

Lecture 09: Model free learning in RL

Radoslav Neychev

MADE, Moscow

01.06.2022

1. MDP properties recap
2. Value-function and Q-function
3. Reward discounting
4. Q-learning, temporal difference
5. Approximate Q-learning

Based on: https://github.com/yandexdataschool/Practical_RL/ weeks 2, 3 and 4

References

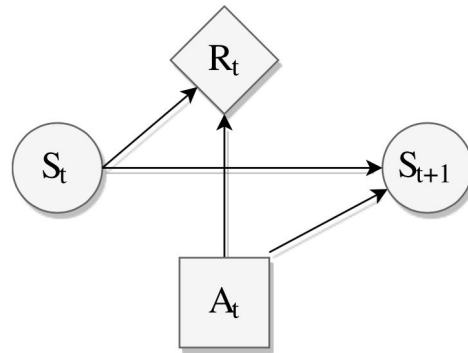
These slides are deeply based on [Practical RL course](#) weeks 2, 3 and 4
Special thanks to YSDA team for making them publicly available.

Given dynamics, how to find an optimal policy?

Definition of Markov Decision Process

MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where

- 1 \mathcal{S} – set of states of the world
- 2 \mathcal{A} – set of actions
- 3 $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \Delta(\mathcal{S})$ – state-transition function, giving us $p(s_{t+1} | s_t, a_t)$
- 4 $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ – reward function, giving us $\mathbb{E}_R [R(s_t, a_t) | s_t, a_t]$.



Markov property

$$p(r_t, s_{t+1} | s_0, a_0, r_0, \dots, s_t, a_t) = p(r_t, s_{t+1} | s_t, a_t)$$

(next state, expected reward) depend on (previous state, action)

Goal: solve an MDP by finding an optimal policy

1. What is the objective?
 - a. Reward: discounting and design
 - b. Expected objective: state- and action-value function

Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal

Cumulative reward is called a return:

$$G_t \triangleq R_t + R_{t+1} + R_{t+2} + \dots + R_T$$

Diagram illustrating the components of the return equation:

- A blue arrow points from the text "Cumulative reward" to the G_t term in the equation.
- A green arrow points from the text "immediate reward" to the R_t term in the equation.
- A red arrow points from the text "end of an episode" to the R_T term in the equation.

E.g.: reward in chess – value of taken opponent's piece

E.g.: data center non-stop cooling system

- **States** – temperature measurements
- **Actions** – different fans speed
- **R = 0** for exceeding temperature thresholds
- **R = +1** for each second system is cool

What could go wrong with such a design?

E.g.: data center non-stop cooling system

- States – temperature measurements
- Actions – different fans speed
- $R = 0$ for exceeding temperature thresholds
- $R = +1$ for each second system is cool

What could go wrong with such a design?

Infinite return for **non optimal** behaviour!

$$G_t = 1 + 1 + 0 + 1 + 1 + 0 + \dots = \sum_{t=1}^{\infty} R_t = \infty$$

E.g.: cleaning robot

- **States** – dust sensors, air
- **Actions** – cleaning / rest / conditioning on or off
- **$R = 100$** for long tedious floor cleaning task done
- **$R = 1$** for turning air conditioning on-off
- Episode **ends** each **day**

What could go wrong with such a design?



OpenAI blog post about faulty rewards: <https://openai.com/blog/faulty-reward-functions/>

E.g.: cleaning robot

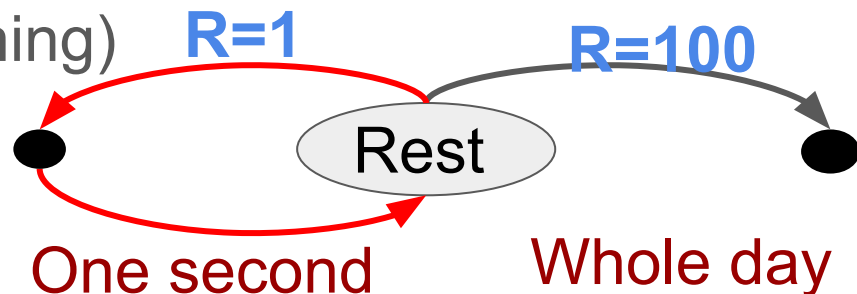
- **States** – dust sensors, air
- **Actions** – cleaning / rest / conditioning on or off
- **$R = 100$** for long tedious floor cleaning task done
- **$R = 1$** for turning air conditioning on-off
- Episode **ends** each **day**

What could go wrong with such a design?

Reward(air) < Reward(cleaning)

Time(air) << Time(cleaning)

Positive feedback loop!



Explaining goals to agent through reward

Reward hypothesis (R.Sutton)

Goals and purposes can be thought of as the maximization of the expected value of the cumulative sum of a received scalar signal

