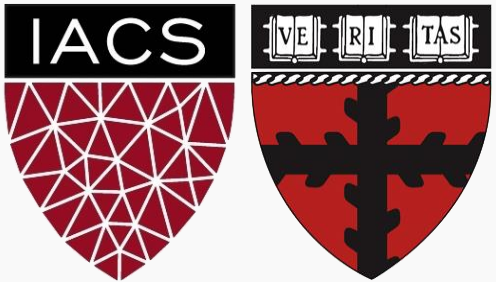
Lecture 14: Introduction to Reinforcement Learning

CS109B Data Science 2

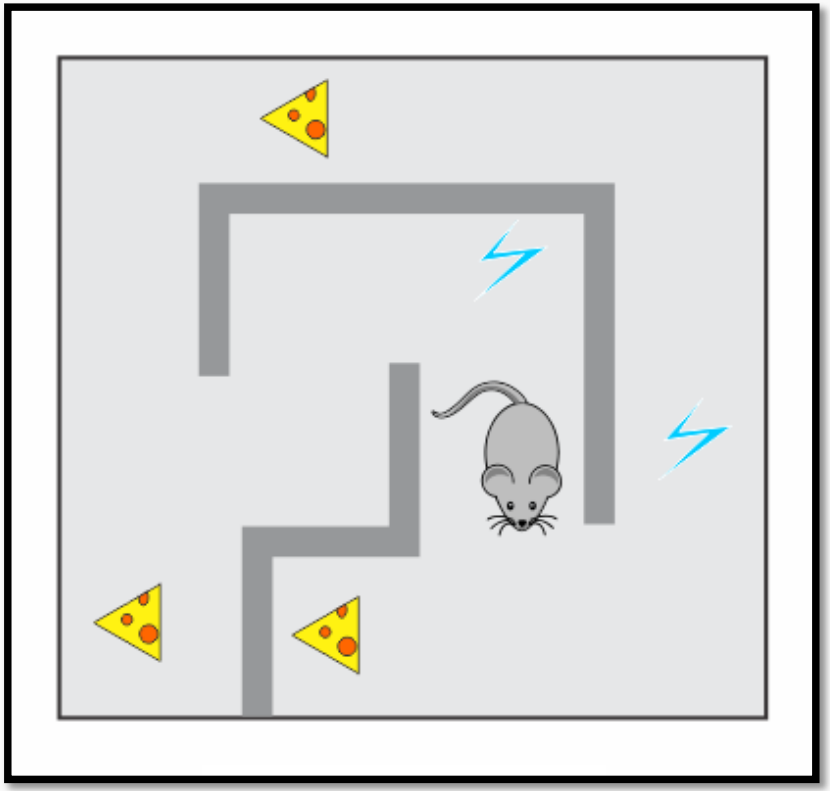
Pavlos Protopapas and Mark Glickman



Outline

- What is Reinforcement Learning
- RL Formalism
    1. Reward
    2. The agent
    3. The environment
    4. Actions
    5. Observations
- Markov Decision Process
    1. Markov Process
    2. Markov reward process
    3. Markov Decision process
- Learning Optimal Policies

What is Reinforcement Learning ?

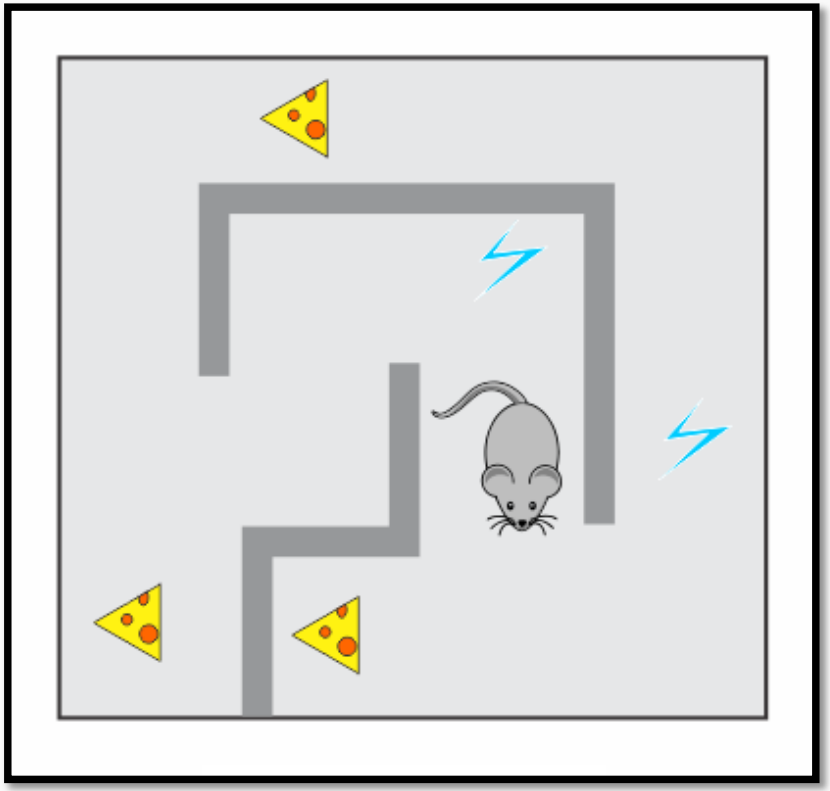


Lapan, Maxim. Deep Reinforcement Learning Hands-On

Describe this:

