Policy Gradient

DAVIDE BACCIU – BACCIU@DI.UNIPI.IT

IN SUPREMÆ DIGNITATIS
1343
UNIVERSITÀ DI PISA

Outline (a 2-days lecture)

- ✓ Introduction
- ✓ Finite Difference Policy Gradient
- ✓ Monte-Carlo Policy Gradient
- ✓ Actor-Critic approaches
- ✓ Natural gradient
- ✓ Deep policy networks
  - ✓ Asynchronous Advantage
  - ✓ (Deep) Deterministic policy gradient
  - ✓ Learning with continuous actions

Introduction

Policy-Based Reinforcement Learning

Previously

- ✓ Approximate value or action-value function using parameters θ

$$V_\theta(s) \approx V^\pi(s)$$

$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

- ✓ Generate policy from the value function (e.g. using ϵ-greedy)

Now

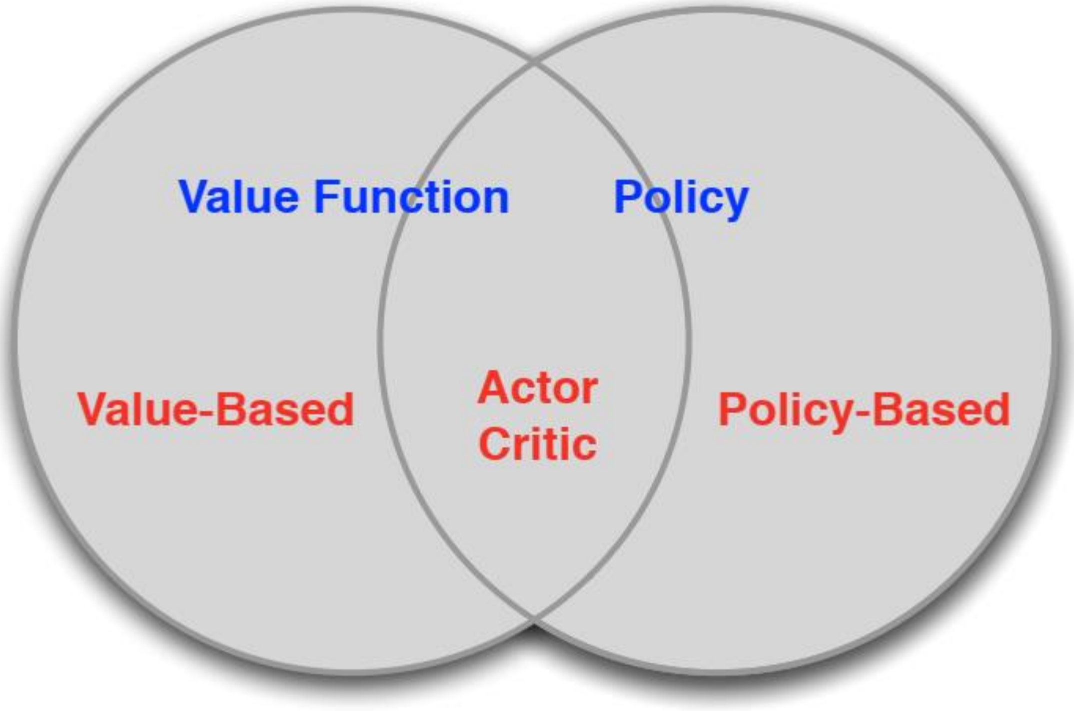
- ✓ Parametrise the policy

$$\pi_\theta(s,a) = P(a|s,\theta)$$

- ✓ Focus again on model-free reinforcement learning

Value, Policy and the Actor-Critic

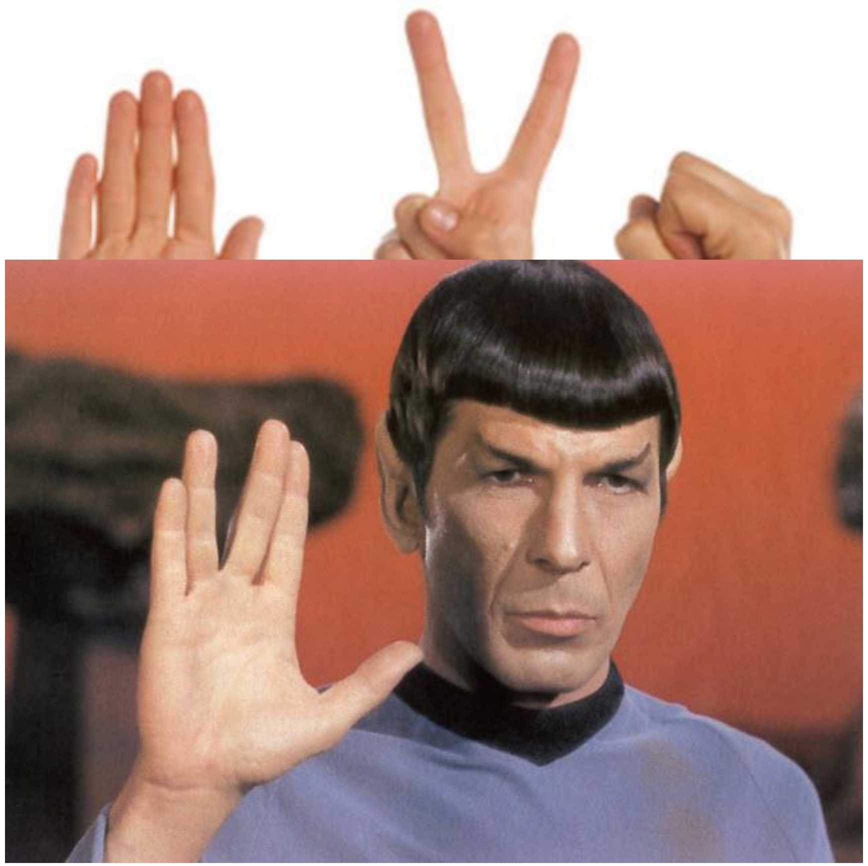
- ✓Value Based
  - ✓Learnt Value Function
  - ✓Implicit policy (e.g. ϵ-greedy)
- ✓Policy Based
  - ✓No Value Function
  - ✓Learnt Policy
- ✓Actor-Critic
  - ✓Learnt Value Function
  - ✓Learnt Policy



Policy-Based RL – Pros and Cons

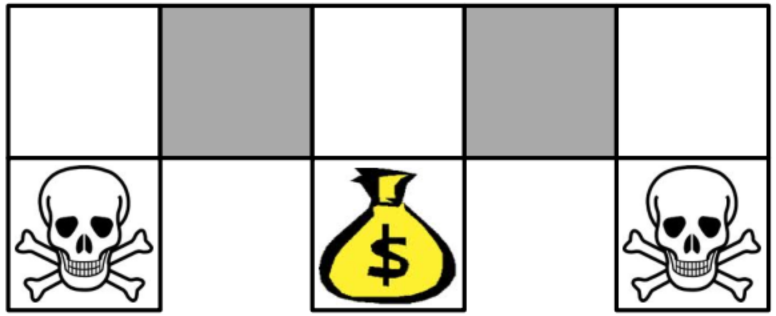
- ✓Advantages
  - ✓Better convergence properties
  - ✓Effective in high-dimensional or continuous action spaces
  - ✓Can learn stochastic policies
- ✓Disadvantages
  - ✓Typically converge to a local rather than global optimum
  - ✓Evaluating a policy is typically inefficient and high variance

Example: Rock-Paper-Scissors



- ✓Two-player game of rock-paper-scissors
  - ✓Scissors beats paper
  - ✓Rock beats scissors
  - ✓Paper beats rock
- ✓Consider policies for iterated rock-paper-scissors
  - ✓A deterministic policy is easily exploited
  - ✓A uniform random policy is optimal (i.e. Nash equilibrium)

Example: Aliased Gridworld (I)



- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W)

$$\phi(s,a) = \mathbf{1}(\text{wall to N; a = move E})$$

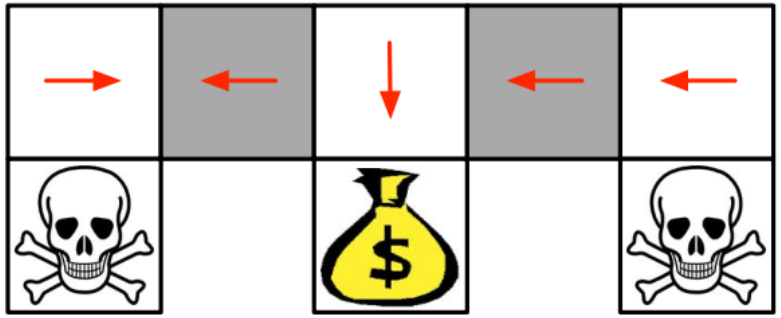
- Value-based RL - Using an approximate value function

$$Q_\theta(s,a) = f(\phi(s,a),\theta)$$

- Policy-based RL - Using a parametrised policy

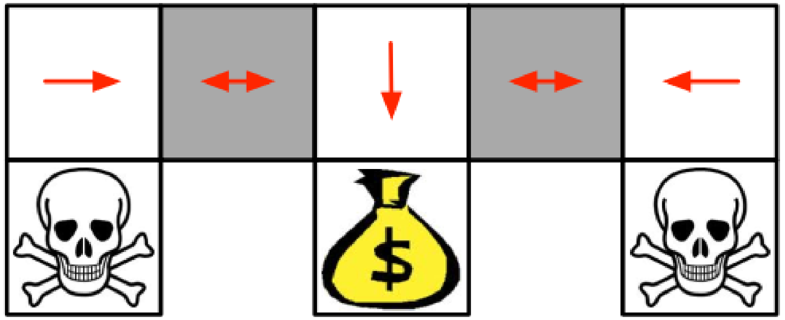
$$\pi_\theta(s,a) = g(\phi(s,a),\theta)$$

Example: Aliased Gridworld (II)



- ✓Under aliasing, an optimal deterministic policy will either
  - ✓move W in both grey states (red arrows)
  - ✓move E in both grey states
- ✓Either way, it can get stuck and never reach the money
- ✓Value-based RL learns a near-deterministic policy (e.g. greedy or ϵ-greedy)
- ✓So it will traverse the corridor for a long time

Example: Aliased Gridworld (III)



- An optimal stochastic policy will randomly move E or W in grey states
  - $\pi_\theta(\text{wall to N and S, move E}) = 0.5$
  - $\pi_\theta(\text{wall to N and S, move W}) = 0.5$
- Reaches goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Objective Functions

- ✓Goal: given policy $\pi_\theta(s, a)$with parameters θ, find best θ
- ✓How to measure the quality of a policy π_θ?
- ✓Episodic environments - Use start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- ✓Continuing environments
  - ✓Average value - $J_{\bar{V}}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
  - ✓Average reward (per time-step) - $J_{\bar{R}}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a)\, \mathcal{R}_s^a$
  - ✓ $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π

Policy Optimization

- Policy based reinforcement learning is an optimisation problem
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex
  - Genetic algorithms
- Greater efficiency often using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- Focus on gradient-based and methods that exploit sequential structure

Policy Gradient

Policy Gradient

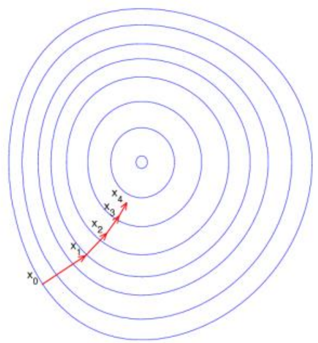
- ✓Let $J(\theta)$ be any policy objective function
- ✓Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy w.r.t. θ

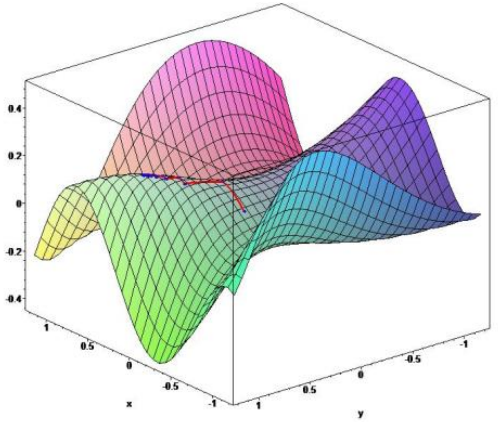
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- ✓$\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

- ✓α is the step size





Gradients by Finite Differences

- ✓To evaluate policy gradient of $\pi_\theta(s, a)$ for each dimension $k \in [1, n]$
  - ✓Estimate k-th partial derivative of objective function w.r.t. θ
  - ✓Perturbe θ by small amount in k-th dimension
  - ✓u_k unit vector with 1 in k-th component and 0 elsewhere

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- ✓Uses n evaluations to compute policy gradient in n dimensions
- ✓Simple, noisy, inefficient (but sometimes effective)
- ✓Works for arbitrary policies, even if policy is not differentiable

Monte-Carlo Policy Gradient

Score Function

- ✓We now compute the policy gradient analytically
- ✓Assume policy π_θ is differentiable whenever it is non-zero and we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- ✓Likelihood ratios exploit the following fundamental identity

$$\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} = \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)$$

- ✓The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Softmax Policy

✓Weight actions using linear combination of features $\phi(s,a)^T\theta$

✓Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) = e^{\phi(s,a)^T\theta}$$

✓The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(\mathrm{s},\mathrm{a}) - \mathbb{E}_{\pi_\theta}[\phi(\mathrm{s},\cdot)]$$

Gaussian Policy

- ✓Natural choice in continuous action spaces
- ✓Mean is a linear combination of state features $\mu(s) = \phi(s)^T\theta$
- ✓Variance σ^2 may be fixed or parametrised
- ✓Gaussian Policy - a$\sim\mathcal{N}(\mu(s), \sigma^2)$
- ✓The score function is

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

One-Step MDPs

- ✓A simple class of one-step MDPs
  - ✓Starting in state $s \sim d(s)$
  - ✓Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- ✓Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a)\mathcal{R}_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_{s,a}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r]$$

Policy Gradient Theorem

- ✓The policy gradient theorem generalises the likelihood ratio approach to multi-step MDPs
- ✓Replaces instantaneous reward r with long-term value $Q^{\pi}(s,a)$
- ✓Policy gradient theorem applies to start state objective, average reward and average value objective

Theorem

For any differentiable policy $\pi_\theta(s,a)$, for any of the policy objective functions $J_1, J_{\bar{R}}, \frac{1}{1-\gamma} J_{\bar{V}}$, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

REINFORCE – MC Policy Gradient

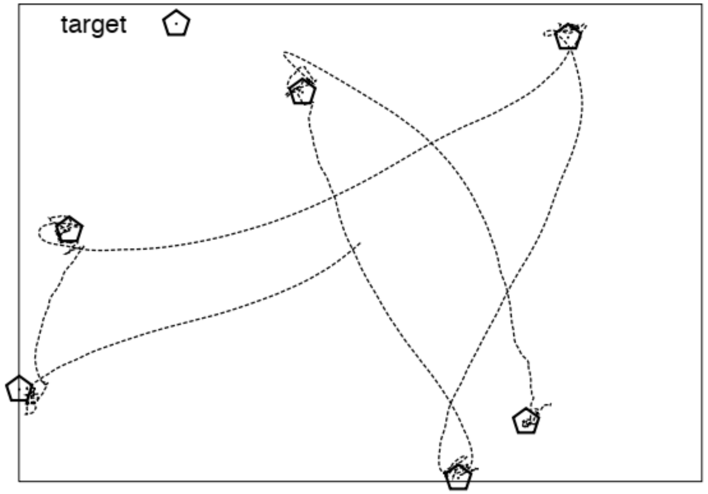
- ✓Update parameters by stochastic gradient ascent
- ✓Using policy gradient theorem

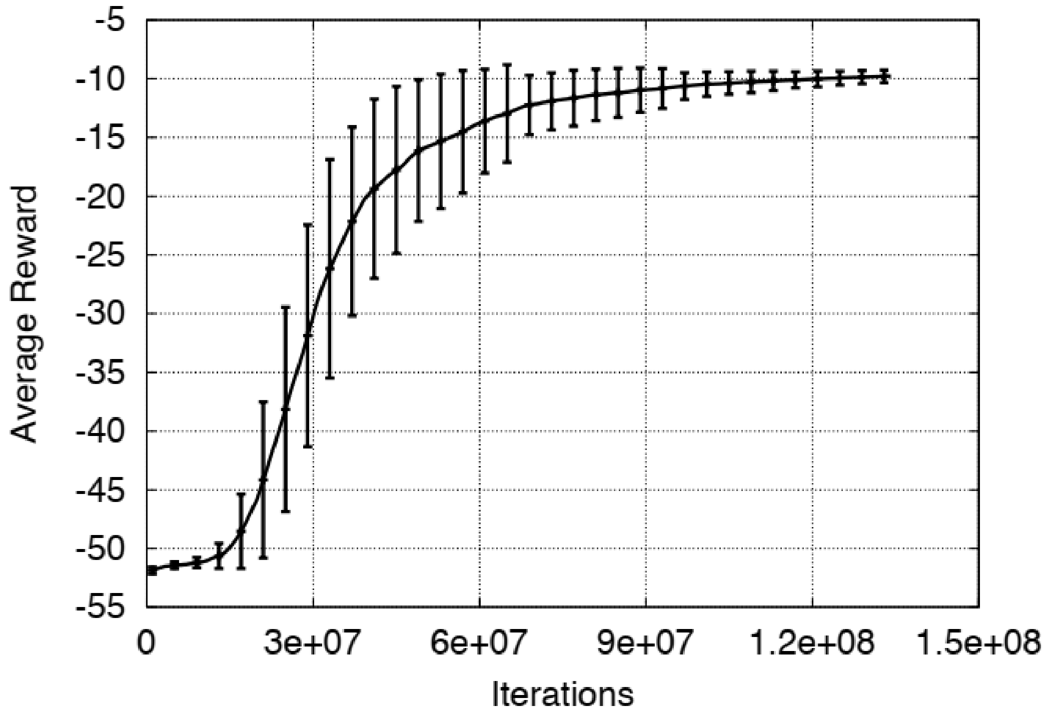
function REINFORCE
  Initialise θ arbitrarily
  for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
    end for
  end for
  return θ
end function