Reward discounting

Get rid of infinite sum by **discounting**

$$0 \leq \gamma < 1$$

$$G_t \triangleq R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

discount factor \longrightarrow

The same cake compared to
today's one worth

- γ times less tomorrow
- γ^2 times less the day after

tomorrow



γ will eat it day by day

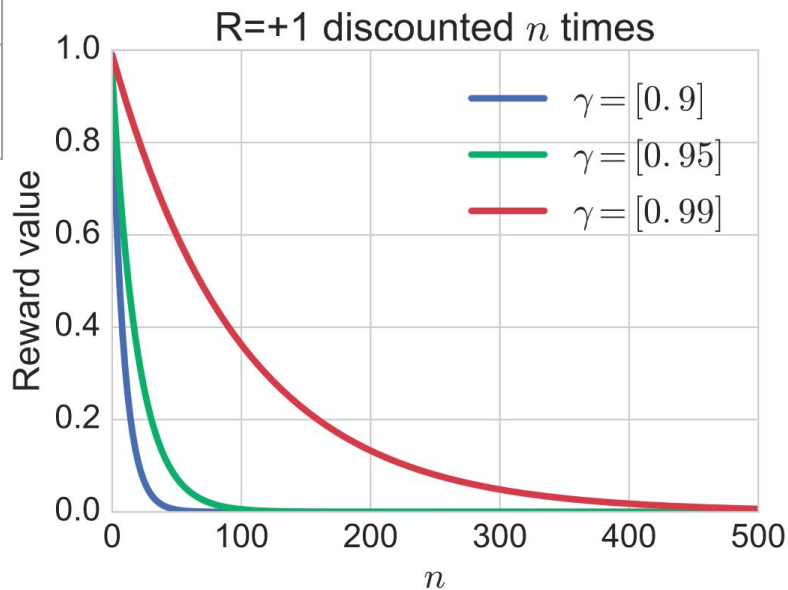
Reward discounting

Discounting makes sums finite

Maximal return for **R = +1**

$$G_0 = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

γ	0.9	0.95	0.99
$\frac{1}{1-\gamma}$	10	20	100



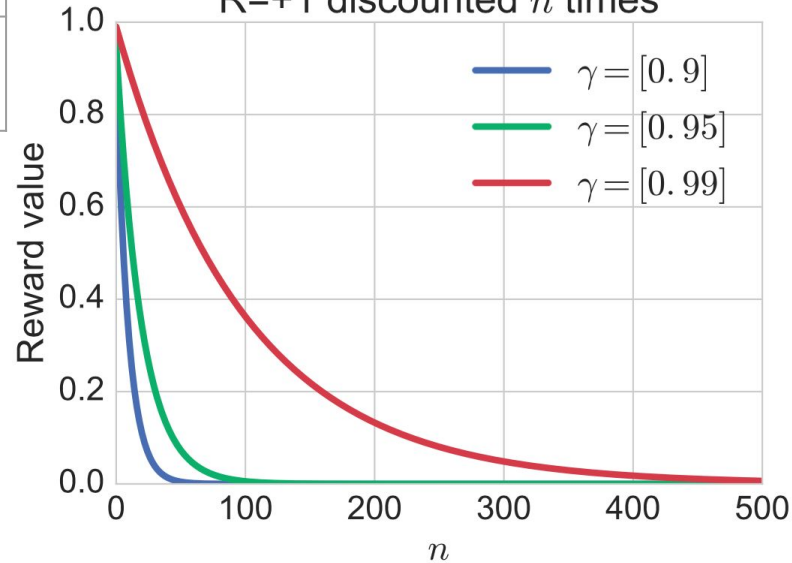
Discounting makes sums finite

Maximal return for **R = +1**

γ	0.9	0.95	0.99
$\frac{1}{1-\gamma}$	10	20	100

$$G_0 = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

R=+1 discounted n times



Any **discounting**
changes optimisation
task and its solution!

State- and **Action-**value functions

State-value function $v(s)$

$v(s)$ is expected **return** conditional on state:

$$\begin{aligned} v_{\pi}(s) &\triangleq \mathbb{E}_{\pi} [G_t \mid S_t = s] \\ &= \mathbb{E}_{\pi} [R_t + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a \mid s) \sum_{r, s'} p(r, s' \mid s, a) \left[r + \gamma \mathbb{E}_{\pi} [G_{t+1} \mid S_{t+1} = s'] \right] \\ &= \sum_a \pi(a \mid s) \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')] \end{aligned}$$

By definition

Intuition: value of following policy π from state s

Action-value function $q(s, a)$

Is expected **return** conditional on state and action:

Intuition: value of following policy π after committing action **a** in state **s**

$$\begin{aligned} q_{\pi}(s, a) &= \mathbb{E}_{\pi} [G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_{\pi} [R_t + \gamma G_{t+1} \mid S_t = s, A_t = a] \\ &= \sum_{r, s'} p(r, s' \mid s, a) \left[r + \gamma \mathbb{E}_{\pi} [G_{t+1} \mid S_{t+1} = s'] \right] \\ &= \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')] \end{aligned}$$

Relations between $v(s)$ and $q(s,a)$

We already know how to write $q(s,a)$ in terms of $v(s)$

$$q_{\pi}(s, a) = \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')]$$

What about $v(s)$ in terms of $q(s,a)$?

$$\begin{aligned} v_{\pi}(s) &= \sum_a \pi(a \mid s) \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')] \\ &= \sum_a \pi(a \mid s) q_{\pi}(s, a) \end{aligned}$$

So, we could now write $q(s, a)$ in terms of $q(s,a)$!

$$q_{\pi}(s, a) = \sum_{r, s'} p(r, s' \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_{\pi}(s', a') \right]$$

Bellman **expectation** equation for $\mathbf{v}(\mathbf{s})$

Recursive definition of $v(s)$ is an important concept in RL

$$\begin{aligned} v_{\pi}(s) &= \sum_a \pi(a \mid s) \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')] \\ &= \mathbb{E}_{\pi} [R_t + \gamma v_{\pi}(S_{t+1}) \mid S_t = s] \end{aligned}$$

Bellman **expectation** equation for **q(s,a)**

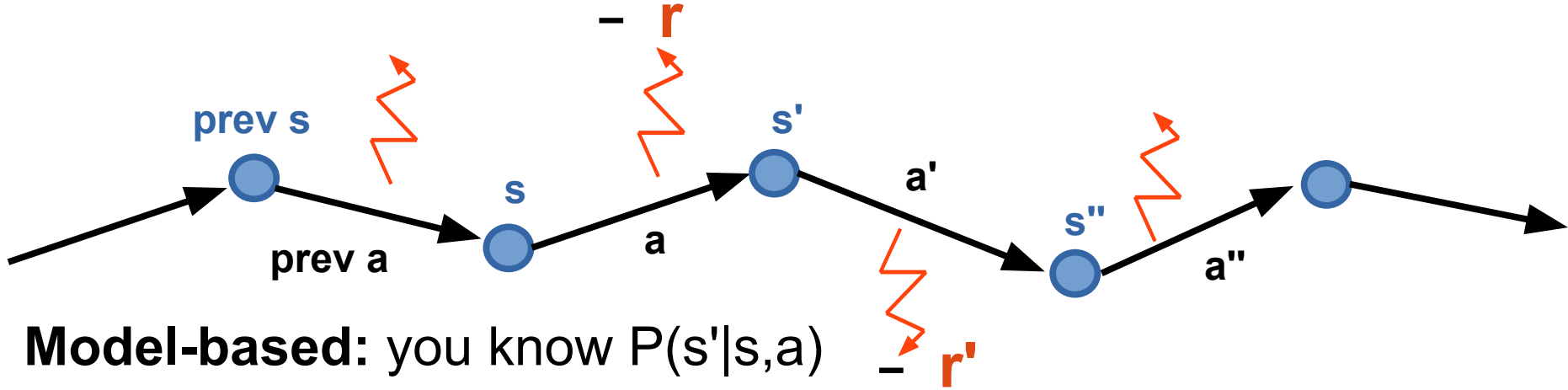
$$\begin{aligned} q_{\pi}(s, a) &= \sum_{r, s'} p(r, s' \mid s, a) [r + \gamma v_{\pi}(s')] \\ &= \sum_{r, s'} p(r, s' \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_{\pi}(s', a') \right] \end{aligned}$$

