Deep Learning (Parte 6)

Índice Geral

1 Introdução ao Aprendizado por Reforço

- Nota: Todo o conteúdo em inglês deste material foi extraído de:
  - ✓ Fei-Fei Li; Justin Johson; Serena Yeung “Deep Reinforcement Learning”, Stanford University. Vídeo disponível em:

    https://www.youtube.com/watch?v=lvoHnicueoE
  - ✓ Busoniu. L.; Babuska, R.; De Schutter, B.; Ernst, D. “Reinforcement learning and dynamic programming using function approximators”, CRC Press, 2010.
- Há diversos vídeos associados a este material. Confira no último slide.
- Confira também o material em: http://karpathy.github.io/2016/05/31/rl/

- Numa tentativa de definição sucinta, é possível afirmar que aprendizado por reforço promove a síntese automática de comportamento inteligente em ambientes dinâmicos complexos, que envolvem múltiplas etapas de tomada de decisão na presença de incerteza e/ou informação incompleta sobre o problema.

So far… Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

→ Cat

Classification

So far… Unsupervised Learning

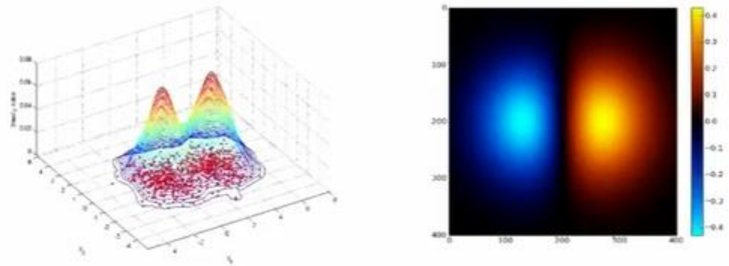
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

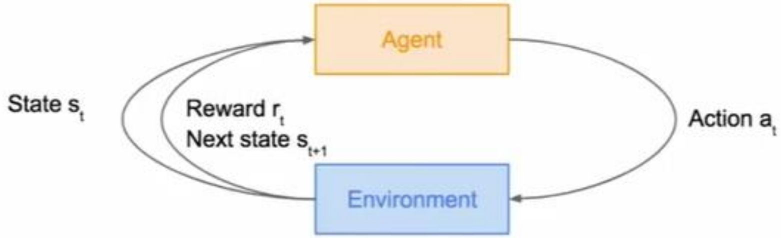
1-d density estimation



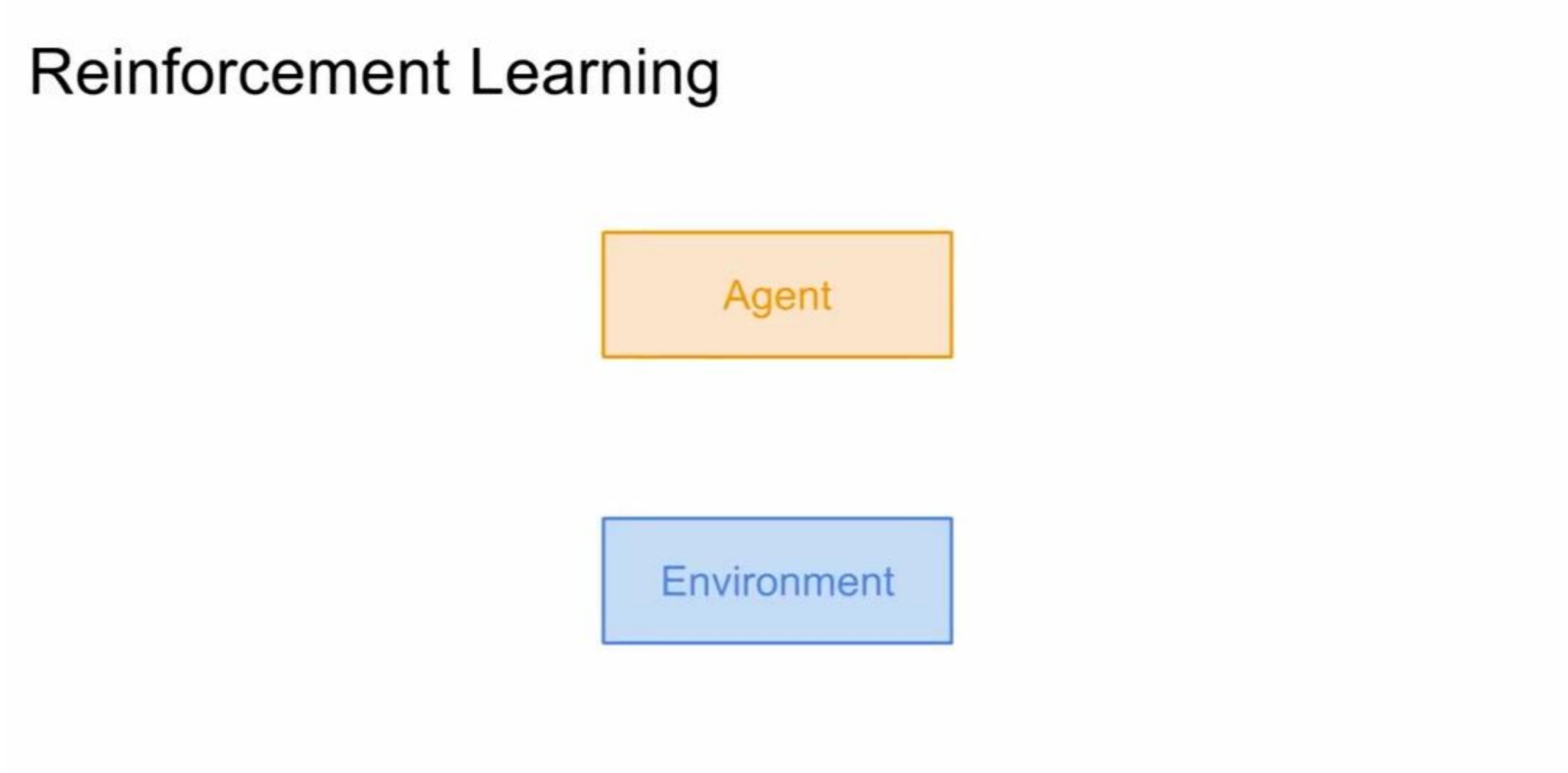
2-d density estimation

Today: Reinforcement Learning

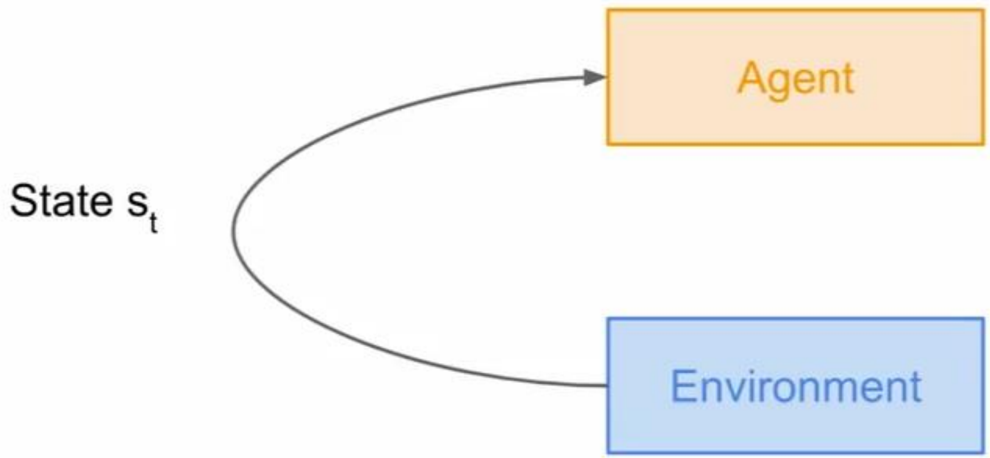
Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