- ✓Using return v_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

Puck World Example





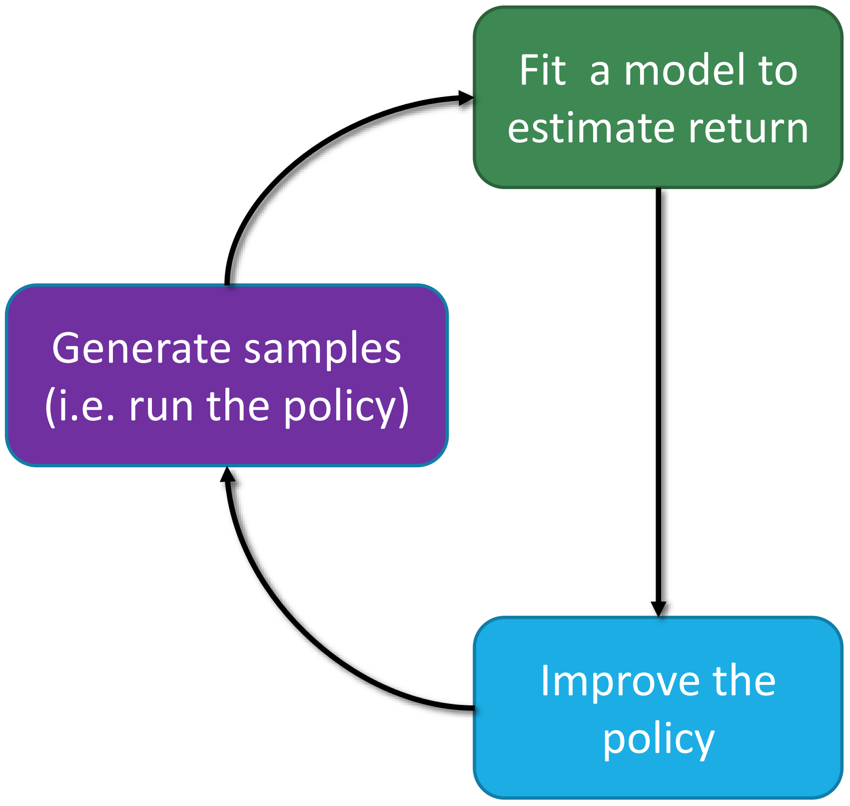
- ✓Continuous actions exert small force on puck
- ✓Puck is rewarded for getting close to the target
- ✓Target location is reset every 30 seconds
- ✓Policy is trained using variant of Monte-Carlo policy gradient

Evaluate the MC Policy Gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} v_t^i$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(s_t, a_t)\right)\left(\sum_{t=1}^{T} v_t^i\right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



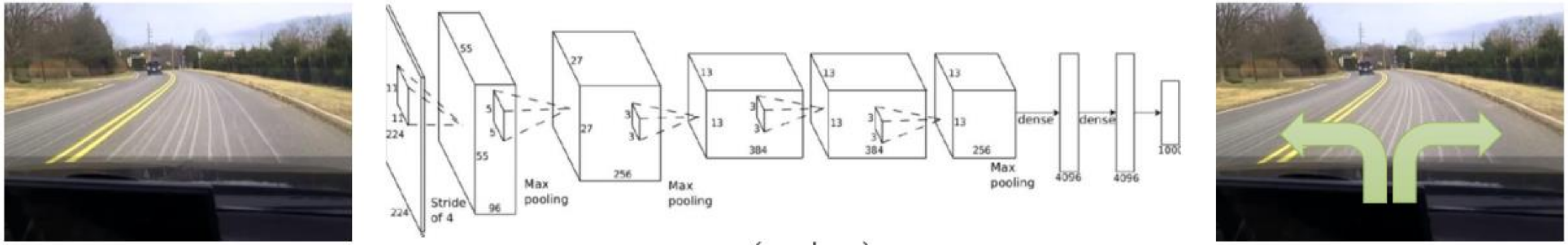
Policy Gradient Vs Maximum Likelihood

Policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t, a_t)\right)\left(\sum_{t=1}^{T} v_t^i\right)$$

Maximum Likelihood

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t, a_t)\right)$$



$\mathbf{s}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ $\mathbf{a}_t$

PG Vs ML – Gaussian Policy

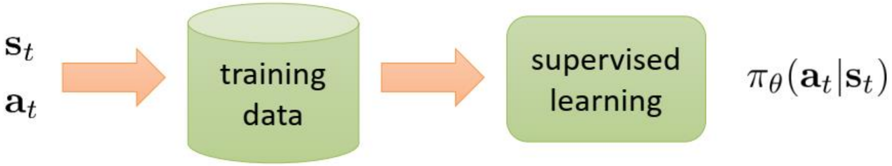
Policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t^i, a_t^i)\right)\left(\sum_{t=1}^{T} v_t^i\right)$$

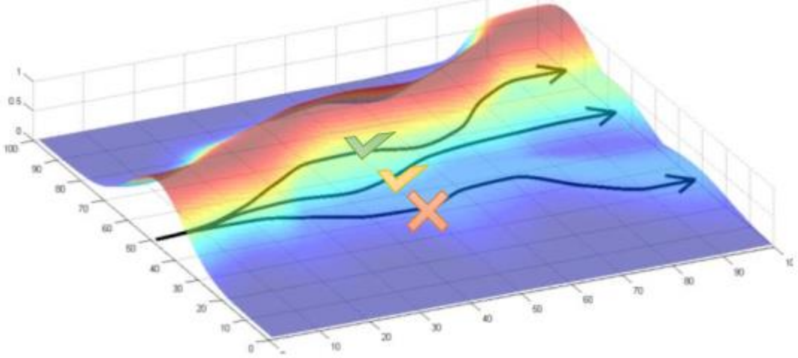
Maximum Likelihood

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t^i, a_t^i)\right)$$

$$\nabla_\theta \log \pi_\theta(s_t, a_t)$$



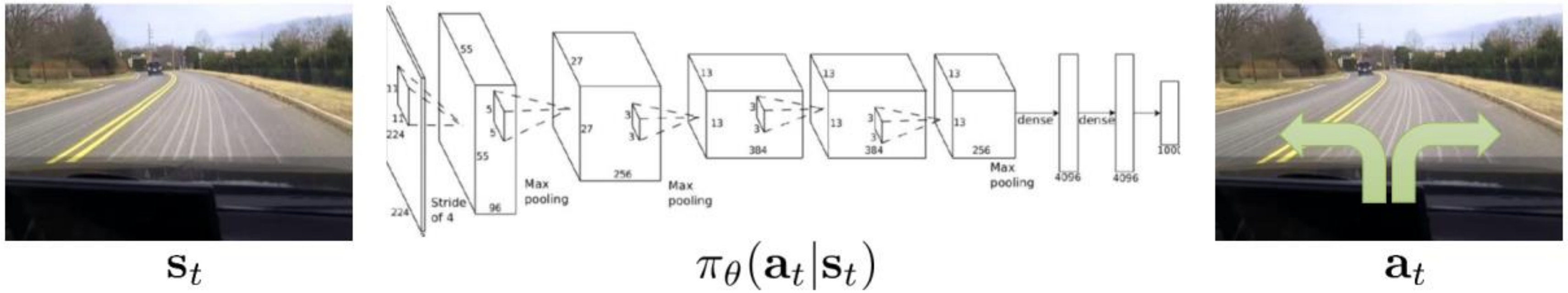
- ✓Good things are made more likely
- ✓Bad things are made less likely
- ✓Formalizes the notion of “trial and error”



Policy gradient is on-policy

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s_1,a_1)\ldots(s_T,a_T)\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$$

This runs on *real-world*



✓Neural networks change only slightly with each gradient step

✓On-policy learning can be extremely inefficient!

