Week 4 CS 1.07.2018

Approximate Value Based Methods

Limitations of Tabular Methods

- in tabular SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]$$

- any single cell update does not affect any other cell

Limitation

- # of parameters > # of state

Memory Data Time

Function approximation in RL

# of parameters independent of # of state

$$\hat{v}(s, w) \approx v_\pi(s)$$

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

Any single parameter affects

values of all states.

It is wise to obtain the following mappings:

$$s \longmapsto v_\pi(s)$$

$$s, a \longmapsto q_\pi(s, a)$$

Reduction to Supervised learning problem - ME

Monte Carlo

Ideal world goals

$$s \longmapsto \mathbb{E}_\pi [G_t \mid S_t = s]$$

$$s, a \longmapsto \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

$$\left(G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k} \right)$$

Sample based estimates of goals

$g(s)$, $g(s, a)$

$$s \longmapsto R(s, \pi(s)) + \gamma G_{t+1}$$

$$s, a \longmapsto R(s, a) + \gamma G_{t+1}$$

] numbers, known at the end of an episode

Temporal difference, TD

Ideal world goals

$$s \mapsto \mathbb{E}_\pi\left[R_{t+2} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

$$s,a \mapsto \mathbb{E}_\pi\left[R_{t+2} + \gamma v_\pi(S_{t+2}) \mid S_t = s, A_t = a\right]$$

Sample based estimates of goals $g(s)$, $g(s,a)$

$$\left.\begin{aligned} s &\mapsto R(s,\pi(s)) + \gamma \hat{v}_\pi(S_{t+1}, w) \\ s,a &\mapsto R(s,a) + \gamma \hat{v}_\pi(S_{t+1}, w) \end{aligned}\right] \text{ functions of parameters } w$$

Loss is the same as for regression problem.

$$L(w) = \frac{1}{2} \sum_{s,a} \rho_\pi(s,a)\left[g(s,a) - \tilde{q}_\pi(s,a,w)\right]^2$$

Weight of "importance" $\rho_\pi(s,a)$ can be thought of as

- fraction of time
  - policy encounter state s
  - and in that state makes action a

Recap: online, offline, on-policy, off-policy

Off-policy - more general, very difficult

- Behaviour policy - collects data (makes actions)
- Target policy - subject to evaluation & improvement

<u>On-policy</u> - less general, easier

- Behaviour policy equals target policy

<u>Usually, but not always</u>

- Online - update policy during the episode (e.g. TD)
- Offline - update policy after an episode ends (e.g. MC)

Loss functions in value based RL

minimizing loss with gradient

loss

$$L(w) = \frac{1}{2} \sum_{s,a} \rho_\pi(s,a) \overbrace{\left[q(s,a) - \tilde{q}_\pi(s,a,w) \right]^2}^{L_{s,a}(w)}$$

Gradient descent (GD)

$$w \leftarrow w - \alpha \nabla_w L(w)$$

Stochastic gradient descent (SGD)

On-policy: $s, a \sim \rho_\pi$

Off-policy: $s, a \sim \rho_b$ (behaviour)

$$w \leftarrow w - \alpha \nabla_w L_{s,a}(w)$$

From gradient to semi-gradient

$$L(w) = \frac{1}{2} \sum_{s,a} \rho_\pi(s,a) \left[\underbrace{g(s,a)} - \hat{q}_\pi(s,a,w) \right]^2$$

goal is a function of w in case of TD (in MC is a simple number)

Stochastic semi-gradient update treats goals as fixed:

$$\nabla_w g(s,a) = 0$$

This assumption simplifies math a lot.

$$w \leftarrow w - \alpha \nabla_w L_{s,a}(w)$$

$$w \leftarrow w + \alpha \left[g(s,a) - \hat{q}_\pi(s,a,w) \right] \nabla_w \hat{q}_\pi(s,a,w)$$

Semi-gradient update

- Treat goals $g(s,a)$ as fixed number
- Changes parameters to move estimates closer to targets.
- Ignores effect update on the target
- Is not a proper gradient

- No SGD's theoretical convergence properties
- Converges reliably in most cases
- More computationally efficient than true SGD

- Meaningful thing to do.

Target $g(s,a)$ define and what and how do we learn

SARSA

$$g(s,a) = R(s,a) + \gamma \hat{q}_\pi(S_{t+1}, A_{t+1}, w)$$

Expected SARSA

$$g(s,a) = R(s,a) + \gamma \sum_a \pi(a|S_{t+1}) \hat{q}_\pi(S_{t+1}, a, w)$$

Q-learning

$$g(s,a) = R(s,a) + \gamma \max_a \hat{q}_\pi(S_{t+1}, a, w)$$

↑ all s are random variables

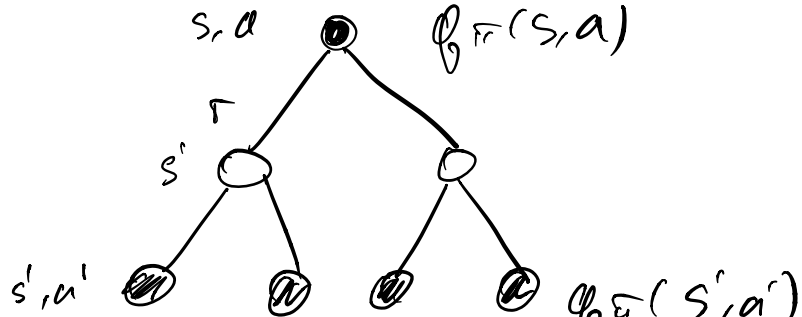
Semi-gradient SARSA (on-policy)

Tabular SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) +$$

$$+\alpha\left[R_{t+2}+\gamma Q(S_{t+1},A_{t+1})-Q(S_t,A_t)\right]$$

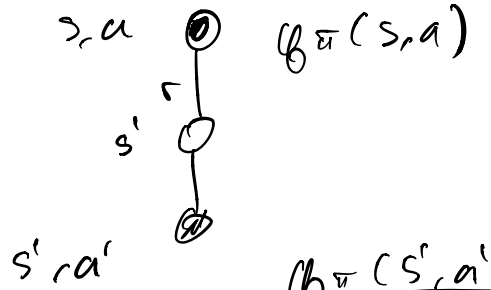
Bellman expectation equation for $q(s,a)$



Full-width backup

Approximate SARSA

$$w \leftarrow w + \alpha\left[R_{t+1}+\gamma \hat{q}(S_{t+1},A_{t+1},w)-\hat{q}(S_t,A_t,w)\right]\nabla \hat{q}(S_t,A_t,w)$$



Sample based estimate

Semi-gradient expected SARSA (off-policy)

Tabular expected SARSA

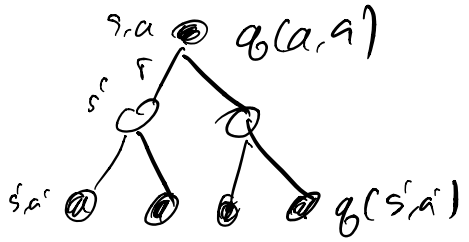
$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) +$$

$$+\alpha\left[R_{t+2}+\gamma\sum_a \pi(a|S_{t+1})\,Q(S_{t+1},a)-Q(S_t,A_t)\right]$$

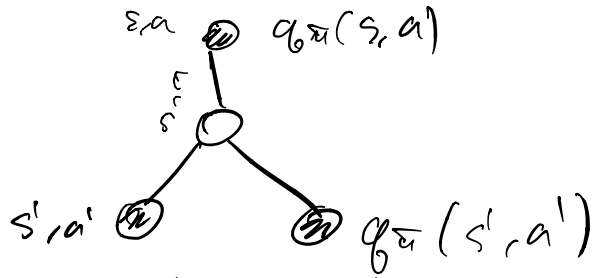
Approximate expected SARSA

$$w \leftarrow w+\alpha\left[R_{t+2}+\gamma\sum_a \pi(a|S_{t+2})\,\hat{q}(S_{t+2},a,w)-\right.$$
$$\left.-\hat{q}(S_t,A_t,w)\right]\cdot\nabla\hat{q}(S_t,A_t,w)$$

Bellman expectation equation for $q_\pi(s,a)$



Full-width backup



sample based estimate

Semi-gradient Q-learning (off-policy)

