Problems of RL

Curse of Dimensionality
In real world problems ist difficult/impossible to define discrete state-action spaces.

(Temporal) Credit Assignment Problem
RL cannot handle large state action spaces as the reward gets too much dilited along the way.

Partial Observability Problem
In a real-world scenario an RL-agent will often not know exactly in what state it will end up after performing an action. Furthermore states must be history independent.

State-Action Space Tiling
Deciding about the actual state- and action-space tiling is difficult as it is often critical for the convergence of RL-methods. Alternatively one could employ a continuous version of RL, but these methods are equally difficult to handle.

Non-Stationary Environments
As for other learning methods, RL will only work quasi stationary environments.

Large Scale RL

Large problems

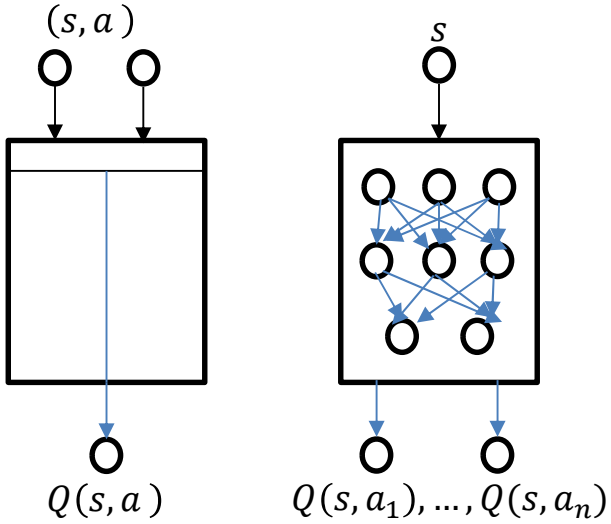
- Backgammon: 10^{20}stati
- Go: 10^{170}stati
- Helicopter: continuous

Chapter 9, Barto
Section 9.1, 9.2, 9.3, 9.4, 9.6

Efficient approximation of value functions and policies

Estimates with a value function approximation:

- $\hat{v}(s, w) \approx v_\pi(s)$
- $\hat{q}(s, a, w) \approx q_\pi(s, a)$
- $\dim(w) \ll |S|$



Differentiable function approximations:

- Linear combination of features
- Neural Networks
- …

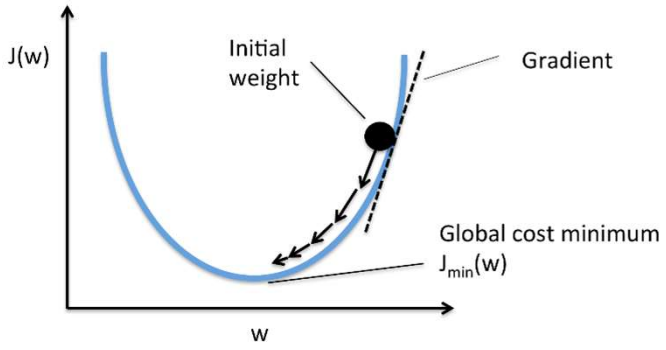
- Non-stationarity target f, not indep. identically distributed data, on-line, non-supervised learning
- More complex RL, many state affected by parameter changes
- but effective, may also work with partially observable problems

Gradient Descent

Let $J(w)$ be a differentiable function of parameter vector w

The gradient of $J(w)$ is

$$\nabla_{\mathrm{w}} J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \cdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$



- To find the local minimum of $J(w)$ adjust the parameters in the direction of the gradient vector

$$\mathrm{w}_{\mathrm{t+1}} = \mathrm{w}_{\mathrm{t}} - \Delta w \qquad \Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$

α positive step-size parameter

- Find vector that minimize the mean-squared error between $\hat{v}(s,w)$ and $v_\pi(s)$

$$\mathrm{J}(w) = E_\pi[\left(v_\pi(s) - \hat{v}(s,w)\right)^2]$$

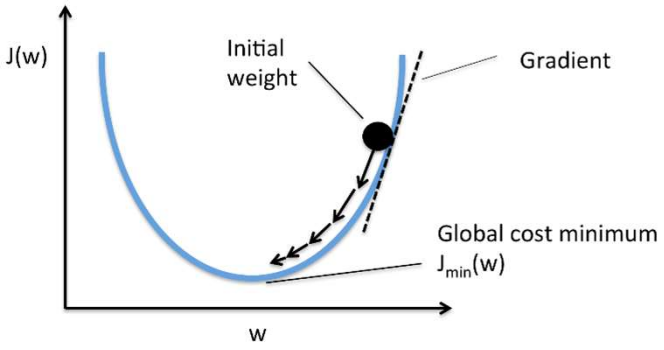
Supervised: oracle provides $v_\pi(s)$

Stochastic Gradient Descent

Let $J(w)$ be a differentiable function of parameter vector w

The gradient of $J(w)$ is

$$\nabla_{\mathrm{w}} J(w) = \begin{pmatrix} \frac{\partial J(w)}{\partial w_1} \\ \cdots \\ \frac{\partial J(w)}{\partial w_n} \end{pmatrix}$$



- Find vector that minimize the mean-squared error between $\hat{v}(s,w)$ and $v_\pi(s)$

$$\mathrm{J}(w) = E_\pi[\big(v_\pi(s) - \hat{v}(s,w)\big)^2]$$

Supervised: oracle provides $v_\pi(s)$

- Gradient descent:

$$\Delta w = -\frac{1}{2}\alpha\nabla_w J(w) = \alpha E_\pi[\big(v_\pi(s) - \hat{v}(s,w)\big)\nabla_{\mathrm{w}}\,\hat{v}(s,w)]$$

- Stochastic gradient descent (SGD) samples the gradient (update on samples)

$$\Delta w = \alpha\big(v_\pi(s) - \hat{v}(s,w)\big)\nabla_{\mathrm{w}}\,\hat{v}(s,w) \qquad \mathrm{w}_{\mathrm{t+1}} = \mathrm{w}_{\mathrm{t}} - \Delta w$$