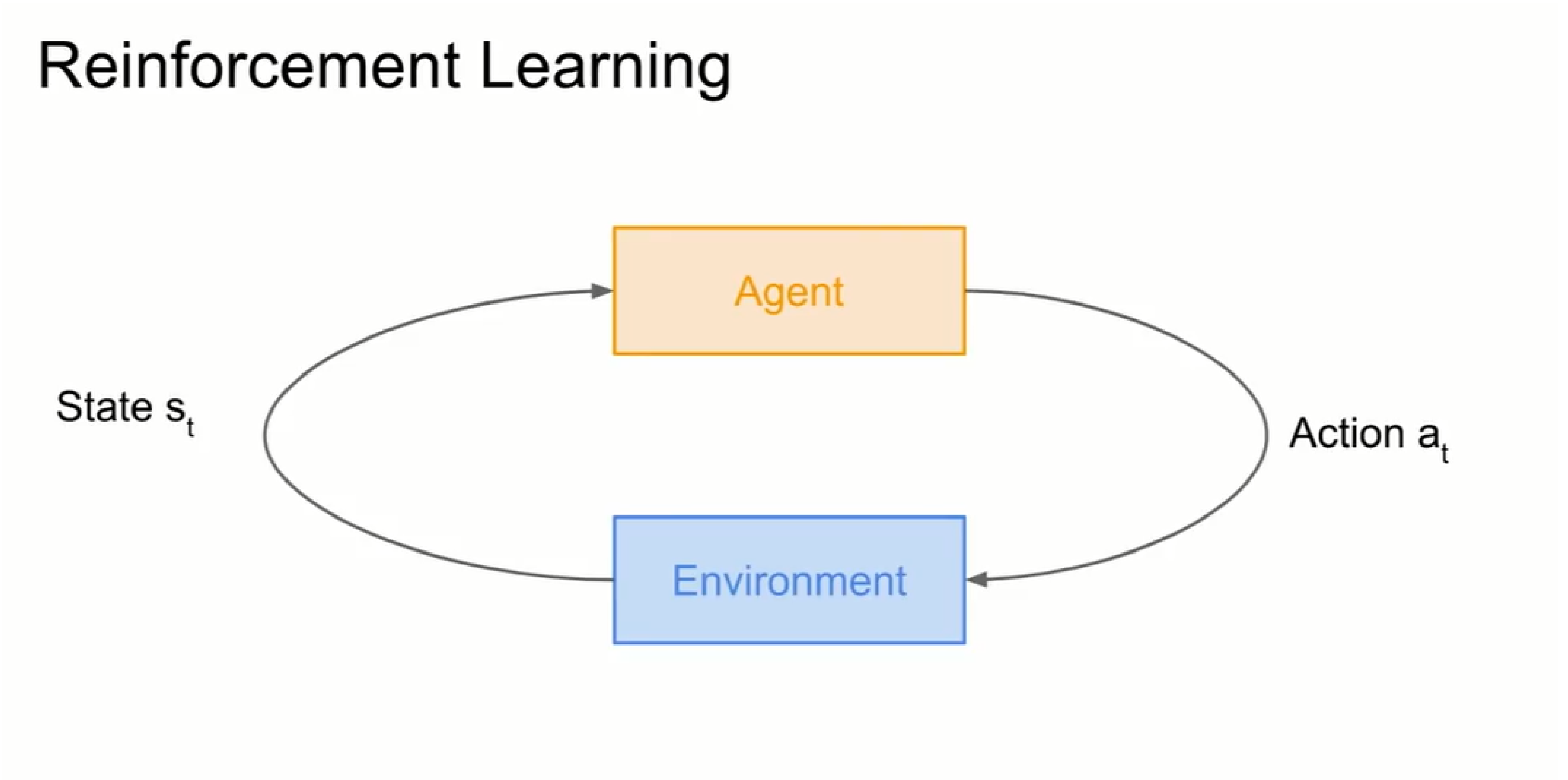


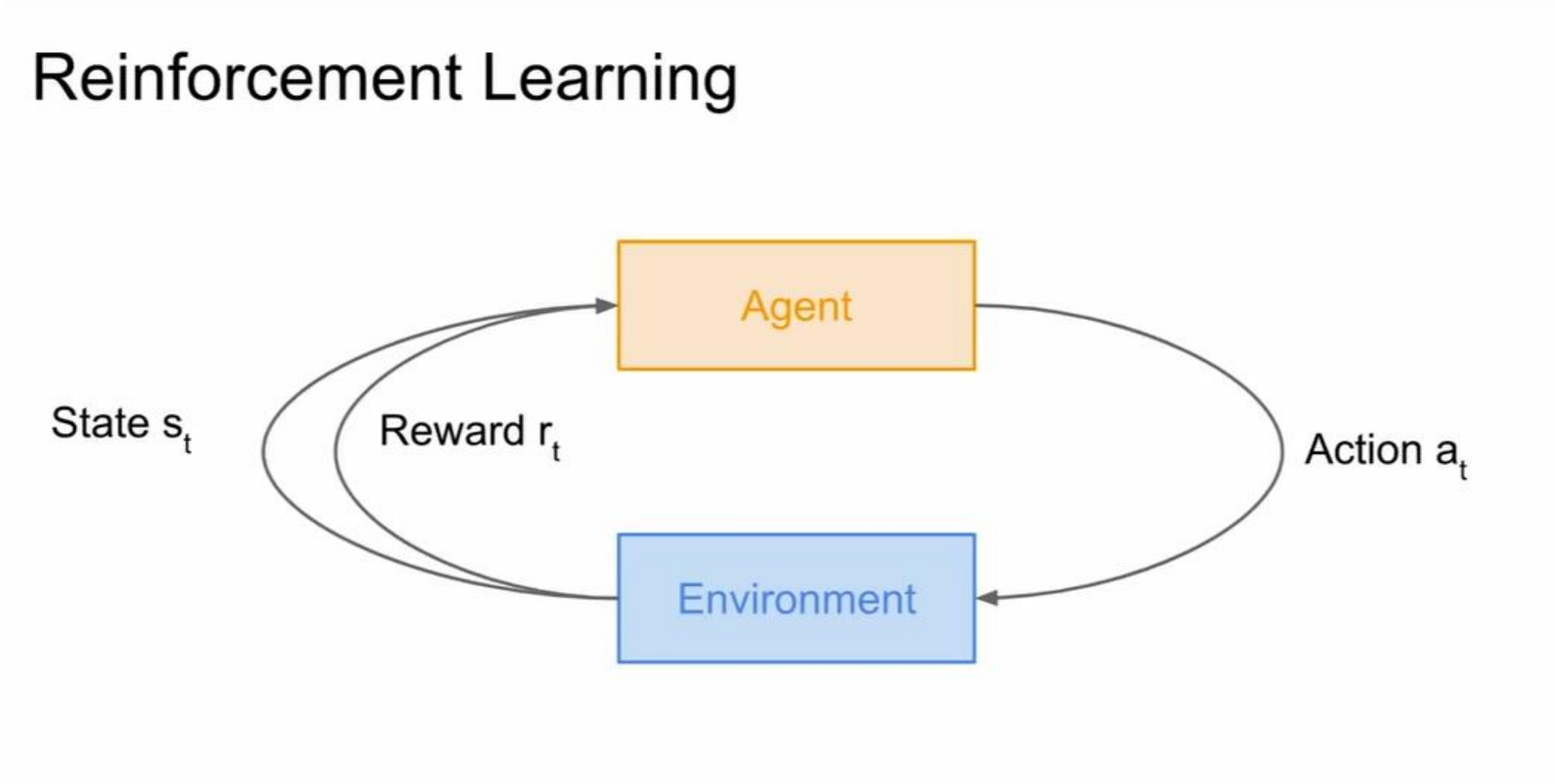
Goal: Learn how to take actions in order to maximize reward