- $V_{\pi}(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π
- $V^*(\mathbf{s})$ – expected G from state \mathbf{s} if you follow π^* – optimal
- $Q_{\pi}(\mathbf{s}, \mathbf{a})$ – expected G from state \mathbf{s}
 - if you start by taking action \mathbf{a}
 - and follow π from next state on
- $Q^*(\mathbf{s}, \mathbf{a})$ – same as $Q_{\pi}(\mathbf{s}, \mathbf{a})$ where $\pi = \pi^*$ – optimal policy

$$Q^*(s, a) = E_{s', r} [r(s, a) + \gamma \cdot V^*(s')]$$

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

Learning from trajectories



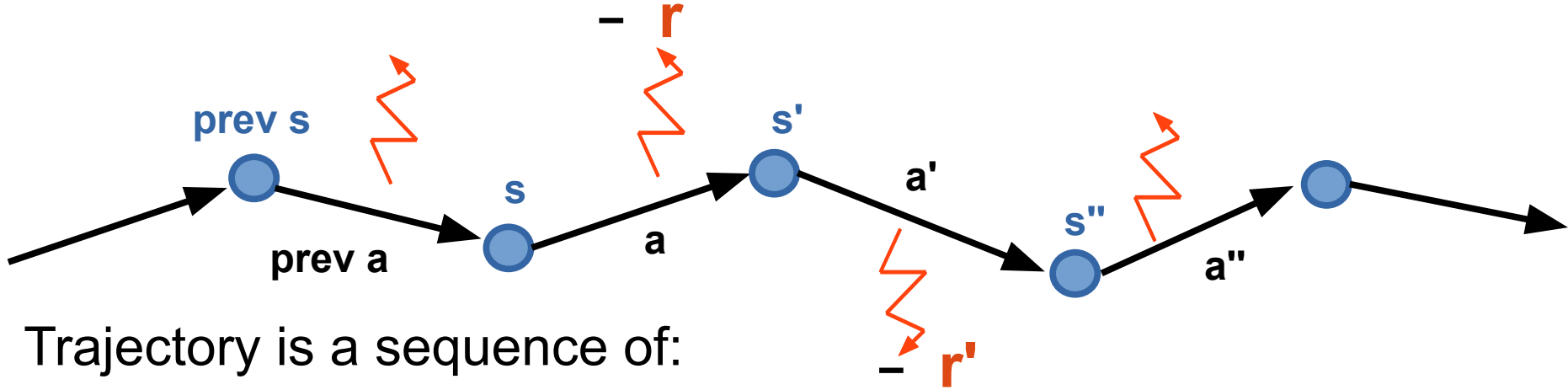
Model-based: you know $P(s'|s,a)$

- can apply dynamic programming
- can plan ahead

Model-free: you can sample trajectories

- can try stuff out
- insurance not included

Learning from trajectories



Trajectory is a sequence of:

- states (s)
- actions (a)
- rewards (r)

Q: What to learn?

