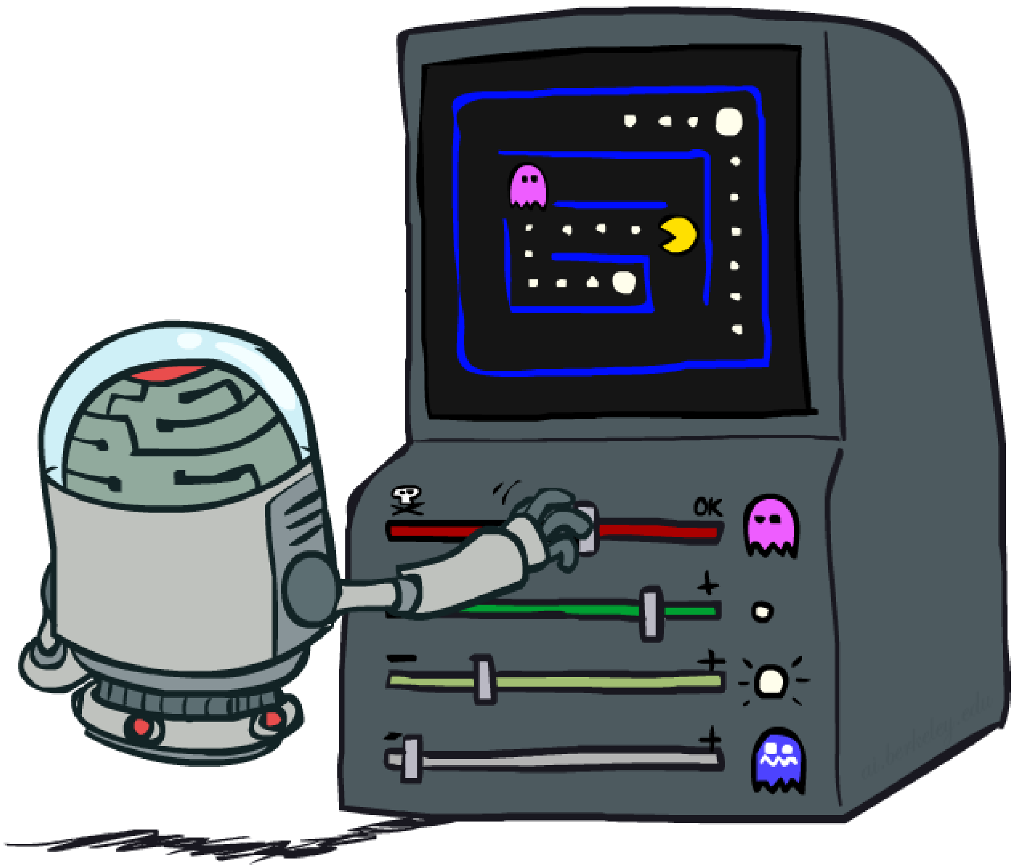
CS 6300: Artificial Intelligence

Reinforcement Learning III: Policy Gradients

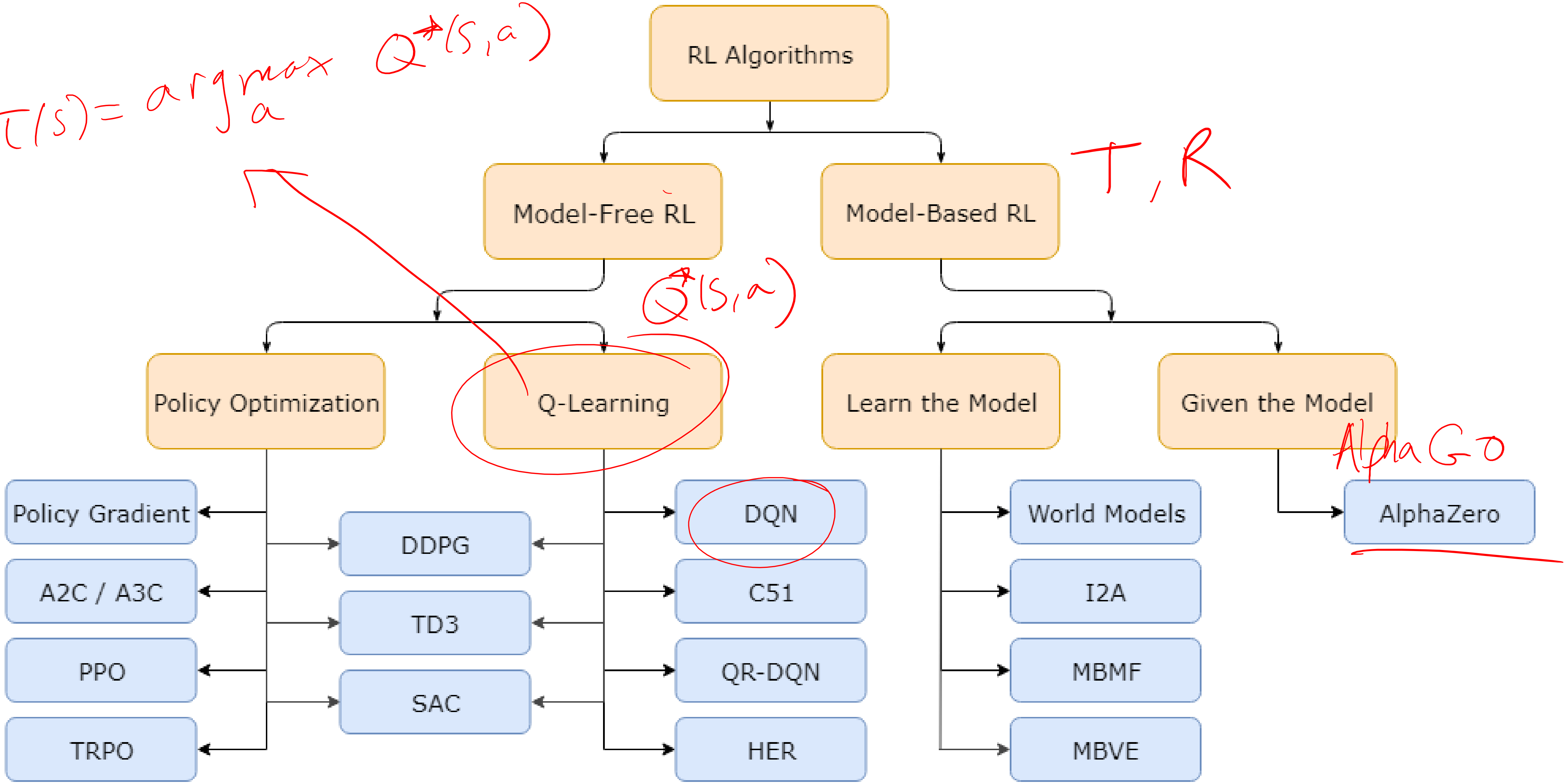


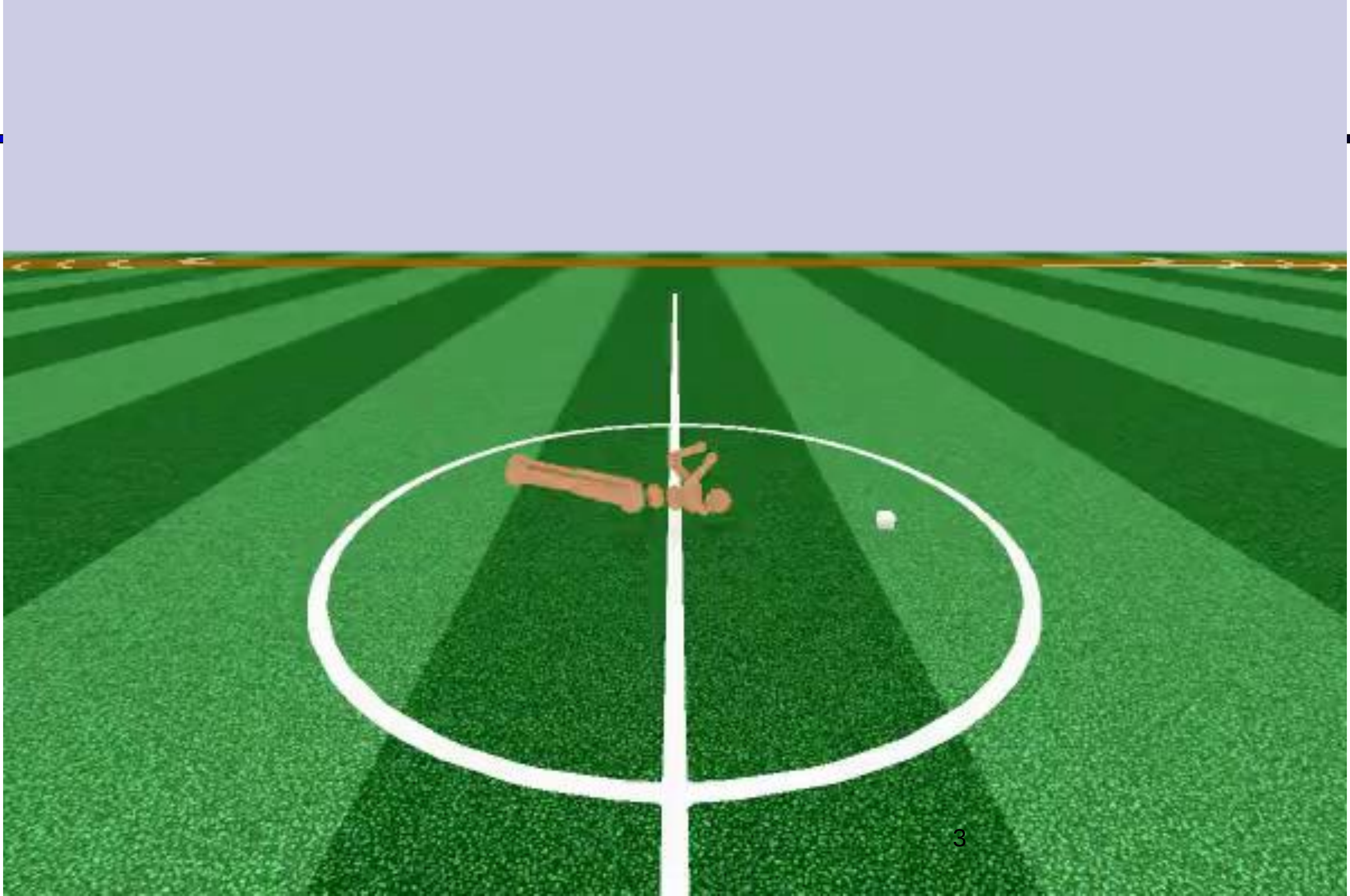
Instructor: Daniel Brown --- University of Utah

[Based on slides created by Dan Klein and Pieter Abbeel http://ai.berkeley.edu.]

Rough Taxonomy of RL Algorithms

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





What is the goal of RL?

