Chapter 1 – part 1

**Artificial Intelligence:**

**Fundamentals of**

**REINFORCEMENT Learning,**

**Introduction to Q-learning**

Reinforcement learning

Bellman Equation, Plan, Markov Decision Process

Policy vs Plan, Penalty

Q-Learning, Temporal Difference

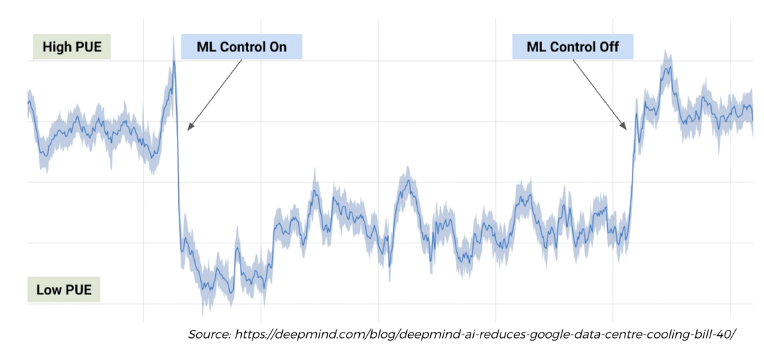
**1.1 AI usage**

AI is used in

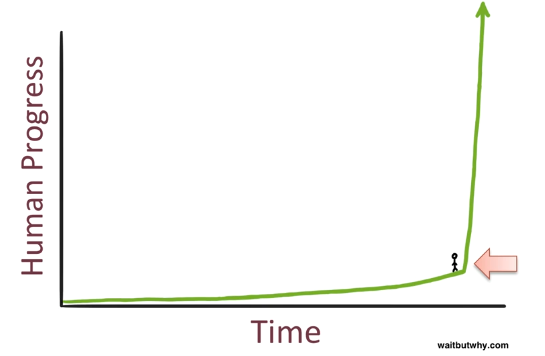
1. Self-driving cars
2. Medicine
3. Heavy machinery.
4. Customer service

|  |  |
| --- | --- |
| * So why AI is so popular right now? The answer is hidden inside Moore's Law. * It says is that the **Power** of the **Average Computer** will double every 18 to 24 months. * We've already surpassed the brain of an *insect* and a *mouse*. By 2025 the power of a computer will be equivalent to human *Brain*. * It's exponentially growing. And so this is a great time to be in *artificial intelligence*. |  |

|  |  |
| --- | --- |
| * Early 70's popular AI program was Deep-Mind, was the first computer to beat Gary Kasparov in chess. * Or in recent years much more complex game "Game of GO" a *computer-program* called **Alpha GO** beats 18 times world champion. * So something wild is happening right now. * So in our case we'll use Games to train AI. * Because games are a confined environment where if we can beat a game using an Artificial Intelligence then we can use the same Principles to apply to Business. * For Example, the team at **Google Deep Mind** who created **Alpha GO**, they applied Artificial Intelligence to Google's warehouse to control the *air cooling*. * And what they found is that the electricity bill/consumption was reduced by AI. And they managed to save 40% of their bill. |  |



* So applying AI in games and applying it to business is very closely related.



**1.2 Course Overview**

1. **Fundamentals Of Reinforcement Learning**

* What is reinforcement learning
* The Bellman Equation
* The Plan
* Markov Decision Process
* Policy vs Plan
* Adding a Living Penalty
* Q-Learning
* Temporal Difference
* Q-Learning Visualization

The Q-learning intuition section lays the foundations for the next chapters. We'll grasp the concepts of reinforcement learning.

1. **Deep Q-Learning: Self Driving car**

* Learning
* Acting
* Experience Replay
* Action Selection Policies

Next, we'll build the model inside a two dimensional self-driving car which will need to go from point A to Point B avoiding certain obstacles. Here we'll talk about the Deep-Q Learning and then we proceed to its practical part.

1. **Deep Convolutional Q-Learning: "DOOM" Game playing AI :: (3h 23 min)**

* Deep Convolutional Q-Learning
* Eligibility Trace

Next part is Deep Convolutioinal Q-Learning. Here we add eyes to our model, these kind of CNN will allows the algorithm to see what's happening on the screen rather than having access to the backend.

Our AI will learn to play the Game of Doom (a game from the 90s).

