

# **Reinforcement Learning (RL) and Deep RL**

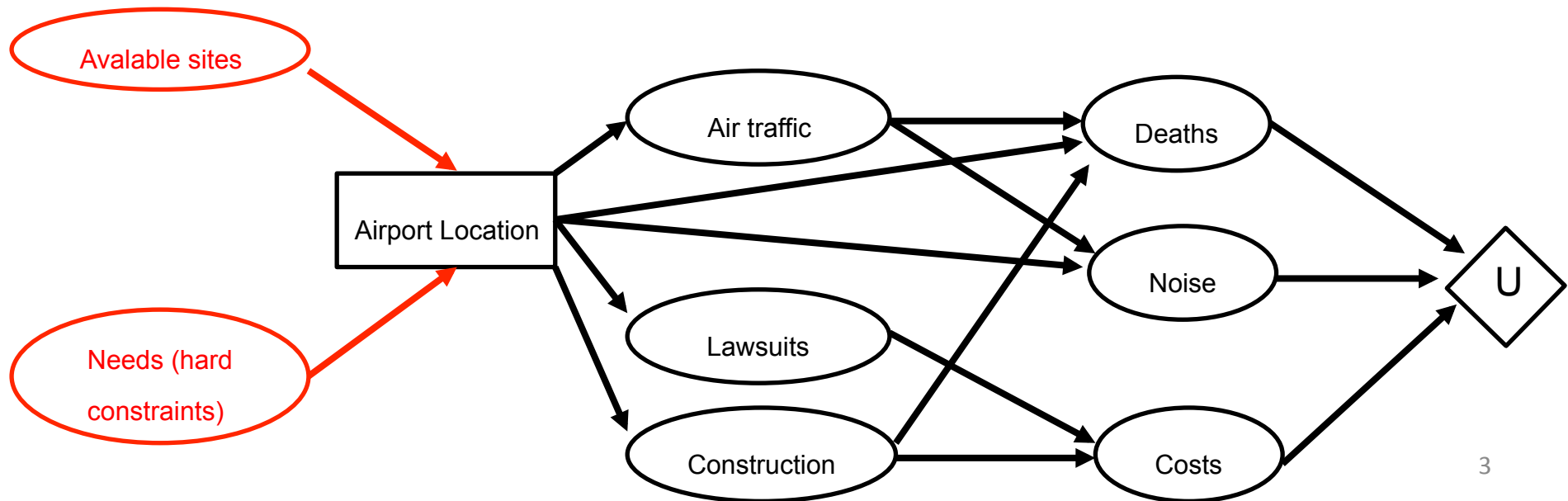
# Au menu

From Decision Theory to  
Reinforcement Learning (RL)

- Markov Decision Processes
- Value Iteration
- Policy Iteration
- Deep RL

# Single Decisions & Decision networks

- We can extend Bayesian networks to include *action nodes* (rectangles) and *utility (or reward) nodes* (diamonds)
- Example: Starting with available sites and needs, identify actions consisting of decisions regarding where to construct an airport.
- We can model the distribution over the air traffic, potential lawsuits and construction; then the number of deaths, the amount of noise and costs, then map these to a utility



# More complex decisions

- We can use the idea of maximizing the expected utility or expected value of outcomes to select which action to take
- However, more complex problems require many action decisions to be taken over time
- *Reinforcement learning* deals with the idea of determining how an 'agent' should take actions so as to maximize rewards over time
- A modeling framework has been proposed to describe this setup known as a *Markov Decision Process* or *MDPs*

# Reinforcement Learning

- In statistical learning we *maximize the likelihood* of data or *minimize a loss* function
- In reinforcement learning we add to our framework the idea of being able to obtain different *rewards* for being in different *states*
- We wish to *maximize the expected value of rewards*, using a discount factor for future rewards
- We can create a model of the future including the probability of arriving in different states as a result of actions we may take
- The mapping from our current state to the action we will take is known as the *policy*

