Dynamic Programming: MDP

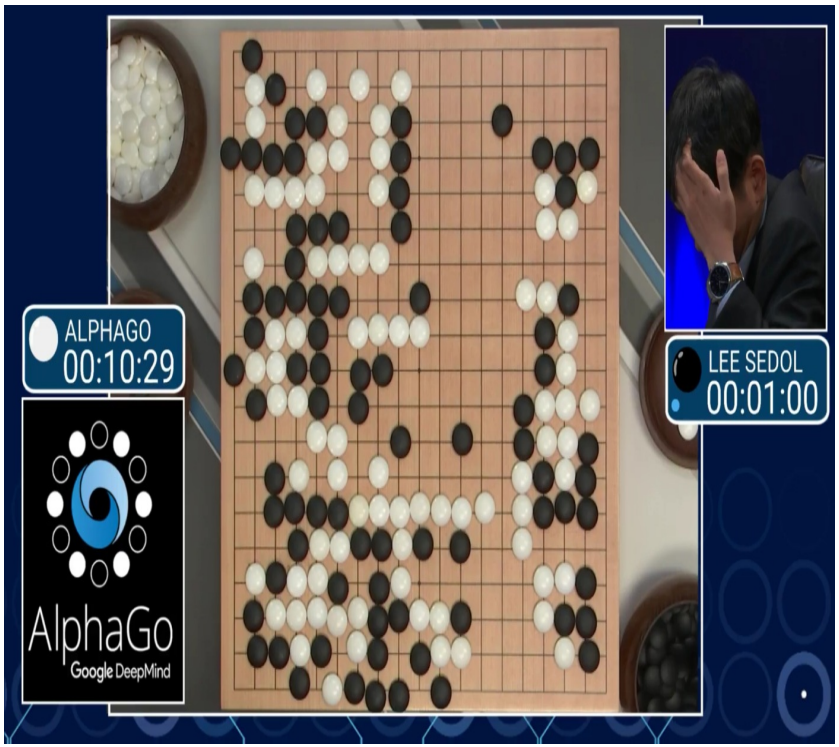
http://bicmr.pku.edu.cn/~wenzw/bigdata2022.html

Acknowledgement: this slides is based on OpenAI Spinning Up and and Prof. Shipra Agrawal's lecture notes

Outline

What Can RL Do?

RL methods have recently enjoyed a wide variety of successes. For example, it's been used to teach computers to control robots in simulation



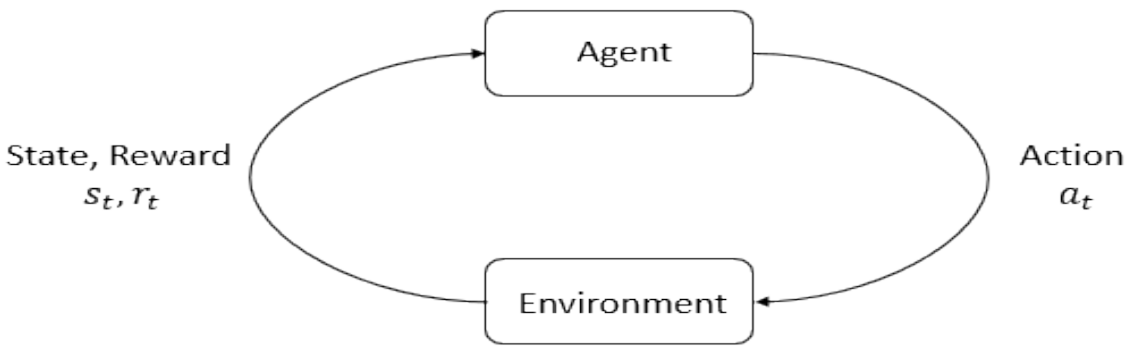
What Can RL Do?

It's also famously been used to create breakthrough AIs for sophisticated strategy games, most notably 'Go' and 'Dota', taught computers to 'play Atari games' from raw pixels, and trained simulated robots 'to follow human instructions'.

- Go: `https://deepmind.com/research/alphago`
- Dota: `https://blog.openai.com/openai-five`
- play Atari games: `https://deepmind.com/research/dqn/`
- to follow human instructions: `https://blog.openai.com/deep-reinforcement-learning-from-human-preferences`

Main characters of RL: agent and environment

- **environment**: the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.
- **agent**: perceives a **reward** signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called **return**. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.



Key characteristics

Let's look at a few things that make reinforcement learning different from other paradigms of optimization, and other machine learning methods like supervised learning.

- Lack of a "supervisor": One of the main characteristic of RL is that there is no supervisor no labels telling us the best action to take, only reward signals to enforce some actions more than others. For example, for a robot trying to walk, there is no supervisor telling the robot if the actions it took were a good way to walk, but it does get some signals if form of the effect of its actions - moving forward or falling down which it can use to guide its behavior.

Key characteristics

- Delayed feedback: Other major distinction is that the feedback is often delayed: the effect of an action may not be entirely visible instantaneously, but it may severely affect the reward signal many steps later. In the robot example, an aggressive leg movement may look good right now as it may seem to make the robot go quickly towards the target, but a sequence of limb movements later you may realize that aggressive movement made the robot fall. This makes it difficult to attribute credit and reinforce a good move whose effect may be seen only many steps and many moves later. This is also referred to as the "credit assignment problem".
- Sequential decisions: Time really matters in RL, the “sequence” in which you make your decisions (moves) will decide the path you take and hence the final outcome.

Key characteristics

- Actions effect observations: Finally you can see from these examples that the observations or feedback that an agent makes during the course of learning are not really independent, in fact they are a function of the agents' own actions, which the agent may decide based on its past observations. This is very unlike other paradigms like supervised learning, where the training examples are often assumed to be independent of each other, or at least independent of learning agents' actions.

These are some key characteristics of reinforcement learning which differentiate it from the other branches of learning, and also make it a powerful model of learning for a variety of application domains.

Outline

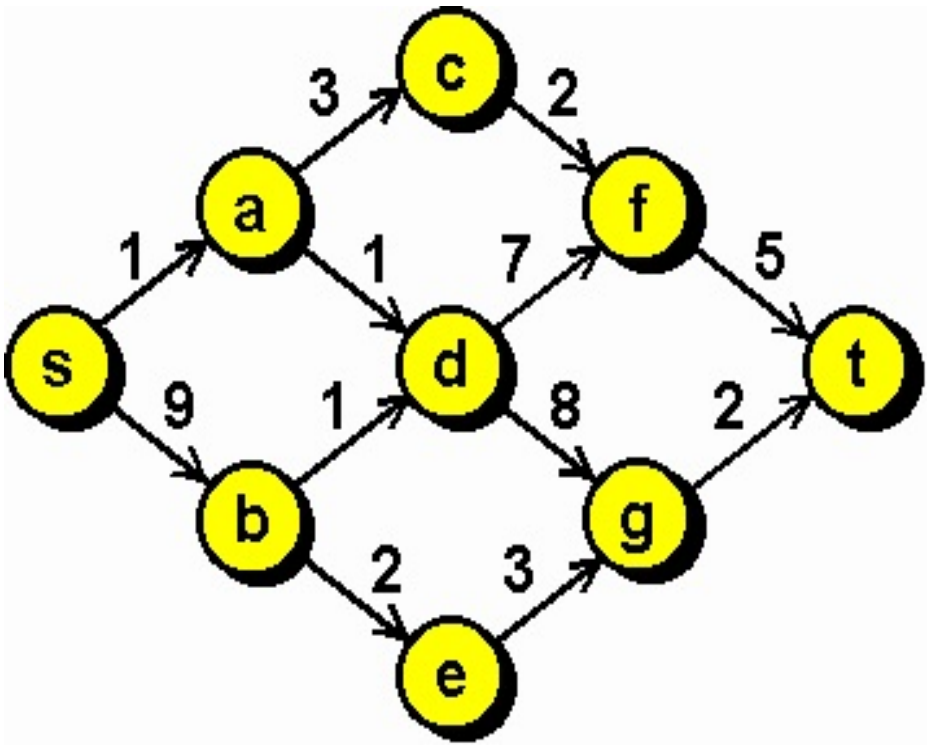
Dynamic Programming

- Basically, we want to solve a big problem that is hard
- We can first solve a few smaller but similar problems, if those can be solved, then the solution to the big problem will be easy to get
- To solve each of those smaller problems, we use the same idea, we first solve a few even smaller problems.
- Continue doing it, we will eventually encounter a problem we know how to solve

Dynamic programming has the same feature, the difference is that at each step, there might be some optimization involved.