1. **AC3 : "Breakdown" Game playing AI**

* The three A's in A3C
* Actor-Critic
* Asynchronous
* Advantage
* LSTM Layer

Finally at part 4, we'll be using A-3C model, the most powerful AI algorithm today, we'll teach it to play the Breakout game.

It is more challenging than the game of doom.

**1.3 Plan of Attack**

1. What *reinforcement learning* actually is and what the *philosophy* behind *reinforcement* *learning*.

* How reinforcement learning actually can be seen in real life and how it relates to things that we observe in real life.

1. We'll talk about the Bellman Equation very fundamental concept to understand ***reinforcement learning*** and ***Q-learning***.
2. Then we'll talk about the **plan** , it is used in AI in order to navigate inside environments.
3. After that we'll talk about Markov Decision Process (MDP) and it adds a layer of sophistication to our Belman equation (to our whole reinforcement learning, to our Q learning concepts).
4. We will talk about Policies Vs Plans. It's a quick tutorial on how **policy** is different from **plans** .
5. Next we'll talk about adding a Living Penalty to our environments. It's kind of another way of adding *complexity* into the *environments* that our agents are going to be operating in.
6. We'll talk about the **intuition** behind **Q-learning**. So up until that section we're going to be talking **values of states**. And then we're going to talk about Values Of Actions or Q-values.
7. Next, we're going to introduce the Temporal Difference.

* It's where everything that we've learned is going to come together. We'll explain how exactly do agents or artificial intelligence learn (how does it update its values) through all the iterative process that is going through.

1. Finally we'll talk about the visualization of Q-learning. We're going to take everything we've learn and going to look how an artificial intelligence actually perform Q-learning and do all the things that we're going to discuss on an intuitive level.

**1.4 What is reinforcement learning**

Before we get started with creating super-powerful AI for beating challenges like Self-Driving Cars, Doom and Breakout, we need to first understand on an intuitive level the key concepts that go into reinforcement learning.

* Intrinsic And Extrinsic Rewards: The thing is we already know how we, humans, learn. We understand the concepts of ***intrinsic*** and ***extrinsic*** ***rewards*** that guide us to become better at things.
* For example, if you're playing bowling - you know that you need to hit the pins and a strike is a perfect shot. We know this because **we are rewarded with points** for **hitting** the **pins** down and we are **"punished"** with no points when your ball ends in the ditch.
* So your brain projects those conditions of the environment onto your *actions* and that's how you know when you are *doing good* and when - not so much. And that's how we learn.
* But how do we explain that to an AI?
* That's exactly what we will be covering off in this Section.
* We will focus on a type of reinforcement learning called Q-Learning. We will cover off many aspects of Q-Learning which will altogether ensure that when we are actually doing the practical tutorials and coding the AI - we understand what is happening in the background.

|  |  |
| --- | --- |
| * Environment: Here, we've got a little maze. This maze is our *representation* of an ***environment*** . * We're going to be dealing with *certain environments* in which our *artificial intelligence* is going to perform. It's going to be taking actions, to beat/win in these environments. * Agent: Also we've got an agent, the agent is our artificial intelligence. * This agent going to be navigating these environments and learning from the feedback that are given to perform certain actions. * The ***agent*** perform certain ***actions*** in this environment. * And as a result the state in which the agent is in will change (the state is going to change because of the action). Also a *reward* will be given to the *agent*. |  |

* Note that, sometimes it might happen that it the *action* *won't change* a *state* or there *won't be a reward* for taking that *action*.
* But nevertheless the agent's going to be taking ***actions***, changing the ***state***, getting ***rewards*** (then it repeats again).
* By doing that process it's *learning* about *what is good for getting good rewards*. Its going to *explore* the *environment*, understanding what *actions* lead to *good rewards* and *favorable states* and what *actions* leads to *bad rewards* and *unfavorable state*.
* Above is a simplistic representation. In general the *environments* actually don't have to be just *mazes* (about getting out of a maze or finding a treasure in a maze).
* An *environment* can be pretty much *anything in life*.
* So imagine you *waking* *up* in the *morning* and cooking an *omelet*. So in order to *make* that *omelet* you need to go through *certain steps*:
* You need to get the salt
* Get the eggs
* Get the frying pans
* Turn the fire on and so on.
* It's an *environment* where you're performing certain *actions*, and these *actions* are taking in *certain states* and those actions lead to *certain* other *states* and sometimes *reward* (well coked omelet).
* For instance if you **heat** the **frying pan too much**, the **omelet** will **burn** (-ve reward).
* With *proper heat you'll get well coked* omelet (+ve reward).
* Putting salt before the omelet gives –ve reward and putting salt after the omelet gets +ve reward.
* Now you will **remember in which state and action you got the +ve reward**. It's important to remember that.
* If you take *all* the *correct actions* in the *correct order* in the *correct states* your final reward could be an omelet which you can eat.