- Find a policy that maximizes expected utility (discounted cumulative rewards)

$$\pi^* = arg\max_{\pi} E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s, \pi(s), s')\right]$$

Two approaches to model-free RL

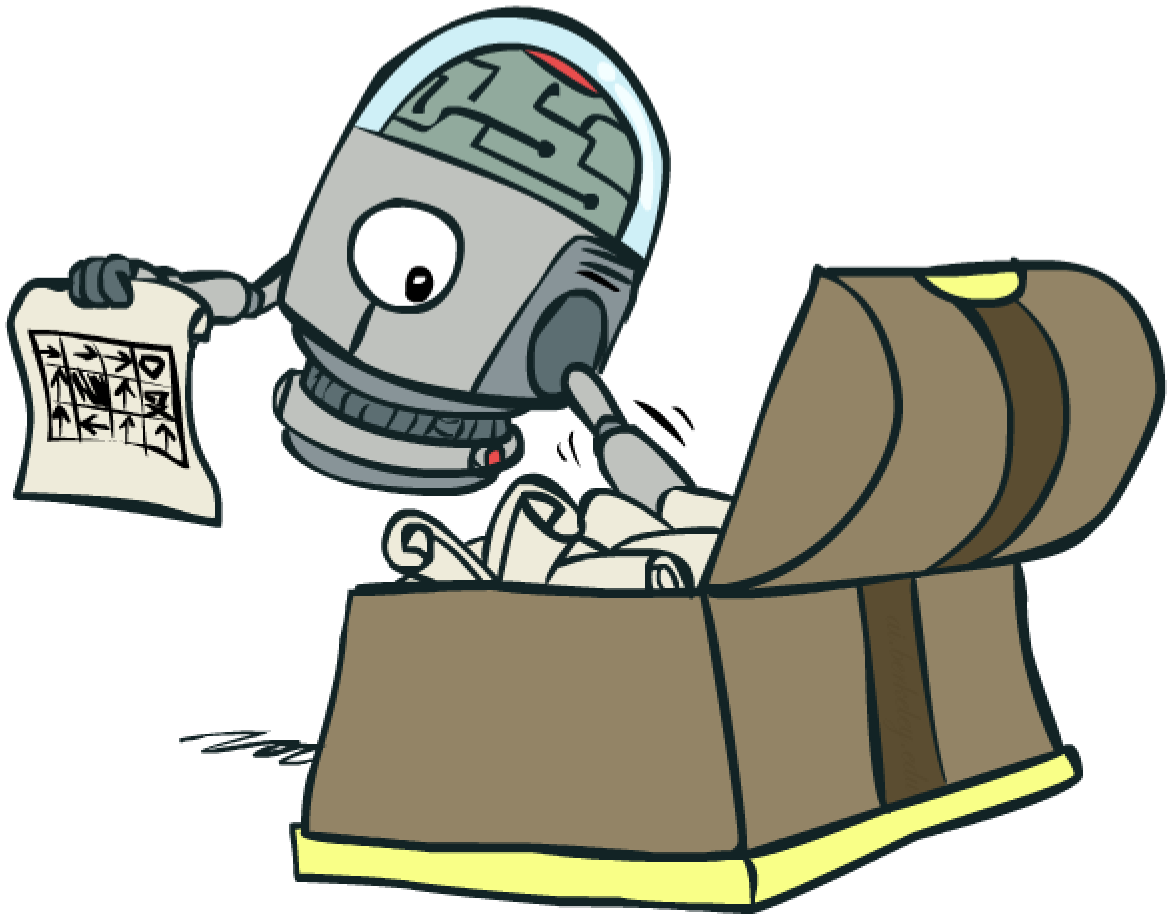
$$\overbrace{Q(s,a)}^{\text{estimation}} \approx \overbrace{r + \gamma \max_{a'} Q(s',a')}^{\text{target}}$$

- Learn Q-values
  - Trains Q-values to be consistent. Not directly optimizing for performance.
  - Use an objective based on the Bellman Equation

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

- Learn Policy Directly
  - Have a parameterized policy π_θ
  - Update the parameters θ to optimize performance of policy.

Policy Search



Policy Search

- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)
  - We'll see this distinction between modeling and prediction again later in the course

$$\pi(s) = \arg\max_a Q^*(s,a) \approx \arg\max_a \sum_{i=1}^{K} w_i f_i(s,a)$$

- Solution: learn policies that maximize rewards, not the values that predict them

- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by hill climbing on feature weights

policy pref of $a = \vec{w}^T f(s,a)$

Policy Search

- Simplest policy search:
  - Start with an initial linear value function or Q-function
  - Nudge each feature weight up and down and see if your policy is better than before

- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical

- Better methods exploit lookahead structure, sample wisely, change multiple parameters…

Policy Search

[Andrew Ng]

[Video: HELICOPTER]

Preliminaries

$\pi(s_0)$

- Trajectory (rollout, episode) $\tau = (s_0, a_0, s_1, a_1, \dots)$
  - $s_0 \sim \rho_0(\cdot), \;\; s_{t+1} \sim P(\cdot\,|s_t, a_t)$ $\quad T(s_0, a_0, s_1)$
- Rewards $r_t = R(s_t, a_t, s_{t+1})$
- Finite-horizon undiscounted return of a trajectory

$$R(\tau) = \sum_{t=0}^{T} r_t$$

T — max timesteps

- Actions are sampled from a parameterized policy π_θ

$$a_t \sim \pi_\theta(\cdot\,|s_t)$$

Preliminaries

$P(x_1, x_2, x_3) = P(x_1) P(x_2|x_1) P(x_3|x_1, x_2)$

- Probability of a trajectory (rollout, episode) $\tau = (s_0, a_0, s_1, a_1, \ldots)$

