**Introduction**

With the amount of computing power and the huge amount of data available for the researchers and companies to draw out insights and providing us with information about how, what, when and where has totally lifted the use of AI and Machine Learning to new heights.

Machine Learning on a very coarse level has been divided into 3 major areas, namely Supervised Learning [1], Unsupervised Learning [2], and Reinforcement Learning [3].

* Supervised Learning [1], it mainly deals with labeled data. Labeled data means that when the data is being trained, we already have told the learning algorithm about given inputs and expected outputs. A few examples of supervised machine learning are of linear regression, naïve bayes, logistic regression, K-Nearest Neighbor, Support-Vector Machine, decision tree, neural networks etc. So the main requirement for this type of machine learning is labeled data.
* Unsupervised Learning [2], mainly uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms try to discover hidden patterns or groupings of data without the need for human interference. These abilities to discover similarities and differences in information makes it the ideal solution for different approaches like exploratory data analysis, bio-informatics, clustering, analysis of sequences, segmentation of customers, image recognition and many more. Few of the unsupervised learning approach are K-Means Clustering, Association Rules, Hierarchical Clustering, Neural Networks.
* Reinforcement Learning [3] finds its root in Psychology. As in day to day life we come across many situations which forces us to take a some actions. After taking such actions we get either positive reward or negative reward. This is the same idea and thought process behind reinforcement learning. You train a agent such that based on certain state it’s present in, it takes few actions and gets rewards. Now based on what we want to do with those rewards is the main idea that is being explored nowadays. Reinforcement learning has started to gain recognition mainly in the field of robotics, supply chain management, self-driving cars, computer vision, finance and trading and much more.

So starting with reinforcement learning, the first ever idea that started was mainly embedded in psychological journals. Taking notes and inspiration from how humans react to different methods of learning. For example, if a child is running a race and he/she wins the race, the child is rewarded with a chocolate or a medal or any sort of appreciation. This is known as “Positive Reinforcement”. If a child is playing in a playground and he accidently touches thorny bush, he immediately starts crying because the thorn is causing a pain. This is an example of “Negative Reinforcement”.

The historical backdrop of reinforcement learning generally deals with two main ideologies. One ideology is concerns itself with learning by trial and error which started with the psychology of how animals learnt and responded. This idea runs along with some of the earlier works in the field of artificial intelligence which led to the resurrection of reinforcement learning in the early 1980s. The other ideology it concerns itself is with the problem of optimal control and whose solution is by generally found by using the approach of value functions and dynamic programming. For the most part, this idea did not inculcates learning. Although the two ideas have been quite independent, there are exceptions revolving around a third and less distinct idea concerned with temporal-difference methods. All the three ideologies culminated together in the late 1980s to produce what we know as today’s modern field of reinforcement learning.

The term "optimal control" came into existence in the late 1950s to describe the design problem dealing with a controller to which minimizes the measure of a dynamic system's behavior over time. One among many of the approaches to this problem was found in the mid-1950s by Richard Bellman and others which was just the extension of a nineteenth century theory proposed by Hamilton and Jacobi [4]. This particular approach used the concepts of a dynamic system's state and that of a value function, or in other words an "optimal return function," [4] to define a equation, now famously known as the “Bellman equation”. These types of methods that solve optimal control problems by solving the “Bellman equation” is widely to known as “dynamic programming”. Bellman also introduced the stochastic and discrete version of the optimal control problem known as Markovian decision processes (MDPs), and Ron Howard was the one who devised the policy iteration method for MDPs [14]. All of the idea discussed above are the most essential elements which underlie in the theory and algorithms of today’s modern day reinforcement learning.

Dynamic programming still to date is widely considered the only feasible way to solve general stochastic optimal control problems. Such optimal problems suffer from what Bellman identify as "the curse of dimensionality," which means that as the number of state variable in the given problem increased the computational requirement needed to actually compute the values of the variable increase exponentially. But even after these setbacks it is still way more efficient, accepted and more widely applicable than any other general method. Dynamic programming has a very extensive arc of development starting since the late 1950s, which also includes and extension to the new and interesting field of partially observable DPs.

Now again focusing back on the other major idea which currently leads us into the modern field of reinforcement learning revolves on the idea of learning from trial and error. This specific idea find its root in psychology, where "reinforcement" theories of learning are abundant and common. Perhaps the first one to express the essence of trial-and-error learning succinctly was Edward Thorndike, who in a nutshell said that any action followed by good (positive) or bad (negative) outcomes have a tendency to be reselected and changed accordingly. Thorndike called this phenomenon the "Law of Effect" because it described the effect of reinforcing events on the tendency to select actions.

Now understanding the third idea, dealing with the history of reinforcement learning, concerned with the idea of temporal-difference learning. Temporal-difference learning methods are very unique in the idea that they are driven by the difference between temporally consecutive estimates of the same quantity, like for example, of the probability of winning in the tic-tac-toe example. This approach is smaller and less distinct than the other two approaches, but it has played a particularly important role in the field, in part because temporal-difference methods seem to be quite new and unique to the field of reinforcement learning.

The origin dealing with that of temporal-difference learning can again be in part be attributed to animal learning psychology, in particular, in the concept of “*secondary reinforcers”*. A *secondary reinforcer* is a stimulus that is generally paired with a *primary reinforcer* such as food or pain and, due to this as a result, comes to take on interchangeable reinforcing properties. The temporal-difference and optimal control ideas were fully brought together in 1989 when Chris Watkins's developed Q-learning [7].

**Literature Survey**

1. **Components of Reinforcement Learning [4]**

After the brief introduction of reinforcement learning, there are a few key concepts that form an integral part of reinforcement learning ecosystem. For that here are the main or key components of a reinforcement learning algorithm:

1. **Agent [4]:** The main component of a reinforcement learning algorithm is agent. This is the entity that take actions in a particular environment. For example: An industrial robot picking up non-biodegradable items in a recycling plant.
2. **Environment [4]:** The surrounding in which the above mentioned agent will work or perform its activities. For example for an agent in computer vision the part of image enclosed by the bounding box can be the environment.
3. **State [4]:** This is the defined as the present status of the interacting agent in a particular environment. For example is the robot arm in motion or it is stationary or is it going up or is it going down.
4. **Action [4]:** Action is the steps the agent takes. More often than not in practical applications collection of actions called as action space is discrete in nature. For example the robotic arm moving left, right, up, down, close, open etc.
5. **Reward [4]:** This is defined as the objective that is implicitly or explicitly defined for the agent to take actions and maximize.