Reinforcement Learning
Agent
Environment

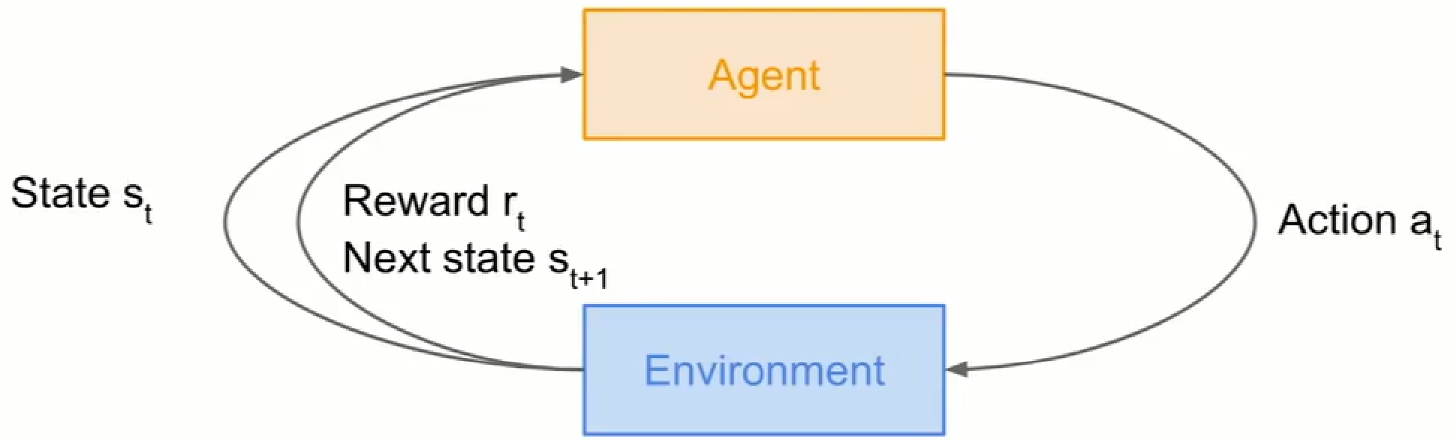
Reinforcement Learning




Reinforcement Learning
Agent
State s_t
Action a_t
Environment

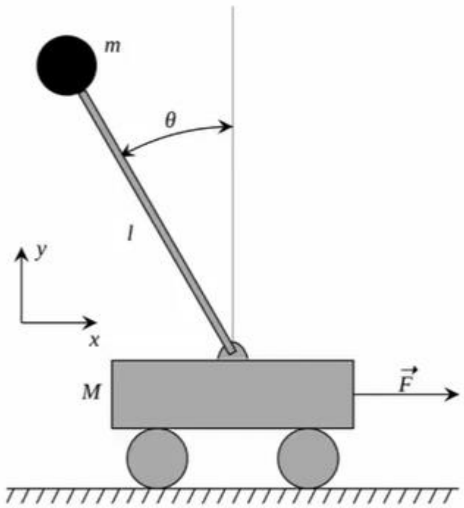

Reinforcement Learning
Agent
State s_t
Reward r_t
Action a_t
Environment

Reinforcement Learning



- Considerando o cenário de um jogo de tabuleiro, com aprendizado por reforço uma máquina pode aprender a se tornar imbatível jogando contra si própria. Por outro lado, considerando o mesmo cenário, com aprendizado supervisionado uma máquina poderia no máximo atingir o desempenho do especialista.

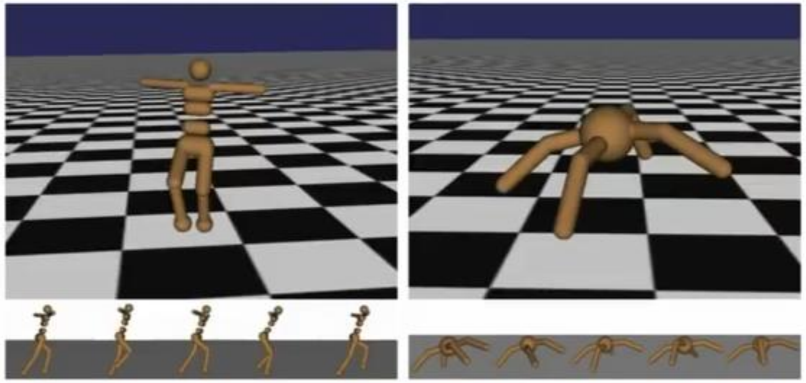
Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