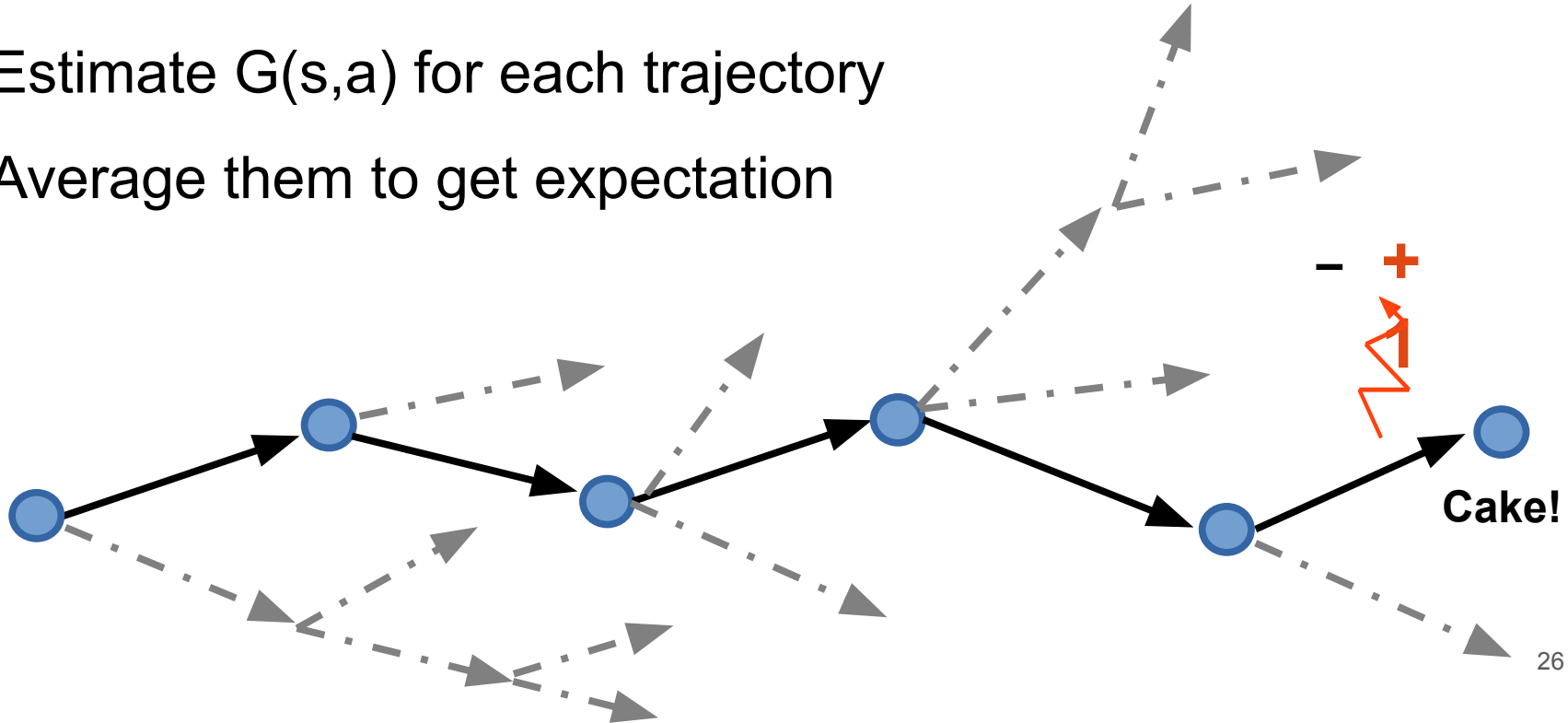
$V(s)$ or $Q(s,a)$

We can only sample trajectories

$V(s)$ is useless
without $P(s'|s,a)$

Idea 1: Monte-carlo

- Get all trajectories containing particular (s,a)
- Estimate $G(s,a)$ for each trajectory
- Average them to get expectation



Idea 2: Temporal difference

- $Q(s, a)$ can be improved iteratively!

$$Q(s_t, a_t) \leftarrow E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

That's $Q^*(s, a)$

That's value for π^*
aka optimal policy

That's something
we don't have

What do we do?

Idea 2: Temporal difference

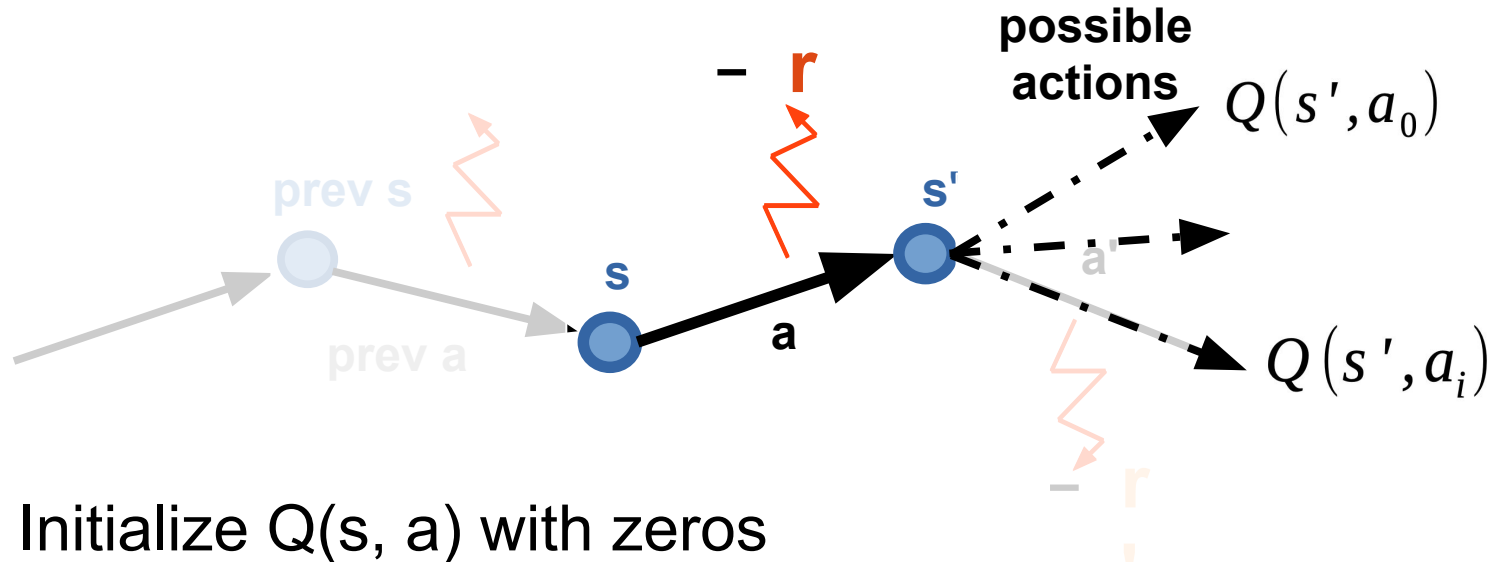
- $Q(s, a)$ can be improved iteratively!

$$Q(s_t, a_t) \leftarrow E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

$$E_{r_t, s_{t+1}} r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \approx \frac{1}{N} \sum_i r_i + \gamma \cdot \max_{a'} Q(s_i^{\text{next}}, a')$$

$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

Q-learning



Initialize $Q(s, a)$ with zeros

- Sample $\langle s, a, r, s' \rangle$ from the environment
- Compute new $Q(s, a)$ estimation:

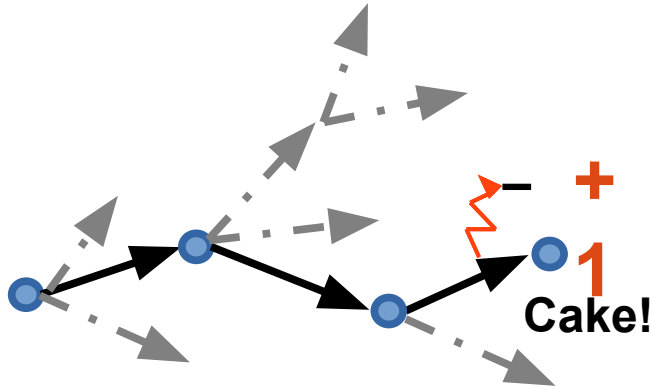
$$\hat{Q}(s, a) = r(s, a) + \gamma \max_{a_i} Q(s', a_i)$$