Tabular Q-learning

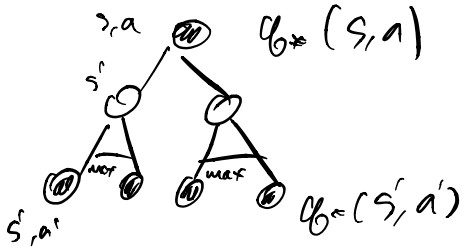
$$Q(S_t,A_t)\leftarrow Q(S_t,A_t)+\alpha\left[R_{t+1}+\right.$$
$$\left.+\gamma\max_a Q(S_{t+1},a)-Q(S_t,A_t)\right]$$

Approximate Q-learning

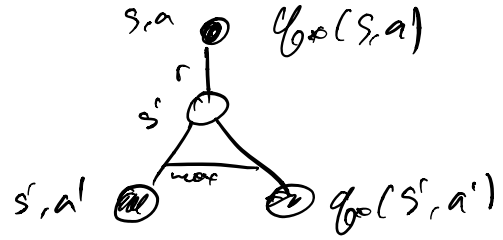
$$w\leftarrow w+\alpha\left[R_{t+2}+\gamma\max_a \hat{q}(S_{t+2},a,w)-\right.$$

$- \hat{q}(S_t, A_t, w) \} \nabla \hat{q}(S_t, A_t, w)$

Bellman optimality equation $q_*(s,a)$



full-width backup



sample based estimate

Difficulties with Approximate methods

Round (0): general well-known boring problems

- Too many state-action pairs

$|A| \cdot |S| = 18 \cdot 256^{210 \times 160 \times 3} \rightarrow 10^{83}$ (atoms)

- Methods should be sample efficient

- Meaningful decisions need accurate value estimates
  - Methods should be very flexible
- Any model has finite # of params
  - Even universal approximators are limited.

- Bias-variance tradeoff - overfitting / underfitting
  - Cross validation on scarce data is unreliable

Problems with Supervised Learning in RL

① Unusual data
   1. Correlation
   2. Dependence on policy
   3. Proximity in space and time.

② Nonstationarity
   1. Policy improvement
   2. Nonstationary task

③ The deadly triad (off-policy learning)

Round (1.1): data-correlated samples

Sequences of highly correlated non-iid data

- Forgetting useful facts and features
- Much of SL rely on iid assumption
- Learning can be inefficient
- SGD loses convergence guarantees

Round (1.2) data-dependence on policy

- Unseen data comes as agent learns new things.
- An agent can learn the fatal behavior
  - New data could be insufficient to unlearn it
- Agent experiments with data stream by exploration

Round (1.3) data-proximity in space and time

- $Q(s,a)$ often changes abruptly in s and a
  - Close states could be arbitrary far in value
  - Successive states could be arbitrary far in value.
  - Unstable gradients, much data needed for SGD
  - Numerical problems.
- TD-specific. Errors in estimates propagette

Approximate Q-learning update

$$w \leftarrow w + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{q}(S_t, A_t, w) \right] \nabla \hat{q}(S_t, A_t, w)$$

Makes $\hat{q}(S_t, A_t, w)$ closer to $R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w)$

Round (2): inherent nonstationarity - GD I

- Cannot assume fixed training data distribution
  - TD targets are invalidated
  - MC targets no longer apply
- Numeric problems (oscillating behaviour)
  - Small change in Q-values
    - → drastic change in policy
    - → drastic change in training data

→ large gradients
→ large update in Q-values.

- The environment can itself be nonstationary

Round(3): The deadly triad - model divergence

1. Off-policy learning
   - E.g. learning target π while following behaviour b
2. Bootstrapping
   - Updating a guess towards another guess (TP, DP)
3. Function approximation
   - Using model with # of params smaller then # of states

Divergence is not connected with
- Sampling, exploration, greediness, control
- Complexity of the model (even linear models diverge)

Programming assignments

Approximate Q Learning - Cart Pole

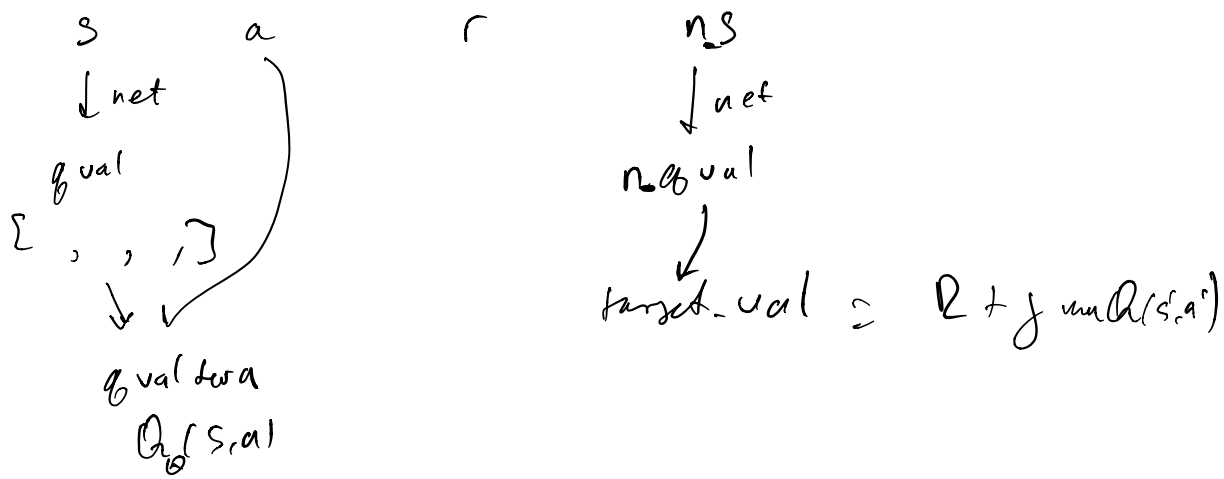
$$L = \frac{1}{N} \sum_i \left(Q_\theta(s,a) - \left[r(s,a) + \gamma \max_{a'} Q_-(s',a') \right] \right)^2$$

$Q_\theta \equiv Q_-$ but don't propagate through $Q_-(s,a)$

tabular

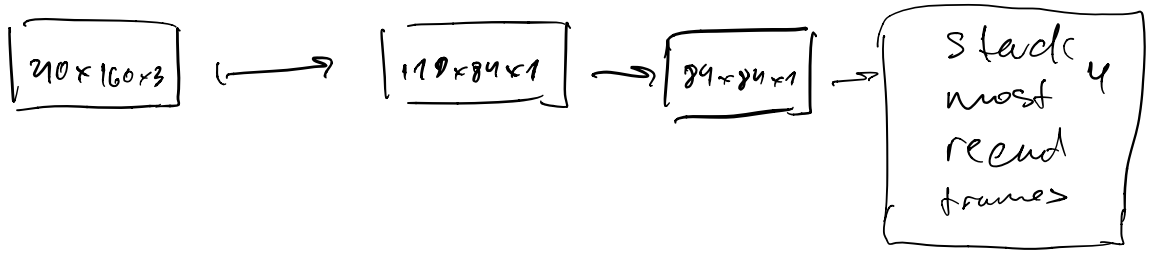
$$V(s) = \max_a Q(s,a)$$

$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \cdot (r + \gamma V(s'))$$



Case Study: Deep Q-Network.

