A Literature Review on Reinforcement Learning in Single, and Multi-Agent Systems, its Scalability, and Applications

Bala Subramanyam Duggirala
(Reviewed by Dr. Leen-Kiat Soh)
UNL School of Computing, 2023

Keywords:

Multi-agent Systems; Reinforcement Learning; Deep Learning; Deep Reinforcement Learning; Actor-Critic methods; Partial Observability; Scalability; Evolution Strategies

1. Abstract

This paper is a review of literature on multi-agent reinforcement and deep learning, including a high-level overview on a popular and modern approach of reinforcement learning, called the *actor-critic* method. It discusses how scalability can be a challenge for reinforcement learning, and how can various methods, including the *actor-critic* method, help resolve this challenge. In the process, it presents us with various applications and evaluation/analysis work carried out in the field, that would showcase reinforcement learning's potential to solve real-world problems. Given the above, the paper also briefly discusses *Evolution Strategies* (ES), a recent technique that has gained traction in the artificial intelligence and machine learning communities as a potential alternative to reinforcement learning.

2. Introduction

An intelligent agent is an autonomous entity that makes reasonable choices of actions to be able to successfully complete the tasks that it has been assigned to. Reinforcement Learning (RL) is a way for the agent to achieve the reasoning needed to do so, by choosing actions on a trial-and-error basis, and eventually learning what the best action for a given situation is. Rewards and penalties are the breadcrumbs that guide the agents to successfully learn these choices of appropriate or even optimal actions, also referred to as policies. In this way, even though the agent has zero knowledge in the beginning on how to deal with the task, it can eventually learn and successfully complete it on its own. Such a system where there is only one agent acting on the environment is called a single agent system.

In addition to the single agent systems, the highly successfully RL that is known for solving various Machine Learning (ML) problems employing sequential decision making [Zhang, et al. 2021], can also be seen in numerous real-world applications where more than one agent is needed so that a collective goal is achieved. For example, popular applications like Poker, Go, autonomous driving, etc., deal with more than one agent, and hence come under the domain of Multi-Agent Reinforcement Learning (MARL). MARL can be sub-divided into fully cooperative, fully competitive, or a mix of both, depending on the goals of each agent or the set of agents in the system, and these settings cover almost all kinds of daily life applications. Given this, however, scaling up with the number of agents becomes a challenge for MARL due to the intense computation required to reason about a highly dynamic environment which is a result of multiple agents acting together [Lorenzo, et al. 2021].

Deep Reinforcement Learning (DRL), also a subset of machine learning like RL, with its higher understanding of the visual world, integrates deep learning into RL, thereby addressing the issue of scalability that is inherent in the RL algorithms [Arulkumaran, et al. 2017]. This enables RL to work on very large sets of data, that are otherwise intractable without DL involved [Arulkumaran, et al. 2017].

In this literature review, in Section 3, we discuss how RL is a good learning strategy for a single agent as well as a multi-agent setting, primarily focusing on fully observable domains which can be considered as a baseline to RL problems. We also look into what is partial observability, and how it can pose a challenge to the single, or the multi agent systems. We start this section by explaining the basic RL model for a single-agent scenario. Then, we discuss a very common problem faced in RL, i.e., exploration vs exploitation, using the popular *multi-armed bandit* example. Next, we review various solutions to the multi-armed bandit problem from the literature. We also address the issue of scalability that plagues RL and review various solution approaches before finally moving to the multi-agent aspect of RL. In Section 4, we discuss how DRL makes RL even better by effectively dealing with the scalability issue that plagues it. This section also discusses fundamental and popular architectures of DRL and the associated algorithms, multiagent aspects and challenges. Then we discuss on a high-level, an important flavor of RL, called *actor-critic* in Section 5, which tries to make the learning better from a 2-dimensional perspective (in trying to select the actions (policy-based) vs. criticizing the actions taken so far (value-based)). This section also discusses how *actor-critic* can be used to scale RL, and how it is a more efficient approach, in general, for both RL and DRL (e.g., it helps them converge faster). Finally, Section 6 reviews Evolution Strategies (ES) and discusses it as a potential alternative to reinforcement learning.

3. Reinforcement Learning – Advantages, Common Issues, and Scalability

In an agent-based learning model, reinforcement learning is a way of programming the agent(s) to learn to perform a task, using rewards and penalty, and without specifying how the task needs to be performed [Kaelbling, Littman, and Moore 1996]. With this potential, in recent years, RL has gained rapid traction in the Artificial Intelligence (AI) and Machine Learning (ML) communities. RL is different from a more widely studied learning method called Supervised Learning, in that, in RL, the learner receives only partial feedback about its predictions of the environment, as opposed to full feedback that it receives if it were a Supervised Learning (SL) model [Szepesv´ari 2009]. In general, RL can be considered a subfield of ML and refers to a learning problem that must learn to control a system to maximize some numerical value (a.k.a. the reward). The idea is to successfully learn and complete a long-term objective [Szepesv´ari 2009], without any external supervision.

3.1 Reinforcement Learning Model

We now discuss a basic RL model described by [Kaelbling, Littman, and Moore 1996], where we have an agent that can sense an input '*i*' from the environment and can change the state '*s*' of the environment by performing an action '*a*'. The change of the state is informed to the agent in the form of a scalar reinforcement signal, '*r*'. Through trial and error, the agent should be able to learn the best actions for each state, a.k.a. policies, over time, with the help of improving reinforcement signals from the environment. The model can be formally described using:

1. The set of all states, *S*
2. The set of all actions that the agent can perform, *A*
3. The set of possible reinforcements for all states, *R*

Since this is a basic model, it comes with a few assumptions to keep it simple. Firstly, we assume that the agent knows, through reinforcements, the exact state of the environment. Such an

environment is called a fully observable environment. Also, the environment is usually non-deterministic, i.e., taking the same action in the same state might not always lead to the same resulting state (a.k.a. next state). This depends on the transition probabilities of the environment. Given this, in an RL model, the agent, by choosing the actions and estimating the likelihood of various resulting states with the help of improving reinforcements, should be able to come up with the policies for each state. Extending this idea for the simulation aspect of applications, a Markov Decision Processes (MDP) is a stochastic sequential decision process [Puterman 1990] in which the rewards, transitions (i.e., the reinforcements from the RL perspective) depend only on the current state of the environment, and the chosen action. The process is broken down into epochs (a.k.a.) time steps in which each agent makes a decision, and the environment transitions to the next state.

It is important to note that, in this discussed model, the background mechanics of the environment are stationary [Kaelbling, Littman, and Moore 1996] i.e., the predetermined transition probabilities or the rewards associated with a state do not change over time. Although there has been some research on implementing the slowly varying non-stationary environments [Kaelbling, Littman, and Moore 1996], there is very little theoretical analysis in this area.

As mentioned priorly, the reinforcement learning model is different from supervised learning where an agent is told which chosen action would have been the best, looking from a long-term perspective. Instead, it is up to the agent to figure this out in reinforcement learning. This is because by choosing an action, the agent only receives the immediate reward from the environment. To do so, the only way for the agent is to explore the environment. But how rigorously should the agent explore? It turns out that there is a trade-off between how much the agent explores vs. how much advantage it takes of what is already known. In the literature, this is commonly referred to as the *exploration vs exploitation trade-off* [Kaelbling, Littman, and Moore 1996].

3.2 Exploration vs Exploitation Trade-off: The K-Armed Bandit Problem

To explain the exploration vs exploitation trade-off, [Kaelbling, Littman, and Moore 1996] present us with a classic research problem explored in the reinforcement learning as well as applied mathematics and statistics literature, called the *k-armed bandit problem*. This is also the simplest reinforcement learning problem that gives us a good understanding of the *exploration vs. exploitation* concept. In this problem, the agent is in a room with *k* gambling machines, a.k.a. one-armed bandits, each with an arm that can be pulled by the agent. When the agent pulls the arm of, say, machine *i*, the machine would pay off a sum of either 1 or 0 to the agent. This payment depends on a fixed underlying probability, which the agent is unaware of, that decides whether the machine must pay 1 or 0. The agent is permitted to pull the arm of any machine for a total of *h* times. The goal of the agent here is to maximize the amount of the payoffs that it receives after choosing these *h* actions. How would the agent know which machine to choose? Let's say that the agent, through trial and error, tests a machine *i*, and now thinks that it gives a good return based off the immediate reward. Should it choose to continuously exploit this machine, or maybe go for other machines with an unknown chance where they might give it a better payoff over the time? It depends on the number of chances the agent is given, i.e., *h*, in this case. If *h* is fairly large, the agent could possibly expand its trials to test out different machines and eventually find the machine which pays off 1, with the highest probability, and still have time to exploit it. On the other hand, if the value of *h* is small, it might be better for the agent to stick with a well-known machine instead of wasting its limited moves on machines that are still uncertain to give good results.

3.2.1 RL Techniques to Solve the K-Armed Bandit Problem

The following are the solutions from the literature to the above single state k-armed bandit problem which were originally reviewed and presented by [Kaelbling, Littman, and Moore 1996]. Although these methods are instructive, they are restricted to simple problems and do not scale well to address more complex problems [Kaelbling, Littman, and Moore 1996].

In the following, the first three methods discussed (in Subsections 3.2.1.1, 3.2.1.2, and 3.2.1.3) are preferred for their robustness, test-based evidence and rigorous theoretical analyses in the research area, the next three methods discussed (in Subsections 3.2.1.4, 3.2.1.5, 3.2.1.6), originally surveyed by [Thrun 1992], are ad-hoc techniques that are popular in RL, not for their optimality but for their fairly sound reasonability, computational tractability, and heuristics.

3.2.1.1 Learning Automata (Finite State Automata)

Learning Automata (LA) are decision making devices that are adaptive and suited to work in unknown environments [Nowé, Verbeeck and Peeters 2005]. They were initially developed for modeling observed behaviors in the area of mathematical psychology studies. But in their current form, they are closely related to RL [Nowé, Verbeeck and Peeters 2005]. LA are strictly updated based on response from the environment, and not based on strategies/feedback (i.e., knowledge) from other automata. For this reason, they can be considered as reinforcement learners of the policy iteration type [Nowé, Verbeeck and Peeters 2005]. [Narendra and Thathachar 2012] quantitatively describe them as a learning paradigm that is described as follows. At a given time, a finite number of actions can be performed in a random environment. When a specific action is performed, the environment provides a random response which is either favorable or unfavorable. The main objective in the design of the automaton is to determine how the choice of action in the current state should be guided by past actions and the respective responses given by the environment.

Learning Automata uses a strategy to determine the action choice, where the internal state of the agent is described as a probability distribution. The actions can be chosen according to this distribution. The probabilities would be adjusted based on the outcome of the previous actions. We now look at an algorithm taken from the mathematical psychology literature [Hilgard and Bowler 1975], called the linear reward-inaction algorithm. Let p_i be the agent's probability of taking action *i*, and α be the learning rate.

1. When action a_i succeeds,
   $p_i = p_i + \alpha\,(1-p_i)$
   $p_j = p_j - \alpha p_j$ for $j \neq i$
2. When action a_i fails, p_j remains unchanged (for all j)

Although this algorithm converges to a single action, with a probability of 1, it does not always converge to the optimal action. The chance that it converges to the wrong action can greatly be reduced by using an arbitrarily small α value [Narendra and Thathachar 1974], because the smaller the value of alpha, the smaller the change in the value of *p* for each iteration. This would in-turn lead to a smoother convergence with lesser error bound.