- Update $Q(s, a)$:

$$Q(s, a) \leftarrow \alpha \cdot \hat{Q}(s, a) + (1 - \alpha) Q(s, a)$$

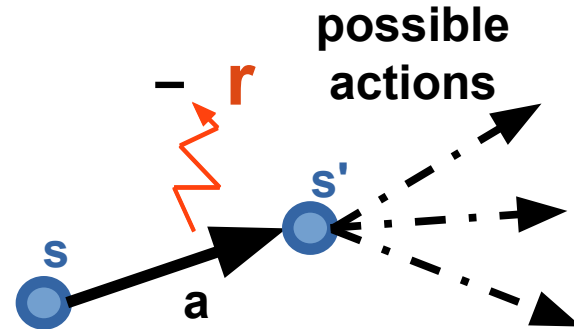
Monte-carlo

Averages Q over
sampled paths



Temporal Difference

Uses recurrent
formula for Q



$$Q^*(s, a) = E_{s', r} [r(s, a) + \gamma \cdot V^*(s')]$$

$$Q(s_t, a_t) \leftarrow \alpha \cdot (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')) + (1 - \alpha) Q(s_t, a_t)$$

$$\pi(s) : \operatorname{argmax}_a Q(s, a)$$

Exploration-exploitation tradeoff

Strategies:

- ϵ -greedy
 - With probability ϵ take random action; otherwise take optimal action.
- Softmax
 - Pick action proportional to softmax of shifted normalized Q-values.

$$\pi(a|s) = \text{softmax}\left(\frac{Q(s,a)}{\tau}\right)$$

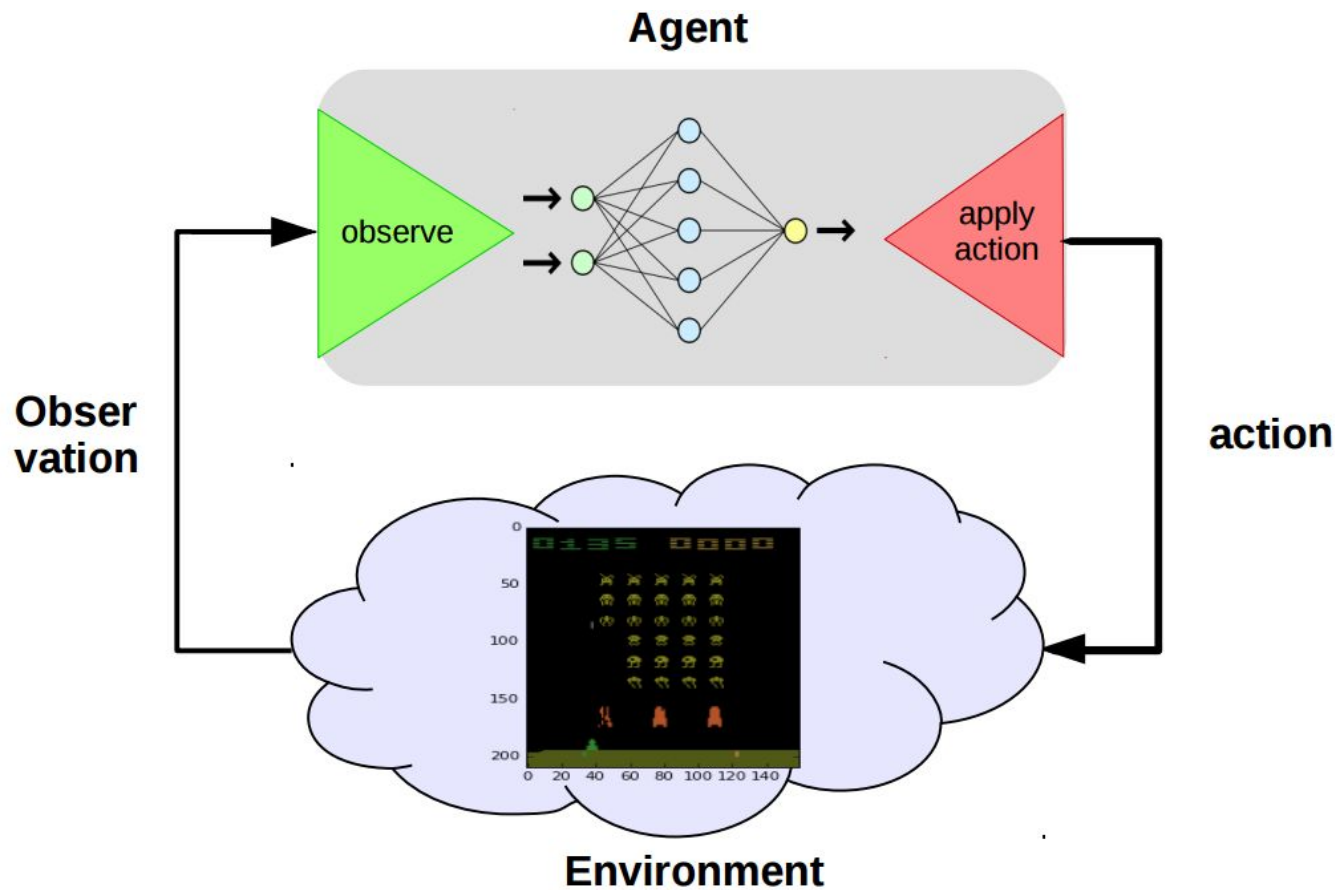
Last step: making it continuous

For now states and actions are discrete. What if states are **not**?