So it's may be a very *basic* *activity* in your *life* but if you think about it, it's actually *an environment* and you are the *agent* that going through this *environment* and perform a *task*.

Remember your *first omelet* that you've *made*, probably you *failed* at first *attempt*. But you will learn from that because you will understand what actions lead to what states and rewards.

* Driving a car is also an environment where you can turn the steering wheel you can accelerate you can break and so on and you're getting feedback from the environment.
* For example, one of those feedbacks is the policeman giving you a speeding fine, if you're going above the *acceptable or allowed speed* limit on that highway.
* And from there you learn that some action could leads to a negative reward.
* So rewards don't have to be just at the very end of the process. They can be throughout the journey, throughout the process. It could be given through learning process for certain actions.

|  |  |
| --- | --- |
| * And in terms of *AI* the simplest way to think of *reinforcement learning* is like *training* a *dog*. * When you train the dog you to give it certain commands and if it obeys those commands then you give it a treat, you give it like a biscuit or some food. If it doesn't obeys you tell it that it's a bad dog or you just don't give it a treat. * And through that process it learns what action it needs to take in certain states and these states are the commands that you're giving it.   And based on that it will get some certain rewards. |  |

|  |  |
| --- | --- |
| * In the world of AI, the rewards are -1 and 1 (-ve reward and +ve reward). You just give it a plus one or a minus one. The rewards are digital and just numbers. * Robot dogs can be trained through reinforcement learning. Some older ones have an algorithm in there and not an AI. Older ones are ***preprogrammed*** ***agents*** and now a day's we got ***reinforcement learning agent*** . * In preprogrammed robot, there is a program behind the dog, in the software that will define walking, how to sit, how to stand and things like that. |  |

|  |  |
| --- | --- |
|  | * Whereas in a Robot Dog that is trained through *reinforcement learning*, you don't need to preprogram it. You don't need any algorithm inside that, which is hard coded into the dog. * Instead you have this ***reinforcement learning algorithm*** which makes the robot dog to learn walking itself. * However, you need to define to it the degrees of freedom: For example   Move *right* *foot*, *left* *foot*, can move *right back foot*, *left back foot*.   * These all are the actions the dog can take and it get rewarded are every time. * We define the rewards like: * Take a step forward it get a **plus one.** Every time it fall over, it get a **minus one**. That's all we have to do. |

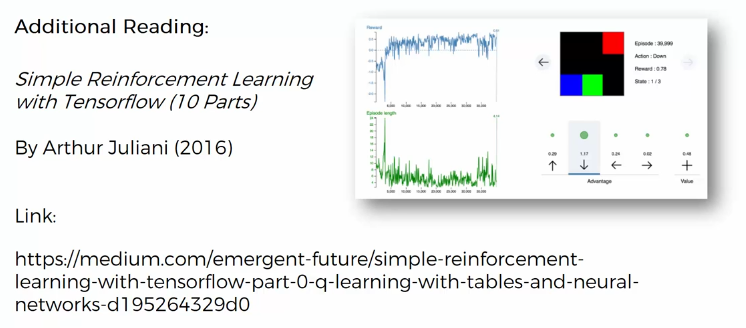
* Then we just leave the dog and let it figure it out on its own.
* At the beginning movement, the dog tries to stand up it falls, then it realizes that "*OK I shouldn't do that action that led to me falling because every time I fall I get a minus one which is not good for me"*.
* After some random movements, it figures out that it can make a step forward by moving its right front foot and because he gets a plus one.
* So it learns from its mistakes and learns to walk on its own.
* Through this learning process, it very quickly understands how it can walk.
* Note that, the dogs that figured out on their own can actually sometimes walk better than dogs that are preprogrammed. Because in preprogrammed way we use our own imagination, whereas a reinforcement learning dog can optimize things on its own.
* And that's how they can Train these Robot Dogs to play Soccer.
* And it's not something that a normal dog has been trained to do or has ever done in process of its evolution.
* Whereas a reinforcement learning robot dogs can do it very easily, and understand how to play soccer as long as you tell them:

