Reactive Reinforcement learning for Robotic Manipulation

A Thesis

submitted to

Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Ameya Pore

Indian Institute of Science Education and Research Pune

Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April 2019

Supervisor: Dr Gerardo-Aragon Camarasa

<u>Certificate</u>

This is to certify that this dissertation entitled **Reactive reinforcement learning for robotic grasping** towards the partial fulfilment of the BS- MS dual degree programme at the **Indian Institute of Science Education and Research, Pune** represents study/work carried out by **Mr. Ameya Pore** at the **University of Glasgow** under the supervision of **Dr Gerardo-Aragon Camarasa, Lecturer, School of Computing Science** during the academic year 2018-19.

[illegible]

[illegible]

Mr. Ameya Pore

Dr. Gerardo-Aragon Camarasa

This thesis is dedicated to *Unicorn* (National Animal of Scotland)

In the hope that the words inscribed here manifest magical powers and anyone who read this thesis finds love and happiness forever!

Note: Unicorns are imaginary animals and hence they are difficult to find. Traditionally, they are believed to have magical healing powers.

<u>Declaration</u>

I hereby declare that the matter embodied in the report entitled **Reactive reinforcement learning for robotic grasping** are the results of the work carried out by me at the **School of computing Science, University of Glasgow**, under the supervision of **Dr. Gerardo-Aragon Camarasa** and the same has not been submitted elsewhere for any other degree.

[illegible] [illegible]

Mr. Ameya Pore Dr. Gerardo-Aragon Camarasa

Acknowledgements

It has been a long journey and a significant transition, from the Biology Labs at IISER Pune to the computing Science department at Glasgow, from Epigenetics to Reinforcement learning. A year of varied experiences, from severe summers to harsh winters!

I'd first extend my gratitude to my second home, IISER Pune, for actively engaging in this exchange mobility program with University of Glasgow. Noteworthy acknowledgements to the International team at IISER Pune (headed by Dr. Naresh Sharma), Deans and other distinguished faculty members who have thrived to build an international reputation and research co-operation at global scale. Being only a decade old, IISERs stand one of the elite education institute in the country. Besides focusing on research based-academics, I feel glad that IISERs promote student-driven activities. I would distinctively acknowledge one specific club, the Entrepreneurship and Innovation cell (EIC), which I was part of for couple of years during my stay at IISER. Special thanks to Prof. Sanjeev Galande, Aditya Kabra who have pushed me beyond my capabilities to explore innovation, participate in startup-culture events and form a network. They have helped me immensely in developing various skills pertaining to communication, presentation and team management.

Next, I'll thank Prof. William Cushley (Dean for Global engagement, S&SE Asia, UoG), Mr. Sanju Dominic (International officer, India, UoG), Ms Sally Baxter (Local coordinator, Erasmus team) and the Erasmus+ team who have been my first contact related to the administrative queries and documentation. It would have been extremely difficult without their funding support to accomplish this project successfully. They have always made sure that my stay at Glasgow is advancing fruitfully.

Most importantly, I'd be indebted to my project supervisor, Dr. Gerardo, who has kept me motivated throughout, despite times when my experiments didn't work out well. I appreciate the informal chat sessions we had occasionally, that helped in maintaining a constructive relation. I still wonder his risk in accepting a student with Biology background without a shred of prior computing science training. I hope I have lived up to his expectations. Further, I will extend my gratitude to Ali-al-quallaf (PhD student) who has been humble enough to teach me the basics of various framework such as *Visual studio code, Pytorch* etc that have been the foundation to the overlying work. Moreover, I will thank other academics, doctoral students and visiting researchers of the Computer vision and automated systems group (CVAS) who have provided important feedback to my project during group meetings.

Additionally, I will thank Dr Rahul Dehiya (TAC member), who has supported me despite all ventures in various documentation and applications for future prospects. Although, not involved directly in Machine learning, he has shown keen interest in the timely updates that I have given him.

Lastly, thanks to the hidden pillars, parents, teachers, relatives, friends (new and old) without whom it’s difficult to imagine life. Special thanks to Tans, who has trained me into an expert cook.

Finally, some words on Scotland, it is blessed with splendid scenes, picturesque landscapes that are unparalleled. Certainly, travelling around Scotland were among the big perks of studying at Glasgow University. Best of all, I enjoyed the strange weather, where one could experience Rain, Snow, Sun, wind at one time. Completely unpredictable!

Cheers and ready for upcoming expeditions!!

Contents

List of figures:

List of tables

Abbreviations used:

1. TD: Temporal difference
2. DQN: Deep Q-Network
3. e2e: End-to-end
4. HRL: Hierarchical Reinforcement learning
5. RL: Reinforcement learning
6. IR: Intrinsic Reward
7. A3C: Asynchronous Actor-Critic
8. A2C: Synchronous Actor-Critic -
9. PPO: Proximal policy approximation
10. TRPO: Trust Region Policy Optimization
11. DNN: Deep Neural network
12. MDP: Markov decision process
13. LSTM: Long short-term memory
14. ICM: intrinsic curiosity module
15. Brain Anatomy: Specified in the context

Reactive Reinforcement learning for Robotic Manipulation

1. Abstract

Robots have transformed the manufacturing industry and have been used for scientific exploration in human inaccessible environments like distant planets, Oceans, etc. However, a major barrier in its universal adoption is its lack of fragility and robustness in a complex and highly-diverse environment. This project constitutes the initial steps towards flexibility regarding exploration strategies that can be applied to challenging problems for autonomous grasping of rigid and deformable objects. Here, we employ recent advances in Deep Reinforcement learning (RL) to generate simple reactive behaviours like approaching, manipulating and retracting to pick an object. Once such simple behaviours are learnt, these could be sequenced in various combinations to give rise to a complex task.

RL is a trial and error optimisation technique where an agent ought to take action in an environment to maximise some notion of cumulative reward. Current research in RL has been formulated on traditional techniques such as Deep Q-learning and policy gradient methods. These methods have worked well when the feedback/reward is dense. Perhaps, in real-life scenarios, the feedback is sparse, and these methods tend to fail in finding the optimum solutions and exploring the environment robustly. In this work, we have implemented two different approaches to solve such sparse reward problem, namely *Curiosity* and *Reactive behaviour repertoire* for long time step tasks. Our results have shown an immense reduction in training steps required to reach the maximum reward state in high-dimensional continuous action space compared to the baselines.

Chapter 1

Introduction

Autonomous robotic manipulation is one of the active areas of research for its vast expanses of application in homes, factories and operating rooms. Inspired by the Moravec's paradox (Moravec, 1988), in robotics, performing simple low-level behaviours like object grasping tends to be more difficult than higher-order logical reasoning skills like solving a puzzle. A classic example is the game of chess. In 1997, IBM's deep blue chess computer defeated the then world champion, but perhaps even after two decades, it needs a human being to do the actual moving of the pieces predicted by the algorithm. Current state-of-the-art robots are trained in highly controlled environments and specialise in a narrow set of skills. On the contrary, humans can adapt to many environments and tasks based on previous experience and knowledge.

1.1 *Background*

Since the advent of Deep learning and high-computing facilities, there have been various breakthroughs in broad applications such as pattern recognition, computer vision, and natural language processing. Quite recently, deep learning techniques combined with reinforcement learning (RL) have shown surprising results where useful feature representations are learnt from high dimensional raw-input data such as images and mapped to perform an optimal action (Mnih *et al.*, 2015, 2016; Schulman, Levine, *et al.*, 2015; Schulman *et al.*, 2017). This research gives us the capability to solve decision-making tasks such as autonomous robotics, self-driving cars and many more, that were intractable for centuries.

RL defines a Markov decision process (MDP) (Sutton and Barto, 1998) where an agent learns by interacting with the environment, observing the results of the action and receiving the reward associated with the transition (Fig 1). Mathematically, MDP is a framework for modelling discrete time decision-making situations where the outcome is partly stochastic and partly under the control of the agent. At each discrete time step, the agent sends an action to the environment, and the environment responds by emitting the next observation and transition reward. RL enables a robot to find an optimum behaviour autonomously through trial and error. It offers to robotics a framework to model hard-to-engineer and sophisticated behaviours (Kober and Peters, 2014). Instead of explicitly designing or hand-engineering solutions, the controller only needs to give feedback in terms of a scalar quantity to label whether the action taken is correct or incorrect.

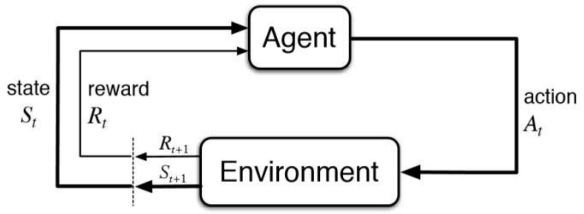


Fig 1. Markov Decision process: Agent in current state S_t interacts with the environment by executing action A_t. As a consequence of this interaction, it reaches a new state S_{t+1} and received a reward R_t. Markovian assumption: The probability of future states depends only on the dynamics of the present state and not on the sequence preceding the present state.

Two broad families of deep learning algorithms have shown promise on RL problems so far: Q-learning methods such as DQN (Mnih *et al.*, 2015), policy gradient methods (Sehnke *et al.*, 2010) (e.g., A3C (Mnih *et al.*, 2016), TRPO (Schulman, Levine, *et al.*, 2015) , PPO (Schulman *et al.*, 2017)).

Deep Q-learning algorithms approximate the optimal Q function with Deep Neural Nets (DNN), yielding policies. Here, policy refers to the action the agent would take given a state. That is, for each given state, the agent chooses an action that maximises the Q-value (Watkins and Dayan, 1992; Mnih *et al.*, 2015; Hessel *et al.*, 2017). Policy gradient methods directly learn the parameters of a DNN policy (Sehnke *et al.*, 2010). The technique optimises the policy and outputs the probability of taking each action in each state. In the context of this project, there have been recent advances in robotic grasping that try to imitate human performance given a large set of tasks, and RL techniques are being applied in an object-centric demonstration so that the algorithm selects the most feasible demonstrations to replicate (Gupta *et al.*, 2016). On top of this, meta-learning algorithms have been employed which use a lesser amount of demonstrations, and that backpropagate to match the current actions with the demonstrated actions, also called *one-shot imitation* (Duan *et al.*, 2017). Further, there have been studies of context translation for imitation learning based on video prediction (Liu *et al.*, 2017). Another study that has shown promising results is *Zero shot visual imitation* where an agent first explores the world without any expert supervision and then transforms its experience into a goal-conditioned skill policy (Pathak *et al.*, 2018).

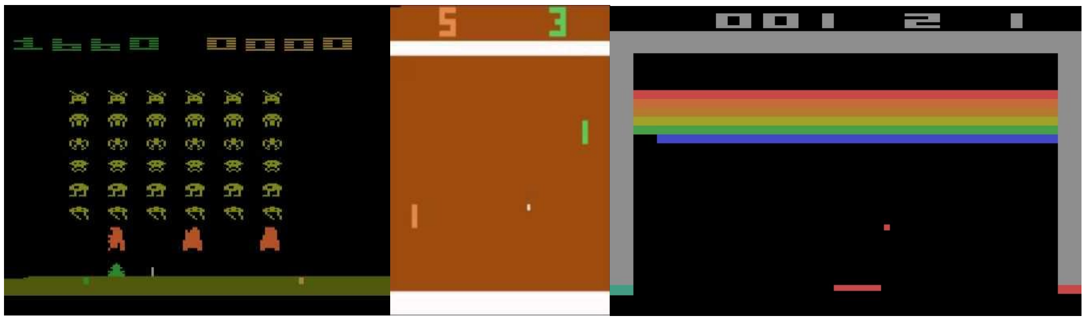


Fig 2. RL validation environments: Video games provide a constrained environment for testing RL algorithms. Demonstrated here are the classic Atari-2600 games from the 80's. From left to right: *SpaceInvaders*, *Pong*, *Breakout*.