Linear Case: Feature Vectors

We can represent the state as a feature vector

$$\boldsymbol{x}(S) = \begin{pmatrix} x_1(S) \\ \dots \\ x_n(S) \end{pmatrix}$$

e.g., distances from landmarks

- Value function as a linear combination of features

$$\hat{v}(s, w) = \boldsymbol{x}(S)^T w = \sum_j x_j(S) w_j$$

- Objective function is quadratic in parameters

$$\mathrm{J}(w) = E_\pi[(v_\pi(s) - \boldsymbol{x}(S)^T w)^2]$$

Convex

- Gradient descent converges on global optimum
- Update rule is simple

$$\nabla_w \hat{v}(s, w) = \boldsymbol{x}(S)$$

Gradient is the feature vector

$$\Delta w = \alpha\big(v_\pi(s) - \hat{v}(s, w)\big)\boldsymbol{x}(S)$$

update = step-size x prediction error x feature value

Table Lookup Features

Table lookup is a special case of linear value function approximation

- Table lookup features

$$\boldsymbol{x}^{table}(S) = \begin{pmatrix} 1\,(S = s_1) \\ \dots \\ 1\,(S = s_n) \end{pmatrix}$$

- Parameter vector gives value of each individual state

$$\hat{v}(s, w) = \boldsymbol{x}^{table}(S)^T w = \sum_j x_j(S) w_j$$

These are the values in the table

Incremental Prediction

The value function $v_\pi(s)$ is not available

Unsupervised learning

- Substitute $v_\pi(s)$ with an estimate
- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

- For TD(0) learning

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

- In MC is like generating “training data”, converges to the local optimum
- In TD is analogous and converges (close) to the local optimum

Incremental Prediction

The value function $v_\pi(s)$ is not available

- Substitute $v_\pi(s)$ with an estimate
- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\, \hat{v}(S_t, w)$$

Similar to supervised learning: each episode collects training data G_t for S_t

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathbf{w}$ as appropriate (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat forever:
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  For $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t, \mathbf{w})\big]\nabla \hat{v}(S_t, \mathbf{w})$

Converges to local optimum

Incremental Prediction

The value function $v_\pi(s)$ is not available

- Substitute $v_\pi(s)$ with an estimate

Only an approximated prediction of $v_\pi(s)$ is available (biased sample of $v_\pi(s)$)

- For TD(0) learning (semi-gradient method)

Training data:
$(S_1, R_2 + \gamma\hat{v}(S_2, w))$, $(S_2, R_3 + \gamma\hat{v}(S_3, w))$, …

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{v}(S_{t+1}, w) - \hat{v}(S_t, w)\big)\nabla_{\mathrm{w}}\,\hat{v}(S_t, w)$$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

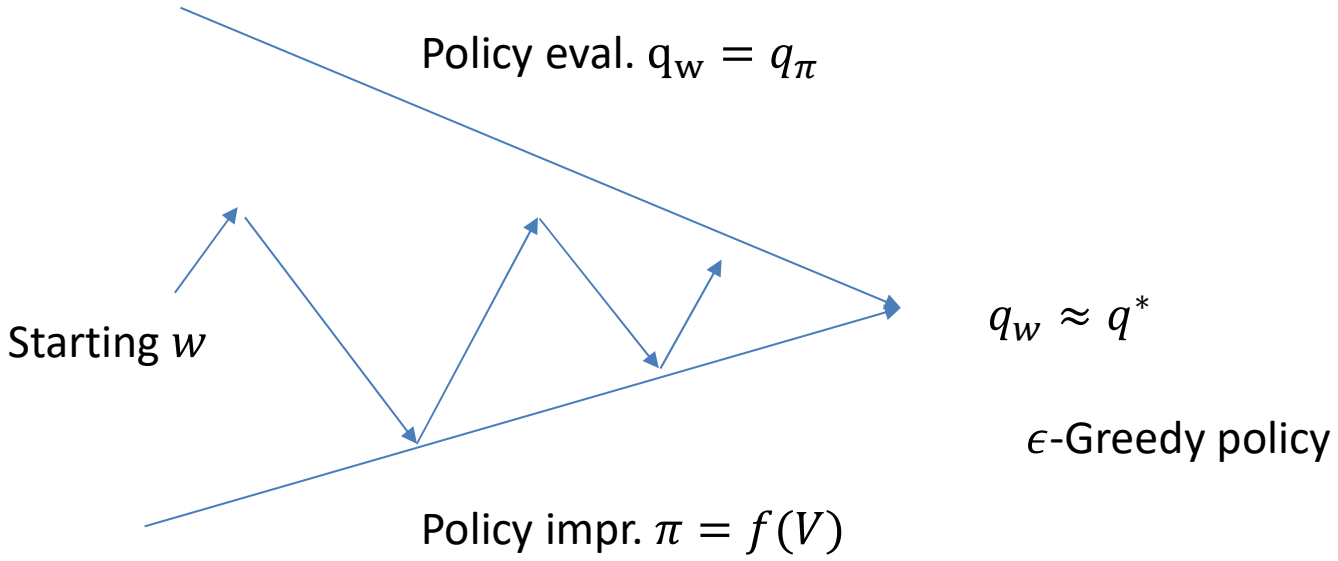
Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$

Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

Control with Function Approximation

Chapter 10, Sutton Barto
Section 10.1, 10.2

- Policy Evaluation
  - Approximate $\hat{q}(s, a, w) \approx q_\pi(s, a)$
- Policy Improvement
  - ϵ-greedy



Action-Value Function Approx

Approximate the action-value function

$$\hat{q}(s, a, w) \approx q_\pi(s, a)$$

- Minimize the mean-squared error

$$\mathrm{J}(w) = E_\pi[\big(q_\pi(s, a) - \hat{q}(s, a, w)\big)^2]$$

- Stochastic Gradient Descent samples the gradient

$$\Delta w = \alpha\big(q_\pi(s, a) - \hat{q}(s, a, w)\big)\nabla_{\mathrm{w}}\hat{q}(s, a, w)$$

