Deep reinforcement learning for robotic manipulation-the state of the art

Smruti Amarjyoti

Robotics Institute, School of Computer Science,

Carnegie Mellon University

samarjyo@andrew.cmu.edu

Abstract

The focus of this work is to enumerate the various approaches and algorithms that center around application of reinforcement learning in robotic manipulation tasks. Earlier methods utilized specialized policy representations and human demonstrations to constrict the policy. Such methods worked well with continuous state and policy space of robots but failed to come up with generalized policies. Subsequently, high dimensional non-linear function approximators like neural networks have been used to learn policies from scratch. Several novel and recent approaches have also embedded control policy with efficient perceptual representation using deep learning. This has led to the emergence of a new branch of dynamic robot control system called deep reinforcement learning(DRL). This work embodies a survey of the most recent algorithms, architectures and their implementations in simulations and real world robotic platforms. The gamut of DRL architectures are partitioned into two different branches namely, discrete action space algorithms(DAS) and continuous action space algorithms(CAS). Further, the CAS algorithms are divided into stochastic continuous action space(SCAS) and deterministic continuous action space(DCAS) algorithms. Along with elucidating an organisation of the DRL algorithms this work also manifests some of the state of the art applications of these approaches in robotic manipulation tasks.

1 Introduction

Reinforcement learning (RL) provides a neuropsychological and cognitive science perspective to animal behavior and sequential decision making. Recent studies

in cognitive science have also demonstrated analogies between the dopaminergic neurons in brains and temporal difference (TD) reinforcement learning algorithms. Other than the nature derived inspiration, several successful implementations of reinforcement learning (RL) in controlling dynamic robotic systems for manipulation, locomotion and autonomous driving [19], [1], [15] have proven the previously theoretical concept to be applicable in real time control of physical systems. Many of these methods use specialized policy structures to represent policies in order to put a cap on the number of iterations that are needed for optimizing the behaviour. Though efficient there is a loss of generality in adopting such an approach as it constricts the policy space to some specific trajectories [2]. Thus, non-linear function approximators like neural networks are used to parametrize the policy. This removes the requirement of using hand engineered policy representations and human supplied demonstrations to initialized them. Moreover, the use of higher number of parameters also theoretically ensures learning of complex behaviours that wouldn't have been possible with linear man made policies.

Another important development in the field of RL has been indirectly borrowed from enormous successes of deep convolutional neural networks(CNN) [3] in image feature extraction. A direct implication of CNNs in reinforcement learning was the use of image pixels as states instead of joint parameters, which was widely in practice in RL landscape. Use of such an expressive parametrization also enabled learning of value function and policies that were previously deemed complicated. The paper by Riedmiller [5] demonstrated that neural networks can effectively be used as q-function approximators using neural fitted q-iteration algorithm. Later introduction of convolutional networks by Mnih et al. [4] turned neural networks based q learning as a base for DRL. Some of the ideas that were introduced like mini batch training and concept of target networks were pivotal to the success of non-linear RL methods. But, the initial algorithms were used to play classic Atari 2600 games with pixels as inputs and discrete actions as policy. The result were extraordinary with the artificial agent getting scores that were higher than human level performance and other model based learning methods. Attempts have been made to use deep q-learning (DQN) for high dimensional robotics tasks but with a very little success [6]. This is essentially because of the fact that most of the physical control tasks have high dimensional action spaces with continuous real valued action values. This posed a problem for introducing DQNs in manipulation tasks as they act as value function approximators and an additional iterative optimization process is necessary to use them for continuous spaces. The algorithms falling under this class are categorized into a group called discrete action space algorithms(DAS) as they are efficient only in discrete action domains.

Another approach to parametrization of RL policies is to encode the policy directly

and search for optimal solutions in the policy space. These methods known as policy search methods are popular as it gives an end-to-end connection between the states and feasible actions in the agent environment. The parameters can then be perturbed in order to optimize the performance output [7]. The advantage of this process over the earlier value approximation methods is that the policies are integrated over both action and state space, thus the search is more comprehensive than Q-learning. And it also solves the discrete action problem as the output policy, $\pi(a|s)$ is a stochastic distribution over the action given a particular state. Thus, the policy representation provides probabilities over over action in a continuous space [8]. This class of continuous action algorithms are grouped into continuous action space algorithms(CAS). They include policy gradient and policy iteration algorithms that encode the policy directly and search over entire policy space. Initially developed and experimented on low dimensional state spaces, CAS algorithms have been integrated into CNN architecture in algorithms like deep deterministic policy gradients (DDPG) [21].