Using human-like dexterous hands possesses challenges concerning high dimensionality of possible actions and physical contacts. Recently, there have been model-free Deep reinforcement learning approaches that have shown favorable results trained in addition to Human demonstrations (Rajeswaran *et al.*, 2017). On the contrary, there have been adaptive-learning approaches that train real robots on simulations to cope with higher learning times, sparse and hard to sample useful actions that be validated in Physical robots (Breyer *et al.*, 2018). But perhaps, behaviours developed in specific simulators are biased towards the characteristics of the simulator. A report lately claimed to bridge this gap by randomising dynamics of the simulator while training and to develop policies that are capable of adapting to different dynamics (Peng *et al.*, 2017). Further, a group from MIT developed a method for object representation called Dense Object Nets (Florence, Manuelli and Tedrake, 2018). This work devised a consistent object representation for visual understanding and manipulation and can be trained quickly on previously unseen objects.

One of the major challenges with robotic grasping is the sparse reward (i.e., for most of the task, the agent receives no reward). Therefore, various studies have devised an intrinsically motivated goal exploration process that enables an agent to sample effective policies in a continuous high dimensional action space where rewards are sparse (Pathak *et al.*, 2017; Laversanne-Finot, Péré and Oudeyer, 2018). These methods use prediction errors of networks trained on the agent's experience to quantify the novelty of new experience and has shown effective results in robot locomotion skills (Forestier, Mollard and Oudeyer, 2017). Curiosity-driven exploration or intrinsic motivation works well for short-horizon/timestep tasks. However, it fails to explore when the tasks are long-time horizon. The reason for this has been elaborated in section

1.2 *Approach*

A reason for RL algorithms struggling in long-time horizons and sparse rewards could be assigned to 1-step Temporal Difference (TD) methods (i.e. at each step the agent tries to minimize the errors computed between predictions made at consecutive time-steps), which are limited to forming reward assignment one-time step at a time (Ke *et al.*, 2018). In addition, learning multiple behaviours end-to-end (e2e) leads to the problem of the Curse of Dimensionality [33]. Curse of Dimensionality refers to a scenario when there is a rapid growth of a function caused by the combinatorics of the number of states in an RL problem.

We, therefore propose a reactive Reinforcement learning architecture, inspired by Subsumption architecture (Rodney A Brooks, 1990; Brooks, 1991), to learn low-level skills independently of each other. Once these low-level skills are acquired, they could be sequenced in different temporal combinations using a high-level actuator we call

Temporal Cortex, to give rise to a diverse repertoire of complex behaviours. We hypothesise that modular reactive behaviours can be trained and sequenced to generate diverse repertoire to skills. Each behaviour shall consist of simple networks trained specifically for that skill. This behaviour-based repertoire would be more efficient, general and transparent in contrast to task-specific e2e learning approach.

1.3 Contributions and Dissertation outline

Implementing algorithms on real-world robotic systems tend to possess various challenges like transfer learning, real-world hardware issues. Hence, we evaluate our algorithms in simulated video-games and robotic environment that gives a constrained and controlled domain. Besides, we compare our approach against the existing baseline (*Actor-critic*) for e2e learning. Our method outperforms the benchmark by a considerable margin, and there is a drastic reduction in the training steps required to learn a specific behaviour.

This thesis is organised as follows: Background review regarding Deep Reinforcement learning and Hierarchical decomposition of tasks is described in chapter 2, with recent advances in Hierarchical RL. In chapter 3, we describe the experimental setup and implementation details that are used to validate the proposed approach. We present our experimental results and comparison of reactive behaviour training with e2e approaches in chapter 4. Our findings are summarised and discussed in chapter 5. Lastly, we end with the main highlights of our work in chapter 6 and future implications and perspectives in chapter 7.

Chapter 2

Background

Deep Networks creates a hierarchical representation of raw inputs, from low-level to higher-level features (LeCun, Bengio and Hinton, 2015). Combining Deep-learning with traditional RL foundations enables an agent to have a good perception of the environment and learn control-policies from high-dimensional data input such as images (Mnih *et al.*, 2015). In RL, an agent interacts with an environment and learn from trial and error. As a consequence of this interaction, it receives a reward, and the goal is to learn to select actions the maximise the expected cumulative reward (Sutton and Barto, 1998).

An RL agent acts in an MDP framework as follows: At each time step *t,* in discrete time intervals, the agent selects an action a from a set of possible actions $A = 1,2,3, \dots k$ at state $s_t \in S$, where S is the set of possible states. Action selection is based on a policy π. The policy is a behaviour descriptor of an agent which decides the actions to be taken for each possible state. As a result of the action, the agent receives a scalar reward $r_t \in R$ and observes the next state $s_{t+1} \in S$. The aim of the learning agent is to find a mapping from states to actions called optimal policy $\pi *$, which choose the action a to take given a state S, maximising the cumulative expected rewards r. The expected discounted return R at time t is defined as follows:

$$R_t = E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots] = E[\textstyle\sum_{k=0}^{\infty} \gamma^k r_{t+k}] \qquad \dots(1)$$

Where $0 < \gamma < 1$is called the discount factor. Next, we define an action-value function $Q^{\pi}(s, a)$, which gives the expected return achievable starting from state s, $s_t \in S$, and performing an action $a \in A$, and then following policy π.

$$Q^{\pi}(s,a) = E_{\pi}[R_t | s_t = s, a_t = a]$$

$$Q^{\pi}(s,a) = E_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a] \qquad \ldots(2)$$

To iteratively find the value of each state-action pair, we start at the "end" (i=0) and then work backward to find Q_{i+1}. From equation 2, we can generate a recursive relation:

$$Q_{i+1}^{\pi}(s,a) = E_{\pi}[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1} | s_t = s, a_t = a]$$

$$= E_{\pi}[r_t + \gamma Q_i^{\pi}(s_{t+1} = s', a_{t+1} = a') | s_t = s, a_t = a]. (3)$$

Now, to learn the optimal policy π* to estimate the trajectory to achieve the greatest future reward in the course of execution, we define an optimal state-value function as follows:

$$Q_{i+1}(s,a) = E_{\pi}[\, r_t + \gamma max_{a'} Q(s', a') | \, s, a] \qquad \ldots(4)$$

Where $s', s \in S$ and $a', a \in A$. This iteration ideally converges to the optimal action-value function, $Q*$ as $i \rightarrow \infty$ and called **value iteration algorithm** or **Bellman equation** (Watkins and Dayan, 1992).

2.1 Deep Reinforcement learning

In real world complex problems, the number of possible states and actions are enormous, and it is impractical computationally to estimate the Q-value for each state-action pair. Therefore, we require a function approximator as an estimator for action-value function. Traditionally, these function approximator have been trivial non-linear approximator such as Decision trees, nearest neighbours etc. (Pyeatt and Howe, 1999), which possess an inherent limitation of inability to approximate non-trivial solutions with high number of optimisation parameters. On the contrary, recent advent in DNN gives us capability to approximate highly complex function with huge number of parameters (in the order of millions) and generalisation potential. Hence, lately, In the context of Q-learning, DNN's have shown surprising results in solving RL problems (Mnih *et al.*, 2015, 2016; Schulman, Levine, *et al.*, 2015; Schulman *et al.*, 2017). Here, we consider a network parametrized by vector θ to estimate the value function $Q(s, a\, ;\, \theta)$.

These parameters θ are optimised using gradient-descent methods by minimizing the following loss function.

$$L(\theta) = E_{\pi}\left[\left(r_t + max_{a'}Q(s', a'; \theta) - Q(s, a; \theta)\right)^2\right] \qquad \ldots(5)$$

Equation 5 is also called 1-step temporal difference (TD) where $r_t + max_{a'}Q(s', a'; \theta)$ is the target value. Here, we compute the error in prediction made at successive time steps. Intuitively, the expectation of our current estimate $Q(s, a;\ \theta)$ are better at future time step than at current time step t because the further one goes in time, the more rewards one collects and eventually the accumulated rewards would be the unbiased estimator to the true Value function $Q(s, a)$. This technique is known as **Deep Q-Network (DQN)**.

Differentiation of the loss function yields the equation for the gradient.

$$\nabla L(\theta) = E\left[\left(r_t + max_{a'}Q(s', a'; \theta) - Q(s, a; \theta)\right)\right]\nabla Q(s, a; \theta) \qquad \ldots(6)$$

$$\theta \leftarrow \theta + \alpha\ \nabla_{\theta}L(\theta)$$

Although, the mathematical framework for this algorithm was proposed in early 90’s (Sutton and Barto, 1998), there were practical issues that needed to be acknowledged, which delayed its application in real-systems until 2015. First, parameters used (weights of network: θ) were the same for estimating the Target Q-value ($r_t + max_{a'}Q(s', a'; \theta)$) and the current Q-value ($Q(s, a;\ \theta)$). Consequently, there is a big correlation between the TD error and the parameters (θ) we are changing. It means that at each training step, our Q-value shifts but also our target Q-value shifts. Hence, the algorithm is not stable because we are getting closer to the target, but the target is moving, resulting in high variability. Thereby, for successful implementation of DQN (Mnih *et al.*, 2015), Google Deepmind[1] maintained two networks parameterised by different weights and alternately switched between these networks for learning and feeding in current action-value estimates as "bootstraps". The two networks denoted here are $Q(s', a'; \theta^{-})$ and $Q(s, a;\ \theta)$, parametrised by different parameters θ^{-} and θ.

[1] Google Deepmind: Pioneer Artificial Intelligence research company based in London (est: 2010)

Therefore, New update rule:

$$\nabla L(\theta) = \left(r_t + max_{a'}Q(s', a'; \theta^-) - Q(s, a; \theta)\right)\nabla Q(s, a; \theta) \qquad \dots(7)$$

At each time step

$$\theta \leftarrow \theta^-$$

Other modifications include Off-policy learning. It means that the policy used to generate a behaviour, usually called the *behaviour* policy, could be different from the policy that is evaluated and improved, called the *estimation* policy. An advantage of this separation is that the *estimation policy* can be deterministic (e.g. greedy), while the *behaviour policy* can continue to sample all possible actions. Improving the policy will lead to different behaviours that should explore actions closer to optimal ones, and the agent want to learn from those. Second, using an experience replay. Experience reply is a memory buffer that stores the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$. For training, the Q-Network randomly samples the data from the buffer, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution. Advantage of using a replay buffer is that previous experience can be used multiple times for learning. Also, the Q-value updates are incremental but converge slowly, thereby training with the same data is beneficial.

Deep Q-learning has worked efficiently for simple Video games (Atari), yet it struggles to find a convergent solution in continuous action space (Lillicrap *et al.*, 2015). Continuous action space refers to continuous variable such as torque, force, velocity etc.