Action-Value Feature Vectors

We can represent the state as a feature vector

$$\boldsymbol{x}(S,A) = \begin{pmatrix} x_1(S,A) \\ \dots \\ x_n(S,A) \end{pmatrix}$$

- Value function as a linear combination of features

$$\hat{q}(s,a,w) = \boldsymbol{x}(S,A)^T w = \sum_j x_j(S,A) w_j$$

- Objective function is quadratic in parameters

$$\mathrm{J}(w) = E_\pi[(q_\pi(s,a) - X(S,A)^T w)^2]$$

- Gradient descent converges on global optimum
- Update rule is simple

$$\nabla_w \hat{q}(S,A,w) = \boldsymbol{x}(S,A)$$

$$\Delta w = \alpha\big(q_\pi(s,a) - \hat{q}(s,a,w)\big)\boldsymbol{x}(S,A)$$

Incremental Control

- Substitute $q_\pi(s)$ with an estimate

- For MC learning

$$\Delta w = \alpha\big(G_t - \hat{q}(S_t, A_t, w)\big)\nabla_{\mathrm{w}}\, \hat{q}(S_t, A_t, w)$$

- For TD(0) learning

$$\Delta w = \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w)\big)\nabla_{\mathrm{w}}\, \hat{q}(S_t, A_t, w)$$

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

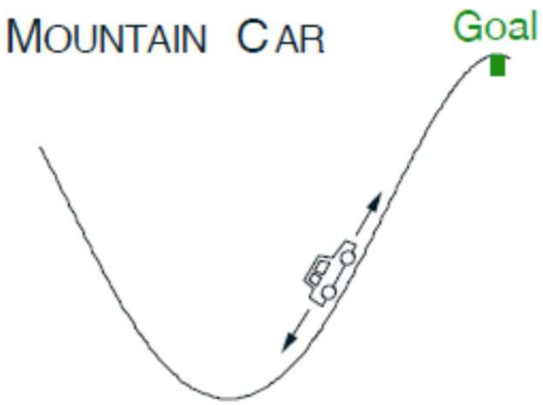
Input: a differentiable function $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$

Initialize value-function weights $\mathrm{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathrm{w} = \mathbf{0}$)
Repeat (for each episode):
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    If S' is terminal:
      $\mathrm{w} \leftarrow \mathrm{w} + \alpha\big[R - \hat{q}(S, A, \mathrm{w})\big]\nabla\hat{q}(S, A, \mathrm{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathrm{w})$ (e.g., ε-greedy)
    $\mathrm{w} \leftarrow \mathrm{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathrm{w}) - \hat{q}(S, A, \mathrm{w})\big]\nabla\hat{q}(S, A, \mathrm{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Mountain Car Task

Underpowered car, gravity is stronger than the car's engine, the only solution is to first move away from the goal and up the opposite slope on the left.

Reward -1 on all time step until the goal is not reached



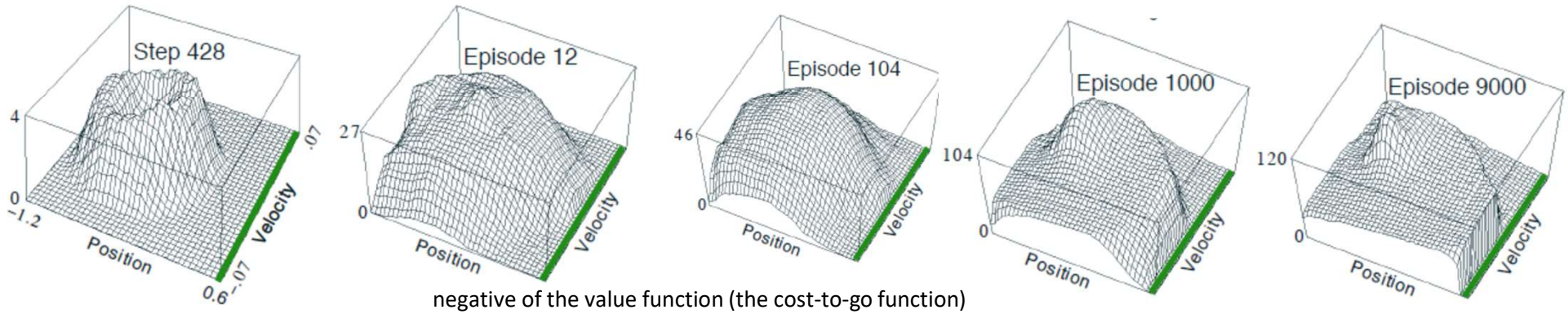
Three actions: *full throttle forward, reverse, zero*
Simplified physics:

$$x_{t+1} \doteq bound\big[x_t + \dot{x}_{t+1}\big]$$

$$\dot{x}_{t+1} \doteq bound\big[\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)\big]$$

$$-1.2 \le x_{t+1} \le 0.5 \text{ and } -0.07 \le \dot{x}_{t+1} \le 0.07$$

Velocity reset to zero when the left bound is reached
Each episode from a random position and zero velocity
Linear approximation $\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_i w_i \cdot x_i(s,a),$

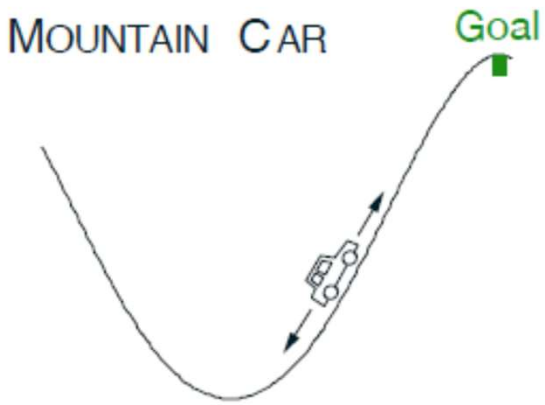


negative of the value function (the cost-to-go function)

Mountain Car Task

Underpowered car, gravity is stronger than the car's engine, the only solution is to first move away from the goal and up the opposite slope on the left.