After discussing and going through a couple of main building blocks, there has to be components that bring these building blocks together because they alone can’t act or execute independently. So connection all the above components here are the binding concepts of Reinforcement Learning. They are:

1. **Policy [4]:** A *policy* defines learning the behavior of an agent and it’s way at a given instance of time. In other words, a policy can be defined as a mapping technique which maps the agent’s behavior from perceived states of the environment to actions it must take when it is present in one of those states. This can be correlated to psychological terms know as associations or stimulus-response rule. In some of the cases the policy can be a very simple function or can be a lookup table, whereas in others instances it can involve very in-depth computation, for example a search process. The policy is the core of a reinforcement learning agent meaning that it all by itself is sufficient enough to determine the behavior of agents. In general, policies can be stochastic.
2. **Reward Function [4]:** A *reward function* is the one which gives the definition of a goal in a reinforcement learning problem. In other words, it maps each state-action pair, in other words knows as perceived state, of the environment in which the agent is present to a single countable number called a *reward*, which indicates intrinsically how desirable a given state is. The sole purpose and objective of a reinforcement learning agent is to maximize the total reward it will receive in the long run of the experiment instead of short term gain. The reward function also defines what good or bad events are for the agent in terms of the reward it gains. In a biological system, this would only be appropriate to relate and identify biological rewards with pleasure or pain. These are the immediate and defining features of the problem the agent faces in an experimental setting. The reward function should mandatorily be unchanged by the agent. It can instead be use as be a basis to alter the agent’s policy. For example, if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. In general, reward functions can be stochastic.
3. **Value function [4]:** A *value function* is the one that tells the agent specifically about what is good for it in the long run. In other words we can say that the *value* of a particular state is the total amount of reward an agent expectedly accumulates over the future run of it’s experiment, starting from that specific state. Rewards are the parameters which helps the agent determine the states immediate intrinsic desirability, whereas values indicate the long-term desirability of states after considering all those states that are likely to come in future and the corresponding rewards that might be available in those states. Let’s say for example, a state might always earn a low immediate reward but still have a high value because the future states that follow the current state tends to earn quite high rewards or vice versa. Let’s take a human analogy. Rewards can be equated to pleasures if the reward is high and pain if reward is low, whereas values correspond to a more refined and futuristic judgment of how satisfied or dissatisfied we are that our environment is in a particular state.
4. **Model [4]:**  *Model* can be thought of as one that tries to copy the behavior of the environment. Let’s say for example, we know the state and the action, the model might be able to predict the expected next state and corresponding next reward. Models are generally used for *planning*, which decides on the course of action the agent must take by taking into consideration all the possible future situations before they are actually experienced. The amalgamation of models and planning in reinforcement learning systems is relatively a quite new development. Early reinforcement learning systems were straightforward trial and error learners, which is actually the total *opposite* of planning. Nonetheless, slowly and gradually it became clearer that reinforcement learning methods are quite closely related to dynamic programming methods which in-turn does use models and are in turn closely related to planning methods involving state and space. Modern reinforcement learning covers the whole spectrum starting from low-level trial and error learning to high-level deliberative planning.

Rewards are in a way primary parameters, whereas values as predictions of rewards can be thought of as secondary parameters. Without rewards there would not be any values and the one and only purpose of estimating the values is to anyhow achieve the most reward. Nevertheless it is values with one should be most concerned about when making and evaluating the decisions. Value judgements are the once based on which action choices are made. We try to seek actions that bring around the states of highest value but not highest reward mainly because these actions obtain the greatest amount of reward for the agent over the long run.

In planning and decision-making, the derived quantity is called value and it is the one with which is the main parameter. Unfortunately, it is easier to determine the rewards rather than determining values. Rewards are directly given by the environment but values are the one’s that needs to be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. As a matter of fact the most important component of almost all the reinforcement learning algorithms is a method on how to efficiently estimate values. The pivotal role of value estimation is arguably the most important thing that is learned about reinforcement learning over the past few decades. Although it’s not necessary to estimate value functions when solving a reinforcement learning problem. Like for example, search methods such as genetic algorithms or genetic programming or simulated annealing and much more, optimization methods have already been used to solve reinforcement learning problems. These methods search directly in the space of policies without ever engaging into value functions. We call these evolutionary methods for the reason because their operation is similar to the way biological evolution produces organisms with skilled behavior even though they have not learnt it during their individual lifetimes. If the policy space is sufficiently small or can somehow be structured so that good policies become common or easy to find then evolutionary methods can be very effective. Also in addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment.

Nonetheless, what it means is that reinforcement learning learns while interacting with the environment which evolutionary methods does not do. Evolutionary methods generally tends to ignore much of the useful structures of a reinforcement learning problem. Such methods do not use the fact that the policy being searched for is actually a function of states to actions and do not notice which states an agent passes through during its lifetime or which actions it selects. In some cases such information can be misleading (e.g., when states are misinterpreted).

1. **Markov Decision Process [4]**

****

**Fig. Agent-Environment Interaction in a Markov Decision Process [4]**

Markov Decision Process (MDPs) [4] in simple words are nothing more but a direct framing up of a learning problem from interactions to achieve a specific goal. The main entity or learner and also the decision maker is known as an *agent*. The thing *agent* interacts with every single time which comprises everything outside the *agent*, is called an *environment*. They both interact again and again with the agent selecting actions and the environment responding to these *actions* and presenting new situations before the *agent*. The *environment* also gives rise to *rewards*, which are nothing but some special numerical values that the *agent* tries to *maximize* over time through its choice of *actions*.

If the states and action spaces are finite, then the problem so formed is known as a finite markov decision process (fMDP). Finite MDPs are very important for reinforcement learning [3] problems and most of literatures out in the scholarly world have assumed that the environment is a finite MDP in their works.

Any reinforcement learning problem can be modeled as a Markov Decision Process [14]. Markov Decision Processes are a classic formulation of sequential decision making, where actions tend to influence not just immediate rewards, but also subsequent situations or states, and through those, future rewards [4].

MDPs involves delayed reward and puts on a heavy emphasis on the need to tradeoff immediate and delayed reward. Conversely in bandit problems the value *q\*(a)* of each action *a* is estimated, whereas in MDPs we try to estimate the value *q\*(s, a)* of each action *a* in each state *s*, or we tend to estimate the value *v\*(s)* of each state given optimal action preferences. These state-dependent quantities are very essential to accurately assign credit for long-term consequences to individual election of actions.

The agent and environment interact with each other at each of the discrete time steps denoted by *t* = 0, 1, 2, 3,… At each discrete time step *t*, the agent receives some sort of representational information of the environment’s current *state* given by *St ,* and on the basis of the state information it then selects an *action* given by *.* Now after one time step reflecting the consequence of an *action*, the agent receives a reward given by *Rt+1R*  and thus finds itself in a new state denoted by *St+1.* Thus all this information above gives rise to a sequence that looks like *S0,A0, R1, S1, A1, R2, S2, A2, R3,……..*[4]

Now in a *finite* MDP, the sets of states, actions, and rewards (*S,A,R)*  have all but finite number of elements in them. In such case, the random variables reward *Rt* and state *St* have well defined discrete probability distributions which is dependent only on the preceding state and corresponding action. Now for a particular value of these random variables, *s’* and *r*, there is a probability of those values occurring at a given time *t* for values particularly given that of the preceding states and corresponding action which can be given by the equation mentioned below:

[4]

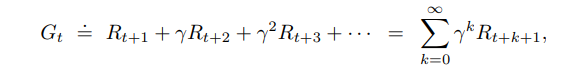
for all *s’, s S, r R,* and *a* *A(s).*

The function *p* here defines the dynamics or the measurements of the specific MDP. The dot over the equals sign in the equation conveys the information that it is a definition, particularly in this case, the function *p* rather than a fact that follows from given previous definitions. The dynamics function given by [0, 1] is an ordinary deterministic function consisting of four arguments. The ‘|’ is borrowed from the notation denoting conditional probability, but in this context it just reminds us that *p* specifies a probability distribution for each choice of *s* and *a,* that is given by below [4],



In a Markov decision process, the probabilities conveyed by *p* completely characterizes the environment’s dynamic nature which in other words can be understood as the probability of each possible value for *St* and *Rt* depending only on the immediate preceding state and action, *St−1* and *At−1*, and given both of the values not at all on all earlier states and/or actions. This can be viewed as a restriction on the state, but not on the decision process itself. The state generally includes all the necessary information about all aspects of all the previous agent-environment interactions which makes always makes a difference in the future. So if the all of the above conditions are met and followed, then the state is said to follow a *Markov Property.* [4]

Discount factors are associated with time horizons as given below [4].



