Deep Learning – Parte 8
Introdução ao Aprendizado por Reforço

Índice Geral

1 Conceitos preliminares e potencial de aplicação 2

2 Uma preparação para o formalismo matemático 26

3 Formalismo matemático 30

4 Deep Q-Learning 42

5 Gradiente de política 52

6 Algoritmo Actor-Critic 55

7 Aplicação num cenário MNIST-like 58

8 Vídeos de apoio 59

9 Projetos computacionais e textos de apoio 60

1 Conceitos preliminares e potencial de aplicação

- Nota: Boa parte do 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.
- Recomenda-se a leitura do seguinte *review*, com os principais avanços na área de *deep reinforcement learning*:
  - ✓ Ivanov, S. "Modern Deep Reinforcement Learning Algorithms", arXiv:1906.10025v2, July 2019.
- Há diversos outros vídeos associados a este material, que ilustram bem o potencial da técnica, como por exemplo (também usados como fonte em alguns slides):

  https://www.youtube.com/watch?v=JgvyzIkgxF0
  https://www.youtube.com/watch?v=zR11FLZ-O9M

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

- O aprendizado supervisionado envolve aquisição de conhecimento a partir de exemplos de entrada-saída. **Aprendizado com base em exemplos**.

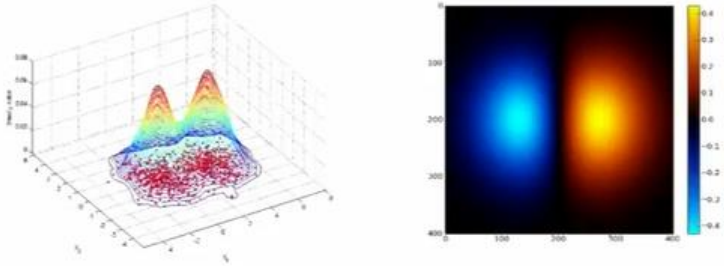
So far… Unsupervised Learning

Data: x
Just data, no labels!

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

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

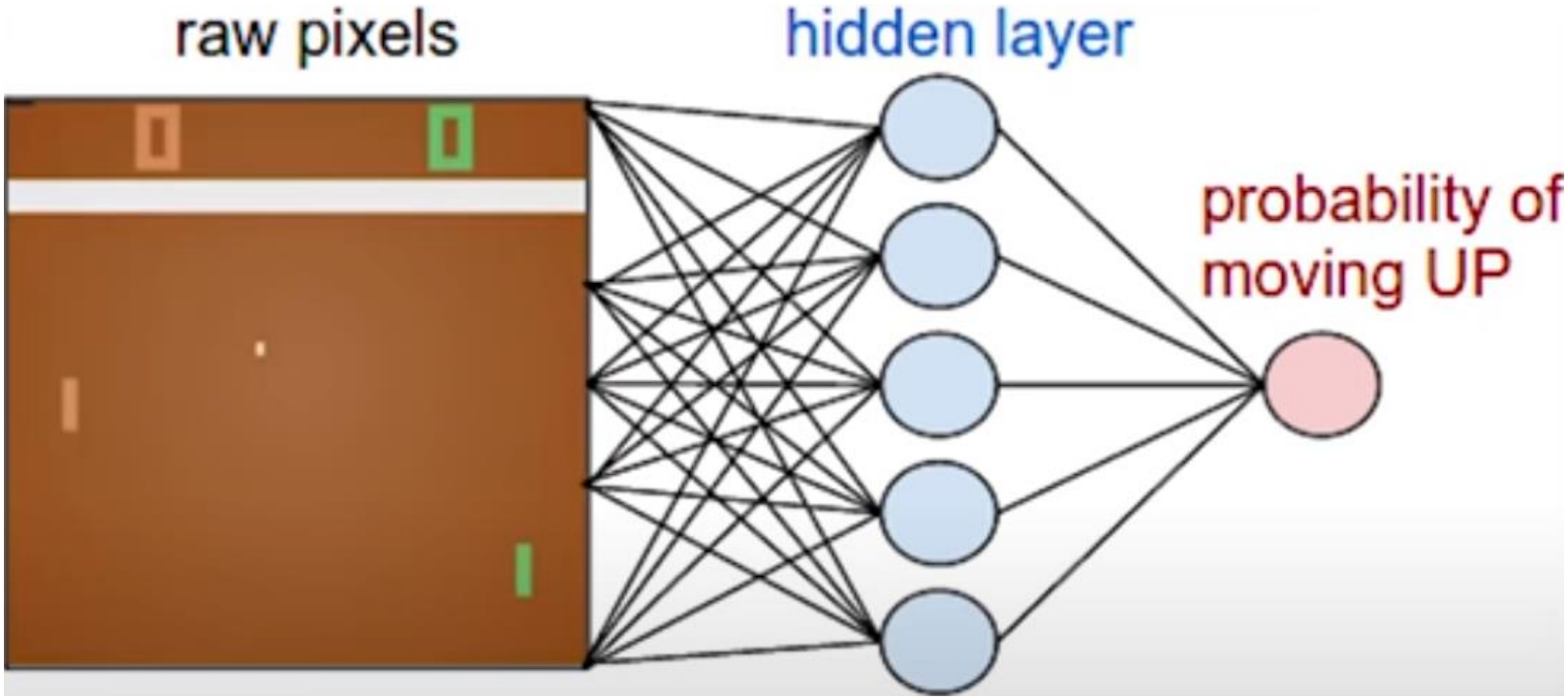
1-d density estimation



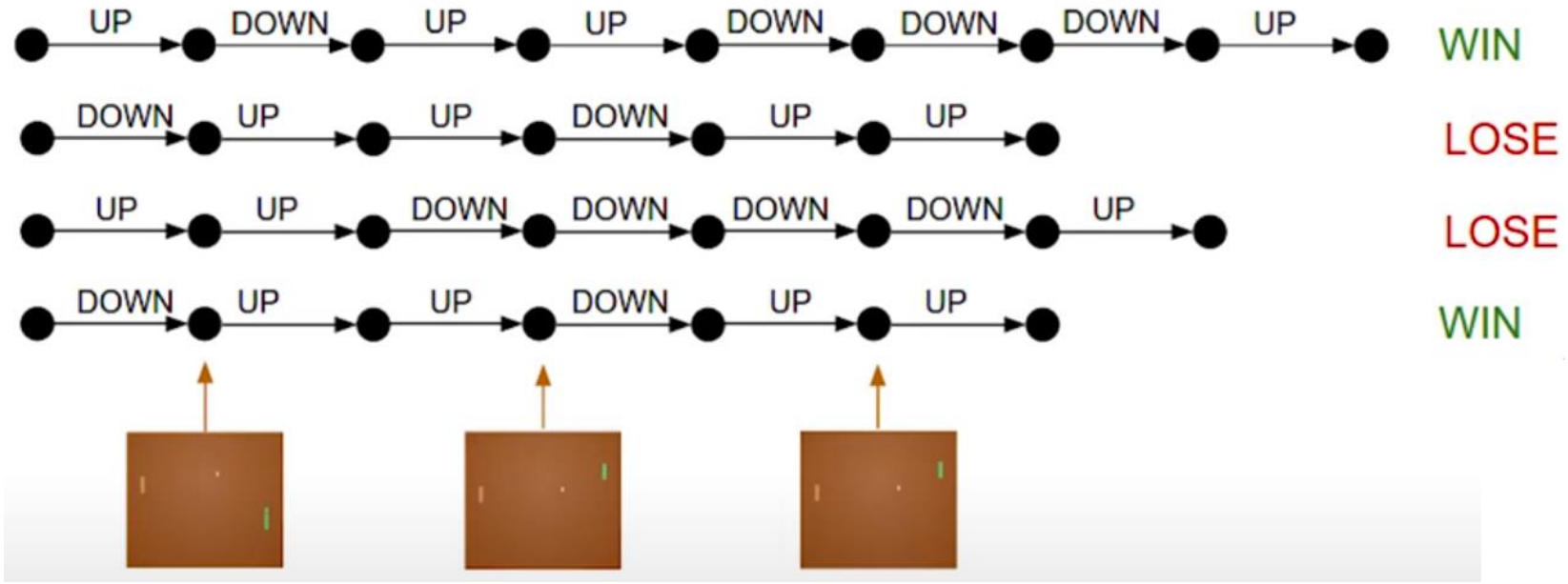
2-d density estimation

- O aprendizado não-supervisionado é caracterizado por não ter uma saída desejada, mas também está otimizando algum critério. Continua a ter um objetivo a ser otimizado.

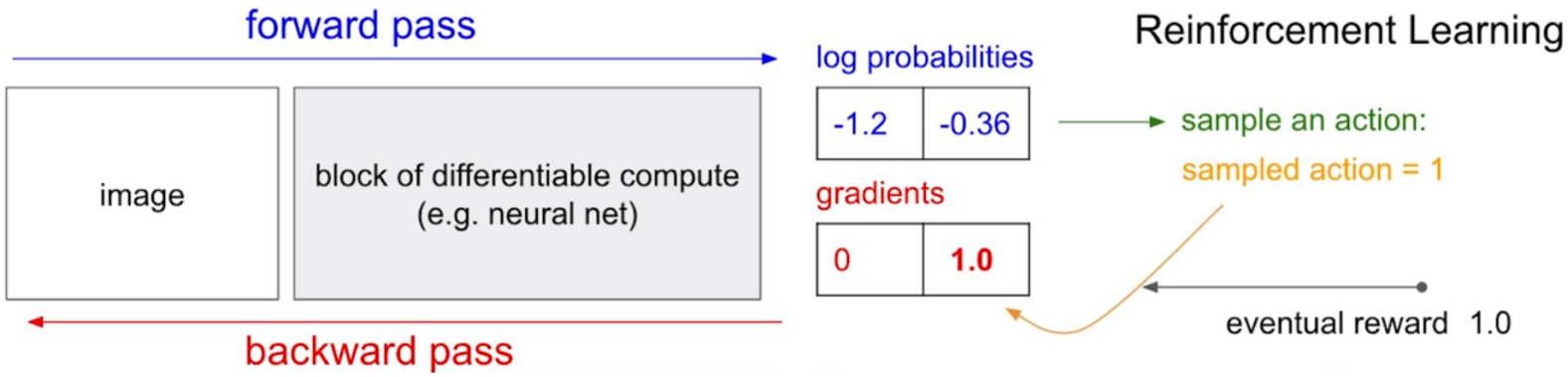
- O aprendizado por reforço envolve a aquisição de conhecimento a partir da reação aos efeitos (punição e recompensa) da atuação sequencial de um agente num ambiente. **Aprendizado com base na experiência** (possivelmente explorando o poder da simulação computacional).