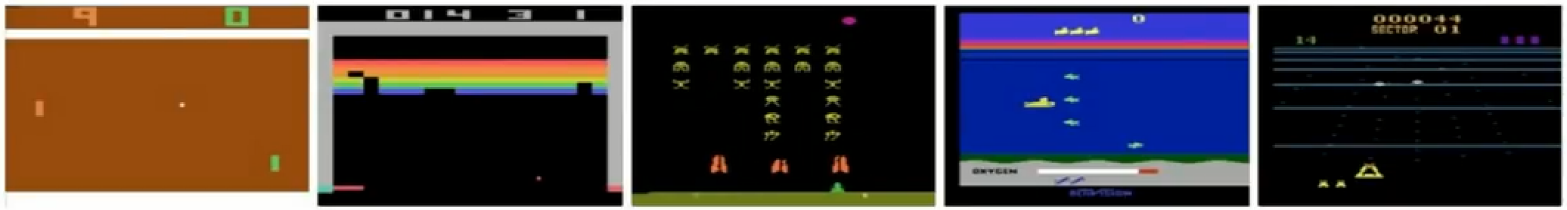
Robot Locomotion



Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

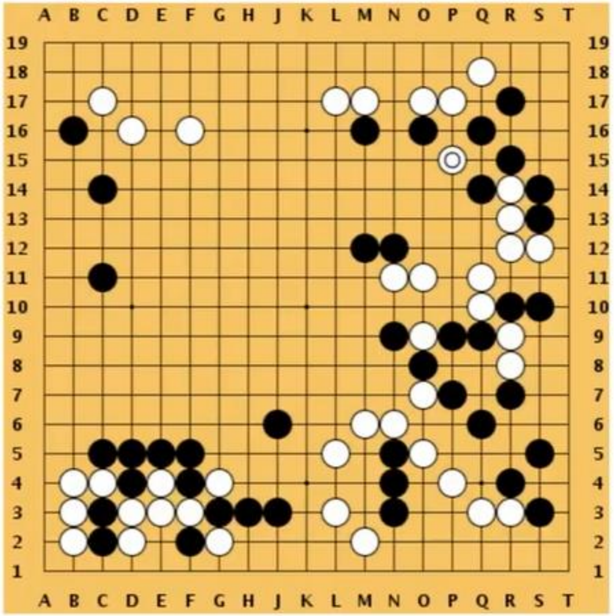
Atari Games



Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

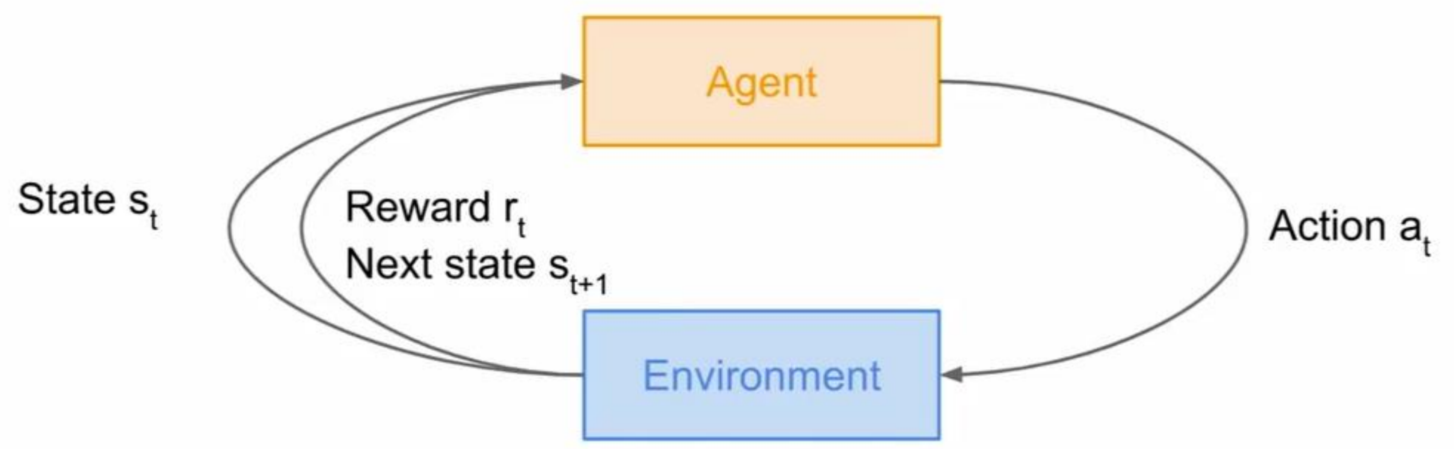
Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

How can we mathematically formalize the RL problem?



Markov Decision Process

- Mathematical formulation of the RL problem
- **Markov property**: Current state completely characterises the state of the world

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - Agent receives reward r_t and next state s_{t+1}

- A policy $\boldsymbol{\pi}$ is a function from S to A that specifies what action to take in each state
- **Objective**: find policy $\boldsymbol{\pi}^*$ that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

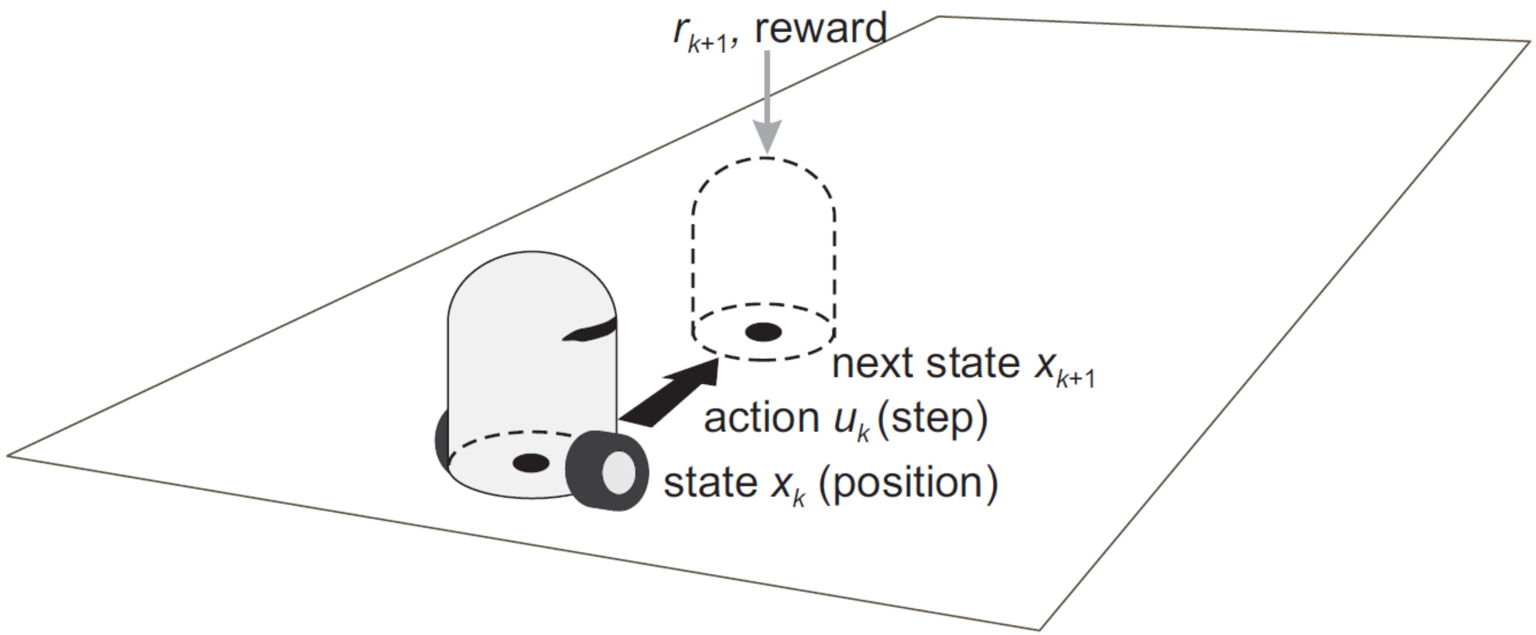


Figura 1 – Numa notação um pouco distinta, esta figura ilustra os conceitos do slide da pg. 17 num contexto de navegação de um robô por um ambiente.

