Playing games with Deep Reinforcement Learning: A Survey Paper

# Abstract

This paper serves as an introduction to Deep Reinforcement learning and a walkthrough of its development using the domain of video games as a benchmark. This survey also introduces some leading-edge research and identifies goals for future direction.

# Introduction

Why should the readers care to read this. What is exciting about deep RL. What has it done. What can it do. (Look to other Deep RL survey paper for inspiration)

One of the main goals of artificial intelligence (AI) is to produce fully autonomous agents that can learn, from their own experience, to behave optimally in their environments. Reinforcement Learning (RL) provides a framework for such experience-driven learning. The advent of deep learning has enabled RL to tackle problems previously intractable, resulting in the field of “Deep Reinforcement Learning”. The Deep Q-Learning (DQN) algorithm was introduced by DeepMind to beat previous state-of-the-art agents at playing Atari 2600. From then on, growth of the field has been exponential.

Deep RL has achieved great success “a decade ahead of its time”[REF DEEPMIND HOMEPAGE] as AlphaGo [REF] defeated Lee Sedol the 18-time Go world champion with a 4-1 victory in Seoul, South Korea. It was thought impossible due to the vast game-tree complexity of Go 10^360 (for comparison chess is 10^123). The same authors then developed AlphaZero [REF] that beat each world-champion computer programs in the games of Chess, Shogi and Go.

Pluribus achieved superhuman performance at 6-player no-limit Texas Hold’em when competing against elite human players in a series of matches [REF]. Two of the players being past 1st place winners of the poker world series. This represents a massive milestone for AI as the algorithm deals with a partially observable six-agent environment. Many real-world strategic interactions happen in similar environments.

OpenAI used RL to train a robotic hand to solve a Rubik’s cube then transferring what was learnt in simulation to be done on physical actuators in the real world with amazing results [REF].

AlphaStar [REF] achieved grandmaster level in StarCraft II, reaching top of the league and beating 99.8% of officially ranked human players.

Video games present an interesting challenge and testbed for Deep RL but they are not the end goal. The vision behind Deep RL is of creating systems that learn and adapt in real world domains.

# Background

## Reinforcement Learning

Reinforcement learning is an iterative trail & error, reward & punishment process over time. At each time step the RL agent acts upon its environment by taking an action. The environment then provides feedback to the agent in the form of an immediate reward. The goal of reinforcement learning is to train an agent through experience to act upon its environment such that the environment yields the most reward to the agent.

RL differs to Supervised Learning because the targets for learning change over time as the agent learns from interactions with the environment. In supervised learning the targets for learning are fixed from the beginning.

### MDP Model

Reinforcement Learning is traditionally modelled as a Markov Decision Process (MDP) <REF>. A MDP is a sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards. It is defined as a set of states, actions, a transition model, a reward function and initial state distribution {S,A,P(s’|s,a),R(s),}.

* States being the set of environment states. The current state being the state of the environment the agent is currently experiencing.
* Actions being the set of actions the agent can take. At each state only a subset of actions may be available.
* Transition model is probability of successful transition for each state-action pair. In deterministic environments this would be 1 but the general formalism of an MDP includes stochastic environments.
* Reward function is the reward the environment yields to the agent for being in a specific state.
* Initial state distribution defines the set of states the agent can start in.

At any given time-step t, an agent perceives its state s-t and selects an action a-t. The environment responds by giving the agent some reward r-t and transitioning the agent into state s-t+1 (s’). When a terminal state is reached a run of the sequential problem is over. A sequence of states and actions between the initial state and the terminal state is called the trajectory, denotes by T.

<Diagram of MDP>

It is an assumption of MDPs that all transitions are **Markovian**, that is, the probability of reaching s’ from s is entirely dependent on s and not any previous states. This means that the value of the future is independent of the past and can only be decided by the actions taken in the present.

The goal of a RL agent is to act in a way that gets as much reward from the environment as possible. In an MDP the goal of an agent is to maximise the expected total future discounted rewards. If an agent can act in this way it will have solved the MDP. Solving an MDP will solve the reinforcement learning problem.

<Equation goal here>

We say discounted rewards because each intermediate reward is typically discounted by a factor <gamma> (0,1]. The discount factor describes the preference of an agent for current rewards over future rewards, it is the trade-off between short-term and long-term planning. I.E. when <gamma> is close to 0 future rewards are viewed as insignificant.

Not all Deep RL uses an MDP as a model although most use at least some part of it. This is the difference between model-free and model-based RL algorithms and something that will be detailed later. It is my experience being able to visualise the problem as an MDP is beneficial to reading the literature.

### Value & Policy based solutions

An MDP can be solved by learning an optimal policy pi-\*. A policy is a mapping of states to actions pi : S -> A that tells our agent what action to take at each step along its trajectory. An optimal policy pi-\* is a policy that yields the highest expected value from each state and consequently the highest reward from the MDP.

Another solution is to calculate the value of each state and use the state values to select an optimal action in each state. The value of each state is defined recursively by the **Bellman Equation**.