3.2.1.2 Dynamic Programming Approach Using Bayesian Learning

Dynamic programming is where an agent dynamically stores and retrieves learning based information, and thereby avoids the overhead of redundant computation. This approach uses Bayesian learning to find an optimal strategy in the *k*-armed bandit problem with *h* actions to choose from [Berry and Fristedt 1985]. It maps belief states to actions, where a belief represents each action choice and its

payoff: $\{n_1, w_1, \ldots, n_k, w_k\}$. Here each n_i, w_i pair represents the number of times the arm-*i* has been pulled, and the number of times there was a pay-off from arm-*i*, respectively. The expected pay-off for the remaining *h*-$(n_1 + n_2 + \cdots + n_k)$ pulls, where *h* is the total number of times the game is played, is given by $V^*(n_1, w_1, \ldots, n_k, w_k)$ [Kaelbling, Littman, and Moore 1996]. If $n_1 + \ldots + n_k = h$, i.e., the agent has used all the available pulls, then $V^*(n_1, w_1, \ldots, n_k, w_k) = 0$,
which forms the basis for the following recursion.

$$V^*(n_1, w_1, \ldots, n_k, w_k) = max_i \, E\left[\begin{array}{l}\text{Future payoff if agent takes action i,}\\ \text{then acts optimally for remaining pulls}\end{array}\right]$$

$$= max_i \left[\begin{array}{l} p_i V^*(n_1,\ w_1,\ \ldots,\ n_i + 1,\ w_i + 1,\ \ldots, n_k,\ w_k) + \\ (1\text{-}p_i) V^*(n_1,\ w_1,\ \ldots,\ n_i + 1,\ w_i,\ \ldots, n_k,\ w_k)) \end{array}\right]$$

where p_i is the posterior subjective probability of action *i* paying off given n_i, w_i and our prior probability.

By using the above equation, if the agent knows the V^* for all the beliefs with *t* pulls remaining, the agent can compute the V^* value of any belief state with *t*+1 pulls remaining and uses this to choose the optimal action.

3.2.1.3 Gittins Allocation Indices

Allocation Indices can be described as a mathematical model which deals with optimizing in a sequential manner, the allocation of effort between several competing projects [Gittins, Kevin, and Richard 2011]. For a problem like the multi-armed bandit problem which deals with allocating certain efforts (a.k.a. actions) to projects (e.g., different arms that the robot can pull), the number of variables is at least equal to the number of projects. An attractive idea, therefore, is to establish and use priority indices for each of such project, such that they depend on a history that is unique to that project, and ultimately allocate the effort, at each time step, to only the project with highest current index value [Gittins, Kevin, and Richard 2011]. Such priority index for each project is called an 'Allocation Index' and can be used to find an optimal action choice at each step of a sequential effort allocation problem like a *k*-armed bandit problem [Gittins 1989]. More formally, let $x_1(t)$, …, $x_n(t)$ be the *n* states of the bandits, where *n* is the number of alternative bandit process of a multiarmed bandit problem. Let $(a_{1,\ldots,}a_{n)}$ be the set of available actions, where taking action a_j corresponds to the agent playing bandit *j*. The reward for this action is given by $R_j(x_j(\text{t})\ )$. While the change of state from this action is Markovian, the states of all other bandits remain unchanged. Let *j*(*t*)represent the bandit that is played during time-step *t*, and the evolution of the process is determined by a policy π, then the value of state x is given by [Weber 1992] as follows.

$$v_{\Pi}(x) = E_{\pi}\left[\sum_{t=0}^{\infty} \beta^t R_{j(t)}\big(x_{j(t)}(t)\big)\middle| x(0) = x\right],$$

where $0 < \beta < 1$.

The solution to the above problem can be characterized by using functions G_j, having the property that playing bandit *j* is optimal at t, if and only if

$$G_j(x_j(t)) = max_{1 \le i \le n}(G_i(x_i(t)))$$

The set of functions, G_j, are called Gittins indices [Weber 1992]. An important property of the above function is that it depends only on the information concerning bandit *j*, and this reduces the dimensionality of both the problem and its solution [Weber 1992]. Given the formal representation, [Gittins and Jones 1974] proved that "The optimal policy is to play at each epoch a bandit of the greatest Gittins index." Hence, by choosing the action which has the largest value of the Gittins Index, we can expect an optimal balance between exploration vs. exploitation. Because of this advantage of optimally balanced exploration, this method has a good potential to be used in more complex applications.

3.2.1.4 Greedy Strategies

Greedy strategies in RL are a set of methods used to feed the reinforcement using the actions that have the maximum expected payoff [Kaelbling, Littman, and Moore 1996], and hence the name "Greedy".

While this approach could save a lot of computational costs, because the agent only naively selects the "perceived" best action, there is always a risk of a sub-optimal action being chosen in place of the optimal action. This is possible because the sub-optimal action could have a higher reward for the current state compared to the optimal action. This would give the sub-optimal action an advantage in the reinforcement process, while the optimal action could end up starving. Only a reasonable course of exploration could bring out the potential of the best action that is otherwise hidden when greedy strategies are applied.

In order to overcome this unwanted behavior of choosing the suboptimal action, *optimism in the face of uncertainty* is a useful heuristic [Kaelbling, Littman, and Moore 1996]. Here, the actions are still selected greedily, but a strongly optimistic prior belief is associated with each action's reward, and this action would be eliminated as a choice only if it results in a strong penalty [Kaelbling, Littman, and Moore 1996]. Although there would still be a risk of starving the optimal action by dropping it in the exploration, this risk would now become arbitrarily small [Kaelbling, Littman, and Moore 1996]. Similar techniques have been used in several RL algorithms including the *exploration bonus* in *Dyna* [Sutton 1990], the *interval exploration method* [Kaelbling 1993a], *curiosity-driven exploration* [Schmidhuber 1991], and the exploration mechanism in *prioritized sweeping* [Moore and Atkeson 1993].

3.2.1.5 Randomized Strategies

This is a simple exploration strategy where an action, that gives the best expected reward, is chosen by default, and a random action is chosen with a probability *p*. The backing idea of this strategy is *controlled exploration*, and this can be done by tuning the value of *p*. It is usually set to a high value initially, to encourage exploration, and is then slowly decreased. A disadvantage of this strategy is that, when choosing the random action, it is no more likely that the optimal action gets picked than other sub-optimal actions. A slightly better and more sophisticated method is the *Boltzmann exploration* [Achbany, et al. 2008]. In this method, the expected reward, ER(a), of taking an action drives the probability of that action being picked, according to the following distribution:

$$P(a) = \frac{e^{ER(a)/T}}{\sum_{a\prime \in A} e^{ER(a\prime)/T}},$$

where *T* is the temperature parameter, to which the exploration is directly proportional to. Although this method suffers when the values of the actions are close, it works well if the best action is reasonably well separated from the other actions, and the temperature is carefully manually tuned [Kaelbling, Littman, and Moore 1996].

3.2.1.6 Interval-Based Techniques

Interval based techniques are the strategies used for choosing an action during the process of reinforcement learning based off the action's potential to belong to an optimal policy [Kaelbling 1993b]. Instead of directly learning the approximation of an action's optimal value, interval-based techniques try to determine an upper bound and a lower bound between which this value falls. These bounds are calculated using a confidence interval, given by $100\cdot(1-\alpha)\%$ on the probability of action being successful, where α is the learning rate. Ultimately, the action which has an interval with the highest upper bound value is selected [Roger 2018].

This method has been deemed useful for applications in empirical trials and is also related to experiment *design methodology* which is a class of statistical techniques [Box and Draper 1987]. These techniques are used to compare and select the best treatment out of a set of given treatments like fertilizers and drugs, using a set of experiments as small as possible, indicating their effective convergence rates.

3.3 Generalization: Addressing Scalability

Generalization is an area of research that overall aims to develop algorithms with robust, transferrable, and adaptive properties, and close the gap between training and testing [Robert, et al. 2021]. Generalization is also a term that usually refers to technique/set of methods used in RL to achieve scalability by compacting the stored information, and by redefining similar action and state spaces to make them less redundant in the reinforcement computations, without unreasonably affecting the decision-making process [Kaelbling, Littman, and Moore 1996]. The methods discussed (Section 3.2) so far assume that the action and state spaces of the environment are capable of being enumerated and stored into tables for performing computation. However, as the size of the environment grows larger and larger, it is impractical to keep storing the variables, given the limited computing memory capacities [Kaelbling, Littman, and Moore 1996]. This also translates to inefficient use of experience [Kaelbling, Littman, and Moore 1996].

In a very large state space, we generally assume the space to be smooth and that similar states have similar values as well as similar optimal actions. Given this, there should be a way to generalize these states and represent them compactly than using tables. As [Kaelbling, Littman, and Moore 1996] put it, while most problems will have large or continuous and discrete state spaces, a few problems will have large or continuous action spaces. The problem of learning in such large spaces can be done through generalization techniques, which come with two advantages: compact storage of information, as well as knowledge transfer between states and actions that are "similar" [Kaelbling, Littman, and Moore 1996].

The architectures and algorithms of the above-discussed methods include storing various mappings like $S\rightarrow A$ (policies), $S\rightarrow R$ (value functions), $S\times A\rightarrow R$ (Q functions and rewards), $S\times A\rightarrow S$ (deterministic transitions), $S\times A\times S\rightarrow[0,1]$ (transition probabilities) [Kaelbling, Littman, and Moore 1996]. Some of these methods, including transitions and immediate rewards, can be learned using supervised learning. Not only is this straightforward, but also can utilize a wide range of function-approximation techniques offered by supervised learning which support noisy training examples [Kaelbling, Littman, and Moore 1996]. Several neural network methods [Rumelhart and McClelland 1986], fuzzy logic [Berenji 1991] [Lee 1991], CMAC [Albus 1981], and local memory-based methods [Moore, Atkenson, and Schaal 1995] are some of the popular techniques in this area [Kaelbling, Littman, and Moore 1981]. A different type of mapping like policy-mapping, on the other hand, requires specialized algorithms because for these methods, the input-output pair training sets are not

available [Kaelbling, Littman, and Moore 1981]. The following subsections discuss the methods that address this problem and are broken down into three categories, i.e., 3.3.1 Generalization over Input (based on states as input), 3.3.2 Generalization over Actions (based on actions as output), and 3.3.3 Hierarchical Methods (miscellaneous), as originally surveyed by [Kaelbling, Littman, and Moore 1981].

3.3.1 Generalization over Input

In environments with very large and intractable state spaces, generalization techniques can be applied over these states which are fed as input to the agent's reinforcement learning process. This would reduce the overhead of the large state space thereby making the environment scalable.

The current state of an RL agent plays an important role in the selection of actions that maximize its reward [Kaelbling, Littman, and Moore 1981]. If we were to view the agent as a state-free black-box, its input would be the description of the current state, whereas the output would be the action selection, or the evaluation of the current state in order to select the best action, and it depends on the agent's architecture. *Structural credit-assignment problem* is a problem which decides how the output is affected by different values of the input [Kaelbling, Littman, and Moore 1981].

In this section, we very briefly discuss the approaches, with relevant example algorithms, to either generate the action or evaluate the current state in order to decide the action, given that the current state is given as input to the agent. It is broken down into two groups, where the first group discusses the special cases where the reward is immediate, and the second group addresses more general problems [Kaelbling, Littman, and Moore 1981].

3.3.1.1 Immediate Reward

When the state transitions of the environment are not affected by the actions of an agent, the agent's objective is to choose an action that maximizes the reward in the current state, i.e., the immediate reward [Kaelbling, Littman, and Moore 1981]. These problems, which come under the class *associative* reinforcement learning, are like the multi-armed bandit problems discussed in Section 3.2, except that the agent now must condition their choice of action based on the current state. The Complementary Reinforcement Backpropagation algorithm (CRBP) [Ackley and Littman 1990], the Associative Reinforcement Comparison algorithm (ARC) [Sutton 1984], the *REINFORCE* algorithms [Williams 1987, 1992] are popular examples that attempt to solve this problem.

3.3.1.2 Value-Function (Delayed-Reward) Approximations

Value function of an RL is composed mainly of two things, i.e., the immediate reward, and the estimate of the future discounted value of the current state, given an action. To estimate the precise value for a given state-action pair requires extensive computation, especially in domains with a large state space. Therefore, in order to scale-up, we look at this function from a generalizability perspective.

