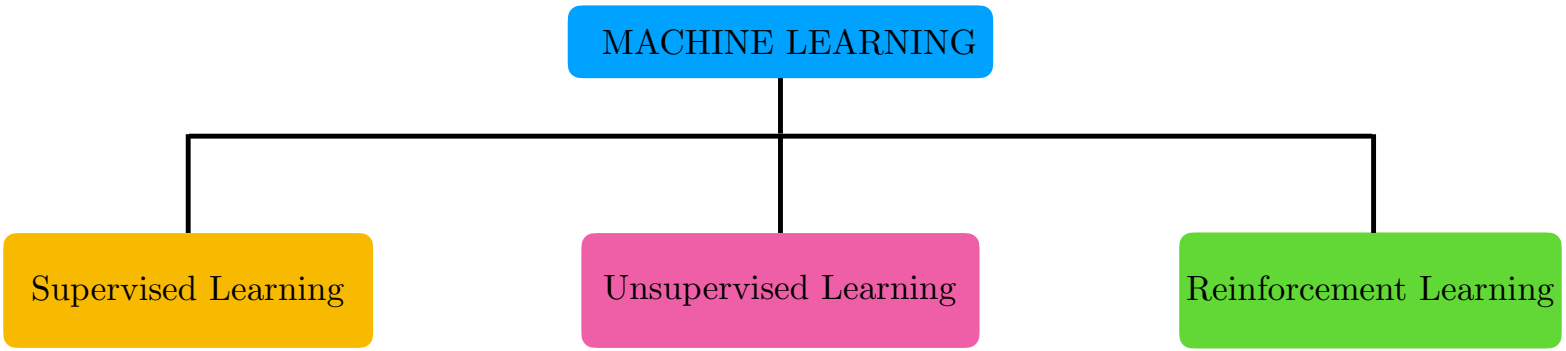
11-695: AI Engineering
DeepRL Introduction

Spring 2019

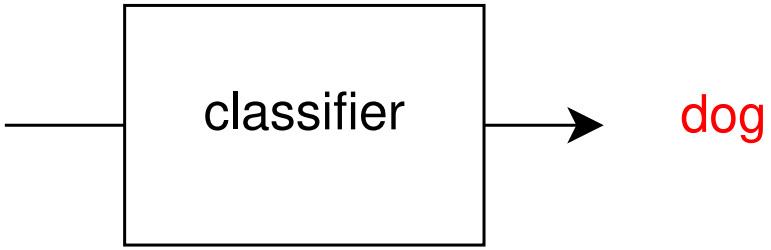
Topics in Machine Learning

Carnegie Mellon



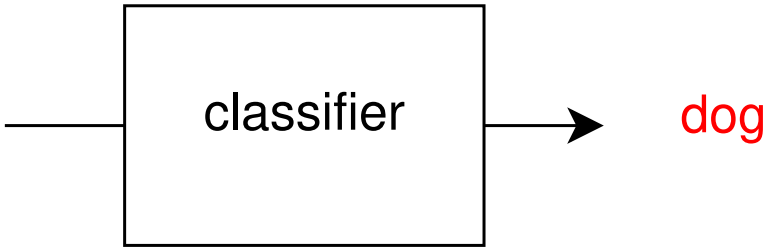
Outline

Supervised Learning



- Given a data point x predict label y
- Training: Using data pairs $\{x^{(i)}, y^{(i)}\}_1^N$
- Target of a ML problem is an optimization one
  - Target: to find a function $\mathbf{f} : \mathcal{X} \rightarrow \mathcal{Y}$
  - Loss: $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{L}(\mathbf{y}, \mathbf{f}(\mathbf{x}, \theta))$ with parameter θ
  - Optimization: make the error loss function as low as possible

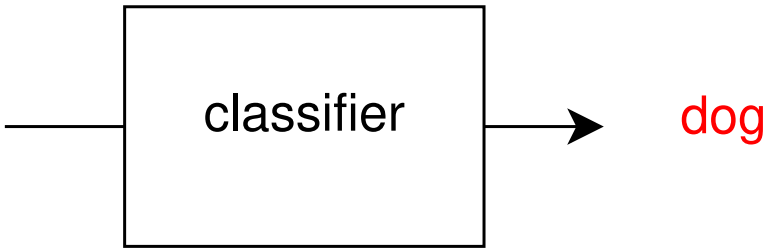
Supervised Learning: Rephrase



- Goal: update parameter θ so that:
  - the loss is lowest possible,
  - or the testing accuracy is highest possible (related to each other).
- The trainer always does one job: use a proper method, say SGD, to tune θ
- The trainer operates in a static environment (Env), nothing else can affect the training (except for some extreme cases such as machine crashes or earthquakes)