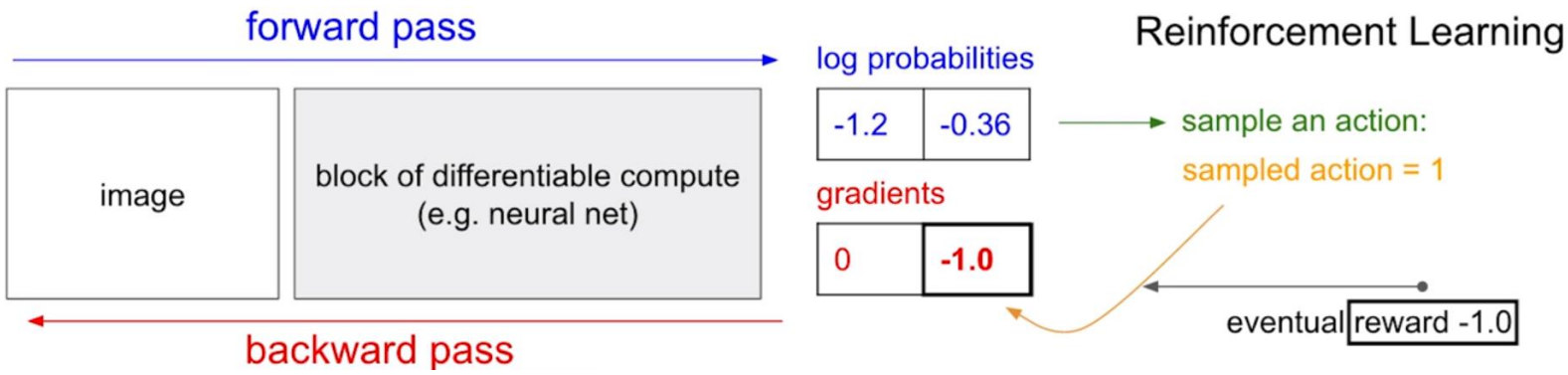
Fonte: https://www.youtube.com/watch?v=JgvyzIkgxF0



- Fontes:
  - http://karpathy.github.io/2016/05/31/rl/
  - https://www.youtube.com/watch?v=JgvyzIkgxF0



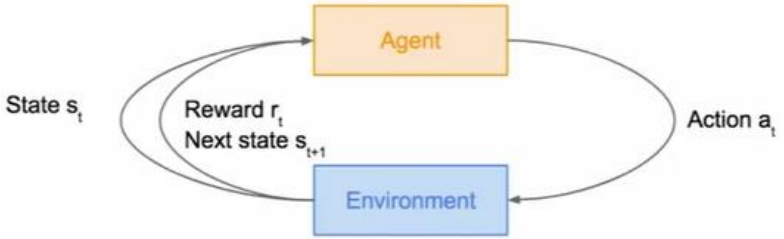
- Fontes:
  - http://karpathy.github.io/2016/05/31/rl/
  - https://www.youtube.com/watch?v=JgvyzIkgxF0



- Fontes:
  - http://karpathy.github.io/2016/05/31/rl/
  - https://www.youtube.com/watch?v=JgvyzIkgxF0

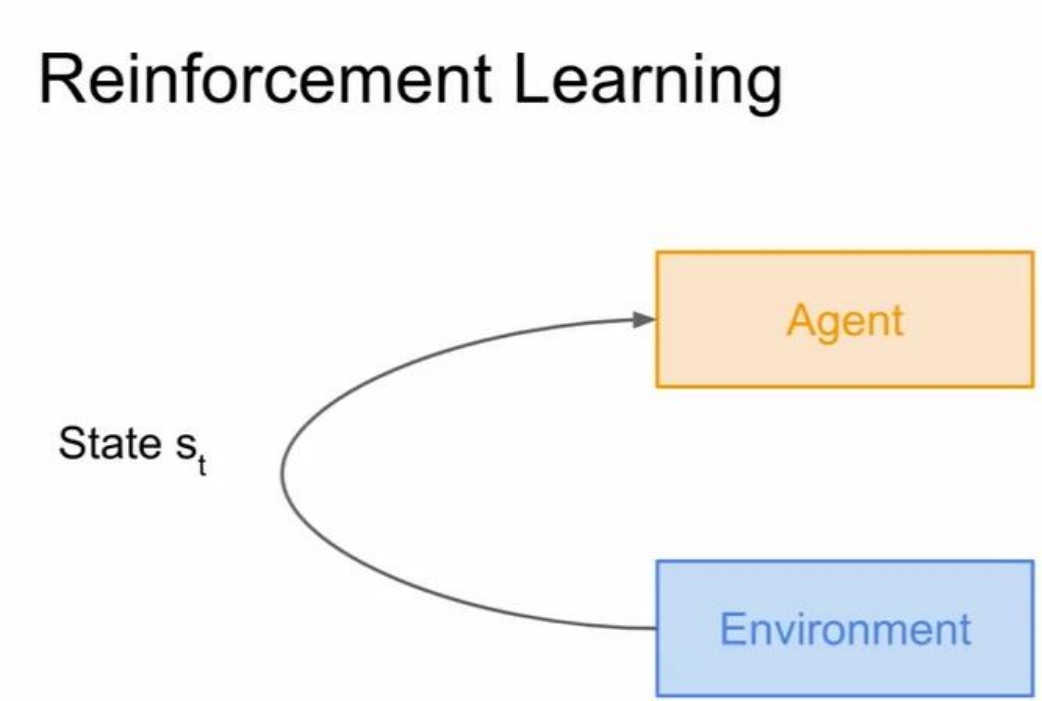
Today: Reinforcement Learning

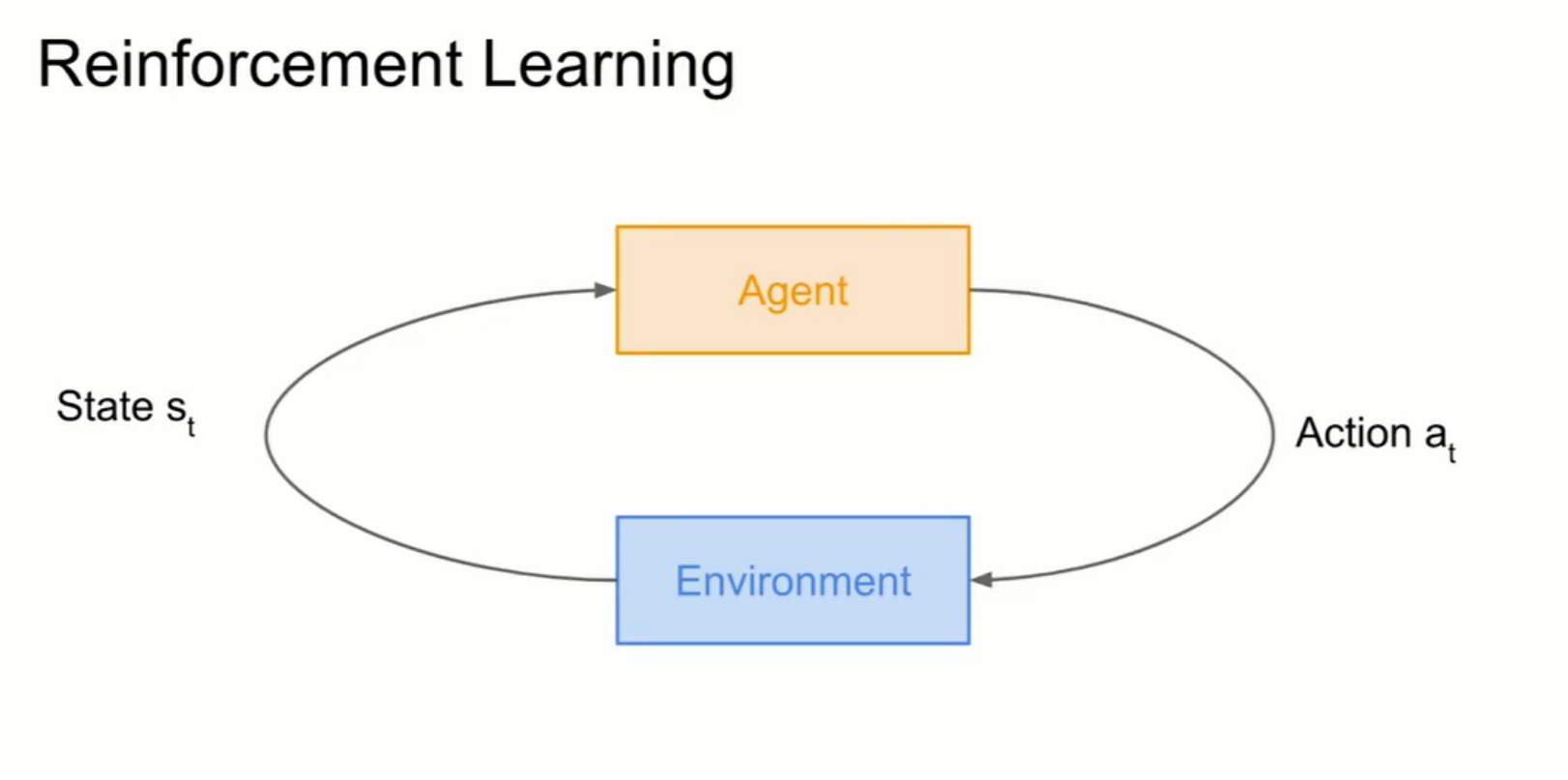
Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