# A Markov Decision Process (MDP) as a decision network

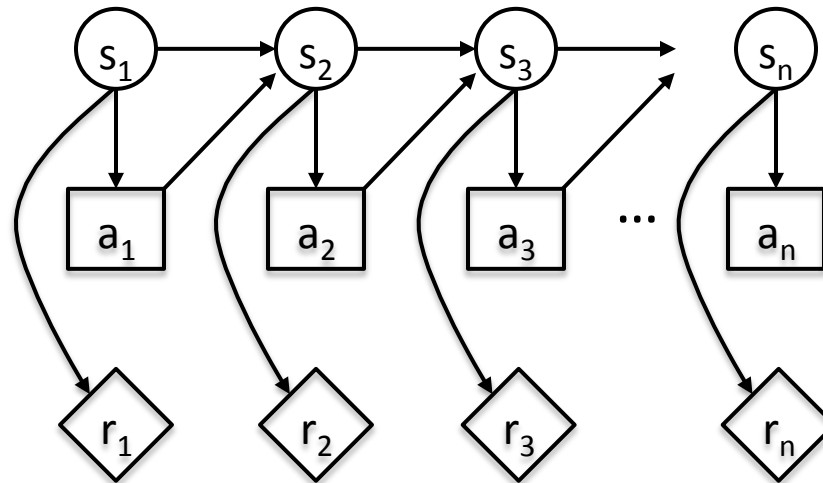
- We can use MDPs to visualize RL problem setups

We have:

states,  $s_t$   
actions,  $a_t$

and

rewards,  $r_t$



- Goal: Given the parameters  
 $R(s)$ ,  $P(s_1)=s_o$ ,  $P(s_{i+1} | s_i, a_i)=T(s',s,a)$ ,  $\gamma$
- Calculate the policy  $a=\pi(s)$  that maximizes

$$E \left[ \sum_{t=1}^{\infty} \gamma^t R(s_t) \mid \pi(s) \right] = V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' | s, \pi(s)) V_{\pi}(s')$$

# The infinite sum of discounted expected future rewards in recursive form

$$\begin{aligned} & E \left[ \sum_{t=1}^{\infty} \gamma^t R(s_t) \mid \pi(s) \right] \\ &= \sum_{s_1} \sum_{s_2} \dots \sum_{s_{\infty}} P(s_1, s_2, \dots, s_{\infty} \mid \pi(s)) \left[ \sum_{t=1}^{\infty} \gamma^t R(s_t) \right] \\ &= R(s_1) + \gamma \sum_{s_2} P(s_2 \mid s_1, \pi(s_1)) R(s_2) + \gamma^2 \sum_{s_2} P(s_2 \mid s_1, \pi(s_1)) \sum_{s_3} P(s_3 \mid s_2, \pi(s_2)) R(s_3) + \gamma^3 \dots \\ &= R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) \left[ R(s') + \gamma \sum_{s''} P(s'' \mid s', \pi(s')) V_{\pi}(s'') \right] \\ &= V_{\pi}(s) \end{aligned}$$

$$V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V_{\pi}(s')$$

- This is a so called Bellman equation: a recursion for expected rewards
- We have the expected reward of being in state  $s$  and following a policy  $\pi(s)$

# Choosing among different policies

- An optimal policy maximizes the expected value:

$$E\left[\sum_{t=1}^{\infty} \gamma^t R(s_t) \mid \pi(s)\right]$$

- Let the value function  $V(s)$  represent the value of being in a given state
- Given a  $V(s)$ , the one step greedy policy is

$$\pi(s) = \arg \max_a \left[ \sum_{s'} P(s' \mid a, s) V(s') \right]$$

- **Key idea:** if  $V(s)$  is the value function of an optimal policy, then the (one step greedy) equation above will give us that optimal policy



# Value Iteration

- Given a model for  $P(s' | s, a)$  and  $R(s)$  one can compute an optimal policy with Value Iteration and greedy policy recovery
- **Value Iteration:** an iterative procedure to calculate the value function of an optimal policy, without the need to have an optimal policy,
- **Greedy policy recovery:** gives us  $\pi_g(s)$  as a function of a given value function  $v(s)$ 
  - When  $v = v^*$  is the value function of an optimal policy, then  $\pi^*$  is also optimal