Reward -1 on all time step until the goal is not reached



Three actions: *full throttle forward, reverse, zero*
Simplified physics:

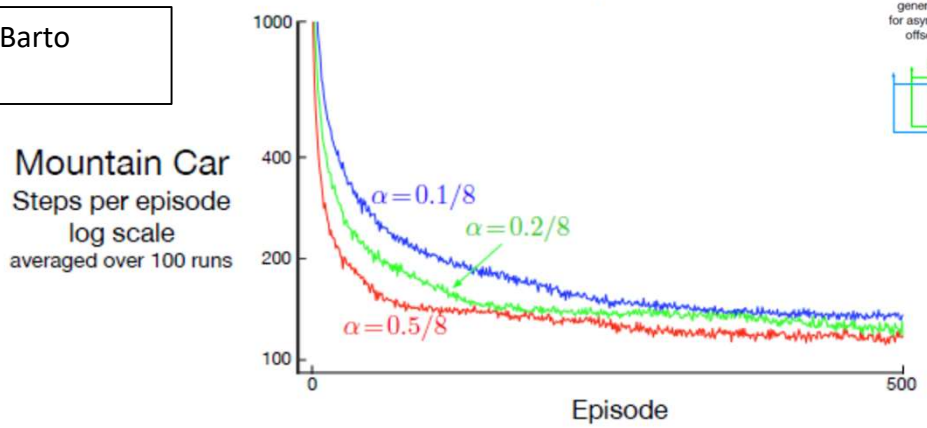
$$x_{t+1} \doteq bound\big[x_t + \dot{x}_{t+1}\big]$$

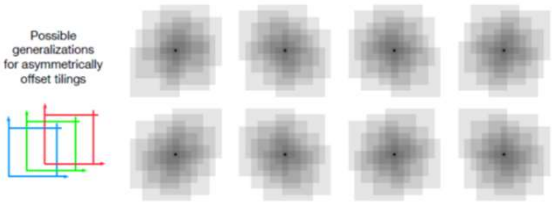
$$\dot{x}_{t+1} \doteq bound\big[\dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)\big]$$

$$-1.2 \le x_{t+1} \le 0.5 \text{ and } -0.07 \le \dot{x}_{t+1} \le 0.07$$

Velocity reset to zero when the left bound is reached
Each episode from a random position and zero velocity
Linear approximation $\hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^\top \mathbf{x}(s,a) = \sum_i w_i \cdot x_i(s,a),$

Chapter 10, Sutton Barto
Section 10.2





tile-coding function approximation
and "-greedy action selection

Chapter 9, Sutton Barto
Section 9.5

Convergence of Evaluation

- On-Policy
  - MC learning
    - Converge with table lookup, linear/non-linear approx
  - TD(0)\TD(λ) learning
    - Converge with table lookup and linear approx.
  - Gradient TD (follows true gradient)
    - Converge with table lookup, linear/non-linear approx
- Off-Policy
  - MC learning
    - Converge with Table, Linear/Non-linear approx
  - TD(0)\TD(λ) learning
    - Converge with table lookup
  - Gradient TD (follows true gradient)
    - Converge with table lookup, linear/non-linear approx

Convergence of Control

- Control Algorithm
  - MC learning
    - Converge with table lookup, (linear approx)
  - SARSA learning
    - Converge with table lookup, (linear approx.)
  - Q-learning
    - Converge with lookup
  - Gradient Q-learning
    - Converge with table lookup, linear approx

(chatters around the near-optimal function)

Batch RL

- Gradient descent is appealing
- It is not sample efficient
- Batch methods try to find the best fitting of the value function given the experience (“training data”)

Least Squares Prediction

- Function approximation $\hat{v}(s, w) \approx v_\pi(s)$
- Experience $D = \{\langle s_1, v_1^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$
- Parameters w for the best fitting value $\hat{v}(s, w)$
- Least squares algorithms to find the w minimizing the sum-squared error between $\hat{v}(s, w)$ and $v_\pi(s)$

$$\mathrm{LS}(w) = \sum_{t=1\ldots T} \left[\left(v_t^\pi - \hat{v}(s_t, w)\right)^2\right] = E_D\left[\left(v_t^\pi - \hat{v}(s_t, w)\right)^2\right]$$

Gradient Descent with Experience Replay

- Store experience $D = \{\langle s_1, v_1^\pi \rangle, \dots, \langle s_T, v_T^\pi \rangle\}$
- Repeat
  - Sample $\langle s , v^\pi \rangle$ from the experience D
  - Apply stochastic gradient descent update towards the target

$$\Delta w = \alpha\big(v^\pi - \hat{v}(s,w)\big)\nabla_w \hat{v}(s,w)$$

Like supervised learning

Converges to least squares solution

$$w^\pi = argmin_w LS(w)$$

Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target

- Take an action a_t according to an ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in memory D
- Sample mini-batch transitions (s, a, r, s') from D
- Compute Q-learning targets w.r.t. old fixed parameters w^-
- Optimize MSE between Q-network and Q-learning targets:

$$Loss_i(w_i) = E_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

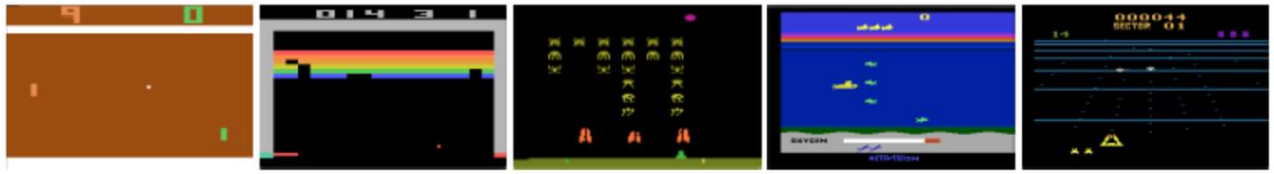
- Minimize the loss with stochastic gradient descent
- After N iteration copy the actual parameters in the target
- Repeat for M episodes

- Key points to stabilize the process:
  - Decouple the policy and the learning data (mini-batch)
  - Decouple the fixed (old and frozen) Q-target and the Q-network
  - After a while switch old and new parameters …