- Mouse
- A maze with walls, food and electricity
- Mouse can move left, right, up and down
- Mouse wants the cheese but not electric shocks
- Mouse can observe the environment

What is Reinforcement Learning ?



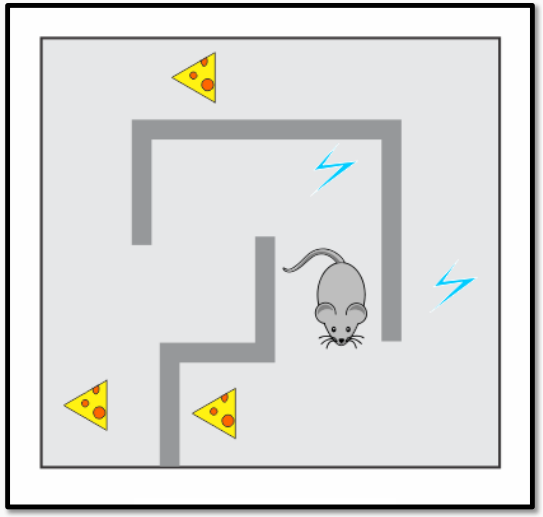
Lapan, Maxim. Deep Reinforcement Learning Hands-On

Describe this:

- Mouse => **Agent**
- A maze with walls, food and electricity => **Environment**
- Mouse can move left, right, up and down => **Actions**
- Mouse wants the cheese but not electric shocks => **Rewards**
- Mouse can observe the environment => **Observations**

What is Reinforcement Learning ?

Learning to make sequential decisions in an environment so as to maximize some notion of overall ***rewards*** acquired along the way.



In simple terms:

The mouse is trying to find as much food as possible, while avoiding an electric shock whenever possible.

The mouse could be brave and get an electric shock to get to the place with plenty of food—this is better result than just standing still and gaining nothing.

What is Reinforcement Learning ?

- Learning to make sequential decisions in an environment so as to maximize some notion of overall ***rewards*** acquired along the way.
- Simple Machine Learning problems have a hidden time dimension, which is often overlooked, but it is important become in a production system.
- **Reinforcement Learning** incorporates time (or an extra dimension) into learning, which puts it much close to the human perception of artificial intelligence.

What we don't want the mouse to do?

- We do **not** want to have best actions to take in every specific situation. Too much and not flexible.

- Find some magic set of methods that will allow our mouse to learn on its own how to avoid electricity and gather as much food as possible.

Reinforcement Learning is exactly this magic toolbox

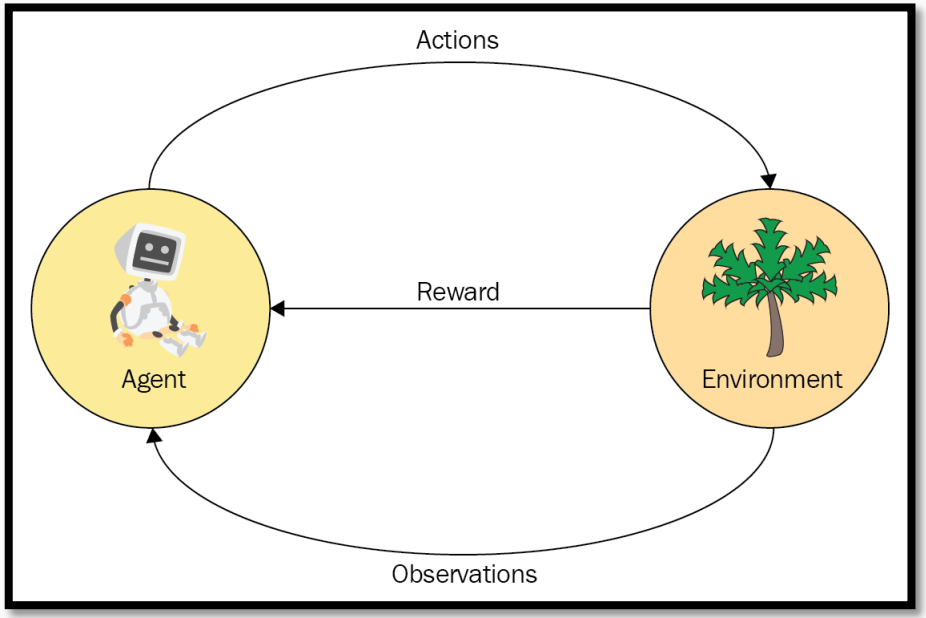
Challenges of RL

A. Observations depends on agent's actions. If agent decides to do stupid things, then the observations will tell nothing about how to improve the outcome (only negative feedback).