The MDP framework is quite conceptual but as well as very flexible and can be applied to many different problems in varied ways. For example, the time steps are not needed to have fixed intervals of real time; they rather can have arbitrary consecutive stages of decision making and performing an action. The actions can be low-level controls, like the voltages applied to the motors of a robot arm, or high-level decisions, like whether or not to have lunch or to go to graduate school. Similar to actions, states can take a lot of diverse forms. They can either be completely determined by low-level sensations, like direct sensor readings, or they can be high-level and abstract in a sense as a symbolic descriptions of objects in a room.

The MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus are there and whatever objective one is trying to achieve. any problem of learning a goal-directed behavior can always be reduced to three signals passing to and fro between an agent and its environment: one signal will represent the choices made by the agent (i.e. actions), one signal to represent the criteria on which the choices are made (i.e. states), and one more signal to define the agent’s goal (i.e. rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable. Just for a fun example below is a typical workday of a person.

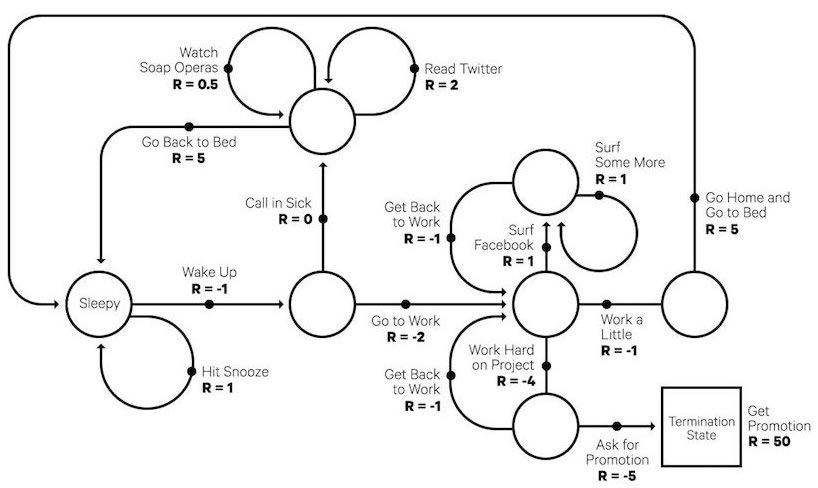
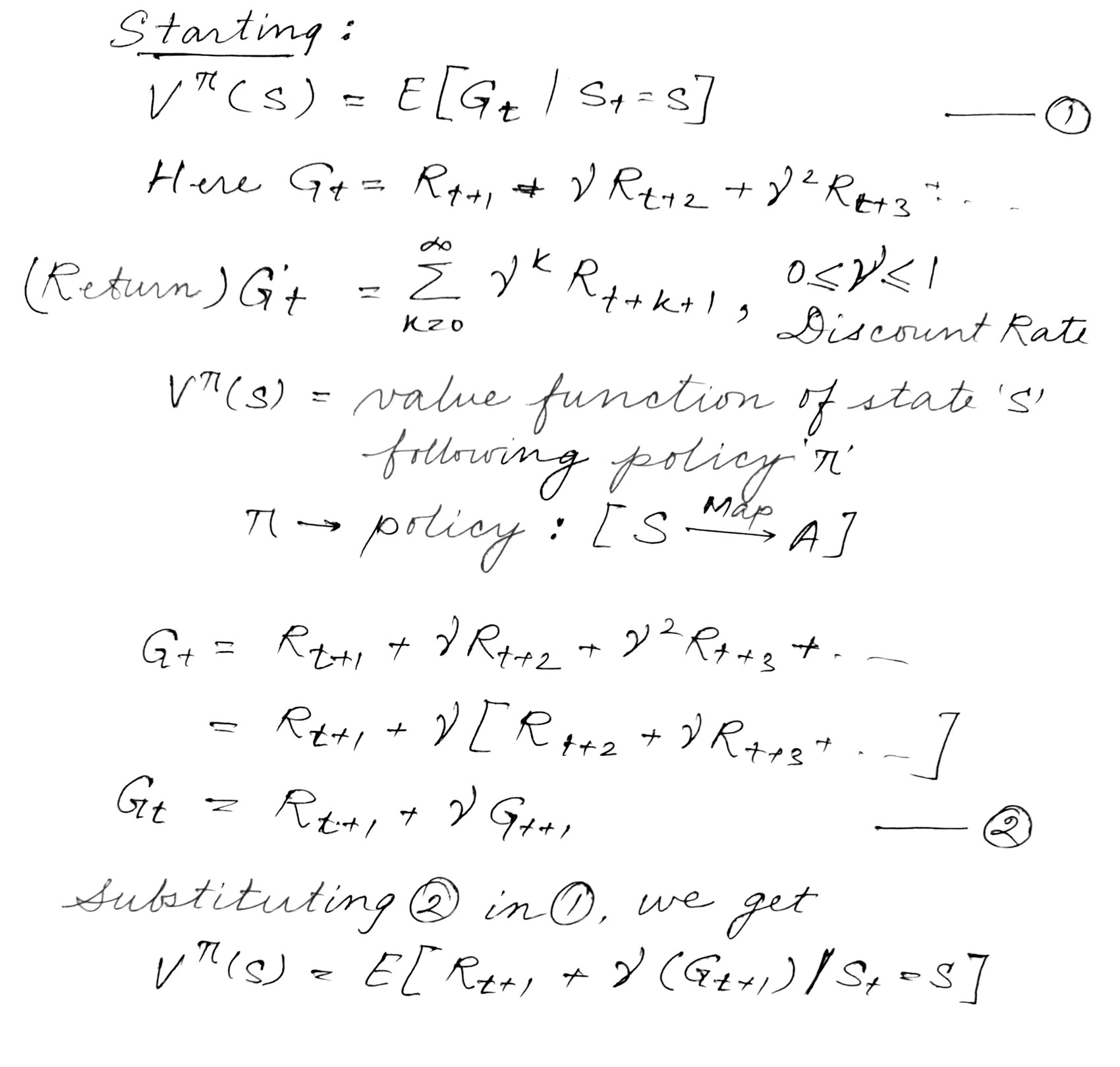
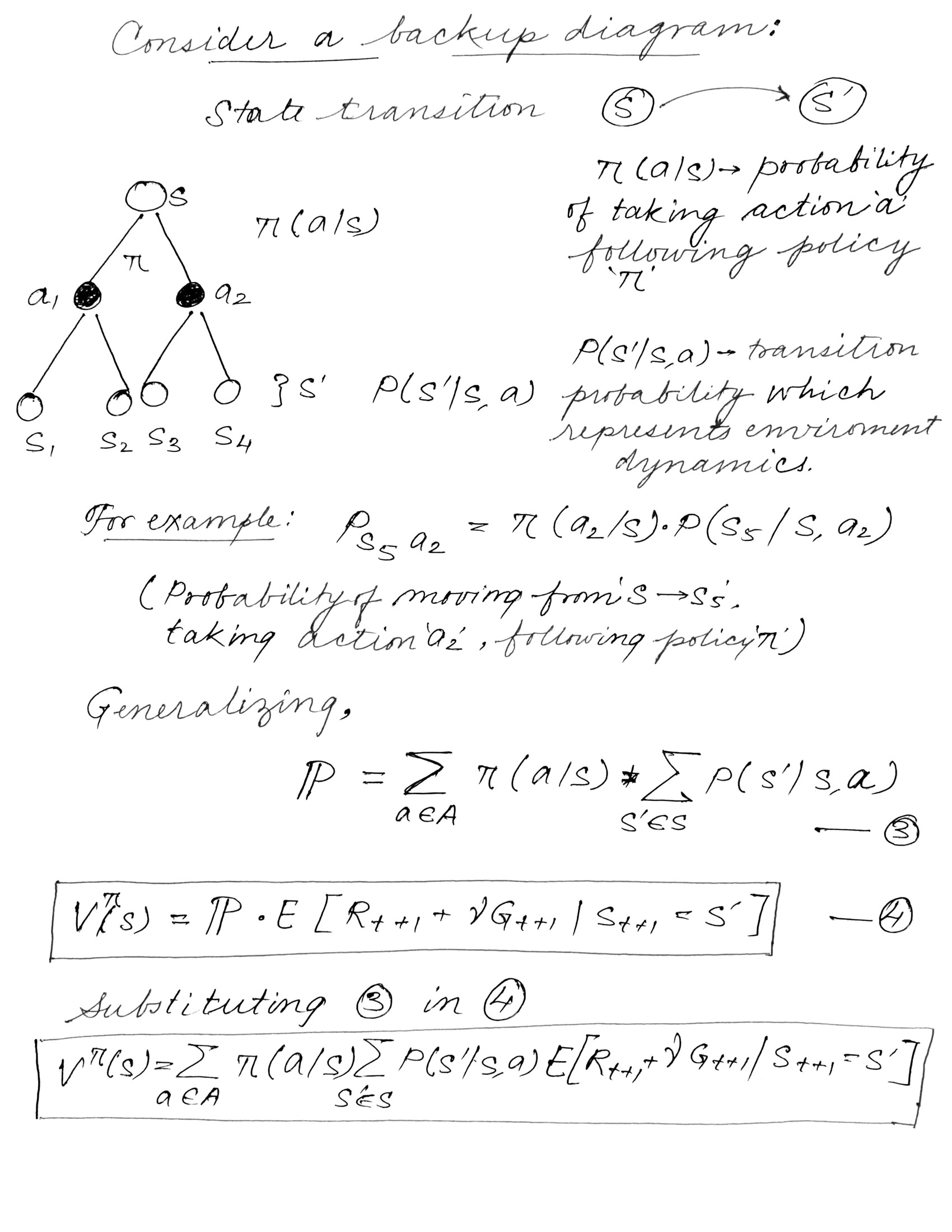