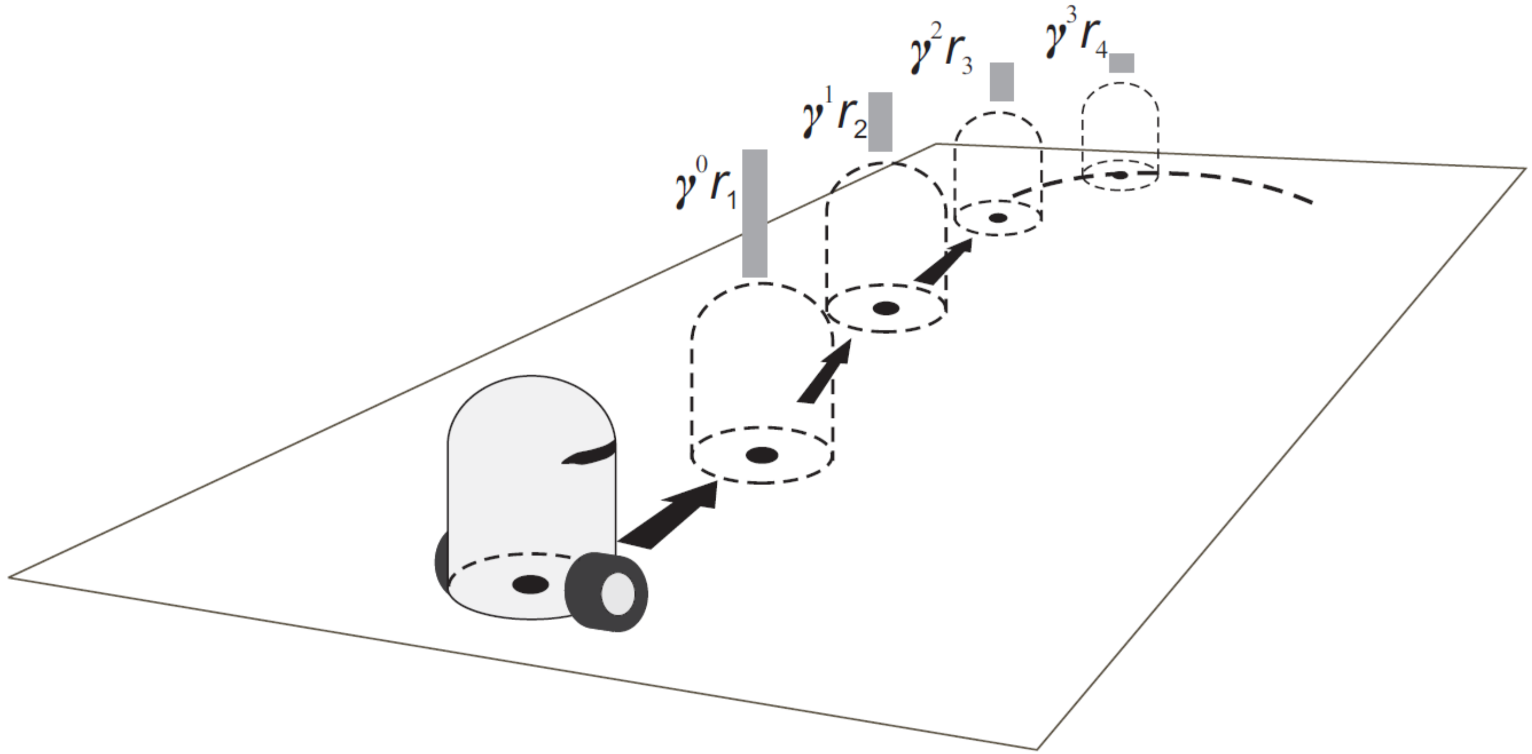


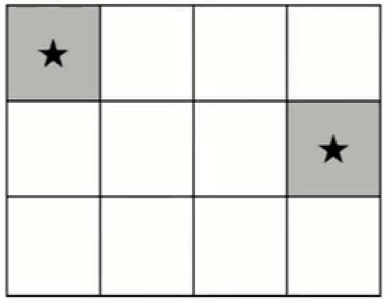
Figura 2 – Exemplo de recompensa acumulada empregando um termo de desconto.

A simple MDP: Grid World

actions = {
1. right
2. left
3. up
4. down

}

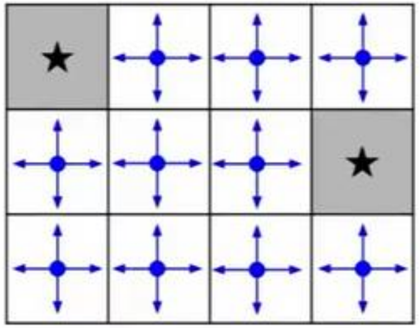
states



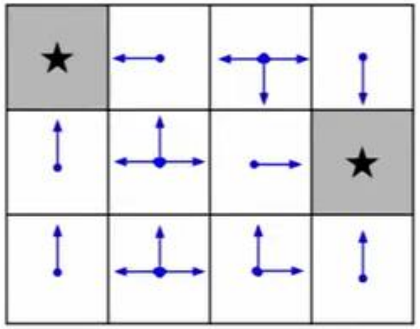
Set a negative "reward" for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

A simple MDP: Grid World



Random Policy



Optimal Policy

The optimal policy $\boldsymbol{\pi}^*$

We want to find optimal policy $\boldsymbol{\pi}^*$ that maximizes the sum of rewards.

How do we handle the randomness (initial state, transition probability…)?
Maximize the **expected sum of rewards!**

Formally: $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$

Definitions: Value function and Q-value function

Following a policy produces sample trajectories (or paths) $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

How good is a state?

The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

How good is a state-action pair?

The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q* satisfies the following **Bellman equation**:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s,a\right]$$

Intuition: if the optimal state-action values for the next time-step Q*(s',a') are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$

- The optimal policy π^* corresponds to taking the best action in any state as specified by Q^*.

Solving for the optimal policy

Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')|s,a\right]$$

Q_i will converge to Q* as i -> infinity

What's the problem with this?
Not scalable. Must compute Q(s,a) for every state-action pair. If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!

Solution: use a function approximator to estimate Q(s,a). E.g. a neural network!