Delayed reward is an RL technique that is modeled based on value iteration and Q-learning, and since it has a value function that can be approximated, it enables these RL techniques to be applied to large state space problems [Kaelbling, Littman, and Moore 1981]. Unlike methods that use immediate reward as mentioned in Section 3.3.1.1, the agent that uses delayed reward techniques must also take into consideration the future states as well as their associated rewards when deciding an action. For example, a self-driving car's delayed reward could be to reach the destination safely. In such methods, the value function can be represented by mapping the state description to a value using a function "approximator" [Kaelbling, Littman, and Moore 1981], which adds the scope for generalization.

Much research has been done in this area [Kaelbling, Littman, and Moore 1981] to develop various methods depending on the application, like using local memory-based methods in combination with value iteration [Boyan and Moore 1994], using backpropagation networks for Q-learning [Lin 1991], using CMAC for Q-learning [Watkins 1989], implementing backpropagation for learning the value function in backgammon [Tesauro 1992, 1995], implementing good strategies for job-shop scheduling using backpropagation and TD(λ) [Zhang and Dietterich 1995], etc.,

Although there were a few good outcomes, especially in the game of backgammon [Tesauro 1992] [Boyan 1992], most of the research, in general, observed an issue of sub-optimal results (of value estimation), arising from the bad interactions between the function approximations used and the learning rules [Kaelbling, Littman, and Moore 1981]. Ideally, in discrete environments where generalization is not applied, the value-update operations which use the *Bellman* equations make sure that the error between the current value and the optimal value is reduced from iteration to iteration. However, this does not hold true when we apply generalization techniques, and [Boyan and Moore 1994] discuss and demonstrate this using a set of simple experiments. As a solution to this problem, they present an algorithm called the *Grow-Support* algorithm, which is not affected by the divergence in the values, and simultaneously benefits from the generalization. The *Grow-Support* algorithm assigns the values using "rollouts", instead of using single-step backups based on Bellman error. Also, [Boyan and Moore 1994] indicate that the example problems that they use to show the ineffectiveness of the generalization can be handled by problem-specific hand-tuning despite unreliability of algorithms that are untuned but nonetheless converge when limited to discrete domains. [Sutton 1996] also shows that [Boyan and Moore 1994]'s counterexample problems of optimal generalization can be successfully converged.

Given the above, whether or not the general principles of generalization using a delayed reward, ideally supported only by theory, can help us evaluate the value function approximation is still a question.

3.3.2 Generalization Over Actions

In domains with very large state and action spaces, if we can generalize the action space (along with the state space) by redefining similar actions that are resulted from similar states into a more compact space, the system can be scaled more efficiently. The problems described in the above Section 3.3.1.1 not only represent generalization of inputs given in the form of state descriptions, but also, as a result, generate a discrete and factored representation of outputs which is nothing but a generalization over the action space [Kaelbling, Littman, and Moore 1996]. In such cases where the actions are described in combination with the input state space, generalizing over the actions becomes important to avoid keeping separate set of statistics for large action spaces [Kaelbling, Littman, and Moore 1996].

The solution approach depends on whether the action space is continuous. For a discrete action space, using neural networks for generalization, we can either use a separate network for each action, or a network with an output that is distinct for each action [Kaelbling, Littman, and Moore 1996]. If the action space is continuous, on the other hand, the alternative option would be to use a single network that takes in the state and the action as inputs and gives out the Q value. One example of a solution for continuous action spaces was given by [Gullapalli 1990, 1992] in the form of a "neural" reinforcement learning unit. To be able to deal with large action spaces in general, when it comes to practical requirements, [Chandak, et al. 2019] have proposed a model that allows the agent to reason for the outcomes of the actions based on their similarity to the actions that were already taken in the past.

3.3.3 Hierarchical Methods

Hierarchical methods are another strategy of implementing generalization, and deal with large state spaces by treating them as a hierarchy of learning problems [Kaelbling, Littman, and Moore 1996]. Hierarchy is introduced by making use of gated structures called "*behaviors".* Various states are grouped into multiple behaviors thereby mapping them to the actions via these behaviors. Then, based on the state of the environment, a gating function decides which actions of a *behavior* the state needs to switch through and execute [Kaelbling, Littman, and Moore 1996]. Since various states are now mapped to a single action, this would help in scaling large state spaces.

As originally surveyed by [Kaelbling, Littman, and Moore 1996], the following are the different approaches taken by various authors and are differentiated based on how they treat the training of the gated functions as well as the *behaviors* using reinforcement learning. [Maes and Brooks 1990] tweaked the original architecture by pre-fixing the individual behaviors while the gating function was trained using RL. This helped their behavior-based robot to learn, based on positive and negative feedback, when its behavior should become active. [Mahadevan and Connell 1991] on the other hand fixed the gating function and enabled reinforcement learning for individual *behaviors*. [Lin 1993] used an approach where they trained the behaviors first, followed by training of the gating function. This hierarchical method empirically showed that the agent was able to boost its exploration in a new domain [Lin 1993].

3.4 Multi-Agent RL

Despite the general success of RL, a large number of real-world, practical problems are hard to solve using just a single agent. Multi-agent system (MAS) is a solution to this problem, where several agents interact simultaneously with the same environment and learn to solve the model [Lorenzo, et al. 2021]. More specifically, MARL is used to tackle the problem of sequential decision-making of several autonomous agents that are part of the same environment, and who try to interact with other agents, and the environment to maximize their own long-term return [Zhang, et al. 2021].

The applications of MARL are spread across different fields including traffic control, energy distribution, communication networks, and economy models, etc., [Lorenzo, et al. 2021]. There has already been tremendous success in a few of these applications such as playing Go, real-time strategy games, robotic control, card games, self-driving cars, etc., most of them owing their success to Deep Neural Networks for function approximation [Zhang, et al. 2021]. In addition to these popular applications, MARL also finds potential applications in other sub-areas that include finance, cyber-physical systems, sensor and communication networks, and social science [Zhang, et al. 2021]. These applications are a result of their ability to model both cooperative and competitive scenarios that involve multiple agents [Zhang, et al. 2021]. For this reason, there has been substantial research in the past few years, to scale the single-agent RL systems to work with multiple agents.

MARL can be challenging despite applying various categorization, optimization, and strategic techniques. MARL problems can generally be categorized, depending on the nature/goal of their interaction into three groups, i.e., fully cooperative, fully competitive, and a mix of both [Zhang, et al. 2021]. Here, the cooperative setting is where the agents collaborate and work towards optimizing a common reward; and the competitive setting is one where the sum of returns adds up to zero [Zhang, et al. 2021]; and the mixed setting is a combination of fully cooperative, and fully competitive settings. Despite employing several frameworks such as dynamic programming, optimization theory, game theory, decentralized control, etc., to these different MARL settings [Zhang, et al. 2021], numerous challenges, especially theoretical, still exist across these different settings.

One of the main challenges to develop MARL techniques is to deal with the dynamic nature of the environment. Empirical evaluations have shown that a naïve extension of a single-agent system to multiple agents will result in sub-optimal solutions [Lorenzo, et al. 2021]. This is because the rewards/feedback are no longer dependent solely on the actions of one agent but are a measure of action of each agent in the system, i.e., they are multi-dimensional with respect to different agents in the system [Zhang, et al. 2021]. This means that the environment, from the perspective of a single agent, is no longer stationary. This problem, called the *non-stationarity* of the environment, is the main problem that plagues the development of multi-agent RL models. In cases where acceptable optimality is achieved, they are usually restricted to a limited number of agents [Lorenzo, et al. 2021], and scaling them becomes another critical challenge. Adding to this, modeling the information structure which defines *who knows what* in MARL is very difficult since each agent has limited/no access to other agents' observations, states, etc., [Zhang, et al .2021], resulting in a sub-optimal local decision model.

Even with the above challenges of this new and budding field, the research community, thanks to the advances in single-agent RL [Zhang, et al .2021], has made reasonable progress in various domains that require multiple agents, like autonomous driving, UAV-related applications, etc. [Christopher D., et al. 2021], [Pham, H.X., et al. 2018], [Joel Z., et al. 2021], [Ming, et al. 2020]. The focus of such MARL research is placed on learning new/improving existing algorithms for various criteria/setups, and multi-agent operations research, making MARL a promising branch of ML [Lorenzo, et al. 2021].

3.5 Partial Observability in RL

In addition to the above-mentioned fully observable environments, there is also a class of RL models where the environment is only partially observable [Spaan 2012] by an agent. With partial observability, the above-defined MDPs (from Section 3.1) can be generalized to Partially Observable Markov Decision Processes (POMDPs) [Jaakkola, et al. 1994]. The learning problem is defined similarly to an MDP, with state space S = $\{s_1, s_2, s_3, \dots, s_n\}$, action space A = $\{a_1, a_2, a_3, \dots, a_n\}$, the rewards defined by *R(s, a)*, and due to partial observability, also including the observation space *O* = $\{o_1, o_2, o_3, \dots, o_n\}$. Here each *o* from the observation space corresponds to a specific state *s* from *S*, where the observation is what the agent 'observes' as the state of a given time step, but not necessarily accurately, given the partial observability (i.e., result of an observation noise).

RL with partial observability, where the agents make sequential decisions with only partial information about their current state, finds applications in almost all aspects of daily life [Chung et al. 2022]. For example, in robotics, the environment might not be completely revealed to the agent due to noisy sensors, obstructions, etc., [Akkaya et al. 2019]. Another example can come from imperfect information games where only local observations are made by the player [Vinyals, et al. 2019]. Other application examples, given by [Chung et al. 2022] include autonomous driving [Levinson, et al. 2011], resource allocation [Bower and Gilbert, 2005], business management, recommendation systems, medical diagnostics [Hauskrecht and Fraser, 2000], etc.,

Observations are non-Markovian in nature [Chung et al. 2022], and this requires the agents to maintain memory and, if needed, reason about possible beliefs states of the environment. Thus, partial observability makes RL very difficult [Chung et al. 2022], and this is supported by the information-theoretic results which show that solving POMDPs requires an exponential number of samples (of the belief space) in the worst cases.

For this reason, although RL systems have seen success in a set of domains that are defined by partial observability, like, Poker [Brown and Sandholm, 2019], Starcraft [Vinyals, et al., 2019], etc., the theoretical understanding of how an agent must act under partial observability remains limited

[Chung et al. 2022]. [Fujita, et al., 2005] describe that a multi-agent setting makes the RL problem more complex, in that, the agent now must consider the reasoning of the other agents, without being able to directly observe their state, and propose an approach to solve one of such problems by using function approximators.

3.6 RL Conclusion

Upcoming (WIP as of 2024/03/25)

4. Deep Reinforcement Learning (DRL) – Background, Relation to RL, Scalability, and Multi-Agent Aspect

Deep reinforcement learning (DRL) is a subfield of ML, that combines DL with RL. DRL uses DL to approximate various components of the reinforcement learning, like, the policy function, value function, and/or the model of the environment [Yuxi 2017], thereby helping to scale. It also has several advantages that come with its higher understanding of the visual world, including, for example, its applications in robotics by using information just from the camera feed, its ability to learn to play video games just by analyzing pixels from the screen, etc., [Arulkumaran, Kai, et al. 2017].

Due to its ability to efficiently process high-dimensional information, DRL can be considered as a solution to the scalability issue of reinforcement learning [Arulkumaran, et al. 2017]. Although reinforcement learning has proven to be a very useful tool for many applications, it is limited by its inability to scale well. This is mainly due to the memory, computational, and sample complexity that it inherits [Arulkumaran, et al. 2017]. Some of the important challenges faced in reinforcement learning can be summarized as follows [Arulkumaran, et al. 2017].

1. "The optimal policy must be inferred by trial-and-error interaction with the environment. The only learning signal that the agent receives is the reward."
2. "The observations of the agent depend on its actions and can contain strong temporal correlations."
3. "Agents must deal with long-range time dependencies: often the consequences of an action only materialize after many transitions of the environment. This is known as the (temporal) credit assignment problem" [Sutton and Barto 1998].