B. Agents need to not only exploit the policy they have learned, but to actively explore the environment. In other words maybe by doing things differently we can significantly improve the outcome. This exploration/exploitation dilemma is one of the open fundamental questions in RL (and in my life).

C. Reward can be delayed from actions. Ex: In cases of chess, it can be one single strong move in the middle of the game that has shifted the balance.

RL formalisms and relations



- Agent
- Environment

Communication channels:

- Actions,
- Reward, and
- Observations:

Lapan, Maxim. Deep Reinforcement Learning Hands-On

Reward

Reward

- A scalar value obtained from the environment
- It can be positive or negative, large or small
- The purpose of reward is to tell our agent how well they have behaved.

reinforcement = reward or **reinforced** the behavior

Examples:

- Cheese or electric shock
- Grades: Grades are a reward system to give you feedback about you are paying attention to me.

Reward (cont)

All goals can be described by the maximization of some expected cumulative reward

The agent

The agent

An agent is somebody or something who/which interacts with the environment by executing certain actions, taking observations, and receiving eventual rewards for this.

In most practical RL scenarios, it's our piece of software that is supposed to solve some problem in a more-or-less efficient way.

Example:

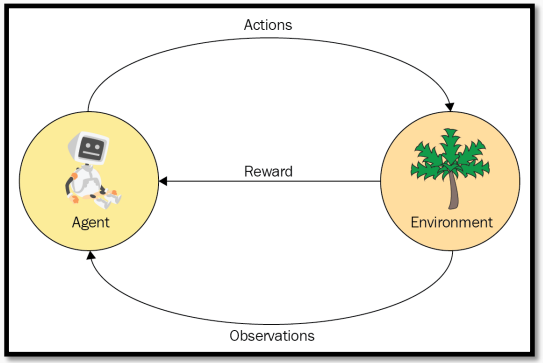
You

The environment

Everything outside of an agent.

The universe!

The environment is external to an agent, and communications to and from the agent are limited to rewards, observations and actions.



Actions

Things an agent can do in the environment.

Can be:

- moves allowed by the rules of play (if it's some game),
- or it can be doing homework (in the case of school).

They can be simple such as move pawn one space forward,
or complicated such as fill the tax form in for tomorrow morning.

Could be discrete or continuous

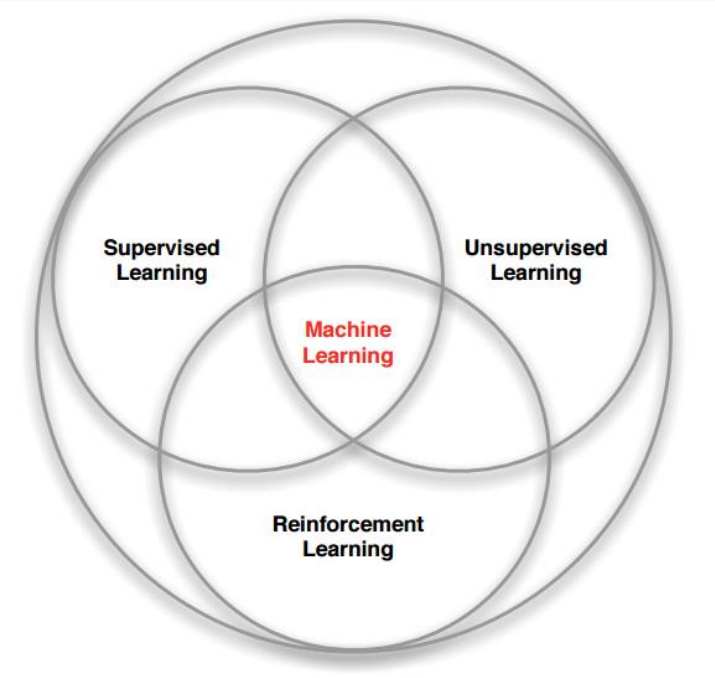
Observations

Second information channel for an agent, with the first being a reward.

Why?