The CAS RL algorithms can further be divided into two subcategories, stochastic continuous action space(SCAS) and deterministic continuous action space(DCAS) algorithms. The main difference between both of the methods is basically the sample complexity. Even though stochastic policy gradient methods provide a better coverage of the policy search space, they require a large number of training samples in order to learn the policy effectively [8]. This is quite infeasible in robotic applications as exploration and policy evaluation comes at a price in such domains. Several methods like natural policy gradients and trust region policy gradients were developed in order to make policy search effective by adding additional constraints on the search process to restrict the agent to explore only promising regions. But, the discovery of deterministic policy gradients has led to an easier method whose performance surpasses stochastic policy algorithms as proven empirically by Silver et al [9].

The most important contribution of this paper is the organisation of the assortment of DRL algorithms on the basis of their treatment of action spaces and policy representations. Present DRL methodologies in literature are classified into the groups, DAS, CAS, SCAS and DCAS whose details has been described above. Following sections include a background of the evolution of reinforcement learning and preliminaries laying a foundation to understand the algorithms and description of some of the basic algorithms encompassing DRL. Experiments and real time implementations associated with these methods are also described to give an insight into the practical complexity of implementing these algorithms on physical robots/simulations.

2 Background

2.1 Preliminaries

All of the reinforcement learning methods studied in this paper are basically control problems in which an agent has to act in a stochastic environment by choosing action in a sequential manner over several time steps, with an intention to maximise the cumulative reward. The problem is modelled as a *Markov decision process* (MDP) which comprises of a state space S, an action space A, an initial state distribution with density $p_1(s_1)$, a stationary transition dynamics model with density $p(s_{t+1}|s_t, a_t)$ that satisfies the Markov property $p(s_{t+1}|s_1, a_1, ...s_t, a_t) = p(s_{t+1}|s_t, a_t$ for any trajectory in the state-action space and a reward function $R(s_t, a_t) : S \times A \to \mathbb{R}$. A policy can be defined as the mapping of state to action distributions and the objective of an MDP is to find the optimal policy. Generally a policy is stochastic and is denoted by $\pi_\theta : S \to P(A)$, where $P(A)$ is the probability distribution of performing that action and $\theta \in \mathbb{R}^n$ is a vector of parameters that define the policy, $\pi_\theta(a_t, s_t)$. A deterministic policy on the other hand is denoted by $\mu(s_t)$ and is a discrete mapping of $S \to A$.

A agent uses the policy to explore the environment and generate trajectories of states, rewards and actions, $\zeta_{1:t} = s_1, a_1, r_1, ..., s_t, a_t, r_t$. The total return or performance is determined by calculating the total discounted reward from time step t onwards.

$$R_t^\gamma = \sum_{k=t}^{\inf} \gamma^{k-t} R(s_k, a_k) \ , where \ 0 < \gamma < 1 \tag{1}$$

Value function of a particular state is defined as the expected total discounted reward if an agent were to initiate from that particular state and generate trajectories thereafter.

$$V^\pi(s) = \mathbb{E}[R^\gamma|S = s; \pi] \tag{2}$$

The action-value function on the other hand is defined as the expected discounted reward if the agent takes an action a from a state s and follows the policy distribution thereafter.

$$Q^\pi(s, a) = \mathbb{E}[R^\gamma|S = s; A = a; \pi] \tag{3}$$

The agent's overall goal is to obtain a policy that results in maximisation of cumulative discounted reward form the start state. This is denoted by finding appropriate π for the performance objective $J(\pi) = \mathbb{E}[R^\gamma|\pi]$.

The density of the state s' after transitioning for t time steps from initial state s is given by $p(s \to s', t, \pi)$. The discounted state distribution is then given by

$\rho^\pi(s^{'}) = \int_s \sum_{t=1}^{\inf} \gamma^{t-1} p_1(s) p(s \to s^{'}, t, \pi) ds$. The performance objective can be represented as a unified expectation function,

$$J(\pi_\theta) = \int_S \rho^\pi(s) \int_A \pi_\theta(s, a) R(s, a) da ds$$

$$= \mathbb{E}_{s \in \rho^\pi, a \in \pi\theta}[R(s, a)]$$

(4)

2.2 Evolution of RL

Early reinforcement learning(RL) algorithms for prediction and control were focused on the process of refinement of optimal policy evaluation techniques and reduction of computational complexity of the approaches. This led to the emergence of exploration vs exploitation techniques, on-policy and off-policy approaches, model free and model based and various PAC(Probable approximate correct) methods. Although the algorithms were feasible computationally and showed convergence to optimal policies in polynomial time, they posed a major hindrance when applied to generate policies for high dimensional control scenarios like robotic manipulation. Two techniques stand out from the newly developed RL methodologies, namely function approximation and policy search. The philosophy of these two approaches is to parameterize the action-value function and the policy function. Further, gradient of the policy value is taken to search for the optimal policy that results in a global maxima of expected rewards. Moreover, due to the hyper dimensional state-space and continuous action-space the robot operates in, policy search methods are the most viable and possible the only method considered suitable for robotics control.