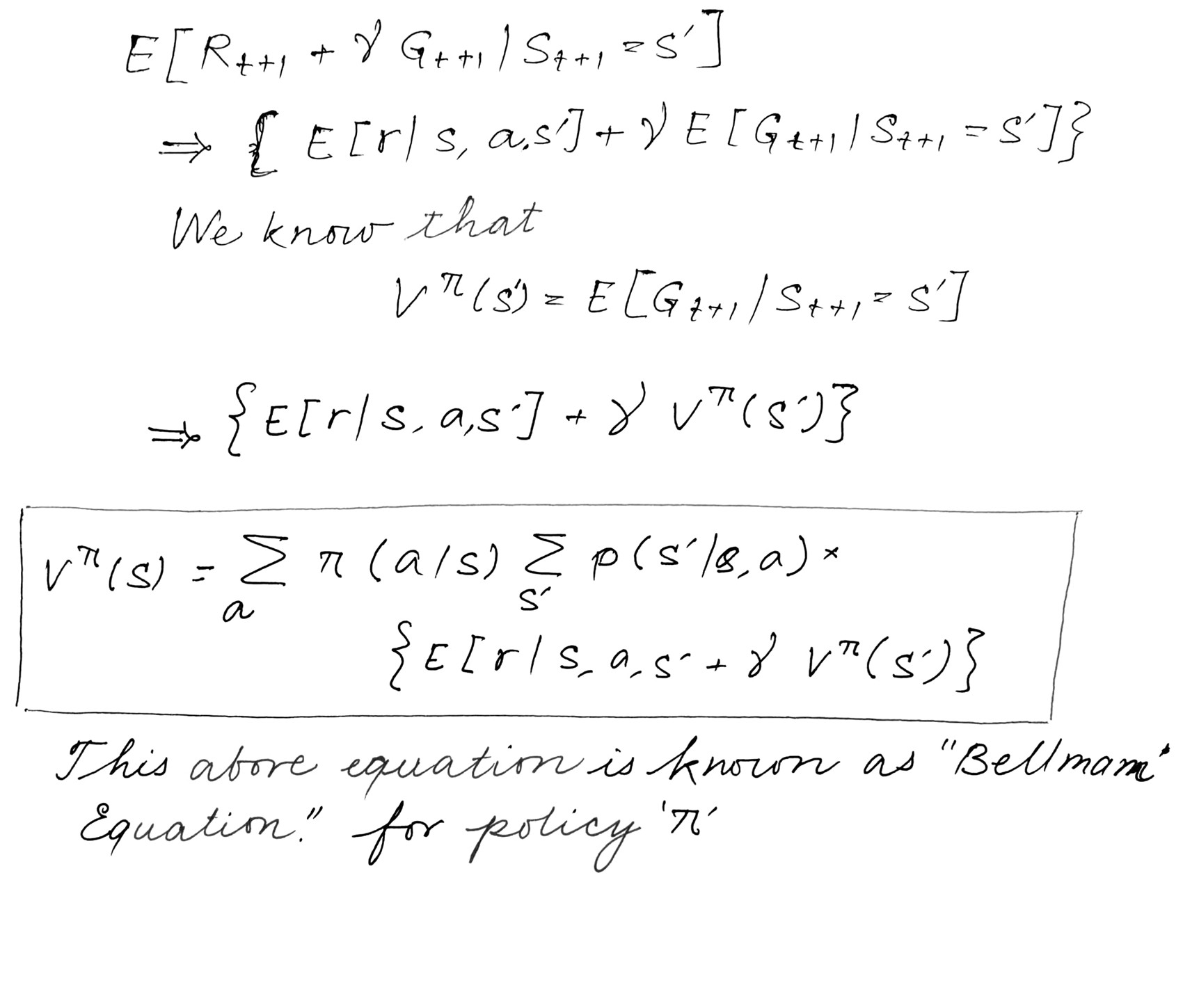
Fig. A simple Markov Decision Process (MDP) that represents a typical workday [16]