# (Classical) Value Iteration

- The utility or value of a state is the immediate reward associated with that state plus the discounted expected future value of the next state (given the agent chooses the optimal action)

$$v_{i+1}(s) = \max_a \left[ r(s) + \gamma \sum_{s'} v_i(s') P(s' | s, a) \right] \rightarrow \mathbf{v} = \mathbf{r} + \gamma \mathbf{P}^{a^*} \mathbf{v}$$

- This is the basis of the value iteration algorithm
- Iterations of value iteration are guaranteed to converge to the value function of an optimal policy, but (key) *the algorithm does not require the use of an optimal policy during iterations*

# Algorithme – itération de la valeur

**fonction** ITERATION-VALEUR( $pdm, \varepsilon$ ) **retourne** fonction d'utilité

**entrées:**  $pdm$ , un processus de décision markovien

$\varepsilon$ , erreur maximale permise

$V'$  est une fonction d'utilité initiale telle que  $V'(x) = 0$

$S$  est l'ensemble des états de  $pdm$

**répéter:**

$V \leftarrow V'$

$\delta \leftarrow 0$

**pour chaque**  $s \in S$  **faire:**

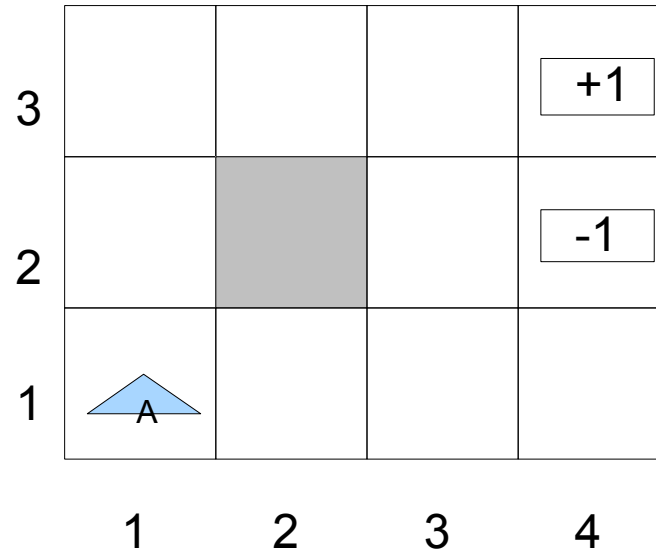
$V'[s] \leftarrow R[s] + \gamma \max_a \sum_{s'} T(s, a, s') V[s']$

**si**  $|V'[s] - V[s]| > \delta$  **alors**  $\delta \leftarrow |V'[s] - V[s]|$

**jusqu'à**  $\delta < \varepsilon(1-\gamma)/\gamma$

**retourner**  $V$

# Exemple



Récompense de -0,04 pour chaque case non finale  
Probabilité d'avancer à la case suivante: 0,8  
Probabilité de se déplacer sur une case latérale: 0,1

# Exemple – politique optimale

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

Récompense de -0,04 pour chaque case non finale  
Probabilité d'avancer à la case suivante: 0,8  
Probabilité de se déplacer sur une case latérale: 0,1

# Exemple – politique optimale

3	→	→	→	+1
2	↑		→	-1
1	→	→	→	↑
	1	2	3	4

*L'agent se précipite  
le plus vite possible  
vers l'état final le plus  
proche, même s'il est  
le moins intéressant*