- Epsilon-greedy exploration



Stack 4 frames = fourth-order markov assumption

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

$$p(r_{t+1}, s_{t+1} | s_0, a_0, \ldots r_t, s_t, a_t) = p(r_{t+1}, s_{t+1} | s_t, a_t, \ldots s_{t-3}, a_{t-3})$$

DQN fixes some instability problems

1. Sequential correlated data
   - may hurt convergence and performance
2. Instability of data distribution due to policy changes
   - policy may diverge and/or oscillate
3. Unstable gradients
   - Absolute value of $Q(s,a)$ may vary much across states

⇓

Solution

1. Experience replay
2. Target networks
3. Reward clipping

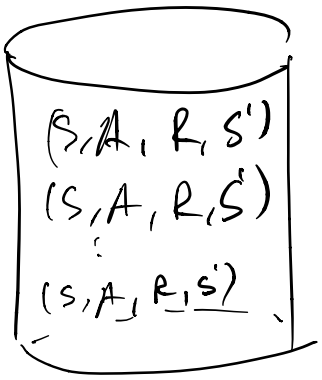
(1) Sequential data ↦ Experience replay

Semi-gradient update for approximate Q-learning

$$w \leftarrow w + \alpha \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{q}(S_t, A_t, w) \right] \nabla \hat{q}(S_t, A_t, w)$$

1. Store tuples (S, A, R, S') in a pool
2. Sample tuple(s) from pool at random
3. Update the model of Q-func
4. Act epsilon-greedy w.r.t Q-func
5. Add new samples to pool
6. Go to 2

↙ just decorrelate data if pool is large



only for off-policy learning

Prop / Cons

+ Help against correlations
+ Increase sample efficiency
+ Reduce variance of updates
+ Computations are easy to parallel
+ Analog of sample based model of the world
- Memory intensive (DQN stored 10^6 interactions)
- Random sampling from pool could be improved
- Older interactions were obtained under weaker policy

(2) Policy oscillation → Target networks

Unwanted source of instability
- Targets are functions of parameters
- Errors in estimates propagate into other estimates

Standard Q-learning update

$$w \leftarrow w + \alpha \left[R_{t+2} + \gamma \max_a \hat{q}(S_{t+1}, a, w) - \hat{q}(S_t, A_t, w) \right] \nabla \hat{q}(S_t, A_t, w)$$

Q-learning update with target networks

$$w \leftarrow w + \alpha \left[R_{t+2} + \gamma \max_a \hat{q}(S_{t+1}, a, w^-) - \hat{q}(S_t, A_t, w) \right] \nabla \hat{q}(S_t, A_t, w)$$

copy parameters to **target network**
- (hard) Copy w to w^- once in awhile
- (soft) Maintain w^- as moving average of w

(3) Unstable gradients ↦ Reward clipping

- Don't know the scale of reward beforehand
- Don't want numeric problems with large Q-values.

- Clip the reward to $[-1; 1]$
  + less peaky Q-values
  + good gradients
  - cannot distinguish between good and great

Honor

DQN: Statistical issues

Recap: Approximate Q-learning

$Q(s, a_0), Q(s, a_1) Q(s, a_2)$ ← model (θ-parameters) ← image

Q-values

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

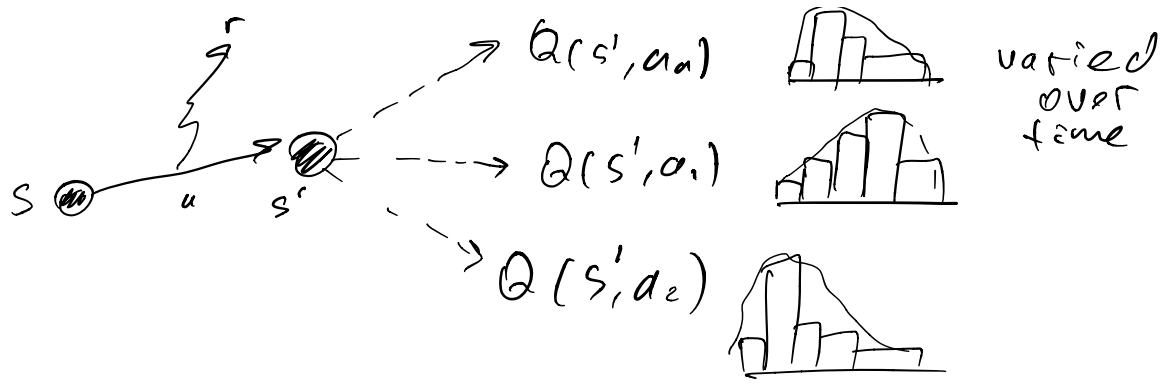
Objective

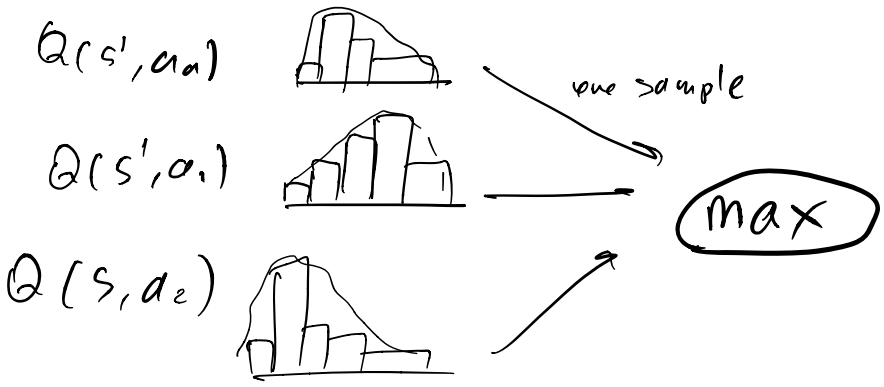
$$L = \left(Q(s_t, a) - [r + \gamma \cdot \max_{a'} Q(s_{t+1}, a')] \right)^2$$

Gradient step

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

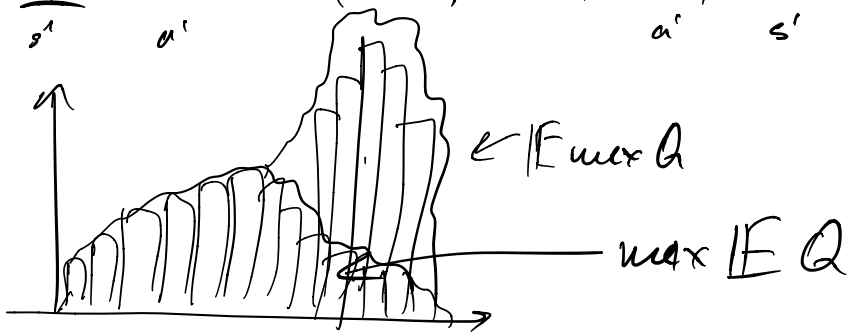
$$L = \left(Q(s_t, a_t) - [r + \gamma \underbrace{\max_{a'} Q(s_{t+1}, a')}_{\text{consider const}}] \right)^2$$





so

$$\mathbb{E}_{s'} \max_{a'} \hat{Q}(s_{t+1}, a') \geq \max_{a'} \mathbb{E}_{s'} \hat{Q}(s_{t+1}, a')$$



$$\mathbb{E} \max(\xi, \zeta) \geq \max(\mathbb{E}\xi, \mathbb{E}\zeta)$$

Double Q-learning

Idea: train two Q-functions separately.

Q_1 and Q_2

Objective for Q_1

$$\hat{Q}_1(s_t, a_t) = r_t + \gamma \cdot Q_2(s_{t+1}, \underset{a^*}{\operatorname{argmax}}\, Q_2(s_{t+1}, a^*))$$

Objective for Q_2

$$\hat{Q}_2(s_t, a_t) = r_t + \gamma\, Q_1(s_{t+1}, \underset{a^*}{\operatorname{argmax}}\, Q_2(s_{t+1}, a^*))$$

Algorithm:

Initialize Q_1, Q_2

forever:

sample s_t, a_t, r_t, s_{t+1} from env

if flip_coin() == 'heads':

$$\hat{Q}_1(s_t, a_t) = r_t + \gamma\, Q_2(s_{t+1}, \underset{a^*}{\operatorname{argmax}}\, Q_1(s_{t+1}, a^*))$$

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha\left[\hat{Q}_1(s_t, a_t) - Q_1(s_t, a_t)\right]$$

else:

$$\hat{Q}_2(s_t, a_t) = r_t + \gamma\, Q_1(s_{t+1}, \underset{a^*}{\operatorname{argmax}}\, Q_2(s_{t+1}, a^*))$$

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha\left[\hat{Q}_2(s_t, a_t) - Q_2(s_t, a_t)\right]$$

Now consider DQN

- Neural net policy
- Experience replay
- Target networks

Idea 1: train two networks separately.