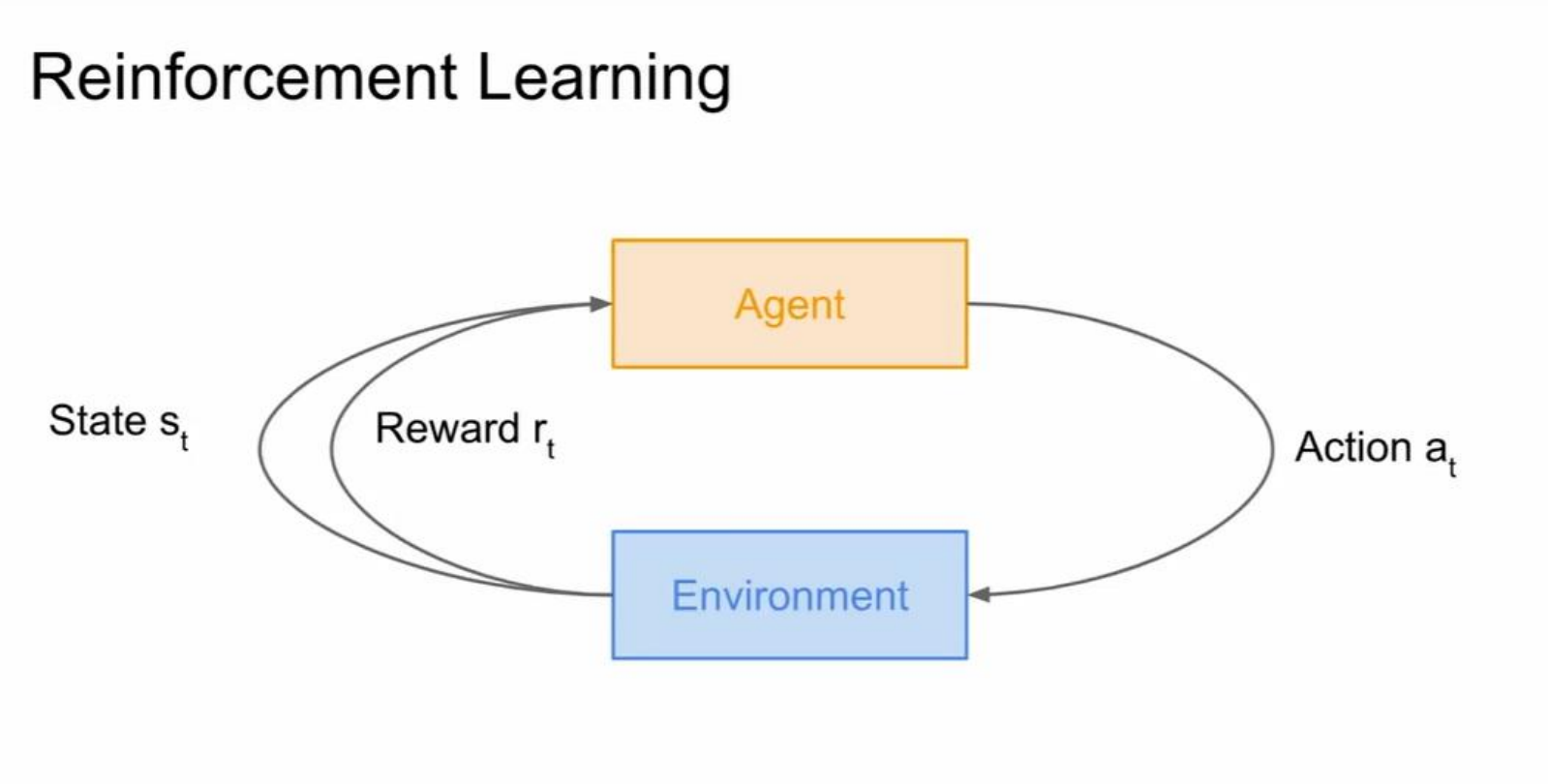


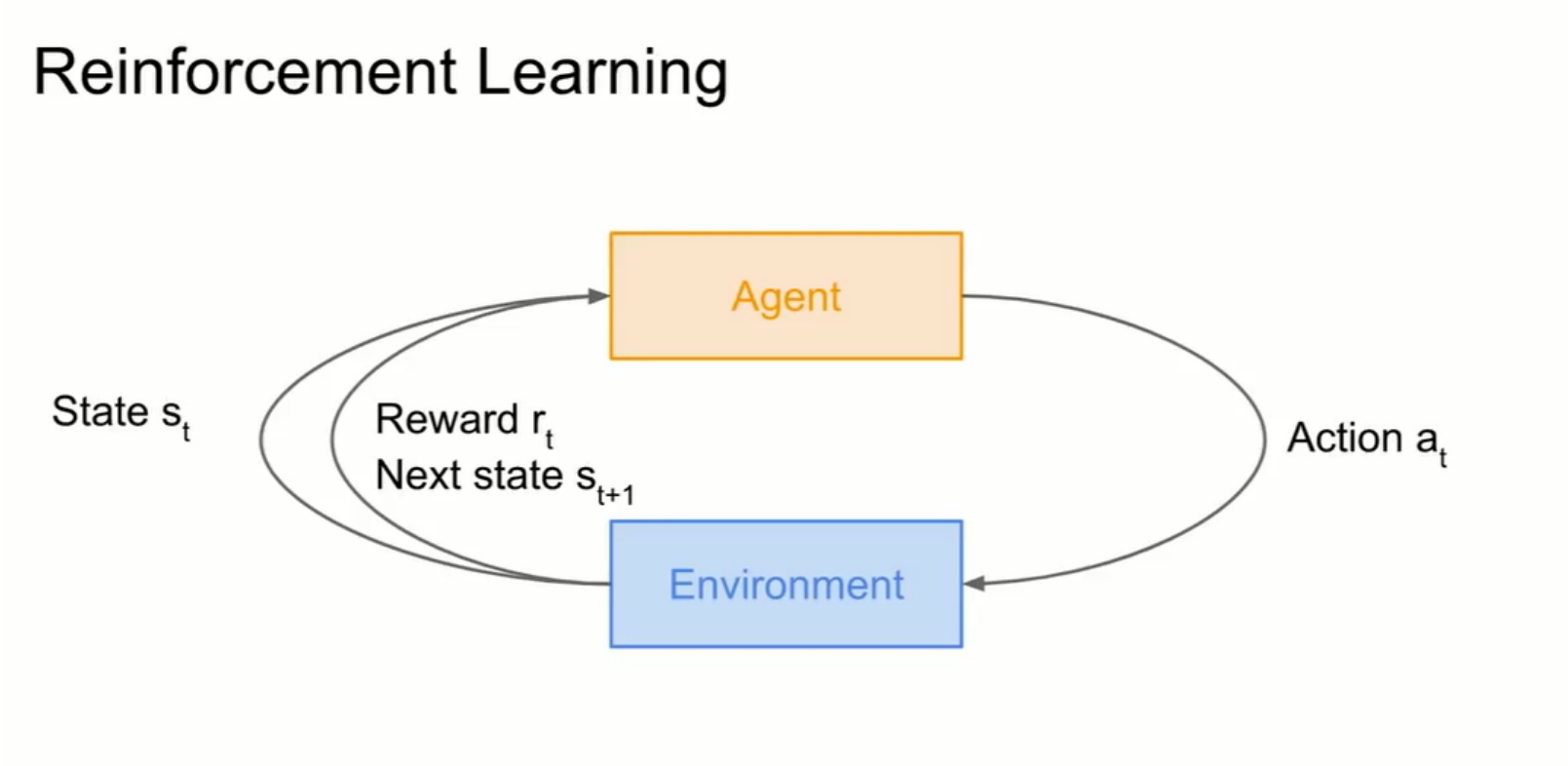
Goal: Learn how to take actions in order to maximize reward


Reinforcement Learning
Agent
Environment


Reinforcement Learning
Agent
State s_t
Environment

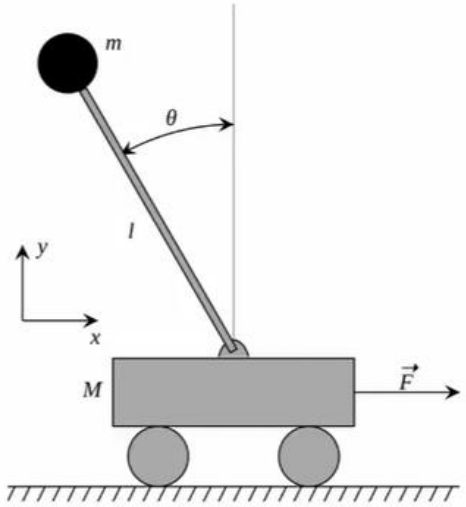

Reinforcement Learning
Agent
State s_t
Action a_t
Environment


Reinforcement Learning
Agent
State s_t
Reward r_t
Action a_t
Environment


Reinforcement Learning
Agent
State s_t
Reward r_t
Next state s_{t+1}
Action a_t
Environment

- Características do cenário de aprendizado:
  - Ambiente: Totalmente observável × Parcialmente observável
  - Agente: Um único agente × Múltiplos agentes
  - Ambiente / Ações: Determinístico(as) × Estocástico(as)
  - Ambiente: Estático × Dinâmico
  - Ambiente / Ações: Discreto(as) × Contínuo(as)
- Principais componentes sob a perspectiva do agente tomador de decisão:
  - Política do agente: função que comanda o comportamento do agente;
  - Funções de qualidade: permitem avaliar o estado (do ambiente) e a ação (do agente), geralmente o par (estado, ação);
  - Modelo de mundo: representação que o agente tem do ambiente. Pode estar implícito na política e nas funções de qualidade.
- **A melhor estratégia para o agente é sempre tomar a ação que maximiza a recompensa futura acumulada (que pode ser uma medida aproximada).**

