Deep Reinforcement Learning Methods for Recommender Systems

ADVANCED DEEP LEARNING COURSE PROJECT

BY BRIEUC DE VOGHEL

AGENDA

- Recommender System overview
- Reinforcement Learning overview
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed Deep Deterministic Policy Gradient (TD3)
- Experimental results of application to Recommender System
- Conclusion

Recommender System overview

- **Goal : suggesting items that best match users' preferences**
- Huge practical applications across the board : search engines, streaming services, news, …
- Shown to be of sequential nature
  - can be formulated as a Markov Decision Process
  - RL employed to solve it
- Based on user-item interactions
  - features describing the users and items
  - interaction events data: explicit & implicit (ratings, reviews, queries, clicks, views, device, location, …)
- Multiple challenges:
  - dynamic nature of the problem
  - very sparse interactions
  - large action space
  - "cold start problem" (new items w/o interaction history)

  $\Rightarrow$ specific RL requirements

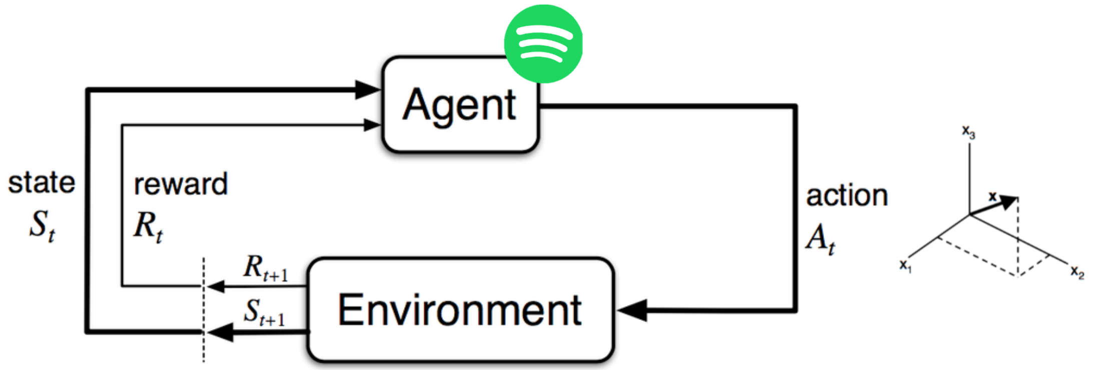
Afsar, M. M., Crump, T., & Far, B. (2021). Reinforcement learning based recommender systems: A survey. arXiv preprint arXiv:2101.06286. ; Sequential nature : [50, 51]

Reinforcement Learning for Recommender Systems

- **Goal : selecting best action (recommendation) to maximize reward (clicks, views, purchases, …)**

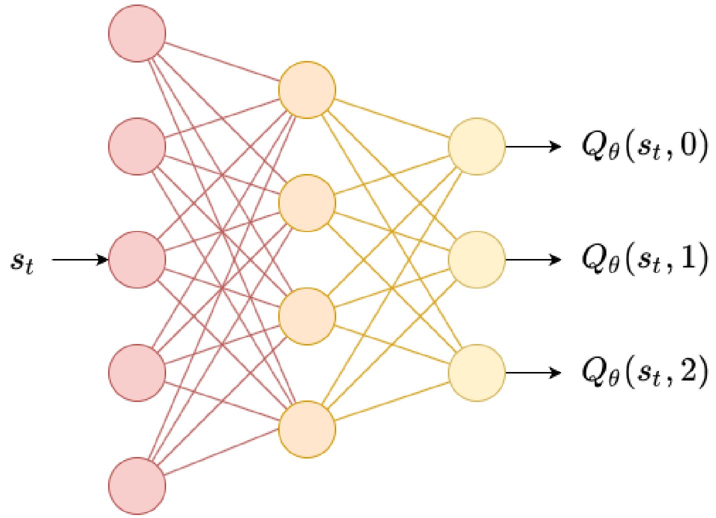
- **Agent** : recommender system / algorithm
- **Environment** : user interacting with recommendations
- **Reward** : number of clicks
- **State** : updated user representation according to behavior
- **Action** : feature vector of predicted favorite features to be found in suggested items

- Once modeled for specific problem :
  - state is embedding of user profile w/ browsing history
  - dot product of action with item features allow to list preferred items; top-k are presented
  - user model and action model (unknown to agent) allow to mimic the behavior of users in the environment



Reinforcement Learning overview

- Q-Learning : Q table where each state has a value for all possible actions
- DQN introduces parametrization by neural network with weights : $Q(s,a) \approx Q(s,a,\theta)$
- Probabilistic method : Q network predicts probability of selecting each action
  - select action by finding $\max_a(Q(s,a))$
- Limitation : only handle discrete & low-dimensional action spaces (one output neuron per action)