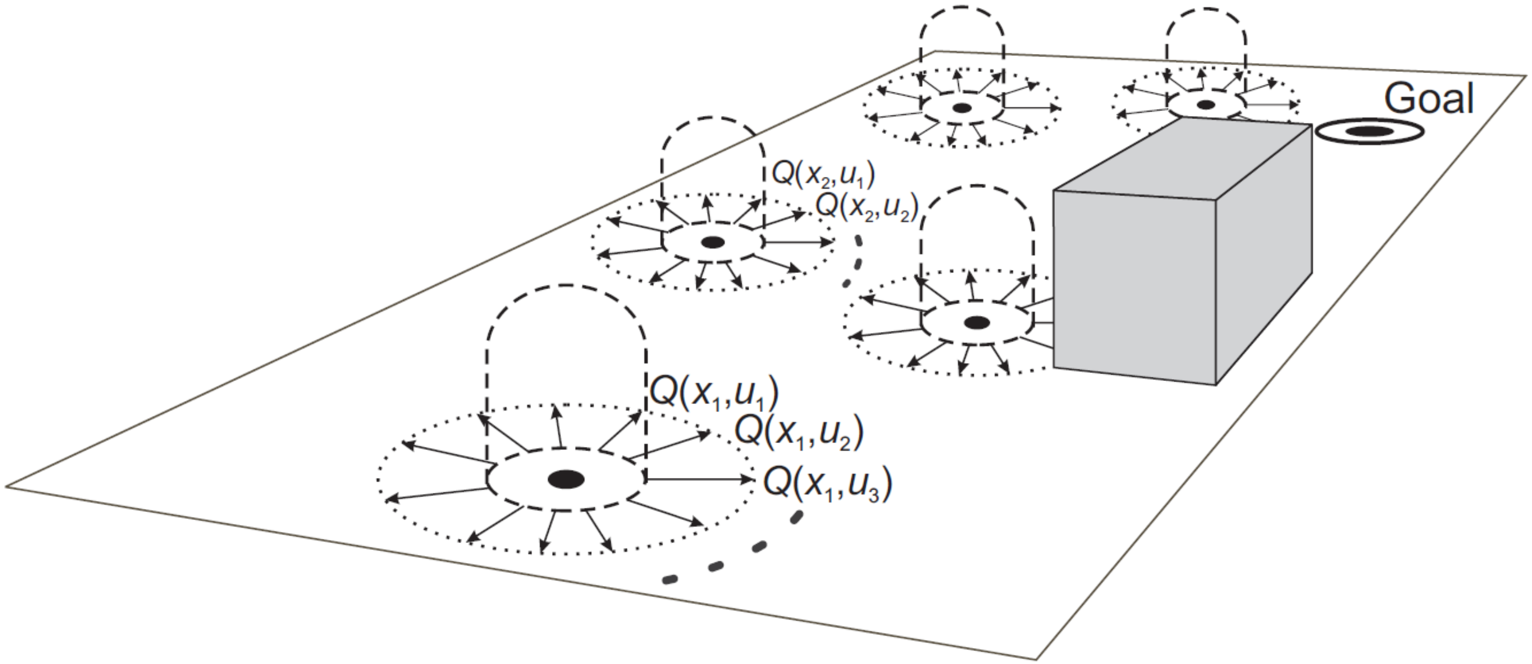


Figura 3 – Exemplo que ilustra a necessidade de se calcular $Q(s,a)$, na figura é usada a notação $Q(x_i,u_i)$, para cada estado possível do ambiente, o que torna o problema intratável para muitos estados e muitas ações por estado.

- De fato, embora muitas aplicações de sucesso já tenham sido viabilizadas, os problemas de *credit assignment* e de *sparse reward* continuam sendo os maiores desafios para um aprendizado por reforço efetivo.

2 Deep Q-learning

Solving for the optimal policy: Q-learning

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

function parameters (weights)

If the function approximator is a deep neural network => **deep q-learning**!

Solving for the optimal policy: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Forward Pass

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy **π***)

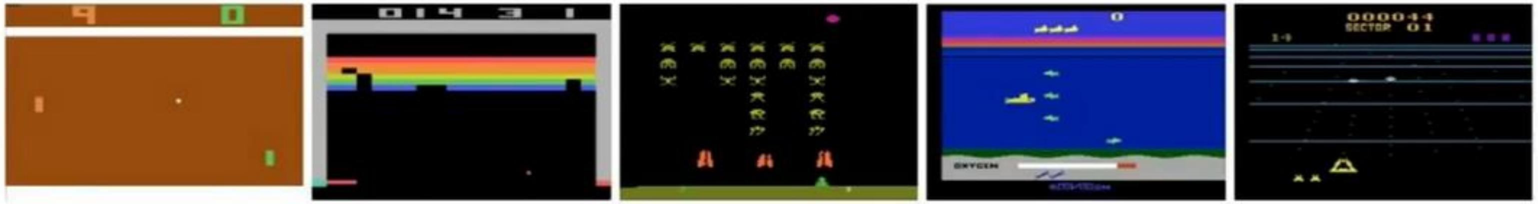
Backward Pass

Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i) \right]$$

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Case Study: Playing Atari Games



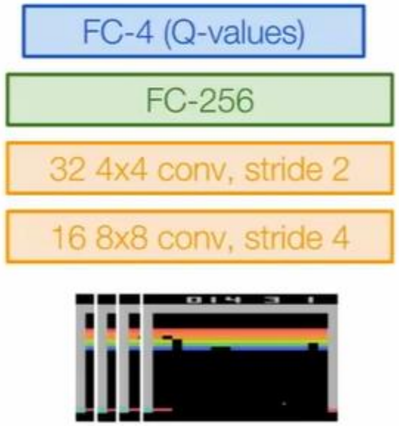
Objective: Complete the game with the highest score

State: Raw pixel inputs of the game state
Action: Game controls e.g. Left, Right, Up, Down
Reward: Score increase/decrease at each time step