1. What the ***rewards*** are
2. What the ***goals*** are
3. What the possible ***actions*** they can take.

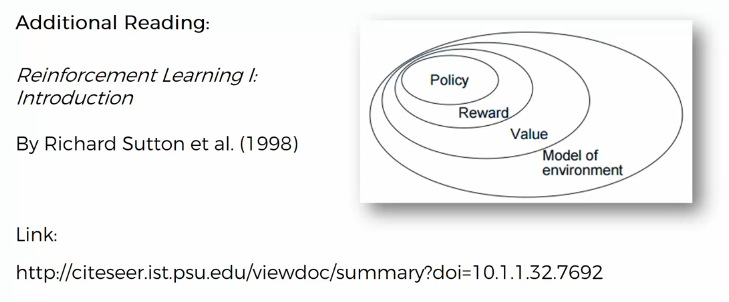
So that is how reinforcement learning works.

Recall 6.1.1 of **Introduction of Machine Learning**.

* Reinforcement Learning: Reinforcement Learning(RL) is a type of machine learning technique that enables an agent to *learn* in an *interactive* *environment* by trial anderror using feedback from its *own* *actions* and *experiences*.
* Though both ***supervised*** and ***reinforcement*** ***learning*** use mapping between ***input*** and ***output***, unlike supervised learning where the feedback provided ***to the agent*** is correct set of actions for performing a task, reinforcement learning uses REWARDS and PUNISHMENTS as signals for ***positive*** and ***negative*** behavior.
* As compared to unsupervised learning, ***reinforcement*** ***learning*** is different in terms of goals. While the goal in ***unsupervised learning*** is to ***find similarities*** and ***differences*** between ***data-points***, in the case of *reinforcement learning* the goal is to *find* a *suitable action model* that would *maximize the total* cumulativereward of the agent.
* ***Reinforcement learning*** is an area of ***Machine Learning***. It is about taking suitable action to ***maximize reward*** in a ***particular*** ***situation***. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation.
* *Reinforcement* *learning* differs from supervised learning in a way that in *supervised* *learning* the ***training*** ***data*** has the ***answer*** ***key*** ***with it*** so the model is ***trained*** ***with*** the ***correct*** ***answer*** itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.
* Additional Readings 1: Here's a great article called SIMPLE REINFORCEMENT LEARNING WITH TENSOR FLOW. It's got ten parts. It is by Arthur Juliani, 2016 article.
* This article is about TF, but here we'll use PyTorch.
* So different implementation but at the same time you might pick up a few things from here and there and build your own AI.



* Additional Readings 2: Next we've got a specific paper to this course, about Reinforcement Learning: there's a paper by Richard Sutton which is called REINFORCEMENT LEARNING I: INTRODUCTION is the 1998 papers are quite old but at the same time you can learn a bit about reinforcement learning some of the examples like that omelet example and other examples of where reinforcement learning can be applied. You'll get a general overview of reinforcement learning.



**1.5 The Bellman Equation**

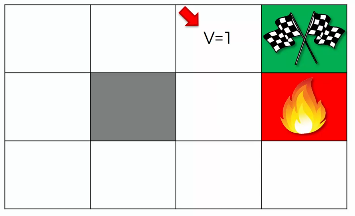
So we're going to have a couple of key concepts (symbols) that we're going to be operating with:

* **s:** stands for states. It is the state in which our agent is or any other possible state it can be.
* **a:** represents an action that an agent can take. So an agent can have access to a certain list of actions.
* actions are very important when they're looked at in a state combination. When you're in a certain state and then you look at actions and it starts to make sense what's going to be the result of those actions because if you'll look an action by itself or a state, it doesn't really make sense, because you don't know where you are and where you can possibly end.
* **R**: stands for *reward* and that's the *reward* that the agent gets for *entering* into a *certain state* .
* : Gamma is the *discount* factor.