2.3 RL for motor control

Application of RL in robotics have included locomotion, manipulation and autonomous vehicle control [19]. Most of the real world tasks are considered episodic and it is also hard to specify a concise reward function for a robotic task. This problem is tackled by the use of a technique called learning by demonstration or apprenticeship learning [22]. One of the methods to solve the uncertain reward problem is inverse reinforcement learning where the reward function is updated continuously and an appropriate policy is generated in the end Another effective method to model the policies is the use of motor policies to represent stochastic policy $\pi(a_t|s_t, t)$, that is inspired from the works of Kober and Peters [15]. They devised an Expectation Maximization (EM) based algorithm called Policy Learning by Weighing Exploration with Returns(PoWER). When learning motor primitives, they turn this deterministic mean policy into a stochastic policy using additive exploration in order to make model-free reinforcement learning possible. Here, the motor primitives

are derived from the concept of Dynamic Motor Primitives (DMPs) that describe movement as a set of differential equations such that any external perturbation is accommodated without losing the motion pattern. Certain other approaches like guided policy search [10] also introduced more versatile policy representations like differential dynamic programming (DDP). These policy have been used for generating guiding samples to speed up the learning process in non linear policies. This gives more flexibility and generalization than earlier structured policy approaches. But, even though such hybrid model based and specialized policy methods work well in robots, there has always been an interest towards learning policies end-to-end from visual stimulus. Thus, convolutional architectures have been introduced into the domain of RL and motor control, known as visuo-motor control policy networks.

2.4   RL for visuo-motor control

Many of the RL methods demonstrated on physical robotic systems have used relatively low-dimensional policy representations, typically with under one hundred parameters, due to the difficulty of efficiently optimizing high-dimensional policy parameter vectors. But the paper by Mnih et al. introduced an effective approach to combine larger policy parameterizations by combining deep learning and reinforcement learning [4]. This concept of generating efficient non-linear representations is transferred into robotic tasks of grasping and continuous servoing in some recent research carried out by Levine et al. [14] and Kober et al [?]. End-to-end learning of visuo-motor policies is made possible with such an approach which in turn learns the features form the observations that are relevant for the specific task. One of the problems that was encountered with neural network learning of policies was the convergence of some weights to infinity when trained with similar instances of input observations [20]. Solving of this difficulty using experience replay methods constituting randomization of the episodes gave the necessary boost to RL in real life control problems. The current state of the art in deep-reinforcement learning includes the algorithms employed by google deepmind research namely DQN (Deep Q network) for discrete actions and Deep deterministic policy gradients (DDPG) for continuous action spaces [16]. DQN is a simple value approximation method while DDPG uses a underlying actor-critic framework for policy evaluation. Efficacy of both of these methods have been demonstrated empirically for performing complex robotic manipulation tasks like door opening and ball catching [21].
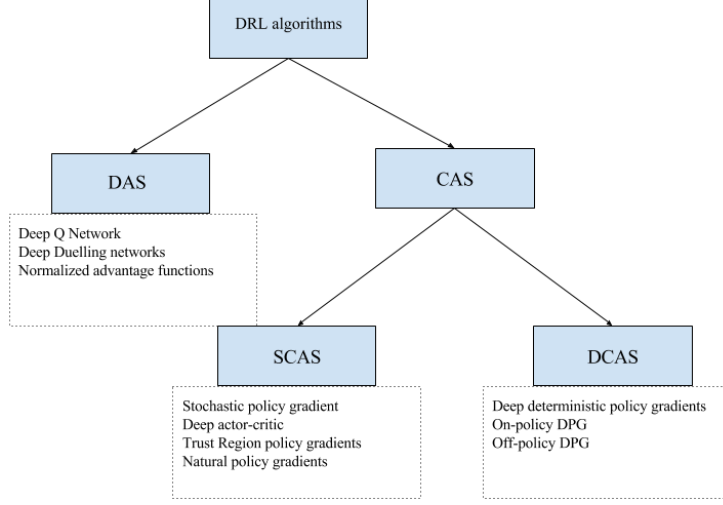
Figure 1: The DRL topology.

3 DRL topology

The Deep reinforcement learning algorithms prevailing currently are structured according to the topology in Fig. 1. The initial problem of planning in continuous and high dimensional state spaces can be considered solved due to the extensive use of neural networks with large number of parameters for function/policy modelling. But, the problem at hand now is the mapping of these continuous states to high-dimensional continuous action spaces. The present concentration in the DRL community based on this issue and hence, it seems quite apt to organise the various learning approaches based on this ground. Moreover it also demonstrates the capabilities and limitations of the prevalent algorithms quite clearly.