Shortest Path Problem

You have a graph, you want to find the shortest path from s to t



Here we use d_{ij} to denote the distance between node i and node j

DP formulation for the shortest path problem

Let V_i denote the shortest distance between i to t.

- Eventually, we want to compute V_s
- It is hard to directly compute V_i in general
- However, we can just look one step

We know if the first step is to move from i to j, the shortest distance we can get must be $d_{ij} + V_j$.

- To minimize the total distance, we want to choose j to minimize $d_{ij} + V_j$

To write into a math formula, we get

$$V_i = \min_j \{d_{ij} + V_j\}$$

DP for shortest path problem

We call this the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i$$

We also know if we are already at our destination, then the distance is 0. i.e.,

$$V_t = 0$$

The above two equations are the DP formulation for this problem

Solve the DP

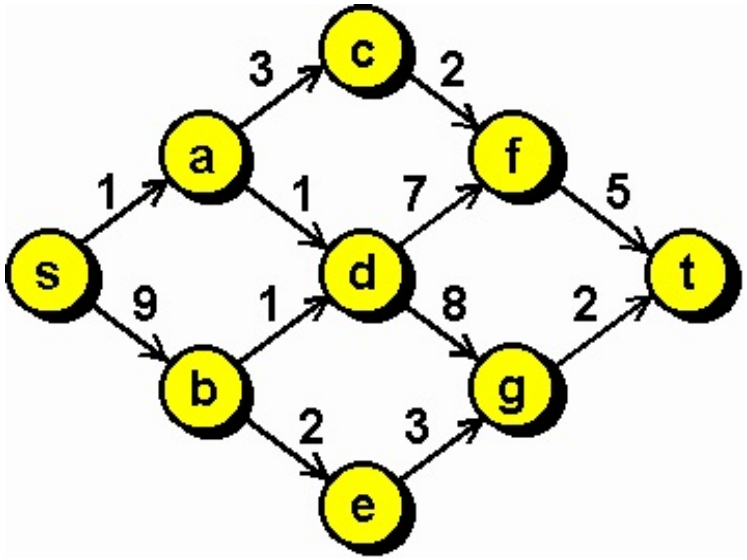
Given the formula, how to solve the DP?

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i, \quad V_t = 0$$

We use backward induction.

- From the last node (which we know the value), we solve the values of V's backwardly.

Example

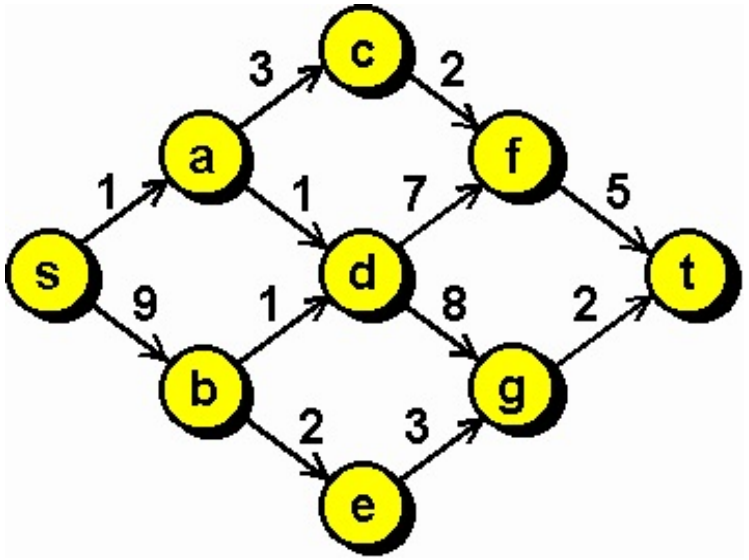


- We have $V_t = 0$. Then we have

$$V_f = \min_{(f,j) \text{ is a path}} \{d_{fj} + V_j\}$$

- Here, we only have one path, thus $V_f = 5 + V_t = 5$
- Similarly, $V_g = 2$

Example Continued 1



- We have $V_t = 0, V_f = 5$ and $V_g = 2$
- Now consider c, d, e. For c and e there is only one path

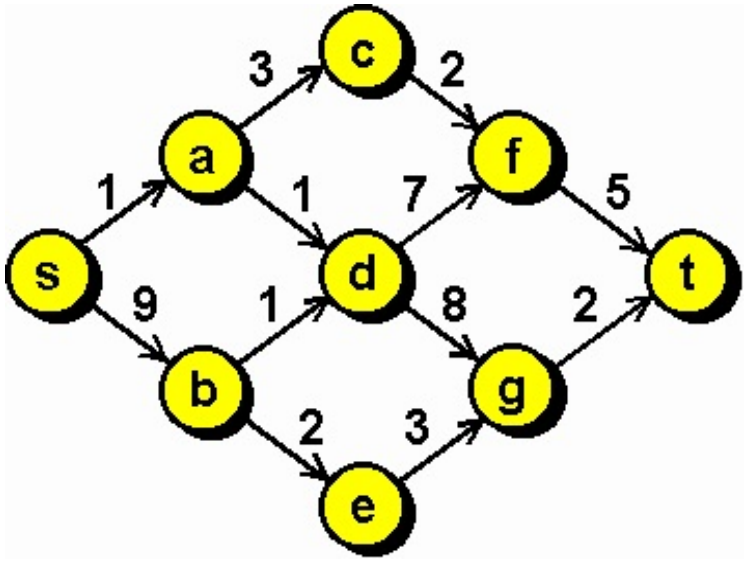
$$V_c = d_{cf} + V_f = 7, \quad V_e = d_{eg} + V_g = 5$$

- For d, we have

$$V_g = \min_{(d,j) \text{ is a path}} \{d_{dj} + V_j\} = \min\{d_{df} + V_f, d_{dg} + V_g\} = 10$$

- The optimal way to choose at d is go to g

Example Continued 2



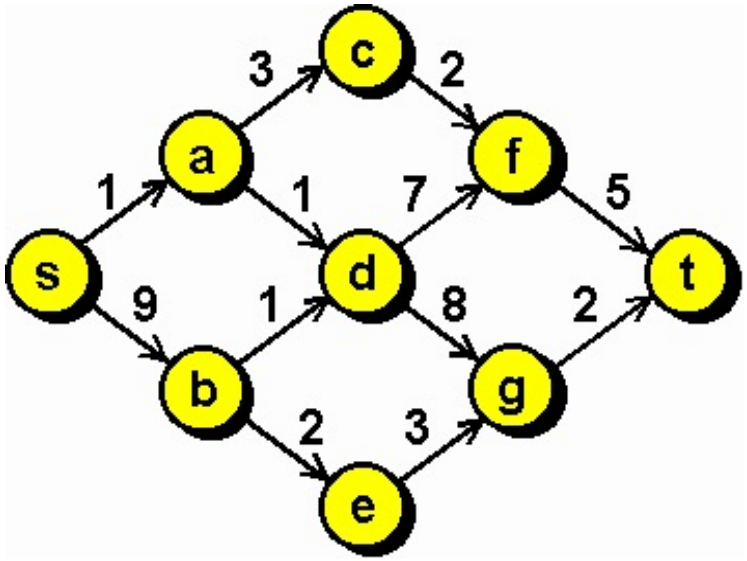
- We got $V_c = 7, V_d = 10$ and $V_e = 5$. Now we compute V_a and V_b

$$V_a = \min\{d_{ac} + V_c, d_{ad} + V_d\} = \min\{3 + 7, 1 + 10\} = 10$$

$$V_b = \min\{d_{bd} + V_d, d_{be} + V_e\} = \min\{1 + 10, 2 + 5\} = 7$$

and the optimal path to go at a is to choose c, and the optimal path to go at b is to choose e.

Example Continued 3



- Finally, we have

$$V_s = \min\{d_{sa} + V_a, d_{sb} + V_b\} = \min\{1 + 10, 9 + 7\} = 11$$

and the optimal path to go at s is to choose a

- Therefore, we found the optimal path is 11, and by connecting the optimal path, we get

$$s \to a \to c \to f \to t$$

Summary of the example

In the example, we saw that we have those V_i's, indicating the shortest length to go from i to t.

- We call this V the value function

We also have those nodes $s, a, b, \ldots, g, t$

- We call them the states of the problem
- The value function is a function of the state

And the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i$$

connects the value function at different states. It is known as the Bellman equation

Dynamic Programming Framework

The above is the general framework of dynamic programming problems.

To formulate a problem into a DP problem, the first step is to define the states

- The state variables should be in some order (usually either time order or geographical order)
- In the shortest path example, the state is simply each node

We call the set of all the state the state space. In this example, the state space is all the nodes.