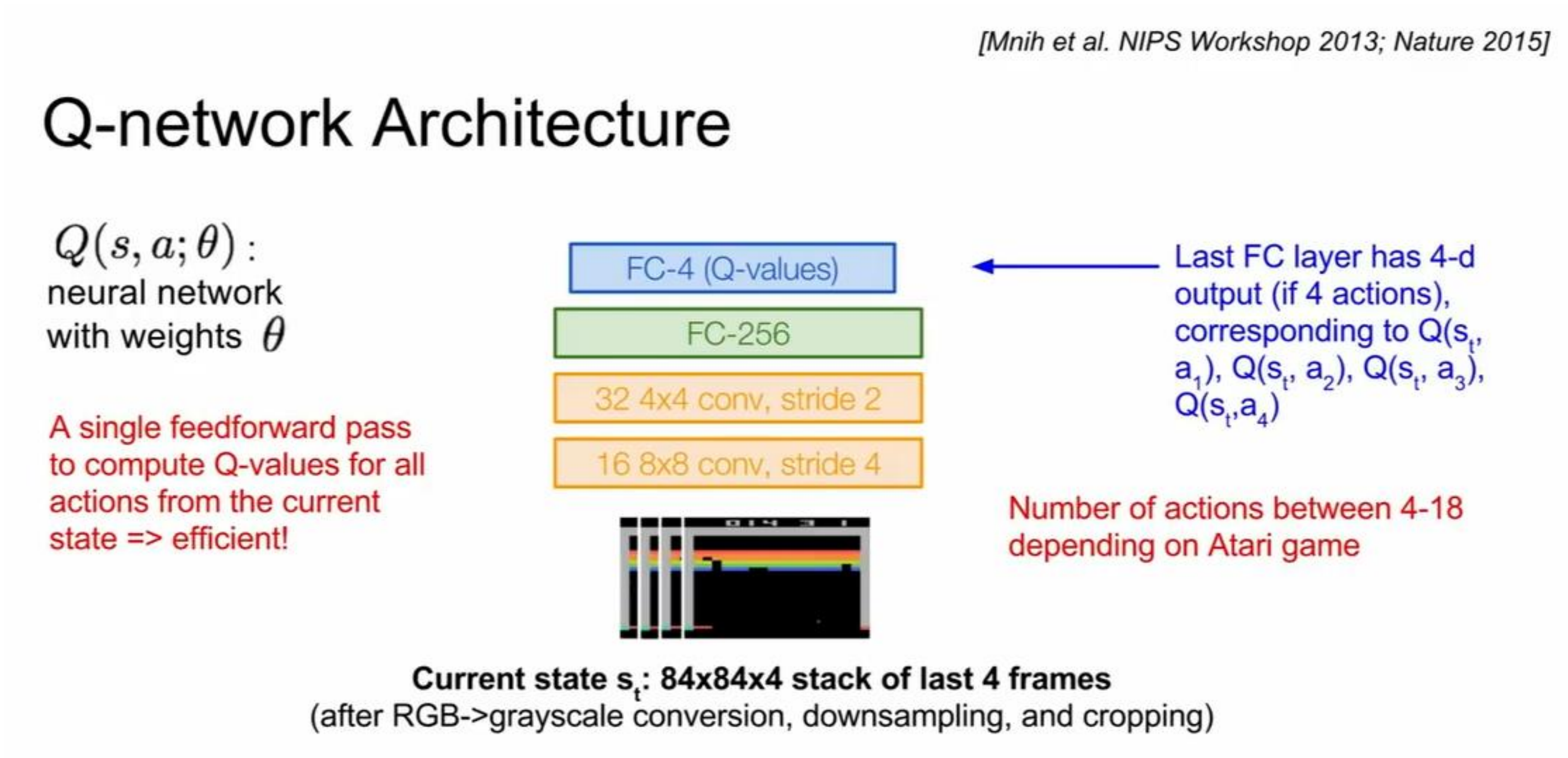
[Mnih et al. NIPS Workshop 2013; Nature 2015]

Q-network Architecture

$Q(s, a; \theta)$:
neural network
with weights θ



Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)


[Mnih et al. NIPS Workshop 2013; Nature 2015]
Q-network Architecture
$Q(s, a; \theta)$:
neural network
with weights θ
A single feedforward pass
to compute Q-values for all
actions from the current
state => efficient!
FC-4 (Q-values)
FC-256
32 4x4 conv, stride 2
16 8x8 conv, stride 4
Last FC layer has 4-d
output (if 4 actions),
corresponding to $Q(s_t, a_1)$, $Q(s_t, a_2)$, $Q(s_t, a_3)$, $Q(s_t, a_4)$
Number of actions between 4-18
depending on Atari game
Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Training the Q-network: Experience Replay

Learning from batches of consecutive samples is problematic:
- Samples are correlated => inefficient learning
- Current Q-network parameters determines next training samples (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) => can lead to bad feedback loops

Address these problems using **experience replay**
- Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
- Train Q-network on random minibatches of transitions from the replay memory, instead of consecutive samples

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Putting it together: Deep Q-Learning with Experience Replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

With small probability, select a random action (explore), otherwise select greedy action from current policy

[Mnih et al. NIPS Workshop 2013; Nature 2015]

Putting it together: Deep Q-Learning with Experience Replay

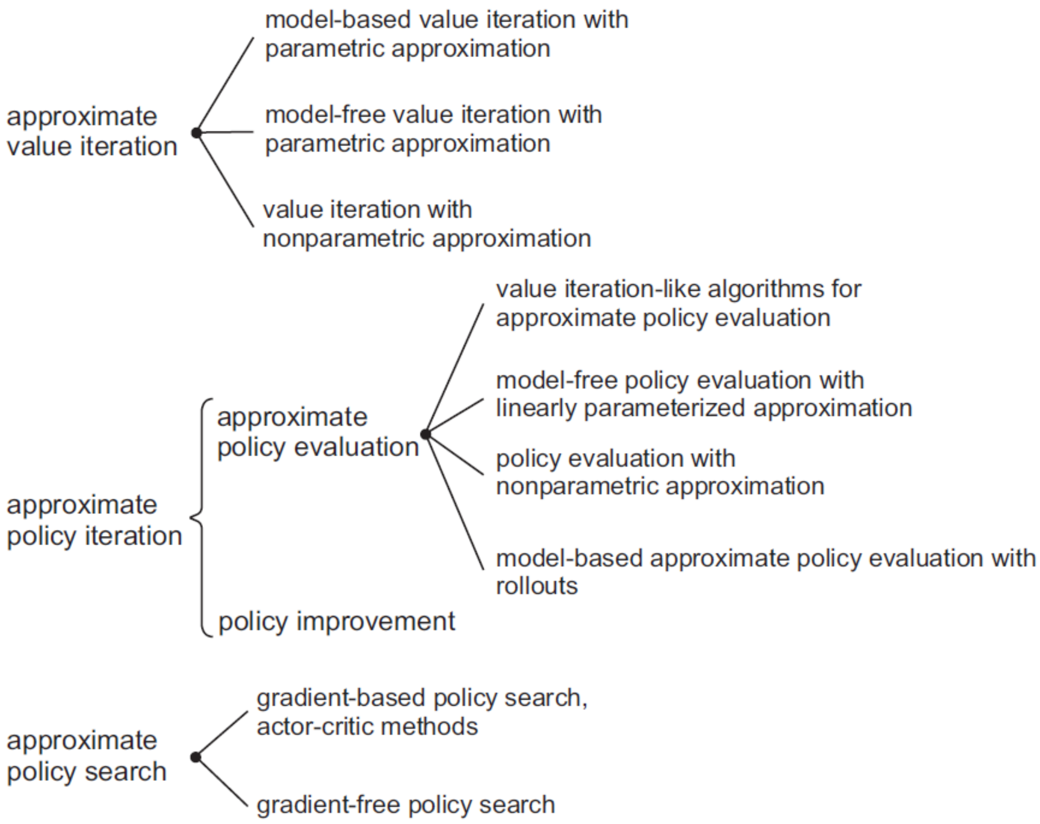
Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Experience Replay: Sample a random minibatch of transitions from replay memory and perform a gradient descent step

3 Gradiente de política

- Iremos tratar a seguir uma das técnicas aproximadas em aprendizado por reforço, dentre várias disponíveis, conforme ilustrado na taxonomia a seguir.