The methods are divided into two sections namely, Discrete action space(DAS) approaches and Continuous action space(CAS) approaches. Further, CAS methods are divided into Stochastic continuous action space(SCAS) and Deterministic continuous action space(DCAS) methods. The various algorithms that come under the purview of DAS are deep q-networks, duelling networks, normalized advantage function and related function approximation approaches to decision making. CAS mostly include policy search approaches that parametrize the policy directly and optimized it using evaluation and gradient based approaches. CAS is further branched into SCAS methods where CNN are used to estimate a stochastic policy and DCAS methods which predicts deterministic policies. Even though this demarcation provides a near comprehensive description of the DRL methods, it misses out on several other RL approaches like likelihood ratio maximisation, black box methods, model-based methods which are not directly related to DRL.

7

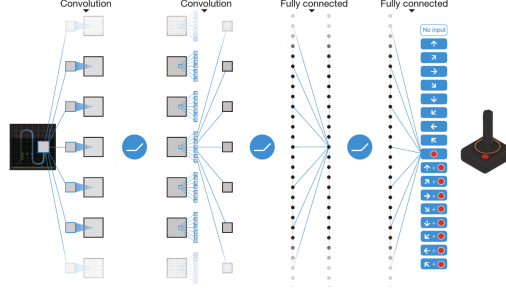# 4 Discrete action space algorithms (DAS)

4.1 Deep Q-network (DQN)

Figure 2: Deep-Q-network architecture.

The DQN architecture was the first successful integration of deep learning with Q-learning framework [4]. Q-learning forms the base of most of the model-free RL algorithms. It includes exploration of the environment using a behaviour policy and learn the Q-function for the possible action-state pairs using the experience that is gathered from the exploration. The following equation described Q-learning, where α is the learning rate and the observations that are obtained after exploration be a, r, s', where a is the action taken, r is the rewards received and s' is the next observed state.

$$Q(s,a)_{sample} = r(s,a) + \gamma.max_a Q(s',a)$$
$$Q(s,a) = \alpha Q(s,a)_{sample} + (1-\alpha)Q(s,a)$$

$$(5)$$

The only difference between naive Q-learning and DQN is the use of CNN as function approximators instead of linear approximators. The use of hierarchical networks enables the use of continuous high dimensional images as states which estimates the optimal action-value function. RL was considered to be unstable when using non-linear approximators such as a neural network, which is because of the correlations present in the sequence of observations and the correlations between the action-values and the target values. In order to solve this, Mnih et al. devised a method of asynchronous training of the Q-network called experience replay. Here, the experience $e = \{s_t, a_t, r_t, s_{t+1}\}$ is stored in a pool and mini-batches of experiences are accessed during training uniformly. This is then used to optimize the loss function,

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma max_a Q(s',a|\theta^-) - Q(s,a|\theta)^2]$$

$$(6)$$

Fig.2 describes the architecture of the Q-network that consists of 3 convolutional layers, with filter sizes 32x8x8;stride 4, 64x4x4;stride 4 and 64x3x3;stride 2. The final two layers are fully connected layers with 512 neurons and outputs are discrete in

(a) 2D baxter arm simulator

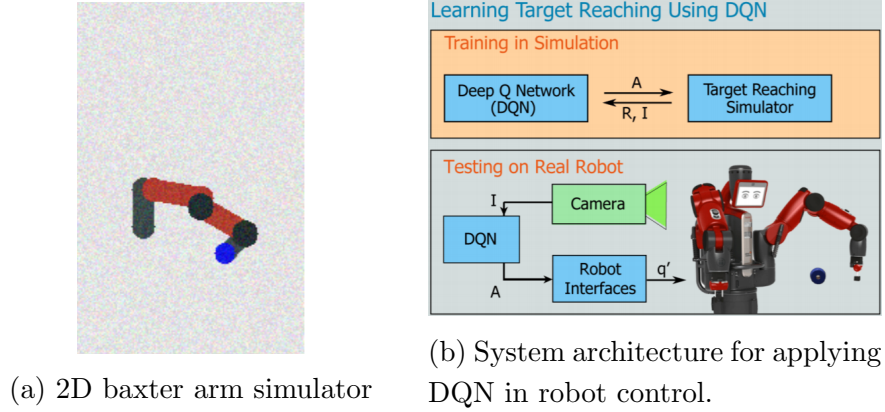(b) System architecture for applying DQN in robot control.

Figure 3: Visuo-motor learning using DQN

the number of actions considered. The activations chosen are rectified linear units. The second important contribution of DQN other than replay buffer was the use of target networks $\hat{Q}(s,a)$ for generating target values for the network's loss function. This helps to reduce oscillations during training and leads to easy convergence. The target network is updated with the online Q network after a specific number of time steps.