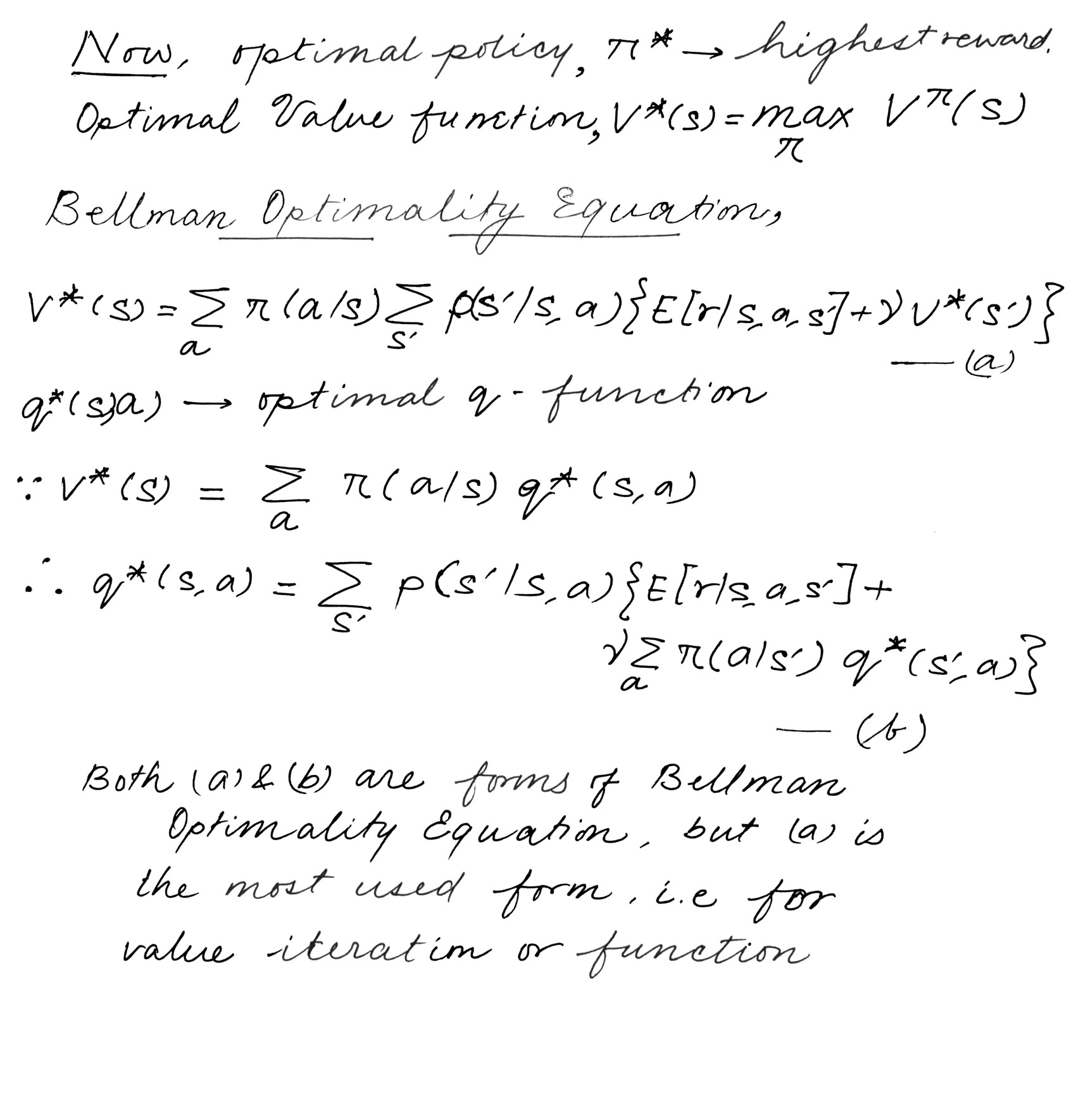
1. **Bellman Equations**

Bellman equations are the equations that help us find value functions and optimal policies. Now from the knowledge one knows that in an environment the policy is bound to change with different experience and correspondingly different policies are bound to have different value functions. Thus the value function which gives the maximum value compared to all the other value functions is called the optimal value function. Below is the derivations of the Bellman Equation [4]









1. **Model Based and Model Free RL Algorithm [5]:**

When going about different approaches to a RL problem, they can be broken broadly in 2 categories. Model-Free and Model-Based algorithms.

Model Free Algorithms that tend to learn through the experience gained from interactions with the environment, i.e. this algorithm tries to estimate the optimal policy without using or estimating the dynamics (transition and reward functions) of the environment.

Model Based Algorithms on the other hand is an approach that uses a learnt-model i.e. transition probabilities and reward function to predict the future action (optimal policy).

1. **Q-Learning [6] :**

Q-learning in its all glory and form was brought as a result of a specific variant of temporal difference learning (TD) initially a part of PhD Thesis by Watkins [6] and adopted & proposed by Watkins and Dayan [7].

Q-learning is a value-based model free learning algorithm. Value based algorithms updates the value function based on an equation (particularly Bellman equation). Whereas the other type, policy-based estimates the value function with a greedy policy obtained from the last policy improvement.

The ‘Q’ in Q-learning [7] stands for quality. Quality here represents how useful a given action is in gaining some future reward.

Speaking about reward, when considering rewards the following the reward tends to follow the same pattern as that of a MDP. A reward *Ra(sj,sk)* is obtained when taking action *ai* in state *sj* and the environment/system changes to state *sk* after the decision maker takes action *ai*. The decision maker follows a policy, π such that π(⋅):S→A, that for each state *sj∈ S* takes an action *ai ∈ A*. So that the policy is what tells the decision maker which actions to take in each state. The policy π may be randomized as well. Longer time horizons have much more **variance** as they include more irrelevant information, while short time horizons are biased towards only short-term gains. The discounted reward phenomenon tends to make an infinite series finite. Thus aiming to maximize the long term reward instead of short term rewards. The discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future. If γ=0, the agent will only learn about actions that produce an immediate reward. If γ=1, the agent will evaluate each of its actions based on the sum total of all of its future rewards.

1. **Bounding Box**

Bounding box in it’s heart and soul is mostly just a rectangle that surround an object present in an image and tries to specify mainly it’s position in the image, it’s class and maybe confidence (probability of how likely that object is in that location). Bounding box are one of the most prominent techniques which draw the attention mainly when it comes to object detection and localization. They are generally used in the task of object detections where the main aim is to identify the position and type of maybe multiple objects in the image. A example’s ensues this line.



Fig. A Bounding-Box example

Now coming to the conventions used when specifying a bounding box are:

1. To specify the box with respect to the respective coordinates of it’s top left and bottom right points.



1. To specify the box with respect to it’s center, height and width.



Now below are the list of parameters that are generally used to define the specifications of the bounding boxes:

1. Class: The first and foremost parameter is denoting the class to which a specific object in an image belongs to. For example car, person, truck, aeroplane etc.
2. (x1, y1): This corresponds to the x and y co-ordinates of the top left corner of the given rectangular box also can be written as xmin and ymin.
3. (x2, y2): This corresponds to the x and y co-ordinates of the bottom right corner of the rectangle also written as xmax and ymax.
4. (xc, yc): This corresponds to the x and y co-ordinates of the center of the bounding box.
5. Width: This represents the width of the bounding box.
6. Height: This represents the height of the bounding box.
7. Confidence: This indicates the probability of the presence of the particular class of object present in that image. For example a confidence of 0.8 indicates that there is a 80% chance that the particular class of object actually exists in that box.
8. **Resnet50[11]**

Deep residual networks have become popular like the infamous ResNet-50[11] model is a convolutional neural network (CNN) that is 50 layers deep. A residual neural network (ResNet) is a special types of artificial neural network (ANN) which tends to stack residual blocks one on the top of other to form a network.

In recent years, the fascinating field of computer vision has undergone very far-reaching transformations due to the introduction of new technologies. As a direct result due to these advancements, it has become possible for computer vision models to actually surpass humans in very efficiently solving different problems dealing with image recognition, object detection, face recognition, image classification and much more.

With this regard, the introduction of deep convolutional neural networks or CNNs deserve a special mention. These neural networks have been extensively used for analyzing visual imagery with excellent accuracy.

Even though it gives us the option of adding more layers to the CNNs might help us solve more complicated tasks in computer vision, it comes with its own set of issues and problems. It is observed most of the time that training the neural networks becomes more difficult with the increase in the number of added layers, and in even in some cases, the accuracy goes down quite a lot.