Deep reinforcement learning, on the other hand, handles the above challenges by relying on powerful function-approximation and representation-learning properties of deep neural networks. It has revolutionized scalability by providing new tools to solve these problems [Arulkumaran, et al. 2017]. Much of DRL's success is owed to scaling up the RL problems to higher dimensions, by using the ability of neural networks to approximate functions and learn low-dimensional feature representations of high-dimensional data, a.k.a. *representation learning*. Using this, DRL can deal efficiently with the curse of dimensionality, unlike tabular and traditional nonparametric methods [Bengio, Courville, and Vincent 2013]. Convolutional Neural Networks (CNNs), for example, using this concept, can enable RL agents to learn directly from raw, high-dimensional visual inputs [Arulkumaran, et al. 2017].

In general, DRL is used to train Deep Neural Networks for approximating the value and policy functions to guide them to an optimal value [Arulkumaran, et al. 2017]. These value-based and policy-based methods are discussed in Subsections 4.2 and 4.3, after we cover the background on Deep Learning in the following Subsection 4.1.

4.1 Background on Deep Learning, and its relation to Reinforcement Learning

Deep Learning (DL), a subset of Machine Learning (ML), "is a class of machine learning algorithms that: (1) uses a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input, (2) learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts" [Deng and Yu 2014].

Deep learning is based on how a human brain learns and processes information, and hence consists of a machine learning model which is comprised of several levels/layers of representation of data [El-Amir, Hisham, and Hamdy 2020]. Each of these layers uses information from the previous layers (think of piecing together various shapes/strokes of a hand-written digits' image that is inputted in the form of pixels to the input layer) to learn the data more deeply and is part of a *neural network*. Abstracted/hidden units called *perceptrons* [El-Amir, Hisham, and Hamdy 2020] are the building blocks of these artificial neural networks (ANNs). Here, a perceptron is an algorithm that classifies linear binary input data (i.e., a binary classifier "which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class [Freund, Y., & Schapire, R. E. 1998]"). In the DL architecture, this *Unit* (i.e., a perceptron) is the simplest function that contains the aspect of machine learning. The ML involved may be of any type, including supervised, unsupervised, semi-supervised or reinforcement [El-Amir, Hisham, and Hamdy 2020]. This unit, in turn, is usually defined by an *activation function,* like regression, logistic regression, or clustering function, etc., [El-Amir, Hisham, and Hamdy 2020]. These activation functions are used in ANNs to transform an input signal into an output signal. The generated output signal is, in-turn, fed as an input signal to the next layer. This is usually done by calculating the sum of product of all the inputs and their respective weights, for the layer at-hand [Sharma, et al. 2017]. Since the weights associated with such inputs play an important role in determining the signal transmitted between any two layers, "training" of a neural network deals with tuning these weights in order optimize the signal generation between the layers. This usually requires a large amount of training data.

It is intuitive by now that more than one unit like the one described above, together create a layer. Each unit of such a layer is connected to each unit of the previous layer to make a fully connected neural network [El-Amir, Hisham, and Hamdy 2020]. In a typical ANN, there are three layers, i.e., input layer, hidden/encoder layer, and an output layer. A network with only 1-2 hidden layers is called a *shallow* neural network, and a network where one hidden layer is connected to a deeper hidden layer and consists of many such hidden layers is called a *deep* neural network [El-Amir, Hisham, and Hamdy 2020]. The following Figure 4.a shows the difference between a shallow vs deep neural network.

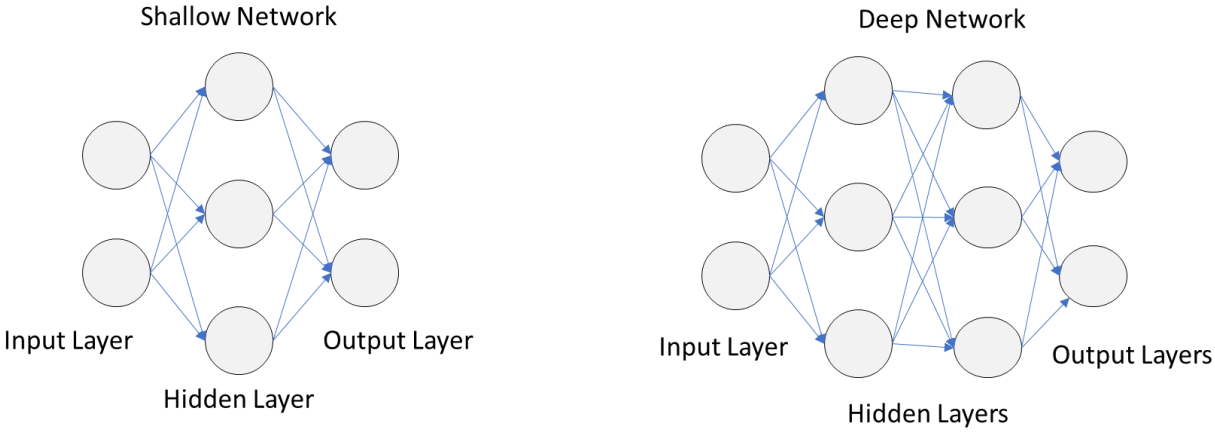

Figure 4.a depicts the representation of a Shallow Neural Network (Left) vs. a Deep Neural Network (Right). The difference between Shallow and Deep Neural Networks is that Deep Neural Networks have at least two hidden layers.

Since these neural networks act as function approximators [Arulkumaran, et al. 2017], they can be very useful in RL, especially when the state/action spaces are too large to be known completely. The value function, i.e., mapping of states to values, or the policy function, i.e., mapping of the state-action pairs to Q-values, can be approximated using these neural networks [Arulkumaran, et al. 2017]. Instead of storing, indexing and accessing all the possible states and their values in a tabular fashion [Arulkumaran, et al. 2017], neural networks train on samples from the state/action spaces in order to predict the value of these states with respect to the goal in RL, thereby enabling high scalability.

4.2 Value-based DRL Methods

Value-based algorithms are a class of algorithms that aim to build a value function and help us define a policy subsequently [Vincent, et al. 2018]. In this Section, we first introduce the basic value-based method, i.e., the *Q-learning* method, in Subsection 4.2.1. We then discuss its variants that aim to address the scalability challenge, i.e., *Fitted Q-learning, Neural Fitted Q-learning*, and *Deep Q-learning* in Subsections 4.2.2 to 4.2.4.

4.2.1 Q-learning

Q-learning is one of the simplest and most popular value-based algorithms [Vincent, et al. 2018]. The Q-learning algorithm stores a lookup table for the quality of the state and action pairs, i.e., a function given by *Q(s,a)*, whose value is a combination of the past experience, and the expected utility of taking the action, *a,* in the state, *s*. For the Q-function, the algorithm uses the *Bellman* equation, whose unique solution, *Q*(s,a)* is given by

$$Q^*(s, a) = (BQ^*)(s, a),$$

where *B* is the *Bellman operator*, mapping any function $K: S \times A \rightarrow R$, into another function, $S \times A \rightarrow R$. It is given by:

$$(BK)(s,a) = \sum_{s' \in S} T(s,a,s')(R(s,a,s') + \gamma max_{a \in A} K(s',a')),$$

where *K* is the function that maps the states to actions, *T* is the transition probability function, *R* is the reward function, and γ is the discount factor.

The above basic equation represents a simple setting which is often not applicable in complex and high-dimensional state-action space [Vincent, et al. 2018]. In such cases, a parameterized value function, $Q(s, a; \vartheta)$ is required, where ϑ refers to some parameters which define the value of the Q-function.

4.2.2 Fitted Q-learning and Neural Fitted Q-learning

Fitted Q-learning is a variant of the Q-learning algorithm which approximates the Q-function to address the scalability issues in complex domains. It does so by breaking down the problems into a series of regression tasks [Srijita, et al. 2020]. The algorithm uses pre-computed experiences in the form of a dataset tuple $D = \langle s,a,r,s' \rangle$, and also initializes the algorithm with random Q-values, Q(s, a; θ_0) to avoid overfitting of the Q-function. Here, θ_0 refers to the initial parameters, which are then updated using the experiences-dataset, as well as further exploration of the state-action space.

The Neural Fitted Q-learning (NFQ) algorithm provides an efficient structure that can compute the value of $max_{a' \in A}$Q(s'a';θ_k), for a given s', in a single-forward pass of the neural network [Vincent,

et al. 2018]. This is possible because the state is provided as an input to the Q-network, and in the output layer, a different output is given for each possible action, helping the algorithm pick the max-action [Vincent, et al. 2018]. Here, the values of the parameter, θ_k can be updated using a neural network compatible algorithm like the stochastic gradient descent, or a variant [Vincent, et al. 2018].

4.2.3 Deep Q-Networks (DQNs)

Deep Q-Networks (DQNs), originally introduced by [Mnih, et. al 2015], approximate the action-value function, i.e., the Q-function, of an agent, by combining reinforcement learning with a class of artificial neural networks, called the *deep neural networks.* Using these approximated functions, DQNs solve the problems where the agent needs to process high-dimensional sensory inputs from the environment, generalize their past representations, and account for new encounters. [Mnih, et. al 2015] demonstrate that, using just the pixels and the games scores as input, DQN algorithms have achieved a performance that is equivalent to that of a human professional, across a set of 49 games in the challenging domain of *Atari 2600* video games [Bellemare, et al. 2015].

Picking up their advantages from the deep neural networks, DQNs employ several layers to develop an abstract representation of the input, thereby enabling the system to learn concepts such as object categories, directly from the input sensory data [Mnih, et al. 2015]. With the problem of dealing with a high-dimensional state space resolved, DQNs now apply reinforcement learning to these abstracted states to estimate the optimal Q-function which given as follows.

$$Q^*(s,a) = max_{\prod} E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots],$$

where E is the estimated value, *r* is the reward, γ is the discount factor, and *t* is the time-step.

4.3 Policy-based DRL Methods

A policy is something that maps a state to an action [Yuxi 2017]. Policy-based methods directly search for an optimal policy $\prod^*$ without having to maintain a value-function model [Arulkumaran, et al. 2017]. A parameterized policy, $\prod_\theta$, is typically chosen using either a gradient-based or gradient-free optimization method, such that the parameters, θ, are updated to maximize the expected return, *E(R|ϑ)* [Arulkumaran, et al. 2017]. Neural networks have been successfully trained to encode these parameters using both gradient-free and gradient-based methods.

Low-dimensional parameter-spaces can be effectively covered by using gradient-free optimization techniques [Arulkumaran, et al. 2017]. Also, there have been some successes in applying these techniques in larger spaces [Arulkumaran, et al. 2017]. Evolutionary algorithms are one such example of the gradient-free policy-based methods. These algorithms use heuristic searches to find the best policy and come under the category of black-box optimization algorithms (black-box optimization means that we cannot access the function but can only provide inputs and observe outputs) [Arulkumaran, et al. 2017]. We will be discussing more about these algorithms in Section 5.

However, the gradient-based methods remain the choice of technique for the most DRL algorithms [Arulkumaran, et al. 2017], because they are more sample efficient when the policies consist of a large set of parameters. The REINFORCE algorithm [Williams 1987, 1992] is a well-known policy-gradient method. The logic behind these algorithms, with its ability to approximate the gradient function, allows them to be applied to the neural networks to learn stochastic policies using a task-dependent method [Arulkumaran, et al. 2017]. For example, it helps the algorithm to learn a stochastic variable that would determine a cropped part of an image, thereby reducing computation required and helping scale-up.

4.4 Model-based DRL Methods

The main idea of model-based RL is that the agent learns the environment's transition model which helps with the simulation of the environment, without directly interacting with the environment itself [Arulkumaran, et al. 2017]. Given this, it plays an important role in cutting down the number of required interactions with the actual environment that might not be practically possible (for example, training a robot by actually moving its arms and legs millions of times might not be practical with respect to time-consumption, and the hardware wear-and-tear) [Arulkumaran, et al. 2017]. Model-based RL, in general, has no assumption of prior knowledge of a specific environment. However, to speed-up the learning, a prior knowledge of the environment, for example, physics-based models [Kansky, Ken, et al. 2017] can also be incorporated.