Execution of this methods is limited to agents requiring discrete action space but, some early works have embedded the DQN technique to learn optimal actions from visual inputs. Zhang et al. have utilized the exact same architecture to learn the optimal control policies for a baxter robot arm. Instead of controlling the entire 7DOF of the robot arm, only the 4DOF shown in the Fig.3(a) simulation are controlled. The actions are discretized into nine distinct outputs, that include going up, going down or staying put in denominations of 0.02 rad. After training, the network was used to control a real robotic arm with marginal success as it was prone to discrepancies in the input image. Moreover, training in simulation and transferring the control system to real-time robots proved to be detrimental for safety and performance.

4.2   Double Deep Q-networks

Double deep Q-networks are an improved version of DQN that was first introduced by Hasselet et al. [11]. In Q-learning and DQN, the max operator utilizes the same values for both behaviour policy and evaluation of actions. This in turn gave overestimating value estimates. In order to mitigate this, DDQN uses the target as:

$$y^{DDQN} = r + \gamma Q(s^{'}, a^{'}|\theta)|\theta^{-})$$
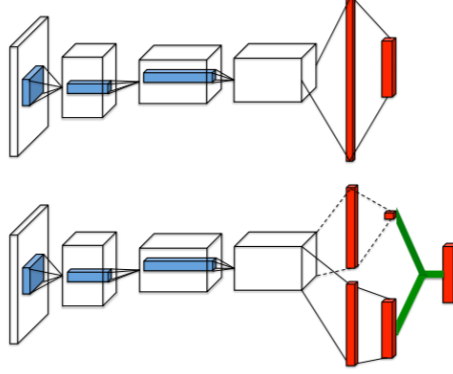
(7)

4.3   Deep duelling network architecture

Figure 4: The top network is a single stream DQN and the bottom network has dual streams to estimate value and advantage function

Duelling architecture, is a model-free algorithm developed by Wang et al. [12] draws its inspiration from residual RL and the concept of Advantage learning and updating by Baird [13]. In advantage learning instead of estimation of action-value function, an advantage function is calculated which is defined as the rate of increase of reinforcement when a particular action is taken. The prime importance of advance learning is that the advantage values have higher variance that leads to easy convergence. Moreover, the policy doesn't change discontinuously with changing values. The duelling architecture maintains both $V(s)$ and $A(s,a)$ with a single deep model and a simple output operation combines both these output to get back the $Q(s,a)$ value. As the output is same as DDQN and DQN, this network can be trained with any value iteration method.

Considering the duelling network described in Fig. 4 where one stream outputs $V(s|\theta,\beta)$ and other $A(s,a|\theta,\alpha)$. θ and α denote the convolutional network parameters. The last module is implemented using a forward mapping function:

$$Q(s,a|\theta,\alpha,\beta) = V(s|\theta,\beta) + (A(s,a|\theta,\alpha) - \frac{1}{\|A\|}\sum_a A(s,a|\theta,\alpha)) \qquad (8)$$

The architecture was used to train an artificial agent learn the 57 games in Atari arcade learning environment from raw pixel observations. The final acquired rewards were compared with that of human performance and DQN networks. Duelling networks performed 75% better than the naive Q-networks as reported in the paper [12]. For applications with discrete action robotic tasks duelling architecture can be used for better performance, though a concrete application is missing from literature.

5 Continuous action space algorithms (CAS)

5.1 Normalized advantage functions (NAF)

Gu et al. proposed a model free approach that used Q-learning to plan in continuous action spaces with deep neural networks, which they refer as Normalized advance functions (NAF). The idea behind NAF is to describe Q function in a way such that its maximum, $argmax_a Q(s_t, a_t)$ can be obtained easily and analytically during the Q-learning update. The inherent processes are equivalent to that of Duelling networks as a separate value function $V(s|\theta)$ and advantage term are estimated. But, the difference is that the advantage in this case is parametrized as a quadratic function of non-linear features of the state:

$$Q(s, a|\theta) = A(s, a|\theta) + V(s|\theta) \tag{9}$$

$$A(s, a|\theta) = -\frac{1}{2}(a - \mu(s|\theta)P(s|\theta)(a - \mu(s|\theta) \tag{10}$$

$P(s|\theta)$ is a state-dependent, positive definite square matrix that is parametrized by $L(s|\theta)L(s|\theta)^T$, where L is a lower triangular matrix whose entries come from the linear activations of the neural network. The rest of the network architecture is similar to that of DQN by Mnih et al. The paper also explored the use of a hybrid model based method by generating imagination rollouts from fitted dynamics model. This incorporated the inclusion of synthetic experience data from the fitted local linear feedback controllers and including them in the replay buffer of on-policy exploration of Q-learning.