Supervised Learning: Make it more *real*

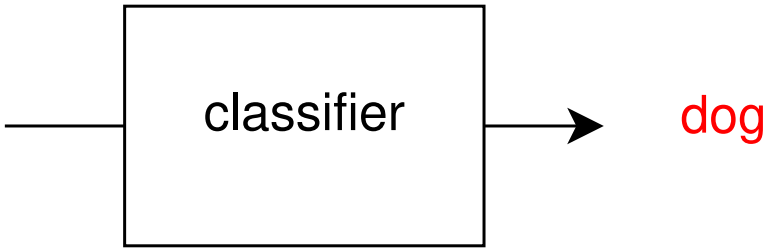
Carnegie Mellon



- Let's call the trainer an agent for now
  - Only 1 action: update θ based on SGD
  - Always make this action until a stopping condition.
- The Env: the agent can poll the following information:
  - current loss value
  - current val/test accuracy
  - ...

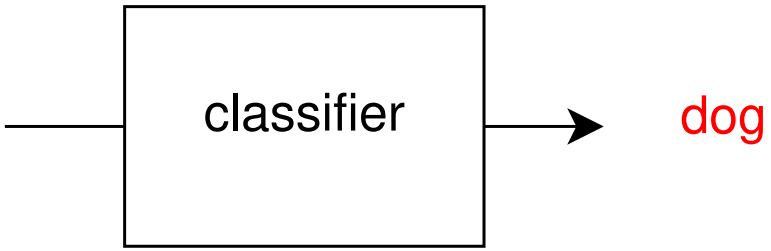
Supervised Learning: Make it more *real*

Carnegie Mellon



- Right now, regardless how bad the training progress is, the agent still takes that only action! *E.g.*
  - It faces some outliers and they make the training digressed/diverged → maybe the agent needs not update this
  - It faces saddle points or slow progress → Maybe it needs some augmentation in training (learning rate, regularization, ...)
  - Other possible bad things

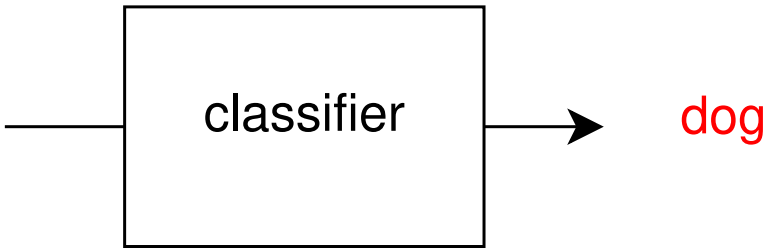
Supervised Learning: Make it more *real*



- To make it more real, let's define something new: a *reward* that is how well you are doing your job towards optimization
  - Either how much loss you reduce for training,
  - Or how much val/test accuracy you have improved
  - Others, freely designed to serve for a good training process
- Along with *reward*, Env also returns the current *state* (e.g. loss, accuracies)

Supervised Learning: Make it more *real*

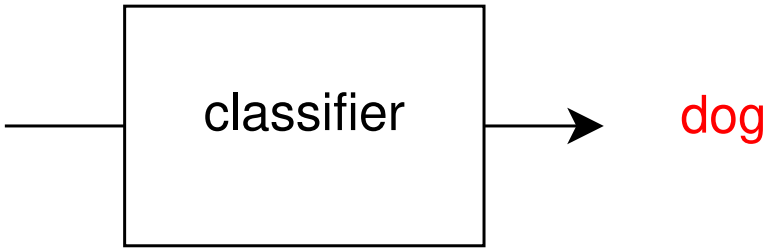
Carnegie Mellon



- We can even make the reward more real:
  - Take such quantities for each batch,
  - How good/bad for the current batch might be temporary, the final (accumulated) rewards for all batches matter the most

Supervised Learning: Make it more *real*

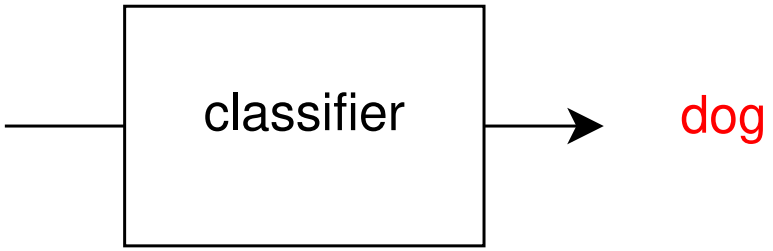
Carnegie Mellon



- Now, the Env is changing and telling the agent as it optimizes: rewards & states.
- The agent has guidance for it to timely fix the bad progress,
- The way it decides action to take based on based on a state is called a *policy* function.