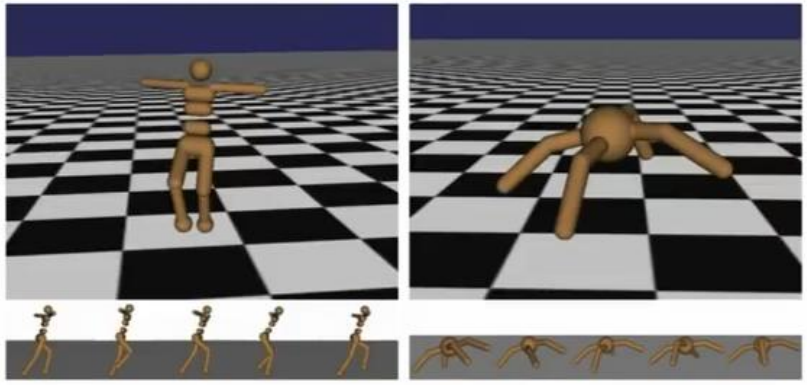
Cart-Pole Problem



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

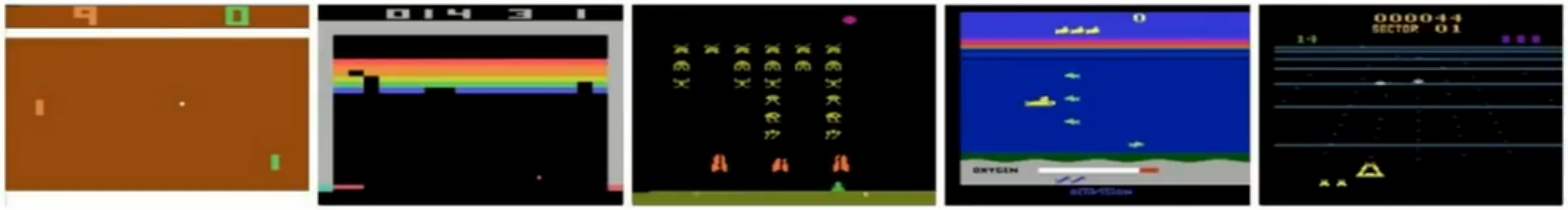
Robot Locomotion



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

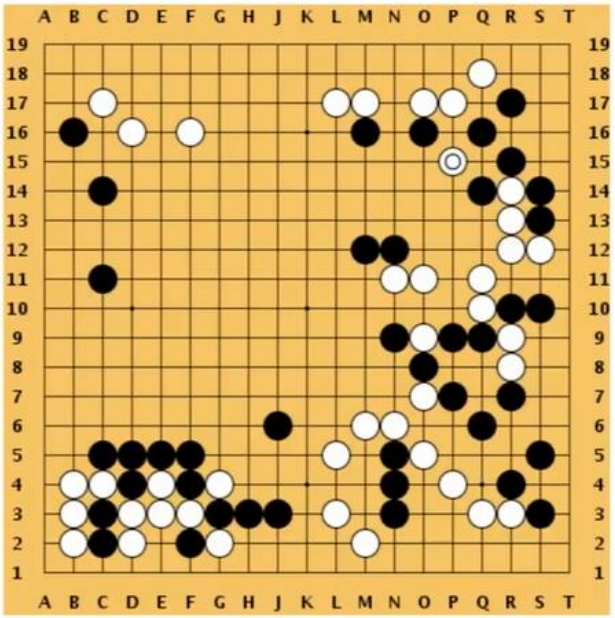
Atari Games



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

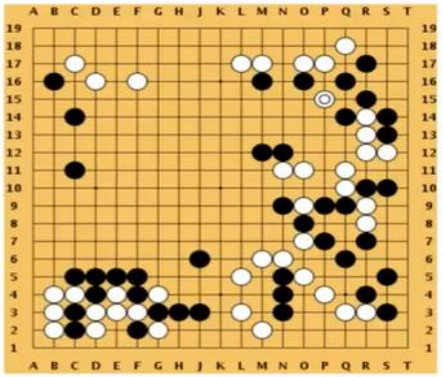
Go



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

More policy gradients: AlphaGo



Overview:

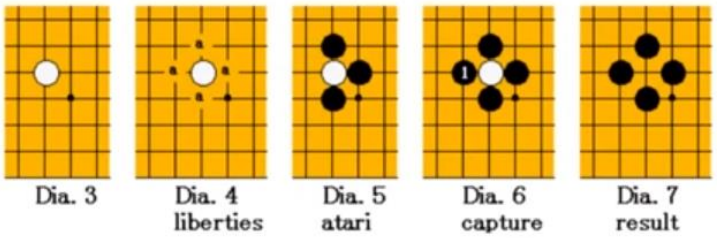
- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

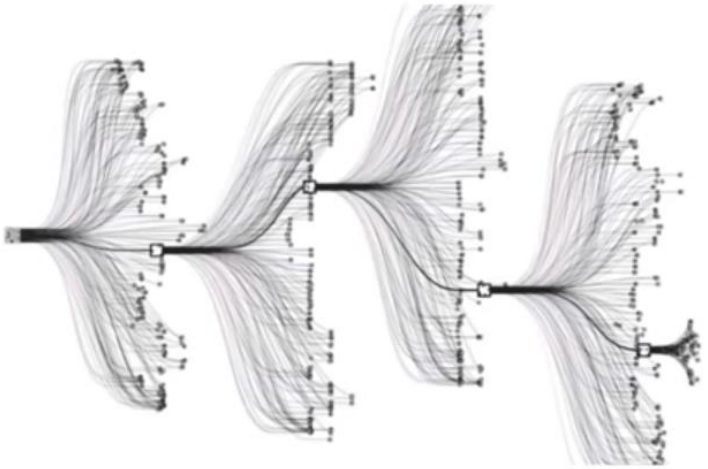
How to beat the Go world champion:

- 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]

Game of Go





Game size	Board size N	3^N	Percent legal	legal game positions (A094777)[11]
1×1	1	3	33%	1
2×2	4	81	70%	57
3×3	9	19,683	64%	12,675
4×4	16	43,046,721	56%	24,318,165
5×5	25	8.47×10^{11}	49%	4.1×10^{11}
9×9	81	4.4×10^{38}	23.4%	1.039×10^{38}
13×13	169	4.3×10^{80}	8.66%	3.72497923×10^{79}
19×19	361	1.74×10^{172}	1.196%	$2.08168199382\times10^{170}$

Fonte: https://www.youtube.com/watch?v=zR11FLZ-O9M

- DeepMind AlphaStar: Real-Time Strategy Game

StarCraft II Deep Reinforcement Learning Agent

- ✓ Teoria de jogos
- ✓ Informação imprecisa / incompleta
- ✓ Planejamento de longo prazo
- ✓ Tomada de decisão em tempo real
- ✓ Gigantesco espaço de ações candidatas

https://www.youtube.com/watch?v=98V6PnwVXCc

- Duas etapas:
  - ✓ Anonymized game replays (treinamento supervisionado)
  - ✓ Real-time gameplay vs itself (os agentes jogaram por 200 anos)

How AlphaStar is trained

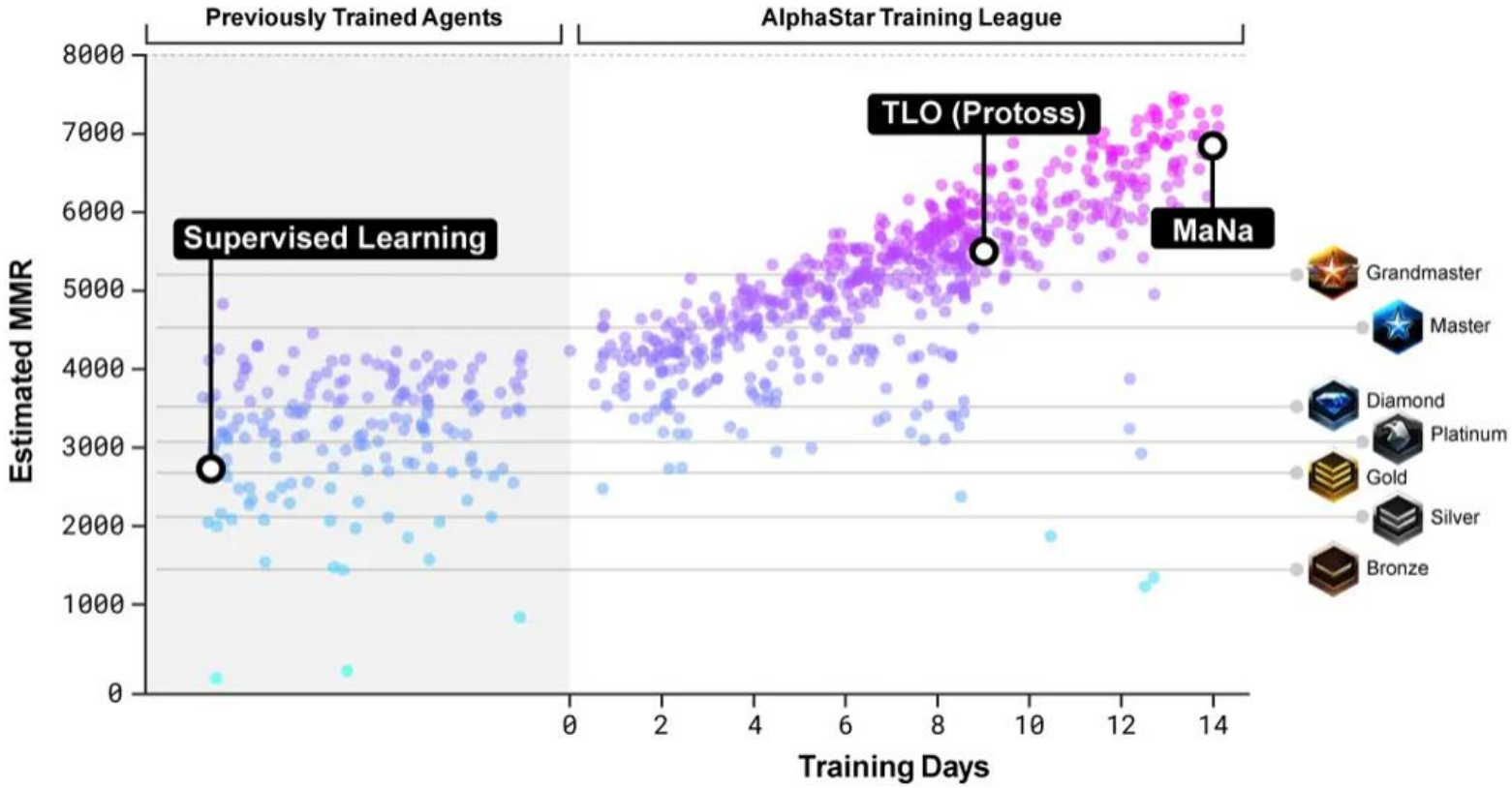
AlphaStar's behaviour is generated by a deep neural network that receives input data from the raw game interface (a list of units and their properties), and outputs a sequence of instructions that constitute an action within the game. More specifically, the neural network architecture applies a transformer torso to the units (similar to relational deep reinforcement learning), combined with a deep LSTM core, an auto-regressive policy head with a pointer network, and a centralised value baseline. We believe that this advanced model will help with many other challenges in machine learning research that involve long-term sequence modelling and large output spaces such as translation, language modelling and visual representations.