To apply model-based techniques to DRL methods, there are several approaches that use pixel information of dynamic models to learn their predictive models. Several model-based DRL algorithms have been proposed which use pixel information to learn policies and models [J. Oh, et al. 2015] [N. Wahlström, et al. 2015] [M. Watter, et al. 2015].

Although model-based deep neural networks are able to reasonably predict the models of the simulated environments, taking hundreds of time-steps [Gu, S., et al. 2016], they usually need to be fed with a large number of samples to tune the large number of parameters. For this reason, [Arulkumaran, et al. 2017] posit that advances in improving neural networks' data efficiency could promote the usage of deep models in model-based DRL.

4.5 Multi-agent DRL

With the ability to apply neural networks as function approximators, and therefore adding scalability that was otherwise not possible, DL has caused a surge of interest in multi-agent RL to now instead use DRL [Gronauer and Diepold 2021] [Arulkumaran, et al. 2017]. Due to the ability to use these function approximators, DL methods have been successfully mastered complex control tasks in areas including robotics [Levine, et al. 2016] [Lillicrap, et al. 2016] and game playing [Mnih, et al. 2015] [Silver, et al. 2016] [Gronauer and Diepold 2021].

[Gronauer and Diepold 2021] have recently done a survey addressing the above, where they discuss this emerging field with regards to how to train the agents, the challenges, and the future scope which are presented in the following subsections 4.5.1, 4.5.2, and 4.5.3.

4.5.1 Training and Execution of a Multi-agent DRL System

Training is a process which helps the agent to obtain data, and in-turn, experience, and helps optimize their policies based on the obtained rewards [Gronauer and Diepold 2021]. For a multi-agent DRL system, given the involvement of multiple agents, the training of the system can be divided into two approaches, i.e., a centralized approach, or a distributed approach [Weiß G 1995].

In a *centralized* approach, the training is applied such that each agent learns their policy based on the mutual exchange of information from the other agents. In a *distributed* approach, however, each agent learns their policy without using the information from the other agents.

Just like the training schemes, there can also be two approaches for how the agent executes their actions in a simulation. While a joint action for each agent is computed by a central system in a centralized execution scheme, each agent processes and decides their own action in a distributed execution scheme [Gronauer and Diepold 2021].

The practically employed training and execution schemes based on the above classification can be broken down into three types [Gronauer and Diepold 2021], i.e., Distributed Training-Decentralized Execution (DTDE), Centralized Training-Centralized Execution (CTCE) and Centralized Training-Decentralized Execution (CTDE).

In a DTDE paradigm, each agent views the environment as non-stationary, and they neither have access to other agents' information nor they know the joint action of all the agents [Gronauer and Diepold 2021]. Due to these factors, the learning of the agents in the system is much slower compared to that of centralized learning [Tan 1993]. This paradigm also suffers from poor scalability with the increasing number of agents. This is due to the independent training of each agent which results in worsening space complexity. On top of that, experiments by [Gupta, et al. 2017] have shown that the policies that are generated have inferior performance compared to the ones generated by a centralized training scheme.

Coming to the CTCE paradigm, this is analogous to a single-agent system in a sense that a single centralized executioner can be utilized to learn the joint policy for all the agents [Gronauer and Diepold 2021]. This advantage lets this paradigm directly apply single-agent actor-critic [Mnih, et al. 2016] and policy gradient methods [Schulman, et al. 2017] to a multi-agent setting. However, the disadvantage of this paradigm comes in the form of the curse of dimensionality of the state-action space of the centralized executioner that worsens with increasing number of agents.

The paradigm that is the best amongst the ones discussed so far for a multi-agent learning setting is the third paradigm, i.e., the CTDE paradigm, which has been very successful in MADRL [Foerster, et al. 2016] [Jorge, et al. 2016]. This paradigm's effectiveness comes mainly from two advantages. First, since it is a centralized training approach, each agent can utilize shared resources/communication to effectively increase the speed when compared to the agents that are independently trained. Secondly, combining the first advantage with the decentralized execution, the 'non-stationarity' that is an outcome of a centralized execution scheme can be bypassed, since the agents, due to a centralized training, would be able to pin-point various transitions in the environment to the actions of other individual agents.

A recently proposed paradigm called the *leader-following* paradigm has better benchmark results than the CTDE paradigm [Zhang, et al. 2022]. This paradigm overcomes the disadvantages of the CTDE approach where all the agents choose their respective actions at the same time possibly ignoring the heterogenous roles of different agents [Zhang, et al. 2022]. It does so by first letting a leader agent broadcast their choice of action in the form of a message, and then letting the various other agents make their decisions, given the leader's decision. [Zhang, et al. 2022] show that their simulation results outperform four existing state-of-the-art benchmark paradigms, including CTDE, in the challenging cooperative environment.

4.5.2 Challenges in Multi-agent DRL

In their survey about multi-agent DRL, [Gronauer and Diepold 2021] present six main challenges that arise from having to deal with multiple agents in the system. They are non-stationarity, learning communication, coordination, credit assignment problem, scalability, and partial observability. They are discussed in the below Subsections 4.5.2.1 through 4.5.2.6.

4.5.2.1 Non-stationarity

Non-stationarity is a major problem in a system where multiple agents try to learn by simultaneously interacting with the environment [Gronauer and Diepold 2021]. The dynamics of the environment

appear to be non-stationary for each agent, given their co-adaption, and this constitutes a moving target problem [Gronauer and Diepold 2021]. Since the Markov property, which assumes that the agents don't need to have memory of the past, is violated, this now becomes a difficult learning problem [Hernandez-Leal, et al. 2017] [Laurent, et al. 2011].

While the naïve approaches of either assuming that there are no other agents [Matignon, et al. 2012], or assuming the other agents' behaviour to be static [Lauer and Riedmiller 2000] enable us to easily apply single-agent RL solutions to a multi-agent setting, this is likely to result in poor performance, especially in complex domains [Lowe, et al. 2017] [Matignon, et al. 2012]**.** [Lanctot, et al. 2017] argue that this could be because the independent learners might overfit the training data simultaneously failing to generalize.

4.5.2.2 Communication

When it comes to communication among the agents of a multi-agent system, the vital problem for the agent is to figure out not just what to communicate, but also who to communicate with, and when. [Gronauer and Diepold 2021] have surveyed and analysed the literature discussing the challenges and the outcomes of addressing the messages, and categorized them as follows: 1) A broadcasting scenario where each agent addresses the messages to each other agent in the system, 2) Targeted communication that utilizes an attention mechanism which determines what messages are to be transmitted to whom and when, 3) Communication networks where each agent communicates with only their local neighbours that are part of the same network.

4.5.2.3 Coordination

Coordination of different agents in a (cooperative) multi-agent system is not only a key aspect for the desired outcome but is also a challenge. "Successful coordination in multi-agent systems requires agents to agree on a consensus" [Wei Ren et al. 2005]. The joint action of all the agents should be able to optimize the joint goal of each cooperating agent [Gronauer and Diepold 2021]. The challenge for this cooperation comes from the stochasticity and rewards of the environment, or from the agent's inability to completely observe the state of the environment [Gronauer and Diepold 2021]. Due to these factors, the actions of one agent might shadow the search space of the other agents in the reinforcement learning process thereby leading to miscoordination. As a result, suboptimal policies are ensued [Gronauer and Diepold 2021].

A solution to the above problem would be to share mutual information between the agents to prevent this miscoordination, in order to help the agents model each other better. This is possible with either 1) explicit communication between the agents, or 2) implicitly modelling the other agents [Gronauer and Diepold 2021]. While the former needs the agents to have the right communication skills to overcome the challenges of communication as addressed in Subsection 4.5.4.2, the latter requires that the agents have the ability to observe the other agents' behaviour and predict their strategies to choose their own action in-turn [Gronauer and Diepold 2021].

4.5.2.4 The Credit Assignment Problem

The credit assignment problem is a problem faced by an agent in a multi-agent system where they try to estimate each other agent's contribution to the reward outcome in order to thrive in a cooperative setting. When the agents do not have access to the joint action, the credit assignment problem becomes a challenge even in a fully observable environment [Gronauer and Diepold 2021]. This becomes a bigger challenge in partially observable environments.

The solution to the credit assignment problems, i.e., the *decomposition* of rewards, has seen early contributions from [Chang et al. 2004] and [Ng, et al. 1999]. While the former applied Kalmann filters to the reward function to estimate the decomposition, the latter tried modifying the reward function using reward shaping. More recent approaches, on the other hand, focused on exploiting the dependencies between the agents in order to decompose the reward, based on each agent's actual contribution to the global reward [Kok and Vlassis 2006].

4.5.2.5 Scalability

Although DRL solves the problem of scalability faced by the traditional RL methods to a great extent by using the function approximation techniques, this is still a challenge with respect to increasing number of agents in the system [Gronauer and Diepold 2021].

As a solution to this problem [Gronauer and Diepold 2021], the agents can accelerate the learning process by 1) re-using the distributed and shared knowledge between different agents, 2) reducing the complexity of the learning problem, and 3) becoming robust against the change in policies of other agents in the system. [Gronauer and Diepold 2021] discuss various techniques that implement these strategies to effectively scale the MADRL systems.

4.5.2.6 Partial Observability

In most of the real-world scenarios, the agents usually are able to neither observe the environment completely nor know the internal states of the other agents. This constitutes for a partially observable environment. Further, since the environment now appears to be non-Markovian to the agent, this becomes a challenge in a multi-agent DRL [Gronauer and Diepold 2021] system. Also, if the agents cannot estimate, due to partial observability, what their peers are doing in the environment, this might lead to a *lazy agent* problem, where an agent might reason that their actions would hinder optimizing the global reward, and therefore reasons not to act [Sunehag, et al. 2018]. Also, in many settings, the agents have limited-to-no communication [Amato, et al. 2017] which makes the learning problem more difficult. Agents locally view the environment as stationary, while this is not true due to the presence of other acting agents in the system that cannot be observed by the subject agent.

Not only can the state space be partially observable, but there can be domains where the agent can only partially observe the possible tasks, and the identities of these tasks are not visible or are only partially visible to the agent. [Amato, et al. 2017] deal with such a multi-task multi-agent setting under partial observability and present an approach that develops a unified policy that can perform reasonably well across different related tasks, whose identity cannot be observed by the agent.

While a natural way to fulfil the gaps in being unable to reason in a partially observable environment is by enabling information sharing, via messages for example, there can also be other ways like memory mechanisms, value-based approaches, actor-critic methods etc., as originally surveyed by [Gronauer and Diepold 2021].

4.5.3 Future Scope and Conclusion of MADRL

Although MARL enjoys a good success from the recent past, the scalability to complex and real-world type of applications had always been a challenge [Gronauer and Diepold 2021]. Breakthroughs in the Deep RL methods have spurred a rapid transformation of the field, and we are able to solve the problems that were previously intractable, and this is becoming possible even for problems with real-world complexity [Baker, et al. 2020], [Berner, et al. 2019], [Jaderberg, et al. 2019], [Vinyals, et al. 2019].

5. Actor-Critic Methods – Introduction and Advantages

Actor-Critic methods are a class of RL algorithms that have been gaining popularity in the recent past, because of their advantage of being able to search for optimal policies using low-variance gradient estimates. They are useful in several real-life applications, such as robotics, power control, and finance [Grondman, et al. 2012]. They form a third class of RL algorithms, which can be considered as a hybrid of the other two, i.e., actor-only and critic only methods, and hence the name – actor-critic. While the term *actor*, here, is analogous to 'policy-based', the term *critic* is analogous to 'value-based' reinforcement learning methods [Grondman, et al. 2012]. The actor-critic method, being a hybrid of these two methods, inherits the advantages from both.

Critic-only methods, such as *Q-Learning* and *SARSA*, use a state-action value function, and no explicit function for the policy [Grondman, et al. 2012]. By relying extensively on the value-function approximation, they aim at learning the Bellman equation's approximate solution, which in-turn will hopefully lead to a near optimal policy [Konda and Tsitsiklis 1999]. Moreover, the critic-only methods that use temporal difference (TD) learning (a class of model-free RL in which the learning starts by bootstrapping a value-function estimate) have a very small variance in the value-function estimates [Grondman, et al. 2012]. A naïve way to derive the policy in critic-only methods is to select greedy actions [Sutton and Barto 1999], i.e., the actions whose current value-function estimate is the highest. To implement such a strategy in a continuous action space, however, can be computationally expensive. Hence the critic-only methods usually must discretize this continuous action space in order to build the value function, thereby affecting the optimality of the value function itself.