Idea 2: use target network instead Q_2:

Classic DQN:

$$\hat{Q}(s_t, a_t) = r_t + \gamma \max_{a^*} Q^{old}(s_{t+1}, a^*)$$

Double DQN:

$$\hat{Q}(s_t, a_t) = r_t + \gamma \cdot Q^{old}(s_{t+1}, \arg\max_{a^*} Q(s_{t+1}, a^*))$$

step-by-step

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

$$\Downarrow$$

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

$$\Downarrow$$

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

More DQN tricks

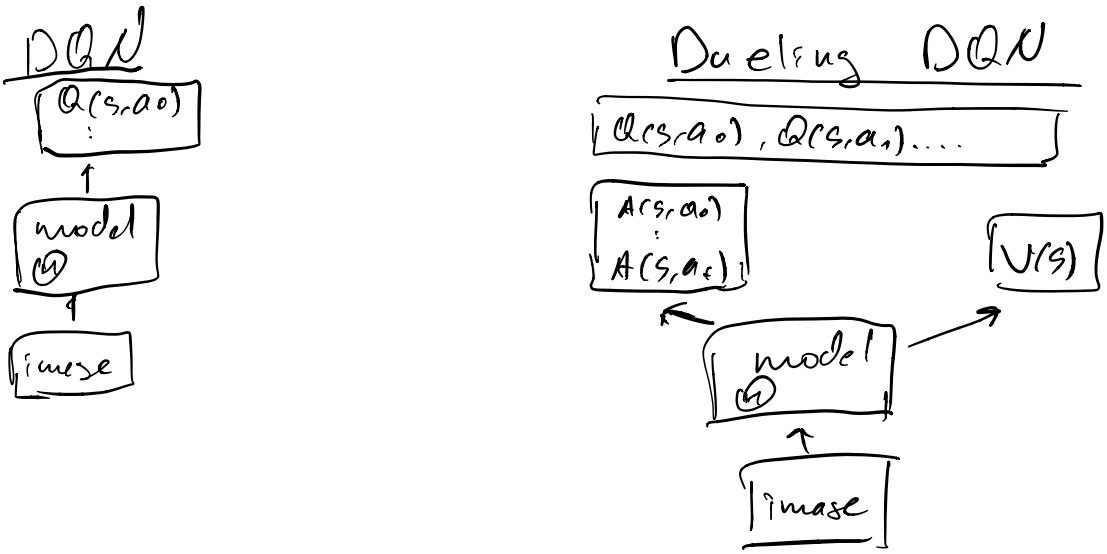
Decomposition: $Q(s,a) = V(s) + A(s,a)$

where $Q(s,a)$ are action values, as usual

$V(s)$ are state values ($V^*(s)$ or $V_\pi(s)$)

$A(s,a)$ is the "advantage"

$A(s,a) = Q(s,a) - V(s)$



Estimating A(s,a)

option 1:

$$A(s,a_i) = nn(s)[i] - \max_j nn(s)[j]$$

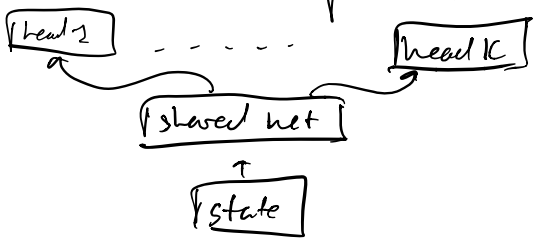
option 2:

$$A(s,a_i) = nn(s)[i] - \frac{1}{|A|}\sum_j nn(s)[j]$$

where nn(s) is a neural network activation for advantage (left branch)

Bootstrap DQN

Each head predicts all q-values



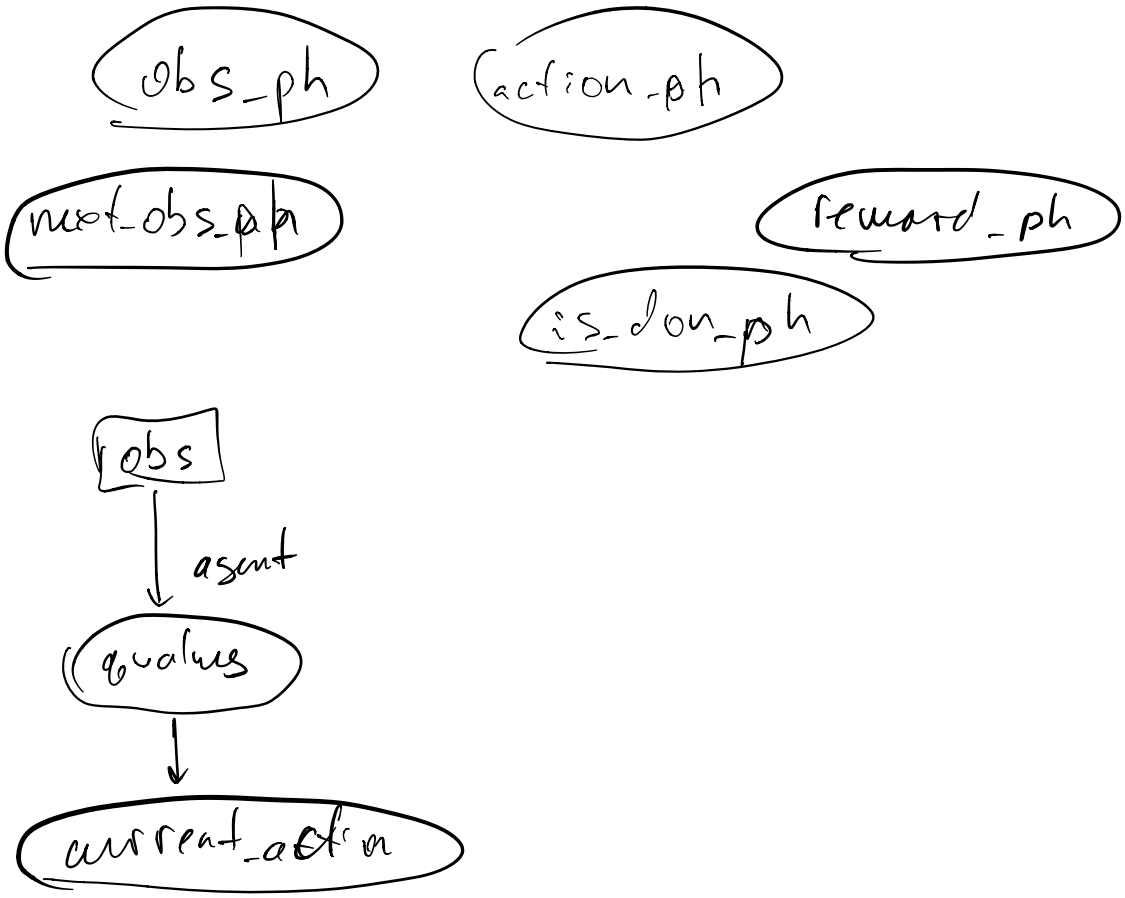
Why: make exploration great again!

How: maintain K separate heads (for example, last 2 layers) and shared body weights.

pick random head from K.

DQN - Breakout

$$Q_{reference}(s,a) = r + \gamma \cdot \max_{a'} Q_{target}(s',a')$$



$$L = \frac{1}{N} \sum_i \left[Q_\theta(s,a) - Q_{reference}(s,a) \right]^2$$

$$Q_{reference}(s,a) = r(s,a) + \gamma \cdot \max_{a'} Q_{target}(s',a')$$

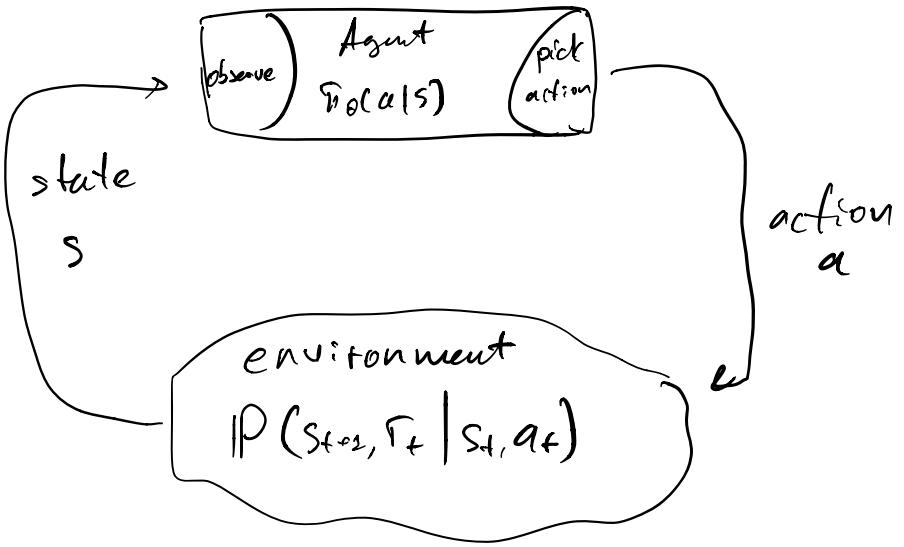
Partial observability

Problem: In most real environments agent observation does not hold all information about system state