Récompense inférieure à -1,6284 pour chaque case non finale

Probabilité d'avancer à la case suivante: 0,8

Probabilité de se déplacer sur une case latérale: 0,1

# Exemple – politique optimale

3	→	→	→	+1
2	↑		←	-1
1	↑	←	←	↓
	1	2	3	4

*L'agent ne prend  
aucun risque*

Récompense entre -0,0221 et 0 pour chaque case non finale

Probabilité d'avancer à la case suivante: 0,8

Probabilité de se déplacer sur une case latérale: 0,1

# Itération de la valeur -simulation

3	0.0	0.0	0.0	0.0
2	0.0		0.0	0.0
1	0.0	0.0	0.0	0.0
	1	2	3	4

$$\gamma = 1$$



# Itération de la valeur -simulation

3	-0.04	-0.04	-0.04	1
2	-0.04		-0.04	1
1	-0.04	-0.04	-0.04	-0.04
	1	2	3	4

meilleure action: *droite*

$$U = -0,04 + 0,8*1 - 0,1*0,04 - 0,1*0,04 = 0,752$$

meilleure action: *gauche*

$$U = -0,04 - 0,8*0,04 - 0,1*0,04 - 0,1*0,04 = -0,08$$

$$\gamma = 1$$

# Itération de la valeur -simulation

3	-0.08	-0.08	0,752	1
2	-0.08		-0.08	1
1	-0.08	-0.08	-0.08	-0.08
	1	2	3	4

meilleure action: *droite*

$$\begin{aligned}
 U &= -0,04 + 0,8*1 - \\
 &\quad + 0,1*0,752 - 0,1*0,08 \\
 &= 0,827
 \end{aligned}$$

meilleure action: *haut*

$$\begin{aligned}
 U &= -0,04 + 0,8*0,752 \\
 &\quad - 0,1*1 - 0,1*0,08 \\
 &= 0,454
 \end{aligned}$$

$$\gamma = 1$$

# Itération de la valeur -simulation

3	-0.12	0,546	0,827	1
2	-0.12		0,454	-1
1	-0.12	-0.12	-0.12	-0.12
	1	2	3	4

$$\gamma = 1$$

# Itération de la valeur -simulation

3	0,372	0,731	0,888	1
2	-0.16		0,567	-1
1	-0.16	-0.16	0,299	-0.16
	1	2	3	4

$$\gamma = 1$$

# Itération de la valeur -simulation

Valeurs finales:

3	0,812	0,868	0,918	1
2	0,762		0,660	-1
1	0,705	0,655	0,611	0,388
	1	2	3	4

$$\gamma = 1$$

# **Policy Iteration**

# Itération de la politique

- **Itération de la politique** : une (autre) procédure itérative pour calculer une politique optimale étant donné  $P(s'|s,a)$  et  $r(s)$ 
  - Initialiser  $\pi_0(s)$  (ex. aléatoirement) et évaluer  $v_0(s)$
  - Répéter
    - Calculer *la politique gloutonne* « one step »  $\pi_{i+1}(s)$
    - Utiliser « policy evaluation » à obtenir  $v_{i+1}(s)$
    - Arrêter quand la différence entre  $v_{i+1}$  et  $v_i$  est petite
- **« Policy Evaluation »** -- l'obtention de l'utilité de chaque état, ou : étant donné une politique  $\pi(s)$ , et un modèle de transition  $P(s'|s,a)$  connu, il est possible d'obtenir l'utilité  $U(s)$  ou la fonction de valeur  $v(s)$  de chaque état

# L'évaluation d'une politique

- Étant donné une politique  $\pi(s)$ , on peut obtenir la fonction de la valeur  $\mathbf{v}(s)$  « the value function » (ou la fonction d'utilité  $U(s)$ ) avec la solution d'un système linéaire d'équations

$$\mathbf{v}_{\pi}(s) = \mathbf{r}(s) + \gamma \sum_{s'} \mathbf{v}_{\pi}(s') P(s' | s, \pi(s))$$

$$\mathbf{v} = \mathbf{r} + \gamma \mathbf{P} \mathbf{v}$$

$$(I - \gamma \mathbf{P}) \mathbf{v} = \mathbf{r}$$

$$\mathbf{v} = (I - \gamma \mathbf{P})^{-1} \mathbf{r}$$

- Étant donné les actions  $a=\pi(s)$  et donc une matrice  $P$ , composée de  $P(s'|s,a)$



# Itération de la valeur vs. politique

- Nous sommes assurés de converger vers la solution optimale avec les deux approches : a) l'itération de la valeur et b) l'itération de la politique
- Toutefois chaque itération est (possiblement) moins chère avec l'itération de la valeur parce qu'on n'a pas besoin d'évaluer la politique (ex. obtenir la solution à un système d'équations linéaire)

# Algorithme – itération de la politique

- En principe l'évaluation d'une politique exige la résolution d'une série d'équations linéaires
- On peut simplifier en itérant  $k$  fois la mise à jour suivante:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} T(s, \pi_i(s), s') U_i[s']$$