The algorithms was tested in several robotic environments as shown in Fig.5. The

Figure 5: Robotic manipulation and locomotion environments in MuJoCo simulator.

environments include the MuJoCo simulator tasks from Todorov et al. that include 3DOF robotic manipulation tasks where an arm gets reward based on the distance between the end effector and the object to be grasped. It also includes a sex joint

2D swimmer and a four legged ant. Policies learnt with this method showed more precise completion of tasks as compared to deep policy gradient methods [17].

5.2 Stochastic policy gradient

Stochastic policy gradient methods parametrize the policy directly rather than trying to optimize the value functions. These are one of the most popular class of continuous action RL algorithms. The central idea behind these algorithms is to adjust the parameters θ of the policy in the direction of the gradient of the performance, i.e $\Delta_\theta J(\pi_\theta)$. The fundamental theorem underpinning these algorithms is the *policygradienttheorm* [18].

$$
\begin{aligned}
\Delta_\theta J(\pi_\theta) &= \int_S \rho^\pi(s) \int_A \Delta_\theta \pi_\theta(a|s) Q(s,a) da\,ds \\
&= \mathbb{E}[\Delta_\theta log \pi_\theta(a|s) Q(s,a)]
\end{aligned}
\tag{11}
$$

The interesting aspect of this theorem is that even though the state distribution depends on the network parameters, the policy gradient doesn't depend on the gradient of the distribution. But, one of the issues that these algorithms have to address is the estimation of the $Q(s,a)$ function as evident from the above equation. Even though policy gradient algorithms provide an end-to-end method for policy search, it is rarely used in robot policy optimization tasks. This is because of the high sample complexity of such algorithms. Policy gradients use on-policy exploration policy and that results in it needing a large number of training data to learn, that is infeasible for robots. Above figure depicts a stochastic policy gradient algorithm
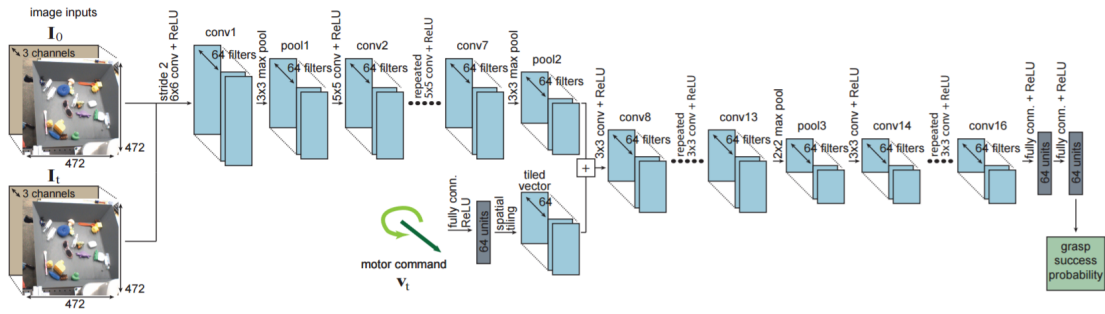

Figure 6: The architecture of CNN grasp predictor used in Levine et al. There are a total of 17 conv layers with inputs as the stationary image of object cluttered environment and the online image, the joint configuration is also passed on after the 7th layer.

that is used by Levine et al. [14] for autonomous grasping of objects in cluttered

environments. The input is a monocular image showing objects and robot end effector and the robot actions in the 7th layer of the deep network. The output is the probability distribution of the action given the particular state. The network takes 800,000 labelled images to train which gives a clear indication of the sample complexity of SCAS methods.

5.3 Stochastic actor-critic methods

Actor-critic methods are widely used architectures that are again based on the Policy gradient theorem. As depicted from the policy gradient equation, the term $Q(s, a)$ is missing from the gradient and needs to be calculated. Hence, the critic network estimates this $Q(s, a)$ value in order to find the derivatives of the actor network,$\Delta \pi_\theta(s)$.

5.4 Trust Region Policy Optimization

TRPO is a policy optimization algorithm that restricts the search space of the policy by applying constraints on the output policy distributions. This is done by penalizing the network parameters using a KL divergence loss function upon the parameters, $D_{KL}^{max}(\theta_{old}, \theta)$. Intuitively this constraint doesn't let large scale changes to occur in the policy distribution and hence, helps in early convergence of the network. The above figure depicts the networks that were used to control the swimmer
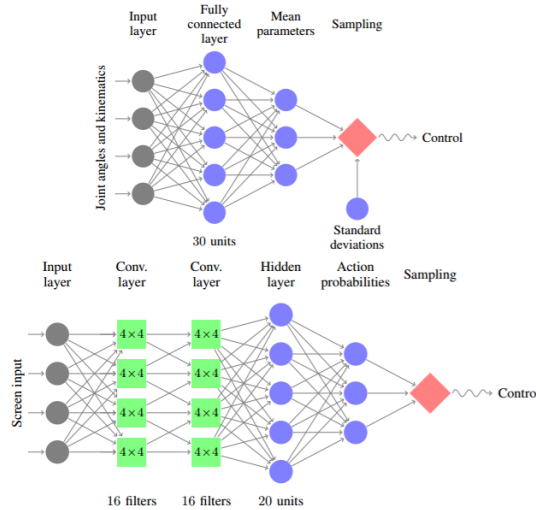
Figure 7: Networks used to represent policy in TRPO.

and hopper tasks in MuJoCo environments. The input state space consisted of joint angles and robot kinematics and the rewards were linear functions.