It is to address this issue that the introduction and use of ResNet[11] become of an utmost importance. Deeper neural networks are quite difficult to train. But with ResNet[11], it becomes possible to surpass the difficulties of training very deep neural networks containing 100’s of layers.

Now what is ResNet? ResNet[11] is an acronym for Residual Network. It is an innovative neural network that was first introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 computer vision research paper titled ‘Deep Residual Learning for Image Recognition’[11].

The reason of this model to be immensely successful, can be understood from the fact is that the ensemble of this won the top position at the ILSVRC 2015 classification competition with an error of only 3.57%. Additionally, it also came first in the ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in the ILSVRC & COCO competitions of 2015.

ResNet in itself has many variants that run on the same concept but have varying numbers of layers embedded in it. Resnet50 is one of the most influential networks where the number ‘50’ is used to denote the variant that can work with 50 neural network layers.

When working with deep convolutional neural networks to solve a problem related to computer vision, machine learning experts engages in stacking more layers. These additional layers help solve complex problems more efficiently as the different layers could be trained for varying tasks to get highly accurate results.

While the number of stacked layers can enrich the features of the model, a deeper network has the possibility to show the issue of degradation. In other words, as the number of layers of the neural network increases, the accuracy levels may get saturated and slowly degrade after a point. As a result, the performance of the model goes down on both of the training and testing data.

Now from the common notion we might think that this degradation is a result of overfitting, but it is not the case. Instead, it might result from the initialization of the networks, different optimization functions and more importantly, the known problem of vanishing or exploding gradients.

ResNet[11] was created with the intent of tackling this exact problem of vanishing and exploding gradients. Deep residual nets makes use of residual blocks to improve the accuracy of the models. The concept of “skip connections,” which lies at the core of the residual blocks, is the strength of this type of neural network.

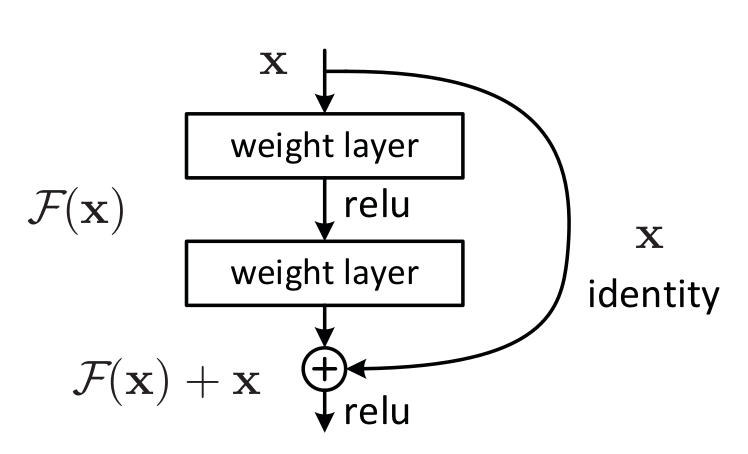


Fig. Skip Connection as mentioned in “Deep Residual Learning for Image Recognition”[11]

Now why skip connections? These skip connections work in two ways. Firstly, they all mitigate the issue of vanishing gradient by setting up an alternative shortcut for the gradient to reach through. In addition, they enable the model to learn an identity function. The aim of this identity function is to ensure that the higher layers of the model do not perform any worse than the lower layers.

Thus short, the residual blocks make it considerably easier for the layers to learn identity functions. As a result, ResNet improves the efficiency of deep neural networks with more neural layers while having the ability to minimize the percentage of errors. In other words we can say that, the skip connections add the outputs from previous layers to the outputs of stacked layers, making it possible to train much deeper networks than previously possible.

The very first ResNet[11] architecture was the Resnet-34 which was initially mentioned in the research paper first introducing ResNet[11], which involved the insertion of shortcut connections in turning a plain network into its residual network counterpart. In this case, the plain neural network was inspired by VGG[16] neural networks (VGG-16, VGG-19), with the convolutional networks having 3×3 filters. However, compared to VGGNets[16], ResNets[11] have fewer filters and comparatively lower complexity. The 34-layer ResNet[11] achieves a performance of 3.6 bn FLOPs, compared to 1.8bn FLOPs of smaller 18-layer ResNets[11].

It also followed two very simple rules of design – the layers had the same number of filters for the same output feature map size, and the number of filters doubled in case the feature map size was halved in order to preserve the time complexity per layer. It consisted of 34 weighted layers.

The shortcut connections were then added to this simple neural network. While the input and output dimensions were kept the same, the identity shortcuts were directly used. With the supposedly increase in the dimensions, two options were needed to be considered. The first option was that the shortcut would still perform identity mapping while padding extra zero entries to increase the dimensions. The other option was to match dimensions using projection shortcut. Below is the image of the ResNet[11] architecture.

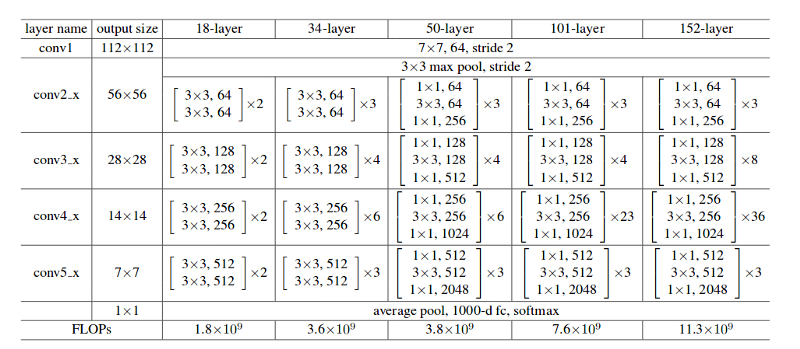


Fig. The different layer architectures of ResNet[11] variants

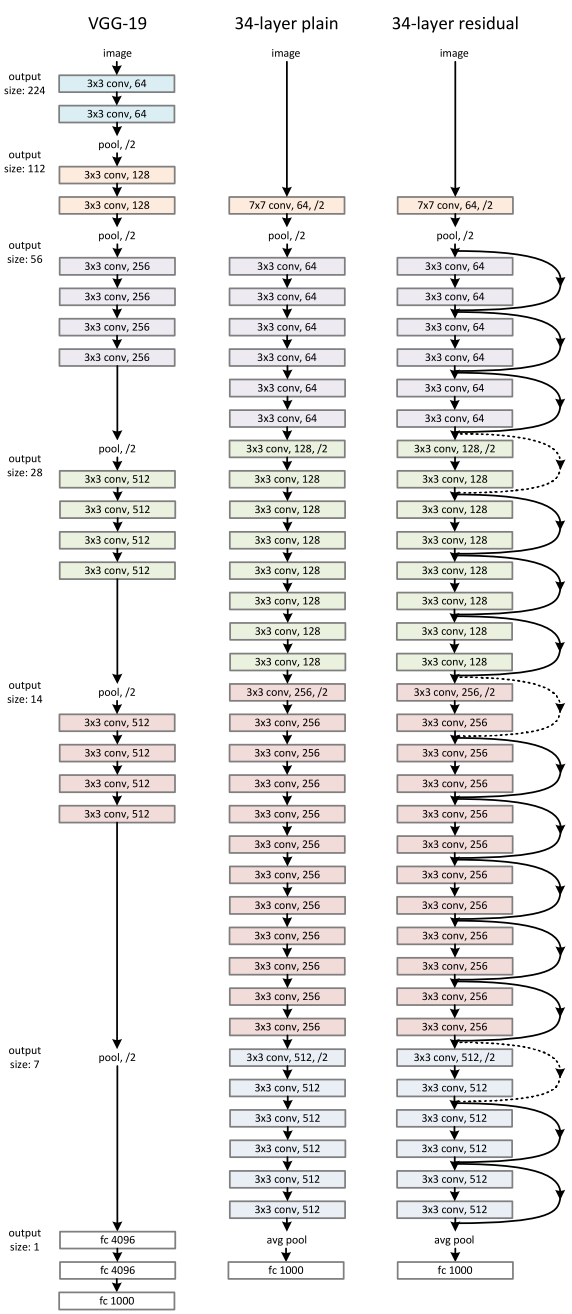


Fig. ResNet Architecture[11]