Hence, a different approach for solving RL problems has evolved parallelly, called the policy gradient methods (Sehnke *et al.*, 2010). Policy gradient methods directly map states to a distribution of actions and directly learn the parameters of the approximator. Each time the agent interacts with the environment, parameters of the neural network are tweaked so that "good" actions are sampled more frequently. For having a mathematical framework, let's assume that we have a policy $\pi_\theta(s, a)$, parameterized by θ such that

$$\pi_\theta(s, a) = P[a|s, \theta] \qquad \dots(8)$$

The goal is to find θ such that it maximises the policy objective R_{t,π_θ} function, under policy π_θ defined as:

$$J(\theta) = E\left[R_{t,\pi_\theta}\right] = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right] \quad \ldots(9)$$

To solve the optimisation problem, calculus approach is used to find the gradient of J, denoted by $\nabla_\theta J(\theta)$. We keep updating J with its gradient until it converges and reaches a maximum, given by the policy gradient formula.

$$\nabla_\theta J(\theta) = E\left[\nabla_\theta log_{\pi_\theta}(s,a) R_{t,\pi_\theta}\right] \quad \ldots(10)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Equation 9 is famously known as the REINFORCE algorithm (Sutton and Barto, 1998). Although, REINFORCE works well, however it is extremely sample inefficient (Fig 3). We must wait until the end of the episode to calculate the reward. We may conclude that if we have a high reward R_{t,π_θ}, all the actions we took during the episode were good, even if some were bad.

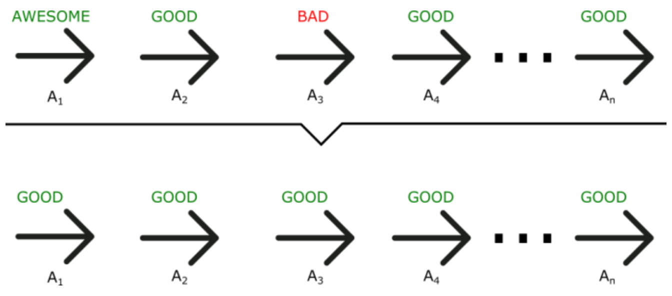


Fig 3. Different trajectories. Upper: All actions except A_3 fetched positive reward. Lower: All actions fetched positive reward. REINFORCE algorithm is unable to distinguish the two scenario and requires high amount of averaging, since rewards are computed at the end of the episode.

For example, in Fig 3, even if A3 was a bad action, all the actions would be averaged as good because total reward was good. Consequently, to find an optimal policy, we require enormous number of samples. This results in slow learning and more time to converge to a specific solution.

Summarising Deep Reinforcement learning methods, there are two broad categories:

- Value based methods (Q-learning, DQN): the aim is to estimate the value function (using function approximator) that maps each state-action pair to a value. Further, we take actions corresponding to highest value. This works well for discrete actions and finite states.
- Policy based methods (REINFORCE): the goal is to directly learn a function that maps each state to an action. We directly optimise policy without using a value function. We use total rewards acquired in the episode as a measure of novelty of the policy.

Quite recently, it was shown that both these methods could be merged to increase the learning efficiency of the agent and update our policy at each time step (Lillicrap *et al.*, 2015; Mnih *et al.*, 2016). This method is known as the *actor-critic*.

2.1.1 Actor-Critic

In *actor critic*, we have two networks that are trained simultaneously

- *Critic* network: which returns the novelty of the action (value- based scalar, $Q_w(s,a)$)
- *Actor* network: which controls the agent and decides the action to take ($log\ \pi_\theta(s,a)$)

New update rule

$$\Delta\theta = \alpha\nabla_\theta\big(log\pi_\theta(s,a)\big)Q_w(s,a) \qquad \ldots(11)$$

Since, we make an update at each step, we cannot use the total reward R(t). Instead, we train a *critic* model that approximates the value function (it estimates the expected cumulative future reward given a state-action pair).

At the beginning, when both the networks are untrained, the *actor* outputs a random action and the *critic* provides a random feedback to the action (Fig 4). Learning from this feedback, the *actor* updates its policy. Simultaneously, the *critic* also updates its network so that it can give better feedback next time.

Hence, two functions are estimated parameterized by two different sets of weights.
Actor (parameterized by θ): $\pi_{\theta}(s, a)$
Critic (parameterized by w):$Q_w(s, a)$

Both these networks update their parameters separately. Following is the update rule.

$$\Delta\theta = \alpha\nabla_{\theta}\left(log\pi_{\theta}(s, a)\right)Q_{w}(s, a) \qquad \ldots(12)$$

$$\Delta w = \beta\left(r_t + \gamma max_{a}' Q_{w}(s', a') - Q_{w}(s, a)\right)\nabla Q_{w}(s, a) \qquad \ldots(13)$$

As mentioned earlier, value-based methods tend to have high variability. To reduce this problem, an advantage function $A(s, a)$is defined as follows:

$$A(s, a) = Q(s, a) - V(s) \qquad \ldots(14)$$

This function determines the improvement in our action-value compared to the average value of that action in that state. It calculates the extra reward the agent would get if it takes this action.
If $A(s, a) > 0$: our gradient is pushed in that direction.
If $A(s, a) < 0$: our gradient is pushed in the opposite direction.

Equation 14 can also be written as

$$A(s, a) = r + \gamma V(s') - V(s) \qquad \ldots(15)$$

Which denotes the TD error similar to equation 6. So, the final update rule for *actor critic* is as follows:

$$\Delta\theta = \alpha\nabla_{\theta}\left(log\pi_{\theta}(s, a)\right) A_{w}(s, a) \qquad \ldots(16)$$

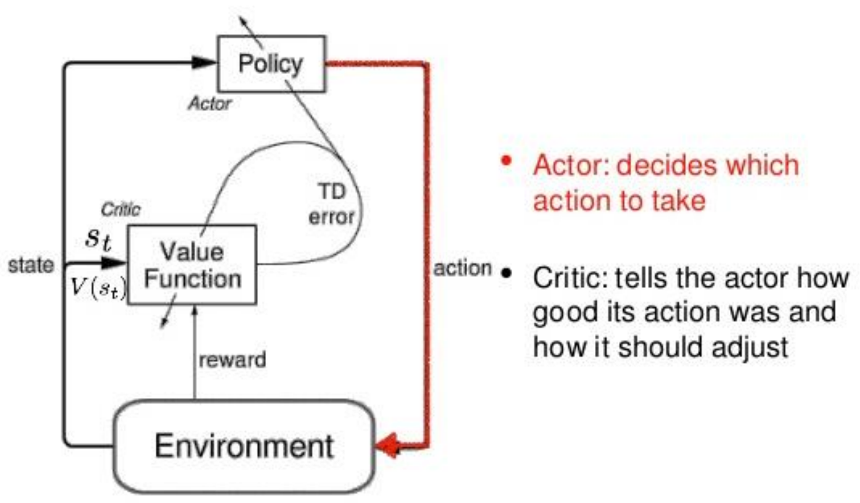

Policy
Actor
Critic
TD error
s_t
$V(s_t)$
state
Value Function
reward
action
Environment
Actor: decides which action to take
Critic: tells the actor how good its action was and how it should adjust

Fig 4 Dynamics of *actor critic*. *Actor* takes input from the state and *critic* to produce an action.

2.2 Curiosity-driven learning

The main challenges in implementing RL in robotic grasping are reward sparseness and high dimensionality of action space. Reward sparseness refers to a scenario wherein the agent receives no reward in most of the states of the environment. To cope with the above problems, studies are being conducted on simpler environments with limited action space such as exploratory video games such as Super Mario Bros, Doom etc. One way to tackle the problem of reward sparseness is through Domain-specific reward shaping (Gupta *et al.*, 2016). An example of reward shaping for the Mario environment would be about providing a reward whenever the agent moves to the right. A major fallback of this approach is that it is not flexible to other environments and one needs to know the context of the environment (in case of Super Mario Bros, the goal is towards the right) to bias rewards.

A recent study showed that one could form an intrinsic reward mechanism, called curiosity, that enables an agent to sample effective policies in a continuous high dimensional action space where rewards are sparse (Pathak *et al.*, 2017; Laversanne-Finot, Péré and Oudeyer, 2018). The method uses prediction errors of networks trained on the agent’s experience to quantify the novelty of a new experience. It learns a next-state predictor model from its experience and uses the prediction error as an intrinsic reward. As a result, the agent actively attempts to seek unpredictable, new and complex regions of the environment to maximise prediction error and receive a reward. This study has shown a drastic reduction in training time compared to other baselines.

Intrinsic motivation such as curiosity provides a RL agent with intrinsic rewards that encourage exploration; it works best only for short Horizon environments. When it comes to completing a task with long timestep, Curiosity-driven exploration faces the so-called *detachment* issue (Ecoffet *et al.*, 2019).

Detachment: Intrinsic reward (IR) is a consumable resource. A curious agent looks for states that it has not visited. If we consider a scenario where the agent discovers multiple regions of state space which gives a high intrinsic reward. In the short term, its policy may focus on one of such areas. After exhausting the IRs in that area, the policy may by chance start exploring other areas offering IR, due to ε-greedy exploration strategy. Once it has exhausted the IR, it is difficult for it to rediscover the frontier it detached from initially as it has already consumed the IR that led to the frontier. Each time such a process happens a potential avenue of exploration could be lost.

For long-range planning, we need to learn and operate at a different level of temporal abstraction. Hierarchical reinforcement learning (HRL) offers such a framework in which multiple layers of policies are trained to perform decision-making and control at the successively higher level of behavioural and temporal abstractions (Frans *et al.*, 2017; Levy *et al.*, 2017; Gomes, Oliveira and Christensen, 2018; Nachum *et al.*, 2018). Having a hierarchy of policies, a complex task can be decomposed into subtask sequences (also called Options), which are themselves built by simpler actions. The lowest-level policy of the hierarchy (subordinate actions) apply actions to the environment whereas the higher-level policies (basic actions) are trained over a longer time scale. Here, we propose a reactive architecture to generate a behaviour based repertoire. Once, a set of reactive behaviours is generated, we sequence these primitive behaviours using a bottom-up training strategy to complete complex tasks.

2.3 Behaviour-based robotic control

The RL methods discussed above are model-free approaches and work well for an e2e state-action mapping. An e2e model learns all the features that can occur between the original state inputs (x) and the final outputs (y). It refers to training a possibly complex learning system by applying gradient-based learning to the entire system (Glasmachers, 2017). Given the computing power and Deep learning framework advances, most of the applied RL algorithms use an e2e approach. However, these model-free approaches do not scale well for more extensive and complex tasks like Robotic manipulation (Lillicrap *et al.*, 2015). Also, if we look at the Human learning pattern of solving a task, it is quite intuitive that we don’t follow an e2e approach. For, example, while learning Tennis, we start with learning basic skills separately such as bouncing the ball, hitting etc., whereas an e2e approach would attempt to optimise all the possible actions involved at once. Optimising all possible actions requires a large amount of trial and error interactions. Humans learn skills in 50-100 attempts whereas a Deep learning agent requires more than 5 million trials using a sophisticated model-free RL algorithm to complete a trivial task on simulations. Also, there has not been enough success in transfering the knowledge from a trained simulator to a real robotic system, called the **reality gap** (Collins, Howard and Leitner, 2018).