Supervised Learning: Make it more *real*

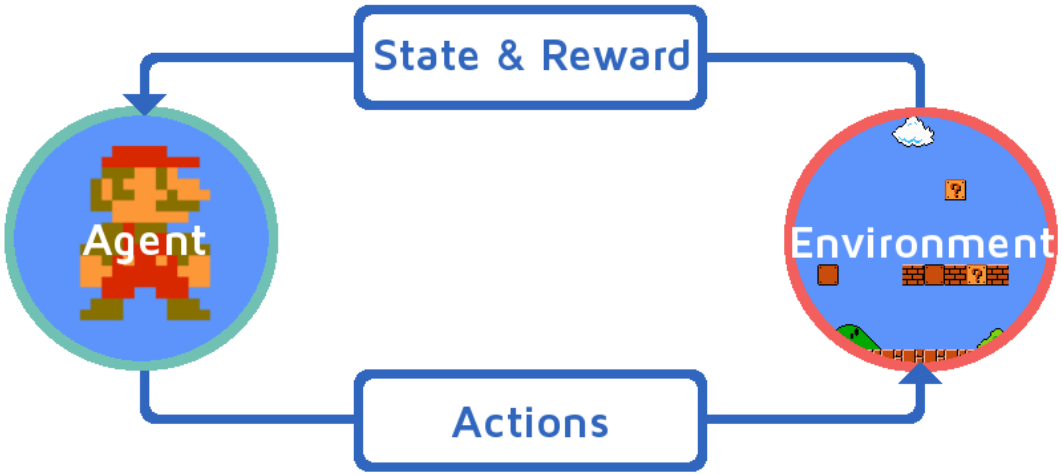
Carnegie Mellon



- We now have a more real example:
  - Agent optimizes towards its objective
  - Agent can change its action: not always blindly optimizes every single batch, using the same way
  - The Env returns the state to tell the agent how good/bad it is doing

Outline

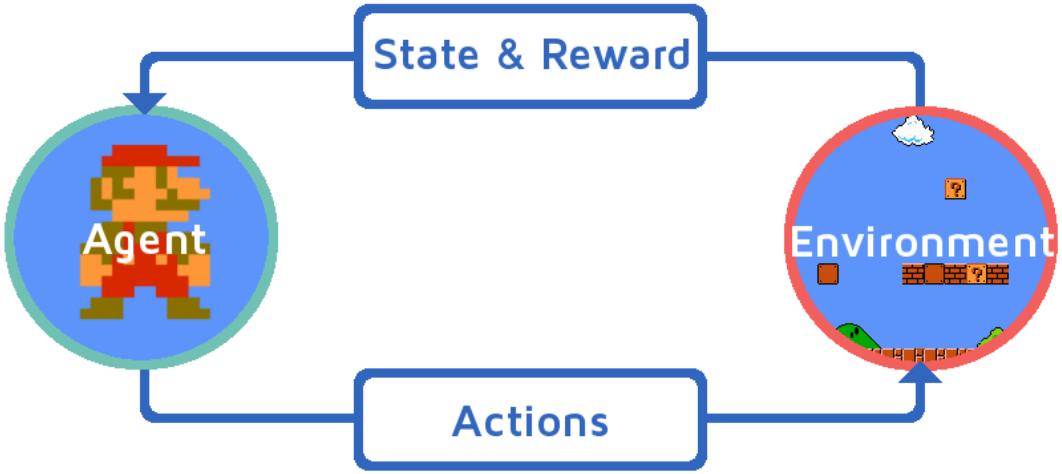
That is RL 1-0-1



- We now have a more real example:
  - Agent optimizes towards its objective
  - Agent can change its action: not always blindly optimizes every single batch, using the same way
  - The Env returns the state to tell the agent how good/bad it is doing
  - The agent takes such guiding information to improve itself
  - It learns a policy function to figure out action to take based on its observation