1. **Deep Q Network (DQN) [8]:**

One of the breakthrough paper on implementation of deep learning for reinforcement learning can be credited to researchers from Deepmind for publishing their paper “Human-level control through deep reinforcement learning” [8] in 2015 which they trained agents on multiple Atari games which the only input being the screen on the game. This laid the foundation for the merging together of two fields Reinforcement Leaning and Deep Learning and birthed a new field of Deep Reinforcement Learning or DRL. Every time the agent makes a move in the environment, in this case the images of the game, it created a tuple of 4 variables called experience which consists current state, action it performed, the state it landed in and the reward it got for performing that operation *(s, a, st+1, r)* and then store these experiences by creating it’s own data set in a replay memory. *Replay Memory* sounds same as you might think. It stores all the past experiences of the RL agent and then replays it again and again and learns from it. Now here the Q values from the Q Learning comes into picture helping the agent to take future action.



Fig. Network structure of Deep Q-Network (DQN), where Q-values Q(s, a) are generated

for all actions for a given state [8]

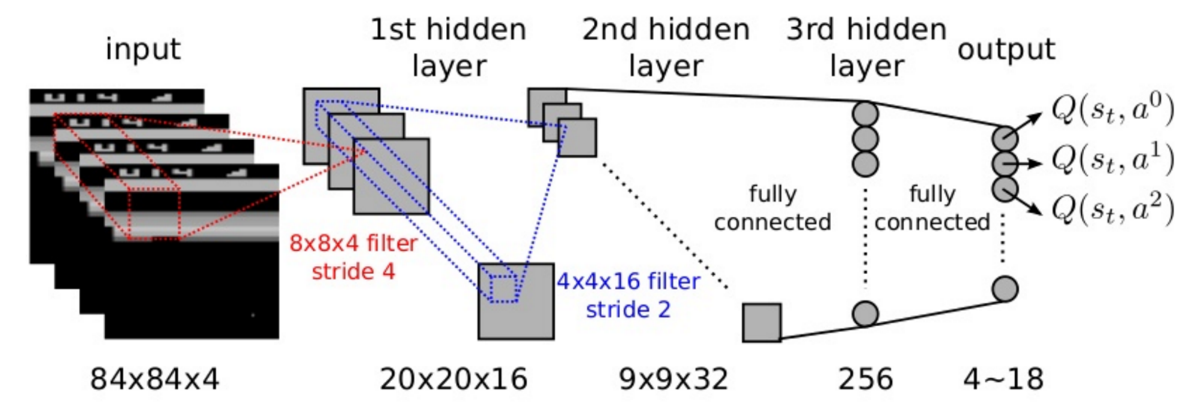


Fig. Exploded View of the above DQN [9]

1. **YOLO - “‘You Only Look Once”[19]**

**Problem Statement**

The base paper that I am aiming to work on and improve is titled “BAR- A Reinforcement Learning Agent for Bounding-Box Automated Refinement” [10]. The paper starts with the brief introduction of how deep learning has basically taken over the world due to its wide ranging applications mainly in the field of computer vision. Object detection and recognition deep learning techniques allowing machines to visualize their environment with pretty high accuracy. Thus we are seeing a high number of industries using computer vision in their day to day operations be it inventory management, process control, quality control and much more. But there is a drawback, deep learning still being a mix of supervised and unsupervised learning needs amount of labeled data with a high degree of accuracy which takes considerable amount of effort, time and money. Image annotations also are highly prone to human error and it the might happen that the images being biased towards one side, the human annotator might label the image wrongly. Even thought the whole dataset is labeled, it might happen that the Region of Interest (RoI) of the target object might change because of industrial circumstances. The cost of re-labelling is just not feasible. To the best of knowledge existing literature tends to generate bounding boxes around the target object but no attempts have been made to correct these bounding boxes. In summary, existing work focuses on reducing the time spent by human annotators on manual labeling. While, in the literature, this goal is achieved by finding alternative ways to generate b-boxes proposals, our work focuses on learning to correct inaccurately generated b-boxes to later refine annotations regardless of their initialization [10].

Every image contains exactly one annotated target object whose b-box is represented by its upper-left corner (*xmin, ymin*) and its lower-right corner (*xmax, ymax*). This b-box is considered inaccurate if its IoU with the groundtruth is below a threshold, denoted by *β*. Given an image and an inaccurate b-box enclosing the target object, the goal of the agent is to correct the b-box as shown in figure here. The agent achieves this goal by executing a series of actions that modify the position and aspect-ratio of the b-box. This series of actions corresponds to an episode that ends with the final correction of the agent [11]. At time step *t*, the agent updates its Q-value estimate in the following manner:

Fig. BAR agent workflow during the testing phase.

Given an image and an inaccurate b-box enclosing the target object, BAR chooses the path *TE* with *T* = *{up,up,left}* [11]

*Qt*+1(*s, a*) = (*α −* 1)*Qt*(*s, a*) + *α*(*r* + *γ* max*a’ Qt*(*s’, a’*)) [10]

The three main components of BAR-DRL are:

1. State: The state is composed of a feature vector *∈* R1238 extracted using ResNet50 [11] pre-trained on ImageNet [12] and a history vector. The b-box enclosed region is resized to 224*×*224, then fed to the feature extractor that outputs a vector of size 512. The history vector encodes the 10 actions of the episode, each of which is represented as a one hot encoder.
2. Actions: These are the eight translation actions shown in table below and the *stop* action.



1. Reward: The reward for a translation action *a* at step *t* is:



A higher negative value is necessary when the IoU decreases to prevent the agent from worsening the initial b-box, which defeats the purpose of a correcting agent. The reward for the *stop* action is:



where: Γ =, *r*1 = 6, *c* = 4, *r*2 = 3 [10].

**Problem Statement:** Most of the literature tends to create bounding boxes, be it for single objects or be it for multiple objects. The approach in this paper can be extended to multiple objects. But what happens if there are overlapping objects in the image and the methodology fails to correctly identify and create the bounding boxes in the image. This sort of application can be huge for the industrial applications and the scenarios where it’s bound to be a lot of overlapping objects. So to formally define my problem statement **“Bounding Box Refinement Agent for Overlapping Object Detection”**.

**Methodology**

For simulation of the paper I am working on Pycharm with Tensorflow 2.7, Python 3.9, cuDNN 8.2.1 and cuda 11.5 on my laptop having 8GB RAM and Nvidia GTX1650 GDDR6 having 4 GB of VRAM. Since the authors used a private dataset, I have instead used PASCAL VOC 2007 [13].