5.5   Deterministic policy gradient algorithm

The deterministic policy gradient algorithm (DPG) is derived from its counterpart stochastic policy gradient algorithm and is dependent of a similar deterministic policy gradient theorem . In continuous action spaces, greedy policy improvement becomes problematic and needs global optimization during policy improvement step [9]. As a result the it is more computationally tractable to update the policy parameters in the direction of the gradient of the Q function.

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}[\Delta_\theta Q(s, \mu_\theta(s)] \tag{12}$$

Here, $\mu_\theta(s)$ is the deterministic policy, α is the learning rate and θ are the policy parameters. Chain can be applied to the above equation in order to get the policy gradient equation:

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}[\Delta_\theta \mu_\theta(s) \Delta_a Q(s, \mu_\theta(s)] \tag{13}$$

The above update rule can be incorporated into a deep neural network architecture where the policy parameters are updated using stochastic gradient ascent. To realise this an actor-critic method is necessary. The critic estimates the action-value function while the actor derives its gradients from the critic to update its parameters. The gradient of policy parameters is the product of the gradient of Q value with respect to action and the action with respect to the policy parameters. Fig. 8 shows the deterministic actor critic network.
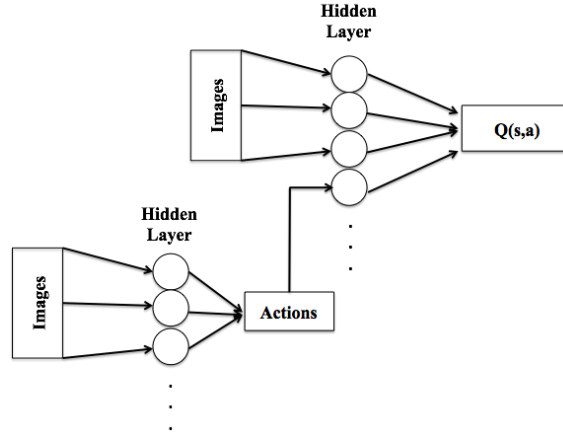

Figure 8: Deep deterministic policy gradient network.

This is also the basis of DDPG (deep deterministic policy gradient algorithm) which performs better than any other continuous action algorithm. Methods such as NAF and DDPG have been used for learning complex robotic manipulation tasks in real time [16]. The authors trained a 7DOF Jaco arm for reaching and door
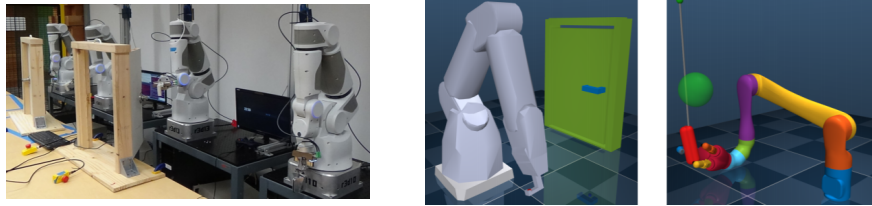
(a) Real-world door opening task setup.
(b) Simulation setup for door opening and object reaching task.

Figure 9: Task representation for DDPG and NAF.

opening tasks without any policy initializations and demonstrations. They used deep network architectures with a 20 dimensional state space consisting of the joint angles, velocities and the end effector pose. The reward function for the reaching task was the distance between the end effector and the object, whereas for door opening the reward was the sum of distance to the door knob and the degrees of rotation of the knob. Another significant contribution of this paper was the use of asynchronous leaning by parallelizing the data collection process. It was proved that using multiple robots reduces the training times by a factor of the number of robots.

6 Discussion

Algorithmic ideas, theories and implementation details of several deep reinforcement learning algorithms have been delineated in detail. It can be concluded that for the purpose of robotic manipulation continuous action domain algorithms are the most fruitful and applicable. Further, it can be observed that there is a trend towards exploration of sample efficient and time efficient algorithms, having solved both continuous state and action space problems. Breakthroughs in these domains will have significant impacts in the field of robotics learning.