# Algorithme – itération de la politique

**fonction** ITERATION-POLITIQUE( $pdm, \epsilon$ ) **retourne** fonction d'utilité

**entrées:**  $pdm$ , un processus de décision markovien

$\epsilon$ , erreur maximale permise

$\pi$  est une politique initialisée aléatoirement

$S$  est l'ensemble des états de  $pdm$

**répéter:**

$U \leftarrow \text{EVALUATION-POLITIQUE}(\pi, U, pdm)$

$inchangé \leftarrow true$

**pour chaque**  $s \in S$  **faire:**

**si**  $\max_a \sum_{s'} T(s, a, s') U[s'] > \sum_{s'} T(s, \pi[s], s') U[s']$  **alors**

$\pi[s] = \arg \max_a \sum_{s'} T(s, a, s') U[s']$

$inchangé = false$

**jusqu'à**  $inchangé$

**retourner**  $\pi$

# Notez Bien

- Dans le deux cas : itération de la valeur et itération de la politique les connues sont:

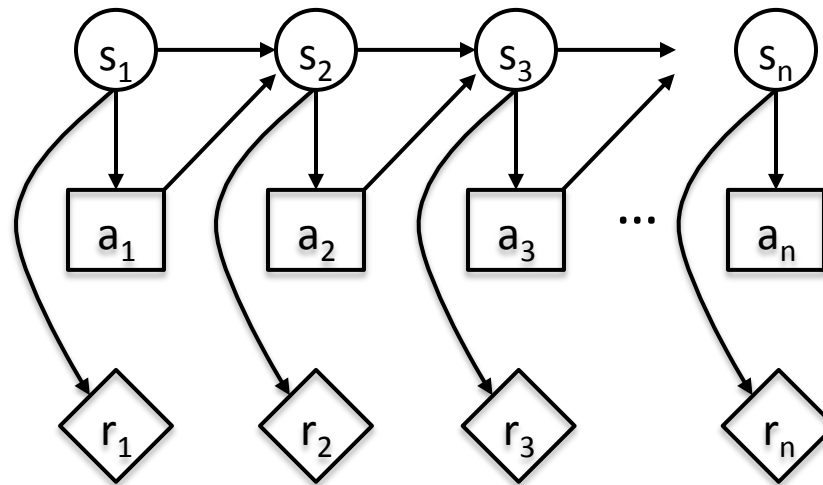
$$R(s), P(s_1)=s_0, P(s_{i+1} | s_i, a_i)=T(s', s, a), \gamma$$

Nous avons:

états,  $s_t$   
actions,  $a_t$

et

récompenses,  $r_t$



- Notre but est de calculer la politique  $a=\pi(s)$  qui va maximiser

$$E \left[ \sum_{t=1}^{\infty} \gamma^t R(s_t) \mid \pi(s) \right] = V_{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) V_{\pi}(s')$$

- Cependant, les connus ci-dessus ne sont pas toujours connus.**

# Apprentissage par renforcement

- Apprentissage passif:
  - On ne connaît pas le modèle de transition
  - On **connaît** la politique
  - On veut déterminer les valeurs d'utilité
- Apprentissage actif:
  - On ne connaît pas le modèle de transition
  - On **ne connaît pas** la politique
  - On veut déterminer la politique

# Apprentissage passif: estimation directe

- On fait un ensemble d'essais, chacun comprenant une séquence de transitions
- Pour chaque occurrence d'une transition, on calcule l'utilité
- Pour chaque transition, on calcule la moyenne des valeurs obtenues
- Cet algorithme converge lentement