|  |  |
| --- | --- |
|  | * The person behind the bellman equation is **Richard Ernest bellman**. * He was a applied mathematician and came up with the concepts of **Dynamic Programming** which we're now which we now call **Reinforcement Learning** in 1953. |

|  |  |
| --- | --- |
| * How it Works: Our agent in the bottom left corner in a maze. The white blocks are blocks in which the agent can step into, the gray block is like a wall in this maze. * The *green* is where the *agent* is should be *aiming* to end up. Here the reward **R = 1**. * The red is fire-pit or agent will lose the game. In the fire pit the reward **R = -1**. So it will be punished with **-1**reward. |  |

* **How does it learn how to operate in this maze:**
* Define the actions that is can do: moving up, right, left or down are four possible actions that it can take.
* First it randomly moves to see what happens.
* It haven't learnt anything yet, it just moving randomly. So far nothing's happened. At the time it ends up in the Green Square, it will get a **+1** reward.
* And that triggers the algorithm to realize that it should end up in the square.
* Then it tries to figure out *how it got into this square*. What was the preceding state it was in and what action that it take to get to square.
* So it looks back.



* It remembers the preceding state (marked by the Red Arrow). This state is so valuable; here it is one step away from getting the maximum reward. All it has to do is move right.
* So how does it remember that the state is valuable?
* As the agent, actually there is *no difference* among the states.
* There's no *difference* in whether it in the *Green Square* or in the white square.
* In the Green Square it got the reward of **+1**.
* So it marks the white square **V=1**, because it leads exactly to reward (just take one more action to move right). **V=1** is the perceived value of being in the state.

|  |  |
| --- | --- |
| * Next the agent's going to remember how did it got into this ***white square***. * It might walk around again and end up in the white-square anyway it finds out that it was from the square in the left. * So as soon as it gets into this square, it knows to move to the next white-square and from there it move to the green-square. All is has to do is go right. * Hence the value of being in this state is also **V=1**. since there's nothing stopping it from going second-white box to first-white box. |  |

|  |  |
| --- | --- |
| * Similarly we give following values to the squares. * That’s how we could possibly think about Designing A Simple Equation that helps an agent go through the maze. * So look at the reward (the green box) then the preceding state gives it a value of equal to reward, and those values create a kind of pathway. |  |

* Starting from RANDOM state:
* Creating a path like above is great but the problem is what happens if our agent for some reason starts from a random state?
* Since all values (**V=1**) are same for all states (**boxes**),
* How does it know/remember which action to take should it go: right/down/left/up?
* How does it remember which is the next continuation from this random state?
* If the only values it can see, is **V=1**, it cannot see what's further away. These values (**V=1**) are identical for our agent.

So that's why above approach doesn't really work.

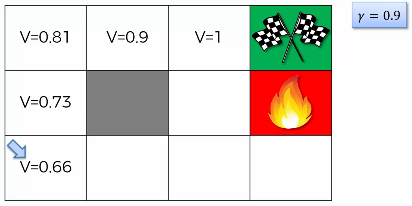
* So how do we solve this problem?
* That’s where the Bellman equation comes into play.
* The BELLMAN Equation: The Bellman equation looks something like following.



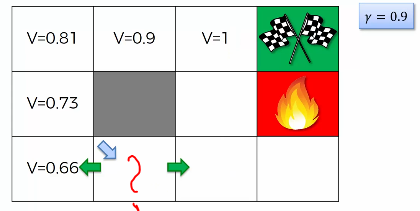
* Notations:
* : we've already talked about the value of being in a certain state.
* : Value of your current state.
* : The following state that you will end up in after the current-state by taking certain action.
* : Is the action. is used since there are many actions that an agent can take.
* **R**: stands for *reward* and that's the *reward* that the agent gets for *entering* into a *certain state* .
* : Gamma is the *discount* factor.
* **So by taking an action what will happen to an agent:**
* Let's say we're in state by taking an action , agent will instantly get a reward by getting into a new state.
* That reward can be **+1** or **-1** (if it's at the end of the game) or it can be **0** if it's throughout the game.
* And then we get into the new state , which has the value .
* So by taking action we get reward and we end up in a new state. So for every possible actions we're going to have a equation like this.
* going to be different value for different action and we're going to look at only the *maximum* because the agent wants to go to the *optimal state*.
* So if the agent is in state he's going to look at these values he's going to ***find the maximum*** of these values. So that, it takes an action , which ***maximize*** .
* So hopefully that makes sense why we're taking the maximum here.
* Role of : helps the agent to decide which way it should go. Once we got the reward and the value , solves that problem of where the agent doesn't know which way to go if the values of two states on both sides are the same. That's why the called the discounting factor.
* Let's analyze our maze again, this time with Bellman Equation:

|  |  |
| --- | --- |
| * We're going to calculate the value of being in that state. * The white-square, next to the green square has the value **V=1**, because it closest to the finish line. Since we're already in the best possible state, the has no value. And the reward **R = 1**. |  |
| * Now things get interesting when we move to the left when we move backwards to the next white-square. Now if we move to the next white-square, say as discounting factor. * Since we are moving behind, the reward is and value of new state (white-square, next to the green square, we calculated above) is . Hence from the equation we get V=0.9 for this white-square. |  |
| * So now similarly for the next white-square behind we have: , , . Hence **V=0.81**. |  |

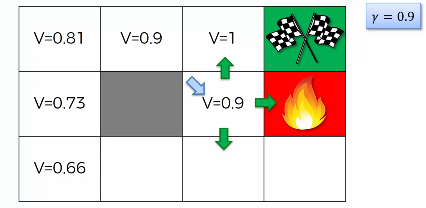
* Similarly we evaluate the values for other squares.



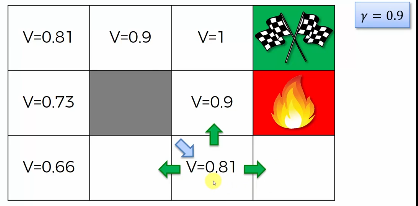
* It is similar to time value of money in the finance theory.
* **The important thing to understand here is that:** it is just a theory, a way that reinforcement learning works. Richard Bellman came up with this equation.
* The equation doesn't have to be this exact equation, you could come up with a different equation where , , could be different.
* But this approach works in similar fashion, further ***away from the goal you are getting the less value*** of it being in the state.
* We're using to inspire agents to be closer to the finish line. So if the agent is in a random square, it can find a direction by following the nearby maximum value. That’s how it can determine its path.
* Lets evaluate Rest of the square: How do we calculate the value for the following square?



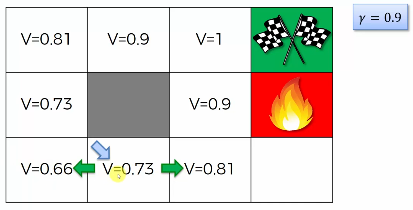
* Well here is where things get tricky, because it might actually be shorter to go another way. That’s why we calculate the value of the following square first. It has three possible actions.



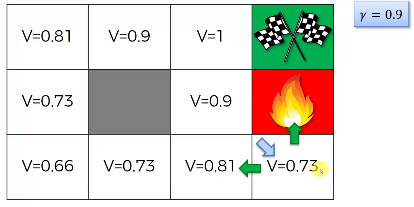
* Then we calculate the value for the square below, using the same approach.



* Then we can find out the value of the skipped square as below:

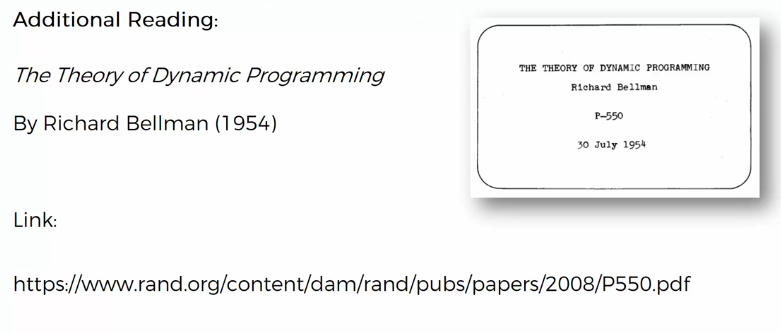


* And for the last square (below the danger/fire):



Now all the squares has the values, and the agent can find its way easily.

* Additional Reading: There is a paper which you can look at and it is the original paper by Richard Bellman. It's called the Theory Of Dynamic Programming from 1954. Bear in mind that this is quite a mathematically heavy paper.



**1.6 The Plan**

* PLAN: Here is our *maze analysis*. Here we can see the *values* of *every single state*. And how an agent can *navigate* this maze. The plan is simply like a ***treasure map*** for artificial intelligence.
* Instead of looking at these values they replace them with arrows, which indicate in which direction the agent should go.

|  |  |
| --- | --- |
|  |  |

* So after the agent explored this environment, it knows the value of each state and therefore it can come up with this map.
* Don't confuse ***plan*** with ***policy*** because policy is very similar to plans but we apply a little trick because the environment's going to be a bit different. It's going to be ***stochastic***.