Traditional solutions to an MDP would perform value-iteration or policy-iteration. Value-iteration uses a dynamic programming approach in the form of Bellman Updates to converge on the actual values of each state. Policy-iteration repeatedly evaluates and improves a policy. The evaluation calculates the value of each state according to the policy. The improvement calculates a new policy by considering a one-step look-ahead that updates each of the state values previously evaluated by the addition of the next theoretical immediate reward. Iteration terminates when policy improvement yields no change in state values.

## Deep Learning

Deep Reinforcement learning is the application of Deep Learning to the Reinforcement Learning problem. Deep learning is a subfield of Machine Learning (ML) that is concerned with the use large artificial neural networks. These ‘deep’ networks often have many intermediate layers and thousands if not millions of parameters. Application of these networks allows RL to scale to problems that were previously intractable. Settings that have high-dimensional state and action spaces, domains where the input is visually rich or where input is required to be encoded over a long history.

Deep neural networks are used to approximate the solutions to an MDP therefore they approximate either the policy function or the value function. Policy networks typically output as a softmax over actions whereas value networks output a numerical value. There are three options for the value function a deep neural network can approximate.

* The state value function
* The state-action value function
* The advantage function

is the value of being in a state s assuming the agent continues to follow policy π. is the value of being in a state s and taking action a assuming the agent continues to follow policy π. describes how much better it is to take a specific action a in state s over other actions assuming the agent keeps acting according to policy π. Sometimes we do not need to describe how good an action is in an absolute sense, but only how much better it is than others on average.

### CNNs

A CNN is the most common type of neural network used in Deep RL. CNNs learn complex end-to-end functions that both represent and categorise input data. CNNs compose non-linear modules that transform raw data into more abstract representations that can then be used for classification REF. CNNs are deep-learning methods designed to deal with data in the form of multi-dimensional arrays such as images. CNNs are often called feature extractors in the sense that each convolutional layer will extract higher level (more abstract) features. The first layer perhaps identifying lines and edges, the next layer identifying shapes, and later layers maybe even recognising whole objects within an image. CNNs are composed of Convolutional layers, Pooling Layers and Fully-Connected Layers (also referred to as dense layers).

A convolutional layer contains a set of filters (also referred to as neurons or kernels). Each filter is a 3D array with a set width & height known as the receptive field and a depth equal to the depth of the input. Filters slide or convolve over the input performing local element-wise multiplication and summing the result. This weighted sum is then passed through a non-linearity, typically a ReLU activation function F(x) = max(0,x). After sliding the filters of a convolutional layer over all the locations, the layer produces an output called an activation map or feature map. Each activation map is produced by a different filter in the layer. Filters typically shift one unit at a time over the input. Strides can be used to reduce the dimensionality of the output, by shifting more than one unit at a time. The output activation maps are used then as input to proceeding layers.

Pooling Layers merge neighbouring features into one. They perform down sampling along the spatial dimensionality of the given input. Pooling layers decrease the dimensionality of the representation and therefore the number of network parameters in later layers. They also make the feature maps more robust to small shifts and distortions of the feature image. This is known as “local translation invariance”. There are multiple types of pooling layers.

Min Pooling – Takes the minimum of each local area defined by the receptive field

Max Pooling – Takes the maximum of each local area defined by the receptive field

Average Pooling – Takes the average of each local area defined by the receptive field

Fully-Connected layers are multi-layer perceptions and perform similar functions found in shallow neural networks. It is the job of the fully-connected layers to classify the output of the preceding convolutional and pooling layers.

### Recurrent networks? LSTM??

DRQN for poMDPs. Definition of poMDP in 17.4.1 of Into to AI book.

Recurrent neural networks are also used in deep RL and often trailed as alternative architectures. The Deep Recurrent Q-Network (DRQN) algorithm uses a recurrent architecture with DQN to play a partially observable version of Atari [REF].

Recurrent networks are neural networks with memory that specialise in processing sequential data. Output of hidden layers are stored in memory neurons. The memory neurons are passed back to the hidden layer as input for the next item of the input sequence that is feed to the recurrent network.

The two most common types of recurrent network are Gated Recurrent Unit (GRU) [REF] and Long Short-Term Memory (LSTM) [REF]. Recurrence can be used in conjunction with CNNs as a single hybrid architecture.

# Taxonomy

## Model-Based & Model-Free

Some agents have access to (or learn) a model of the environment. This allows agents to plan, by thinking time steps ahead, what the result of the environment would be for future actions they could take. Algorithms using an environment model are called model-based, those that do not are called model-free.

### Model-Given & Model-Learned

Sometimes a model of the environment is given to the agent in the form of a simulator, sometimes it has to be learnt by the agent from scratch. Often a ground-truth model of the environment is unavailable because the true environment dynamics are unknown. AlphaZero [REF] is a famous example of a model-based agent that learns the model from scratch, because at the beginning of training it does not know any of the game rules.

## On-Policy & Off-Policy

Off-policy RL is when the behavioural policy is different from the target policy. On-policy RL is when the behavioural policy is that same as the target policy. The behavioural policy is the policy the agent uses when acting on the environment to generate new experiences. The target policy is the policy that is used when learning from past experiences.