# Apprentissage passif: programmation dynamique adaptative

**fonction** AGENT-PDA-PASSIF( $s'$ ,  $r'$ ) **retourne** une action

**entrées:**  $s'$ , état actuellement perçu

$r'$ , récompense actuellement perçue

$\pi$  est une politique fixe

$pdm$  est le processus de décision markovien défini par  $U$ ,  $R$  et  $T$

$s$ ,  $a$ , état et action précédents, initialement nuls

**si**  $s'$  est nouveau **alors faire**  $U[s'] \leftarrow r'$ ;  $R[s'] \leftarrow r'$

**si**  $s'$  non nul **alors faire**

incrémenter  $N_{sa}[s, a]$  et  $N_{sas'}[s, a, s']$

**pour tout**  $t$  tel que  $N_{sas'}[s, a, t] \neq 0$  **faire**

$T[s, a, t] \leftarrow N_{sas'}[s, a, t] / N_{sa}[s, a]$

$U \leftarrow \text{DETERMINER-VALEUR}(\pi, U, pdm)$

**si**  $\text{TERMINAL?}[s]$  **alors**  $s, a \leftarrow \text{nul}$  **sinon**  $s, a \leftarrow s', \pi[s']$

**retourner**  $a$

# Apprentissage actif

- Rappel, ici :
  - On ne connaît pas le modèle de transition
  - On **ne connaît pas** la politique
  - On veut déterminer la politique
- On peut adapter l'algorithme d'apprentissage passif avec un « model based approach »
- On apprend d'abord le modèle de transition
- Puis on applique un algorithme pour obtenir la politique optimale
- On répète



# Apprentissage actif

- Une question: l'agent doit-il choisir la prochaine action selon sa politique optimale courante?
- Réponse: pas toujours.
- L'agent doit aussi faire un peu d'exploration
- Une méthode simple: choisir une action aléatoirement une fois de temps en temps
- Une meilleure approche: donner du poids à des actions rarement choisies, tout en essayant d'éviter des actions qui sont reconnues pour leur peu d'utilité

# Asynchronous Value Iteration

- Key observation: need not loop over all states - could consider states one at a time  
(Provably convergent if we visit infinitely often)
- Usually Implemented by storing  $Q(s,a)$

# Q-Learning

- Allows us to learn an (optimal) policy
- Doesn't use a model of the environment!
  - It is a “model free method”
- Based on using an **action-value** “ $Q$ ” function
- If  $V^*(s)$  is the expected value of following the optimal policy in state  $s$
- $Q^*(s,a)$  is the expected value of taking action  $a$  in state  $s$  - then following the optimal policy

# $Q^*(s,a)$ and $V^*(s)$

- Defined as functions of one another

$$Q^*(s, a) = \sum_{s'} P(s'|a, s) (r(s, a, s') + \gamma V^*(s'))$$

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

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

Ex. see slides from Kevin L.B. @ UBC & many others...

# Value Iteration Revisited

## - using Qs, becomes

- Randomly initialize  $V_i(s)$ , then update

$$Q_{i+1}(s, a) = \sum_{s'} P(s'|a, s) (r(s, a, s') + \gamma V_i(s'))$$

$$V_{i+1}(s) = \max_a Q_{i+1}(s, a)$$

- Could combine the above to get a modified update to V for value iteration

$$V_{i+1}(s) = \max_a \sum_{s'} P(s'|a, s) (r(s, a, s') + \gamma V_i(s'))$$

# Policy Iteration Revisited

## – using Qs becomes

- In Policy Iteration we had iterations of the form:

$$\mathbf{v}_{\pi}(s) = \mathbf{r}(s) + \gamma \sum_{s'} \mathbf{v}_{\pi}(s') P(s' | s, \pi(s)) \rightarrow \mathbf{v} = (I - \gamma \mathbf{P})^{-1} \mathbf{r}$$

- Rather than solving a linear system we could use an approach called TD(0)

$$\mathbf{v}_{i+1}(s) = \mathbf{v}_i(s) + \alpha \left[ \left( \mathbf{r}(s') + \gamma \mathbf{v}_i(s') \right) - \mathbf{v}_i(s) \right]$$

- With Qs we have

$$\mathbf{Q}_{\pi}(s, a) = + \sum_{s'} P(s' | s, \pi(s)) \left[ \mathbf{r}(s, a, s') + \gamma \mathbf{v}_{\pi}(s') \right]$$

# A Simple but widely used idea in RL

- It is possible to estimate value functions  $V(s)$ , or action value functions  $Q(s,a)$  by computing *averages of observed rewards*
- Also possible to compute averages of a function  $F=Q(s,a)$  or  $F=V(s)$  incrementally, with constant time and memory costs
- If  $r_i$  is the reward obtained the  $i^{\text{th}}$  time we have visited a state, or after selecting an action while being in a particular state