Experience Replay in DQN

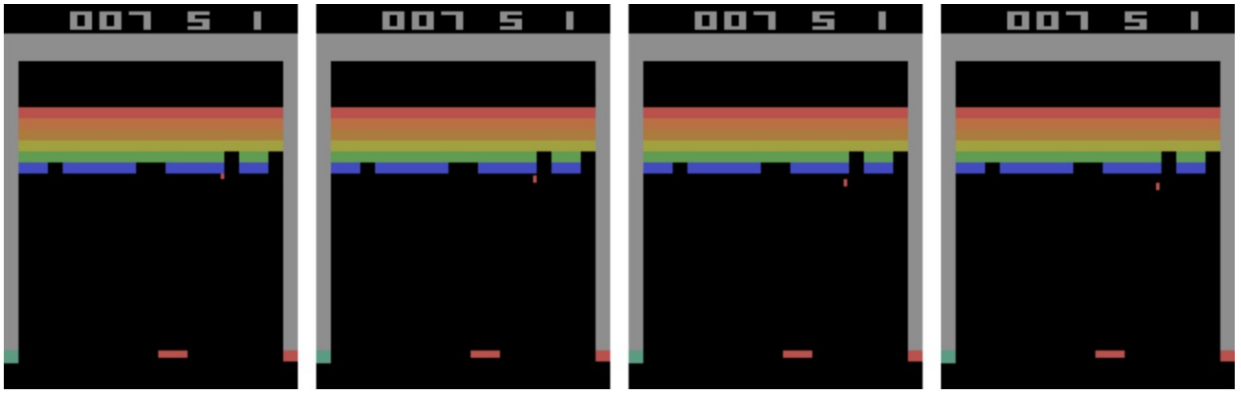
Playing Atari with Deep Reinforcement Learning [NIPS 2013]

Deep Q-Networks uses experience replay and fixed Q-target



Atari 2600 Playing:

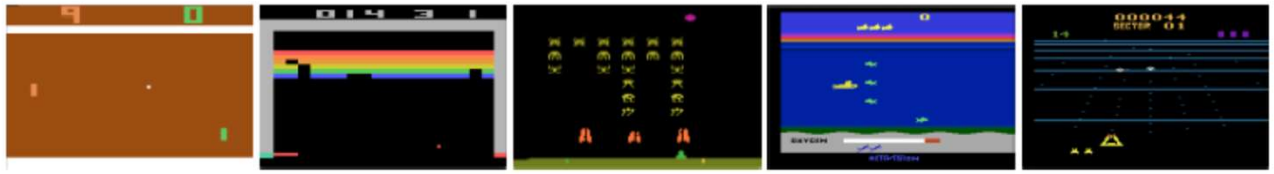
Input: 84 x 84 x 4 image produced by the preprocessing (gray-scale and down-sampling, crop), Atari frames are 210x160 pixel images, 128 color, 4 images



Position, direction, velocity, acceleration

Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target

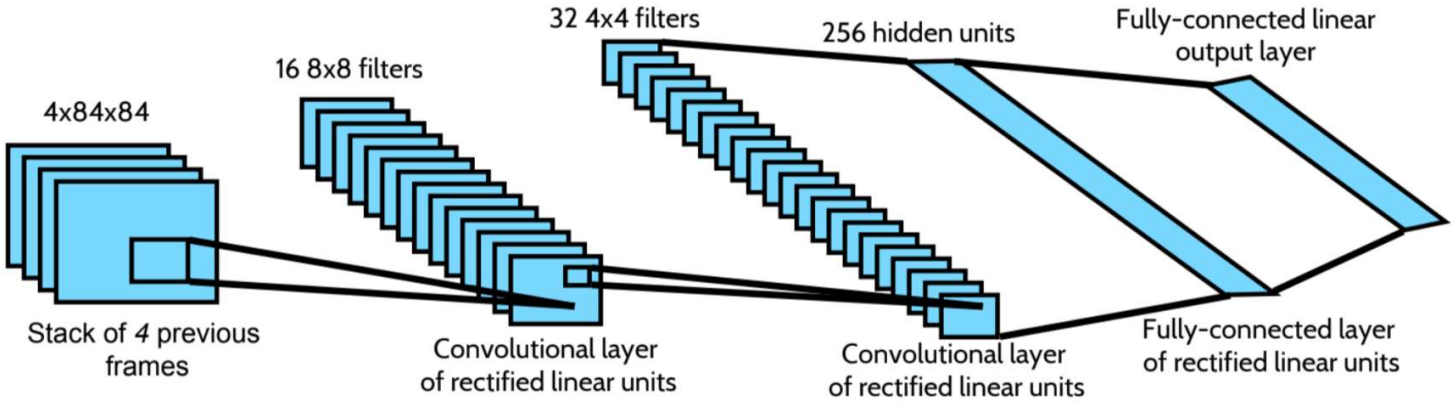


Atari 2600 Playing:

Hidden: 3 hidden conv. layers and a fully-connected hidden layer.

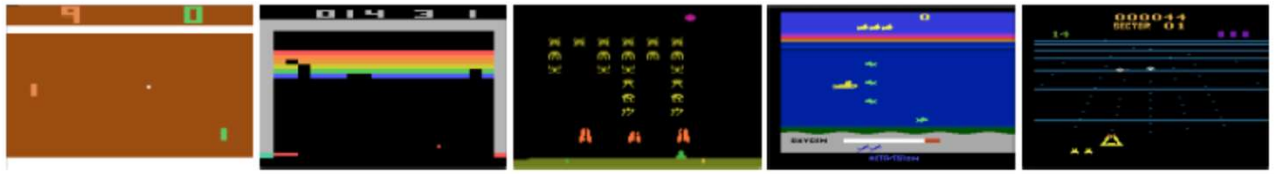
1. 16 8x8 filters, stride 4, rectifier
2. 32 4x4 filters, stride 2, rectifier
3. 256 hidden fully connected rectifier units

Output:
One for each possible action (4 to 18)



Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target



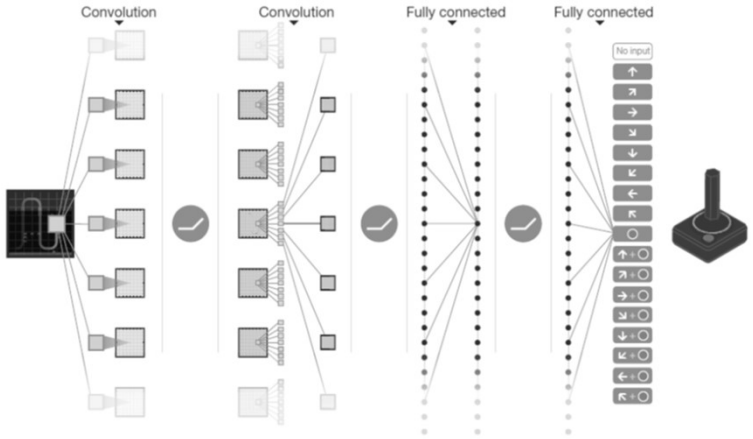
Atari 2600 Playing:

Hidden: 3 hidden conv. layers and a fully-connected hidden layer.

1. 16 8x8 filters, stride 4, rectifier
2. 32 4x4 filters, stride 2, rectifier
3. 256 hidden fully connected rectifier units

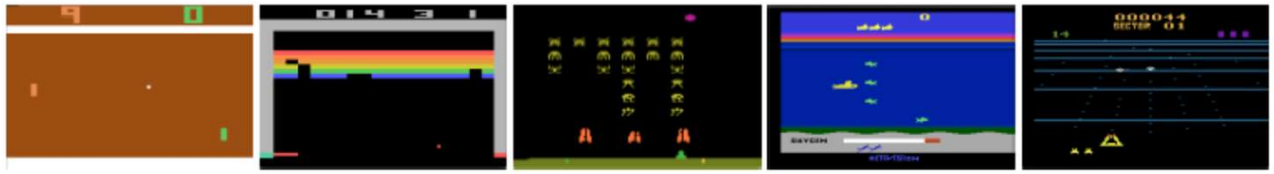
Output:

One for each possible action (4 to 18)



Experience Replay in DQN

Deep Q-Networks uses experience replay and fixed Q-target



Atari 2600 Playing:

Hidden: 3 hidden conv. layers and a fully-connected hidden layer.

1. 16 8x8 filters, stride 4, rectifier
2. 32 4x4 filters, stride 2, rectifier
3. 256 hidden fully connected rectifier units

Output:
One for each possible action (4 to 18)

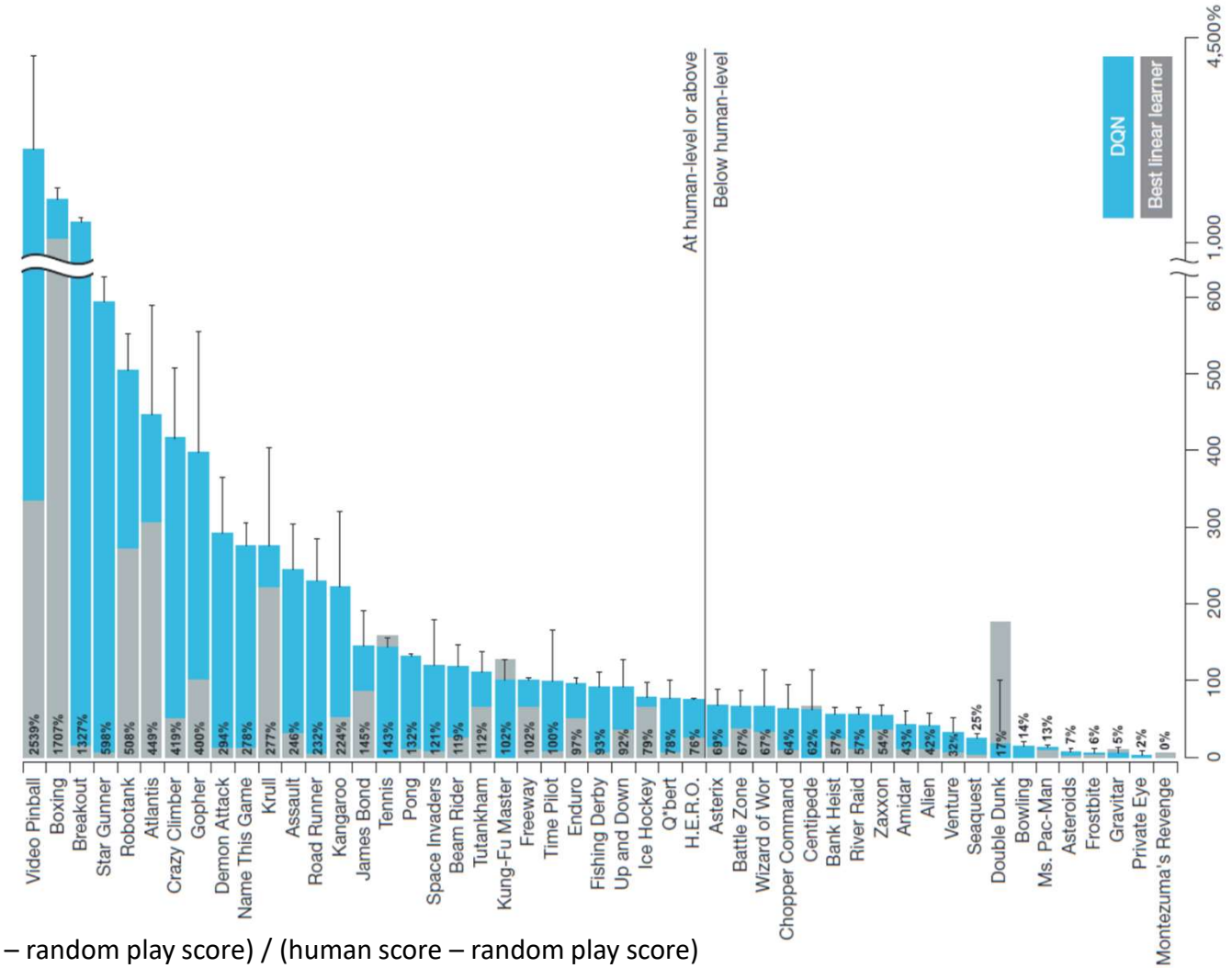
Sarsa algorithm uses linear policies on hand-engineered feature sets. Contingency uses a similar approach with augmented the feature sets learned representation