Defining the appropriate state space is the most important step in DP.

Definition

A state is a collection of variables that summarize all the (historical) information that is useful for (future) decisions. Conditioned on the state, the problem becomes Markov (i.e., memoryless).

DP: Actions

There will be an action set at each state

- At state x, we denote the set by $A(x)$
- For each action $a \in A(x)$, there is an immediate cost $r(x; a)$
- If one exerts action a, the system will go to some next state $s(x; a)$
- In the shortest path problem, the action at each state is the path it can take. There is a distance for each path which is the immediate cost. And after you take a certain path, the system will be at the next node

DP: Value Functions

There is a value function $V(\cdot)$ at each state. The value function denotes that if you choose the optimal action from this state and onward, what is the optimal value.

- In the shortest path example, the value function is the shortest distance between the current state to the destination

Then a recursion formula can be written to link the value functions:

$$V(x) = \min_{a \in A(x)} \{r(x,a) + V(s(x,a))\}$$

In order to be able to solve the DP, one has to know some terminal values (boundary values) of this V function

- In the shortest path example, we know $V_t = 0$
- The recursion for value functions is known as the Bellman equation.

Some more general framework

The above framework is to minimize the total cost. In some cases, one wants to maximize the total profit. Then the $r(x, a)$ can be viewed as the immediate reward.

The DP in those cases can be written as

$$V(x) = \min_{a \in A(x)} \{r(x, a) + V(s(x, a))\}$$

with some boundary conditions

Stochastic DP

In some cases, when you choose action a at x, the next state is not certain (e.g., you decide a price, but the demand is random).

- There will be $p(x, y, a)$ probability you move from x to y if you choose action $a \in A(x)$

Then the recursion formula becomes:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \sum_{y} p(x, y, a) V(y)\}$$

or if we choose to use the expectation notation:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \mathbf{E}V(x, a))\}$$

Example: Stochastic Shortest Path Problem

Stochastic setting:

- One no longer controls which exact node to jump to next
- Instead one can choose between different actions $a \in \mathcal{A}$
- Each action a is associated with a set of transition probabilities $p(j|i;a)$ for all $i, j \in \mathcal{S}$.
- The arc length may be random w_{ija}

Objective:

- One needs to decide on the action for every possible current node. In other words, one wants to find a policy or strategy that maps from $\mathcal{S}$ to A.

Bellman Equation for Stochastic SSP:

$$V(i) = \min_a \sum_{j \in S} p(j|i;a)(w_{ija} + V(j)), \quad i \in \mathcal{S}$$

Bellman equation continued

We rewrite the Bellman equation

$$V(i) = \min_a \sum_{j \in \mathcal{S}} p(j|i;a)(w_{ija} + V(j)), \quad i \in \mathcal{S}$$

into a vector form

$$V = \min_{\mu: \mathcal{S} \to \mathcal{A}} g_\mu + P_\mu V, \quad \text{where}$$

- average transitional cost starting from i to the next state:

$$g_\mu(i) = \sum_{j \in \mathcal{S}} p(j|i, \mu(i)) w_{ij,\mu(i))}$$

- transition probabilities

$$P_\mu(i,j) = p(j|i, \mu(i))$$

- $V(i)$ is the expected length of stochastic shortest path starting from i, also known as the value function or cost-to-go vector
- V^* is the optimal value function or cost-to-go vector that solves the Bellman equation

More Applications of (Approximate) DP

Control of complex systems:

- Unmanned vehicle/aircraft, automated vehicle control
- Robotics
- Planning of power grid
- Smart home solution

Games:

- 2048, Go, Chess, chatbots
- Tetris, Poker

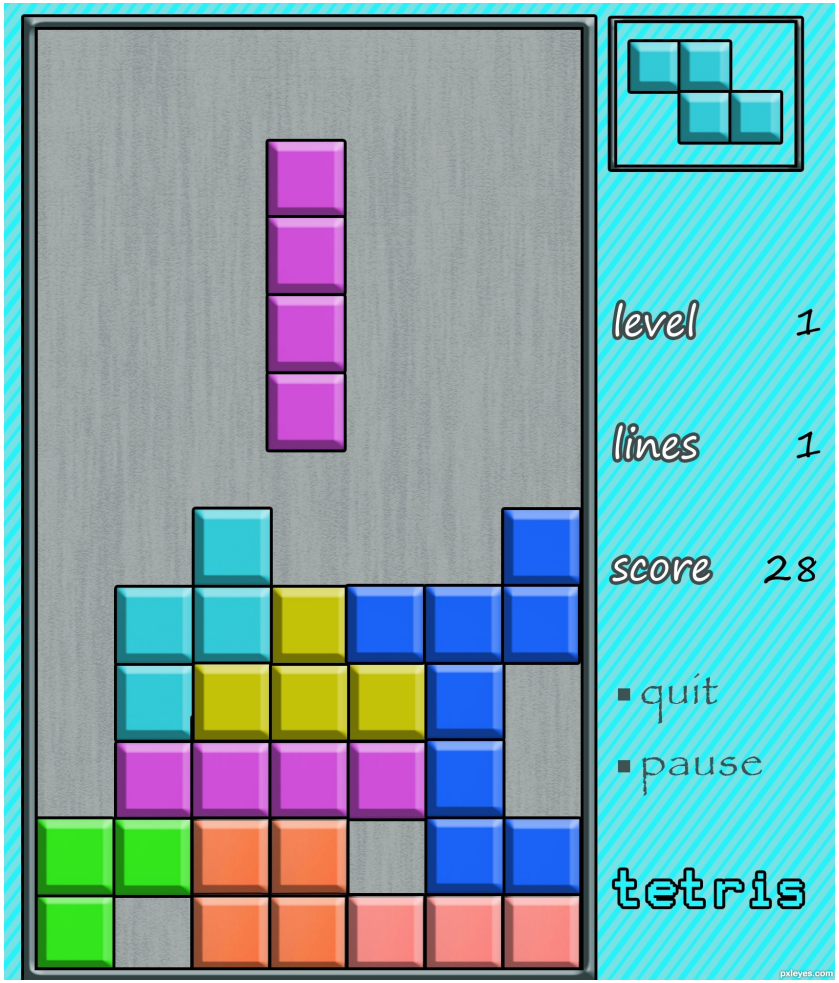
Business:

- Inventory and supply chain
- Dynamic pricing with demand learning
- Optimizing clinical pathway (healthcare)

Finance:

- Option pricing
- Optimal execution (especially in dark pools)
- High-frequency trading

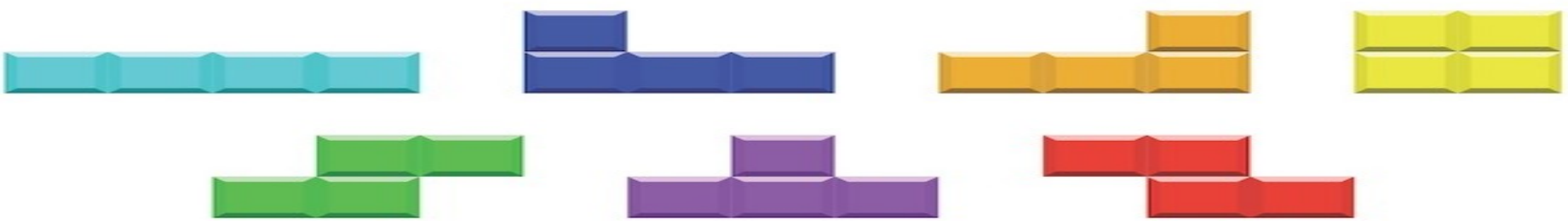
Tetris



- Height: 12
- Width: 7
- Rotate and move the falling shape
- Gravity related to current height
- Score when eliminating an entire level
- Game over when reaching the ceiling

DP Model of Tetris

- State: The current board, the current falling tile, predictions of future tiles
- Termination state: when the tiles reach the ceiling, the game is over with no more future reward
- Action: Rotation and shift
- System Dynamics: The next board is deterministically determined by the current board and the player's placement of the current tile. The future tiles are generated randomly.
- Uncertainty: Randomness in future tiles
- Transitional cost g: If a level is cleared by the current action, score 1; otherwise score 0.
- Objective: Expectation of total score.