Image credit: medium.freecodecamp.org

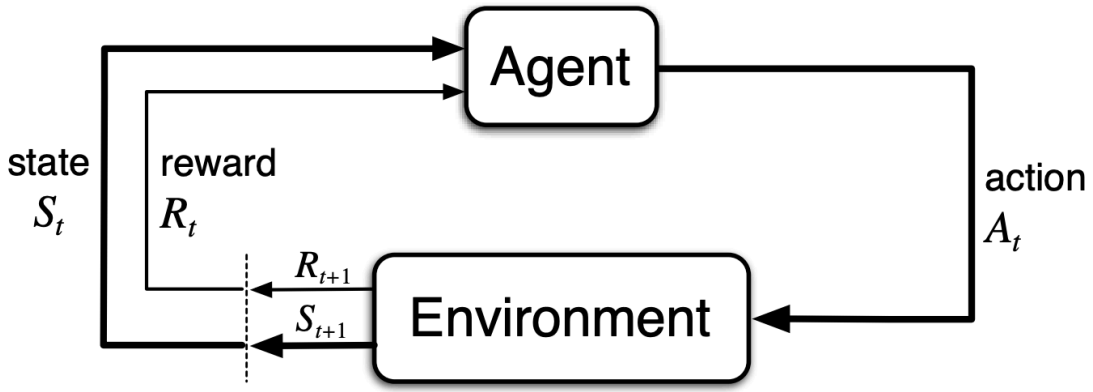
RL vs. Optimal Control



- Both are similar if not the same, but having different perspective:
  - Control focuses on improving *action* in a *well-defined* model → model-based
  - RL often makes *model-free* prediction of actions
- Some prominent commons:
  - Agent uses history to improve itself
  - Agent makes sequential actions, errors compounded

Image credit: medium.freecodecamp.org

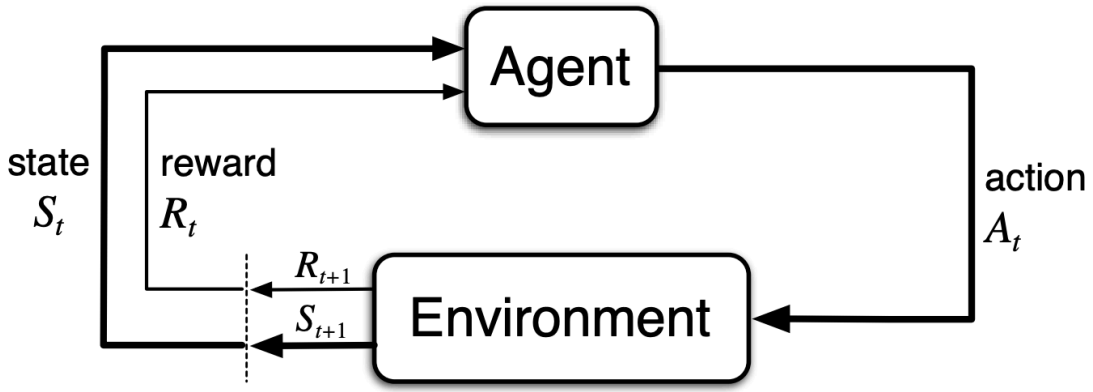
Markov Assumption



- An important assumption: Markov property in that Env. returns a new state based on the most current actions, ignoring previous ones.
- The classical model used to solved this problem is called MDP (Markov Decision Process)
- Assumption: Env is fully observable,

Image credit: Sutton & Barto: RL: An Introduction 2017

Markov Assumption: Unrealistic?



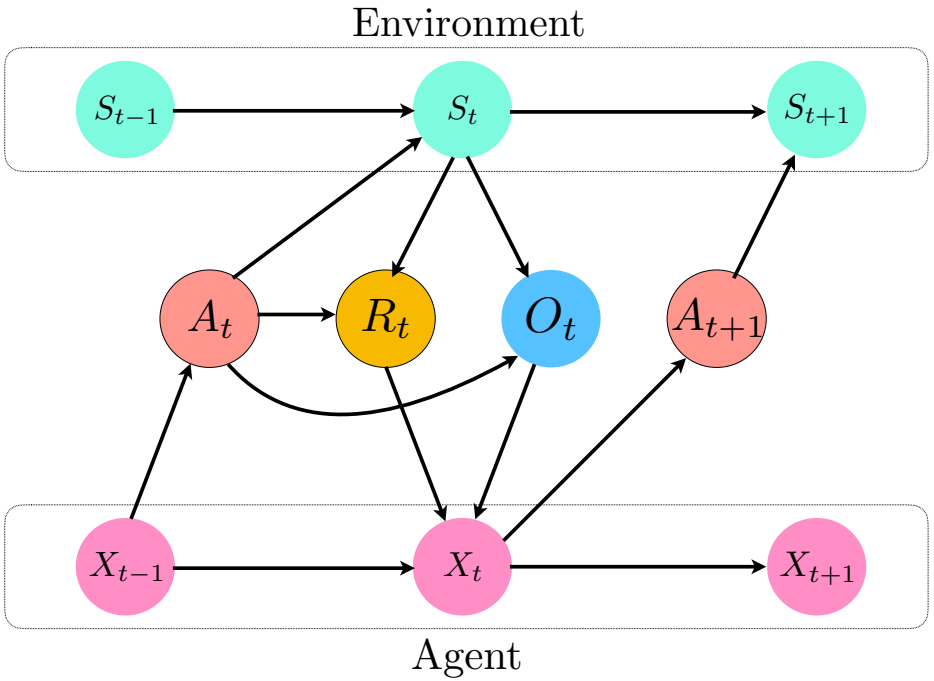
- Markov assumption is unrealistic, but it's needed for a strong theories (at least by now)
- But in real cases, for example a robot navigating a maze, it can only see a partial state
- This case, we have a model called POMDP (Partially-Observed MDP), and Markov assumption also does not hold.
- There are also other methods such as first-order type we will not mention.