Actor-only methods, on the other hand, work with a parameterized family of policies, i.e., the gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in a direction of improvement [Konda and Tsitsiklis 1999]. For this reason, a major advantage of actor-only methods over critic-only methods is that they allow the policy to generate actions in the complete continuous action space [Grondman, et al. 2012]. However, the actor-only methods suffer from a high-variance in the gradient-estimation, and this leads to slow learning [Grondman, et al. 2012].

Actor-Critic methods have the respective advantages of the actor-only and critic-only methods. While the actor-only methods bring the advantage of efficiently computing a continuous action space without needing the value function optimizations, the critic-only methods provide low-variance feedback on the actor's policy. This means that this feedback from the critic would allow the actor to update with gradients with less-variance, which in-turn speeds up the learning process [Grondman, et al. 2012]. These advantages have made *actor-critic* the choice of RL algorithms, even for real-life applications [Grondman, et al. 2012].

Along with the advantages mentioned above, some of the defining components of the actor-critic algorithms, for example, function approximation, bring in a disadvantage in the form of overestimated value estimates, and suboptimal policies [Fujimoto, et al. 2018]. Although this problem is extensively studied under RL that uses discreet action spaces, it is largely left untouched with actor-critic methods employed for continuous control domains [Fujimoto, et al. 2018]. This overestimation bias (that comes from using function approximations) is a property of the Q-learning algorithms where consistent overestimation of a value-function results from the maximization of a noise value estimate [Thrun & Schwartz, 1993]. While such a bias is an obvious by-product of maximization in a discreet action setting, its presence, and hence the effect, are less clear in gradient-descent-employing actor-critic methods where the policy is updated via a gradient descent. [Fujimoto, et al. 2018], in their popular publication, establish that overestimation is also a problem in actor-critic methods, and also

propose novel mechanisms that effectively deal with this issue by using double Q-learning. In this method, they take the minimum value between a pair of critics, thereby minimizing the over-estimated value. Their results show that mitigating this problem greatly improves the performance of modern actor-critic algorithms.

[Li, et al. 2008] have first introduced an actor-critic model that scales to a multi-agent setting. This model simply assumes that the actor-agent observes each other agent's action and rewards and maintains their policy-parameters and joint Q-values [Li, et al. 2008]. These Q-values are updated in a linear fashion (i.e., separately for each other agent) using a temporal best-response strategy, so that the computation is simpler than similar algorithms like *NASH-Q* and *EXORL*, which use a quadratic programming solution [Li, et al. 2008] to achieve value-equilibria. The critic-agent, on the other hand computes a state-value function, just like in a single-agent setting. Much recently, another work by [Lowe, et al. 2017] proposed a simple extension of the policy-gradient techniques of the actor-critic methods, where the critic now has additional information about the other agents, while the actor only deals with local information, like in a single-agent setting. This algorithm operates under the constraints that 1) the learned policies can only use local information in the form of each agent's own observation at the execution time, 2) a differential model of the environment dynamics is not assumed, and no particular communication channel is assumed between the agents. These constraints make sure that the algorithm stays a general-purpose multi-agent learning algorithm that can be applied to not only cooperative games with communication included, but also to competitive games, and games where only physical interactions between the agents are involved [Lowe, et al. 2017]. [Lowe, et al. 2017] prove that their (now-popular) method outperforms the traditional RL algorithms in various multi-agent environments like cooperative, competitive, etc., Given this, partial observability remains a challenge of its own, i.e., with partial observability, policies usually map to values and actions from the observation history of the agent, making the learning problem harder even with cooperative settings to begin with [Srinivasan, et al., 2018].

Given there are various kinds of requirements in the modern real-world applications, and there are numerous solutions to choose from, [Grondman, et al. 2012], with their survey, helpfully discuss a few thumb rules regarding when to choose which algorithms, and mention that the type of control policy should be the first thing that needs to be considered to do so. For example, if the control policy needs to produce actions in a continuous space, it might not be helpful to use critic-only algorithms, as they perform poorly with continuous action spaces (like we discussed above), and vice-versa. If presented with a continuous state and action spaces on the other hand, in a (quasi-) stationary environment, actor-critic algorithms, due to the least variance they provide, could be the best solution [Grondman, et al. 2012].

6. Evolution Strategies as a Potential Alternative to Reinforcement Learning

The goal of this section is to refer readers to a set of algorithms called *Evolution Strategies (ES)*, and compare their advantages/disadvantages with their counterpart, i.e., the RL algorithms. In this section, we don't delve into the technicalities of the ES, but just their potential to be an alternative to the RL.

"Evolution Strategies (ES) is a class of black box optimization algorithms [Rechenberg and Eigen, 1973], [Schwefel, 1977] that are heuristic search procedures inspired by natural evolution: At every iteration ("generation"), a population of parameter vectors ("genotypes") is perturbed ("mutated") and their objective function value ("fitness") is evaluated" [Salimans, et al. 2017]. The parameter vectors that score the highest are recombined and become the next generation's population. This procedure is repeated until the objective function is fully optimized.

Black box optimization (BBO), the class that encompasses ES has several attractive properties like indifference towards reward-distribution (dense vs. sparse), not needing backpropagation gradients, working well even with very long time-horizons, and is a good candidate as an alternative to RL algorithms [Salimans, et al. 2017].

[Salimans, et al. 2017] in particular explore ES against popular MDP-based RL techniques, i.e., Q-learning and policy-gradients. The results with experiments on popular domains like *Atari* and *MuJoCo* show that this is indeed a viable option, which scales extremely well with the number of available CPUs [Salimans, et al. 2017]. They also implement a novel communication strategy between the resources, which makes it possible for the system to scale to over a thousand parallel workers (CPUs). Using this, [Salimans, et al. 2017] were able to solve a 3D humanoid-walking in 10 minutes, and with just 1 hour of training, were able to obtain competitive results on most *Atari* games.

In their survey, they also go on to explain what makes it easy to make ES scalable and parallelisable; how ES are different from the traditional RL algorithms, in that, they derive their learning signal by sampling instantiations of policy parameters as opposed to RL's way of sampling actions from a stochastic policy, and what is its impact; in what cases are the ES better than using policy gradient methods etc.,

7. Conclusion

With this literature review, we present the readers with the fundamentals of a promising AI branch, i.e., RL, and discuss its various flavours, challenges, and applications, like multi-agent RL, DRL, multi-agent DRL, scalability-issues and solutions, the concept of *Actor-Critic,* and also a potential alternative to RL called *Evolution Strategies*. Although we don't dive deep into the details of some of these techniques, we still provide numerous references including some of the best work in the field. Hopefully, this review should be able to give the readers a fundamental idea of the field, and also guide them to more extensive work of their specific interest.

8. References

[Achbany, et al. 2008] Achbany, Youssef, et al. Tuning continual exploration in reinforcement learning: An optimality property of the Boltzmann strategy. *Neurocomputing* 71.13-15: 2507-2520, 2008.

[Ackley and Littman 1989] Ackley, David, and Michael Littman. Generalization and scaling in reinforcement learning. *Advances in neural information processing systems 2*, 1989.

[Ahmadian, A. S. 2016]. Amir Sharif Ahmadian. Numerical Modeling and Simulation, in Numerical Models for Submerged Breakwaters, *Coastal Hydrodynamics and Morphodynamics* [Book], 2016.

[Akkaya et al. 2019] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving Rubik's cube with a robot hand. arXiv preprint arXiv:1910.07113, 2019.

[Albawi, et al. 2017] Albawi, Saad, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network*. 2017 international conference on engineering and technology (ICET). Ieee*, 2017.

[Amato, et al. 2017] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., & Vian, J. (2017, July). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning* (pp. 2681-2690). PMLR, 2017.

[Arulkumaran, et al. 2017] Arulkumaran K., Deisenroth M.P., Brundage M., Bharath A.A. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process Mag*; 34(6):26-38, 2017.

[Arulkumaran, Kai, et al. 2017] Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866, 2017.

[Baker, et al. 2020] Baker B, Kanitscheider I, Markov T, Wu Y, Powell G, McGrew B, Mordatch I. Emergent tool use from multi-agent autocurricula. *International conference on learning representations*. https://openr eview.net/forum?id=SkxpxJBKwS, 2020.

[Bellemare, et al. 2015] Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res*. 47, 253–279, 2013.

[Bengio, Courville, and Vincent 2013] Bengio, Y., Courville, A. & Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Machine Intell*. 35, 1798–1828, 2013.

[Berner, et al. 2019] Berner C, Brockman G, Chan B, Cheung V, Debiak P, Dennison C, Farhi D, Fischer Q, Hashme S, Hesse C, Józefowicz R, Gray S, Olsson C, Pachocki JW, Petrov M, de Oliveira Pinto HP, Raiman J, Salimans T, Schlatter J, Schneider J, Sidor S, Sutskever I, Tang J, Wolski F, Zhang S. Dota 2 with large scale deep reinforcement learning. ArXiv arxiv: abs/1912.06680, 2019.

[Berry and Fristedt 1985] Berry, D. A. and B. Fristedt. Bandit Problems. Sequential Allocation of Experiments. Monographs on Statistics and Applied Probability. *Chapman and Hall, London/New York*, 275 S. Biom. J., 29: 20-20, 1985.

[Box and Draper 1987] Box, George EP, and Norman R. Draper. Empirical model-building and response surfaces. *John Wiley & Sons*, 1987.

[Boyan 1992] Boyan, Justin A. Modular neural networks for learning context-dependent game strategies. *University of Cambridge. Computer Laboratory*, 1992.

[Boyan and Moore 1994] Boyan, Justin, and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing* systems 7, 1994.

[Christopher D., et al. 2021] Hsu, Christopher D., et al. Scalable Reinforcement Learning Policies for Multi-Agent Control. *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021.

[Collobert and Bengio 2004] R. Collobert and S. Bengio. Links between Perceptrons, MLPs and SVMs. *Proc. Int'l Conf. on Machine Learning* (*ICML*), 2004.

[Chandak 2019] Chandak, Yash, et al. Learning action representations for reinforcement learning*. International conference on machine learning. PMLR*, 2019.

[Chang, et al. 2004] Chang Y, Ho T, Kaelbling LP. All learning is local: Multi-agent learning in global reward games. In: Thrun S, Saul LK, Schölkopf B (eds) *Advances in neural information processing systems 16, MIT Press*, pp 807–814. http://papers.nips.cc/paper/2476-all-learning-is-local-multi-agent-learn ing-in-global-reward-games.pdf, 2004.

[Chung et al. 2022] Liu, Q., Chung, A., Szepesvári, C., & Jin, C. (2022, June). When Is Partially Observable Reinforcement Learning Not Scary?. In *Conference on Learning Theory (pp. 5175-5220). PMLR*, 2022.

[Cybenko 1989] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4), 303–314, 1989.

[Deng and Yu 2014] L. Deng, and D. Yu, Deep Learning: Methods and Applications, *Foundations and Trends in Signal Processing*. 7 (3–4): 197–387, 2014.

[El-Amir, Hisham, and Hamdy 2020] El-Amir, Hisham, and Mahmoud Hamdy. Deep learning pipeline. 279-343, 2020.

[Fausset 1994] Fausset, L.V. Fundamentals of Neural Network: Architecture, Algorithm, and Application. *New Jersey: Prentice Hall*, 1994.

[Foerster, et al. 2016] Foerster J, Assael IA, de Freitas N, Whiteson S. Learning to communicate with deep multi-agent reinforcement learning. In: Lee DD, Sugiyama M, Luxburg UV, Guyon I, Garnett R (eds) *Advances in neural information processing systems 29, Curran Associates, Inc*., pp 2137–2145. http://papers. Nips.cc/paper/6042-learning-to-communicate-with-deep-multi-agent-reinforcement-learning.pdf, 2016.