Convenience

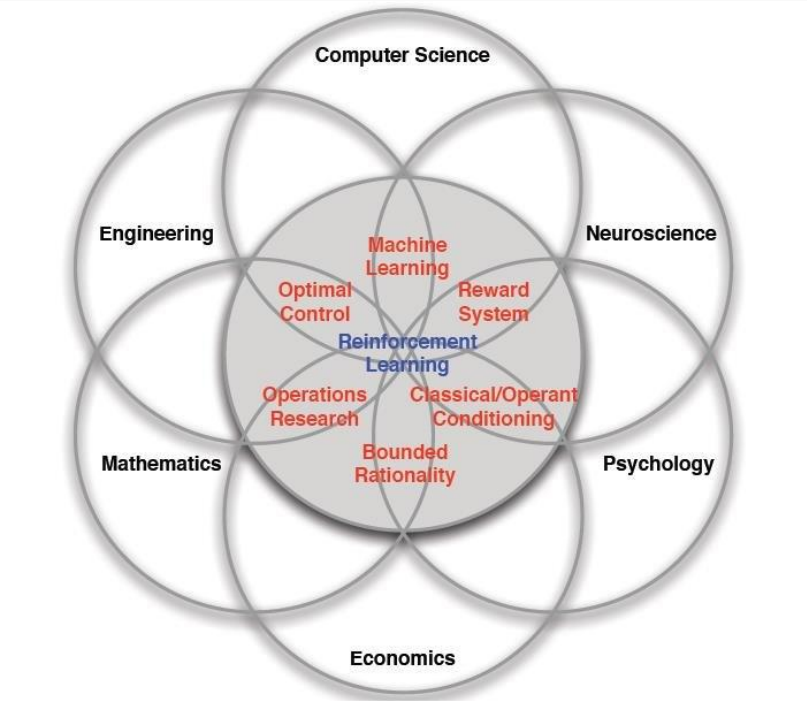
RL within the ML Spectrum



What makes RL different from other ML paradigms ?

- No supervision, just a reward signal from the environment
- Feedback is sometimes delayed (Example: Time taken for drugs to take effect)
- Time matters - sequential data
- Feedback - Agent's action affects the subsequent data it receives (not i.i.d.)

Many Faces of Reinforcement Learning



- Defeat a World Champion in Chess, Go, BackGammon
- Manage an investment portfolio
- Control a power station
- Control the dynamics of a humanoid robot locomotion
- Treat patients in the ICU
- Automatic fly stunt manoeuvres in helicopters

Outline

What is Reinforcement Learning

RL Formalism

1. Reward
2. The agent
3. The environment
4. Actions
5. Observations

Markov Decision Process

1. Markov Process
2. Markov reward process
3. Markov Decision process

Learning Optimal Policies

MDP + Formal Definitions

Markov Decision Process

More terminology we need to learn

- state
- episode
- history
- value
- policy

Markov Process

Example:

System: Weather in Boston.

States: We can observe the current day as sunny or rainy

History: . A sequence of observations over time forms a chain of **states**, such as

[sunny, sunny, rainy, sunny, …],

Markov Process

- For a given system we observe **states**
- The system changes between states according to some dynamics.
- We do not influence the system just observe
- There are only finite number of states (could be very large)
- Observe a sequence of states or a chain => **Markov chain**

Markov Process (cont)

A system is a **Markov Process**, if it fulfils the **Markov property**.

The future system dynamics from any state have to depend on this state only.

- Every observable state is self-contained to describe the future of the system.
- Only one state is required to model the future dynamics of the system, not the whole history or, say, the last N states.

Markov Process (cont)

Weather example:

The probability of sunny day followed by rainy day is independent of the amount of sunny days we've seen in the past.

Notes:

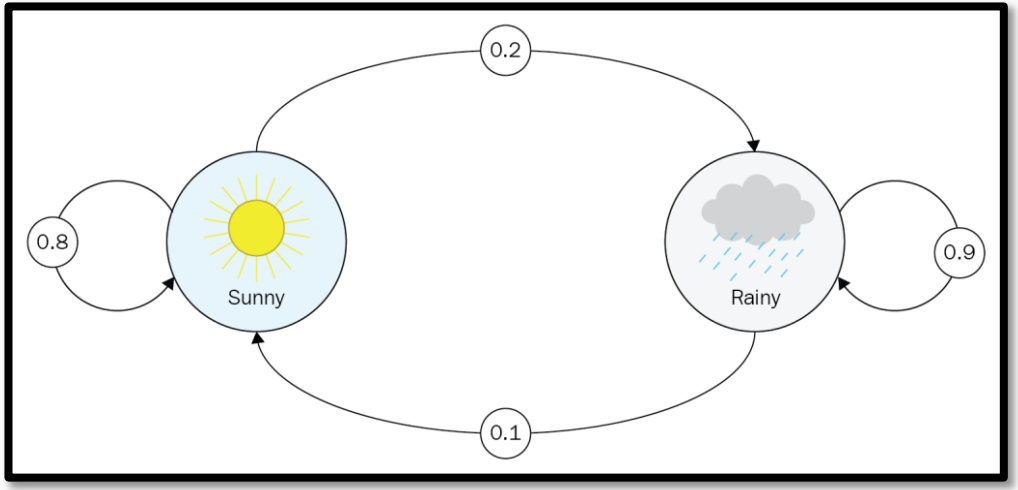
This example is really naïve, but it's important to understand the limitations.