Image credit: Sutton & Barto: RL: An Introduction 2017

Outline

POMDP General Model

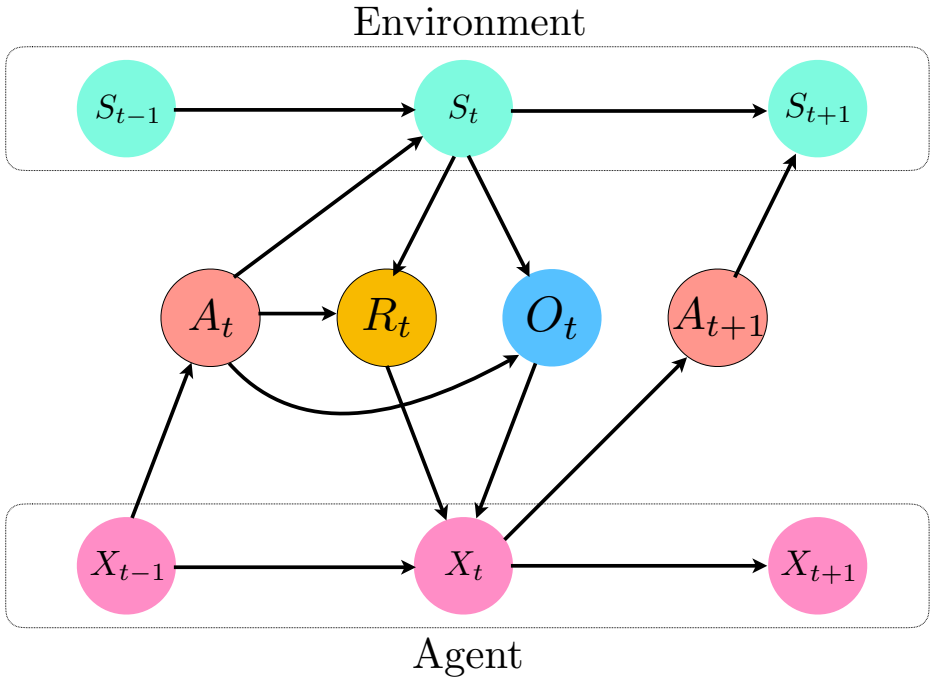
Carnegie Mellon



- $\mathcal{S}$: set of *states*
- $\mathcal{A}$: set of *actions*
- $\mathcal{R}$: set of *rewards*
- $\mathcal{O}$: set of *observations*
- $\mathcal{X}$: set of internal agent's *states*

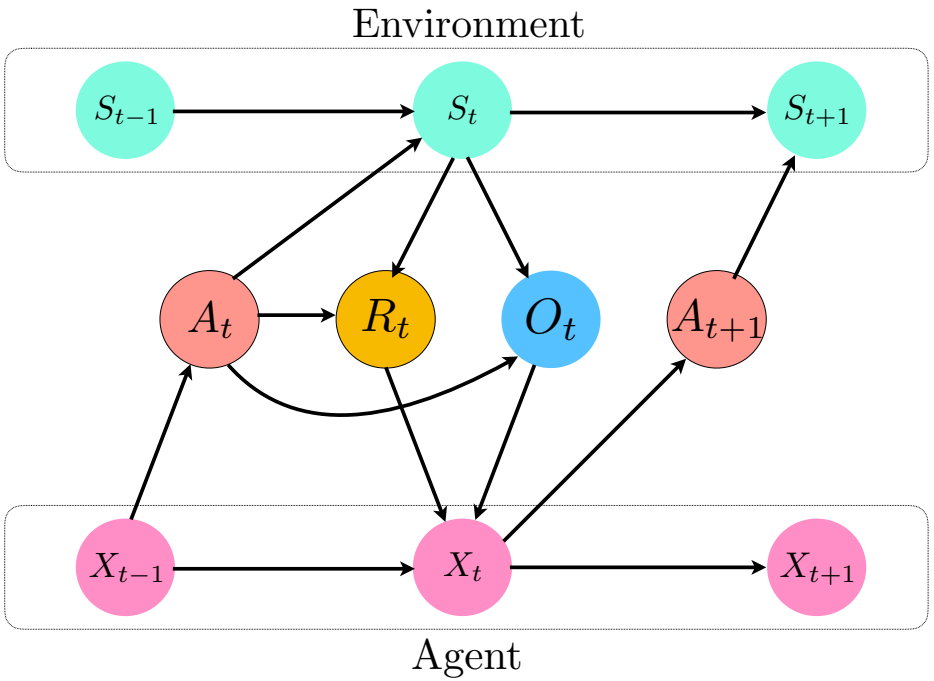
POMDP General Model

Carnegie Mellon



- State transition function: $\mathbf{f}_T : S \times A \to \mathcal{S}$, modeling $p(s'|a, s)$
- Reward function: $\mathbf{r} : S \times A \to \mathcal{R}$
- Observation function: $\mathbf{f}_O : S \times A \to \mathcal{O}$