**1.7 Markov Decision Process**

Here we'll discuss about Markov Decision Processes or MDP.

|  |  |
| --- | --- |
| * Previously we've calculated the values based on the BELLMAN EQUATION we can derive this map for our agent on this maze. So that wherever the agent starts it knows exactly which steps to take in order to get to the finish line. * Is it really that simple? The reality is that it's not actually that simple. * This is where the MDP comes into play. |  |

* ***Deterministic Search vs Non-Deterministic Search:*** Before discussing MDP, we're going to talk about two things.

1. Deterministic Search
2. Non-Deterministic Search

|  |  |
| --- | --- |
| * Deterministic Search: Here is our agent in the maze and *deterministic search* means that if the *agent* decides to go ***up*** then there is ***100% probability*** that it will go up. * There's no other options. * Nondeterministic search: When our agent says it wants to go up. They are actually couple of options. * Here we have three options, but the no. of options depends on the randomness of the problem. * Our three options could be: * 80% chance it does go up. * 10% chance to go to the Left. * 10% chance to go to the Right. In this case it falls into the Fire-Pit. |  |

* A Nondeterministic Search is a stochastic process and the point of this is to make a more *realistic model*. Because in real world there is always will be some uncertainty.
* There will be something that is not under the control of the agent, something that cannot be predicted. Sometimes you might have a different situation, where the possibilities might be different.
* We have to deal with it and learn how that affects the Bellman equation and whole reinforcement learning process.

That's what we are modeling here in nondeterministic search.

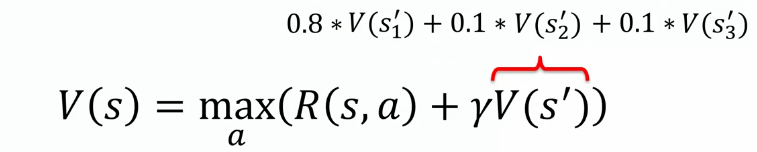
* MP and MDP: Now it is time to introduce two new concepts a **Markov processes** and or a **Markov Decision processes (MDP)** so let's have a look at these.
* Markov process: A **stochastic process** has the **Markov property** if the conditional probability distribution of ***future*** ***states***  of the process (conditional on both *past* and *present* states) depends only upon the ***present*** ***states***, ***not*** on the *sequence of events* that *preceded* it. A process with this property is called a Markov process. (Wikipedia).
* Markov property: A Markov property is when your future states (not just your choice but the whole thing- your choice and the environment, i.e. the results of all of the action you take in that environment) will only depend on where you are now. It will not depend on how you got there.
* And a process which has this property is called the Markov process.
* **Stochastic:** Having a random probability distribution or pattern that may be analyzed statistically but may not be predicted precisely.
* Stochastic refers to the *property* of being well described by a *random probability distribution*.
* **Stochasticity:** the quality of lacking any predictable order or plan.

|  |  |
| --- | --- |
| * If our agent decides to go up, in our case of nondeterministic search example he actually might go left and right. That's because we have that stochasticity/ randomness inside our environment. * But the key-point is that this is a Markov process, we don't care how the agent got here. * He could have come from anywhere in that maze. Or, it could've a moved around "here & there" like 100000 times and then got in that specific square. It does not matter what happened before, it only matters is which state is he in now – the Current State. |  |

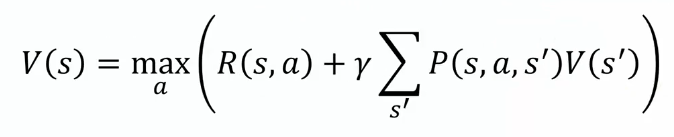
* So the ***probabilities*** of going ***left*** or ***right*** or ***up*** they will always be the same if he's in this state now. Doesn't matter how it got there. The future is only determined by the state you're in now, the actions you will take and the randomness.

So that's a Markov process.