Interesting facts about Tetris

- First released in 1984 by Alexey Pajitnov from the Soviet Union
- Has been proved to be NP-complete.
- Game will be over with probability 1.
- For a 12×7 board, the number of possible states $\approx 2^{12 \times 7} \approx 10^{25}$
- Highest score achieved by human $\approx$ 1 million
- Highest score achieved by algorithm $\approx$ 35 million (average performance)

Outline

States and Observations

- **state**: s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An **observation** o is a partial description of a state, which may omit information.
- In deep RL, we almost always represent states and observations by a "real-valued vector, matrix, or higher-order tensor". For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.
- When the agent is able to observe the complete state of the environment, we say that the environment is **fully observed**. When the agent can only see a partial observation, we say that the environment is **partially observed**.
- We often write that the action is conditioned on the state, when in practice, the action is conditioned on the observation because the agent does not have access to the state.

Action Spaces

- Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the **action space**. Some environments, like Atari and Go, have **discrete action spaces**, where only a finite number of moves are available to the agent. Other environments, like where the agent controls a robot in a physical world, have **continuous action spaces**. In continuous spaces, actions are real-valued vectors.
- This distinction has some quite-profound consequences for methods in deep RL. Some families of algorithms can only be directly applied in one case, and would have to be substantially reworked for the other.

Markov decision processes

- The defining property of MDPs is the Markov property which says that the future is independent of the past given the current state. This essentially means that the state in this model captures all the information from the past that is relevant in determining the future states and rewards.
- A Markov Decision Process (MDP) is specified by a tuple (S, s_1, A, P, R, H), where S is the set of states, s_1 is the starting state, A is the set of actions. The process proceeds in discrete rounds $t = 1, 2, \cdots, H$, starting in the initial state s_1. In every round, t the agent observes the current state $s_t \in S$, takes an action $a_t \in A$, and observes a feedback in form of a reward signal $r_{t+1} \in \mathbb{R}$. The agent then observes transition to the next state $s_{t+1} \in S$.

Formal definition

- The probability of transitioning to a particular state depends only on current state and action, and not on any other aspect of the history. The matrix $P \in [0,1]^{S \times A \times S}$ specifies these probabilities. That is,

$$\begin{aligned} Pr(s_{t+1} = s' \mid \textit{history till time } t) &= Pr(s_{t+1} = s' \mid s_t = s, a_t = a) \\ &= P(s, a, s') \end{aligned}$$

- The reward distribution depends only on the current state and action. So, that the expected reward at time t is a function of current state and action. A matrix R specifies these rewards.

$$\mathbb{E}[r_{t+1} \mid \textit{history till time } t] = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = R(s, a)$$

- Let $R(s, a, s')$ be the expected (or deterministic) reward when action a is taken in state s and transition to state s' is observed. Then, we can obtain the same model as above by defining

$$R(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = \mathbb{E}_{s' \sim P(s,a)}[R(s, a, s')]$$

Policy

- A policy specifies what action to take at any time step. A history dependent policy at time t is a mapping from history till time t to an action. A Markovian policy is a mapping from state space to action π: $S \to A$. Due to Markovian property of the MDP, it suffices to consider Markovian policies (in the sense that for any history dependent policy same performance can be achieved by a Markovian policy). Therefore, in this text, policy refers to a Markovian policy.
- A deterministic policy π: $S \to A$ is mapping from any given state to an action. A randomized policy $\pi : S \to \Delta^A$ is a mapping from any given state to a distribution over actions. Following a policy π_t at time t means that if the current state $s_t = s$, the agent takes action $a_t = \pi_t(s)$ (or $a_t \sim \pi(s)$ for randomized policy). Following a stationary policy π means that $\pi_t = \pi$ for all rounds $t = 1, 2, \ldots$

Policy

- Any stationary policy π defines a Markov chain, or rather a 'Markov reward process' (MRP), that is, a Markov chain with reward associated with every transition.
- The transition probability vector and reward for this MRP in state s is given by $\Pr\left(s'|s\right) = P_s^\pi, \mathbb{E}\left[r_t|s\right] = r_s^\pi$, where P^π is an $S \times S$ matrix, and r^π is an S-dimensional vector defined as:

$$P_{s,s'}^\pi = \mathbb{E}_{a\sim\pi(s)}\left[P\left(s,a,s'\right)\right], \forall s,s' \in S$$

$$r_s^\pi = \mathbb{E}_{a\in\pi(s)}[R(s,a)]$$

- The stationary distribution (if exists) of this Markov chain when starting from state s_1 is also referred to as the stationary distribution of the policy π, denoted by d^π:

$$d^\pi(s) = \lim_{t\to\infty} \Pr\left(s_t = s|s_1,\pi\right)$$

Goals, finite horizon MDP

- The tradeoffs between immediate reward vs. future rewards of the sequential decisions and the need for planning ahead is captured by the goal of the Markov Decision Process. At a high level, the goal is to maximize some form of cumulative reward. Some popular forms are total reward, average reward, or discounted sum of rewards.

finite horizon MDP

- actions are taken for $t = 1, \ldots, H$ where H is a finite horizon. The total (discounted) reward criterion is simply to maximize the expected total (discounted) rewards in an episode of length H. (In reinforcement learning context, when this goal is used, the MDP is often referred to as an episodic MDP.) For discount $0 \leq \gamma \leq 1$, the goal is to maximize

$$\mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | s_1\right]$$

Infinite horizon MDP

- Expected total discounted reward criteria: The most popular form of cumulative reward is expected discounted sum of rewards. This is an asymptotic weighted sum of rewards, where with time the weights decrease by a factor of $\gamma < 1$. This essentially means that the immediate returns more valuable than those far in the future.

$$\lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1\right]$$

Infinite horizon MDP

- Expected total reward criteria: Here, the goal is to maximize

$$\lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} r_t | s_1\right]$$

The limit may not always exist or be bounded. We are only interested in cases where above exists and is finite. This requires restrictions on reward and/or transition models. Interesting cases include the case where there is an undesirable state, the reward after reaching that state is 0. For example, end of a computer game. The goal would be to maximize the time to reach this state. (A minimization version of this model is where there is a cost associated with each state and the game is to minimize the time to reach winning state, called the shortest path problem).

Infinite horizon MDP

- Expected average reward criteria: Maximize

$$\lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

Intuitively, the performance in a few initial rounds does not matter here, what we are looking for is a good asymptotic performance. This limit may not always exist. Assuming bounded rewards and finite state spaces, it exists under some further conditions on policy used.

Advantages of Discounted sum of rewards

- it is mathematically convenient as it is always finite and avoids the complications due to infinite returns. Practically, depending on the application, immediate rewards may indeed be more valuable.
- Further, often uncertainty about far future are not well understood, so you may not want to give as much weight to what you think you might earn far ahead in future. The discounted reward criteria can also be seen as a soft version of finite horizon, as the contribution of reward many time steps later is very small.
- As you will see later, discounted reward MDP has many desirable properties for iterative algorithm design and learning. Due to these reasons, often the practical approaches which actually execute the MDP for finite horizon, use policies, algorithms and insights from infinite horizon discounted reward setting.

Gain of the MDP

Gain (roughly the 'expected value objective' or formal goal) of an MDP when starting in state s_1 is defined as (when supremum exists):

- episodic MDP:

$$J(s_1) = \sup_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | s_1\right]$$

- Infinite horizon expected total reward:

$$J(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} r_t | s_1\right]$$

- Infinite horizon discounted sum of rewards:

$$J(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1\right]$$

Gain of the MDP

- infinite horizon average reward:

$$J(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

Here, expectation is taken with respect to state transition and reward distribution, supremum is taken over all possible sequence of policies for the given MDP. It is also useful to define gain ρ^π of a stationary policy π, which is the expected (total/total discounted/average) reward when policy π is used in all time steps. For example, for infinite average reward:

$$J^\pi(s_1) = \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

where $a_t = \pi(s_t), t = 1, \ldots, T$

Optimal policy

- Optimal policy is defined as the one that maximizes the gain of the MDP.
- Due to the structure of MDP it is not difficult to show that it is sufficient to consider Markovian policies. Henceforth, we consider only Markovian policies.
- For infinite horizon MDP with average/discounted reward criteria, a further observation that comes in handy is that such a MDP always has a stationary optimal policy, whenever optimal policy exists. That is, there always exists a fixed policy so that taking actions specified by that policy at all time steps maximizes average/discounted/total reward.
- The agent does not need to change policies with time. This insight reduces the question of finding the best sequential decision making strategy to the question of finding the best stationary policy.

Optimal policy

The results below assume finite state, action space and bounded rewards.

Theorem 1 (Puterman [1994], Theorem 6.2.7)

For any infinite horizon discounted MDP, there always exists a deterministic stationary policy π that is optimal.

Theorem 2 (Puterman [1994], Theorem 7.1.9)

For any infinite horizon expected total reward MDP, there always exists a deterministic stationary policy π that is optimal.