MDP



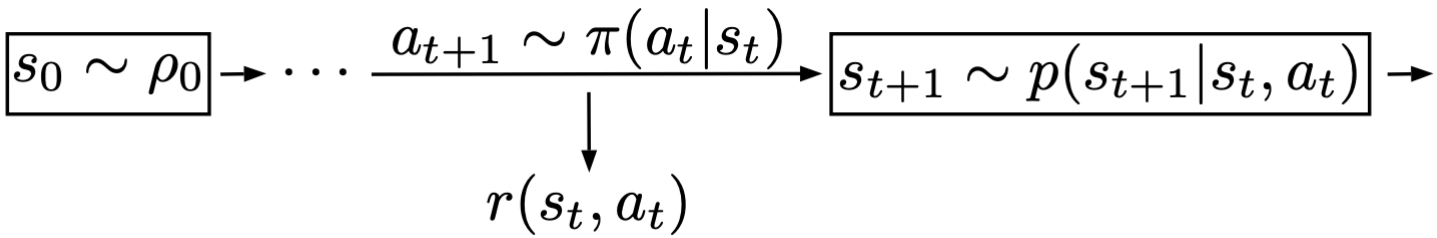
- Agents will get R_t, O_t from Env
- For MDP: $O_t = S_t$: Env is fully observable,
- and uses its policy function $\pi : \mathcal{S} \to \mathcal{A}$ modeling as $\pi(a|s)$
- *Note*: for stochastic policy: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$

Trajectory

$$\boxed{s_0 \sim \rho_0} \rightarrow \cdots \xrightarrow{a_{t+1} \sim \pi(a_t|s_t)} \boxed{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} \rightarrow$$

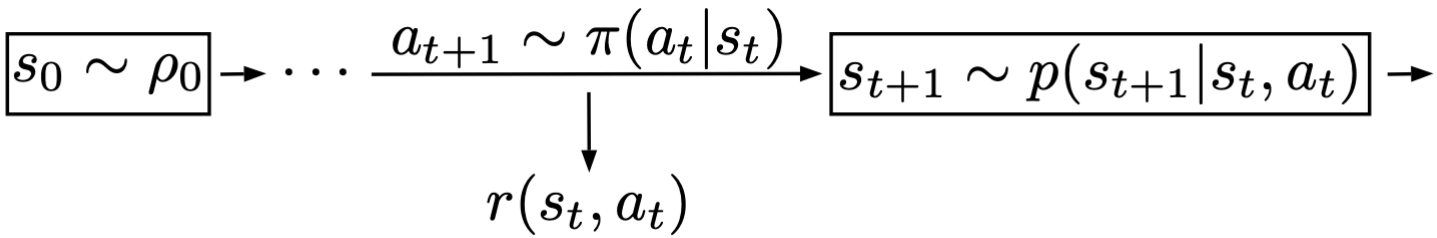
- Trajectory: $\tau = (s_0, a_0, s_1, a_1, ...) \sim \pi$ with initial state $s_0 \sim \rho_0$
- Probability of a trajectory: $p_\pi(\tau) = p(s_0) \prod_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$
- Loglikelihood
  $log\, p_\pi(\tau) = log\, p(s_0) + \sum_{t=0}^{\infty} (log\, \pi(a_t|s_t) + log\, p(s_{t+1}|s_t, a_t)))$
- Although in theory going to ∞, empirical trajectory usually ends finitely, *e.g.* when an episode ends in a game.

Reward



- Current (instant) reward: $r(s, a)$
- Discount factor: $\gamma \in [0, 1]$
- Discounted reward at time step T: $R_T(\tau) = \sum_{t=0}^{\infty} \gamma^t \, r(s_{T+t}, a_{T+t})$
- Why discounted?

Reward



- Current (instant) reward: $r(s, a)$
- Discount factor: $\gamma \in [0, 1]$
- Discounted reward at time step T: $R_T(\tau) = \sum_{t=0}^{\infty} \gamma^t \, r(s_{T+t}, a_{T+t})$
- Why discounted? Borrowed from classical economics: time value of money.