We can for example extend the state space to include other factors.

Markov Process (cont)

Transition probabilities is expressed as a **transition matrix**, which is a square matrix of the size N×N, where N is the number of states in our model.

	sunny	rainy
sunny	0.8	0.2
rainy	0.1	0.9



Markov Reward Process

Extend Markov process to include rewards.

Add another square matrix which tells us the reward going from state i to state j.

Often (but not always the case) the reward only depends on the landing state so we only need a number:

$$R_t$$

Note: Reward is just a number, positive, negative, small, large

Markov Reward Process (cont)

For every time point, we define **return** as a sum of **subsequent** rewards

$$G_t = R_{t+1} + R_{t+2} + \ \ldots$$

But more distant rewards should not count as much so we multiply by the discount factor raised to the power of the number of steps we are away from the starting point at time t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0} \gamma^k R_{t+k+1}$$

Markov Reward Process (cont)

The **return** quantity is not very useful in practice, as it was defined for every specific chain. But since there are probabilities to reach other states this can vary a lot depending which path we take.

Take the expectation of return for any state we get the quantity called a **value of state:**

$$V(s) = \mathbb{E}[G|S_t = s]$$

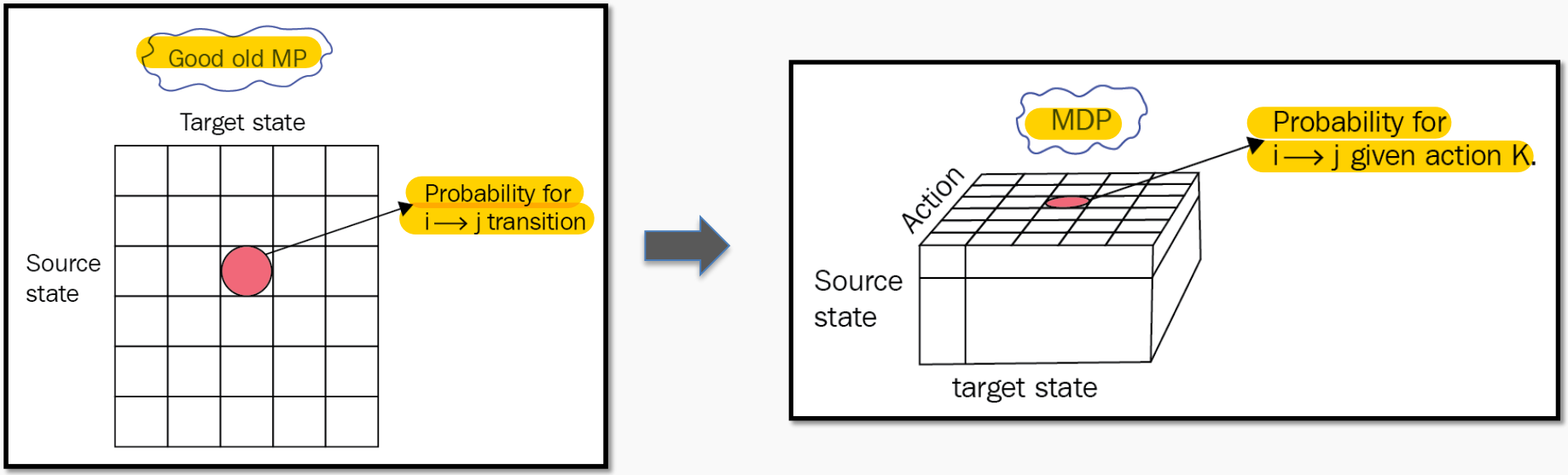
Markov Decision Process

How to extend our Markov Return Process to include **actions**?

We must add a set of actions (A), which has to be finite. This is our agent's *action space*.

Condition our transition matrix with action, which means the transition matrix needs an extra action dimension => turns it into a cube.

Markov Decision Process (cont)



Lapan, Maxim. Deep Reinforcement Learning Hands-On

Markov Decision Process (cont)

By choosing an action, the agent can affect the probabilities of target states, which is GREAT to have.

Finally, to turn our MRP into an MDP, we need to add actions to our reward matrix in the same way we did with the transition matrix: our reward matrix will depend not only on state but also on action.

In other words, it means that the reward the agent obtains now depends not only on the state it ends up in but also on the action that leads to this state. It's similar as when putting effort into something, you're usually gaining skills and knowledge, even if the result of your efforts wasn't too successful.