## Value-Based & Policy-Based

If the agent is trying to solve the problem by learning an optimal policy function the algorithm is policy-based. If the agent is trying to learn an optimal value function the algorithm is value-based. Actor-critic methods are hybrid algorithms where an agent is trying to learn both the policy and value functions. The actor (policy) acts in the environment and the critic (value) evaluates the actor’s actions and dictates improvements.

# Deep RL Algorithms (MAIN)

## Deep Q-Learning

Deep Q-Learning with Experience Replay (DQN) substantially launched the field of Deep RL. DQN uses a RL framework and deep learning to play Atari 2600 games implemented in the Arcade Learning Environment [REF]. Instead of using traditional value iteration algorithms the authors used a CNN to approximate the optimal state-action value function Q\*(s,a). The CNN takes in raw visual input of the Atari games as states and outputs a Q-value for each possible action. The agent then plays Atari by selecting actions according to the predictions of the network.

<Equation VI>

<Equation NN>

The game state is inputted to the CNN as 4 stacked histories each grey-scaled, down sampled and cropped into an 84x84 region. Multiple histories are used because a state cannot be fully represented in a single frame, for example, direction and velocity of the ball in ‘Pong’ cannot be shown by a single frame.

At each time step DQN stores experiences in a replay buffer to learn from later. This is significant because the batching and sampling of past experience enables stable training of the deep CNN. The network is trained by minimising a sequence of loss functions. Each function’s target for learning y-i comprises a Q-value as predicted by the previous iteration’s network and an immediate reward returned from the environment. Stochastic Gradient Descent (SGD) is used when updating the weights of the network according to the loss.

<Equation LOSS>

<Equation Li TARGET>

The network is trained for 10 million frames with a replay memory 1 million most recent frames. During training an e-greedy policy is used for action selection with greediness being annealed linearly from 1 to 0.1 over the first million frames. E-greedy selection is a trade-off between exploration and exploitation selecting the highest Q-value action with probability 1-e or an alternative action uniformly at random. In other words, the greater the value of E the more exploration and visa-versa.

<PsuedoCode>

Q-Value increased over training showing deep networks can be stably trained to solve RL problems. The results against previous art and human players show DQN outperform any traditional RL approach and even human score on 3 of the 7 games tested.

<RESULT GRAPH AND TABLE>

There have been many improvements and alterations to DQN since its publication. Deep Recurrent Q-Networks [REF], Double DQN (DDQN) [REF], Prioritized DDQN [REF], Duelling DDQN [REF], Distributional DQN [REF] and Noisy DQN [REF] are a few examples. With publications even successfully combining improvements to DQN. RAINBOW is an algorithm that integrates many of the previously mentioned DQN variants into a single hybrid algorithm resulting in better Atari performance.

This algorithm is a value-based algorithm because it attempts to learn the optimal state-action value function. It does not concern itself with learning the optimal policy, this would be role of a policy-based algorithm. DQN is model-free RL because it only predicts Q-values and does not learn an MDP model of the environment. DQN is off-policy RL because the behavioural policy is simply an E-greedy selection. Off-policy RL is when the behavioural policy is different from the target policy. On-policy RL is when the behavioural policy is that same as the target policy.

## A3C

The Asynchronous Advantage Actor-Critic (A3C) algorithm was introduced in a paper by Mnih .et.al [REF] that presented a general asynchronous framework for training deep RL agents. The framework used parallelism in the form of simultaneous actor-learner agents running on different CPU cores. Each actor-learner would explore environment states using different exploration policies and as they learnt they would update a global parameter set that would then be propagated to all other actor-learners. The framework reduced the hardware requirements for training Deep RL agents, provided a stabilising role to the learning that was previously done exclusively through experience replay, and enabled a large range of RL algorithms to be used effectively in Deep RL.

To test their framework the authors implemented asynchronous variants of four popular RL algorithms. The most successful of these algorithms was A3C an on-policy actor-critic policy gradient method. A3C uses an advantage function (eqref) as its critic computed by calculating feed forward n-step returns (eqref) using state evaluations according to the network V(st;th-t). A3C maintains a policy pi()and an estimate of the value function V(). A single CNN approximates both policy and value with a separate output head for each. The agent’s policy and value function are updated after a set number of actions are taken or a terminal state is reached. The update of the network performs gradient ascent using the policy gradient theorem (ref) . The authors use RMSProp [REF] for optimising updates.

[RESULTS A3C]

A3C beat previous art at playing Atari 2600, including DQN. A3C outperformed all agents in half the training time while being trained using 16 CPU instead of a GPU.

# Future Direction

For these algoithms to be used real world domains with significant consequcmes they need to be Iterpretable. Recent work in the field of interpretable machine learning is gonna solve this.

Model Extraction (White box view of black box classifiers)

<https://distill.pub/2020/circuits/zoom-in/> THIS PAPER it visualises neurons and claims neurons are typically understandable.

# NOTES

* Need a little sentence describing each sections goals