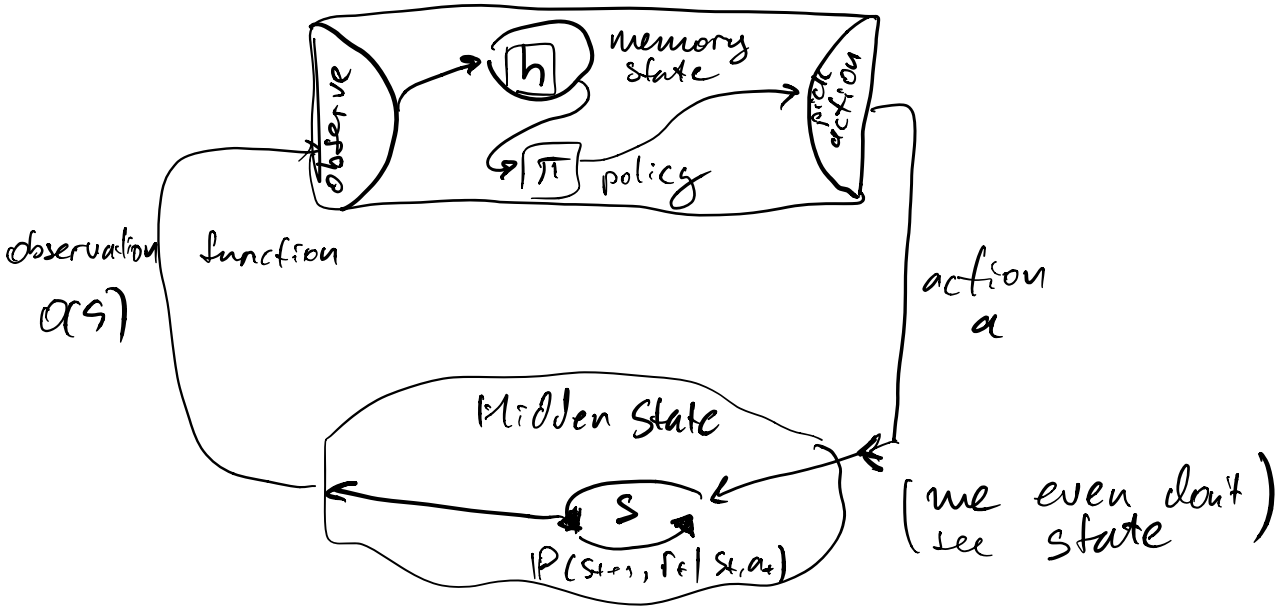
Examples:

- Robotics: field of view
- Trading bots: market state
- Atari Breakout: ball velocity

Recap: MDP



Partial Observable MDP



N-gram heuristic

idea:

$s_t \neq O(s_t)$

$s_t \approx (O(s_{t-n}), a_{t-n}, \dots O(s_{t-1}), a_{t-1}, O(s_t))$

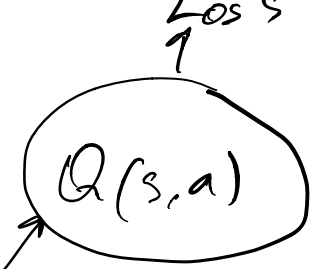
e.g. ball movement in breakout

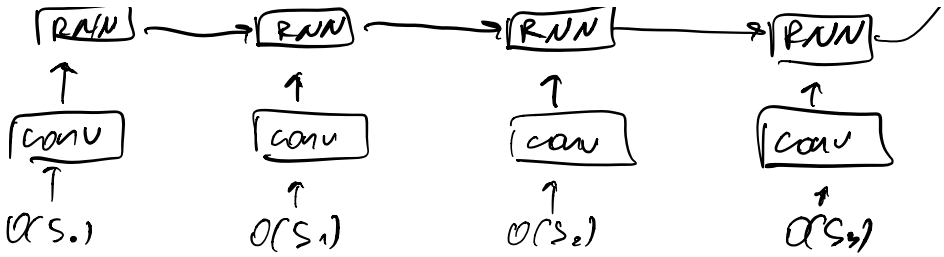
Recurrent neural network

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

$$Q(s_t, a_t) \approx Q(h_t(h_{t-1}, O(s_t)))$$

$$h_{t+1} = \sigma(W_{hid} \cdot h_t + W_{inp} \cdot O(s_t) + b)$$





chain rule unroll

$$\frac{\partial Loss}{\partial P(t_n)} \cdot \frac{\partial P(t_n)}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_n} \cdot \frac{\partial h_2}{\partial h_3} \cdot \frac{\partial h_1}{\partial h_2} \cdot \frac{\partial h_0}{\partial h_1} \cdots$$

DRQN

- use LSTM on top of DQN body
- predict Q-values from hidden state
- train on partial trajectories
- uses other existing DQN tricks

Week 5 CS 11.07.2018

Policy-based methods

Policy based RL vs Value-based RL

Intuition

Approximation error

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

contr-example

	True	A	B
$Q(s_0, a_0)$	1	1	2
$Q(s_0, a_1)$	2	2	1
$Q(s_1, a_0)$	3	3	3
$Q(s_1, a_1)$	100	50	100

A is better → → optimal policy

B is worse → → but L^2 smaller.

All kinds of Policies

Policies

Deterministic policy $a = \pi_\theta(a|s)$

Stochastic policy $a \sim \pi_\theta(a|s)$

(sampling takes care of exploration)

Two approaches

- **Value based**

Learn value function $Q_\theta(s,a)$ or $V_\theta(s)$

infer policy $\pi(a|s) = \#[a = \arg\max_a Q_\theta(s,a)]$

- **Policy based**

Explicitly learn policy $\pi_\theta(s,a)$ or $\pi_\theta(s) \mapsto a$. Implicitly maximize reward over policy

Policy gradient formalism

main idea

Objective

- Expected reward

$$J = \mathop{\mathbb{E}}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a) \\ \dots}} R(s,a,s',a',\dots)$$

- Expected discounted reward

$$J = \mathop{\mathbb{E}}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} G(s,a)$$

$$G(s,a) = r + \gamma \cdot G(s',a')$$

- Consider an 1-step process for simplicity

$$J = \mathop{\mathbb{E}}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} R(s,a) =$$

$$= \int_s p(s) \cdot \int_a \pi_\theta(s|a) \cdot R(s,a)\, da\, ds$$

$$J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in z_i} R(s,a)$$ ← Reward for agent's action

← Sample N sessions by following $\pi_\theta(a|s)$

The log-derivative trick

$$\nabla \log \pi(z) = \frac{1}{\pi(z)} \cdot \nabla \pi(z)$$

$$\pi \cdot \nabla \log \pi(z) = \nabla \pi(z)$$

Analytical inference

$$J = \int_s p(s) \int_a \pi_\theta(a|s)\, R(s,a)\, da\, ds$$

$$\nabla_\theta J = \nabla_\theta \int_s p(s) \int_a \pi_\theta(a|s)\, R(s,a)\, da\, ds =$$

$$= \int_s p(s) \int_a \nabla_\theta \pi_\theta(a|s)\, R(s,a)\, da\, ds =$$

$$= \int_s p(s) \cdot \underbrace{\int_a \pi_\theta(a|s) \cdot \nabla \log \pi(a|s)\, R(s,a)\, da}_{\text{that's expectation}}\, ds$$

Reinforce

Policy gradient

- Policy gradient

$$\nabla J = \mathbb{E}_{\substack{s \sim p(s) \\ a \sim \pi_\theta(s|a)}} \nabla \log \pi_\theta(a|s) \cdot Q(s,a)$$

- Approximate with sampling

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in Z_i} \nabla \log \pi_\theta(a|s) \cdot Q(s,a)$$

REINFORCE algorithm [on-policy]

- Initialize NN weights $\theta_0 \leftarrow$ random
- Loop
  - Sample N sessions Z under current $\pi_\theta(a|s)$

- Evaluate policy gradient

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in \tau_i} \nabla \log \pi_\theta(a|s) \cdot Q(s,a)$$

Ascend $\quad \theta_{new} \leftarrow \theta_{old} + \alpha \cdot \nabla J$

$Q(s,a) = V(s) + A(s,a)$

Actions influence $A(s,a)$ only, so $V(s)$ is irrelevant

$$\nabla J \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in \tau_i} \nabla \log \pi_\theta(a|s) \cdot (Q(s,a) - b(s))$$

Anything that doesn't depend on action directly, $b(s) = V(s)$

$b(s)$ - baseline.