AlphaStar also uses a novel multi-agent learning algorithm. The neural network was initially trained by supervised learning from anonymised human games released by Blizzard. This allowed AlphaStar to learn, by imitation, the basic micro and macro-strategies used by players on the StarCraft ladder. This initial agent defeated the built-in "Elite" level AI - around gold level for a human player - in 95% of games.

The neural network weights of each agent are updated by reinforcement learning from its games against competitors, to optimise its personal learning objective. The weight update rule is an efficient and novel off-policy actor-critic reinforcement learning algorithm with experience replay, self-imitation learning and policy distillation.

https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii

- O treinamento levou 14 dias empregando 16 TPUs por agente.



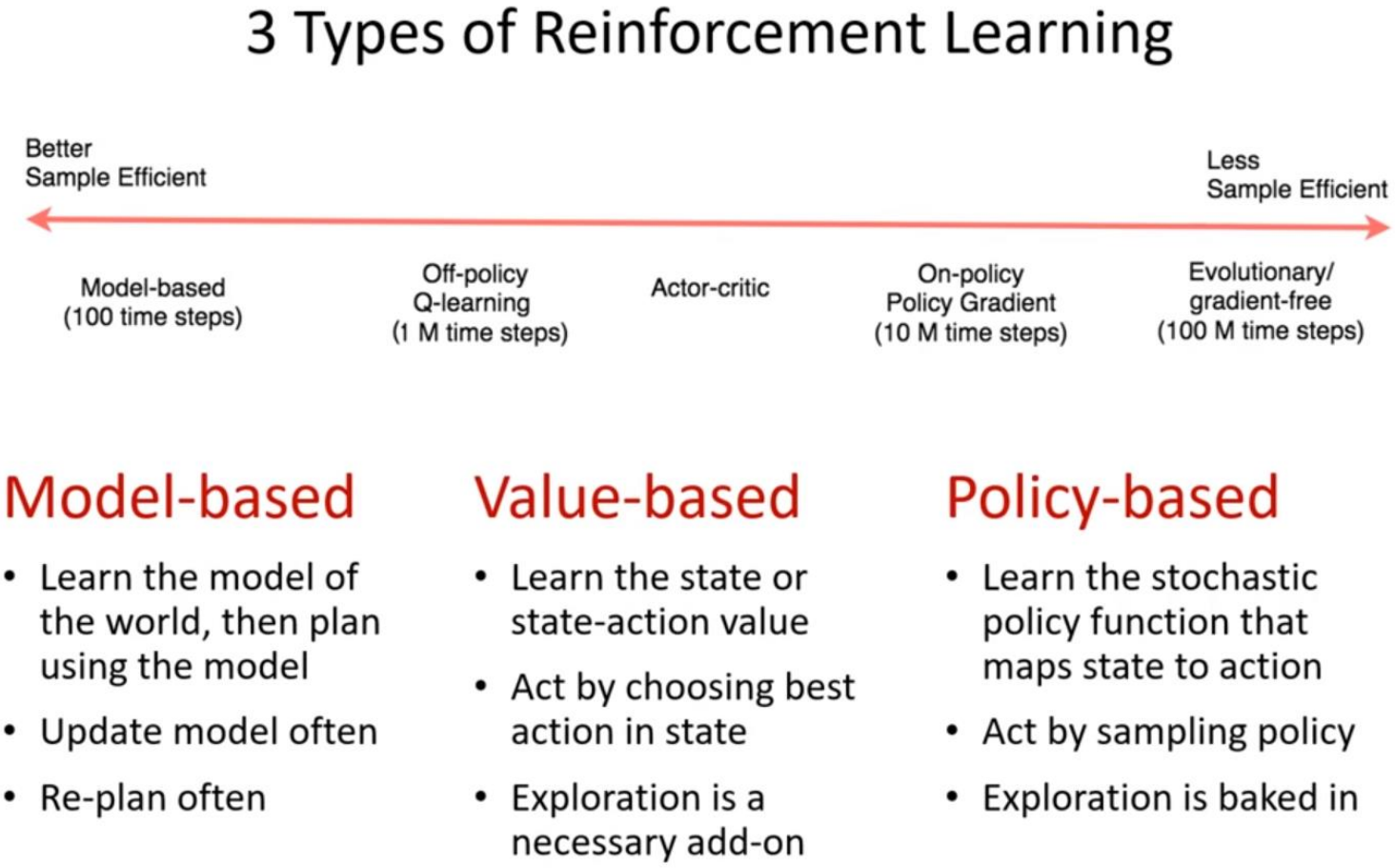
Fonte: https://www.youtube.com/watch?v=cUTMhmVh1qs

2 Uma preparação para o formalismo matemático

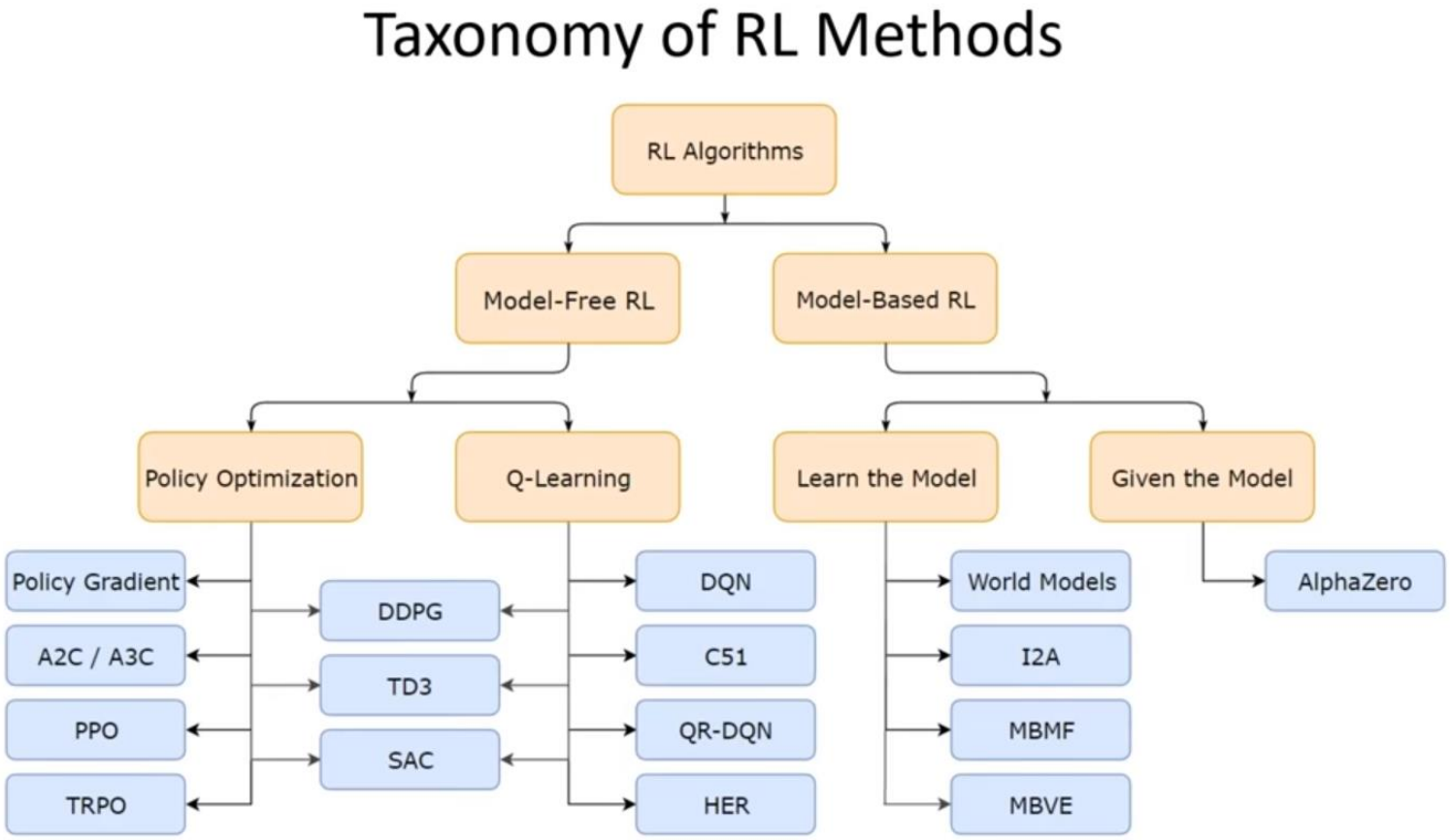
- A generalidade do aprendizado por reforço permite a sua aplicação tanto em domínio contínuo quanto em domínio discreto.
- Aplicações em robótica, em jogos de tabuleiro e em vários tipos de ambientes simulados têm se destacado.
- O aprendizado por reforço está fortemente conectado à programação dinâmica, ao controle ótimo e à teoria de jogos, sendo uma formulação geral para tomada de decisão sequencial.
- Um aspecto surpreendente da técnica, em algumas formulações, é o fato dela não requerer um conhecimento prévio do modelo de mundo e de sua estrutura, de modo que o ambiente pode ser tomado como uma caixa preta, o que abre infinitas possibilidades de aplicação e torna os algoritmos de aprendizado imediatamente transferíveis de um tarefa para outra, muitas vezes até sem a necessidade de ajuste de hiperparâmetros.

- Quando o objetivo do aprendizado por reforço é aprender uma política, busca-se uma função que mapeia todas as possíveis observações de um agente em ações sobre um ambiente (*on-policy*, *policy gradient*).
- Trata-se, portanto, de uma técnica de aprendizado de máquina. No entanto, diferente do cenário de aprendizado a partir de dados que caracteriza problemas de aprendizado supervisionado e não-supervisionado, o agente aprende a partir de sua experiência de interação com o ambiente.
- Esse diferencial é, ao mesmo tempo, um aspecto positivo e negativo da técnica. Se, por um lado, não requer dados de treinamento, requer a exploração continuada e ampla do ambiente e precisa lidar com recompensas esparsas e/ou com atraso.
- O uso conjunto de aprendizado por reforço e aprendizado profundo se popularizou com o algoritmo *Deep Q-Learning*, introduzido em:

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. “Playing Atari with Deep Reinforcement Learning”, arXiv:1312.5602, 2013.