[Freund, Y., & Schapire, R. E. 1998] Freund, Yoav, and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Proceedings of the eleventh annual conference on Computational learning theory*, 1998.

[Fujimoto, et al. 2018] Fujimoto, Scott, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *International conference on machine learning. PMLR*, 2018.

[Fukushima and Miyake 1982] K. Fukushima and S. Miyake, Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets, pp. 267–285, Springer*, 1982.

[Fujita, et al., 2005] Ishii, S., Fujita, H., Mitsutake, M., Yamazaki, T., Matsuda, J., & Matsuno, Y. (2005). A reinforcement learning scheme for a partially observable multi-agent game. *Machine Learning*, 59, 31-54.

[Gittins 1989] Gittins, J.C. Multi Armed Bandit Allocation Indices. *Wiley-Interscience series in systems and optimization*. Wiley, Chichester, NY, 1989.

[Gittins, Kevin, and Richard 2011] Gittins, John, Kevin Glazebrook, and Richard Weber. Multi-armed bandit allocation indices. *John Wiley & Sons*, 2011.

[Gronauer and Diepold 2021] Gronauer, S., Diepold, K. Multi-agent Deep Reinforcement Learning: A Survey. *Artif Intell Rev* 55, 895–943. https://doi.org/10.1007/s10462-021-09996-w, 2021.

[Grondman, et al. 2012] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems*, *Man, and Cybernetics, Part C* (Applications and Reviews), 42(6):1291–1307, 2012.

[Gupta, et al. 2017] Gupta JK, Egorov M, Kochenderfer M. Cooperative multi-agent control using deep reinforcement learning. In: Sukthankar G, Rodriguez-Aguilar JA (eds) autonomous agents and multiagent systems. *Springer, Cham*, pp 66–83, 2017.

[Gu, S., et al. 2016] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep Q-learning with model-based acceleration, in *Proc. Int. Conf. Learning Representations*, 2016.

[Haykin 1999] Haykin, S., Neural Networks: A Comprehensive Foundation. 2nd Edition, *Prentice- Hall, Englewood Cliffs, NJ*, 1999.

[Hernandez-Leal, et al. 2017] Hernandez-Leal P, Kaisers M, Baarslag T, de Cote EM. A survey of learning in multiagent environments: dealing with non-stationarity. *CoRR* arxiv: abs/1707.0918, 2017.

[Hilgard and Bowler 1975] Hilgard, E.R., and Bower, G. H. Theories of Learning (fourth edition). *Englewood Cliffs, N.J.: Prentice-Hall, Inc., NASSP Bulletin*, 60(400), 134–134, 1975.

[Hochreiter and Schmidhuber 1997] Hochreiter, Sepp and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation* 9: 1735-1780, 1997.

[Hubel and Wiesel 1962] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology, vol. 160, no. 1, pp. 106–154*, 1962.

[Jaakkola, et al. 1994] Jaakkola, Tommi, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. *Advances in neural information processing systems 7*, 1994.

[Jaderberg, et al. 2019] Jaderberg M, Czarnecki WM, Dunning I, Marris L, Lever G, Castañeda AG, Beattie C, Rabinowitz NC, Morcos AS, Ruderman A, Sonnerat N, Green T, Deason L, Leibo JZ, Silver D, Hassabis D, Kavukcuoglu K, Graepel T. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science 364(6443):859–865,* 2019.

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

[Jeffrey 1990] Elman, Jeffrey L. Finding Structure in Time. Cognitive Science. 14 (2): 179–211. doi:10.1207/s15516709cog1402_1, 1990.

[Joel Z., et al. 2021] Leibo, Joel Z., et al. Scalable evaluation of multi-agent reinforcement learning with melting pot. *International Conference on Machine Learning*. PMLR, 2021.

[Jorge, et al. 2016] Jorge E, Kågebäck M, Gustavsson E. Learning to play guess who? And inventing a grounded language as a consequence. *CoRR* arxiv: abs/1611.03218, 2016.

[Kaelbling 1993a] Kaelbling, Leslie Pack. Hierarchical learning in stochastic domains: Preliminary results. *Proceedings of the tenth international conference on machine learning.* Vol. 951, 1993.

[Kaelbling 1993b] Kaelbling, Leslie Pack. Learning in Embedded Systems. *MIT Press*, 1993.

[Kaelbling, Littman, and Moore 1996] Kaelbling L. P., Littman M. L., and Moore, A.W. Reinforcement Learning, A Survey. *Journal of Artificial Intelligence Research*, Vol 4, 237-285, 1996.

[Kansky, et al. 2017] Kansky, Ken, et al. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *International conference on machine learning. PMLR*, 2017.

[Kok and Vlassis 2006] Kok, Jelle R., and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research* 7: 1789-1828, 2006.

[Konda and Tsitsiklis 1999] Konda, V., and Tsitsiklis, J.N. Actor-Critic Algorithms. *Advances in neural information processing systems* 12, 1999.

[Kravaris, Vouros 2022] Kravaris, T., Vouros, G. A. Deep Multiagent Reinforcement Learning Methods Addressing the Scalability Challenge, in *I. Sheremet (ed.)*, Multi-Agent Technologies and Machine Learning [Working Title], *IntechOpen*, London. 10.5772/intechopen.105627, 2022.

[Lanctot, et al. 2017] Lanctot M, Zambaldi V, Gruslys A, Lazaridou A, Tuyls K, Perolat J, Silver D, Graepel T. A unifed game-theoretic approach to multiagent reinforcement learning. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) *Advances in neural information processing systems 30, Curran Associates, Inc.,* pp 4190–4203. http://papers.nips.cc/paper/7007- a-unifed-game-theoretic-approach-to-multiagent-reinforcement-learning.pdf, 2017.

[Lauer and Riedmiller 2000] Lauer M, Riedmiller M. An algorithm for distributed reinforcement learning in cooperative multiagent systems. *Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann*, pp 535–542, 2000.

[Laurent, et al. 2011] Laurent GJ, Matignon L, Fort-Piat NL. The world of independent learners is not markovian. *Int J Knowl-Based Intell Eng Syst* 15(1):55–64. http://dl.acm.org/citation.cfm?id=1971886.1971887, 2017.

[LeCun, Bengio, et al. 1995] Y. LeCun, Y. Bengio, et al., Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[LeCun, et al. 1998] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324*, 1998.

[Levine, et al. 2016] Levine S, Finn C, Darrell T, Abbeel P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17(1):1334–1373. http://dl.acm.org/citation.cfm?id=2946645. 2946684, 2016.

[Levinson, et al. 2011] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[Li, et al. 2008] Li, Chun-Gui, Meng Wang, and Qing-Neng Yuan. A multi-agent reinforcement learning using actor-critic methods. *2008 International Conference on Machine Learning and Cybernetics. Vol. 2.* IEEE, 2008.

[Li and Soh 2004] Li, X., Soh, L.-K. Investigating Reinforcement Learning in Multiagent Coalition Formation. Technical report no. WS-04-06, *American Association of Artificial Intelligence Workshop on Forming and Maintaining Coalitions and Teams in Adaptive Multiagent Systems*, pp. 22–28, 2004.

[Lillicrap, et al. 2016] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D. Continuous control with deep reinforcement learning. In: *ICLR* (Poster). http://arxiv.org/arxiv: abs/1509.02971, 2016.

[Lin 1991] Lin, Long Ji. Programming robots using reinforcement learning and teaching. *AAAI*, 1991.

[Lin 1993] Lin, L-J. Hierarchical learning of robot skills by reinforcement. *IEEE International Conference on Neural Networks*. IEEE, 1993.

[Lorenzo, et al. 2021] Canese, Lorenzo, et al. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences* 11.11: 4948, 2021.

[Lowe, et al. 2017] Lowe R, WU Y, Tamar A, Harb J, Pieter Abbeel O, Mordatch I. Multi-agent actor-critic for mixed cooperative-competitive environments. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) *Advances in neural information processing systems 30, Curran Associates, Inc.,* pp 6379–6390. http://papers.nips.cc/paper/7217-multi-agent-actor-critic-formixed-cooperative-competitive-environments.pdf, 2017.

[Maes and Brooks 1990] Maes, Pattie, and Rodney A. Brooks. Learning to Coordinate Behaviors. *AAAI*. Vol. 90, 1990.

[Mahadevan and Connell 1991] Mahadevan, Sridhar, and Jonathan Connell. Scaling reinforcement learning to robotics by exploiting the subsumption architecture. *Machine Learning Proceedings 1991*. Morgan Kaufmann. 328-332, 1991.

[Matignon, et al. 2012] Matignon L, Gj Laurent, Le fort piat N. Review: independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. Knowl Eng Rev 27(1):1–31. https://doi. Org/10.1017/S0269888912000057, 2012.

[Medsker, et al. 1999] Medsker, Larry, and Lakhmi C. Jain, eds. Recurrent neural networks: design and applications. CRC press, 1999.

[Ming, et al. 2020] Zhou, Ming, et al. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. arXiv preprint arXiv:2010.09776, 2020.

[Moore and Atkeson, 1993] Moore, Andrew W., and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. Machine learning 13.1: 103-130, 1993.

[Mnih, et al. 2015] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D. Human-level control through deep reinforcement learning. Nature 518:529 EP –. https://doi.org/10.1038/nature14236, 2015.

[Mnih, et al. 2016] Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Balcan MF, Weinberger KQ (eds) *Proceedings of The 33rd international conference on machine learning, PMLR, New York, New York, USA, Proceedings of machine learning research*, vol 48, pp 1928–1937. http://proceedings.mlr.press/v48/mniha 16.html, 2016.

[Murugan and Shanmugasundaram 2017] Murugan, Pushparaja, and Shanmugasundaram Durairaj. Regularization and optimization strategies in deep convolutional neural network. arXiv preprint arXiv:1712.04711, 2017.

[Narendra and Thathachar 1974] Narendra, Kumpati S. and Mandayam A. L. Thathachar. Learning Automata – A Survey. *IEEE Trans. Syst. Man Cybern.* 4: 323-334, 1974.

[Narendra and Thathachar 2012] Narendra, Kumpati S., and Mandayam AL Thathachar. Learning automata: an introduction. *Courier corporation*, 2012.

[Ng, et al. 1999] Ng AY, Harada D, Russell S. Policy invariance under reward transformations: theory and application to reward shaping. *In Proceedings of the sixteenth international conference on machine learning, Morgan Kaufmann*, pp 278–287,1999.

[Nowé, Verbeeck and Peeters 2005] Nowé, Ann, Katja Verbeeck, and Maarten Peeters. Learning automata as a basis for multi agent reinforcement learning. International Workshop on Learning and Adaption in Multi-Agent Systems. *Springer*, Berlin, Heidelberg, 2005.

[N. Wahlström, et al. 2015] N. Wahlström, T. B. Schön, and M. P. Deisenroth, From pixels to torques: policy learning with deep dynamical models, in *ICML Workshop on Deep Learning*, 2015.

[Pham, H.X., et al. 2018] Pham, H.X.; La, H.M.; Feil-Seifer, D.; Nefian. A. Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage. arXiv 2018, arXiv:1803.07250, 2018.

[Puterman 1990] Martin L. Puterman, Chapter 8 Markov decision processes, Handbooks in Operations Research and Management Science,Elsevier,Volume 2,Pages 331-434,ISSN 0927-0507,ISBN 9780444874733, https://doi.org/10.1016/S0927-0507(05)80172-0. (https://www.sciencedirect.com/science/article/pii/S0927050705801720), 1990.

[Rechenberg and Eigen, 1973] I. Rechenberg and M. Eigen. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. *Frommann-Holzboog Stuttgart*, 1973.

[Robert, et al. 2021] Kirk, Robert, et al. A survey of generalisation in deep reinforcement learning. arXiv preprint arXiv:2111.09794, 2021.

[Roger 2018] McFarlane, Roger. A survey of exploration strategies in reinforcement learning. *McGill University*, 2018.