Off-policy Learning Policy Gradient

Sample experience from a behaviour policy $\mathrm{s,a} \sim \bar{\pi}$

$$J(\theta) = \mathbb{E}_{\mathrm{s,a} \sim \bar{\pi}}\left[\frac{\pi_\theta(s,a)}{\bar{\pi}(s,a)} Q^{\pi_\theta}(s,a)\right]$$

Importance sampling reweighting scheme

Sample from old policy $\bar{\pi} = \pi_\theta(s,a)$ and estimate new $\pi_{\theta'}(s,a)$

$$\nabla_{\theta'} J(\theta') = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_{\theta'}(s_t^i, a_t^i)\left(\prod_{t'=1}^{T} \frac{\pi_{\theta'}(s_{t'}^i, a_{t'}^i)}{\pi_\theta(s_{t'}^i, a_{t'}^i)}\right)\left(\sum_{t'=t}^{T} v_{t'}^i\right)\right)$$

Exponential on T

NN-PG with Automatic Differentiation

- One would like to avoid to have to compute this explicitly

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t, a_t)\, v_t^i\right)$$

- We know something fairly similar

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_t, a_t)\right)$$

- It is sufficient to compute the loss by reweighting the maximum likelihood
  - Cross-entropy (Softmax)
  - Squared Error (Gaussian)

In Pseudo-Code – Maximum Likelihood

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = tf.reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

In Pseudo-Code – Policy Gradient

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Policy Gradient – Wrap-Up

- ✓Policy gradient is on-policy
  - ✓Off-policy variant
  - ✓Incorporate example demonstrations using importance sampling
- ✓Policy gradient has high variance
  - ✓Not supervised learning
- ✓Gradients will be noisy
  - ✓Use larger batches to control noisy estimates
- ✓Tweaking learning rates is harder
  - ✓Adaptive step size rules (ADAM as a fist approximation)

Actor Critic

Reducing Variance Using a Critic

- ✓Monte-Carlo policy gradient still has high variance
- ✓We use a critic to estimate the action-value function,

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$

- ✓Actor-critic algorithms maintain two sets of parameters
  - ✓Critic - Updates action-value function parameters w
  - ✓Actor - Updates policy parameters θ, in direction suggested by critic
- ✓Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$$

Estimating the Action-Value Function

- ✓The critic is solving a familiar problem: policy evaluation
- ✓How good is policy π_θ for current parameters θ?
- ✓This problem was explored in previously
  - ✓Monte-Carlo policy evaluation
  - ✓Temporal-Difference learning
  - ✓TD(λ)
- ✓Could also use least-squares policy evaluation

Action-Value Actor-Critic

- ✓Simple actor-critic algorithm based on action-value critic
- ✓Using linear approximation $Q_w(s,a) = \phi(s,a)^T w$
  - ✓Critic - Updates w by linear TD(0)
  - ✓Actor - Updates θ by policy gradient

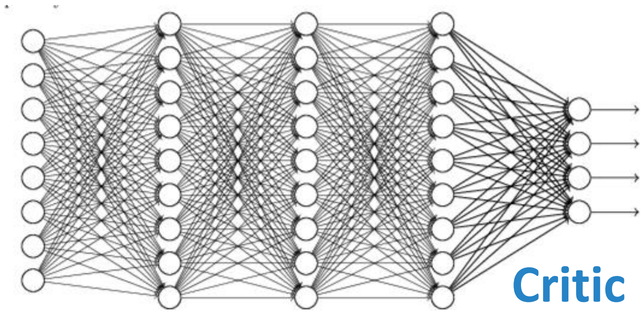
function QAC
- Initialise s, θ
- Sample $a \sim \pi_\theta$
- **for** each step **do**
  - Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
  - Sample action $a' \sim \pi_\theta(s', a')$
  - $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
  - $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
  - $w \leftarrow w + \beta \delta \phi(s, a)$
  - $a \leftarrow a', s \leftarrow s'$
- **end for**

end function

Actor-Critic Architectures

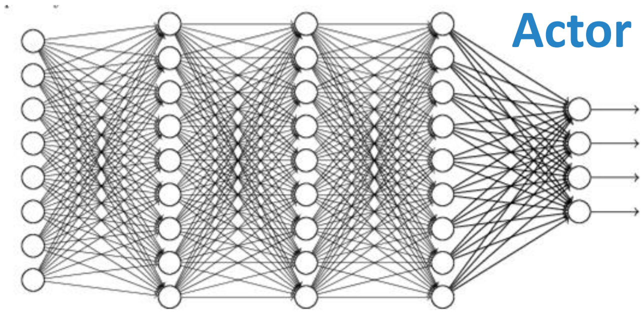
Independent

s



$Q_w(s, a)$

s



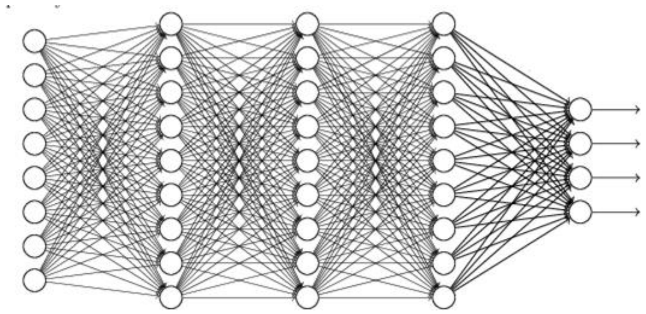
$\pi_\theta(s, a)$

+Simpler and stabler

- No feature sharing

Joint

s



$Q_w(s, a)$

$\pi_w(s, a)$

Bias in Actor-Critic Algorithms

- ✓Approximating the policy gradient introduces bias
- ✓A biased policy gradient may not find the right solution
  - ✓e.g. if $Q_w(s,a)$ uses aliased features, can we solve gridworld example?
- ✓Luckily, if we choose value function approximation carefully, we can avoid introducing any bias
  - ✓We can still follow the exact policy gradient

Compatible Function Approximation

Theorem (Compatible Function Approximation)

If the following two conditions are satisfied:

1. Value function approximator is compatible to the policy

$$\nabla_w Q_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

2. Value function parameters w minimise the mean-squared error

$$\epsilon = \mathbb{E}_{\pi_\theta}\left[\left(Q^{\pi_\theta}(s,a) - Q_w(s,a)\right)^2\right]$$

Then the policy gradient is exact

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$

Reducing Variance Using a Baseline

- ✓ We subtract a baseline function $B(s)$ from the policy gradient
- ✓ Can reduce variance without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) B(s)] = \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(s,a) B(s)$$

$$= \sum_{s \in \mathcal{S}} d^{\pi_\theta}(s) B(s) \nabla_\theta \sum_{a \in \mathcal{A}} \pi_\theta(s,a) = 0$$

- ✓ A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- ✓ So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

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

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Estimating the Advantage Function (I)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$
- Using two function approximators and two parameter vectors

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

- And updating both value functions by, e.g., TD learning

Estimating the Advantage Function (II)

- ✓Given true value function $V^{\pi_\theta}(s)$ the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

is an unbiased estimate of the advantage function

$$\mathbb{E}[\delta^{\pi_\theta}|s,a] = \mathbb{E}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s)$$
$$= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) = A^{\pi_\theta}(s,a)$$

- ✓We can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\delta^{\pi_\theta}]$$

- ✓In practice need to approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- ✓Only one set of critic parameters v is needed

Critics at Different Time-Scales

Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales

✓MC - Target is return return v_t

$$\Delta\theta = \alpha\left(v_t - V_\theta(s)\right)\phi(s)$$

✓TD(0) - Target is the TD target $r + \gamma V(s')$

$$\Delta\theta = \alpha\left(r + \gamma V(s') - V_\theta(s)\right)\phi(s)$$

✓Forward TD(λ) - Target is the λ-return

$$\Delta\theta = \alpha\left(v_t^\lambda - V_\theta(s)\right)\phi(s)$$

✓Backward TD(λ) - Equivalent target with eligibility traces

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

$$E_t = \lambda\gamma E_{t-1} + \phi(s_t)$$

$$\Delta\theta = \alpha\delta_t E_t$$

Actors at Different Time-Scales

- ✓ The policy gradient can also be estimated at many time-scales

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

- ✓ Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha\big(v_t - V_v(s_t)\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- ✓ Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha\big(r + \gamma V_v(s_{t+1}) - V_v(s_t)\big)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Policy Gradient with Eligibility Traces

- As with forward-view TD(λ), we can mix over time-scales

$$\Delta\theta = \alpha \left(v_t^{\lambda} - V_v(s_t) \right) \nabla_\theta \log \pi_\theta(s_t, a_t)$$

where $v_t^{\lambda} - V_v(s_t)$ is a biased estimate of advantage function

- Eligibility traces can be used as in backward-view TD(λ)
  - By equivalence with TD(λ), substituting $\phi(s) = \nabla_\theta \log \pi_\theta(s, a)$

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
$$E_t = \lambda\gamma E_{t-1} + \nabla_\theta \log \pi_\theta(s_t, a_t)$$
$$\Delta\theta = \alpha \delta_t E_t$$