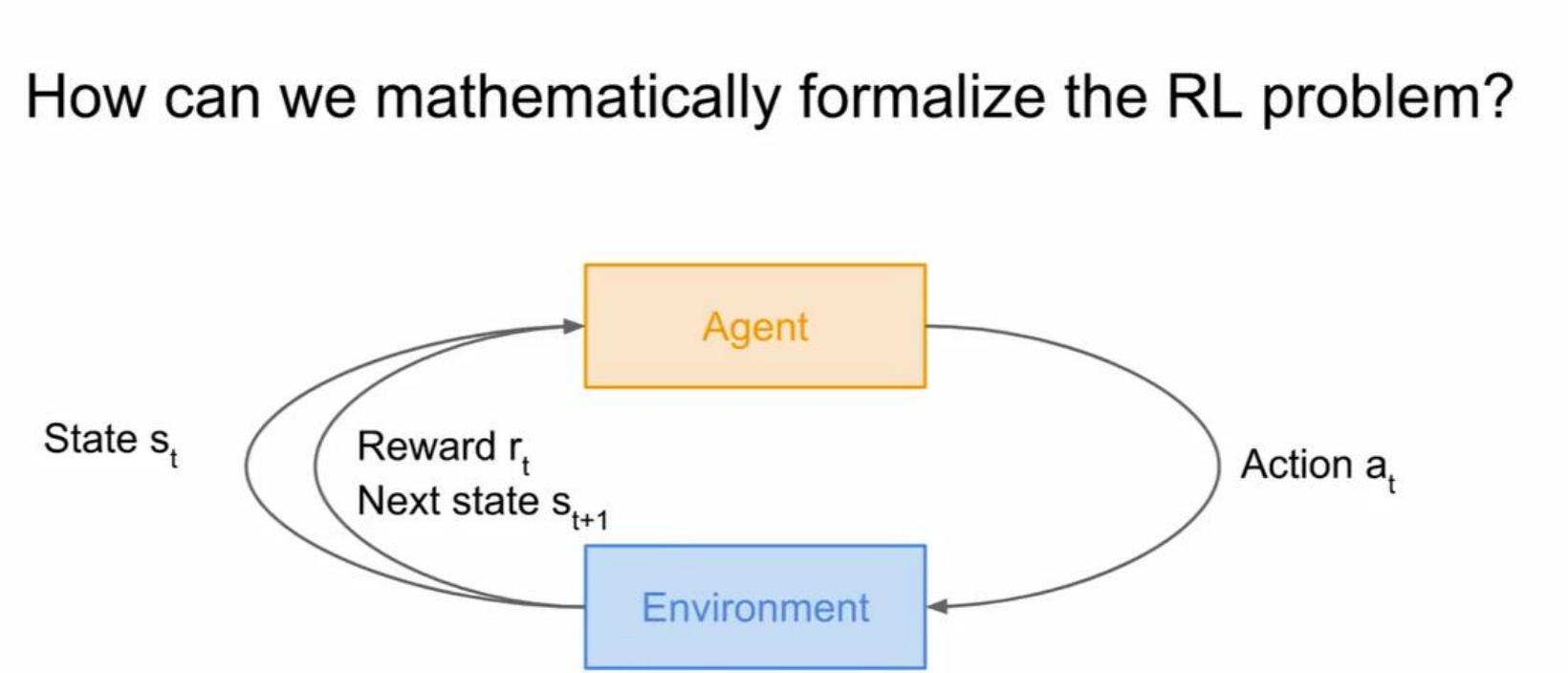


Fonte: https://www.youtube.com/watch?v=zR11FLZ-O9M



Link: https://spinningup.openai.com

3 Formalismo matemático



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

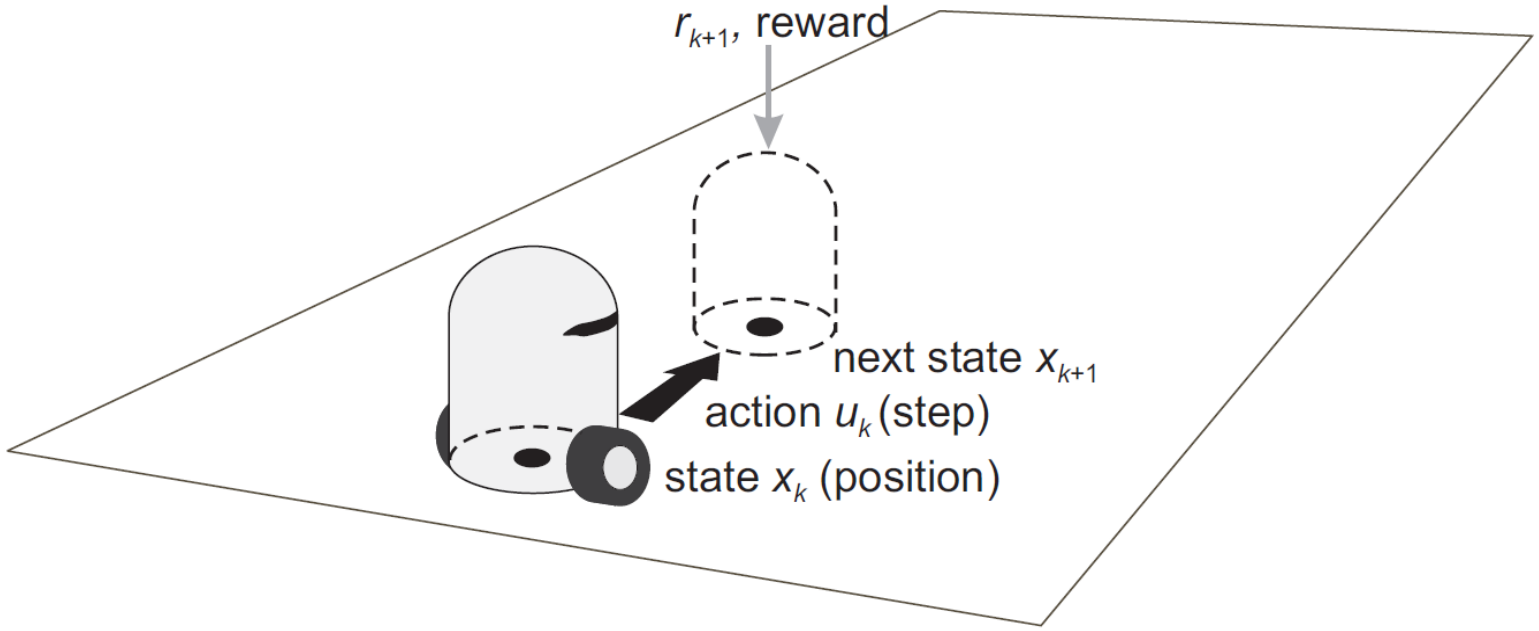


Figura 1 – Numa notação um pouco distinta, esta figura ilustra os conceitos do slide anterior num contexto de navegação de um robô por um ambiente.

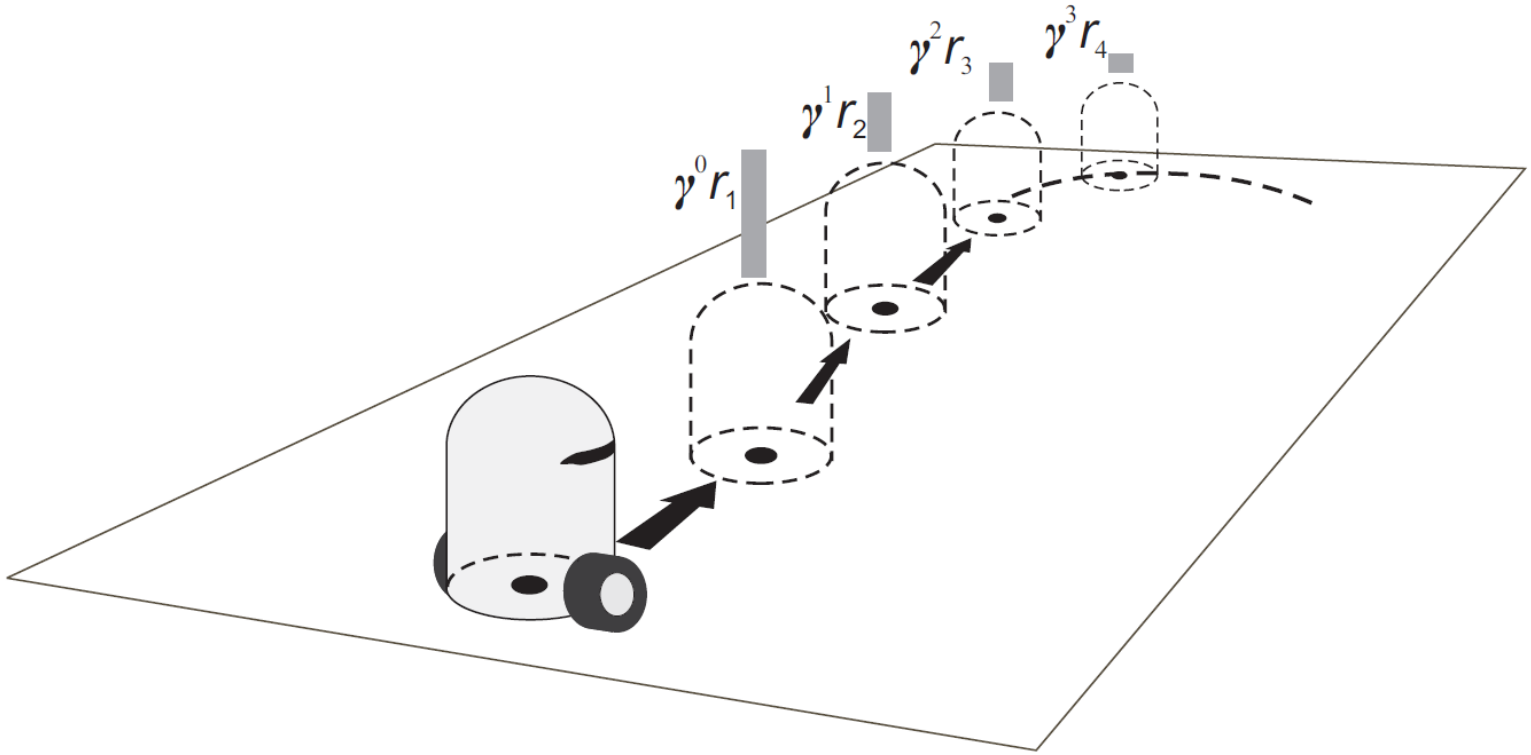
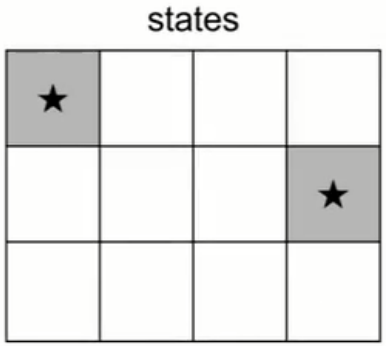


Figura 2 – Exemplo de recompensa acumulada empregando um termo de desconto.