Markov Decision Process

More terminology we need to learn

- state ✓
- episode ✓
- history ✓
- value ✓
- policy

Policy

We are finally ready to introduce the most important central thing for MDPs and Reinforcement Learning:

policy

The intuitive definition of policy is that it is some set of rules that controls the agent's behavior.

Policy (cont)

Even for fairly simple environments, we can have a variety of policies.

- Always move forward
- Try to go around obstacles by checking whether that previous forward action failed
- Funnily spin around to entertain
- Choose an action randomly

Policy (cont)

Remember: The main objective of the agent in RL is to gather as much return (which was defined as discounted cumulative reward) as possible.

Different policies can give us different return, which makes it important to find a good policy. This is why the notion of policy is important, and it's the central thing we're looking for.

Policy (cont)

Formally, policy is defined as the probability distribution over actions for every possible state:

$$\pi(a|s) = P(A_t = a|S_t = s)$$

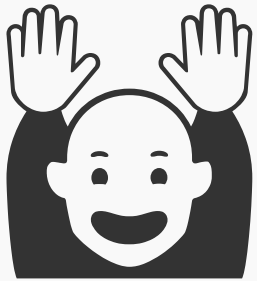
An optimal policy $\boldsymbol{\pi}^*$ is one that maximizes the expected value function :

$$\boldsymbol{\pi}^* = \text{argmax}_{\boldsymbol{\pi}} V_{\boldsymbol{\pi}}(s)$$

Markov Decision Process

More terminology we need to learn

- state ✓
- episode ✓
- history ✓
- value ✓
- policy ✓



Learning Optimal Policies

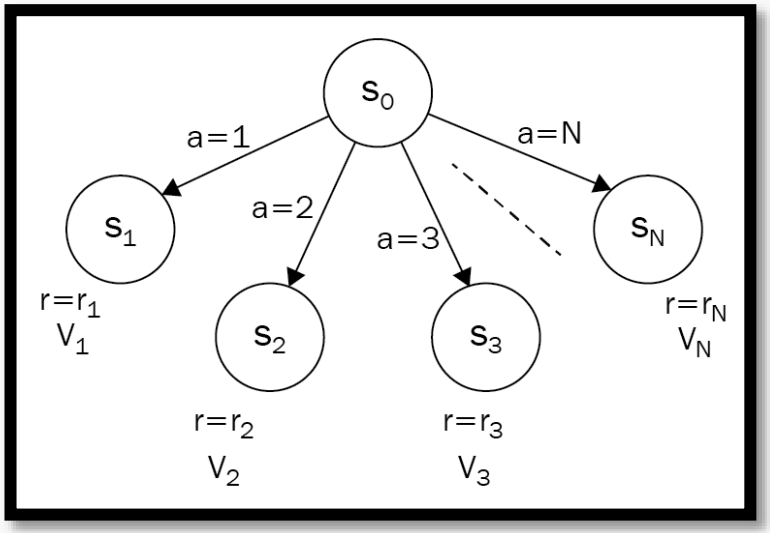
Dynamic Programming Methods (Value and Policy Iteration)

Value of Action Q(s,a)

- The total reward of the one-step rewards for taking action a in state s and can be defined via $V(s)$.
- Provides a convenient form for policy-optimization and learning policies Q-learning.

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma \mathbb{E}_T[V_\pi(s_{t+1})]$$

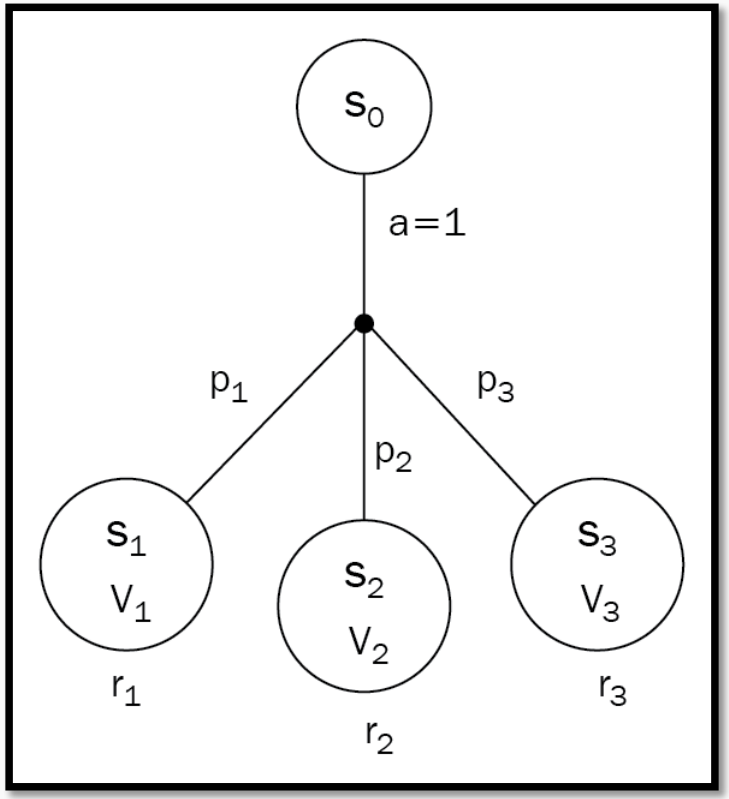
Bellman equation (deterministic)