Minimize loss given $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$

$$L = [Q(s_t, a_t) - Q^{true}(s_t, a_t)]^2$$

$$L \approx [Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$



$$G_t = \sum_{t'=t}^T \gamma^{(t'-t)} r_{t'}$$

$$Q^\pi(s, a) = E_\pi[G_t | s_t = s, a_t = a]$$

$$V^\pi(s) = E_\pi[G_t | s_t = s]$$

Recurrent relations

$$Q^\pi(s, a) = E_{s_{t+1}}[r_t + \gamma V^\pi(s_{t+1})]$$

$$Q^\pi(s, a) = E_{s_{t+1}, a_{t+1} \sim \pi}[r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

For all π, s, a : $Q^{\pi^*}(s, a) \geq Q^{\pi}(s, a)$

$$\pi^*(s) = \operatorname{argmax}_a Q^{\pi^*}(s, a)$$

Bellman optimality equation

$$Q^*(s_t, a) = E_{s_{t+1}}[r_t + \max_{a'} Q^*(s_{t+1}, a')]$$

Training step

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

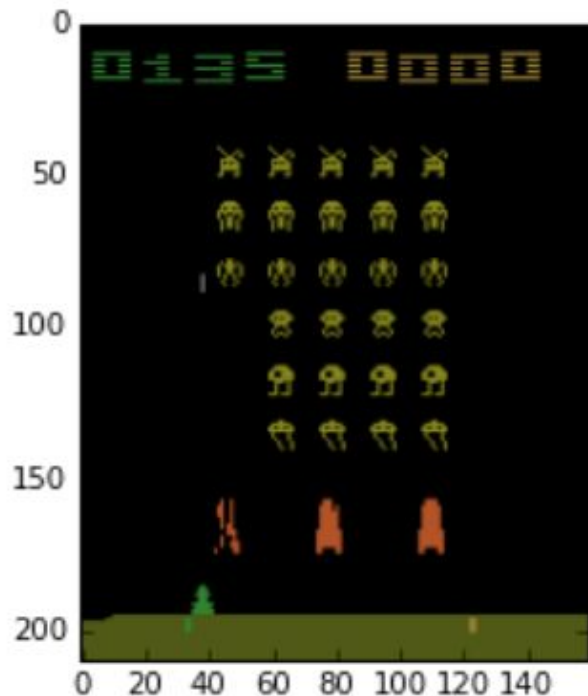
Q-learning as MSE minimization

$$L = (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))^2$$

Const

$$\nabla L = 2 \cdot (r_t + \gamma \max_{a'} \boxed{Q(s_{t+1}, a')} - Q(s_t, a_t))$$

What's wrong here?



How many states are there?
approximately

$$|S| = 2^{210 \cdot 160 \cdot 8 \cdot 3}$$

Q-learning: make it continuous

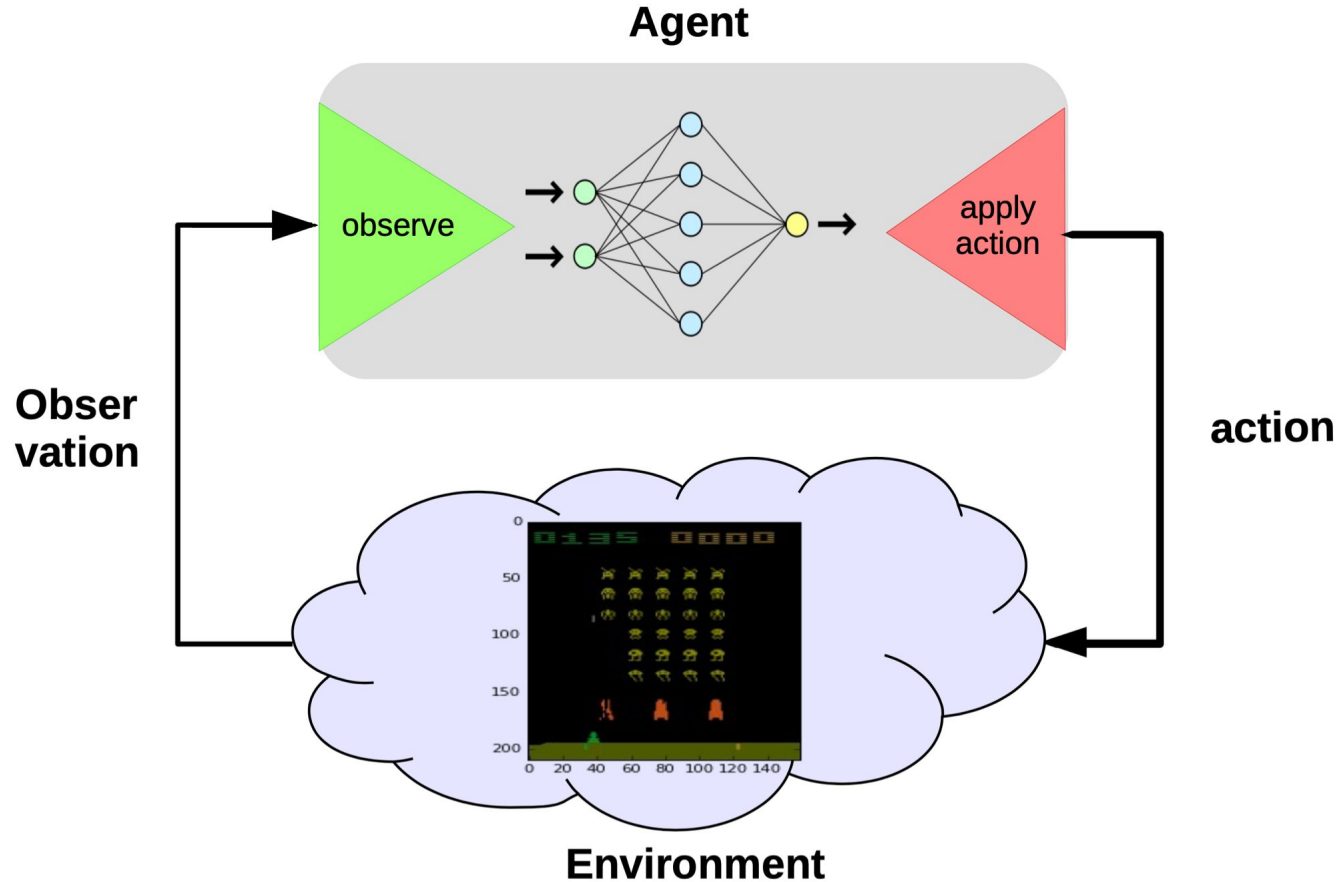
For now states and actions are discrete. What if states are **not**?