Policy Gradients

What is a problem with Q-learning?
The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand
Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | \pi_\theta\right]$$

We want to find the optimal policy $\theta^* = \arg\max_\theta J(\theta)$

How can we do this?

Gradient ascent on policy parameters!

REINFORCE algorithm

Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right]$$
$$= \int_{\tau} r(\tau) p(\tau;\theta) \mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \ldots)$

REINFORCE algorithm

Expected reward:
$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] = \int_\tau r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Now let's differentiate this: $\nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)\mathrm{d}\tau$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\dfrac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

If we inject this back:

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right)p(\tau;\theta)\mathrm{d}\tau = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can estimate with Monte Carlo sampling

- With a nice trick, a gradient of expectations was transformed into an expectation of gradients.

REINFORCE algorithm

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau;\theta) = \prod_{t\geq 0} p(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)$

Thus: $\log p(\tau;\theta) = \sum_{t\geq 0} \log p(s_{t+1}|s_t,a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on transition probabilities!

Therefore when sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta (a_t | s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta (a_t | s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta (a_t | s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn’t necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

4 Algoritmo Actor-Critic

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t)$

Actor-Critic Algorithm

Problem: we don't know Q and V. Can we learn them?

Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the **advantage function** how much an action was better than expected

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ
For iteration=1, 2 … **do**
    Sample m trajectories under the current policy
    $\Delta\theta \leftarrow 0$
    For i=1, …, m **do**
        For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

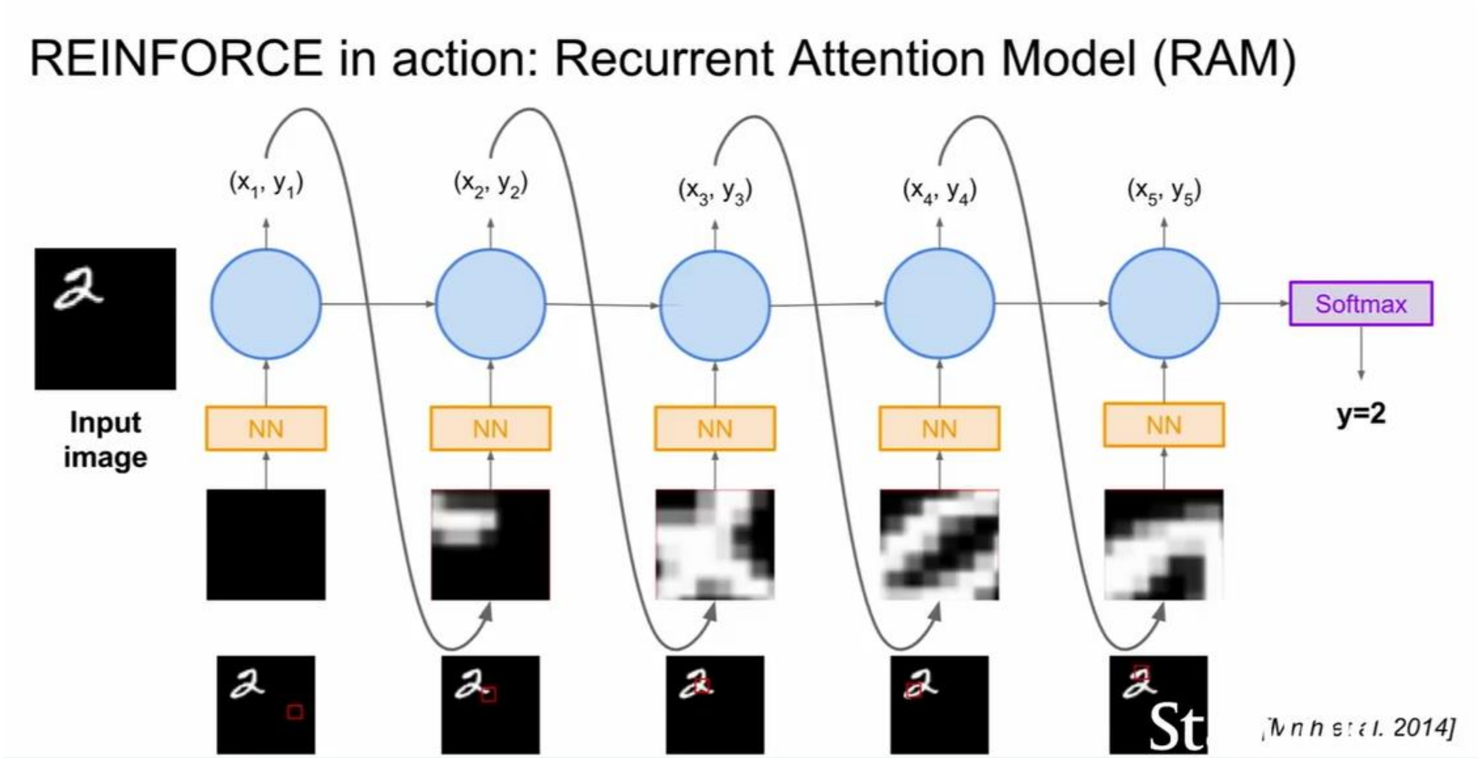
$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

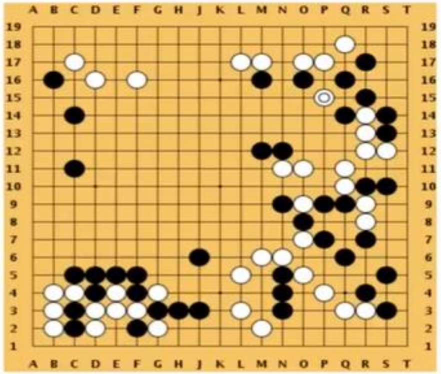
End for

5 Aplicações

- Há muitas aplicações relevantes, sendo que iremos destacar aqui apenas duas, bem representativas do potencial prático do aprendizado por reforço.



More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016]

6 Vídeos associados

AlphaGo Zero:

https://www.youtube.com/watch?v=MgowR4pq3e8

Digital Creatures Learn To Walk | Two Minute Papers #8:

https://www.youtube.com/watch?v=kQ2bqz3HPJE

Emergence of Locomotion Behaviours in Rich Environments:

https://www.youtube.com/watch?v=hx_bgoTF7bs

Google DeepMind's Deep Q-learning playing Atari Breakout:

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Google's self-learning AI AlphaZero masters chess in 4 hours:

https://www.youtube.com/watch?v=0g9SlVdv1PY

Phase-Functioned Neural Networks for Character Control:

https://www.youtube.com/watch?v=Ul0Gilv5wvY