Instead of having a task directed goal, we propose a framework to generate reactive behaviours. Reactive behaviour refers to a set of techniques for action selection by autonomous agents. They differ from classical approaches in the way that behaviours operate in timely order and can cope with difficulties related to highly dynamic and unpredictable environments. The idea is motivated from quite famous architecture that was developed in the 1980s for robotic control by Rodney Brooks, called Subsumption architecture (Rodney A Brooks, 1990; Brooks, 1991). In this architecture each behaviour is represented as a separate layer, having direct access to sensory information. Each layer has an individual specific goal. In this work, we use a trivial form of imitation learning, called Behaviour cloning (Nakanishi *et al.*, 2004; Bojarski *et al.*, 2016), to train these distinct behaviour layers. Currently, the low-level reactive behaviours are choreographed by the developer.

There have been approaches to model behaviours (Frans *et al.*, 2017; Levy *et al.*, 2017; Nachum *et al.*, 2018) but the results are not convincing enough for the comparison with baseline model-free approaches due to the following reasons:

1. HRL algorithms autonomously decide how to segment the main task into sub-tasks (Options). This segmentation is highly task specific i.e. the decomposed subtask once trained would hardly be able to generalise to different task.
2. Most HRL implementation use a top-down optimisation i.e. Given a main task, first the algorithm optimises for segmentation of the subtasks. Once the segmentation is completed, it optimises the low-level actions to take for each subtask. Both these optimisation takes place simultaneously, referred to a e2e learning.

To tackle the above-mentioned challenges, in our work, we use manual inputs to decompose the task into basic behaviours. An advantage of this is that these decomposed behaviours once trained could be used to accomplish different tasks and are not highly specific to a task. Further, we use a bottom-up approach for training these modular behaviours. We train all the modular behaviour independently and then combine them sequentially to complete the task.

Humans roughly tend to assume a hierarchical structure. For example, tea and coffee making, the overall task can be subdivided into discrete subtasks (adding sugar, adding cream). Our goal is to expand the basic RL framework to include temporally abstract actions, and learn representations that group together similar intercalated actions (For example: grasping the cutlery, using spoon to scoop up sugar, moving the spoon over the cup etc.) casting them as a single higher-level action or skill ("add sugar''). These actions are described as temporally abstract as they describe low-level action sequences extended over time (Fig 5).

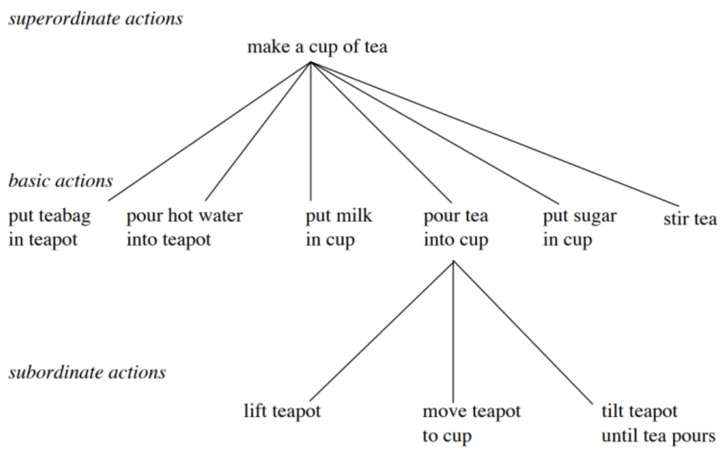


Fig 5 Hierarchical setup: Decomposition of the main task into several subtask. For completing each of this subtasks, subordinate actions are carried to comprise a basic action. These basic actions put together give rise to superordinate actions.

Our resulting architecture is a parallel neural network that maps the positional coordinates and kinematic state stimuli inputs to action outputs via a learned and distributed internal representation. Note that we demonstrate our experiments on a simplistic simulator which provides kinematic state vector as an input and does not operate on a camera-based vision system.

2.3.1 Subsumption Theory

Brooks believed in following the evolutionary path of intelligence of starting with simpler behaviours. After a successful basic design, one can extend to higher level intelligence. Subsumption architecture is a layering approach that connects perception to actions for robot control systems(Rodney A. Brooks, 1990; Brooks, 1991). ” Subsumption” refers to the coordination process between non-identical layered behaviours. The complex actions subsume a set of simpler behaviours. A task is accomplished by activating the appropriate layer, which then enables the lower layers below it.

Concurrently, research based on reward-based learning in neuroscience and behavioral studies has started convergence to canonical model of Hierarchical behaviours decomposition and RL. It has been shown that particular regions in the Prefrontal cortex implicate in learning the rules of the environment using trial and error strategy (Maggi, Peyrache and Humphries, 2018).

2.5 Neuroscientific implications

Just after the mathematical framework for actor-critic was developed, it was shown that in human midbrain, some neurotransmitters give a paradigm of temporal-difference reward signals for goal-directed behaviour. Dopamine (DA), in particular, has been associated with the function of signaling reward prediction error (Wang *et al.*, 2018). It has also been established that sections of Prefrontal Cortex (Ventral striatum (VS), see Table1) represent the expected values of actions, objects and states. Drawing correspondence with the actor in RL, there are sections (Dorsolateral striatum (DLS), see Table1) that are considered for extract representations that guide temporally integrated, goal-directed behaviour. Apart from the actor, there are regions (Dorsolateral prefrontal cortex (DLPFC) and Premotor cortex (PMC), see Table 1) that represent task sets that code for particular sequences of low-level actions (Botvinick, Niv and Barto, 2009). A single activation pattern in these neurons represents a map from stimuli to responses, i.e. policy. Research on Frontal Cortex (FC) also aligns with HRL in a way that temporally abstract actions organise into hierarchies, with higher level policy (e.g., making coffee) calling a lower-level policy (e.g., adding sugar). This research presents evidence that FC represents actions at multiple nested levels of temporal structure (Frank and Claus, 2006).

Under HRL, in addition to a top-level value function, the critic must also maintain a value function for each option. In terms of neural structure, another area in PFC (Orbitofrontal cortex (OFC), see Table 1) shows a strong connection with the option-specific actor (Dorsolateral prefrontal cortex (DLPFC) and Premotor cortex (PMC), see Table 1). OFC neurons have been extensively implicated in determining values of events.

Table 1 - Key relations between HRL and human brain anatomy with the functional details.

	Anatomy	Function	Details
1	Dorsolateral striatum (DLS)	*Actor*	Coordinates higher level actions
2	Dorsolateral prefrontal cortex (DLPFC), Premotor cortex (PMC)	Option specific *actor*	Builds representation of specific temporally abstracted action.
3	Ventral striatum (VS), Dopaminergic system	*Critic*	Coordinates higher level value function of the event
4	Orbitofrontal cortex (OFC)	Option-specific *critic*	Estimates option specific state values.

Having established a reward-driven system following a hierarchical structuring in neural anatomy, an interesting question to explore is how sensory inputs or proprioceptions mapped to actions. One such instance would be understanding how vision inputs are processed and replicating such systems.

2.5.1 Visual pathway

Visual information from the retina passes through the thalamus to the primary visual cortex. Although the visual processing mechanisms are not yet completely comprehended, physiological and anatomical studies in monkeys suggest that visual signals are fed into two separate processing systems, also known as the two-stream hypothesis (van Polanen and Davare, 2015). The two streams- Dorsal and ventral have different functions. Ventral stream is associated with the identification of the object, whereas the dorsal stream is involved in the guidance of the action and spatial localisation of the object. The dorsal stream retrieves information regarding the object identification from the ventral stream. This dorsal stream links to areas in the prefrontal cortex to give rise to goal-directed visual behaviour (Sciberras-Lim and Lambert, 2017).

The synergies with human and animal behaviour explicitly related to the visual pathway motivate our implementation. Our model is canonical to the organisation of the visual pathway. In our architecture, we have a feature extraction layer (similar to the Dorsal and ventral stream). Next, we train modular reactive behaviours (identical to motor cortex). Further, we link the feature extraction layer to the reactive behaviours manually such that these behaviours are sequenced to complete a specified task. In neural system, this manual input link depicts the connection between the dorsal stream and Prefrontal cortex. For simplicity and to showcase our proof-of-concept, we use a simulator that operates on kinematic state vector as input. However, further steps of this implementation would be to scale our experiments on a vision-based simulator.

Summarising our hypothesis: reactive reinforcement learning architecture gives us a direction towards creating digital algorithms that could imitate psychological accounts of hierarchically organised behaviour.

Chapter 3

Experiments and Implementation details

In this chapter, we report implementation of two different experimental setups:

1. Actor-critic and Intrinsic curiosity module (Pathak *et al.*, 2017): Note that both these models have been published previously. However, the published code appears to be loosely organised with inappropriate labelling drafted in Tensorflow. Here, we have devised and executed an independent code of the same models in Pytorch (Deep-learning framework that supports accelerated GPU processing with multiple parallel threads) that is more intuitive to the readers. This work serves as a baseline to implement state-of-the-art RL algorithm.
2. Reactive network architecture: In this work, we present an architecture, inspired from Subsumption architecture, in which simple fully connected networks are trained to generate reactive behaviours. This is a very preliminary implementation where each of the modular behaviours are trained using demonstrations (Behaviour Cloning) and sequenced manually to accomplish a task.

3.1 Advantage *Actor-critic* (A3C)

In the case of DQN, the agent uses a single environment to train its policy. On the contrary, advantage actor-critic utilises multiple processing to run an agent in multiple environments parallelly. The algorithm explores a more significant portion of state-space in much less time by having multiple independent workers with their own set of parameters in a separate environment. Actor-critic is an on-policy policy search method, i.e. they learn and act simultaneously using the same policy.
There are two types of parallel processing actor-critic implementation based on their synchronicity of update:

1. **Asynchronous *Actor-critic* (A3C)**
2. **Synchronous *Actor-critic* (A2C)**

In A3C, there is a global network with multiple workers functioning in parallel threads with their own set of network parameters (Fig 6). Each worker trains separately and updates the global network periodically. The global network holds shared parameters. The updates do not happen simultaneously and hence it's called asynchronous. After each update, the agents reset their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again. Although, the experience of each agent is independent of the experience of others, yet there is information flow between the agents as each agent resets their weights by the global network, which contains the shared parameters.

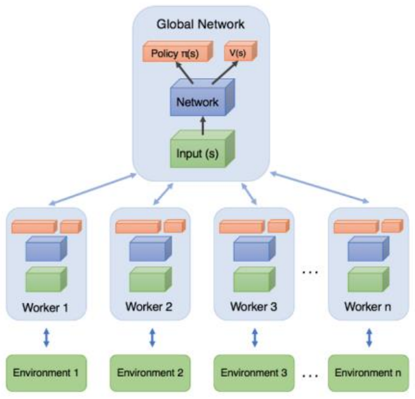


Fig 6. Schematics of Asynchronous *actor-critic*: Parameters of the shared global network are updated asynchronously by parallel workers functioning simultaneously. Each worker consists of separate network for *actor* and *critic*.