- Again, backward can be applied online to incomplete sequences

Actor-Critic – Wrap-Up

- ✓ Actor-critic algorithms
  - ✓ Reduce variance of policy gradient
- ✓ Policy evaluation
  - ✓ Fitting value function to policy
- ✓ Advantage function
  - ✓ Reduce variance
  - ✓ Update both value functions or just state-value
- ✓ Can effectively use n-step and eligibility trace

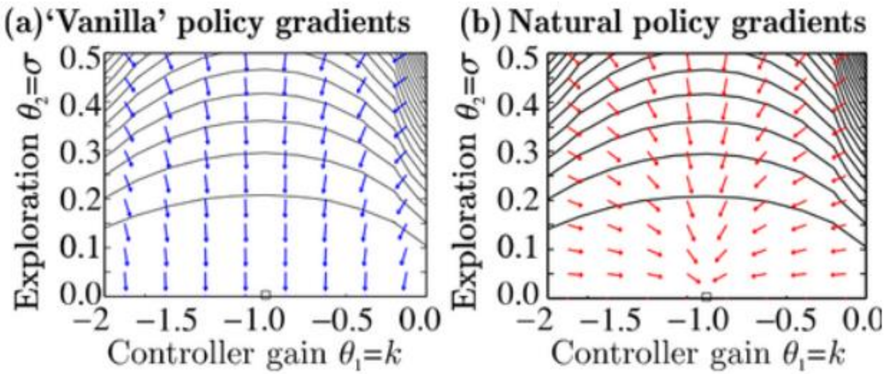
Natural Policy Gradient

Alternative Policy Gradient Directions

- ✓ Gradient ascent algorithms can follow any ascent direction
- ✓ A good ascent direction can significantly speed convergence
- ✓ Also, a policy can often be reparametrised without changing action probabilities
- ✓ For example, increasing score of all actions in a softmax policy
- ✓ The vanilla gradient is sensitive to these reparametrisations

In practice, since changing some parameters can affect probabilities (i.e. policy) more than others, is there a mean to rescale the gradient to avoid this effect?

Covariant/Natural Policy Gradient



✓The covariant/natural policy gradient is parametrisation independent

✓It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount

$$\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

✓where G_θ is the Fisher information matrix

$$G_\theta = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \, \nabla_\theta \log \pi_\theta(s,a)^T]$$

Natural Actor-Critic

- ✓ Using compatible function approximation

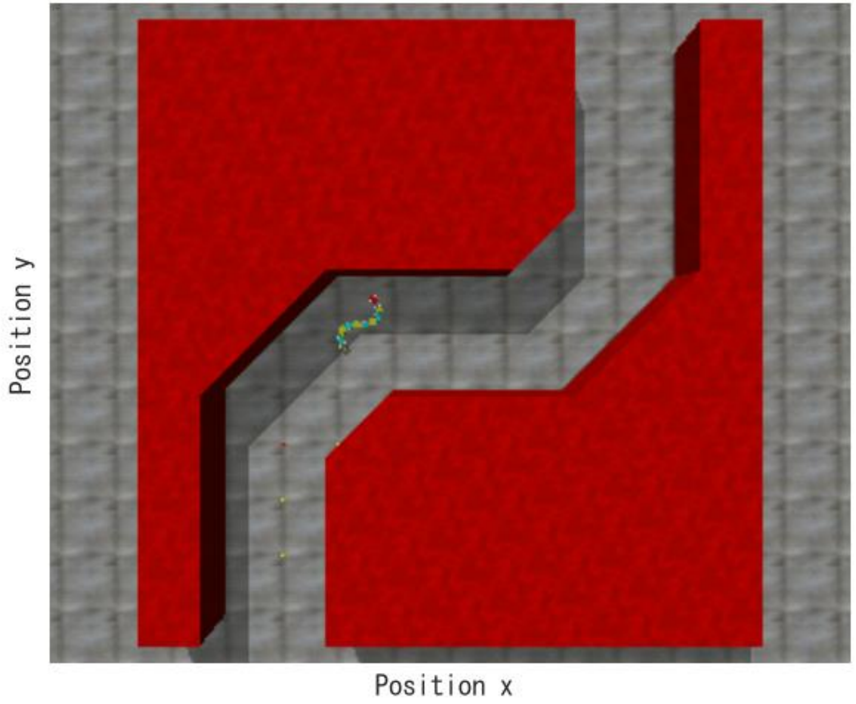
$$\nabla_w A_w(s,a) = \nabla_\theta \log \pi_\theta(s,a)$$

- ✓ The natural policy gradient simplifies

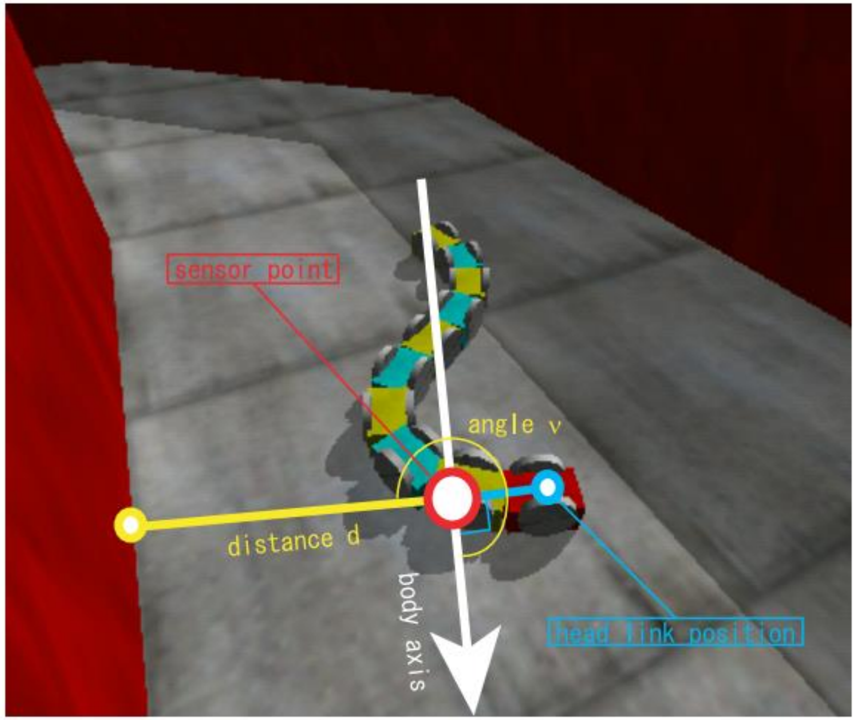
$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$
$$= \mathbb{E}_{\pi_\theta}\,[\nabla_\theta \log \pi_\theta(s,a)\, \nabla_\theta \log \pi_\theta(s,a)^T\, w] = G_\theta w$$
$$\nabla_\theta^{nat} \pi_\theta(s,a) = w$$

- ✓ Update actor parameters in direction of critic parameters

Natural Actor Critic - Snake Example (I)