For execution, multiple experiments involving images of “aeroplane” class was chosen from PASCALVOC 2007 [13]. Since the aim of my experiment is to reduce the manual effort of correcting the bounding boxes by the proposal produced by the reinforcement learning agent below will contain the table and list of all the experimental parameters that I created for my own testing. Wrong manual annotations where done on these 200 images and below contains the IoU of those images for different epochs starting from 50, 80,100 and thresholds varying from 0.50 to 0.80. The parameter for the metric used is Average IoU when testing on images.

The deep q neural network consists of two fully-connected layers of 500 neurons each with ReLu activation and random normal initialization, and an output layer of 9 neurons with linear activation. Mean square error loss with Adam optimizer and learning rate of 0*.*001 are used, and the discount factor *γ* for the Q-function is set to 0*.*90 [10].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Image Name | Epochs | Threshold | Initial IoU | Final IoU | Max IoU |
| 009365.jpg | 50 | 50 | 0.36 | 0.33 | 0.4 |
|  | 80 | 50 | 0.36 | 0.33 | 0.4 |
|  | 100 | 50 | 0.36 | 0.33 | 0.4 |
|  | 50 | 70 | 0.36 | 0.33 | 0.4 |
|  | 80 | 70 | 0.36 | 0.33 | 0.4 |
|  | 100 | 70 | 0.36 | 0.36 | 0.4 |
|  | 50 | 75 | 0.36 | 0.33 | 0.4 |
|  | 80 | 75 | 0.36 | 0.33 | 0.4 |
|  | 100 | 75 | 0.36 | 0.48 | 0.48 |
|  | 50 | 80 | 0.36 | 0.33 | 0.4 |
|  | 80 | 80 | 0.36 | 0.33 | 0.4 |
|  | 100 | 80 | 0.36 | 0.33 | 0.4 |
|  |  |  |  |  |  |
| 009461.jpg | 50 | 50 | 0 | 0 | 0 |
|  | 80 | 50 | 0 | 0.03 | 0.03 |
|  | 100 | 50 | 0 | 0.03 | 0.03 |
|  | 50 | 70 | 0 | 0.03 | 0.03 |
|  | 80 | 70 | 0 | 0.01 | 0.01 |
|  | 100 | 70 | 0 | 0 | 0.01 |
|  | 50 | 75 | 0 | 0 | 0 |
|  | 80 | 75 | 0 | 0.03 | 0.03 |
|  | 100 | 75 | 0 | 0.05 | 0.05 |
|  | 50 | 80 | 0 | 0.03 | 0.03 |
|  | 80 | 80 | 0 | 0.03 | 0.03 |
|  | 100 | 80 | 0 | 0.03 | 0.03 |
|  |  |  |  |  |  |
| 009480.jpg | 50 | 50 | 0.58 | 0.73 | 0.76 |
|  | 80 | 50 | 0.58 | 0.56 | 0.6 |
|  | 100 | 50 | 0.58 | 0.65 | 0.74 |
|  | 50 | 70 | 0.58 | 0.48 | 0.6 |
|  | 80 | 70 | 0.58 | 0.73 | 0.76 |
|  | 100 | 70 | 0.58 | 0.53 | 0.6 |
|  | 50 | 75 | 0.58 | 0.73 | 0.76 |
|  | 80 | 75 | 0.58 | 0.48 | 0.6 |
|  | 100 | 75 | 0.58 | 0.51 | 0.6 |
|  | 50 | 80 | 0.58 | 0.58 | 0.58 |
|  | 80 | 80 | 0.58 | 0.54 | 0.6 |
|  | 100 | 80 | 0.58 | 0.48 | 0.6 |
|  |  |  |  |  |  |
| 009615.jpg | 50 | 50 | 0.91 | 0.56 | 0.91 |
|  | 80 | 50 | 0.91 | 0.61 | 0.93 |
|  | 100 | 50 | 0.91 | 0.56 | 0.91 |
|  | 50 | 70 | 0.91 | 0.9 | 0.93 |
|  | 80 | 70 | 0.91 | 0.82 | 0.91 |
|  | 100 | 70 | 0.91 | 0.61 | 0.93 |
|  | 50 | 75 | 0.91 | 0.44 | 0.93 |
|  | 80 | 75 | 0.91 | 0.82 | 0.93 |
|  | 100 | 75 | 0.91 | 0.61 | 0.93 |
|  | 50 | 80 | 0.91 | 0.9 | 0.93 |
|  | 80 | 80 | 0.91 | 0.93 | 0.93 |
|  | 100 | 80 | 0.91 | 0.75 | 0.93 |
|  |  |  |  |  |  |
| 009702.jpg | 50 | 50 | 0.04 | 0.02 | 0.04 |
|  | 80 | 50 | 0.04 | 0.03 | 0.04 |
|  | 100 | 50 | 0.04 | 0.03 | 0.04 |
|  | 50 | 70 | 0.04 | 0.03 | 0.04 |
|  | 80 | 70 | 0.04 | 0.02 | 0.04 |
|  | 100 | 70 | 0.04 | 0.05 | 0.05 |
|  | 50 | 75 | 0.04 | 0.02 | 0.04 |
|  | 80 | 75 | 0.04 | 0.03 | 0.04 |
|  | 100 | 75 | 0.04 | 0.05 | 0.05 |
|  | 50 | 80 | 0.04 | 0.03 | 0.04 |
|  | 80 | 80 | 0.04 | 0.03 | 0.04 |
|  | 100 | 80 | 0.04 | 0.03 | 0.04 |

**Future Work**

As for future work to be done , one of the work will be increasing the dataset size, involving more images and train the DRL agent on more than one class of object and analyzing the parameter tuning to improve the value on the final & average IoU. One more area of possible work is building a different neural network for actually extracting the features from images.

**References**

[1] Nasteski, Vladimir. "An overview of the supervised machine learning methods." *Horizons. b* 4 (2017): 51-62.

[2] Celebi, M. Emre, and Kemal Aydin, eds. *Unsupervised learning algorithms*. Berlin: Springer International Publishing, 2016.

[3] Sutton, Richard S., and Andrew G. Barto, "Reinforcement learning", *Journal of Cognitive Neuroscience* 11.1 (1999): 126-134.

[4] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Le, N., Rathour, V. S., Yamazaki, K., Luu, K., & Savvides, M. (2021). Deep reinforcement learning in computer vision: a comprehensive survey. *Artificial Intelligence Review*, 1-87.

[6] Watkins, C. J. C. H. (1989). Learning from delayed rewards, PhD Thesis

[7] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292,

1992.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[9] Artificial Intelligence, Leonardo Araujo dos Santos, Independent Textbook

[10] Ayle, M., Tekli, J., El-Zini, J., El-Asmar, B., & Awad, M. (2020). BAR — A Reinforcement Learning Agent for Bounding-Box Automated Refinement. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(03), 2561-2568

[11] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[12] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255

[13] Everingham, M., Van Gool, L., Williams, C.K.I. *et al.* The PASCAL Visual Object Classes (VOC) Challenge. *Int J Comput Vis* **88,**303–338 (2010).

[14] Howard, Ronald A. "Dynamic programming and markov processes.", Technology Press of Massachusetts Institute of Technology, (1960).

[15] Blog Post Series by Jake Bennett, (<https://randomant.net/reinforcement-learning-concepts/>)

[16] Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[17] Lee, S., Kwak, S. and Cho, M., 2018, December. Universal bounding box regression and its applications. In *Asian Conference on Computer Vision* (pp. 373-387). Springer, Cham.

[18] Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

[19] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).