In the case of A2C, the update of global network parameters happens synchronously. Each agent waits for other agents to complete their segment and then update the global network weights and reset the worker agent parameters. It is speculated in recent reports that A2C performs well in some tasks compared to A3C (Wu *et al.*, 2017). The main drawback of asynchrony is that some agents play with older version parameters that result in a decrease in performance.

A3C was first implemented by Google deepmind in 2016 and produced a huge impact on the RL community due to its simplicity, robustness and drastic increase in performance compared to policy gradient and DQN approaches. It was shown that A3C could beat all other existing RL benchmark algorithms in the standard Atari video games. In the first part of this project, we implement A3C to solve the *SuperMarioBros* environment.

3.1.1 Environment details

Super Mario Bros. is a video game (Fig 7) produced and published by Nintendo. It consists of a player taking the role of Mario. The objective is to race through the kingdom, collect maximum points, eliminate the enemies and save the princess. The game world highlights coins spread around the level and special bricks labelled with a question mark (?), which when hit from underneath by Mario, fetches it rewards. Mario's primary attack is bouncing on top of enemies to eliminate them. The game consists of 8 worlds with 4 sublevels called "stages" in each world [1]. *SuperMarioBros* is a challenging game compared to Atari 2600 games as it requires exploration and knowledge of different enemies.

Fig 7 SuperMarioBros environment: The agent discovers how to play and reach the end flag. The environment is rendered using the Nintendo Entertainment system (NES) emulator.

3.1.2 Architecture details

The input state S_t is passed through a succession of four convolution layers with 32 filters each, kernel size 3x3, stride 2 and padding 1. An ELU (exponential linear unit) is used after each convolution. In order to build temporal connections between previous actions and states, LSTM (Long short-term memory) layer is used with 256 inputs. The output of the LSTM is passed through two separate fully connected layers that are used to predict the action and the value function. This architecture used is motivated by Generalised advantage estimation (GAE) which showed that a single network could be used to predict the action and the value function (Schulman, Moritz, *et al.*, 2015). The initial parameters for feature extraction from the state are shared between the two networks.

3.1.3 Training details

The Mario environment used has been derived from Kaunteja Github [2]. Here, the action space is discretised to seven simple actions, using Nes_py wrappers [3]. Moreover, to accelerate the training time, the original image size (240 x 256 RGB) is downscaled to (84 x 84 grey-scale). Further, the state representation ($S_t S_t$) of the environment is built by concatenating the current frame with the previous three frames. This is done to model temporal dependencies Also, there are 16 workers used in parallel to speed up training. The code of the implementation can be found in github [4].

3.2 Intrinsic curiosity module

The model is based on an *actor-critic* model. This *actor-critic* model is integrated with an Intrinsic curiosity module which computes the prediction error as mentioned earlier.

The intrinsic curiosity module consists of a forward and an inverse model (Fig 8). The inputs of the model are the present state S_t, action a_t (sampled from the policy trained on *actor-critic*), and next state S_{t+1} (which the agent reaches after taking action a_t). First, the state tensors are passed through four convolution layers to extract the features $\phi(S_t)$ and $\phi(S_{t+1})$ corresponding to S_t and S_{t+1} respectively. The forward model is constructed by concatenating $\phi(S_\mathrm{t})$ with a_t and passing it into a sequence of two fully connected layers, the output being $\phi'(S_{t+1})$. The inverse model concatenates $\phi(S_t)$ and $\phi(S_{t+1})$ to output a'_t. Training this neural network accounts to learning functions g and f defined as:

$$a'_t = g(S_t, S_{\mathrm{t+1}}; \theta_\mathrm{i}) \qquad \dots(17)$$

$$\phi'(S_{t+1}) = f(\phi(S_\mathrm{t}), a_\mathrm{t}; \theta_\mathrm{f}) \qquad \dots(18)$$

Where θ_i and θ_f are neural network parameters for the inverse and forward model respectively. Here a'_t is the predicted estimate of the action and $\phi'(S_{t+1})$ is the predicted estimate of $\phi(S_{t+1})$.

The parameters θ_i and θ_f are optimised by minimising the following loss functions L_i and L_f respectively

$$min_{\theta_i}[L_i\ (a'_t, a_t)] \qquad \ldots(19)$$

$$L_f\big(\phi(S_{t+1}),\ \phi'(S_{t+1})\big) = \left(\left|\left|\phi'^{(S_{t+1})} - \phi(S_{t+1})\right|\right|^2\right) / \ 2 \qquad \ldots(20)$$

The intrinsic reward signal r_t^i is computed as

$$r_t^i = \left(\left|\left|\phi'^{(S_{t+1})} - \phi(S_{t+1})\right|\right|^2\right) \eta / \ 2 \qquad \ldots(21)$$

The total loss function of the network is given as

$$min_{\theta_p, \theta_i, \theta_f}\left[-\lambda E_{\pi(S_t, \theta_p)}[\Sigma_t r_t] + (1-\beta)L_i + \beta L_f\right] \qquad \ldots(22)$$

Where $0 \leq \beta \leq 1$ and $\lambda > 0$ are scalars that weighs the inverse model loss to the forward model loss and the significance of the policy gradient loss to the significance of learning the curiosity reward signal respectively.

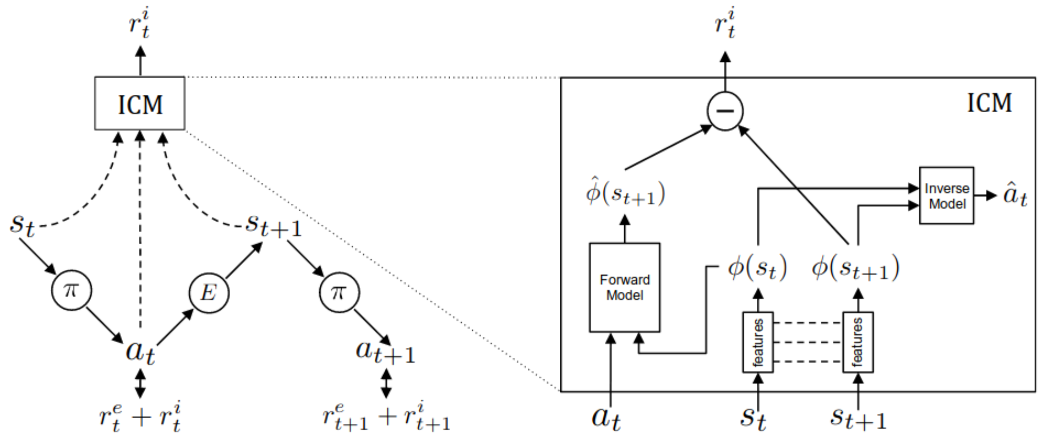


Fig 8: Intrinsic curiosity module: The agent in state s_t takes action a_t sampled from the policy. As a result, it goes to a new state s_{t+1}. The policy π is trained to optimise the sum of extrinsic r_t^e and intrinsic r_t^i rewards. The environment offers the extrinsic reward and the intrinsic reward is generated by the Intrinsic curiosity module (ICM). ICM extracts the features of state s_t and s_{t+1} into features $\phi(s_t)$ and $\phi(s_{t+1})$. The forward model takes input $\phi(s_t)$ and a_t and predict the features $\hat{\phi}(s_{t+1})$ of the next state s_{t+1}. Curiosity-driven intrinsic reward r_t^i is the prediction error in the feature space. The inverse model takes $\phi(s_t)$ and $\phi(s_{t+1})$ as inputs to predict the action $\hat{a_t}$. The actions $\hat{a_t}$ and a_t are compared and used for backpropagation to encode only features in $\phi(s_t)$and $\phi(s_{t+1})$ which affect the dynamics of the agent.

3.2.1 Experiments

We evaluate the performance of the learned policy qualitatively and quantitatively using the proposed Intrinsic curiosity module in two different setting on the *SuperMarioBros* environment: Dense reward, Sparse reward. In the original work, the authors have evaluated the algorithm on a different environment: Vizdoom with three different reward setting: Dense ,sparse and very sparse rewards.

The sparseness of the reward is determined by the distance it needs to explore in order to receive reward. A larger distance means the probability of reaching the goal by random exploration is low and hence the reward is said to be sparse. In the dense reward setting, the agent receives the reward after each timestep. In Sparse reward setting, agent receives reward after 10% of the total game distance. We compare two different algorithms

1. ICM+A3C - Refers to our full algorithm that combines ICM with Actor-critic
2. A3C - Baseline

The original work has considered another algorithm:

3. ICM-pixel + A3C - Refer to a variant of ICM without the inverse model.

ICM-pixel is close to ICM, but it is incapable of removing feature representations for uncontrolled part of the environment. It only uses the forward model to compute the prediction error in the Observation feature space.

3.3 Reactive reinforcement architecture

In this work, we consider the task of picking a block and taking it to a target position. The environment used for validation is *FetchPickandPlace* [5], part of an open-source library designed by OpenAI.

3.3.1 Environment details

Gym is an open-source module that provides a toolkit for developing and comparing RL algorithms. It contains collection of various environments to test RL algorithms on such as Atari video games: pong, pinball and continuous control environment like Humanoid walking, jumping etc.

FetchPickandPlace- For continuous control tasks like Robot manipulation, gym uses a physics simulator called mujoco. A physics simulator is a software that provides an approximate simulation of specific physical systems like rigid-body dynamics(collision), soft-body dynamics, fluid dynamics. *FetchPickandPlace* simulated the Fetch research platform (Fig 9). A goal is randomly chosen in 3D space. RL algorithms should control Fetch's end effector (using inverse kinematics) to grasp and lift the block up to reach that goal. Instead of a vision-based camera, this simulation provides the state input in the form of coordinates and orientation of block, gripper position and its velocity and rotation. This scenario is a sparse reward condition as the agent receives a reward (+1), only when it successfully takes the block to the target position. In all other circumstances, it gets a reward of (-1).

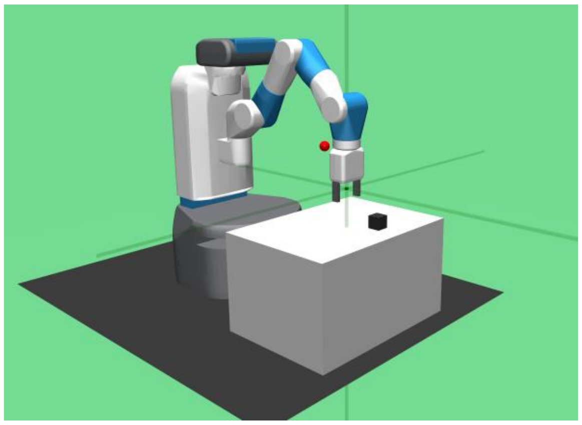


Fig 9 OpenAI *fetchpickandplace* environment: Fetch is a robot with a single arm manipulator and four degrees of freedom. The goal is to pick the block and take it to the target position (Red colored dot)

First, we decompose the main task into basic skills/behaviours. The primary goal is to pick and take the block to a target location whereas the sub goals are primitive behaviours such as approaching, manipulating a block and retracting to a target point (Fig 10). The idea we propose here is canonical to human learning pattern in order to complete a task. We learn skills separately and then combine them to accomplish a task. Similarly, we train the robot to master each skill independently. For, example, if it is approaching towards the block, we explicitly teach it to move in X, Y and Z cartesian space. We further repeat this for other sub actions. Once these sub actions are learned, we combine them to pick the block. The network which decides the movement of the end-effector in x,y,z are trained to accomplish that specific subtask. Currently the input on which sub action to take (e.g., approach(a), manipulate(b) and retract(c)) is given by the developer, but our future work involves developing a highly reactive network based on activation and inhibition mechanism which fires in a particular temporal sequence to give rise to complex behavior. We use a simple loss function to imitate from demonstration instead of learning new policies from scratch.