* **Markov Decision processes (MDP)**: Markov Decision processes (MDP) provide a mathematical framework for modeling *decision making* in situations where outcomes are *partly random* and *partly under control* over *decision* *maker*.
* Note that, MDP and Markov Process are different.
* Markov Decision processes is exactly what we've been discussing until now - so that the agent lives in this environment where, it has maximum control (partly under control) with a little bit of uncertainty (partly random).
* For example, if it want go up there's 80% chance to go up, 10% to go right and 10% to go left.
* Hence not everything is fully under its control, there is some randomness in this environment and that's exactly what a Markov Decision Process (MDP) is.
* **Markov decision process** is the framework that the agent will use in order to understand what to do in this environment.
* So we've got an environment with some stochasticity/randomness, and the agent has to choose for instance should go up / down / left / right.
* He has to make that decision.
* He doesn't know what to do.
* And in order to make that decision (what's going to happen or where it have to go) is going to apply a framework called Markov Decision processes.
* At the same time the environment has the Markov Property.
* So basically here we have two concepts:
* Markov process: What happens in future doesn't depend on the past, it depends on the current state.
* Markov Decision processes is the framework that the agent is going to apply in order to solve this environment.
* **Improving the Bellman Equation:** However, the *MDP* is just an *add-on* to our *Bellman Equation*. So let's have a look at that.
* Following is our ***previous Bellman equation***:
* Now because we have some randomness in our whole process this part will change. Because *we don't actually know* which *state* it will *end up*, we don't know what will be (going *up* or *left* or *down* or *right*).
* So we have to replace with the *expected value of the next state*. are the three possible states we can end up.

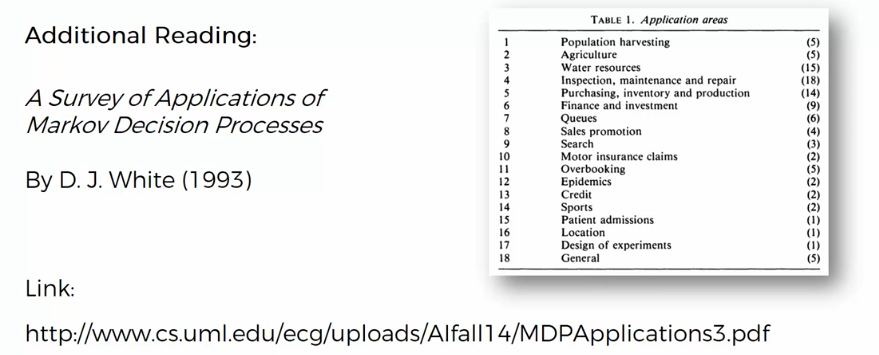


* And so we're going to replace that with these values **, ,** .
* Then we're going to multiply these with the possibilities: 80% chance to go up, 10% chance to go left, 10% chance to go right, like follows:
* Then we end up with following equation:



* In previous equation, represents the *Deterministic Search* (you knew which states you'll get into), but now we don't know which state we'll get into.
* Represents the probability of getting to the sate from the state by taking action .
* It is multiplied with the corresponding sate's value .
* Then we take the sum and finally multiply with .
* So this is our **new Bellman equation** applying MDP. That agents use this framework to solve this whole Stochastic Nondeterministic Search Problem where there's *random events* that are happening that *they cannot control*.
* There are probabilities involved in the consequences of agents action.
* That's how a MDP works and the underlying equation behind it. This is more close to real world problems or game scenarios.
* Because not everything is straightforward, there is always some randomness involved.
* And taking an action in a certain state will not always lead to the same outcome.
* Additional Reading::

We found a very applied paper. It's called A SURVEY OF APPLICATIONS OF MARK OF DECISION PROCESSES and it was written by D. J. White in 1993.



* It'll show you examples of where Markov decision processes actually are used to model real life problems.
* For example, population harvesting: Let's say you have some fish and you know what the population of fish is (current state) you need to decide how many fish can be catch this year (the action).
* We want to calculate the possible outcomes of that.
* How many fish will we have next year or the year after and so on.
* And it's not deterministic because you cannot take out 90% of the population and next year you'll get back to 100%.
* There are certain random factors involved which are out of our control.

Therefore we have to model what's going to happen, that's where a MDP is used.

* Similarly MDP can be applied to agriculture, harvesting crops.
* MDP can also be applied to finance and investment and marketing.
* From this paper you can get the idea where AI can be applied in Real Life situation.
* This possibly could *trigger some ideas* for you: how you could apply AI in the future to make the world a better place.