Also, as demonstrated from current state of the art in DRL, the approaches fail to handle complex policies. A reason could be that complicated policies require more samples to learn and even a sophisticated reward function. This observation highlights a void in RL in robotics. There is a need to learn highly complicated reward functions and methods to represent highly skilled behaviours and skills. This are of Inverse reinforcement learning needs to be paid more attention while learning policies using DRL. After all, complexity of the reward function is proportional to the policy complexity.

Reinforcement learning is an evolved form of the cognitive architecture SORE. There is a need to reconnect the new DRL approaches to its roots in cognitive science. The

problems in DRL might find extremely useful insights from theories and empirical evidence from cognitive psychology.

One of the important drawbacks of DRL algorithms and visuo-motor architectures are the lack of capability of transfer learning. It is difficult to transfer skills and use the knowledge of already learnt policies to learn even complicated policies. A mechanism needs to be developed so that policies doesn't have to be learnt from scratch, but can be inherited.

Many problems with temporally spread out rewards lead to credit assignment problem in RL. Thus, the reward structure too needs to be redesigned. There have been several works in incorporating intrinsic motivation in reinforcement learning as a method to induce temporal abstractions in agents [23]. These setups known as semi-Markovian decision process can be used to learn hierarchical planning actions by learning step by step about the task at hand, just as a human does.

Another most important aspect of DRL that hasn't been touched upon in the main body is an approach known as guided policy search (GPS). This is because of it incipient stages in DRL currently, but the approach holds significant potential in learning robotic tasks with minimal trials. The central idea behind the algorithm is to mix model based and model free algorithms and use linear models to generate samples in order to guide the learning process. This seems like a valid assumption as humans/animals don't always learn actions from scratch, but take advantage of already well developed models of their body and physics.

References

[1] Deisenroth, M.P., Neumann, G. and Peters, J., 2013. A Survey on Policy Search for Robotics. Foundations and Trends in Robotics, 2(1-2), pp.1-142.

[2] Levine, S. and Koltun, V., 2013. Guided Policy Search. In ICML (3) (pp. 1-9).

[3] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), pp.529-533.

[5] Riedmiller, M., 2005, October. Neural fitted Q iterationfirst experiences with a data efficient neural reinforcement learning method. In European Conference on Machine Learning (pp. 317-328). Springer Berlin Heidelberg.

[6] Zhang, F., Leitner, J., Milford, M., Upcroft, B. and Corke, P., 2015. Towards vision-based deep reinforcement learning for robotic motion control. arXiv preprint arXiv:1511.03791.

[7] Sutton, R.S. and Barto, A.G., 1998. Reinforcement learning: An introduction (Vol. 1, No. 1). Cambridge: MIT press.

[8] Schulman, J., Levine, S., Moritz, P., Jordan, M.I. and Abbeel, P., 2015. Trust region policy optimization. CoRR, abs/1502.05477.

[9] Lever, G., 2014. Deterministic policy gradient algorithms.

[10] Levine, S. and Abbeel, P., 2014. Learning neural network policies with guided policy search under unknown dynamics. In Advances in Neural Information Processing Systems (pp. 1071-1079).

[11] Van Hasselt, H., Guez, A. and Silver, D., 2015. Deep reinforcement learning with double Q-learning. CoRR, abs/1509.06461.

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

[13] Baird III, L.C., 1999. Reinforcement learning through gradient descent (Doctoral dissertation, US Air Force Academy, US).

[14] Levine, S., Pastor, P., Krizhevsky, A. and Quillen, D., 2016. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. arXiv preprint arXiv:1603.02199.

[15] Kober, J. and Peters, J.R., 2009. Policy search for motor primitives in robotics. In Advances in neural information processing systems (pp. 849-856).

[16] Gu, S., Holly, E., Lillicrap, T. and Levine, S., 2016. Deep Reinforcement Learning for Robotic Manipulation. arXiv preprint arXiv:1610.00633.

[17] Gu, S., Lillicrap, T., Sutskever, I. and Levine, S., 2016. Continuous Deep Q-Learning with Model-based Acceleration. arXiv preprint arXiv:1603.00748.

[18] Sutton, R.S., McAllester, D.A., Singh, S.P. and Mansour, Y., 1999. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In NIPS (Vol. 99, pp. 1057-1063).

[19] Kober, J., Bagnell, J.A. and Peters, J., 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research, p.0278364913495721.

[20] Levine, S., Finn, C., Darrell, T. and Abbeel, P., 2016. End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39), pp.1-40.

[21] 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.

[22] Abbeel, P. and Ng, A.Y., 2004, July. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the twenty-first international conference on Machine learning (p. 1). ACM.

[23] Chentanez, N., Barto, A.G. and Singh, S.P., 2004. Intrinsically motivated reinforcement learning. In Advances in neural information processing systems (pp. 1281-1288).