RL Optimization Problem

RL Objective

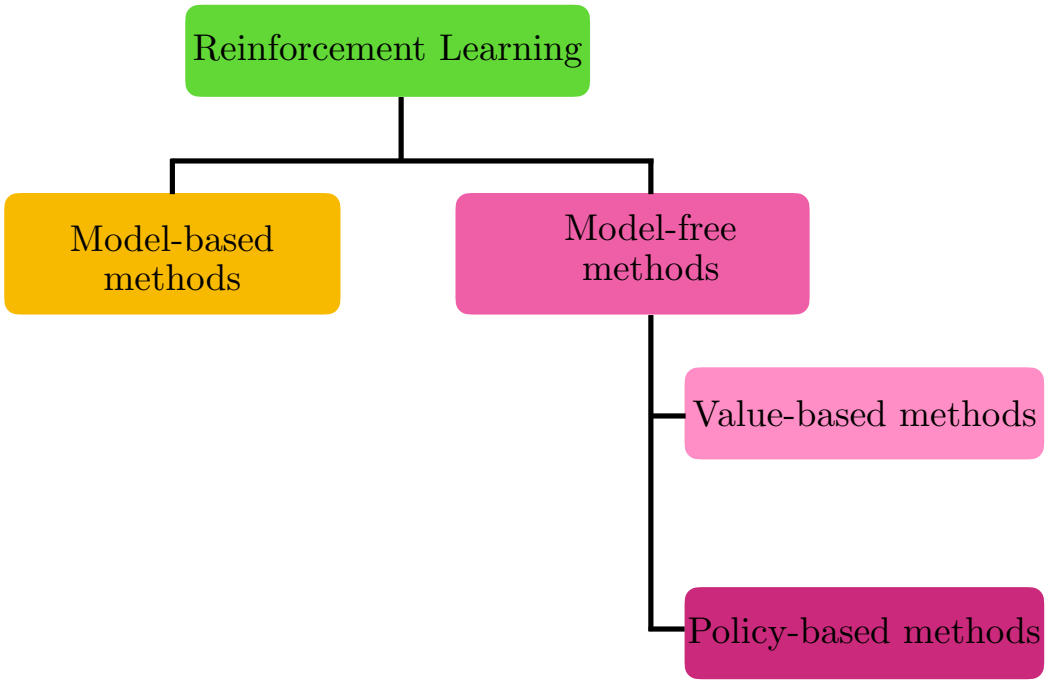
Find an optimal policy π^* that maximizes the expected discounted reward of all trajectories

$$\pi^* = \underset{\pi}{\text{argmax}}\, \mathbb{E}_{\tau \sim \pi}[R_0(\tau)] \tag{1}$$

where

$$R_0(\tau) = \sum_{t=0}^{\infty} \gamma^t\, r(s_t, a_t) \tag{2}$$

RL Big Picture



Model-based

- Model-based: estimating from observations
  - state transition function $T(s, a)$
  - reward function $r(s)$
- Then use MDP to solve
- Examples: Policy iteration, Value iteration
- Inefficient if the state space is big

Value-based (Model-free)

- Value: $V_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}[V_\pi(s')]]$
- Q-value: $Q_\pi(s,a) = r(s,a) + \gamma\mathbb{E}_{s'\sim p(s'|s,a)}[V_\pi(s')]$
- Temporal Difference (TD, stochastic policy iteration), SARSA: on-policy
- Q-learning: off-policy, approximate optimal action-value function $Q*$ independent of the policy being followed

Q-learning

- Q-function: $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- Q-value:

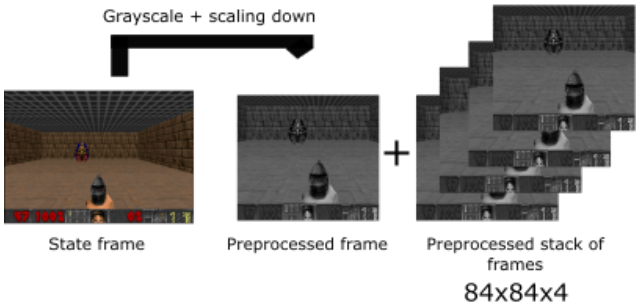
$$Q_\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)}[V_\pi(s')] \quad (3)$$