init start dist

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

- Expected Return of a policy $J(\pi)$

$E[x] = \sum P(x) x$

$$J(\pi) = \sum_{\tau} P(\tau|\pi) R(\tau) = E_{\tau \sim \pi}[R(\tau)]$$

$R(\tau) = \sum_{(s,a,s') \in \tau} R(s,a,s')$

- Goal of RL: Solve the following optimization problem

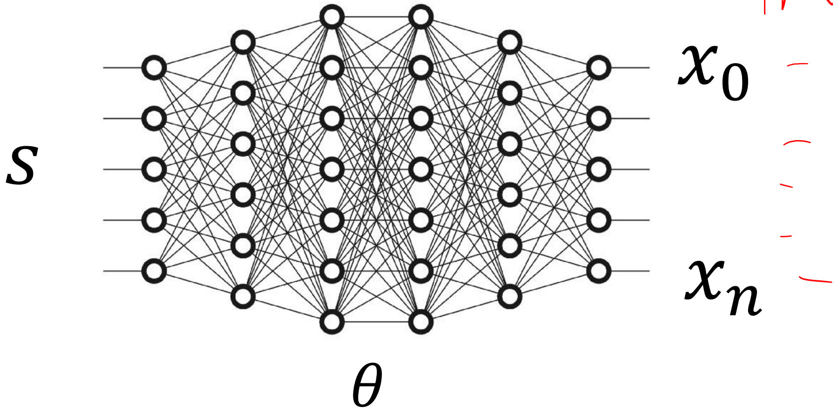
$$\pi^* = \underset{\pi}{\operatorname{argmax}} J(\pi)$$

How should we parameterize our policy?

- We need to be able to do two things:
  - Sample actions $a_t \sim \pi_\theta(\cdot\,|s_t)$
  - Compute log probabilities $\log \pi_\theta(a_t|s_t)$
- Categorical (classifier over discrete actions)
  - Typically, you output a value x_i for each action (class) and then the probability is given by a softmax equation

softmax eqn

$|A| = n$



$$\pi_\theta(a_i|s) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

How should we parameterize our policy?

- Diagonal Gaussian (distribution over continuous actions)

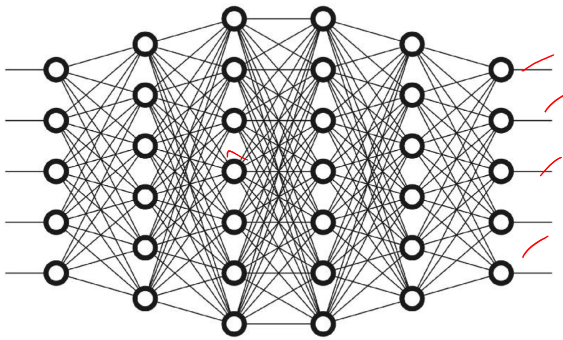
$$a \sim N(\mu, \Sigma)$$

where Σ has non-zero elements only on the diagonal.

Thus, an action can be sampled as

$$a = \mu_\theta(s) + \sigma_\phi(s) \odot z, \qquad z \sim N(0, I)$$

s

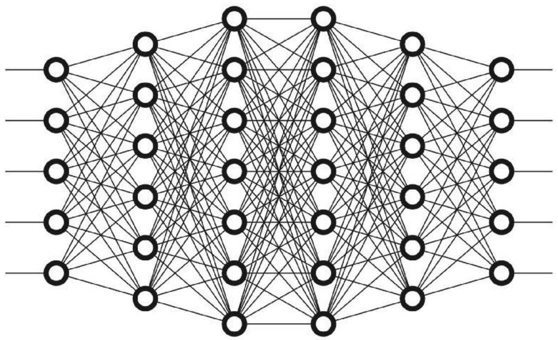


joint angle

μ_θ

θ

s



$\log \sigma_\phi$

$\exp \log \sigma_\phi$
$= \sigma_\phi$

ϕ

Goal: Update Policy via Gradient Ascent

- We have a parameterized policy and we want to update it so that it maximizes the expected return.
- We want to find the gradient of the return with respect to the policy parameters and step in that direction.

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \Big|_{\theta_k}$$

Policy gradient

Fact #1

- Probability of a trajectory:
  - The probability of a trajectory $\tau = (s_0, a_0, \dots s_{T+1})$ given that actions come from π_θ is

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$$

Fact #2

- Log-probability of a trajectory:
  - The log-probability of a trajectory $\tau = (s_0, a_0, \dots s_{T+1})$ given that actions come from π_θ is

$$\log P(\tau|\pi) = \log\left(\rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)\right)$$

$$= \log \rho_0(s_0) + \sum_{t=0}^{T}(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

Fact #3

- Grad-Log-Prob of a Trajectory
  - Note that gradients of everything that doesn't depend on θ is 0.

$$\nabla_\theta \log P(\tau|\theta) = \cancel{\nabla_\theta \log \rho_0(s_0)}^{\,0} + \sum_{t=0}^{T} (\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)}^{\,0} + \nabla_\theta \log \pi_\theta(a_t|s_t))$$

$$= \sum_{t=0}^{T} (\nabla_\theta \log \pi_\theta(a_t|s_t))$$

Fact #4

- Log-Derivative Trick:

  $\frac{d}{dx} \log x = \frac{1}{x}$

  - This is based on the rule from calculus that the derivative of log x is 1/x

$$\nabla_\theta P(\tau|\pi) = P(\tau|\pi)\nabla_\theta \mathrm{logP}(\tau|\theta)$$

π_θ

$$\frac{d}{dx}\log g(x) = \frac{1}{g(x)}\frac{d}{dx}g(x) \quad \Rightarrow \quad g(x)\frac{d}{dx}\log g(x) = \frac{d}{dx}g(x)$$

Derivation of Policy Gradient

$E[x] = \sum P(x)\, x$

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)]$$

$$= \nabla_\theta \sum_\tau P(\tau|\theta) R(\tau)$$

$$= \sum_\tau \nabla_\theta P(\tau|\theta) R(\tau)$$

grad log trick

$$= \sum_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta)\, R(\tau)$$

Fact #4

$$= E_{\tau\sim\pi_\theta}[\nabla_\theta \log P(\tau|\theta)\, R(\tau)]$$

Expectation of R.V.

$$= E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)\right]$$

Fact #3

The Policy Gradient

REINFORCE

- We can now perform gradient ascent to improve our policy!