Theorem 3 (Puterman [1994], Theorem 8.1.2)

For infinite horizon average reward MDP, there always exist a stationary (possibly randomized) policy which is an optimal policy.

Optimal policy

Therefore, for infinite horizon MDPs, optimal gain:

$$J^*(s) = \max_{\pi\text{: Markovian stationary}} J^\pi(s)$$

(limit exists for stationary policies [Puterman Proposition 8.1.1])

- These results imply that the optimal solution space is simpler for infinite horizon case, and make infinite horizon an attractive model even when the actual problem is finite horizon but the horizon is long.
- Even when such a result on optimality of stationary policy is not available, 'finding the best stationary policy' is often used as an alternate convenient and more tractable objective, instead of finding the optimal policy which may not exist or may not be stationary in general.

Outline

Value Functions

It's often useful to know the **value** of a state, or state-action pair. By value, we mean the expected return if you start in that state or state-action pair, and then act according to a particular policy forever after. **Value functions** are used, one way or another, in almost every RL algorithm.

- The **On-Policy Value Function** $V^{\pi}(s)$, which gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s\right]$$

- The **On-Policy Action-Value Function** $Q^{\pi}(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s, a) = \lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s, a_1 = a\right]$$

Value Functions

- The **Optimal Value Function** $V^*(s)$, which gives the expected return if you start in state s and always act according to the *optimal* policy in the environment:

$$V^*(s) = \max_\pi V^\pi(s)$$

- The **Optimal Action-Value Function**, $Q^*(s,a)$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the *optimal* policy in the environment:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

Value Functions

- When we talk about value functions, if we do not make reference to time-dependence, we only mean expected **infinite-horizon discounted return**. Value functions for finite-horizon undiscounted return would need to accept time as an argument. Can you think about why? Hint: what happens when time's up?
- There are two key connections between the value function and the action-value function that come up pretty often:

$$V^{\pi}(s) = \mathop{\mathbf{E}}_{a \sim \pi} \left[Q^{\pi}(s,a) \right],$$

and

$$V^{*}(s) = \max_{a} Q^{*}(s,a).$$

These relations follow pretty directly from the definitions just given: can you prove them?

The Optimal Q-Function and the Optimal Action

- There is an important connection between the optimal action-value function $Q^*(s, a)$ and the action selected by the optimal policy. By definition, $Q^*(s, a)$ gives the expected return for starting in state s, taking (arbitrary) action a, and then acting according to the optimal policy forever after.
- The optimal policy in s will select whichever action maximizes the expected return from starting in s. As a result, if we have Q^*, we can directly obtain the optimal action, $a^*(s)$, via

$$a^*(s) = \arg\max_a Q^*(s, a).$$

- Note: there may be multiple actions which maximize $Q^*(s, a)$, in which case, all of them are optimal, and the optimal policy may randomly select any of them. But there is always an optimal policy which deterministically selects an action.

Bellman Equations

- All four of the value functions obey special self-consistency equations called **Bellman equations**. The basic idea is: *The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.*
- The Bellman equations for the on-policy value functions are

$$V^{\pi}(s) = \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[R(s,a,s') + \gamma V^{\pi}(s')\right],$$

$$Q^{\pi}(s,a) = \mathop{\mathbf{E}}_{s'\sim P} \left[R(s,a,s') + \gamma \mathop{\mathbf{E}}_{a'\sim\pi} \left[Q^{\pi}(s',a')\right]\right],$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$, indicating that the next state s' is sampled from the environment's transition rules; $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$; and $a' \sim \pi$ is shorthand for $a' \sim \pi(\cdot|s')$.

Proof of Bellman equations

Proof. $V^\pi = R^\pi + \gamma P^\pi V^\pi$:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots | s_1 = s\right] \\ &= \mathbb{E}\left[r_1 | s_1 = s\right] + \gamma \mathbb{E}\left[\mathbb{E}\left[r_2 + \gamma r_3 + \gamma^2 r_4 + \ldots | s_2\right] | s_1 = s\right] \end{aligned}$$

The first term here is simply the expected reward in state s when action is given by $\pi(s)$. The second term is γ times the value function at $s_2 \sim P(s, \pi(s), \cdot)$

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[R\left(s, \pi(s), s_1\right) + \gamma V^\pi\left(s_2\right) | s_1 = s\right] \\ &= R(s, \pi(s)) + \gamma \sum_{s_2 \in S} P\left(s, \pi(s), s_2\right) V^\pi\left(s_2\right) \\ &= R^\pi(s) + \gamma \left[P^\pi V^\pi\right](s) \end{aligned}$$

Bellman Optimal Equations

- The Bellman equations for the optimal value functions are

$$V^*(s) = \max_a \mathop{\mathbf{E}}_{s' \sim P} \left[R(s,a) + \gamma V^*(s') \right],$$

$$Q^*(s,a) = \mathop{\mathbf{E}}_{s' \sim P} \left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \right].$$

  The crucial difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the $\max$ over actions. Its inclusion reflects the fact that whenever the agent gets to choose its action, in order to act optimally, it has to pick whichever action leads to the highest value.

- The term "Bellman backup" comes up quite frequently in the RL literature. The Bellman backup for a state, or state-action pair, is the right-hand side of the Bellman equation: the reward-plus-next-value.

Proof of Bellman optimality equations

Proof. for all s, from the theorem ensuring stationary optimal policy:

$$\begin{aligned} V^*(s) = \max_\pi V^\pi(s) &= \max_\pi \mathbb{E}_{a\sim\pi(s),s'\sim P(s,a)}\left[R\left(s,a,s'\right)+\gamma V^\pi\left(s'\right)\right] \\ &\leq \max_a R(s,a)+\gamma\sum_{s'} P\left(s,a,s'\right)\max_\pi V^\pi\left(s'\right) \\ &= \max_a R(s,a)+\gamma\sum_{s'} P\left(s,a,s'\right)V^*\left(s'\right) \end{aligned}$$

Now, if the above inequality is strict then the value of state s can be improved by using a (possibly non-stationary) policy that uses action $\arg\max_a R(s,a)$ in the first step. This is a contradiction to the definition $V^*(s)$. Therefore,

$$V^*(s) = \max_a R(s,a)+\gamma\sum_{s'} P\left(s,a,s'\right)V^*\left(s'\right)$$

Bellman optimality equations

- Technically, above only shows that V^* satisfies the Bellman equations.
- Theorem 6.2.2 (c) in Puterman [1994] shows that V^* is in fact unique solution of above equations.
- Therefore, satisfying these equations is sufficient to guarantee optimality, so that it is not difficult to see that the deterministic (stationary) policy

$$\pi^*(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V^*\left(s'\right)$$

is optimal (see Puterman [1994] Theorem 6.2.7 for formal proof).

Linear programming

The fixed point for above Bellman optimality equations can be found by formulating a linear program. It amounts to :

$$\begin{aligned} \min_{\mathbf{v} \in \mathbb{R}^S} \quad & \sum_s w_s \mathbf{v}_s \\ \text{s.t.} \quad & \mathbf{v}_s \geq R(s,a) + \gamma P(s,a)^\top \mathbf{v} \quad \forall a, s \end{aligned}$$

Proof. V^* clearly satisfies the constraints of the above LP. Next, we show that $\mathbf{v} = V^*$ minimizes the obj. fun. The constraint implies that

$$\mathbf{v}_s \geq R\left(s, \pi^*(s)\right) + \gamma P\left(s, \pi^*(s)\right)^\top \mathbf{v}, \forall s$$

(Above is written assuming π^* is deterministic, which is in fact true in the infinite horizon discounted reward case.) Or,

$$\left(I - \gamma P^{\pi^*}\right) \mathbf{v} \geq R^{\pi^*}$$

Proof

Because $\gamma < 1, (I - \gamma P^{\pi})^{-1}$ exists for all π, and for any $u \geq 0$

$$(I - \gamma P^{\pi})^{-1} u = \left(I + \gamma P^{\pi} + \gamma^2 (P^{\pi})^2 + \cdots\right) u \geq u$$

Therefore, from above

$$\left(I - \gamma P^{\pi^*}\right)^{-1} \left(\left(I - \gamma P^{\pi^*}\right) \mathbf{v} - R^{\pi^*}\right) \geq 0$$