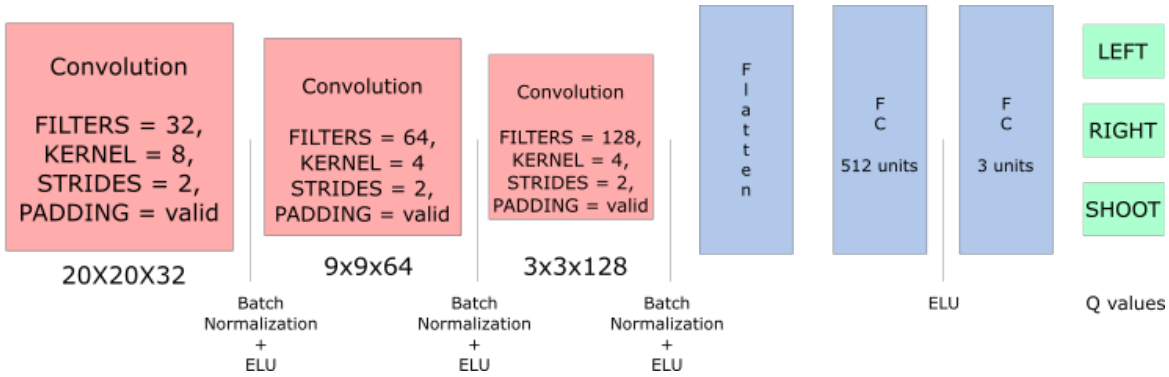
$$= \mathbb{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots | s_t, a_t] \quad (4)$$

- Q-table has a size of $|\mathcal{S}| \times |\mathcal{A}| \rightarrow$ from Q-value & state, can choose action
- Update Q-value based on Bellman equation:

$$Q_\pi(s,a) := Q_\pi(s,a) + \alpha[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q'_\pi(s',a') - Q_\pi(s,a)] \quad (5)$$

Deep Q-learning



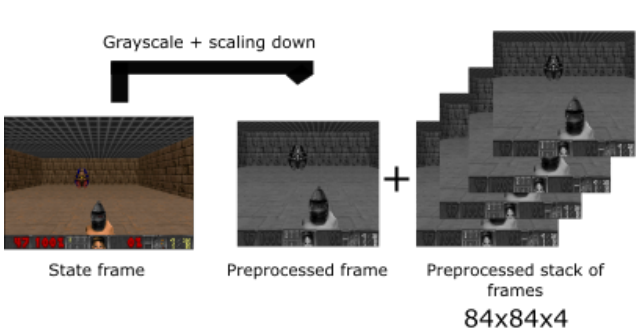


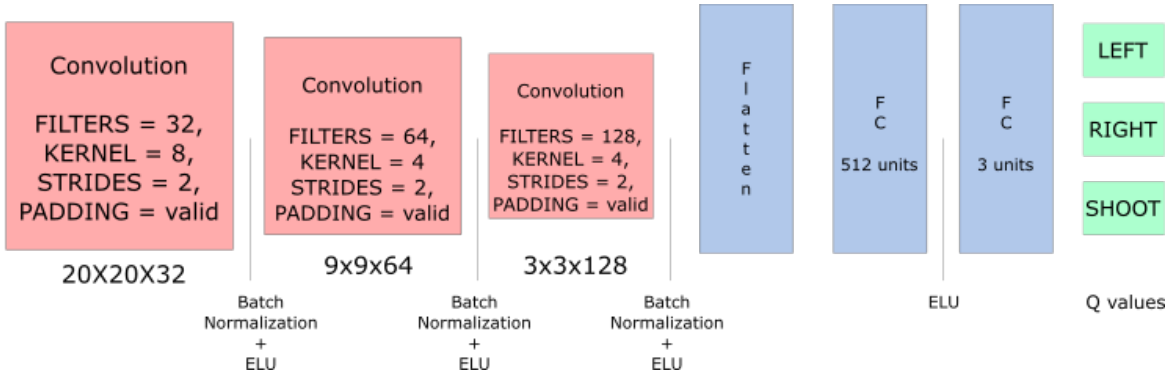
- Deep Q-learning = Deep NN + Q-learning
- NN extract features from state (*e.g.* images) and model Q-value for all actions in the action space.
- Tricks (inspired from Double Q-learning from Deepmind):
  - Stack frames together (for temporal information)
  - Have a replay buffer storing experience (diversify the learning process, avoid overfitting to immediate results)
  - Have 2 parallel Q-functions to help each other from overfitting to some actions

Image credit: Thomas Simonini (medium.freecodecamp.org

Deep Q-learning

Carnegie Mellon





- Now Q-value has parameter θ: $Q(s, a, \theta)$: an approximator
- Current predicted Q-value: $\hat{Q}(s, a, \theta)$ and its gradient $\nabla_\theta \hat{Q}(s, a, \theta)$
- SDG rule:

$$\theta := \theta - \alpha[r(s, a) + \gamma \max_{a \in \mathcal{A}} \hat{Q}(s', a, \theta) - \hat{Q}(s, a, \theta)]\nabla_\theta \hat{Q}(s, a, \theta) \quad (6)$$

- Demo video: ▸ Atari

Image credit: Thomas Simonini (medium.freecodecamp.org