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta) \Big|_{\theta_k}$$

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ \underbrace{R(\tau)}_{\text{total reward of traj } \tau}\right]$$

Estimate with a sample mean over a set D of policy rollouts given current parameters

$$\approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)$$

How would you implement this?

1. Start with random policy parameters θ_0
2. Run the policy in the environment to collect N rollouts (episodes) of length T and save returns of each trajectory.

$$a_t \sim \pi_\theta(\cdot\,|s_t) \Rightarrow (s_0, a_0, r_0, s_1, a_1, r_1, \dots, r_T, s_{T+1}) \qquad s_0 \sim \rho_0(\cdot)$$

$$D = \{\tau_1, \dots \tau_N\}, \qquad R = \{R(\tau_1), \dots R(\tau_N)\}$$

3. Compute policy gradient

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)\right]$$

4. Update policy parameters

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta)\Big|_{\theta_k}$$

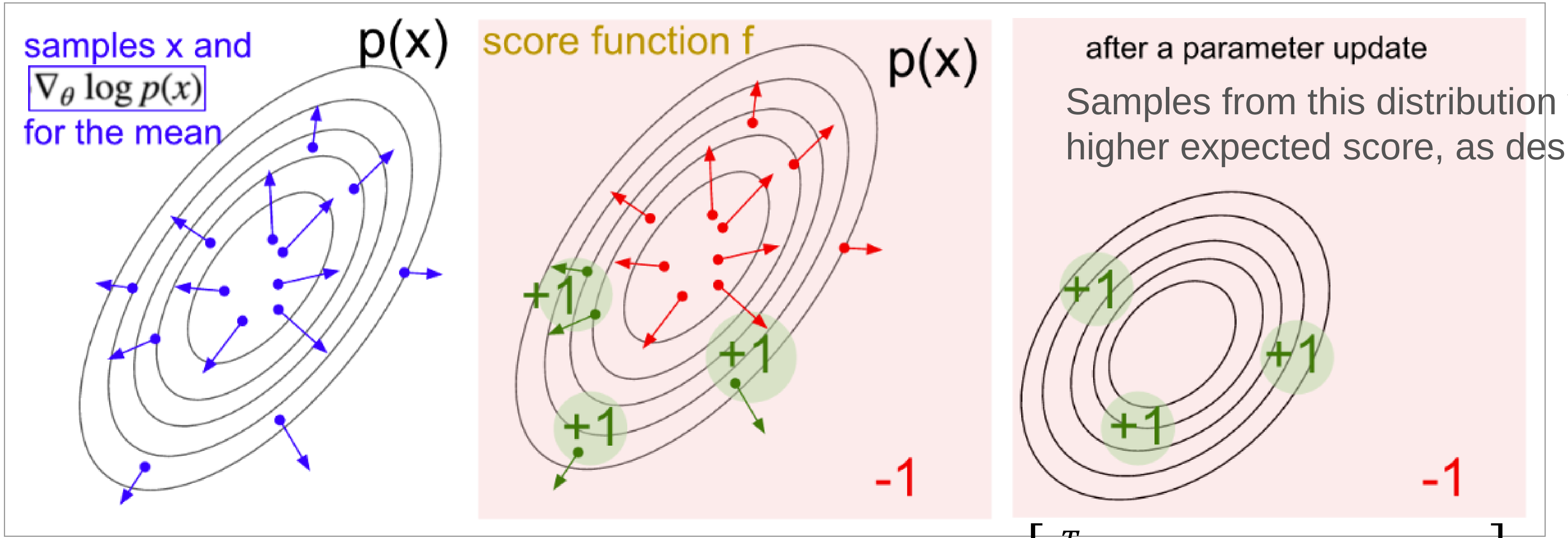
5. Repeat (Go to 2)

Some more intuition (thanks to Andrej Karpathy)

- Blue Dots: samples from Gaussian
- Blue arrows: gradients of the log probability with respect to the gaussian's mean parameter

- We score each sample
- Red have score -1
- Green have scores +1

- To update the Gaussian mean parameter, we average up all the green arrows, and the *negative* of the red arrows.



Samples from this distribution will have a higher expected score, as desired.

$$\nabla_\theta J(\pi_\theta) = E_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\ R(\tau)\right]$$

https://karpathy.github.io/2016/05/31/rl/

Policy Gradient RL Algorithms

- We can directly update the policy to achieve high reward.
- Pros:
  - Directly optimize what we care about: Utility!
  - Naturally handles continuous action spaces!
  - Can learn specific probabilities for taking actions.
  - Often more stable than value-based methods (e.g. DQN).
- Cons:
  - On-Policy -> Sample-inefficient we need to collect a large set of new trajectories every time the policy parameters change.
  - Q-Learning methods are usually more data efficient since they can reuse data from any policy (Off-Policy) and can update per sample.

Many forms of policy gradients

$s_0\, a_0\, r_1\, s_1\, a_1\, r_2\, s_2 \;\ldots\; s_t\, a_t\, r_{t+1}$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

Σ

What we derived: $\Phi_t = R(\tau),$

Follows a similar derivation: $\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}),$

https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

- What is better about the second approach?
  - Focuses on rewards in the future!
  - Less variance -> less noisy gradients.