Minimize loss given $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s}' \rangle$

$$L = [Q(s_t, a_t) - Q^{true}(s_t, a_t)]^2$$

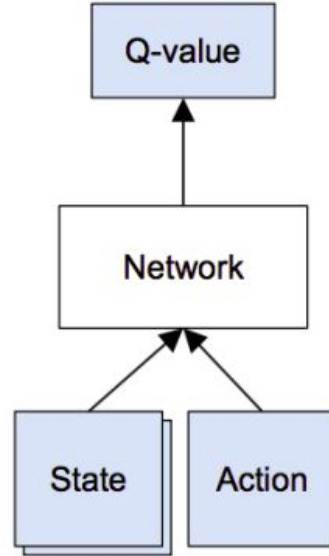
$$L \approx [Q(s_t, a_t) - (r_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a'))]^2$$

Approximate Q-learning



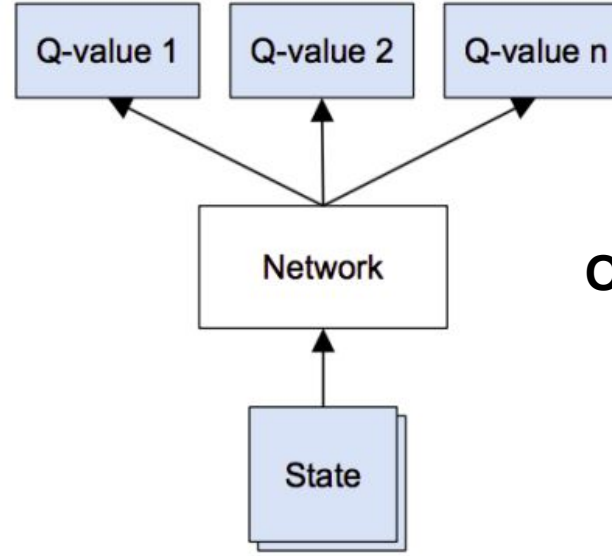
Possible architectures

**Continuous
control or large
number of
actions**



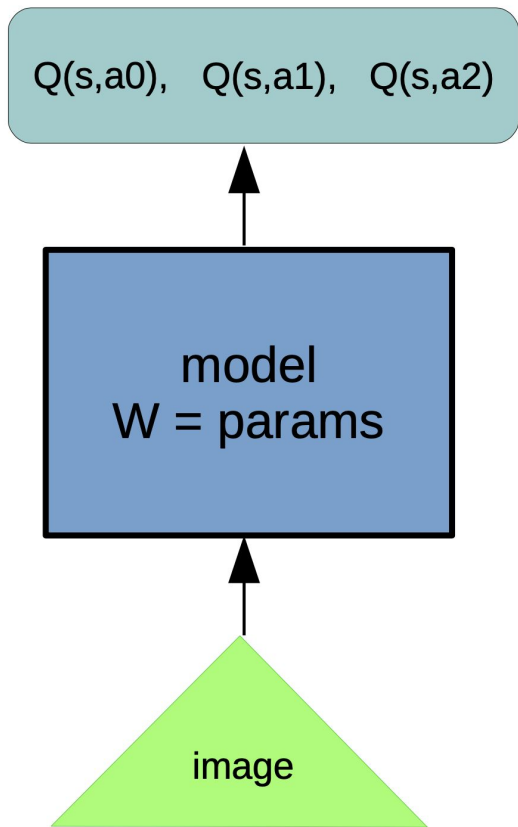
Given **(s,a)**
Predict $Q(s,a)$

**One pass for all
actions**



Given **s** predict all q-values
 $Q(s,a_0)$, $Q(s,a_1)$, $Q(s,a_2)$

Approximate Q-learning



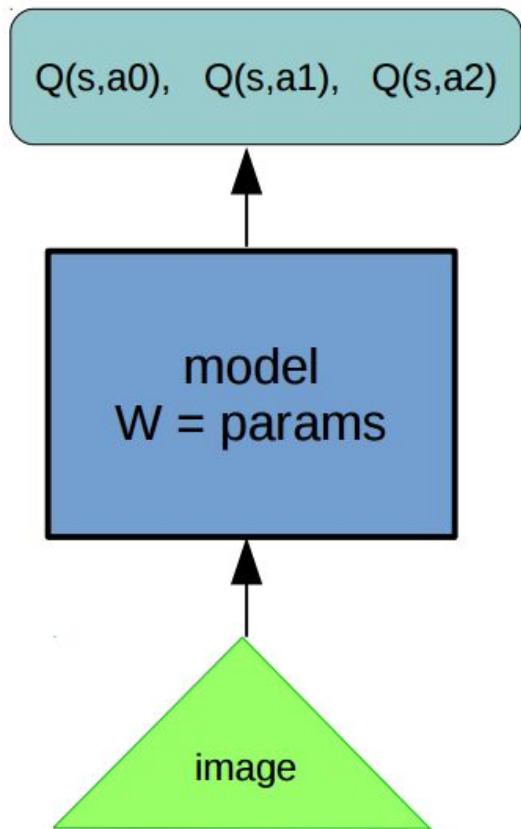
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} \hat{Q}(s_{t+1}, a')$$

$$L = \left(Q(s_t, a_t) - \left[r + \gamma \cdot \max_{a'} Q(s_{t+1}, a') \right] \right)^2$$

Consider const

$$w_{t+1} = w_t - \alpha \cdot \frac{\delta L}{\delta w}$$

Approximate Q-learning



Objective:

$$L = \left(Q(s_t, a_t) - \underbrace{\hat{Q}(s_t, a_t)}_{\text{consider const}} \right)^2$$