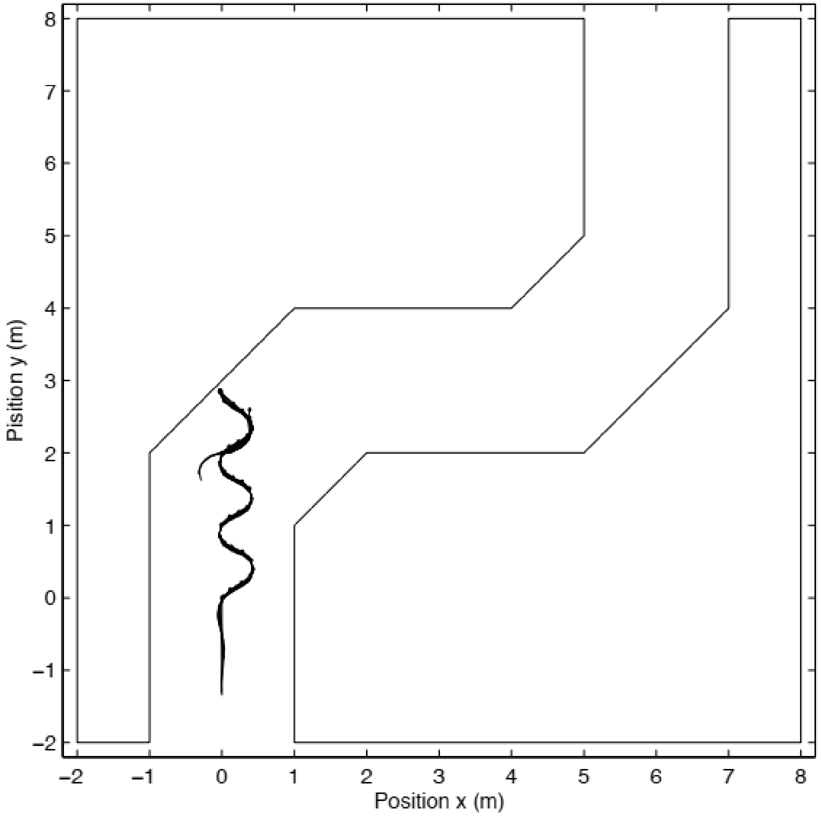


Crank course

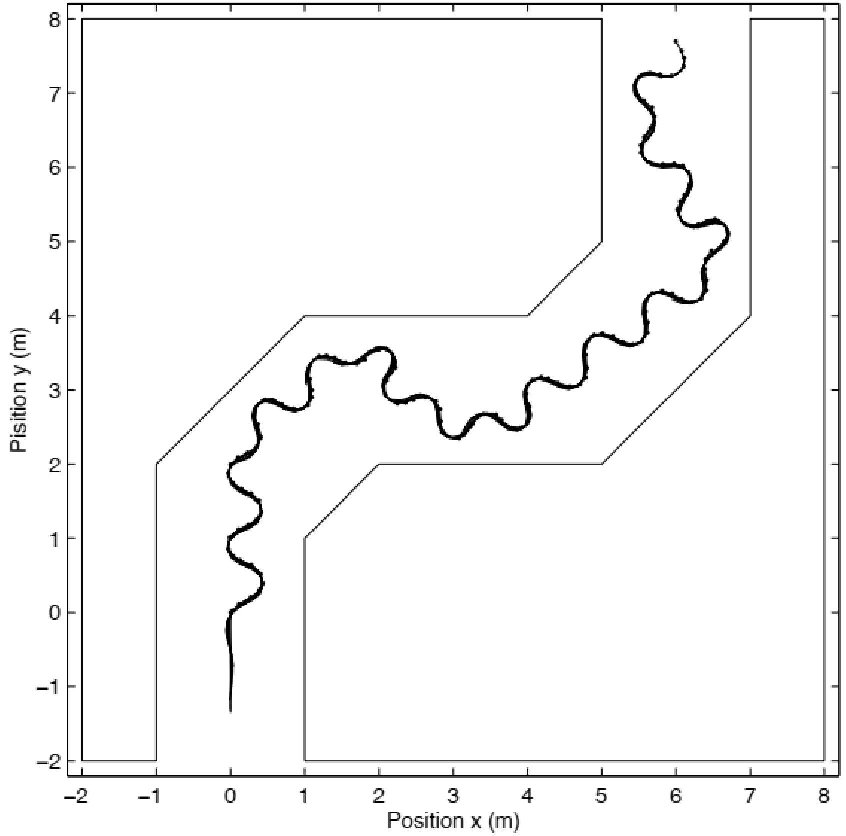


Sensor Setting

Natural Actor Critic - Snake Example (II)

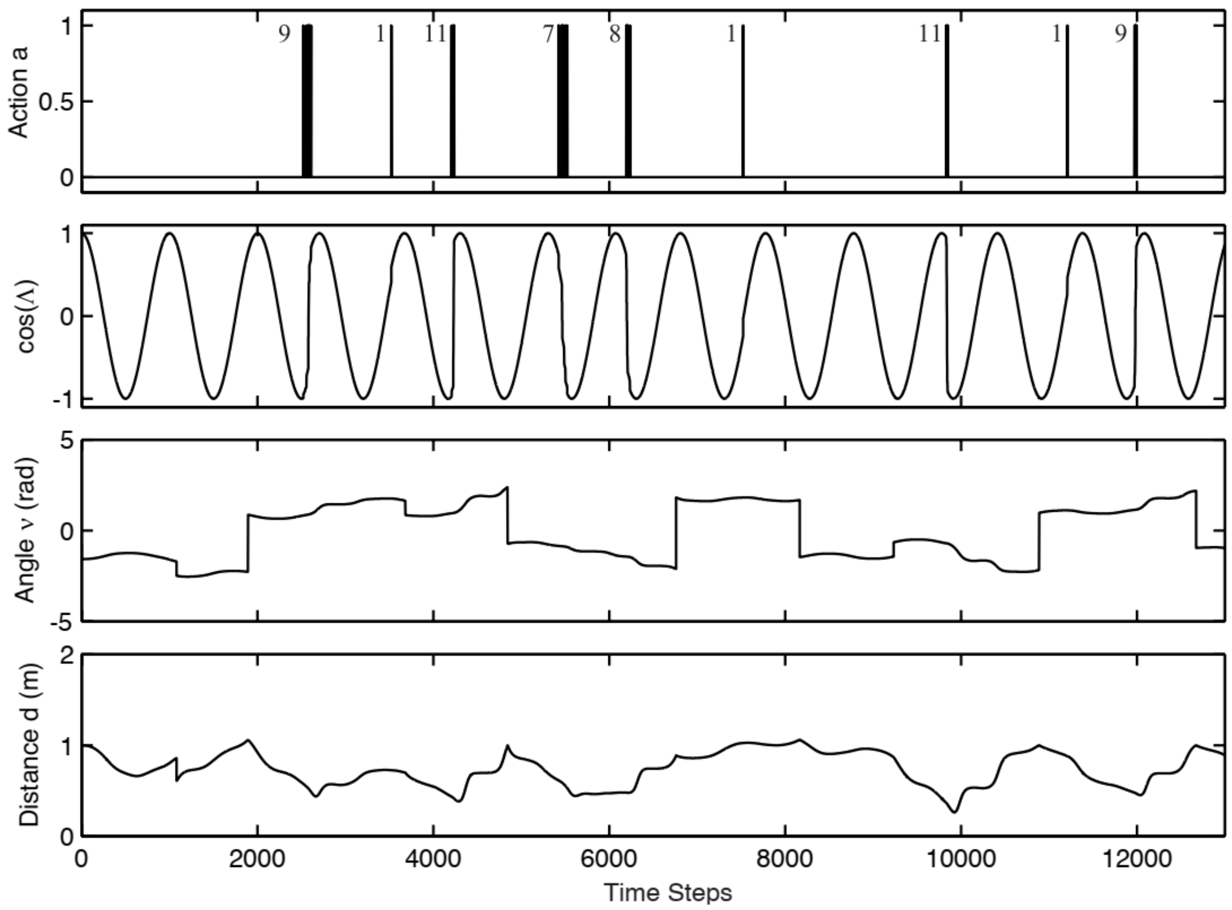


Before learning



After learning

Natural Actor Critic - Snake Example (III)



Summary of Policy Gradient Algorithms

- ✓The policy gradient has many equivalent forms

$\nabla_\theta J(\theta)$	$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right]$	REINFORCE
	$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right]$	Q Actor-Critic
	$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right]$	Advantage Actor-Critic
	$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right]$	TD Actor-Critic
	$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right]$	TD(λ) Actor-Critic
$G_\theta^{-1}\nabla_\theta J(\theta)$	$= w$	Natural Actor-Critic

- ✓Each leads a stochastic gradient ascent algorithm
- ✓Critic uses policy evaluation (e.g. MC or TD learning) to estimate $V^{\pi_\theta}(s)$, $Q^{\pi_\theta}(s,a)$ or $A^{\pi_\theta}(s,a)$

Natural Policy Gradient – Wrap-Up

- ✓Natural policy gradient

$$\theta' = \theta + \alpha G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

  - ✓Generally a good choice to stabilize policy gradient training
  - ✓Taylor expansion of KL-divergence between old and new policy
  - ✓Practical issue: requires efficient Fisher-vector products (non-trivial without computing the full matrix)

$$\alpha = \sqrt{\frac{2\epsilon}{\nabla_\theta \pi_\theta(s,a)^T G_\theta \nabla_\theta \pi_\theta(s,a)}}$$

- ✓Trust region policy optimization
  - ✓Generalizes natural policy gradient
  - ✓Optimizes expected advantage under new policy state distribution
  - ✓Uses importance sampling to align old-new policies in expectation (regularizes to stay close to old policy)

Deep Policy Networks

Deep Policy Networks

✓Represent policy by deep network with weights u

$$a = \pi_u(a|s) \text{ or } \pi_u(s)$$

✓Define objective function as total discounted reward

$$J(u) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 \dots |u]$$

✓Optimise objective end-to-end by stochastic gradient descent

✓Adjust policy parameters u to achieve more reward

Policy Gradients

How to make high-value actions more likely

- ✓The gradient of a stochastic policy $\pi(a|s,\boldsymbol{u})$ is given by

$$\nabla_u J(u) = \mathbb{E}_\pi[\nabla_u \log \pi_u(a|s)\, Q^\pi(s,a)]$$

- ✓The gradient of a deterministic policy a = $\pi(s)$ is given by

$$\nabla_u J(u) = \mathbb{E}_\pi[\nabla_a Q^\pi(s,a) \nabla_u a]$$

- ✓Assuming a continuous and Q differentiable

Actor-Critic Algorithm

- ✓ Estimate value function $Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$
- ✓ Update policy parameters u by stochastic gradient ascent

$$\frac{\partial J(u)}{\partial u} = \frac{\partial \log \pi_u(a|s)}{\partial u} Q_w(s,a)$$

or

$$\frac{\partial J(u)}{\partial u} = \frac{\partial Q_w(s,a)}{\partial a} \frac{\partial a}{\partial u}$$