Many forms of policy gradients

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

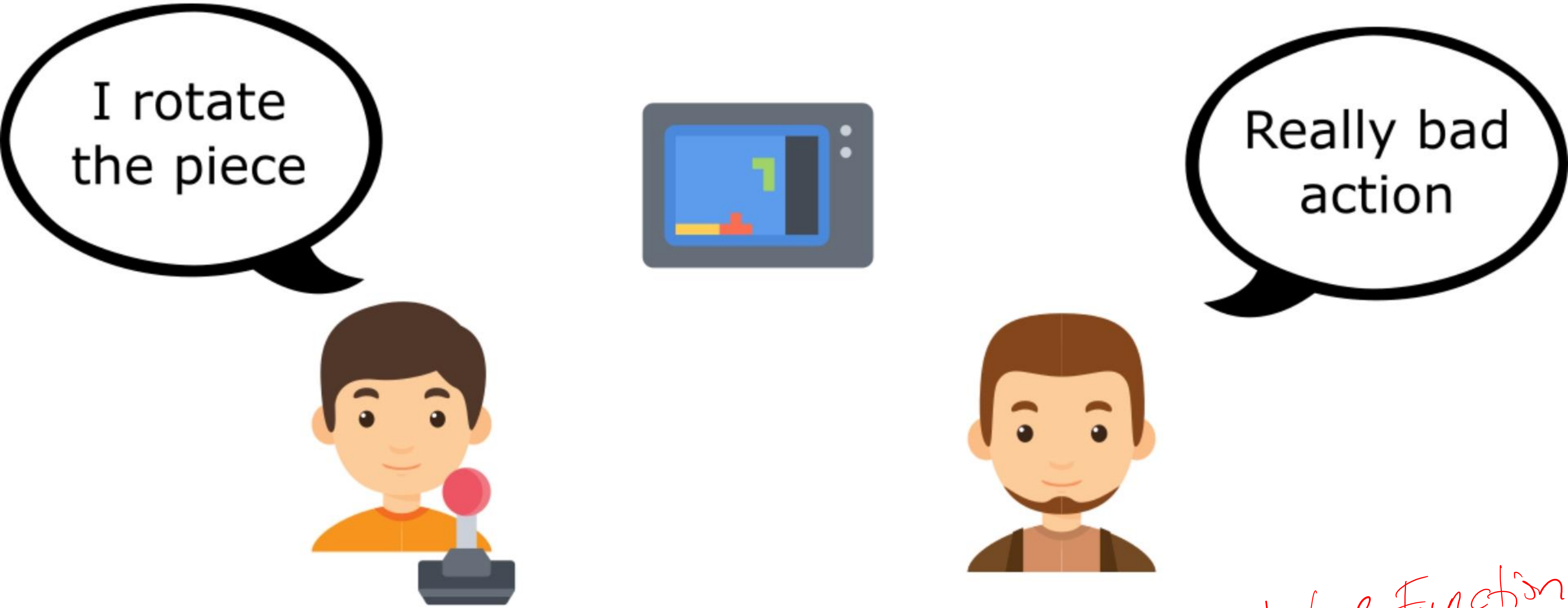
$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}),$$

Looks familiar….

$$\Phi_t = Q^{\pi_\theta}(s_t, a_t)$$

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t'=t}^{T} R(s_t, a_t, s_{t+1}) \,\middle|\, s_t = s, a_t = a\right]$$

- Now we have an approach that combines a parameterized policy and a parameterized value function!



Actor π_θ

Critic Value Function

Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

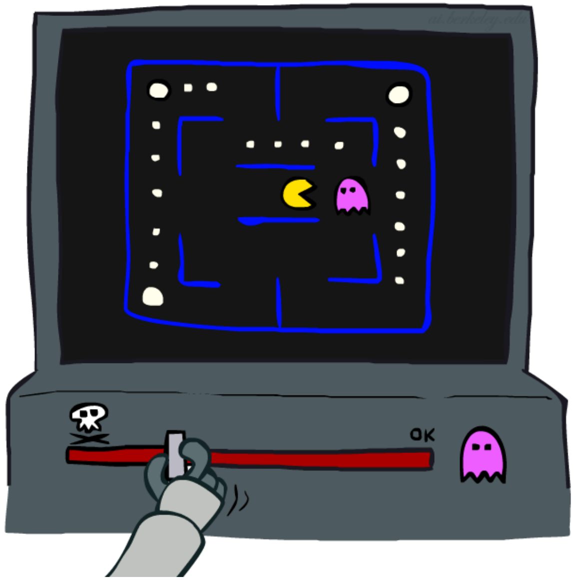
- Q-learning with linear Q-functions:

$$\text{transition } = (s,a,r,s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \,[\text{difference}]$$ Exact Q's

$$w_i \leftarrow w_i + \alpha \,[\text{difference}]\, f_i(s,a)$$ Approximate Q's



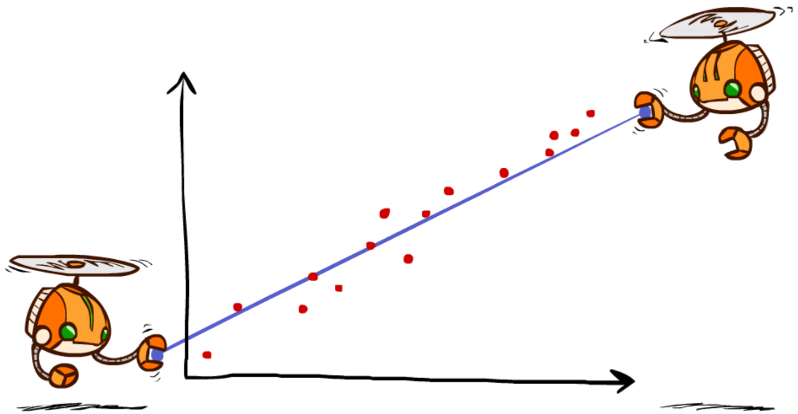
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: disprefer all states with that state's features

- Formal justification: online least squares

Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$



Approximate q update explained:

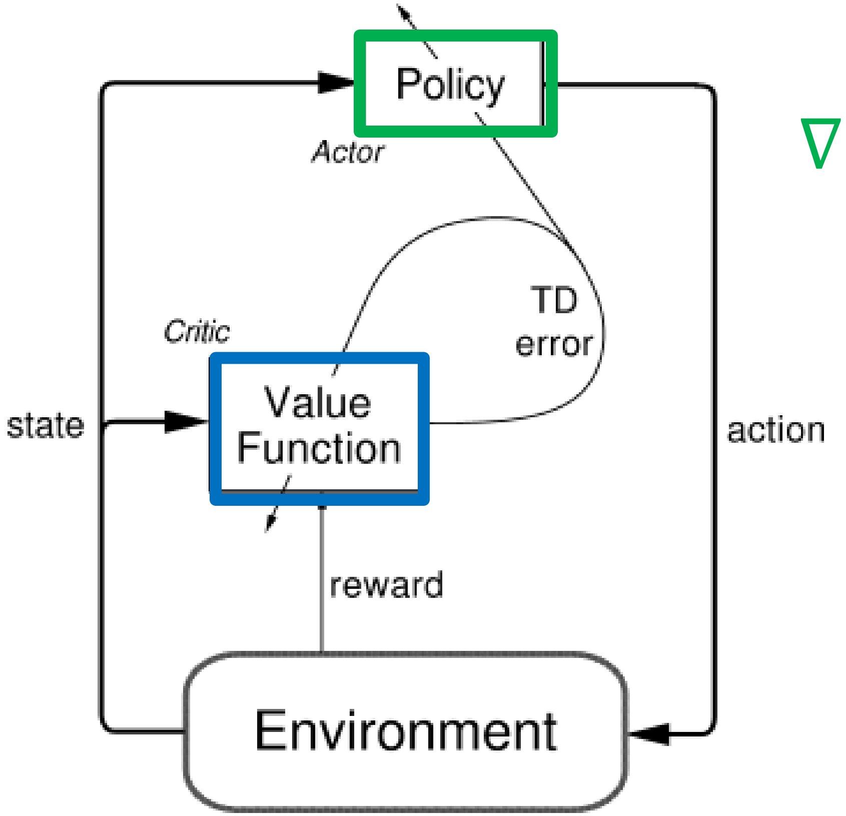
$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

“target” “prediction”

$$\theta \leftarrow \theta + \alpha\left(r + \gamma \max_{a'} Q_T(s', a'; \theta^-) - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta)$$

Actor Critic Algorithms

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function



$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_w^{\pi_\theta}(s_t, a_t)\right]$$

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta)\Big|_{\theta_k}$$

$$\delta = (r_t + \gamma Q_w^{\pi_\theta}(s_{t+1,}a_{t+1}) - Q_w^{\pi_\theta}(s_t, a_t))$$

$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_\theta Q_w^{\pi\_\theta}$$

Q Actor Critic Algorithm Pseudo Code

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \ldots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:

$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

  and use it to update the parameters of Q function:

$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$

  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

π
actor

TD
update

Adapted from Lilian Weng's post "Policy Gradient algorithms"

Many forms of policy gradients

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t \right]$$

$$\Phi_t = R(\tau), \qquad \Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}), \quad \approx \quad \Phi_t = Q^{\pi_\theta}(s_t, a_t)$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - \underbrace{b(s_t)}_{\text{baseline}}$$

? better than ave

$$\Phi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \quad > 0 \quad < 0?$$

Advantage Function

Advantage Actor Critic (A2C)

- Combining value learning with direct policy learning
  - One example is policy gradient using the advantage function

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

$$\Phi_t = A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

TD error $\delta_t = \overbrace{r(s_t, a_t) + \gamma V^\pi(s_{t+1})}^{\text{Expectation}} - V^\pi(s_t)$