Programming Assignment: Reinforce

$$E(p) = -\sum_x p(x) \log p(x)$$

$$E(p) = -\int_x p(x) \log p(x)$$

$$G_t \triangleq R_{t+1} + \gamma R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} =$$

$$= R_t + \gamma G_{t+1}$$

$t = 0 \ldots T$

T

$G_0 = \sum_{k=0}^{T} \gamma^k R_k$ $\qquad$ $G_T = \sum_{k=0}^{} \gamma^k R_{T+k} = \{k=0\} =$

$= \gamma^0 \cdot R_T = R_T$

$[0, 0, 1, 0, 0, 1, 0]$
$\quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6$

$G_6 = 0$

$G_5 = 1 + \gamma \cdot G_6$

$G_4 = 0 + \gamma \cdot G_5$

Actor-critic

Advantage actor-critic

- Learn both $V(s)$ and $\pi_\theta(a|s)$
- Hope for best of both worlds

- Ideal: learn both $\pi_\theta(a|s)$ and $V_\theta(s)$
- Use $V_\theta(s)$ to learn $\pi_\theta(a|s)$ faster!

Non-trivia: how can we estimate $A(s,a)$ from (s, a, r, s') and V-function?

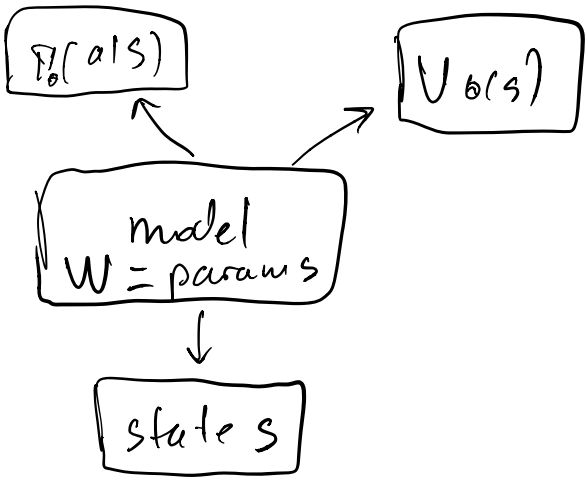
$$A(s,a) = Q(s,a) - V(s)$$
$$Q(s,a) = r + \gamma \cdot V(s')$$
$$A(s,a) = r + \gamma \cdot V(s') - V(s)$$

$\Downarrow$

$$A(s,a) = r + \gamma \cdot V(s') - V(s)$$

$$Q(s,a) = r + \gamma \cdot V(s')$$

$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{a,s \in \tau_i} \nabla \log \pi_\theta(a|s) \cdot \underbrace{A(s,a)}_{\text{consider const}}$$



$$\nabla J_{actor} \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in \tau_i} \nabla \log \pi_\theta(a|s) \cdot A(s,a)$$

$$L_{critic} \approx \frac{1}{N} \sum_{i=0}^{N} \sum_{s,a \in \tau_i} \left(V_\theta(s) - [r + \gamma \cdot V(s')] \right)^2$$

Duct tape zone

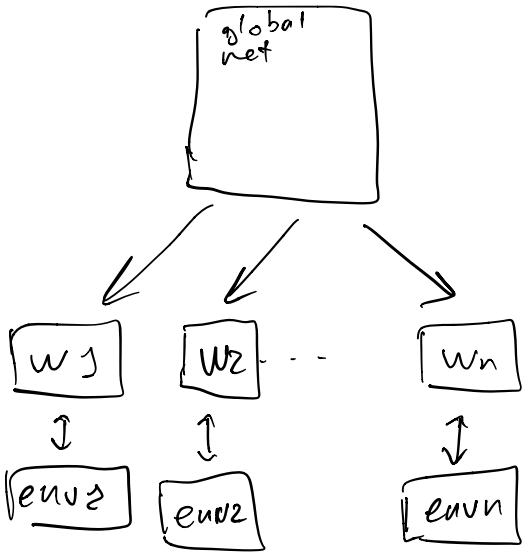
- $V(s)$ errors less important than in Q-learning
  - ator still learn even if critic is random, just slower

- Regularize with entropy
  - to prevent premature convergence
- Learn on parallel sessions
  - Or super-small experience replay
- Use log softmax for numerical stability

Policy-based vs Value-based

Value base	Policy base + Actor-critic
Q-learning SARSA value-iteration	REINFORCE Advantage Actor-Critic Crossentropy Method
off policy easyer from guize	IR Stochastic policy from guize
• Solves harder problem • Explicit exploration • Evaluate states and action • Easier to train off-policy	Solves easier problem Innate exploration and stochasticity Easier for continuous actions Compatible with supervised learning

Case study: A3C



- No experience replay
- Many parallel sessions
- Asynchronous updates

well-known A3C-LSTM

Combining supervised & reinforcement learning

Supervised pre-training

Reinforcement learning usually takes long to find optimal policy completely from scratch

We can use existing knowledge to help it.

- Human experience
- Known heuristic
- Previous system

So

- Supervised learning: $\left(\nabla \log \pi_\theta(a_{opt}|s)\right)$

$$\nabla \ell\ell h = \mathop{\mathbb{E}}_{x, y_{opt} \sim D} \nabla \log \mathbb{P}_\theta(y_{opt}|x)$$

- Policy gradient:

$$\nabla J = \mathop{\mathbb{E}}_{\substack{s \sim d(s) \\ a \sim \pi(a|obs(s))}} \nabla \log \pi(a|s) \cdot Q(a,s)$$

main difference: D - reference
$s \sim d(s)$ - generated

More out there

This domain is huge!

- Trust Region Policy optimization
  - policy gradient on steroids
- Deterministic policy gradients
  - off-policy & less variance
- Many many more!
  - Research happens as you read this line

A policy-based quiz

$Q(s,a)$ - discounted reward

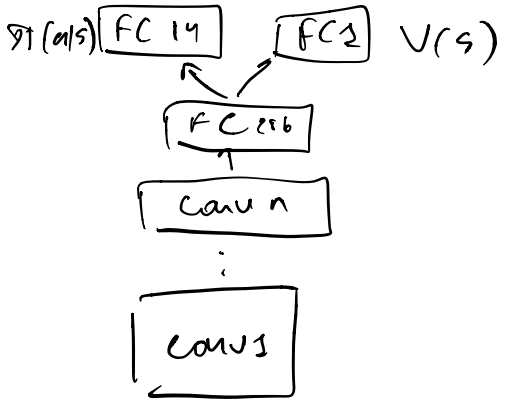
$r(s,a)$ - immediate reward

gamma

Practice Actor-Critic

env: (images $\mathbb{R}^{\text{height} \times \text{width} \times 4}$, num_actions 14)

agent:



sample action $\pi(a|s)$

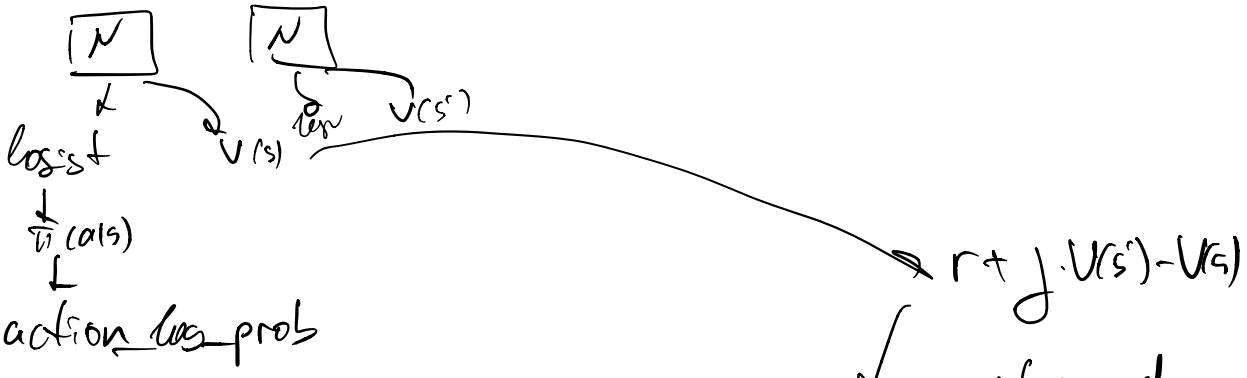
logits

↓

softmax $\equiv \pi_\theta(a|s)$

algorithm:

s, s', a, r,



actor_loss = $-\sum$ action_log_prob · advantage_stopgrad

critic_loss = $-\sum \left[V(s) - [r + \gamma \cdot V(s') \right]^2$

Week 6 22.07.2018

Measuring exploration

Recap: bandits

Exploration & exploitation previously

- Value-based methods:

  We know $Q(s,a)$ → we know optimal policy

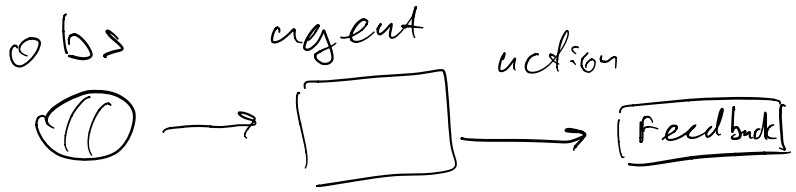
- Model-free setting: don't have $p(s', r | s, a)$

  Q-learning, SARSA, Expected Value SARSA

- Large states spaces

  Approximate Q-learning, DQN, Double DQN.