A simple MDP: Grid World

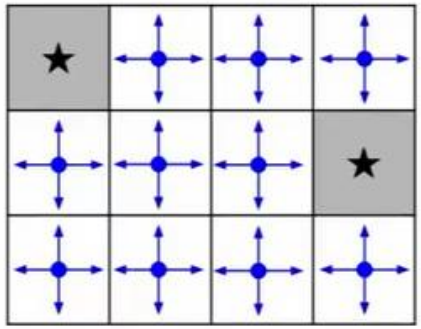
actions = {
1. right
2. left
3. up
4. down
}



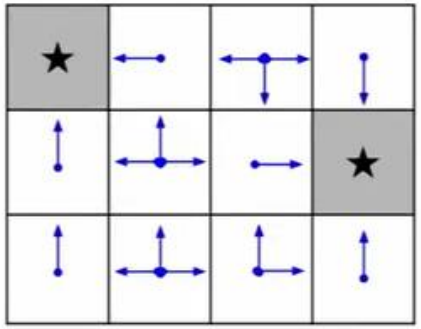
Set a negative "reward" for each transition (e.g. $r = -1$)

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 π^*

We want to find optimal policy π^* 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's Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

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!

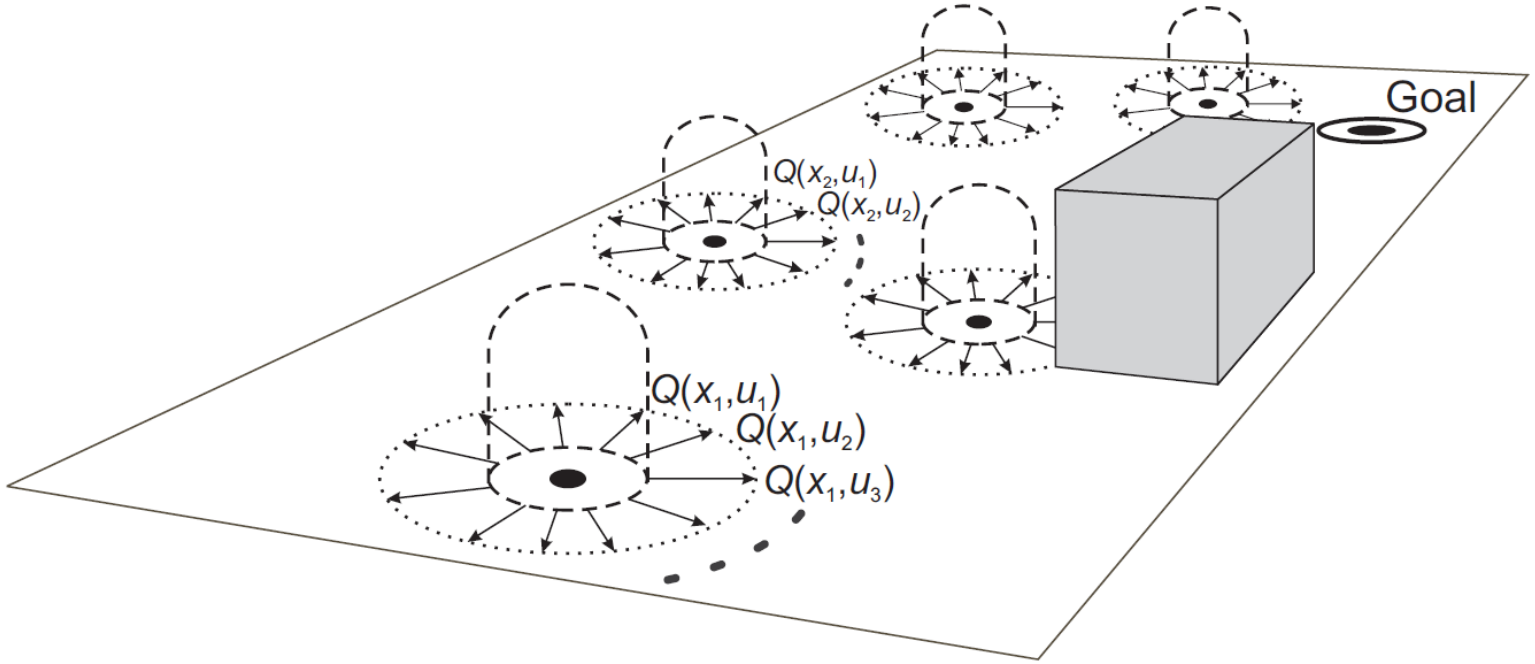


Figura 3 – Exemplo que ilustra a necessidade de se calcular $Q(s,a)$, na figura é usada a notação $Q(x_i,u_i)$, para cada estado possível do ambiente, o que torna o problema intratável para muitos estados e muitas ações por estado.

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

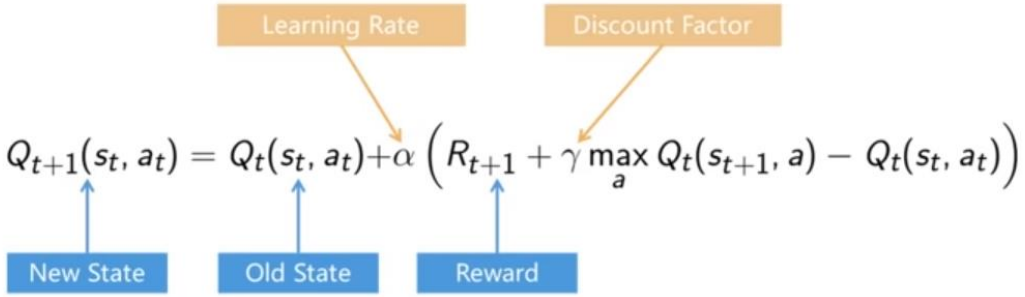
Iteratively try to make the Q-value close to the target value (y_i) it should have, if Q-function corresponds to optimal Q* (and optimal policy π^*)

Backward Pass

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

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

Q-Learning: Value Iteration



	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Fonte: https://www.youtube.com/watch?v=zR11FLZ-O9M

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

Case Study: Playing Atari Games

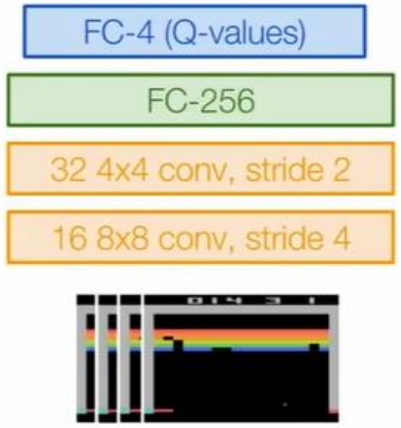
Objective: Complete the game with the highest score

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



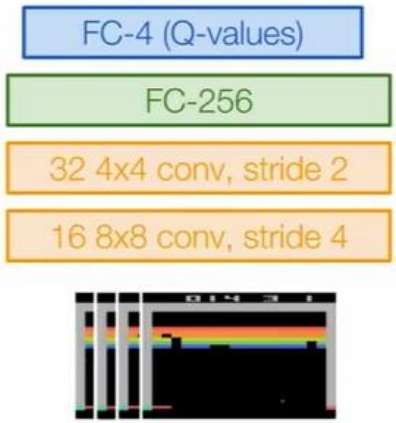
Current state s_t: 84x84x4 stack of last 4 frames
(after RGB->grayscale conversion, downsampling, and cropping)

[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!



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

- Equation 3 é a última equação do slide 40.

[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

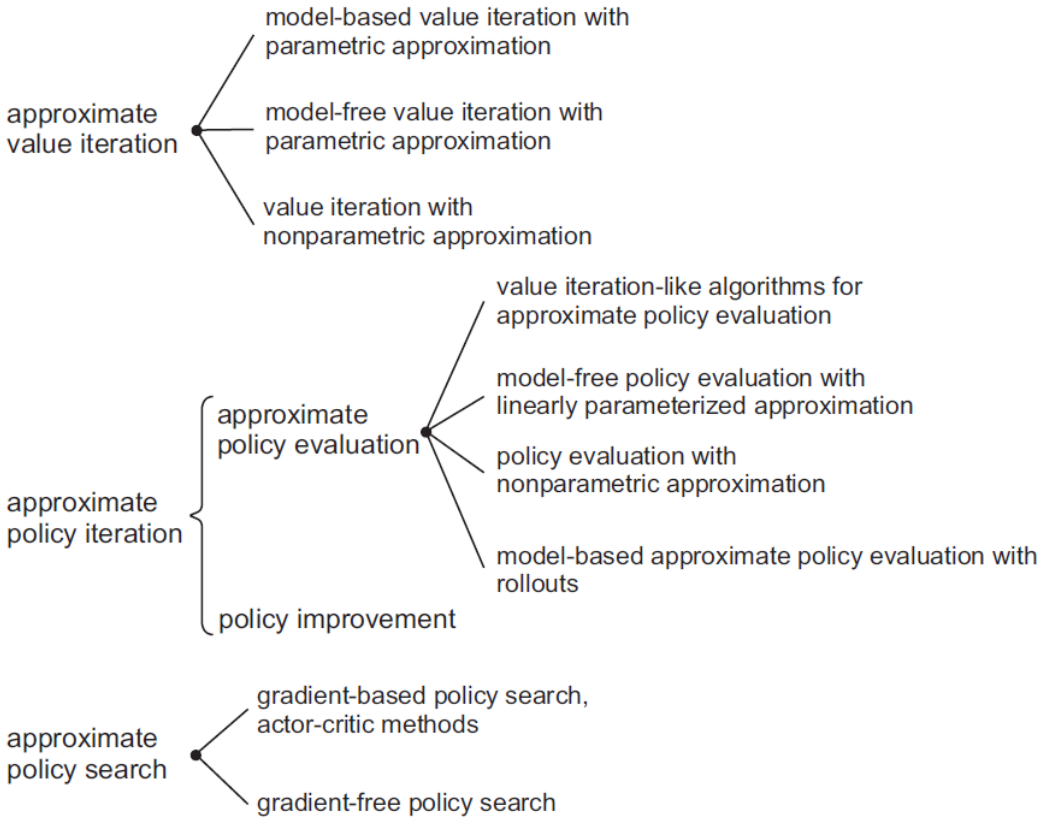
Algorithm 1 Deep Q-learning with Experience Replay

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

5 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!