Q-learning:

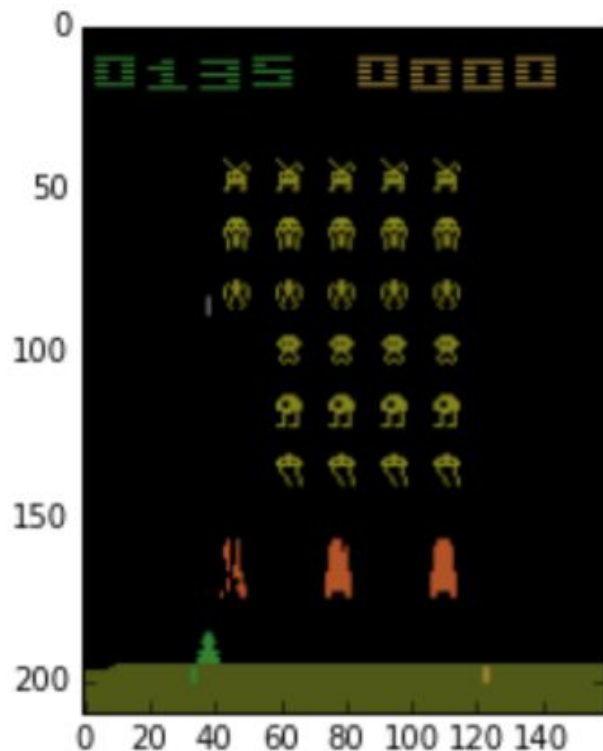
$$\hat{Q}(s_t, a_t) = r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot Q(s_{t+1}, a_{t+1})$$

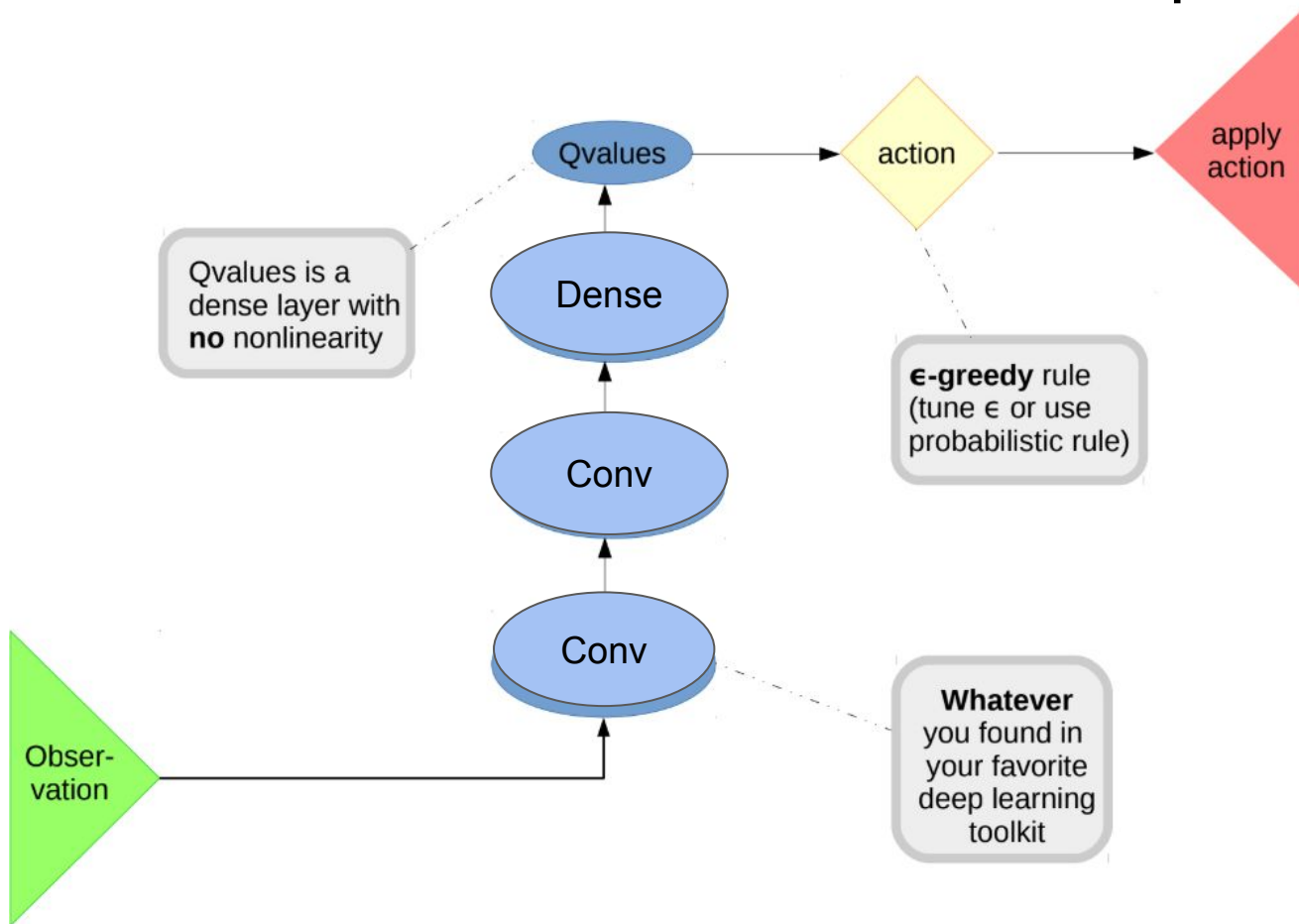
Expected Value SARSA:

$$\hat{Q}(s_t, a_t) = r + \gamma \cdot E_{a' \sim \pi(a|s)} Q(s_{t+1}, a')$$



What kind of network digests images well?

Basic deep Q-learning





- Q-learning allows to learn some approximation of the reward function and environment model
 - So we can use it to solve the desired problem
- Remember what $Q(s, a)$ and $V(s)$ functions do
- Remember both about exploration and exploitation
 - At least using greedy policy or softmax smoothing