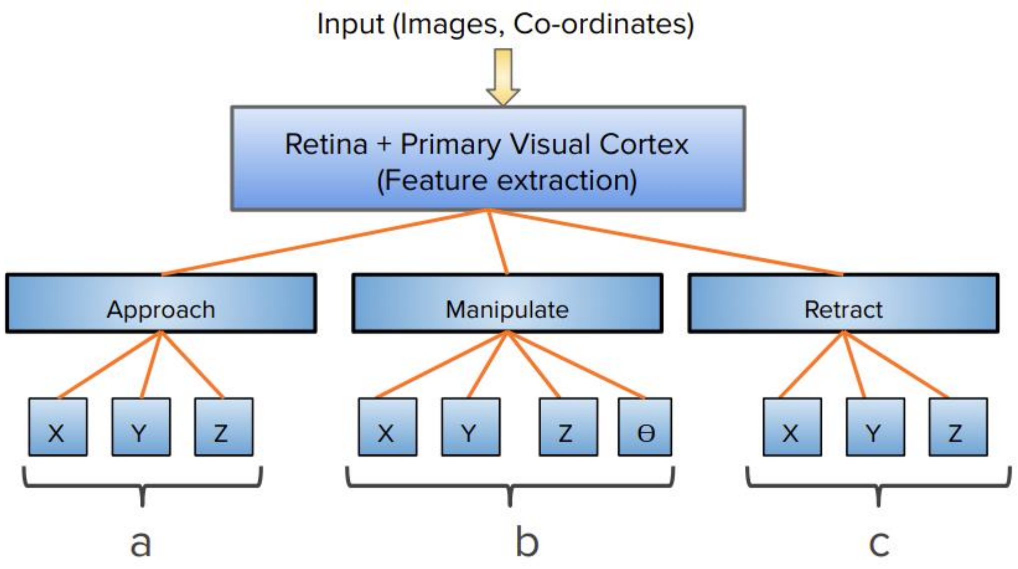

Input (Images, Co-ordinates)
Retina + Primary Visual Cortex (Feature extraction)
Approach
Manipulate
Retract
X
Y
Z
X
Y
Z
Θ
X
Y
Z
a
b
c

Fig 10: Schematics of reactive behaviour architecture: The primary goal of picking the object is subdivided into simpler subtasks. For each subtask, there are reactive networks which are trained specially for movement in x, y and z. First, the state vector (in the form of coordinates of objects) is given as an input to the Feature extraction layer. The extracted features are relayed on to the reactive layers deciding the movement of the end-effector. To simplify terminology, we use the following corresponding letters for denoting the subordinate actions: Approach (a), manipulate (b) and retract (c).

3.3.2 Architecture details

The raw input of coordinates is fed to feature extraction network consisting of two fully connected layers with 128 neurons each. The output is mapped to three neurons in the last layer that determines the movement in x, y and z for each process approach(a), manipulate(b) and retract(c).

3.3.3 Experimental details

For training the robot, we use a conventional method in imitation learning, called Behaviour cloning (BC) which learns a policy through supervised learning on demonstration state-action pairs (Nakanishi *et al.*, 2004; Bojarski *et al.*, 2016). More advanced forms of imitation learning include Inverse Reinforcement learning (IRL), where a reward function is deduced from the demonstrations. Here, we assume the knowledge of the reward function, therefore omit the comparison with IRL. We introduce a loss function computed on the demonstration examples as follows:

$$L_{BC} = \sum_{i=1}^{N_D} \left|\left|\pi(s_i|\theta_\pi) - a_i\right|\right|^2$$

Employing this loss, the learned policy is always near the demonstrated policy, as the actor is always tied back to demonstrations. The code for the implementation can found at Github [6]. Here, we have evaluated four different experimental setups:

1. Sequential- First, we train the approach (a) behaviour with a hand-engineered solution for manipulate (b) and retract (c). Once process *a* is trained (reaches 100% success rate), process *b* is trained using the output from a network and hand-engineered solution for process *c*. Similarly scaling up, once process *b* is learnt, we train on process *c*, using outputs from network *a* and *b*.
2. Sequential + Freezing- In this case, we start with training approach (a), with a hand-engineered solution for process *b* and *c*. After process *a* is trained, we freeze the layers that extract features from the raw input (In fig-10, retina+primary visual cortex). Further, we train process *b*, using the freezed weights for feature extraction and the output of network *a*, and hand-engineered solution for *b*. Similar method is applied for scaling up and training on process *c*.
3. Separate- In this case, we start with training approach (*a)* with a hand-engineered solution for process *b* and *c*. Once, *a* is trained, we do not use the output of the trained network. For training process *b*, we use hand-engineered solution for *a* and *c*. Similarly, training is carried out for process *c*. Once, each module is trained separately, use combine all the skills to accomplish the task.
4. Separate + Freezing - In this case, we start with training approach with a hand-engineered solution for process *b* and *c*. After process a is trained, we freeze the layers that extract features from raw input (In fig-10, retina+primary visual cortex). For training process *b*, output from network trained for process a is not used, we use hand-engineered solution for *a* and *c* with frozen weights of the feature

extraction layer. Similarly, training is carried out for process *c*. Once, each module is trained separately, use combine all the skills to accomplish the task.

Intuitively, the freezing model is motivated by the early development of some aspects of visual pathway. Infant visual brain is not completely mature. It poorly characterises traits such as acuity, shape, and color perception. Gradually, some visual capacities (for example: motion perception) mature over 6-8 months, whereas some properties take time to become plastic and adult like. Some research opinion suggests that the initial interaction of newborn's triggers rapid innate circuiting of some parts of the brain (subcortical structures) and eventually this rate slows down with age (Buiatti *et.al*, 2019). Similarly, we train our network for an initial period in approach (a). Once, the network optimises the parameters to extract patterns from the data; we use the same weights for further training on process b and c.

Chapter 4

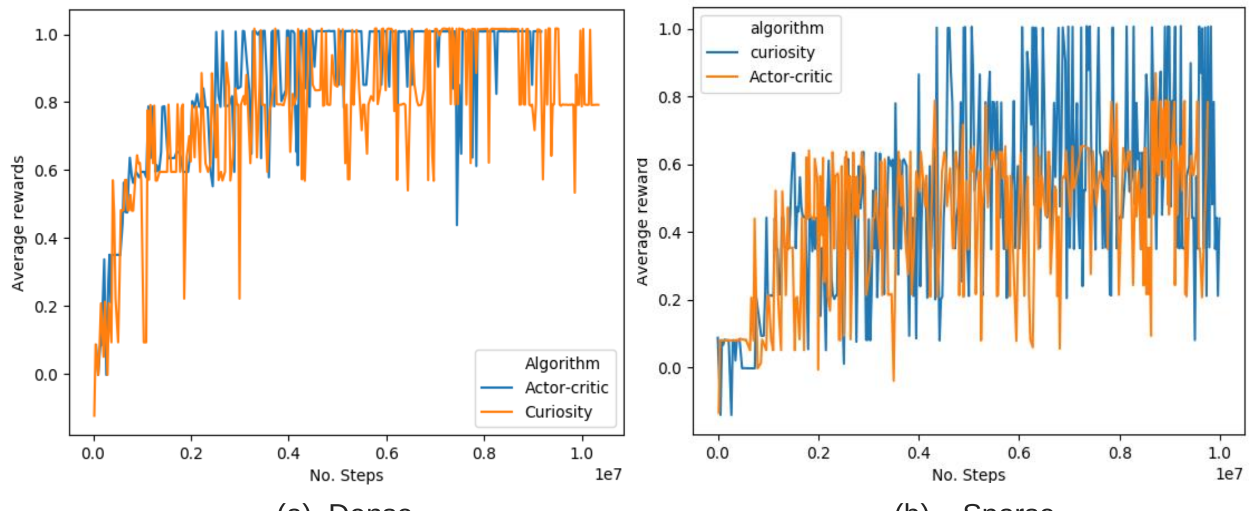
Results and Discussion

Following are the results of the implementation

1. Comparison of Actor-critic and curiosity model
2. Comparison of e2e learning and our proposed hierarchical model

4.0 Actor-critic + Curiosity

Results in Fig 10 depicts that the performance of the baseline A3C degrades with sparser rewards, ICM+A3C can achieve superior results. In dense reward setting, curious agent tends to learn faster suggesting more exploration compared to ϵ-greedy exploration of the environment. In Sparse condition, baseline fails as expected, whereas the curious agent is still able to explore effectively and learn the task quickly.



(a) Dense (b) Sparse

Fig 11: Performance comparison of the A3C agent without curiosity (yellow) against the full ICM (blue) in two different reward setting (left to right: dense to very sparse). The results suggest that as the exploration gets harder, A3C finds it difficult to explore. ICM+A3C is able to perform well in the scenarios. The exploration becomes

difficult as the agent goes from Dense to Sparse rewards. For each experiment, three independent runs are carried out.

We further experiment with training the Mario agent based only of curiosity rewards without any extrinsic rewards. Surprising, the Mario agent learns to cross about 30% of level-1. The agent received no reward signal for killing the enemies or jumping over pits, yet it discovered these behaviours autonomously. One reason for it to learn these behaviours of killing is only then it could reach a new part of the game space to receive curious rewards. In other words, curiosity provides indirect supervision for learning new behaviours

4.0.1 Discussion

One of the major challenges at the frontiers of Machine learning research is how to explore the parameter space efficiently. In conditions when the feedback is continuous, standard techniques such as backpropagation with hyper-parameter tuning work best to find optimal solution. However, when extrinsic rewards are sparse, it becomes extremely difficult for current RL algorithms to explore and reach out to “novel” states, since it is not incentivised to do so and it is highly unlikely that it would reach the pre-specified goal through random exploration. Existing algorithms find it difficult to scale up to high-dimensional visual inputs such as images and predict pixel values to next state due to stochasticity of environment. Curiosity generated intrinsic reward outperforms other methods because it only predicts the changes in the environment that could possibly be due to agents’ actions or affect the agent. Instead of predicting the raw sensory space, intrinsic curiosity module transforms raw images to a feature space and predict features that affect the dynamics of the agent. It ensures that the exploration strategy of the agent is unaffected by nuisance factors in the environment beyond agents’ control.

4.1 Reactive Robotic Behaviour

As mentioned in the implementation we compare the existing e2e methods, where the actions are optimised for the entire task from the first timestep, to our proposed reactive architecture. In our reactive structure, the main task is broken down into subtasks and each task is learnt individually. We evaluate four different scenarios namely - Sequential, Sequential + Freezing, Separate, Separate+Freezing (see section 3.3.3).

The results are shown in Fig 12. For an e2e learning process, after training for 20000 episodes, it reaches a success rate of 60%. Sequential learning of behaviours and using the output of the previous behaviour to learn new behaviour in addition to freezing layers after initial training also seems to take the same amount of data (20000 episodes) to reach 60% success accuracy.