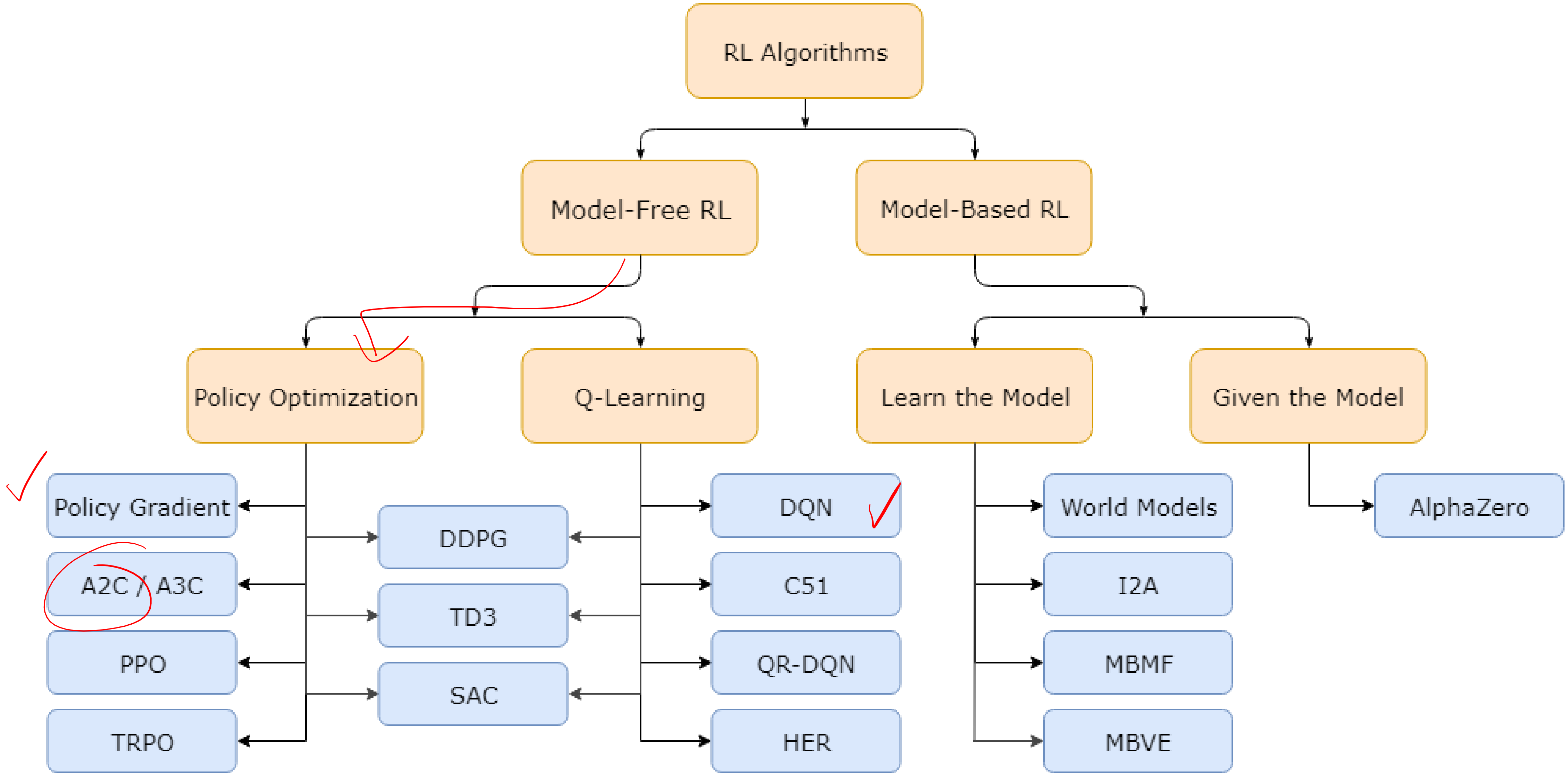
Policy gradient update

$$\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\pi_\theta)\Big|_{\theta_k}$$

TD-Learning update

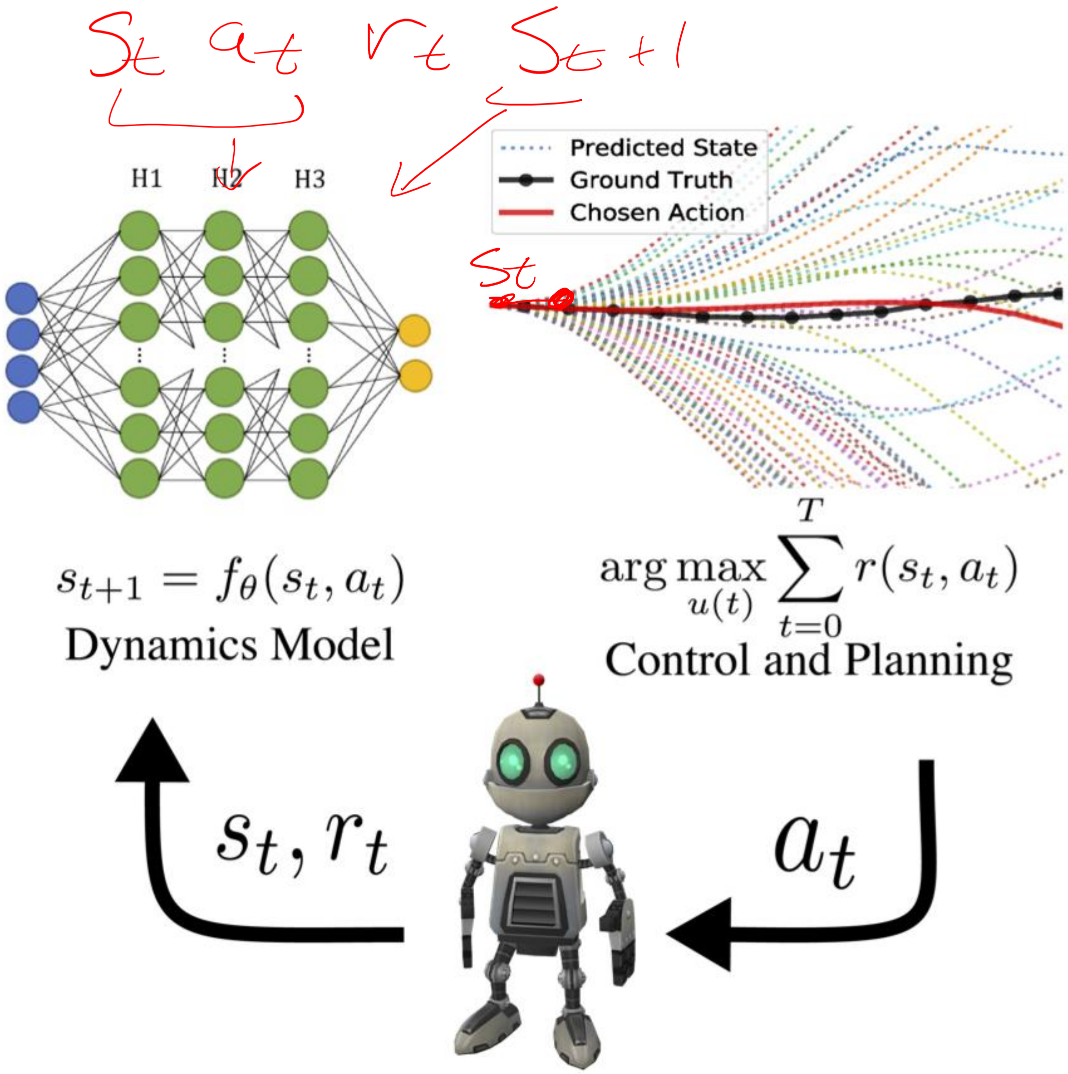
$$w_{k+1} \leftarrow w_k + \alpha \delta_t \nabla_w V(s, a; w)$$

Rough Taxonomy of RL Algorithms



Model-Based RL via Model-Predictive Control

- Use model to plan good looking sequence of actions.
- Take a step
- Update model of transitions
- Repeat



- Next time: Alpha Go