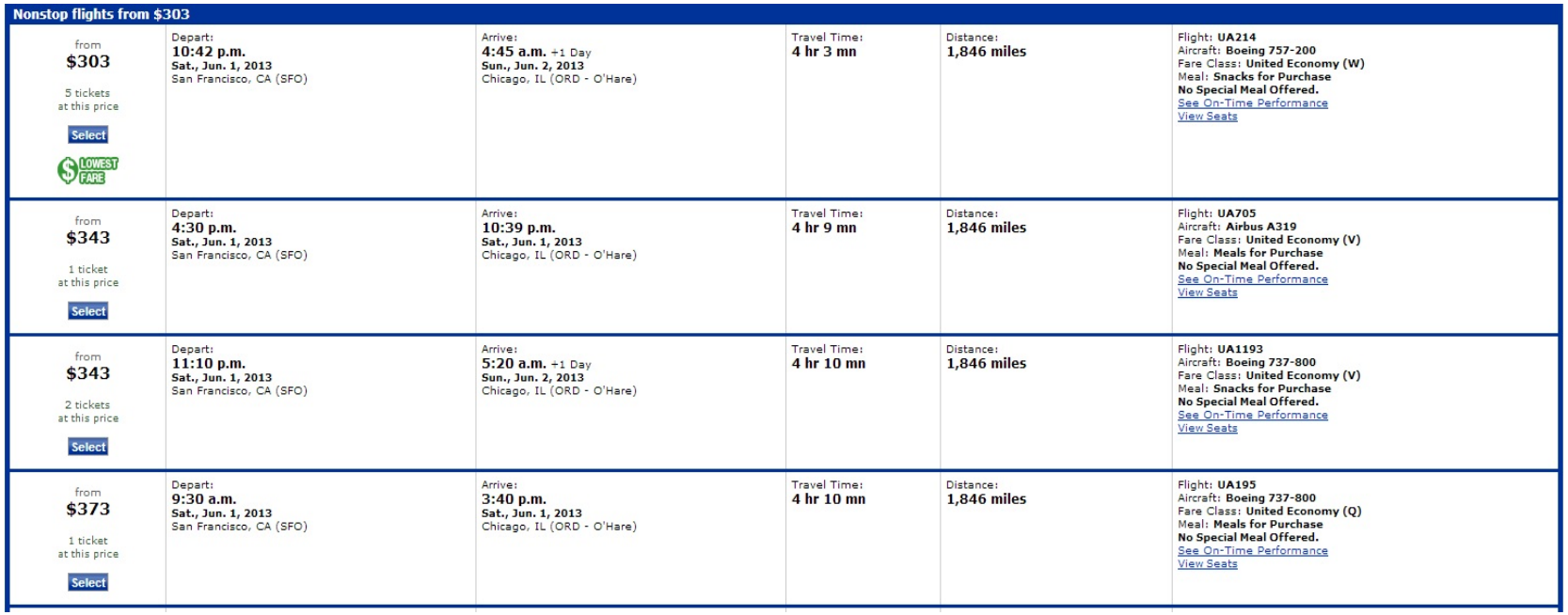
Or,

$$\mathbf{v} \geq \left(I - \gamma P^{\pi^*}\right)^{-1} R^{\pi^*} = V^*$$

Therefore, $\mathbf{w}^{\top} \mathbf{v}$ for $w > 0$ is minimized by $\mathbf{v} = V^*$.

Outline

An Example in Revenue Management: Airfare Pricing



Nonstop flights from $303

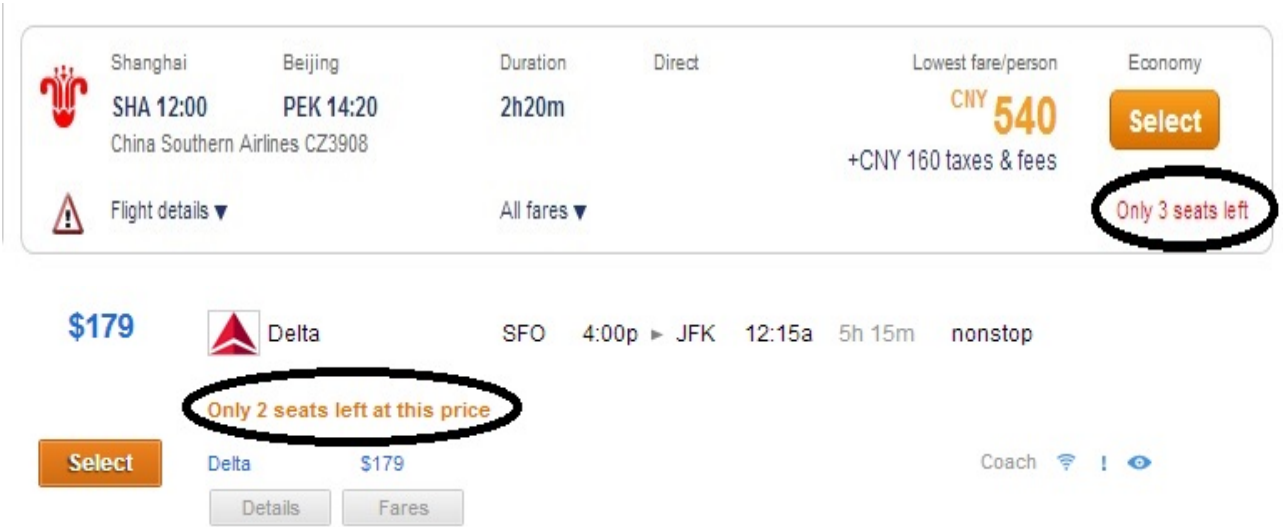
	Depart:	Arrive:	Travel Time:	Distance:	
from **$303** 5 tickets at this price Select LOWEST FARE	**10:42 p.m.** Sat., Jun. 1, 2013 San Francisco, CA (SFO)	**4:45 a.m.** +1 Day Sun., Jun. 2, 2013 Chicago, IL (ORD - O'Hare)	**4 hr 3 mn**	**1,846 miles**	Flight: **UA214** Aircraft: **Boeing 757-200** Fare Class: **United Economy (W)** Meal: **Snacks for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$343** 1 ticket at this price Select	**4:30 p.m.** Sat., Jun. 1, 2013 San Francisco, CA (SFO)	**10:39 p.m.** Sat., Jun. 1, 2013 Chicago, IL (ORD - O'Hare)	**4 hr 9 mn**	**1,846 miles**	Flight: **UA705** Aircraft: **Airbus A319** Fare Class: **United Economy (V)** Meal: **Meals for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$343** 2 tickets at this price Select	**11:10 p.m.** Sat., Jun. 1, 2013 San Francisco, CA (SFO)	**5:20 a.m.** +1 Day Sun., Jun. 2, 2013 Chicago, IL (ORD - O'Hare)	**4 hr 10 mn**	**1,846 miles**	Flight: **UA1193** Aircraft: **Boeing 737-800** Fare Class: **United Economy (V)** Meal: **Snacks for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$373** 1 ticket at this price Select	**9:30 a.m.** Sat., Jun. 1, 2013 San Francisco, CA (SFO)	**3:40 p.m.** Sat., Jun. 1, 2013 Chicago, IL (ORD - O'Hare)	**4 hr 10 mn**	**1,846 miles**	Flight: **UA195** Aircraft: **Boeing 737-800** Fare Class: **United Economy (Q)** Meal: **Meals for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats

The price corresponding to each fare class rarely changes (this is determined by other department), however, the RM department determines when to close low fare classes

- From the passenger's point of view, when the RM system closes a class, the fare increases
- Closing fare class achieves dynamic pricing

Fare classes

And when you make booking, you will frequently see messages like



This is real. It means there are only that number of tickets at that fare class (there is one more sale that will trigger the next protection level)

- You can try to buy one ticket with only one remaining, and see what happens

Dynamic Arrival of Consumers

Assumptions

- There are T periods in total indexed forward (the first period is 1 and the last period is T)
- There are C inventory at the beginning
- Customers belong to n classes, with $p_1 > p_2 > ... > p_n$
- In each period, there is a probability λ_i that a class i customer arrives
- Each period is small enough so that there is at most one arrival in each period

Decisions

- When at period t and when you have x inventory remaining, which fare class should you accept (if such a customer comes)
- Instead of finding a single optimal price or reservation level, we now seek for a decision rule, i.e., a mapping from (t, x) to $\{I | I \subset \{1, ..., n\}\}$.