Astonishingly, In the scenario where we learn each skill separately with freezing layers after initial training, the algorithm is able to reach 100% success rate in 6000 episodes of training. Further, separate skill training without freezing shows a minor enhancement in success rate. These results seem to verify the hypothesis we propose that learning can be more effective if each skill is learnt in isolation and then combined after learning to accomplish the task. Secondly, as in humans, once the network knows how to extract features, freezing the knowledge (in this case, the weights) shows more learning potential for other subtasks.

In sequential learning, where each module is trained incrementally, the results show similarities with the learning curve of the e2e process. One reason for this counterintuitive result may be the random position of the target point at each run. Although, the algorithm has learnt how to approach a randomly positioned block and grasp it. It is difficult to move in the 3d cartesian space to a random position in each run.

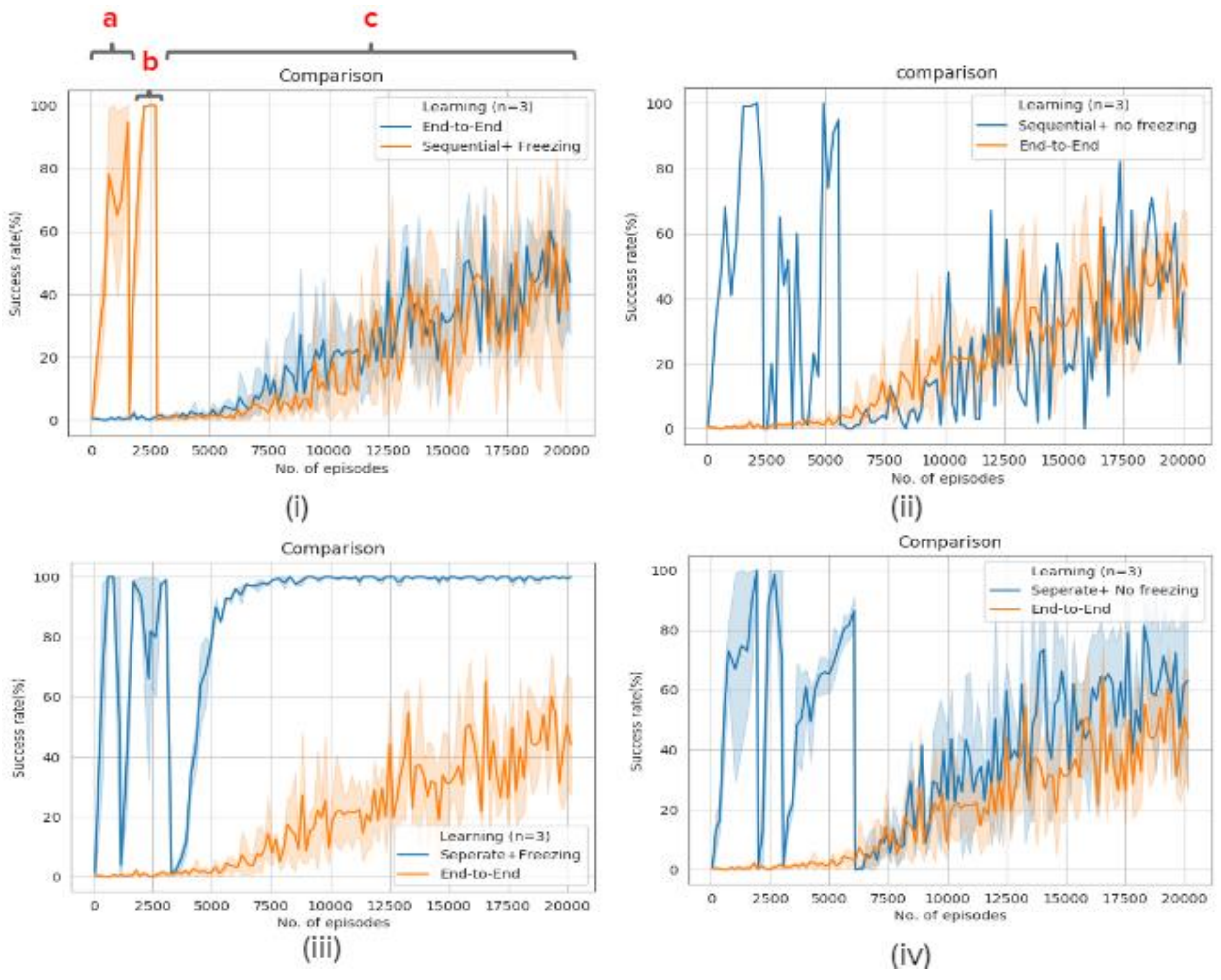


Fig 12: Comparison of e2e learning with different proposed implementations:
i) Sequential + Freezing architecture (Yellow) and e2e learning process (Blue)
ii) Sequential (Blue) and e2e process (yellow)
iii) Separate + Freezing (Blue) and e2e process (yellow)
iv) Separate (blue) and e2e (yellow).

In case of hierarchical implementation, the first spike, when the success rate reaches 100 % denotes the completion of training process a (approach), the second spike denotes the completion of training for process b (manipulate). After the second spike, the success rate drops down and started training for process c (retract). The results suggest that it takes the same amount of training episodes for e2e process and Sequential, Sequential+Freezing. Whereas separate+ freezing shows drastic increase in success rate. Separate also shows increase in efficiency but further experiments need to be carried out for longer no. of episodes. Each algorithm is run independently for three runs with different seeds.

4.1.1 Discussion

Today’s deep learning systems are composed of modules, layers and group of layers with specific roles to extract feature representations. Once the network is designed, the weights are initialised randomly and they are trained subsequently e2e, with variants of stochastic gradient descent. The power of e2e learning has been demonstrated on multiple tasks, like playing a whole array of Atari video games with a single deep network architecture. As we stride towards creating non-trivial agents resembling human brain, we end up with customized modules for sensory data, language processing, motor control, decision making, memory and more. This problem decomposition is central to solving complex problems. Dividing the problem into subtasks is heavily used while designing a complex learning machine. However, training such a system in an e2e way means to explicitly ignore the problem of decomposition. Instead, we just hope that preconditioning while designing is enough to drive methods like gradient descent from a random initialisation seed to a non-trivial solution.

Data efficiency is the second issue. Unmodeled interaction between modules may require large amounts of training data. As the number of modules increases, the training data required also increase exponentially, also called combinatorial explosion.

In this work, we attempt to show that the above-mentioned problems could be solved using

1. Dividing a complex task into multiple sub-tasks that are trivial to learn. Each of these subtasks denote some trivial behaviour. These behaviours are modelled using simple fully connected networks that are trained specifically for that behaviour e2e.
2. Learning each of these trivial behaviours separately reduces the data-required for learning.

This setup is different from standard RL, since the agent performs only a single high-level action with a *reactive* policy. The selected action consists of sub-actions that need to be pre-programmed/pre-trained.

Our results suggest that training could be more effective if each module is trained separately with other modules already trained or using hand-engineered solution. Also, freezing the feature extraction layer after initial training and using these freezed weights for training other module has an enormous reduction in training time. This indicates that training of complex learning systems should be accomplished in a structured fashion, training simple modules first and independent of the rest of the network. It aligns with the paradigm of bottom-up learning approach where complex behaviours could arise by generating and combining simple ones (Duarte *et al.*, 2016; Gomes, Oliveira and Christensen, 2018).

4.1.2 Applicability

Our study highlights the versatility of hierarchical decomposition of behaviours and offers significant advantage compared to other works.

1. Incrementally increasing the reactive behaviours: We start from one single behaviour and increase the number of these reactive behaviours as we increase the complexity of task.
2. Generality- For different tasks in the same environment, the reactive behaviours can be sequenced differently manually to complete the task. A set of reactive behaviours can give rise to multiple task. Whereas, in existing e2e methods, for each new task, the agent needs to be trained again from scratch.
3. Behaviour reuse- Such reactive behaviours can be reused multiple times to complete long-term tasks.
4. Control systems using Behaviour decomposition are more understandable and transparent compared to a black-box-optimiser. Such intelligibility is useful when the developer needs to modify certain aspects of behaviour.

Our approach allows for these advantages; however, it also mitigates the following disadvantages:

1. The designer needs to pre-program the subtasks and hence plan upfront. One needs to have the information of the environment dynamics. Whereas, e2e

methods is trained holistically towards an objective function, although the learning is slow.

2. Extremely difficult and tedious to pre-program behaviours involving high amount of degree of freedom. For example: While manipulating a block, it is challenging to decompose the task into simple behaviors and understanding the Degrees of Freedom associated with that behaviour. Learning problems become much harder if we do not know how to decompose them, or if we simply get it wrong.

Chapter 5

Future work and Conclusions

Our future work concerns with training a recurrent network to identify and sequence the connection between the reactive behaviour primitive to complete a variety of task. This extension is directed towards adding a sequential reasoning layer that could mitigate the need for end-to-end training. This would increase automation and reduce human involvement since the algorithm would decide to reuse, switch between the primitive behaviours to complete different types of tasks. Also, in our current implementation, we have used Behaviour cloning to train the reactive behaviour using demonstration. However, further work would attempt to use advanced RL techniques like PPO in addition to behaviour cloning to speed up the learning and perform better than the demonstrations. Additional directions would be to add more levels of Hierarchy to show more general behaviour. Note that in all these experimentations, we plan to use a vision-based simulator in contrast to our currently used simulator which gives kinematic state inputs. This will enable us to develop complex convolution network for feature extraction from images. Finally, we plan to transfer the knowledge from simulation to a real robotic system.

On a separate line, instead of using a back propagation based gradient algorithms, recent researches have shown that Evolutionary strategies and genetic algorithms perform astonishingly on benchmark RL platforms despite its simplicity (Salimans *et al.*, 2017; Such *et al.*, 2017). These algorithms reflect the biological process of natural selection, where the fittest individuals are selected for reproduction in order to produce offspring for the next generation. There have been limited experimental studies in GA's, specifically neuroevolution, but it offers an unconventional, yet theoretically well-studied, approach to select hyper-parameters, best performing models and faster training cycles. ES and GAs were believed to not generalise to high dimensional state spaces until a team from

OpenAI[2] showed that these algorithms exhibit competitive performance on Atari games. In the context of our work, we plan to incorporate GA's to evolve general behavioural repertoire. Further to this, a higher-level arbitrator would be evolved to sequence the primitive actions.

In case of gradient based approaches, usually the optimisation gets stuck in a local minimum in the parameter space. Also, the ε-greedy exploration strategy becomes biased towards one particular trajectory. GA's in combination with Novelty search algorithms have shown competitive results in terms of robustness and structured exploration. Novelty search selects individual networks that are different from other network and show a different behavior. Although overlooked by most research studies, GA's provide an interesting direction towards future research.

5.1 Conclusion

We propose an approach to generate repertoire of general behaviours using reactive network architecture. Our contribution is as follows: We use simple networks to train specific behaviours incrementally. This type of layering has been inspired from Subsumption theory, famous in 1980's. With the recent advent in Deep learning and Reinforcement learning, most implementations have used an end-to-end approach which is neither scalable to high-dimensional input nor efficient. Existing RL algorithms has shown success in trivial tasks of playing Atari games that provide continuous feedback, however, these algorithms fail to perform in sparse-feedback and long time-step environments like Robotic grasping. In this work, we have demonstrated, in simulations, that such long-time tasks can be decomposed and could be learnt independently. These independently trained networks combined manually, can give rise to behaviours that could accomplish variety of tasks. The proposed approach opens doors for further exploration in modular behavioural control.

