşek et al., 2005] Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pages 816–823. ACM.

[Stadie et al., 2015] Stadie, B. C., Levine, S., and Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

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

[Strehl and Littman, 2008b] Strehl, A. L. and Littman, M. L. (2008b). Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1417–1424.

[Sutton, 1990] Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[Sutton et al., 1999a] Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063.

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

[Tang et al., 2016] Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). # exploration: A study of count-based exploration for deep reinforcement learning. *arXiv preprint arXiv:1611.04717*.

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

[Tessler et al., 2016] Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016). A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE.

[Van Hasselt et al., 2015a] Van Hasselt, H., Guez, A., and Silver, D. (2015a). Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*.

[Van Hasselt et al., 2015b] Van Hasselt, H., Guez, A., and Silver, D. (2015b). Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*.

[Wang et al., 2016] Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

[Wang et al., 2015] Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

[Watter et al., 2015] Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.