Asynchronous Advantage Actor-Critic (A3C)

- ✓ Estimate state-value function

$$V_v(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 \ldots | s]$$

- ✓ Q-value estimated by an n-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_v(s_{t+n})$$

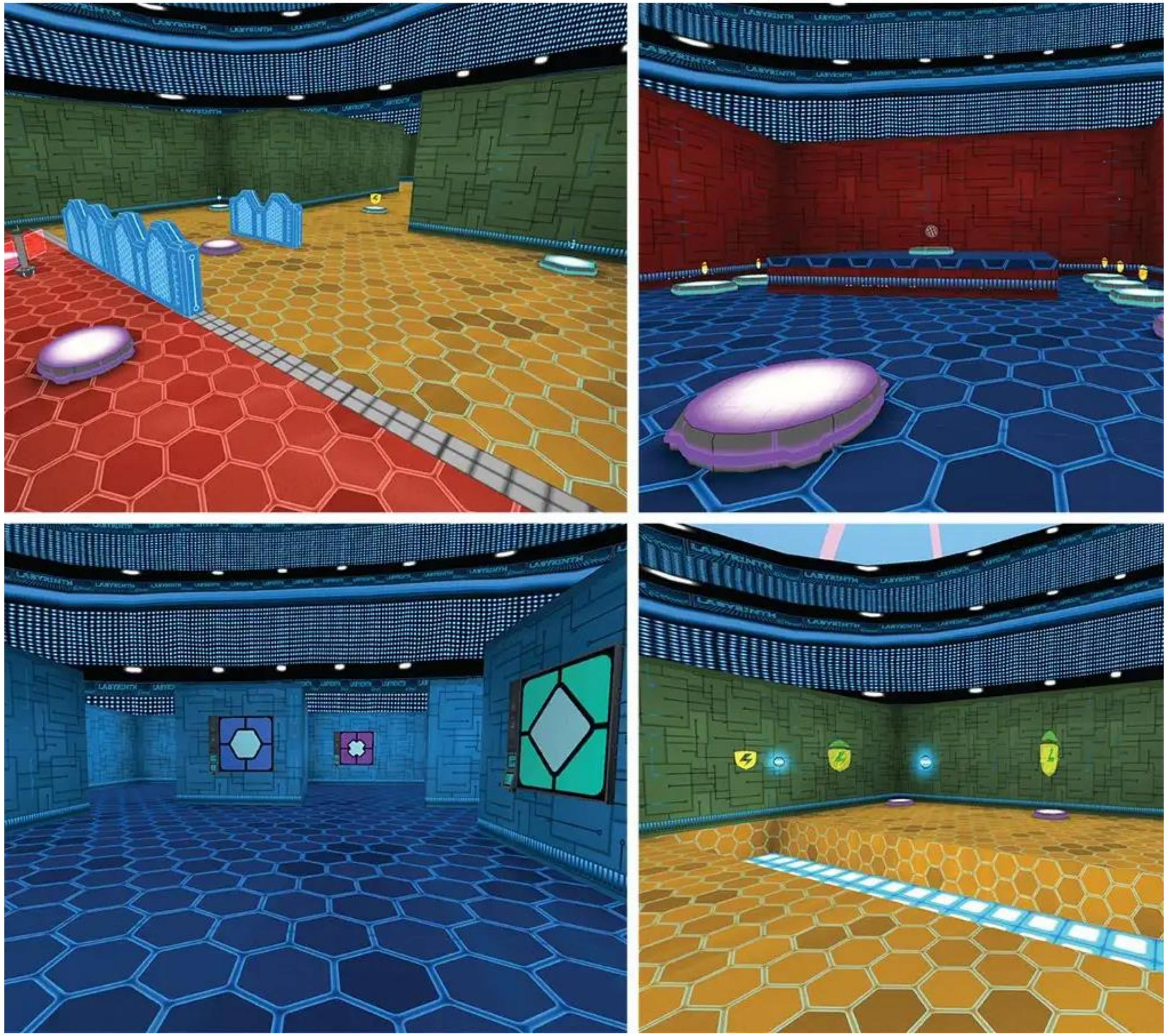
- ✓ Actor is updated towards target

$$\frac{\partial J(u)}{\partial u} = \frac{\partial \log \pi_u(a_t | s_t)}{\partial u} (q_t - V_v(s_t))$$

- ✓ Critic is updated to minimise Mean Squared Error w.r.t. target

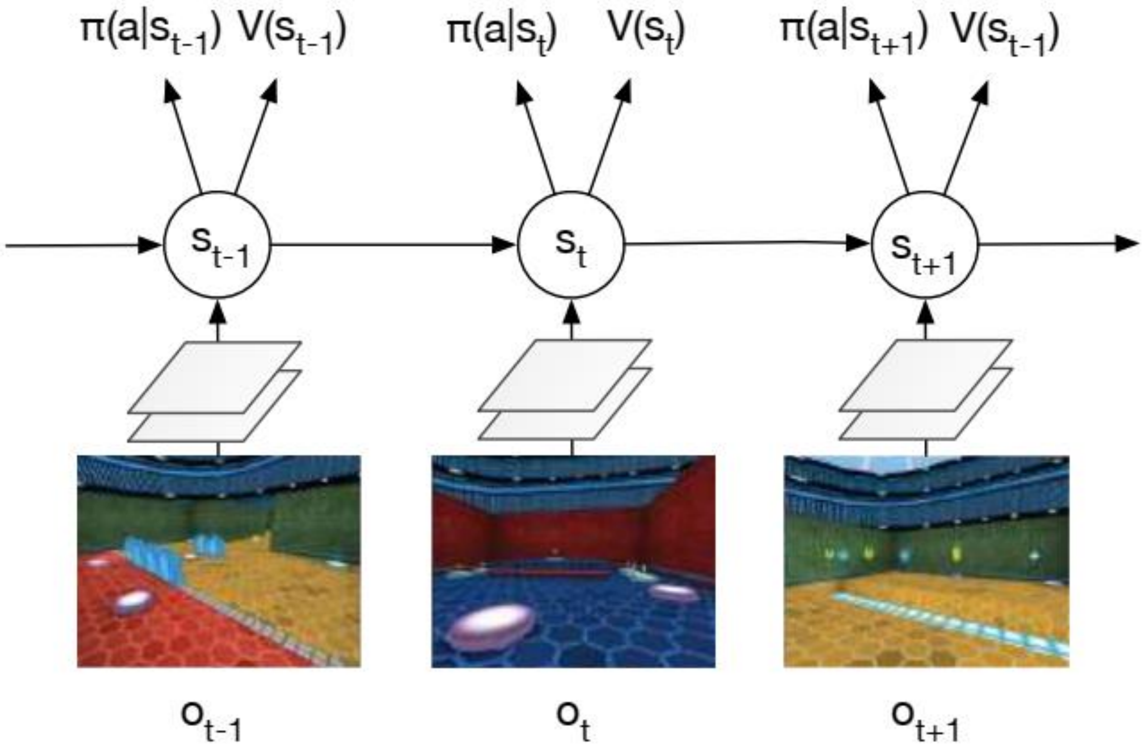
$$J(v) = \left(q_t - V_v(s_t)\right)^2$$