[Rosenblatt, F. 1961] Rosenblatt, Frank. X. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. *Spartan Books*, Washington DC, 1961.

[Rumelhart, et al. 1986] Rumelhart, David E., Geoffrey E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. David E. Rumelhart, James L. McClelland, and the PDP research group. (editors), *Parallel distributed processing: Explorations in the microstructure of cognition, Volume 1: Foundation. MIT Press,* 1986.

[Salimans, et al. 2017] Salimans, Tim, et al. Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864, 2017.

[Sharma, et al. 2017] Sharma, Sagar, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards data science* 6.12: 310-316, 2017.

[Schmidhuber 1991] Schmidhuber, Jürgen. Curious model-building control systems. *Proc. International joint conference on neural networks*, 1991.

[Schulman, et al. 2017] Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. CoRR arxiv: abs/1707.06347, 2017.

[Silver, et al. 2016] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484 EP –. https://doi.org/10.1038/nature16961, 2016.

[Spaan 2012] Spaan, Matthijs TJ. Partially observable Markov decision processes. Reinforcement Learning. *Springer, Berlin, Heidelberg*. 387-414, 2012.

[Srijita, et al. 2020] Das, Srijita, et al. Fitted q-learning for relational domains. arXiv preprint arXiv:2006.05595, 2020.

[Srinivasan, et al., 2018] Srinivasan, S., Lanctot, M., Zambaldi, V., Pérolat, J., Tuyls, K., Munos, R., & Bowling, M. (2018). Actor-critic policy optimization in partially observable multiagent environments. *Advances in neural information processing systems*, 31, 2018.

[Sutton 1990] Sutton, Richard S., Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Machine learning proceedings* 1990. *Morgan Kaufmann*, 216-224, 1990.

[Sutton and Barto 1999] Andrew, A.M.: Reinforcement learning: An Introduction by Richard S. Sutton and Andrew G. Barto, *Adaptive Computation and Machine Learning Series, MIT Press* (bradford book), Cambridge, Mass., xviii+ 322 pp, isbn 0-262-19398- 1, (hardback,£ 31.95).-. Robotica 17(2) 229–235, 1999.

[Sunehag, et al. 2018] Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K, Graepel T. Value-decomposition networks for cooperative multi-agent learning based on team reward. *Proceedings of the 17th international conference on autonomous agents and multiagent systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS '18*, pp 2085–2087. http://dl.acm.org/citation.cfm?id=3237383.3, 2018.

[Szepesv´ari 2009] Csaba Szepesvári. Algorithms for Reinforcement Learning. *Morgan & Claypool Publishers*, 2009.

[Tan 1993] Tan M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In: *In Proceedings of the tenth international conference on machine learning, Morgan Kaufmann, pp 330–337*, 1993.

[Tesauri 1991] Tesauro, Gerald. Practical issues in temporal difference learning. *Advances in neural information processing systems* 4, 1991.

[Tesauro 1995] Tesauro, Gerald. Temporal difference learning and TD-Gammon. *Communications of the ACM* 38.3 (1995): 58-68, 1995.

[Thrun 1992] Thrun, S. B. The role of exploration in learning control. In White, D. A., & Sofge, D. A. (Eds.), Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches. *Van Nostrand Reinhold, New York, NY*, 1992.

[Valueva, et al. 2020] Valueva, Maria V., et al. Application of the residue number system to reduce hardware costs of the convolutional neural network implementation. *Mathematics and computers in simulation* 177: 232-243, 2020.

[Vinyals, et al. 2019] Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, Choi DH, Powell R, Ewalds T, Georgiev P, Oh J, Horgan D, Kroiss M, Danihelka I, Huang A, Sifre L, Cai T, Agapiou JP, Jaderberg M, Vezhnevets AS, Leblond R, Pohlen T, Dalibard V, Budden D, Sulsky Y, Molloy J, Paine TL, Gulcehre C, Wang Z, Pfaf T, Wu Y, Ring R, Yogatama D, Wünsch D, McKinney K, Smith O, Schaul T, Lillicrap T, Kavukcuoglu K, Hassabis D, Apps C, Silver D. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature* 575(7782):350–354. https://doi.org/ 10.1038/s41586-019-1724-z, 2019.

[Vincent, et al. 2018] François-Lavet, Vincent, et al. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning 11.3-4*: 219-354, 2018.

[Wasserman and Schwartz 1988] Wasserman, P.D., Schwartz, T. Neural networks II. What are they and why is everybody so interested in them now? Page(s): 10-15; *IEEE Expert, Volume 3, Issue 1*, 1988.

[Watkins 1989] Watkins, Christopher John Cornish Hellaby. Learning from delayed rewards, 1989.

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

[Thrun & Schwartz, 1993] Thrun, S. and Schwartz, A. Issues in using function approximation for reinforcement learning. *In Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

[Wei Ren et al. 2005] Wei Ren, Beard RW, Atkins EM. A survey of consensus problems in multi-agent coordination. In*: Proceedings of the 2005, American control conference, 2005*., pp 1859–1864 vol. 3. https://doi. org/10.1109/ACC.2005.1470239, 2005.

[Weiß G 1995] Weiß G. Distributed reinforcement learning. In: Steels L (ed) The biology and technology of intelligent autonomous agents. *Springer, Berlin,* pp 415–428, 1995.

[Yu and He 2006] Yu, B., He, X., Training radial basis function networks with differential evolution. In: *Proceedings of IEEE International Conference on Granular Computing*, Atlanta, GA, USA, 369-372, 2006.

[Yuxi 2017] Li, Yuxi. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274, 2017.

[Zell 1994] Zell, Andreas. Simulation Neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.). *Addison-Wesley*. p. 73. ISBN 3-89319-554-8. 1994.

[Zhang and Thomas 1995] Zhang, Wei, and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. *IJCAI*. Vol. 95, 1995.

[Zhang, et al. 2022] Zhang, Feiye, Qingyu Yang, and Dou An. A leader-following paradigm based deep reinforcement learning method for multi-agent cooperation games*. Neural Networks* 156: 1-12, 2022.

[Zhang, et al. 2021] Zhang, Kaiqing, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control* (2021): 321-384, 2021.

9. Appendix with Information on Basic Deep Neural Networks

This subsection discusses different types of deep neural networks from the list originally presented by by [El-Amir, Hisham, and Hamdy 2020], and briefly explains each one. The idea is to present the readers with various kinds of basic neural networks that would expose them to various DL architecture types, simultaneously presenting them with the areas where these networks are most applied.

9.1 Feedforward Neural Network

This is a type of ANN where the connections between different layers do not form a cycle, and the information only moves in the forward direction, i.e., from the input layers to the hidden layers, and then to the output layers [Zell 1994]. The advantage of these feedforward neural networks is that they are they have the simplest architecture, and hence enable boosted machine learning.

9.2 Radial Basis Function Neural Network (RBFNN)

This type of ANN uses radial basis functions as activation functions. RBFNNs are commonly used for approximation problems [Ahmadian, A. S. 2016]. They are a type of feedforward neural network that has only three layers, i.e., the input, hidden, and the output layers, each of which are designed to handle different tasks [Haykin, S. 1999] [Yu and He 2006]. The main advantages of this type of neural networks are their universal approximation and faster learning speed [Haykin, S. 1999].

9.3 Multilayer Perceptron

A Multilayer Perceptron (MLP) is a type of feedforward ANN that is fully connected, with three types of layers, i.e., input, hidden and output layers, where the hidden and output layers use a non-linear activation function. MLP uses a supervised learning technique called back-propagation for training [Rosenblatt, F. 1961] [Rumelhart, et al. 1986]. Backpropagation is a mechanism where the output layer, based on the differences between the expected and the actual outcomes (also defined as a "cost/reward" function), influences its predecessor layer to adjust their weights accordingly. With the optimally adjusted weights, this predecessor layer would now be able to reduce the error in the cost function and send an optimal signal to the output layer, ultimately resulting in optimal output. The predecessor layer in-turn recursively updates its own predecessor layer and so on, and hence the name backpropagation. As an added advantage to backpropagation, by using a non-linear activation function, MLPs can distinguish data that is linearly not separable [Cybenko 1989].

MLPs, due to their ability to solve problems stochastically, are often applied to obtain approximate solutions for very complex problems. As given by Cybenko's theorem [Cybenko 1989], MLPs are universal function approximators, and using regression analysis, they can be used to develop mathematical models. These neural networks were especially popular in the 1980s, and had applications in various fields like image recognition, speech recognition, and machine translation software [Wasserman and Schwartz 1988]. Although they had since then faced strong competition from Support Vector Machines (SVMs), which were much simpler and related [Collobert and Bengio 2004], interest in back-propagation (and hence MLPs) returned with successes in deep learning.

9.4 Convolutional Neural Network

In the field of deep learning, Convolutional Neural Networks (CNNs) are a class of ANNs, which are considered as a promising tool for solving the pattern recognition problem [Valueva, et al. 2020s]. CNNs take their name from convolution, a mathematical linear operation between matrices [Albawi, et al. 2017]. CNN architecture consists of four main background ideas, i.e., local connection, sharing the weights, pooling the data, and implementation of multilayers [LeCun, Bengio, et al. 1995]. CNNs are made of multiple layers that include convolutional and non-linearity layers that have calculation parameters, and the pooling and fully connected layers that don't have any calculation parameters [Albawi, et al. 2017].

These neural networks have a great advantage when it comes to pattern recognition, especially, when compared against the traditional MLPs which have a quite a few drawbacks when they are applied in image processing. One of the major problems is the scalability due to the large sets of data associated with typical imagery. Due to their full connectivity, MLPs also tend to overfit the

training data, making it difficult for the architecture to generalize new data that was not part of the original training data [Murugan and Shanmugasundaram 2017]. CNNs address this problem through "regularization" (penalizing parameters, trimming connectivity etc.,) of this training data [Murugan and Shanmugasundaram 2017]. This is possible for CNNs because these networks are inspired by biological processes and were originally developed to solve visual imagery recognition based on mathematical problems [Murugan and Shanmugasundaram 2017]. In their study on a cat's visual cortex system, [Hubel and Wiesel 1962] proposed that the visual cortex is capable of sensing and responding to small sub-regions of the overall field of vision. This region "is known as *receptive field*, where simple cells in the visual cortex respond to edge like patterns, and the larger receptive fields are invariant to the position of the natural images" [Hubel and Wiesel 1962]. Based on this location and orientation sensitivity of the neurons in the visual cortex, [Fukushima and Miyake 1982] introduced CNNs to solve the problem of pattern recognition. These were later developed by [LeCun, et al.] for identifying the digital characters. Hence, the main idea behind CNNs, that distinguishes them from traditional MLPs, is that the neurons from respective local fields gather elementary information and combine them over several layers to produce sensible patterns from complex original images [Murugan and Shanmugasundaram 2017].

9.5 Recurrent Neural Network – Long Short-Term Memory

A recurrent neural network (RNN) is a network with feedback connections involving closed loops [Fausset 1994]. These are different from the feedforward neural networks where the connections between the neurons don't form cycles. RNNs are designed to learn sequential or time-varying patterns [Medsker, et al. 1999]. The architecture of the recurrent neural networks ranges from fully connected to partially connected networks [Medsker, et al. 1999]. Long Short-Term Memory (LSTM) is a type of RNN whose name is accounted for both kinds of memory properties of a typical RNN, i.e, the long-term and the short-term memories. The weights and biases of a typical RNN change once every training step, and this is analogous to a "long-term memory", like some physiological memories the human brain stores. Activation patterns, on the other hand, change from time-step to time-step, and are analogous to how a human brain stores short-term memories [Jeffrey 1990]. Since the LSTM can sustain a short-term memory over thousands of time-steps, it gets the name "long" short-term memory.

Given their defining properties, LSTMs are well-suited for applications in classifications, processing, and prediction tasks that are based on time-series data. Their long short-term memory comes in handy in such applications where the lags between important events in a time series data are usually of unknown durations. For example, when trying to recognize a live speech, the input depends on how fast the speaker talks.