Multi-armed bandits



Examples

- banner ads (RTB)
- recommendations
- medical treatment

Basically it's 1-step MDP where

- $G(s,a) = r(s,a)$
- $Q(s,a) = \mathbb{E}\, r(s,a)$
- 50% shorter formulae

Regret: measuring the quality of exploration

Regret of policy $\pi(a|s)$:

Consider an optimal policy. $\pi^*(a|s)$

Regret per tick = optimal - yours

$$\eta(t) = \sum_{t} \mathbb{E}_{s,a \sim \pi^*} r(s,a) - \mathbb{E}_{s,a \sim \pi_t} r(s,a)$$

finite horizon: $t < max\_t$
infinite horizon: $t \longrightarrow inf$

> The message just repeat: Regret, Regret, Regret

Strategies:

- ε-greedy
  With probability ε take a uniformly random action. Otherwise take optimal action
- Boltzman
  Pick action proportionally to transformed Q values

$$\pi(a|s) = softmax(Q(s,a)/\tau)$$

- Policy based: add entropy

Uncertainty-based exploration

Intuitive explanation

Thompson Sampling

Uncertainty in rewards

We want to try actions if we believe there's a chance they turn out optimal.

Idea: Let's model how certain we are that $Q(s,a)$ is what we predicted.

- Policy: $P(Q|data)$
  - sample once from each Q distribution
  - take argmax over samples
  - which actions will be taken?

Optimism in face of uncertainty

Idea:

Prioritize actions with uncertain outcomes!

More uncertain = better

Greater expected value = better

Math: try until upper confidence bound is small enough.

- Policy:
  - Compute 95% upper confidence bound for each a
  - Take action with hightest confidence bound
  - What can we tune here to explore more/less?

UCB-1 (upper confidenc bound)

Frequentist approach

There's a number of inequalities that bound

$\mathbb{P}(x > t) <$ something

- E.g. Hoeffding inequality (arbitrary x in $[0,1]$)

$$\mathbb{P}\left(x - \mathbb{E}x \geq t\right) \geq e^{-2nt^2}$$

UCB-1 for bandits

Take actions in proportion to

$$\tilde{v}_a = v_a + \sqrt{\frac{2 \log N}{n_a}}$$

- N number of time-steps so far
- n_a times action a is taken

Upper conf. bound for r in $[0,1]$
If not - divide by r_{max}

Bayesian UCB

The usual way:
- Start from prior $\mathbb{P}(Q)$
- Learn posterior $\mathbb{P}(Q \mid data)$
- Take α-th percentile

What models can learn that?

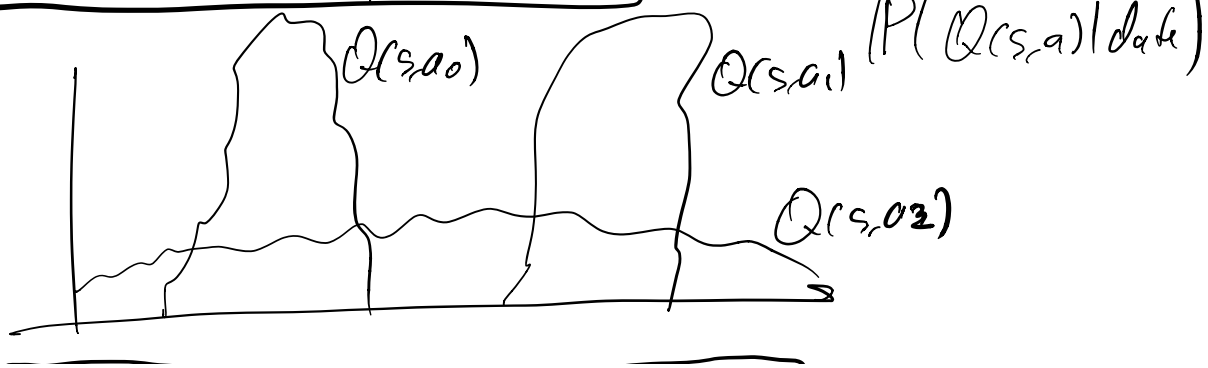
Approach 1: learn parametric $P(Q)$ e.g. normal

Approach 2: use bayesian neural networks

- UCB vs Bayesian UCB
  - choose any distribution you want
  - learn actual distribution, not upper bound
  - only as good as your prior.

- Regret: "money you could have saved" if you knew optimal actions from the get go
  - Less regret = better exploration
- ϵ-greedy and boltzmann never behave optimally
  - Make them greedy in the limit
- Using uncertainty
  - Thompson Sampling, UCB-2, Bayesian UCB

Quize: Exploration



Programming Assignment: Bandits

K actions $\quad r = 1 \quad 0 \le \theta_k \le 1$

$$\zeta(T) = T \cdot \theta^* - \sum_{t=1}^{T} r_t$$

$$\theta^* = \max_k \{\theta_k\}$$

K-pillows T-patients

BerBan: n_actions : n

$p_1, \dots p_n$

optimum reward $\max(p_i)$

Epsilon-greedy agent:

for $k = 1 \dots K$:

$\hat{\theta}_k = \alpha_k / (\alpha_k + \beta_k)$

$x_t \leftarrow$ argmax $\hat{\theta}_k$ with prob $1-\varepsilon$ else random.

apply x_t, r

$(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t}, \beta_{x_t}) + (r_t, 1 - r_t)$

done ■

Planning with Monte Carlo Tree Search

Introduction to planning

Monte Carlo Tree Search

Exploration reduces uncertainty

Model-free setting

- Unknown environment dynamics $p(s', r \mid s, a)$

Model-based settings

- Known model of the world
- Distribution model: $p(s', r \mid s, a)$
- Sample model: $(s', r) \sim G(s, a)$

Still don't know the global values of any actions! [we know only immediate R]

We don't know global value estimates

Given the model of the world, what is the best action?

1. Immediate rewards are insufficient
2. Time constraints on recovering sum of future rewards

Planning

Transform
- know immediate rewards & dynamics

into
- value / action-value estimates / explicit policy

Types of planning

Model-free - learn the policy by trial and error
Model-base - plan to obtain the policy

- Background planning (DP)
  - learning from simulated experience
  - improves estimates in arbitrary states
- Decision-time planning (heuristic search, MCTS)
  - focuses on current state
  - plans action selection for current state only
  - commit an action only after planning is finished

Guided planning - heuristic search

1. Exhaustive
2. Full-width with fixed depth
3. Full-width with fixed depth, approximate at leaves.
4. Heuristic.

Heuristic search - an umbrella term for many algorithms

- lookahead planning guided by heuristic function
  - select only "valuable" nodes to expand

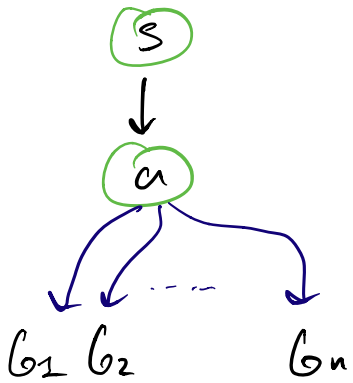
Heuristic search for RL

+ resources focused only on valuable paths
+ nearest states contribute the most to the return
- bad planning results when the model is unreliable
- dependence on the quality of heuristic