6 Algoritmo Actor-Critic

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

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

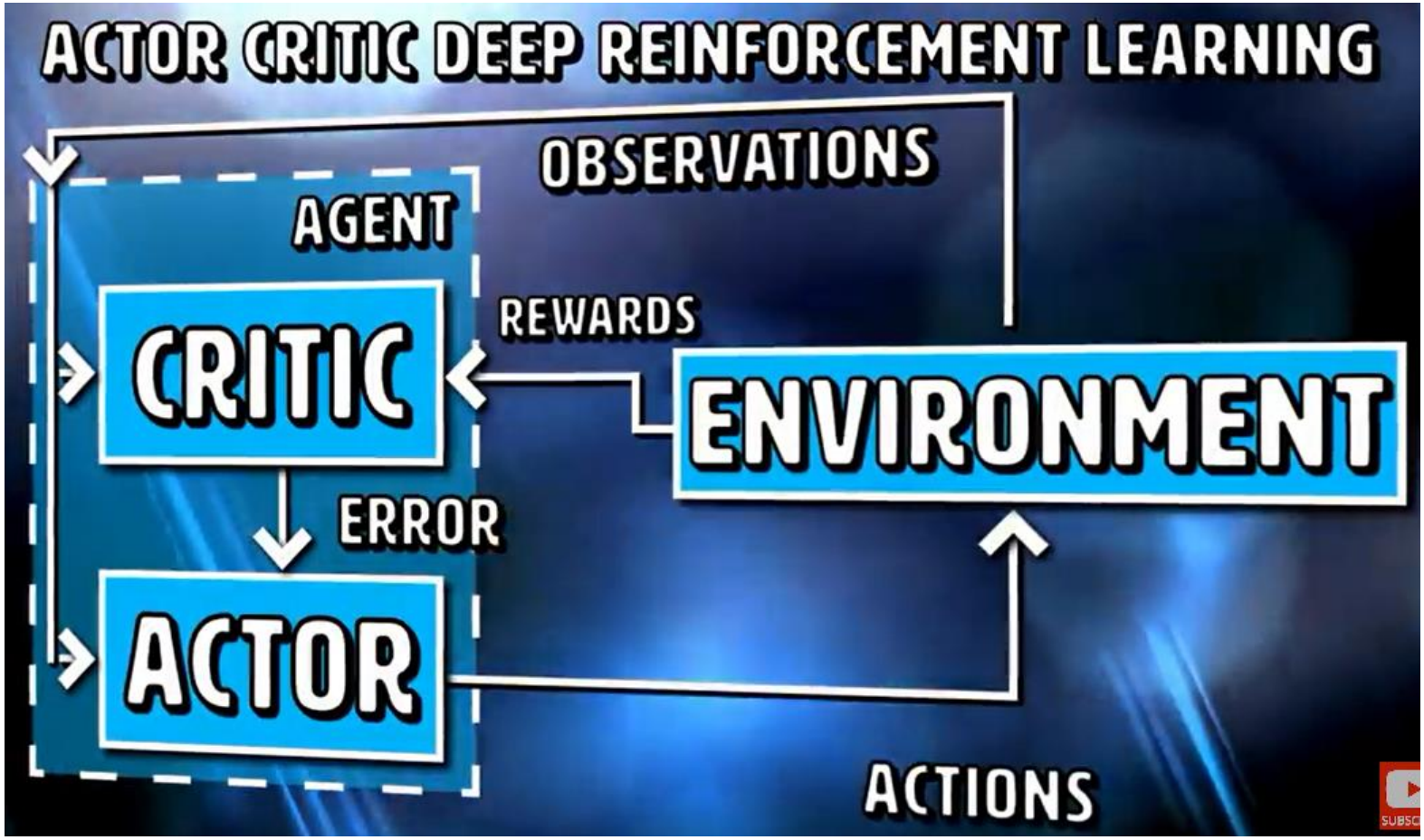
Actor-Critic Algorithm

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**

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

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

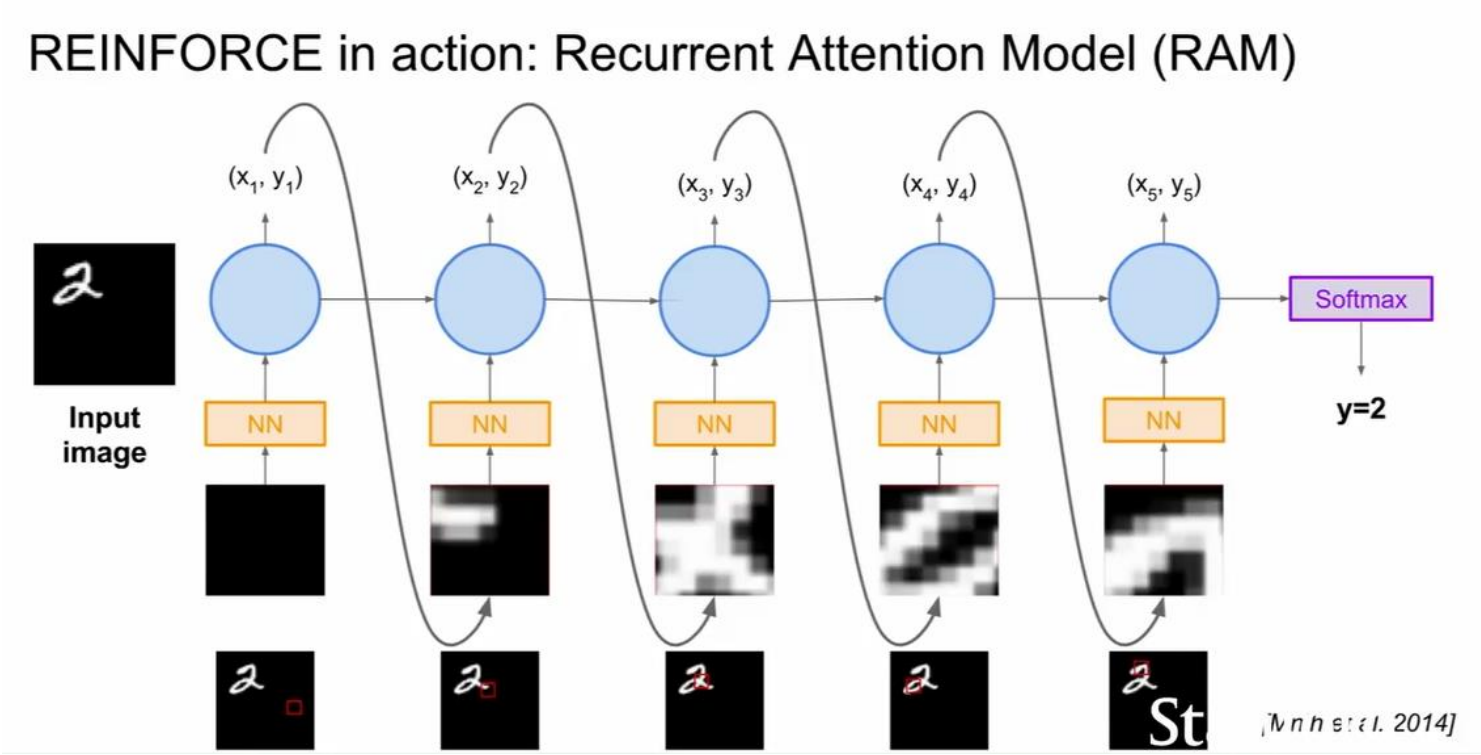
  $\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$
  $\theta \leftarrow \alpha\Delta\theta$
  $\phi \leftarrow \beta\Delta\phi$
End for



Fonte: https://www.youtube.com/watch?v=98V6PnwVXCc

7 Aplicação num cenário MNIST-like

- Há muitas aplicações relevantes, como as já destacadas na Seção 1, sendo que iremos destacar aqui apenas mais uma, representativa do potencial prático do aprendizado por reforço.



8 Vídeos de apoio

- Emergence of Locomotion Behaviours in Rich Environments:

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

  https://arxiv.org/pdf/1707.02286.pdf

- Técnicas fundamentadas em computação evolutiva também podem ser adotadas para locomoção:

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

  https://www.goatstream.com/research/papers/SA2013/SA2013.pdf

- Técnicas baseadas em gradiente descendente também podem ser adotadas para locomoção:

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

  http://www.daniel-holden.com/media/uploads/other_stuff/phasefunction.pdf

- AlphaGo Zero:

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

- 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

9 Projetos computacionais e textos de apoio

- https://analyticsindiamag.com/python-libraries-reinforcement-learning-dqn-rl-ai/
- https://awesomeopensource.com/projects/deep-reinforcement-learning
- https://github.com/araffin/rl-baselines-zoo
- https://www.amazon.com.br/Hands-Reinforcement-Learning-Python-reinforcement-ebook/dp/B079Q3WLM4
- https://www.amazon.com.br/Reinforcement-Learning-Hands-Maxim-Lapan/dp/1788834240/
- https://arxiv.org/pdf/1708.05866.pdf
- https://arxiv.org/pdf/1701.07274.pdf
- https://arxiv.org/pdf/1810.06339.pdf
- Terminando com o ponto de partida (livro-texto de referência):
  - ✓ https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
  - ✓ https://www.amazon.com.br/Reinforcement-Learning-Introduction-Richard-Sutton/dp/0262039249
  - ✓ https://www.amazon.com.br/Reinforcement-Learning-Introduction-Richard-Sutton/dp/0262193981/