# A Simple but widely used idea in RL

- After  $n-1$  examples we could write

$$F_n \equiv \frac{r_1 + r_2 + \dots + r_{n-1}}{n-1}$$

- But we don't want to (re)compute that sum each time we get a new reward, so we observe

$$\begin{aligned} F_{n+1} &= \frac{1}{n} \sum_{i=1}^n r_i &= \frac{1}{n} \left( r_n + \sum_{i=1}^{n-1} r_i \right) \\ &= \frac{1}{n} (r_n + (n-1)F_n) &= \frac{1}{n} (r_n + nF_n - F_n) \\ F_{n+1} &= F_n + \frac{1}{n} (r_n - F_n) \end{aligned}$$



# Leads to a general form

- New estimate is the old estimate adjusted by an error in the estimate

$$F_{n+1} = F_n + \frac{1}{n} (r_n - F_n)$$

$$\text{NewEstimate} = \text{Old} + \text{StepSize} \times \underbrace{[\text{Target-Old}]}_{\text{Error}}$$

- Where the target is a (possibly noisy) direction that you wish to move.
- See Sutton and Barto Textbook for lots more!  
<http://incompleteideas.net/book/bookdraft2017nov5.pdf>

# Comparing Value & Policy Iteration vs “Q Learning”

- In Value Iteration we iterate:

$$v_{i+1}(s) = \max_a \left[ r(s) + \gamma \sum_{s'} v_i(s') P(s' | s, a) \right] \rightarrow \mathbf{v} = \mathbf{r} + \gamma \mathbf{P}^{a^*} \mathbf{v}$$

- In Policy Iteration we evaluate the policy, iterating:

$$\mathbf{v}_\pi(s) = \mathbf{r}(s) + \gamma \sum_{s'} \mathbf{v}_\pi(s') P(s' | s, \pi(s)) \rightarrow \mathbf{v} = (I - \gamma \mathbf{P})^{-1} \mathbf{r}$$

– Then update the policy and repeat (evaluating it)

- Q-Learning is based on the so-called TD(0) approach where we have updates

$$v_{i+1}(s) = v_i(s) + \alpha \left[ (r(s') + \gamma v_i(s')) - v_i(s) \right]$$

- Where  $\langle s, a, r, s' \rangle$  are *experience tuples*
- Intuition:  $r(s') + \gamma v_i(s')$  gives a better estimate of  $v$

# Deep Q-learning

- Formulated using Q functions we have

$$Q_{i+1}(s, a) = Q_i(s, a) + \alpha \left[ \left( r(s') + \gamma \max_{a' \text{ critic}} Q_i(s', a') \right) - Q_i(s, a) \right]_{\text{actor}}$$

- Deep Learning*: parameterize  $Q(s, a; \theta)$  using a deep convolutional neural network and learn via gradient descent using the loss

$$L(\theta_i) = E \left[ \left( \left[ r(s') + \gamma \max_{a'} Q_i(s', a'; \theta_i^-) \right] - Q_i(s, a; \theta_i) \right)^2 \right] \tilde{P}(s, a, r, s')$$

- The expectation is with respect to mini-batches of experience vectors  $\langle s, a, r, s' \rangle$