Dynamic Arrival - a T-stage DP problem

- State: Inventory level x_k for stages $k = 1, ..., T$
- Action: Let $u^{(k)} \in \{0, 1\}^n$ to be the decision variable at period k

$$u_i^{(k)} = \begin{cases} 1 & \text{accept class i customer} \\ 0 & \text{reject class i customer} \end{cases}$$

  decision vector $u^{(k)}$ at stage k, where $u_i^{(k)}$ decides whether to accept the ith class

- Random disturbance: Let $w_k, k \in \{0, ..., T\}$ denotes the type of new arrival during the kth stage (type 0 means no arrival). Then $P(w_k = i) = \lambda_i$ for $k = 1, \ldots, T$ and $P(w_k = 0) = 1 - \sum_{i=1}^{n} \lambda_i$

Value Function: A Rigorous Definition

- State transition cost:

$$g_k(x_k, u^{(k)}, w_k) = u^{(k)}_{w_k} p_{w_k}$$

where we take $p_0 = 0$. Clearly, $\mathbf{E}[g_k(x_k, u^{(k)}, w_k)|x_k] = \sum_{i=1}^{n} u_i^{(k)} p_i \lambda_i$

- State transition dynamics

$$x_{k+1} = \begin{cases} x_k - 1 & \text{if } u^{(k)}_{w_k} w_k \neq 0 \text{ (with probability } \sum_{i=1}^{n} u_i^{(k)} \lambda_i) \\ x_k & \text{otherwise (with probability } 1 - \sum_{i=1}^{n} u_i^{(k)} \lambda_i) \end{cases}$$

- The overall revenue is

$$\max_{\mu_1, \ldots, \mu_T} \mathbf{E}\left[\sum_{k=0}^{T} g_k(x_k, \mu_k(x_k), w_k)\right]$$

subject to the $\mu_k : x \to \{u\}$ for all k

A Dynamic Programming Model

- Let $V_t(x)$ denote the optimal revenue one can earn (by using the optimal policy onward) starting at time period t with inventory x

$$V_t(x) = \max_{\mu_t,\ldots,\mu_T} \mathbf{E}\left[\sum_{k=t}^{T} g_k(x_k, \mu_k(x_k), w_k) | x_t = x\right]$$

- We call $V_t(x)$ the value function (a function of stage t and state x)
- Suppose that we know the optimal pricing strategy from time $t+1$ for all possible inventory levels x.
- More specifically, suppose that we know $V_{t+1}(x)$ for all possible state x. Now let us find the best decisions at time t.

Bellman's Equation for Dynamic Arrival Model

We just proved the Bellman's equation. In the airfare model, Bellman's equation is

$$V_t(x) = \max_u \left\{ \sum_{i=1}^{n} \lambda_i (p_i u_i + u_i V_{t+1}(x-1)) + (1 - \sum_{i=1}^{n} \lambda_i u_i) V_{t+1}(x) \right\}$$

with $V_{T+1}(x) = 0$ for all x and $V_t(0) = 0$ for all t

We can rewrite this as

$$V_t(x) = V_{t+1}(x) + \max_u \left\{ \sum_{i=1}^{n} \lambda_i u_i (p_i + V_{t+1}(x-1) - V_{t+1}(x)) \right\}$$

For every (t, x), we have an equality and an unknown. The Bellman equation bears a unique solution.

Dynamic Programming Analysis

$$V_t(x) = V_{t+1}(x) + \max_u \left\{ \sum_{i=1}^{n} \lambda_i u_i (p_i - \Delta V_{t+1}(x)) \right\}$$

Therefore the optimal decision at time t with inventory x should be

$$u_i^* = \begin{cases} 1 & p_i \geq \Delta V_{t+1}(x) \\ 0 & p_i < \Delta V_{t+1}(x) \end{cases}$$

This is also called bid-price control policy

- The bid-price is $\Delta V_{t+1}(x)$
- If the customer pays more than the bid-price, then accept
- Otherwise reject

Dynamic Programming Analysis

Of course, to implement this strategy, we need to know $\Delta V_{t+1}(x)$

- We can compute all the values of $V_{t+1}(x)$ backwards
- Computational complexity is $O(nCT)$
- With those, we can have a whole table of $V_{t+1}(x)$. And we can execute based on that

Proposition (Properties of the Bid-prices)

For any x and t, i) $\Delta V_t(x+1) \leq \Delta V_t(x)$, ii) $\Delta V_{t+1}(x) \leq \Delta V_t(x)$

Intuitions:

- Fixed t, the value of the inventory has decreasing marginal returns
- The more time one has, the more valuable an inventory worth
- Proof by induction using the DP formula

Outline

Value Iteration

- Indirect method that finds optimal value function (value vector v in above), not explicit policy.

Pseudocode

- Start with an arbitrary initialization v^0. Specify $\epsilon > 0$
- **Repeat** for $k = 1, 2, \ldots$ **until** $\left\|\mathbf{v}^k(s) - \mathbf{v}^{k-1}(s)\right\|_\infty \leq \epsilon \frac{(1-\gamma)}{2\gamma}$:
  - for every $s \in S$, improve the value vector as:

$$\mathbf{v}^k(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s'} P\left(s, a, s'\right) \mathbf{v}^{k-1}\left(s'\right) \tag{1}$$

- Compute optimal policy as

$$\pi(s) \in \arg\max_a R(s,a) + \gamma P(s,a)^\top \mathbf{v}^k \tag{2}$$

Bellman operator

- It is useful to represent the iterative step (1) using operator $\mathbf{L} : \mathbb{R}^S \to \mathbb{R}^S$.

$$\mathbf{L}V(s) := \max_{a \in A} R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V\left(s'\right)$$

$$\mathbf{L}^{\pi}V(s) := \mathbb{E}_{a \in \pi(s)}\left[R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V\left(s'\right)\right] \tag{3}$$

- Then, (1) is same as

$$\mathbf{v}^k = \mathbf{L}\mathbf{v}^{k-1} \tag{4}$$

- For any policy π, if V^{π} denotes its value function, then, by Bellman equations:

$$V^* = \mathbf{L}V^*, V^{\pi} = \mathbf{L}^{\pi}V^{\pi} \tag{5}$$

Bellman operator

Below is a useful 'contraction' property of the Bellman operator, which underlies the convergence properties of all DP based iterative algorithms.

Lemma 6

The operator $\mathbf{L}(\cdot)$ and $\mathbf{L}^{\pi}(\cdot)$ defined by (3) are contraction mappings, i.e.,

$$\|\mathbf{L}v - \mathbf{L}u\|_{\infty} \leq \gamma \|v - u\|_{\infty}$$

$$\|\mathbf{L}^{\pi}v - \mathbf{L}^{\pi}u\|_{\infty} \leq \gamma \|v - u\|_{\infty}$$

Proof of contraction

Proof. First assume $\mathbf{L}v(s) \geq \mathbf{L}u(s)$.
Let $a_s^* = \arg\max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s')\, v(s')$

$$
\begin{aligned}
0 &\leq \mathbf{L}v(s) - \mathbf{L}u(s) \\
&\leq R(s,a_s^*) + \gamma \sum_{s'} P(s,a_s^*,s')\, v(s') - R(s,a_s^*) - \gamma \sum_{s'} P(s,a_s^*,s')\, u(s') \\
&= \gamma P(s,a_s^*)^\top (v-u) \\
&\leq \gamma \|v-u\|_\infty
\end{aligned}
$$

Repeating a symmetric argument for the case $\mathbf{L}v(s) \geq \mathbf{L}u(s)$ gives the lemma statement. Similar proof holds for $\mathbf{L}^\pi$.

Convergence

Theorem 7 (Theorem 6.3.3, Section 6.3.2 in Puterman [1994])

The convergence rate of the above algorithm is linear at rate γ. Specifically,

$$\left\|\mathbf{v}^k - V^*\right\|_\infty \leq \frac{\gamma^k}{1-\gamma}\left\|\mathbf{v}^1 - \mathbf{v}^0\right\|_\infty$$

Further, let π^k be the policy given by (2) using v^k. Then,

$$\left\|V^{\pi^k} - V^*\right\|_\infty \leq \frac{2\gamma^k}{1-\gamma}\left\|v^1 - v^0\right\|_\infty$$

Proof of Convergence

Proof. By Bellman equations $V^* = \mathbf{L}V^*$

$$\begin{aligned}
\left\|V^* - \mathbf{v}^k\right\|_\infty &= \left\|\mathbf{L}V^* - v^k\right\|_\infty \\
&\leq \left\|\mathbf{L}V^* - \mathbf{L}v^k\right\|_\infty + \left\|\mathbf{L}v^k - v^k\right\|_\infty \\
&= \left\|\mathbf{L}V^* - \mathbf{L}v^k\right\|_\infty + \left\|\mathbf{L}v^k - \mathbf{L}v^{k-1}\right\|_\infty \\
&\leq \gamma \left\|V^* - v^k\right\| + \gamma \left\|v^k - v^{k-1}\right\| \\
&\leq \gamma \left\|V^* - v^k\right\| + \gamma^k \left\|v^1 - v^0\right\| \\
\left\|V^* - v^k\right\|_\infty &\leq \frac{\gamma^k}{1-\gamma} \left\|v^1 - v^0\right\|
\end{aligned}$$