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

Experience Replay in DQN

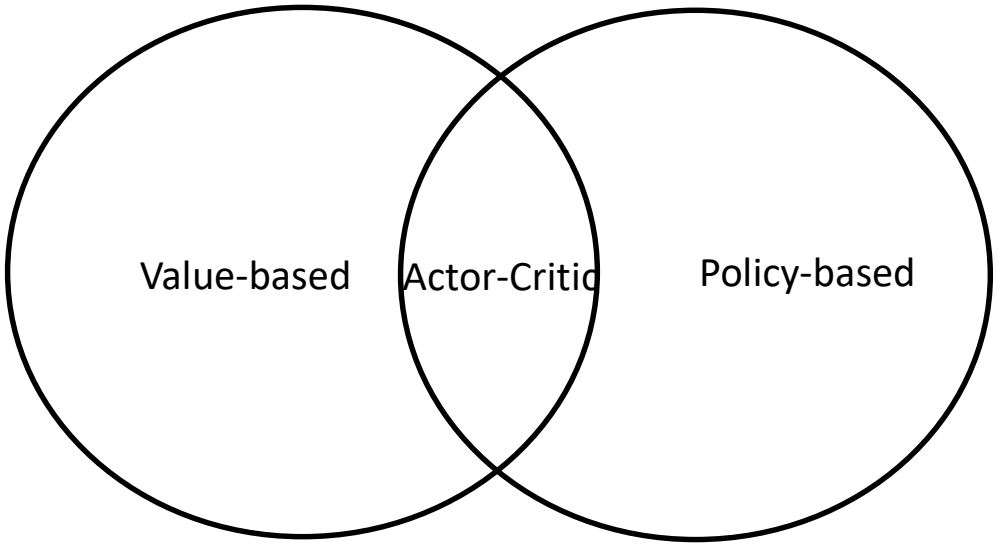
Human-level control through deep reinforcement learning, Nature Letter 2015



100 x (DQN score – random play score) / (human score – random play score)

Value-based vs Policy-based RL

- Value function
  - learnt and implicit policy, e.g., ϵ-greedy
- Policy learnt,
  - no value function, learnt policy $\pi_\theta(s, a) = P(a|s, \theta)$
- Actor-Critic
  - Learnt value function and policy



Policy-based RL

- Advantages:
  - Better convergence properties
  - Effective in high-dimentional/continuous spaces
  - Can learn stochastic policies
    - Es. games like rock-paper-scissors,
    - Partial observable domain
- Disadvantages:
  - Typically converge to a local
  - Evaluating a policy may be inefficient with naive methods

Policy Objective Function

- Given a policy $\pi_\theta(s, a)$ with parameters θ find the best θ
- How do we measure the quality of a policy?
- Episodic
  - $J_1(\theta) = V^{\pi_\theta}(s_1) = E_{\pi_\theta}[v_1]$
- Continuing (average)
  - $J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$
- Continuing per time-step (average per time-step)
  - $J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$

Same policy gradient

Policy Optimization

- Policy based RL is an optimization problem
- Find θ that maximizes $J(\theta)$
- Gradient and not gradient methods
- We focus on gradient methods
  - Gradient ascent method
  - Find a local maximum of $J(\theta)$ by ascending the gradient of the policy

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

$$\theta_{t+1} = \theta_t + \alpha\,\nabla_\theta J(\theta)$$

Policy Optimization

- If the action state is discrete
  - Parametrized preferences $\pi_\theta(a, s) = h(s, a, \theta)$ …
  - … actions with the high preference have high probability to be selected
  - $\pi_\theta(a, s) = \frac{\exp(h(s,a,\theta))}{\sum_b \exp(h(s,b,\theta))}$
  - $h(s, a, \theta) = \theta^T x(a, s)$, with $x(a, s)$ feature vector

Score Function

- Assume policy differentiable with gradient $\nabla_\theta \pi_\theta(s,a)$
- Given the following
  - $\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a) \frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} = \pi_\theta(s,a) \nabla_\theta log\, \pi_\theta(s,a)$
- The score function is $\nabla_\theta log\, \pi_\theta(s,a)$
- Likelihood ratios $\pi_\theta(s,a) \nabla_\theta log\, \pi_\theta(s,a)$
  - E.g., gaussian policy $a \sim N(\mu(s), \sigma^2)$, with mean linear combination of features $\mu(s) = \phi^T \theta$ score function $\nabla_\theta log\, \pi_\theta(s,a) = \frac{(a-\mu(s))\phi(s)}{\sigma^2}$

One-Step MDPs

- MDP,
  - start in state $s \sim d(s)$
  - End after one action with $r = R_{s,a}$

- Policy
  - $\mathrm{J}(\theta) = \mathrm{E}_{\pi}[\mathrm{r}] = \sum_s d(s) \sum_a \pi_\theta(s,a) R_{s,a}$

- Policy gradient
  - $\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a)\, \nabla_\theta log\, \pi_\theta(s,a)\, R_{s,a} = \mathrm{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) r]$

Policy Gradient Theorem

- Multi-sep MDPs
- Replace instantaneous reward with $Q^{\pi}(s,a)$

- For any differentiable policy $\pi_{\theta}(s,a)$, for any policy objective function $J = J_1, J_{avR}, \frac{1}{1-\gamma} J_{avV}$ the policy gradient
  - $\nabla_{\theta} J(\theta) = \mathrm{E}_{\pi_{\theta}}[\nabla_{\theta} log \pi_{\theta}(s,a) \mathrm{Q}^{\pi_{\theta}}(\mathrm{s},\mathrm{a})]$