[2] OpenAI - A non-profit AI research company primarily working on RL

References

Bojarski, M. *et al.* (2016) ‘End to End Learning for Self-Driving Cars’. Available at: https://arxiv.org/abs/1604.07316 (Accessed: 18 March 2019).

Botvinick, M. M., Niv, Y. and Barto, A. C. (2009) ‘Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective’, *Cognition*. Elsevier, 113(3), pp. 262–280. doi: 10.1016/J.COGNITION.2008.08.011.

Breyer, M. *et al.* (2018) ‘Comparing Task Simplifications to Learn Closed-Loop Object Picking Using Deep Reinforcement Learning’. Available at: http://arxiv.org/abs/1803.04996 (Accessed: 18 March 2019).

Brooks, R. A. (1990) ‘Elephants don’t play chess’, *Robotics and Autonomous Systems*. North-Holland, 6(1–2), pp. 3–15. doi: 10.1016/S0921-8890(05)80025-9.

Brooks, R. A. (1990) *Elephants Don’t Play Chess*, *Robotics and Autonomous Systems*. Available at: http://people.csail.mit.edu/brooks/papers/elephants.pdf (Accessed: 15 March 2019).

Brooks, R. A. (1991) ‘Intelligence without representation’, *Artificial Intelligence*. Elsevier, 47(1–3), pp. 139–159. doi: 10.1016/0004-3702(91)90053-M.

Buiatti, M., Di Giorgio, E., Piazza, M., Polloni, C., Menna, G., Taddei, F., Baldo, E. and Vallortigara, G., 2019. Cortical route for facelike pattern processing in human newborns. *Proceedings of the National Academy of Sciences*, *116*(10), pp.4625-4630.

Collins, J., Howard, D. and Leitner, J. (2018) ‘Quantifying the Reality Gap in Robotic Manipulation Tasks’. Available at: https://arxiv.org/abs/1811.01484 (Accessed: 18 March 2019).

Duan, Y. *et al.* (2017) ‘One-Shot Imitation Learning’. Available at: http://arxiv.org/abs/1703.07326 (Accessed: 18 March 2019).

Duarte, M. *et al.* (2016) ‘EvoRBC’, in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference - GECCO ’16*. New York, New York, USA: ACM Press, pp. 93–100. doi: 10.1145/2908812.2908855.

Ecoffet, A. *et al.* (2019) ‘Go-Explore: a New Approach for Hard-Exploration Problems’. Available at: http://arxiv.org/abs/1901.10995 (Accessed: 4 February 2019).

Florence, P. R., Manuelli, L. and Tedrake, R. (2018) ‘Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation’. Available at: http://arxiv.org/abs/1806.08756 (Accessed: 18 March 2019).

Forestier, S., Mollard, Y. and Oudeyer, P.-Y. (2017) ‘Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning’. Available at: http://arxiv.org/abs/1708.02190 (Accessed: 18 March 2019).

Frank, M. J. and Claus, E. D. (2006) ‘Anatomy of a decision: Striato-orbitofrontal interactions in reinforcement learning, decision making, and reversal.’, *Psychological Review*, 113(2), pp. 300–326. doi: 10.1037/0033-295X.113.2.300.

Frans, K. *et al.* (2017) ‘Meta Learning Shared Hierarchies’. Available at: https://arxiv.org/abs/1710.09767 (Accessed: 18 March 2019).

Glasmachers, T. (2017) ‘Limits of End-to-End Learning’. Available at: http://arxiv.org/abs/1704.08305 (Accessed: 12 March 2019).

Gomes, J., Oliveira, S. M. and Christensen, A. L. (2018) ‘An approach to evolve and exploit repertoires of general robot behaviours’, *Swarm and Evolutionary Computation*. Elsevier, 43, pp. 265–283. doi: 10.1016/J.SWEVO.2018.06.009.

Gupta, A. *et al.* (2016) ‘Learning Dexterous Manipulation for a Soft Robotic Hand from Human Demonstration’. Available at: http://arxiv.org/abs/1603.06348 (Accessed: 18 March 2019).

Hessel, M. *et al.* (2017) ‘Rainbow: Combining Improvements in Deep Reinforcement Learning’. Available at: http://arxiv.org/abs/1710.02298 (Accessed: 18 March 2019).

Ke, N. R. *et al.* (2018) ‘Sparse Attentive Backtracking: Temporal CreditAssignment Through Reminding’. Available at: https://arxiv.org/abs/1809.03702 (Accessed: 18 March 2019).

Kober, J. and Peters, J. (2014) ‘Reinforcement Learning in Robotics: A Survey’, in. Springer, Cham, pp. 9–67. doi: 10.1007/978-3-319-03194-1_2.

Laversanne-Finot, A., Péré, A. and Oudeyer, P.-Y. (2018) ‘Curiosity Driven Exploration of Learned Disentangled Goal Spaces’. Available at: http://arxiv.org/abs/1807.01521 (Accessed: 18 March 2019).

LeCun, Y., Bengio, Y. and Hinton, G. (2015) ‘Deep learning’, *Nature*. Nature Publishing Group, 521(7553), pp. 436–444. doi: 10.1038/nature14539.

Levy, A. *et al.* (2017) ‘Learning Multi-Level Hierarchies with Hindsight’. Available at: https://arxiv.org/abs/1712.00948v4 (Accessed: 18 March 2019).

Lillicrap, T. P. *et al.* (2015) ‘Continuous control with deep reinforcement learning’. Available at: http://arxiv.org/abs/1509.02971 (Accessed: 18 March 2019).

Liu, Y. *et al.* (2017) ‘Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation’. Available at: http://arxiv.org/abs/1707.03374 (Accessed: 18 March 2019).

Maggi, S., Peyrache, A. and Humphries, M. D. (2018) ‘An ensemble code in medial prefrontal cortex links prior events to outcomes during learning’, *Nature Communications*. Nature Publishing Group, 9(1), p. 2204. doi: 10.1038/s41467-018-04638-2.

Mnih, V. *et al.* (2015) ‘Human-level control through deep reinforcement learning’, *Nature*. Nature Publishing Group, 518(7540), pp. 529–533. doi: 10.1038/nature14236.

Mnih, V. *et al.* (2016) ‘Asynchronous Methods for Deep Reinforcement Learning’. Available at: http://arxiv.org/abs/1602.01783 (Accessed: 18 March 2019).

Moravec, H. P. (1988) *Mind children : the future of robot and human intelligence*. Harvard University Press. Available at: http://www.hup.harvard.edu/catalog.php?isbn=9780674576186 (Accessed: 17 March 2019).

Nachum, O. *et al.* (2018) ‘Data-Efficient Hierarchical Reinforcement Learning’. Available at: https://arxiv.org/abs/1805.08296 (Accessed: 18 March 2019).

Nakanishi, J. *et al.* (2004) ‘Learning from demonstration and adaptation of biped locomotion’, *Robotics and Autonomous Systems*. North-Holland, 47(2–3), pp. 79–91. doi: 10.1016/J.ROBOT.2004.03.003.

Pathak, D. *et al.* (2017) ‘Curiosity-driven Exploration by Self-supervised Prediction’. Available at: http://arxiv.org/abs/1705.05363 (Accessed: 18 March 2019).

Pathak, D. *et al.* (2018) ‘Zero-Shot Visual Imitation’. Available at: http://arxiv.org/abs/1804.08606 (Accessed: 18 March 2019).

Peng, X. Bin *et al.* (2017) ‘Sim-to-Real Transfer of Robotic Control with Dynamics Randomization’. doi: 10.1109/ICRA.2018.8460528.

van Polanen, V. and Davare, M. (2015) ‘Interactions between dorsal and ventral streams for controlling skilled grasp’, *Neuropsychologia*. Pergamon, 79, pp. 186–191. doi: 10.1016/J.NEUROPSYCHOLOGIA.2015.07.010.

Pyeatt, L. D. and Howe, A. E. (1999) ‘Decision Tree Function Approximation in Reinforcement Learning’. Available at: https://www.semanticscholar.org/paper/Decision-Tree-Function-Approximation-in-Learning-Pyeatt-Howe/1655cde00456867e6f12de9952fe3a78170fe7bb (Accessed: 18 March 2019).

Rajeswaran, A. *et al.* (2017) ‘Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations’. Available at: http://arxiv.org/abs/1709.10087 (Accessed: 18 March 2019).

Salimans, T. *et al.* (2017) ‘Evolution Strategies as a Scalable Alternative to Reinforcement Learning’. Available at: http://arxiv.org/abs/1703.03864 (Accessed: 18 March 2019).

Schulman, J., Moritz, P., *et al.* (2015) ‘High-Dimensional Continuous Control Using Generalized Advantage Estimation’. Available at: http://arxiv.org/abs/1506.02438 (Accessed: 18 March 2019).

Schulman, J., Levine, S., *et al.* (2015) ‘Trust Region Policy Optimization’. Available at:

http://arxiv.org/abs/1502.05477 (Accessed: 18 March 2019).

Schulman, J. *et al.* (2017) 'Proximal Policy Optimization Algorithms'. Available at: http://arxiv.org/abs/1707.06347 (Accessed: 18 March 2019).

Sciberras-Lim, E. T. and Lambert, A. J. (2017) 'Attentional Orienting and Dorsal Visual Stream Decline: Review of Behavioral and EEG Studies.', *Frontiers in aging neuroscience*. Frontiers Media SA, 9, p. 246. doi: 10.3389/fnagi.2017.00246.

Sehnke, F. *et al.* (2010) 'Parameter-exploring policy gradients', *Neural Networks*. Pergamon, 23(4), pp. 551–559. doi: 10.1016/J.NEUNET.2009.12.004.

Such, F. P. *et al.* (2017) 'Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning'. Available at: https://arxiv.org/abs/1712.06567 (Accessed: 18 March 2019).

Sutton, R.S. and Barto, A.G., 1998. *Introduction to reinforcement learning* (Vol. 135). Cambridge: MIT press.

Wang, J. X. *et al.* (2018) 'Prefrontal cortex as a meta-reinforcement learning system'. doi: 10.1038/s41593-018-0147-8.

Watkins, C. J. C. H. and Dayan, P. (1992) *Technical Note Q,-Learning*. Available at: http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf (Accessed: 18 March 2019).

Wu, Y. *et al.* (2017) 'Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation'. Available at: http://arxiv.org/abs/1708.05144 (Accessed: 18 March 2019).

Links for Github repositories and online sources

1. *Super Mario Bros. Instruction Booklet* (PDF). USA: Nintendo of America. 1985. Archived (PDF) from the original on June 23, 2017. Retrieved July 4, 2017
2. https://github.com/Kautenja/gym-super-mario-bros
3. https://github.com/Kautenja/nes-py/wiki/Wrappers
4. https://github.com/Ameyapores/Mario
5. https://gym.openai.com/envs/#robotics
6. https://github.com/Ameyapores/Hierarchical-RL-for-robotic-grasp