A3C vs DQN => 4*x* mean Atari score



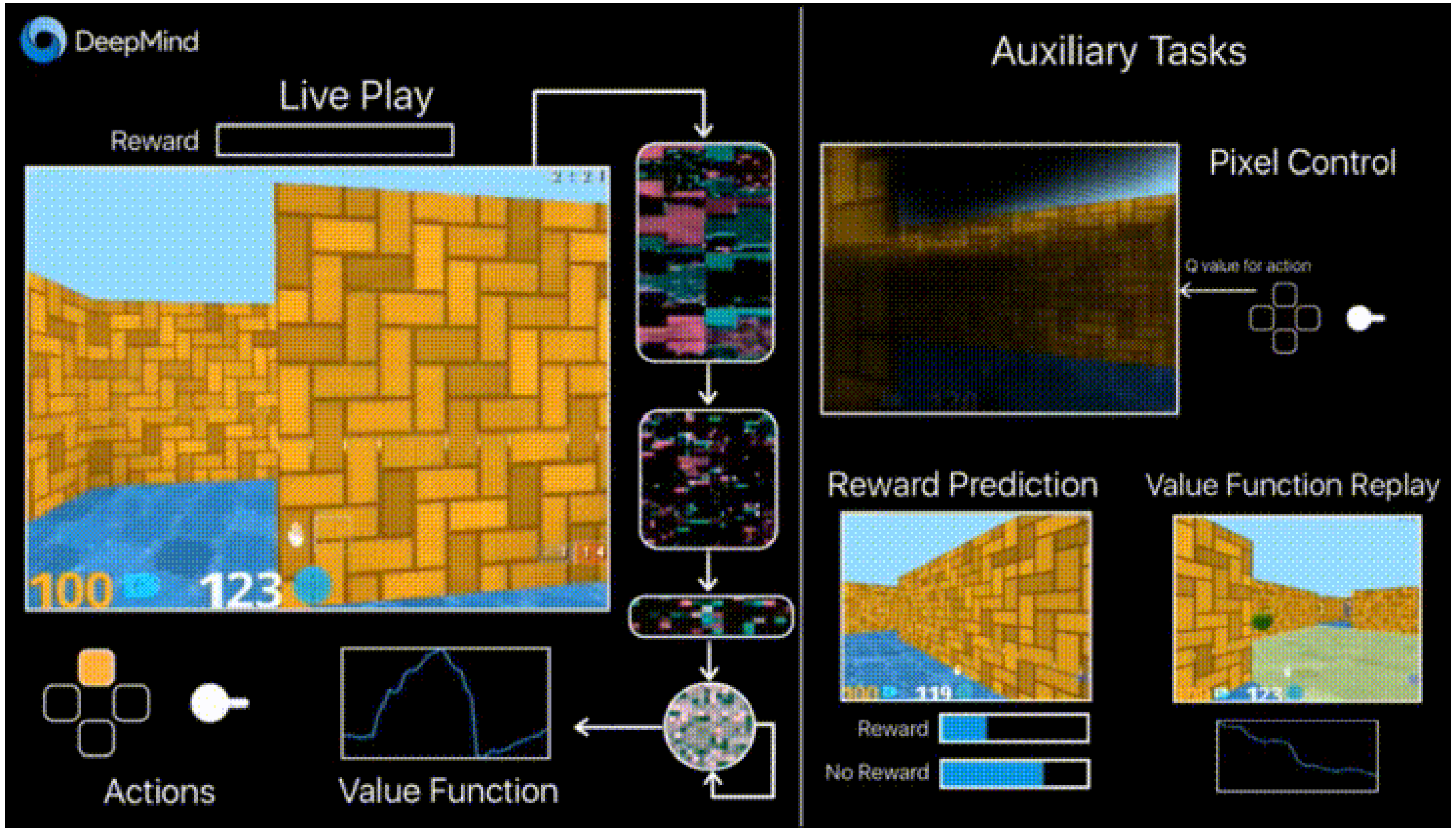
Labyrinth

A3C in Labyrinth



- ✓End-to-end learning of softmax policy $\pi(a|s_t)$ from pixels
- ✓Observations o_t are raw pixels from current frame
- ✓State $s_t = f(o_1, \dots, o_t)$ is obtained through a Long Short Term Memory
- ✓Outputs both value function $V(s)$ and a softmax over actions $\pi(a|s_t)$
- ✓The task is to collect fruits (+1 reward) and escape (+10 reward)

A3C-Labyrinth Demo



Demo
www.youtube.com/watch?v=nMR5mjCFZCw&feature=youtu.be

Source code (unofficial)
https://github.com/cgnicholls/reinforcement-learning/tree/master/a3c

https://github.com/miyosuda/async_deep_reinforce

Deep Reinforcement Learning with Continuous Actions

- ✓ How can we deal with high-dimensional continuous action spaces?
- ✓ Cannot easily compute $\max_a Q(s, a)$
  - ✓ Actor-critic algorithms learn without max
- ✓ Q-values are differentiable with respect to a
  - ✓ Deterministic policy gradients exploit knowledge of $\frac{\partial Q(s,a)}{\partial a}$

Deep Deterministic Policy Gradients (DPG)

DPG is the continuous analogue of Deep Q Networks (DQN)

✓ Experience replay - Build dataset from agent's experience

✓ Fixed target - To deal with non-stationarity, use fixed target networks $u^-,\ w^-$

✓ Critic estimates value of current policy by DQN

$$J(w) = \left(r + \gamma Q_{w^-}\left(s', \pi_{u^-}(s')\right) - Q_w(s,a)\right)^2$$

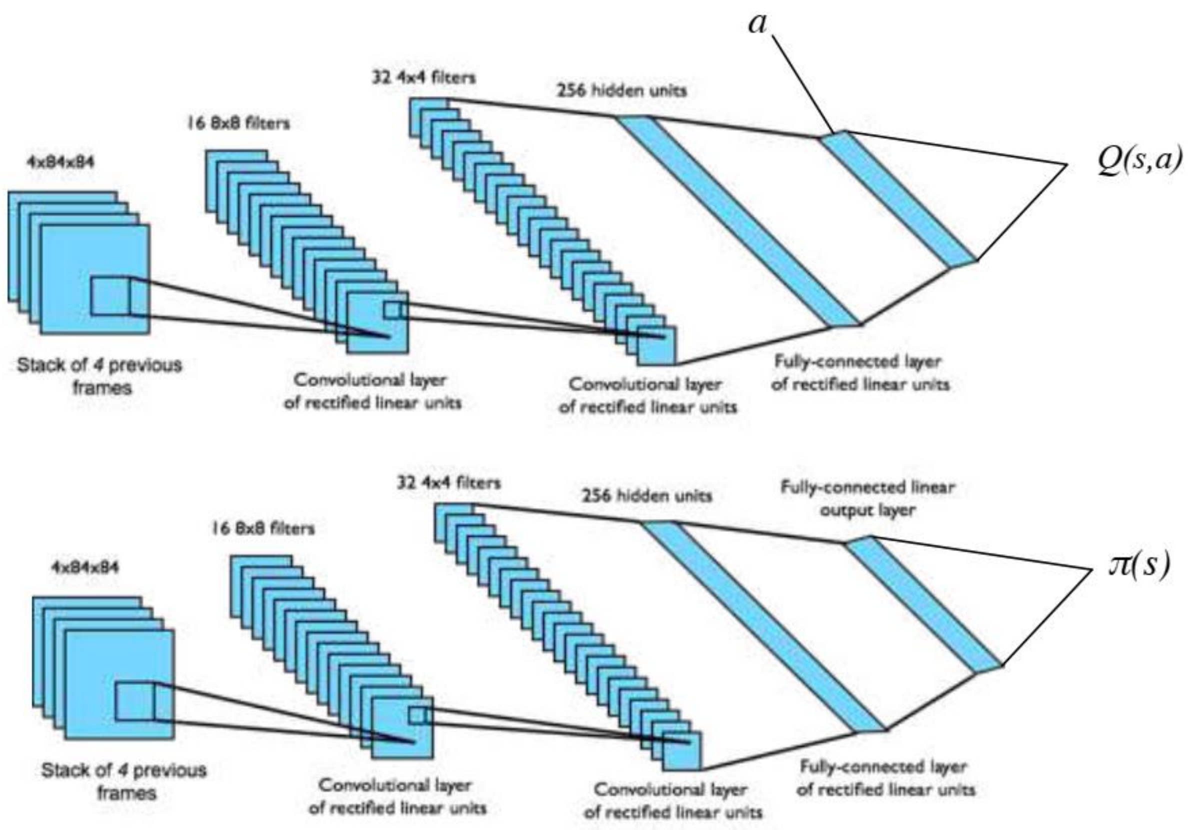
✓ Actor updates policy in direction that improves Q

$$\frac{\partial J(u)}{\partial u} = \frac{\partial Q_w(s,a)}{\partial a} \frac{\partial a}{\partial u}$$

✓ A.K.A. critic provides loss function for actor

DPG in Simulated Physics

- ✓Physics domains are simulated in MuJoCo (but in OpenAI Gym as well)
- ✓End-to-end learning of control policy from raw pixels s
- ✓Input state s is (as in DQN) a stack of raw pixels from last 4 frames
- ✓Two separate CNNs are used for Q and π
- ✓Policy π is adjusted in direction that most improves Q



DPG in Simulated Physics Demo

Wrap-up

Take Home Messages

- ✓ Policy gradient
  - ✓ Effective in complex action spaces, can learn stochastic policies and have better convergence
  - ✓ Typically high variance and local optima
- ✓ REINFORCE - A formalization of trial and error
- ✓ Actor-critic - Fitting value function (critic) to a learned policy (actor) to reduce variance
- ✓ Natural gradient - Stabilize policy training with localized updates in distribution space
- ✓ Deterministic Policy Gradients for continuous actions

Next Lecture

- ✓ Integrating Learning and Planning
- ✓ Model-based reinforcement learning
- ✓ Real vs simulated experience
- ✓ Monte-Carlo Tree Search
- ✓ Alpha-GO (if it fits in the time)