Let $\pi = \pi^k$ be the policy at the end of k iterations. Then, $V^\pi = \mathbf{L}^\pi V^\pi$ by Bellman equations. Further, by definition of $\pi = \pi^k$,

$$\mathbf{L}^\pi v^k(s) = \max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) v^k\left(s'\right) = \mathbf{L}v^k(s)$$

Proof of Convergence

Therefore,

$$
\begin{aligned}
\left\|V^{\pi}-v^{k}\right\|_{\infty} &= \left\|\mathbf{L}^{\pi}V^{\pi}-v^{k}\right\|_{\infty} \\
&\leq \left\|\mathbf{L}^{\pi}V^{\pi}-\mathbf{L}^{\pi}v^{k}\right\|_{\infty}+\left\|\mathbf{L}^{\pi}v^{k}-v^{k}\right\|_{\infty} \\
&= \left\|\mathbf{L}^{\pi}V^{\pi}-\mathbf{L}^{\pi}v^{k}\right\|_{\infty}+\left\|\mathbf{L}v^{k}-\mathbf{L}v^{k-1}\right\|_{\infty} \\
&\leq \gamma\left\|V^{\pi}-v^{k}\right\|+\gamma\left\|v^{k}-v^{k-1}\right\| \\
\left\|V^{\pi}-v^{k}\right\|_{\infty} &\leq \frac{\gamma}{1-\gamma}\left\|v^{k}-v^{k-1}\right\| \\
&\leq \frac{\gamma^{k}}{1-\gamma}\left\|v^{1}-v^{0}\right\|
\end{aligned}
$$

Adding the two results above:

$$
\left\|V^{\pi}-V^{*}\right\|_{\infty} \leq \frac{2\gamma^{k}}{1-\gamma}\left\|v^{1}-v^{0}\right\|_{\infty}
$$

Convergence

- In average reward case, the algorithm is similar, but the Bellman operator used to update the values is now $\mathbf{L}V(s) = \max_a r_{s,a} + P(s,a)^\top V$. Also, here $\mathbf{v}^k$ will converge to $\mathbf{v}^* + c\mathbf{e}$ for some constant c. Therefore, the stopping condition used is instead $sp\left(v^k - v^{k-1}\right) \leq \epsilon$ where $\mathrm{sp}(v) := \max_s v_s - \min_s v_s$. That is, span is used instead of L_∞ norm. Further since there is no discount $(\gamma = 1)$, a condition on the transition matrix is required to prove convergence. Let

$$\gamma := \max_{s,s',a,a'} 1 - \sum_{j \in S} \min\left\{P(s,a,j), P\left(s',a',j\right)\right\}$$

- Then, linear convergence with rate γ is guaranteed if $\gamma < 1$. This condition ensures that the Bellman operator in this case: is still a contraction. For more details, refer to Section 8.5.2 in Puterman [1994].

Q-value iteration

- $Q^*(s,a)$: expected utility on taking action a in state s, and thereafter acting optimally. Then, $V^*(s) = \max_a Q^*(s,a)$. Therefore, Bellman equations can be written as,

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right)\left(\max_{a'} Q^*\left(s',a'\right)\right)$$

Based on above a Q-value-iteration algorithm can be derived:
Pseudocode

- Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S\times A}$.
- In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \mathbb{E}_{s'}\left[\max_{a'} Q^{k-1}\left(s',a'\right)|s,a\right], \forall s,a$$

- Stop if $\left\|Q^k - Q^{k-1}\right\|_\infty$ is small.

Policy iteration

Start with an arbitrary initialization of policy π^1. The k-th policy iteration has two steps:

- **Policy evaluation**: Find $\mathbf{v}^k$ by solving $\mathbf{v}^k = \mathbf{L}^{\pi^k}\mathbf{v}^k$, i.e.,

$$\mathbf{v}^k(s) = \mathbb{E}_{a\sim\pi(s)}\left[R\left(s,a,s'\right) + \gamma\sum_{s'} P\left(s,a,s'\right)\mathbf{v}^k\left(s'\right)\right], \forall s$$

- **Policy improvement**: Find π^{k+1} such that $\mathbf{L}^{\pi^{k+1}}\mathbf{v}^k = \mathbf{L}\mathbf{v}^k$, i.e.,

$$\pi^{k+1}(s) = \arg\max_a R(s,a) + \gamma\mathbb{E}_{s'}\left[\mathbf{v}^k\left(s'\right)|s,a\right], \forall s$$

Outline

Platform

- Gym: a toolkit for developing and comparing reinforcement learning algorithms. Support Atari and Mujoco.
- Universe: measuring and training an AI across the world's supply of games, websites and other applications
- Deepmind Lab: a fully 3D game-like platform tailored for agent-based AI research
- ViZDoom: allows developing AI bots that play Doom using only the visual information

Platform

- RIlab: mainly supports for TRPO, VPG, CEM, NPG
- Baselines: supports for TRPO, PPO, DQN, A2C...
- Github
- Implement your algorithms through these packages

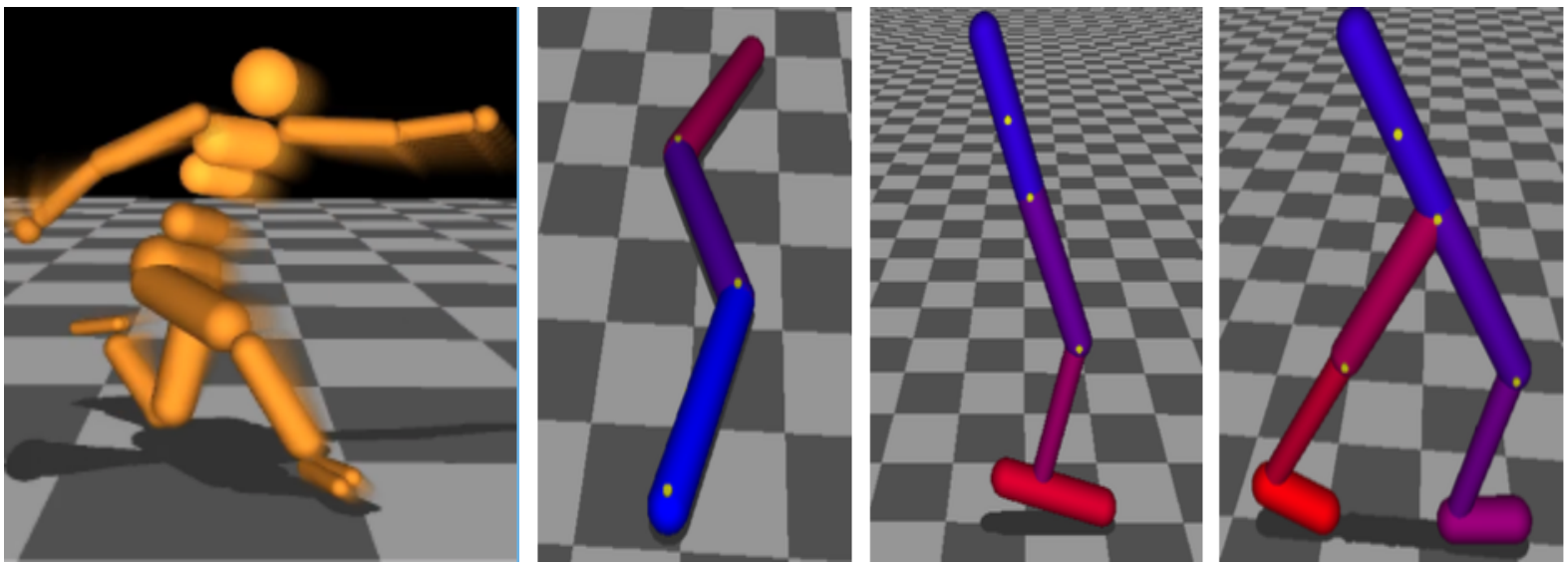
Gym

- A simple implementation of sampling a single path with gym

```
import gym, numpy
env = gym.make("MsPacman-v0")
ob = env.reset()
env.render()
rew = []
while True:
    action = pi.act(ob)
    ob, reward, done, info = env.step(action)
    rew.append(reward)
    if done:
        ret = numpy.sum(rew)
        break
```

Environments

- Mujoco, continuous tasks
- A physics engine for detailed, efficient rigid body simulations with contacts
- Swimmer, Hopper, Walker, Reacher,...
- Gaussian distribution



Environments

- Atari 2600, discrete action space
- Categorial distribution

A Taxonomy of RL Algorithms