The best possible action, the agent needs to calculate the resulting values for every action and choose the maximum possible outcome. (not totally greedy)

$$V_0 = \max_{a \in 1 \ldots N} (r_a + \gamma V_a)$$

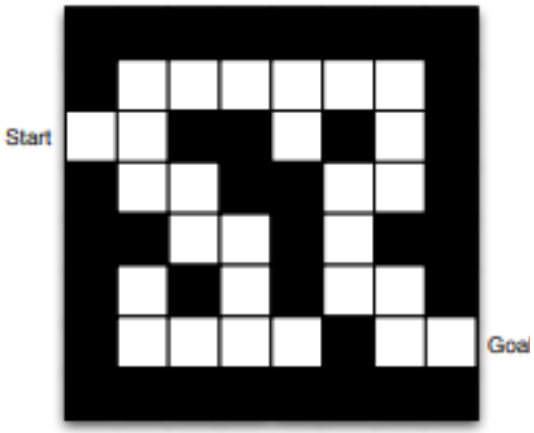
Bellman equation (stochastic)



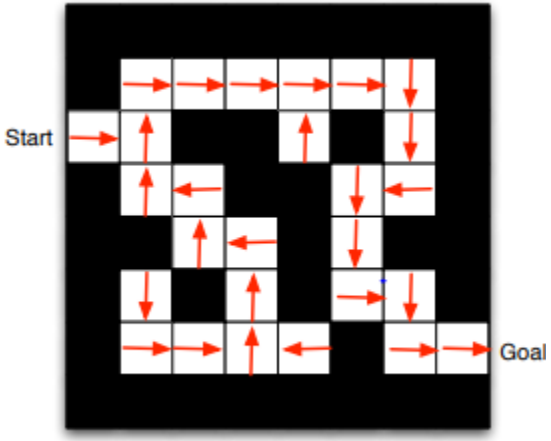
Bellman optimality equation for the general case:

$$V_0 = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s}(r_{s,a} + \gamma V_s)$$

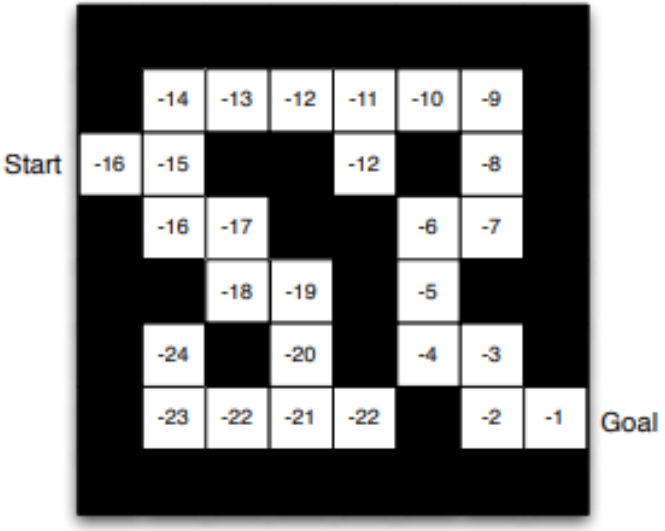
Maze Example



Rewards of -1 for each step.
Four actions - U,D,L,R
Goal ends the episode.



Policy



Value Function under the policy

Dynamic Programming

- Remember that value functions are recursive.

- Dynamic Programming - Breaking down a big problem into smaller sub-problems and solving the smaller sub-problems, store its values and backtrack towards bigger problems.

WORKING BACKWARDS :

$V(s_{T-1},a_{T-1}) = R(s_{T-1},a_{T-1})$; $V(s_{T-2},a_{T-2}) = R(s_{T-1},a_{T-1}) + V(s_{T-1},a_{T-1})$; …… (T is terminal state)

Value Iteration

1. Start with some arbitrary value assignments V_0
2. Update Policy and repeat until $|V_{n+1}(s) - V_n(s)| < \epsilon$

$$Q_n(s,a) = R(s,a) + \gamma \mathbb{E}_T[V(s')]$$

$$V_{n+1}(s) = \text{argmax}_a Q_n(s,a)$$

INTUITION : Iteratively improve your value estimates using Q, V relations.