# Updates

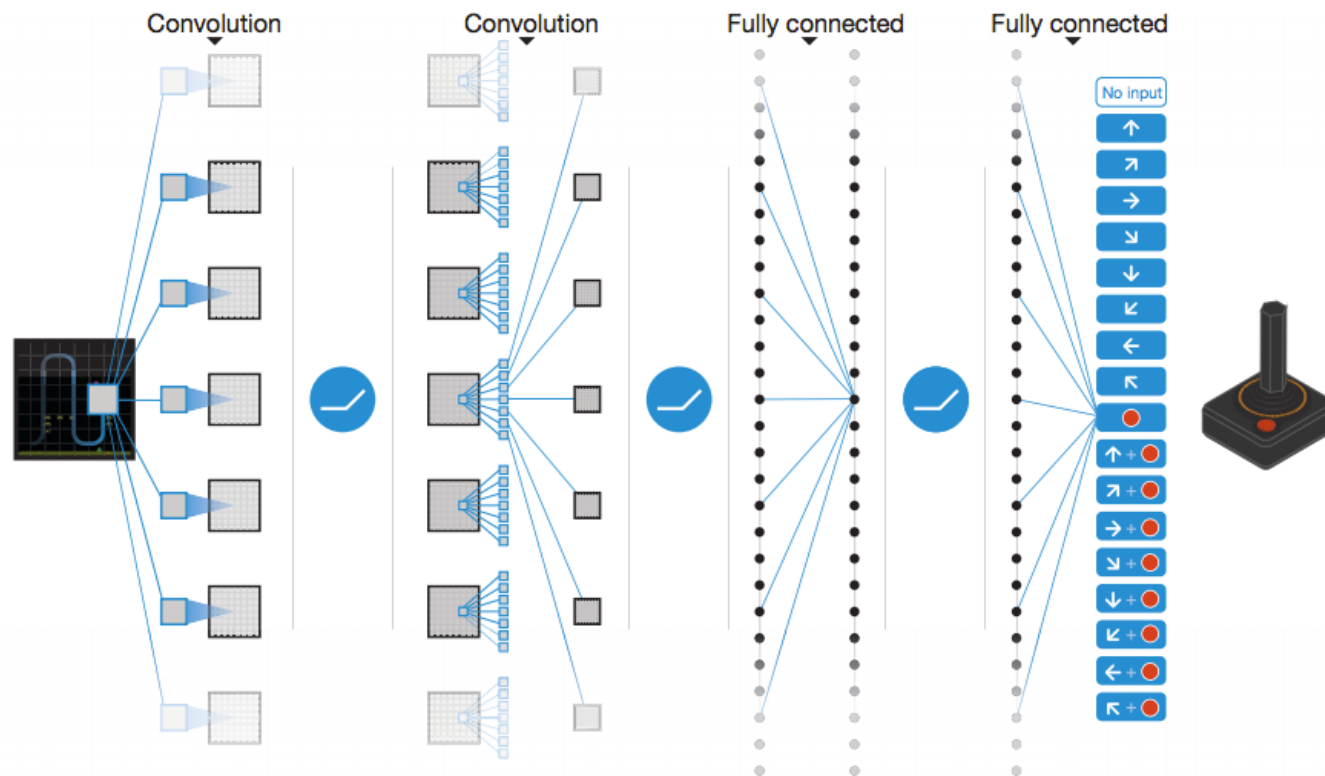
$$L(\theta_i) = E \left[ \left( \left[ r(s') + \gamma \max_{a'} Q_i(s', a'; \theta_i^-) \right] - Q_i(s, a; \theta_i) \right)^2 \right] \tilde{P}(s, a, r, s')$$

- Where  $\theta_i$  are the parameters of the Q-network at iteration  $i$
- $\theta_i^-$  are the network parameters used to compute the target at iteration  $i$
- The target network parameters are only updated with the Q-network parameters every C steps
- Updates applied to samples drawn uniformly from experiences to create mini-batches

# Q-Learning

- Goal is to obtain a policy for an agent which doesn't have a transition model for the environment or even the reward function.
- Simply choose the action that maximizes  $Q$  based on our current estimate
- Occasionally select a random action
- As time goes on, take fewer random actions

# Mnih et al., (2015) “Human-level control through deep reinforcement learning”, i.e. Teaching computers to Play Atari Games



**Figure 1 | Schematic illustration of the convolutional neural network.** The details of the architecture are explained in the Methods. The input to the neural network consists of an  $84 \times 84 \times 4$  image produced by the preprocessing map  $\phi$ , followed by three convolutional layers (note: snaking blue line

symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is,  $\max(0, x)$ ).

# Le Projet

- Discuter de quelques idées avec votre professeur, le chargé de laboratoire et vos collègues.  
Identifiez :
- un article de recherche relevant (important pour les étudiants aux études supérieures), ou une partie d'un chapitre de notre manuel
- Indiquez en grandes lignes votre choix pour le sujet de votre projet dans un document Google sur Moodle
- **16 Avril.** Présentations de projets, conférence en classe (session d'affiches "Poster session" ).