MC Policy Gradient

- Update parameters by stochastic gradient ascent
- Exploit policy gradient theorem
- Use v_t as unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$
  - $\Delta\theta_t = \alpha \nabla_\theta log \pi_\theta(s, a) v_t$

Algorithm REINFORCE

Initialise θ arbitrarily

for each episode $\{s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T\}$ **do**

for t=1 to T – 1 **do**

$$\theta \leftarrow \theta + \alpha \nabla_\theta log \pi_\theta(s, a) v_t$$

end for

end for

return θ

end

Actor-Critic

- MC policy gradient has high variance
- Use a Critic to estimate action-value function
  - $Q_w(s,a) \approx Q^{\pi_\theta}(s_t, a_t)$

- Actor-Critic algorithms maintain two sets of parameters
  - Critic updates action-value function
  - Actor updates policy parameters in the direction suggest by the Critic

- Approximate policy gradient
  - $\nabla_\theta J(\theta) \approx \mathrm{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) \mathrm{Q_w(s,a)}]$
  - $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$

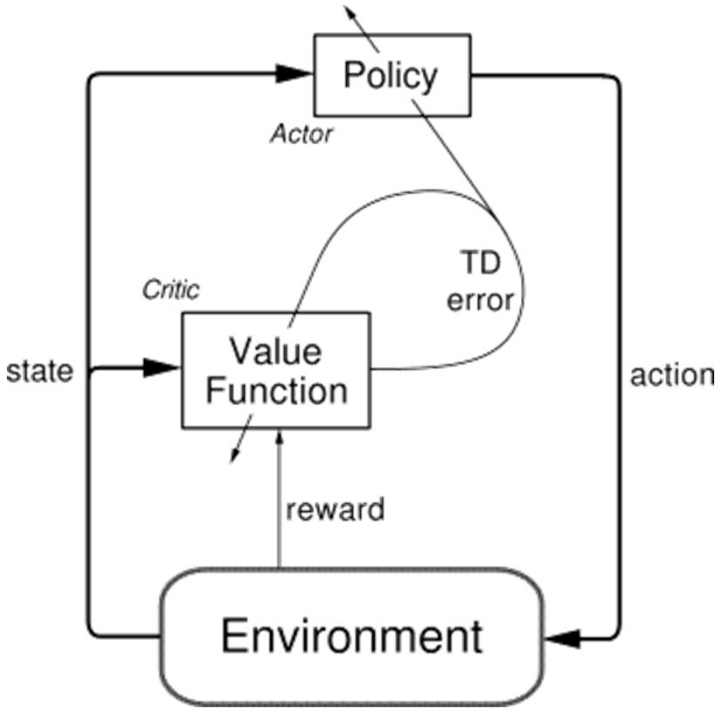
Actor-Critic

- Approximate policy gradient
  - $\nabla_\theta J(\theta) \approx \mathrm{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(s,a) \mathrm{Q_w}(\mathrm{s},\mathrm{a})]$
  - $\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)$

- How to assess π_θ?
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(λ)
  - …

Actor Critic Method

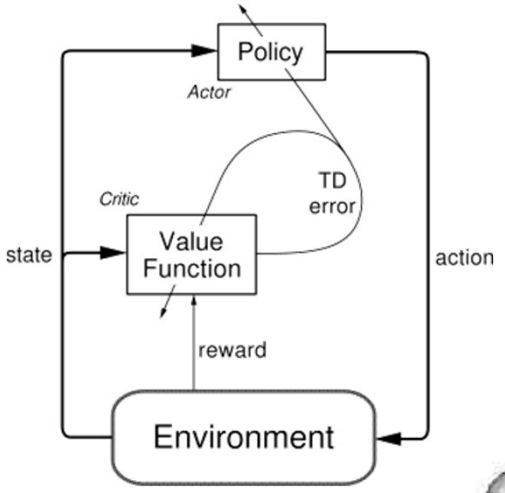
- Policy structure (*actor)*: it selects the actions,
- Value function (*critic)*: it criticizes the actions made by the actor.

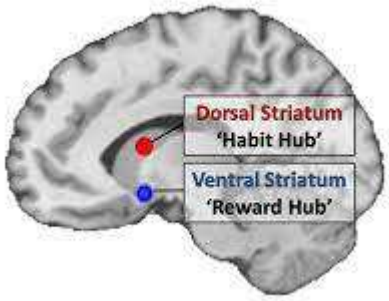


- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

Actor Critic Method

- Policy structure (*actor)*: it selects the actions,
- Value function (*critic)*: it criticizes the actions made by the actor.





- Explicit representation of policy as well as value function
- Minimal computation to select actions
- Can learn an explicit stochastic policy
- Can put constraints on policies
- Appealing as psychological and neural models

- two parts of the striatum - dorsal and ventral subdivisions – may work as actor and critic
- TD error has the dual role of being the reinforcement signal for both the actor and the critic, with different influence

Actor-Critic

TD-error is used to evaluate actions:

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

If actions are determined by preferences, $p(s,a)$, as follows:

$$\pi_t(s,a) = \Pr\{a_t = a \,|\, s_t = s\} = \frac{e^{p(s,a)}}{\sum_b e^{p(s,b)}}, \qquad \text{(softmax)}$$

then you can update the preferences like this:

$$p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta \delta_t$$

p(s, a) tendency to select
(*preference* for) each action

Actor-Critic

- Action-value critic
- Linear value function approximation
  - $Q_w(s,a) = \phi(s,a)^T w$
  - Critic updates w by linear TD(0)
  - Actor updates θ by policy gradient

Algorithm QAC

Initialise s, θ arbitrarily
Sample $a \sim \pi_\theta$
for each step **do**
Sample reward $r = R_{s'}^a$ and transition $s' \sim P_s^a$
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
return θ
end

Actor-Critic

- Action-value critic
- Linear value function approximation
  - $Q_w(s,a) = \phi(s,a)^T w$
  - Critic updates w by linear TD(0)
  - Actor updates θ by policy gradient

One-step Actor–Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} I \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$