Obtain value estimates with rollouts

1. While within computational budget:
   a. From state s make action a
   b. Follow rollout policy until episode terminates
   c. Obtain return G_i
2. Output $\hat{q}(s,a) = \frac{1}{n} \sum_{i=1}^{n} G_i$

- No function approximation
+ Trials are independent → parallelization
- Dependence on # of returns averaged



Using rollouts for action selection

Greedy policy - policy improvements as in Value Iteration

$$\pi(s) \leftarrow \arg\max_a \hat{q}_\theta(s,a)$$

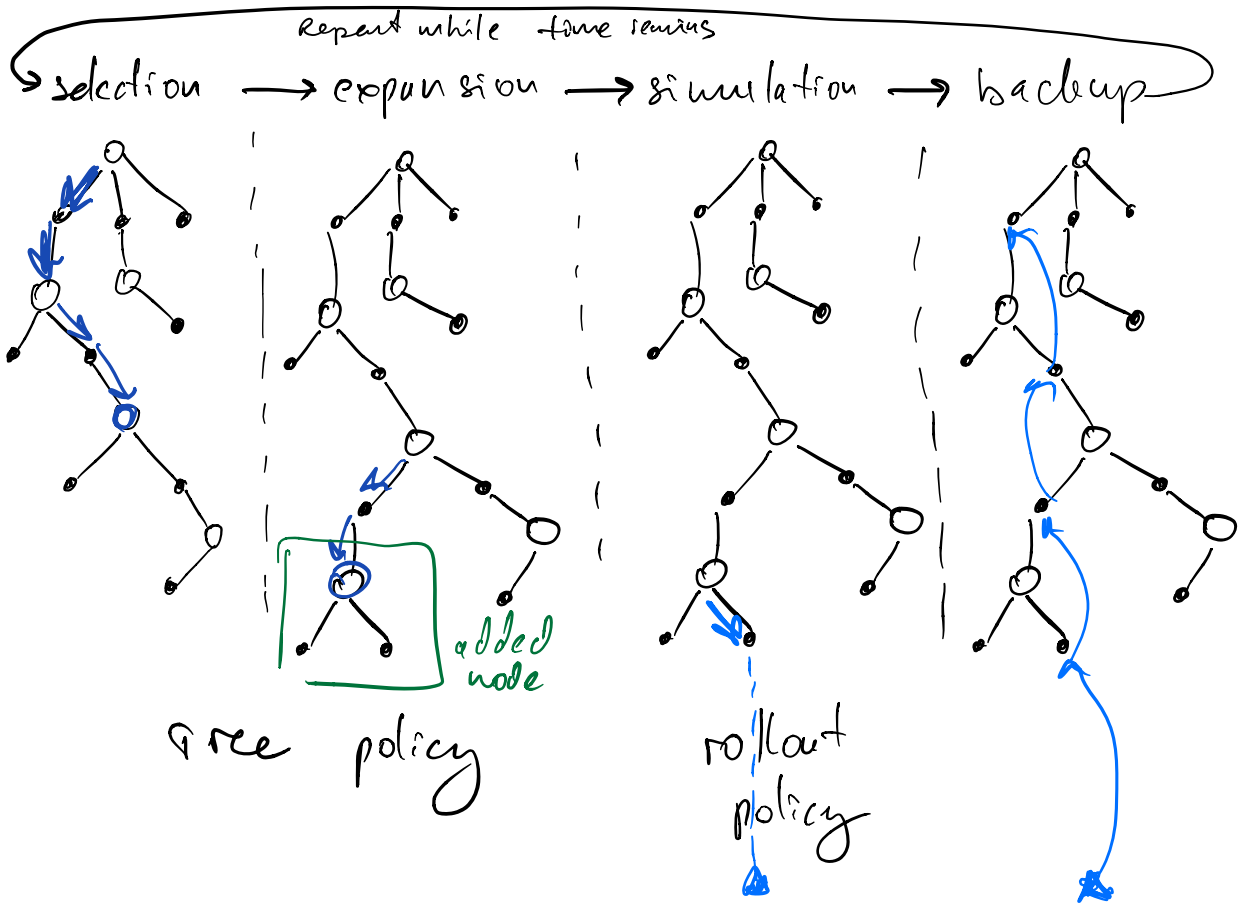
- No estimation of q_* } Good rollout policy is needed!
- No further policy improvement }
- Good rollout policy may require much of valuable time
  - May result in less reliable estimates of $\hat{q}_\theta(s,a)$
- Dependence on quality of rollout policy
+ Quite good results even for random rollout policy
- No estimates preserved between successive states
- Does not respect uncertainty in action-value estimates.

Monte Carlo Tree Search

Monte Carlo Tree Search at a glance

- Maintain two policies:
  - tree policy - respects uncertainty, guides the search

- rollout policy - estimates the return



. Exploitation - actions with existing good estimates

. Exploration - rarely tested actions

Treating action selection as multiarmed bandit (UCB1)

$$a_{tree}(s) = \arg\max_{a} \hat{q}(s,a) + 2C\sqrt{\frac{2\ln N(s)}{N(s,a)}}$$

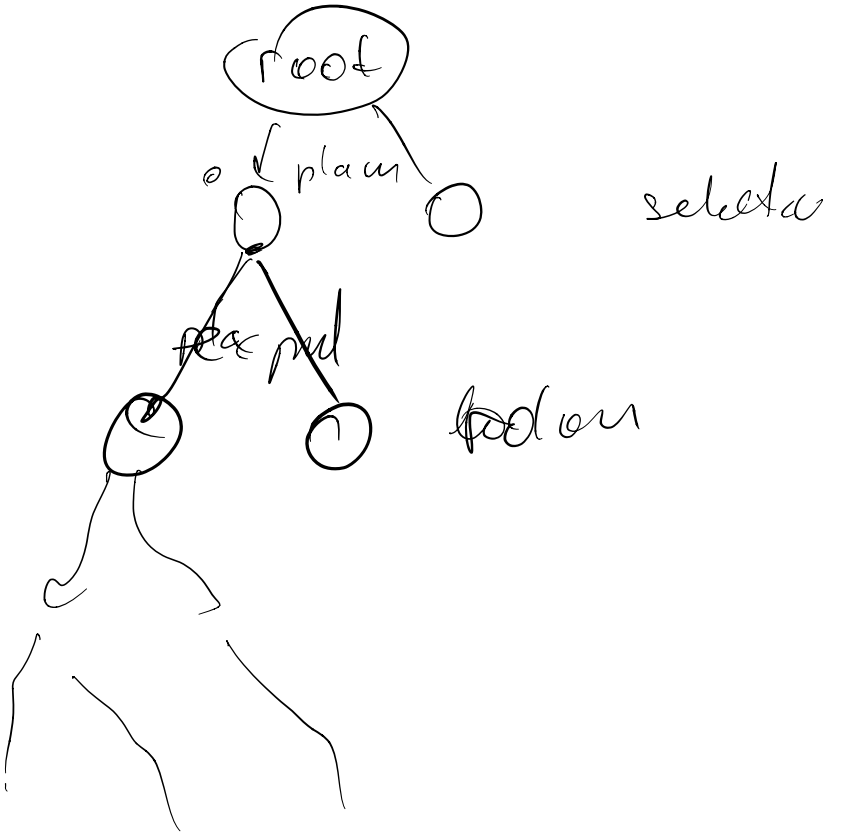
QUIZ: MCTS

UCB score

$\frac{x}{y}$ - numerator value_sum
$\frac{x}{y}$ - denominator times visited

Node:

is_leaf
is_root
get_mean_value = $\frac{\text{value\_sum}}{\text{value\_visit}}$
ucb_score: mu + ucb



RL outside games

General formalism

- Maximize $J = \mathbb{E}_{s \sim p(s),\ a \sim \pi(a|obs(s))} R(s,a)$ over π
- $R(s,a)$ or $R(session)$ is a black box
  - special case: $R(s,a) = r(s,a) + \gamma R(s', a')$
- Markov property: $\mathbb{P}(s'|s,a,*) = \mathbb{P}(s'|s,a)$
- Special case: $obs(s) = s$, fully observable

Reinforcement Learning for seq2seq

encoder - decoder api / translate / score / weight

self-critical policy gradient

$$\nabla J = \mathbb{E}_{x \sim p(s)} \mathbb{E}_{y \sim \pi(y|x)} \nabla \log \pi(y|x) \cdot (R(x,s) - b(x))$$

R - negative levenshtein distance

$b(x) = R(x, y_{greedy}(x))$