Policy Iteration

1. Start with some policy $\boldsymbol{\pi}_0$
2. Compute the value of the states V(s) using current policy. *(Policy Evaluation)*
3. *(Policy Improvement)* Update Policy and repeat steps (2) and (3) until $\boldsymbol{\pi}_{k+1} = \boldsymbol{\pi}_k$

$$\boldsymbol{\pi}_1(s_i) = \text{argmax}_a\{r(s_i,a_i) + \boldsymbol{\gamma} E_T[V_{\boldsymbol{\pi}0}(s_j)]\}$$ - Transition from s_i to s_j

INTUITION : At each step, you are modifying your policy by picking that action which gives you the highest Q-value. (Remember the fixed point eqns from slide 13)

Model-Free Methods

Q-Learning and SARSA

Why Model-Free Methods ?

- Learning or providing a transition model can be hard in several scenarios.
    - Autonomous Driving, ICU Treatments, Stock Trading etc.

<u>What do you have then ?</u>

An ability to obtain a set of simulations/trajectories with each transition in the episodes of the form (s,a,r,s’)

E.g. Using sensors to understand robot’s new position when it does an action, Recording new patient vitals when given a drug from a state etc.

On-Policy vs Off-Policy Learning

- On-Policy Learning
  - Learn on the job.
  - Evaluate policy π when sampling experiences from π.

- Off-Policy Learning
  - Look over someone's shoulder.
  - Evaluate policy π (target policy) while following a different policy Ψ (behavior policy) in the environment.

Some domains prohibit on-policy learning. For instance, treating a patient in ICUs you cannot learn about random actions by testing them out.

Temporal Difference (TD) Learning

Remember : $V_{\pi}(s) = R(s,a \sim \boldsymbol{\pi}) + \boldsymbol{\gamma} E_T[V(s')]$. For any policy, execute and learn V.

Given a transition (s,a,r,s'), a TD Update adjusts the value function estimate in line with Bellman-Equation

$$V_{\pi}^{new}(s) \leftarrow V_{\pi}^{old}(s) + \alpha[R(s, a\sim\pi] + \gamma\, V_{\pi}^{old}(s) - \mathrm{V}^{\mathrm{old}}{}_{\boldsymbol{\pi}}(\mathrm{s})\,]$$

Perform many such updates over several transitions and we should see convergence. When it converges($V^{new}=V^{old}$), we expect Bellman Equation to hold. i.e.

$$R(s,a \sim \boldsymbol{\pi}) + \boldsymbol{\gamma} V(s') - V_{\boldsymbol{\pi}}(s) = 0$$

Q-Learning

- Start with a random Q-table (S X A). For all transitions collected according to any behavior policy, perform this TD Update

$$\mathbf{Q(s,a) \leftarrow Q(s,a) + \alpha\,[\,R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)\,]}$$

- OVER-OPTIMISTIC : Assumes the best things would happen from the next state onwards - Greedy (Hence the max operation over future Q-values)
- OFF-POLICY : Q directly approximates the optimal action value function independently of the policy being followed (max over all actions)

SARSA

- Start with a random Q-table (S X A). For all transitions (collected by acting according to π that maximizes Q) perform this TD Update

$$Q(s,a) \leftarrow Q(s,a) + \alpha [R(s,a) + \gamma Q(s',a' \sim \pi) - Q(s,a)]$$

π - Data collection policy

- ON-Policy Learning : While learning the optimal policy it uses the current estimate of the optimal policy to generate the behaviour

Q-Learning and SARSA Algorithm

1. Start with a random Q-table (S X A).
2. Choose one among the two actions
    a. (ε-greedy) With probability ε, choose a random action (EXPLORATION)
    b. With probability 1-ε, an action that maximizes Q-value from a state.(EXPLOITATION)
3. Perform an action and collect transition (s,a,r,s')
4. Update Q-table using the corresponding TD updates.
5. Repeat steps 2-5 till convergence of Q-values across all states.

Q-Learning vs SARSA

Demo : https://studywolf.wordpress.com/2013/07/01/reinforcement-learning-sarsa-vs-q-learning/

- Q-Learning converges faster since Q values directly try to approximate the optimal value.
- Q-Learning is more risky since it is over-optimistic of what happens in the future. Could be risky for real-life tasks such as robot navigation over dangerous terrains.

Parametric Q-Learning

- Often hard to learn Q-values in tabular form. E.g. Huge number of states, Continuous state spaces etc.
- Parametrize Q(s,a) using any function approximator f - linear model, neural networks etc. and do usual Q-learning.

$$Q(s,a) = f(s,a;\boldsymbol{\theta}) \qquad \boldsymbol{\theta}\text{- model params}$$

Example : Image Frames in a game - Use ConvNets to parametrize Q(s,a)