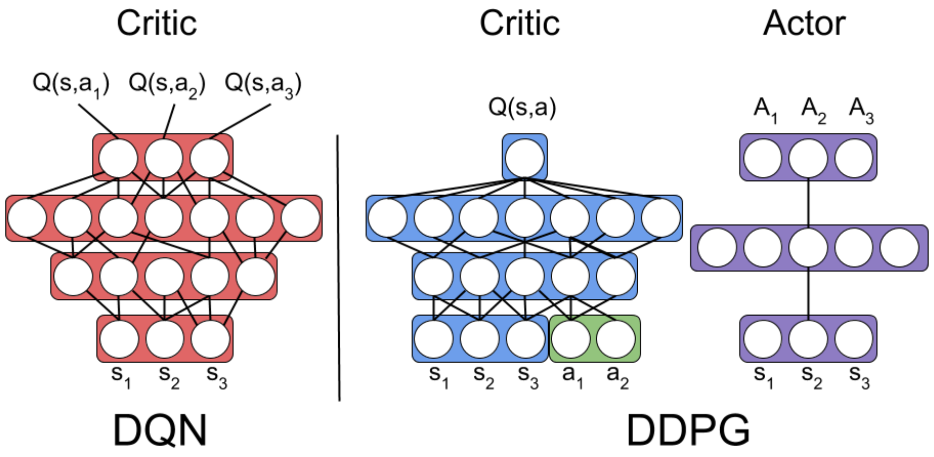


Deep Q Network with s as input and Q values per action as output (probabilities of taking the action)

Deep Deterministic Policy Gradient (DDPG)

DDPG: Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N.M., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2016). Continuous control with deep reinforcement learning. CoRR, abs/1509.02971.

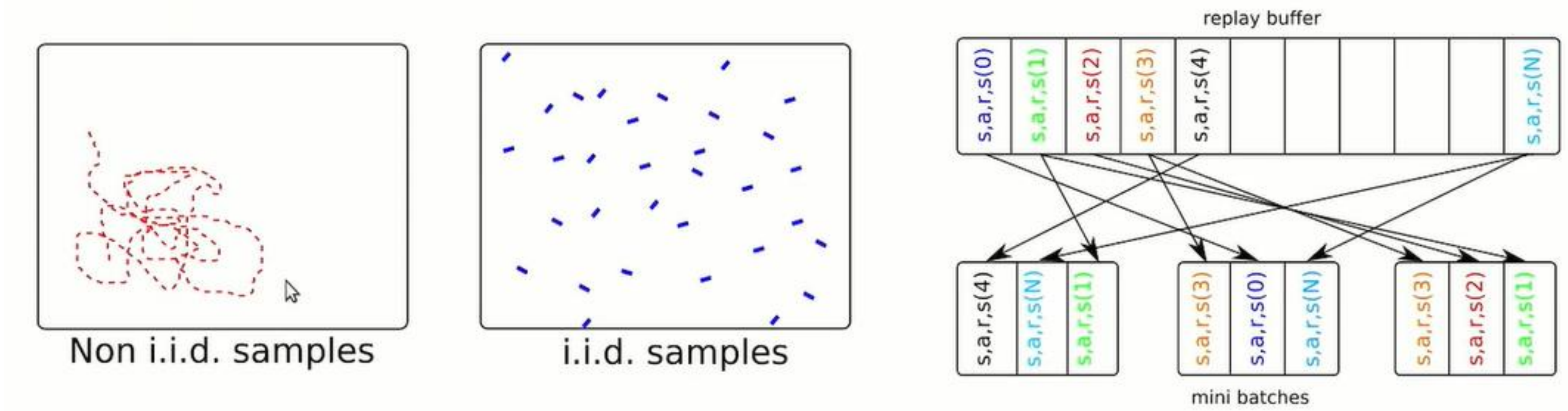
- Methods and techniques from :
  - DQN
    - replay buffer
    - shuffling
    - target Q-network
  - Deterministic Policy Gradient (DPG)
  - Batch normalization (not used : proved inconclusive)
- Deterministic : direct action prediction (vs stochastic : probability distribution)
- Off-policy learning : use of replay buffer
- Actor $\pi_\theta(a_t|s_t)$ - Critic $\hat{Q}_\phi^\pi(s_t, a_t)$
  - actor : policy-based
  - critic : value-based (measures how good the action is)
- All updates based on SDG



DDPG : Replay buffer & Shuffling

- Replay buffer stores experienced sequences of *<state, action, reward, next_state>*
  - used for training of the actor and critic networks (see later)

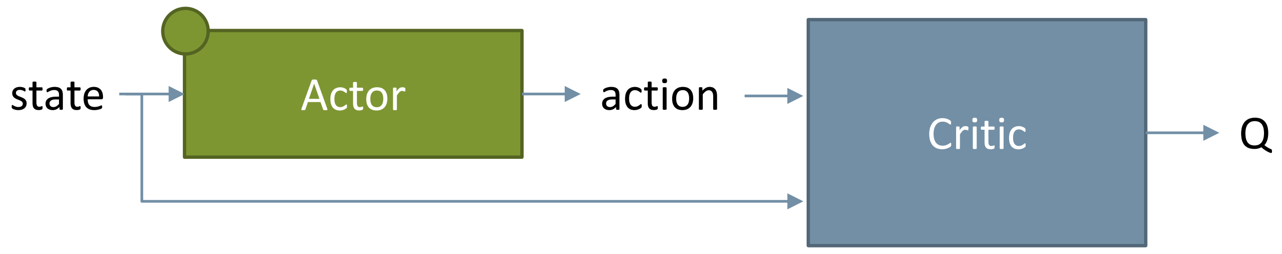
- Agent’s experiences are not sampled independently and identically distributed (i.i.d.)
  - optimization algo assumes i.i.d. samples for weight update
  - ⟹ agent learns on mini-batches, rather than online
  - improves sample efficiency



  - different replay buffer management strategies (random sampling, prioritized experience replay, …) are optimal in different problems (is problem dependent)

DDPG : Training the Actor

- Assuming we have an already trained critic
- Using observed state from the replay buffer :

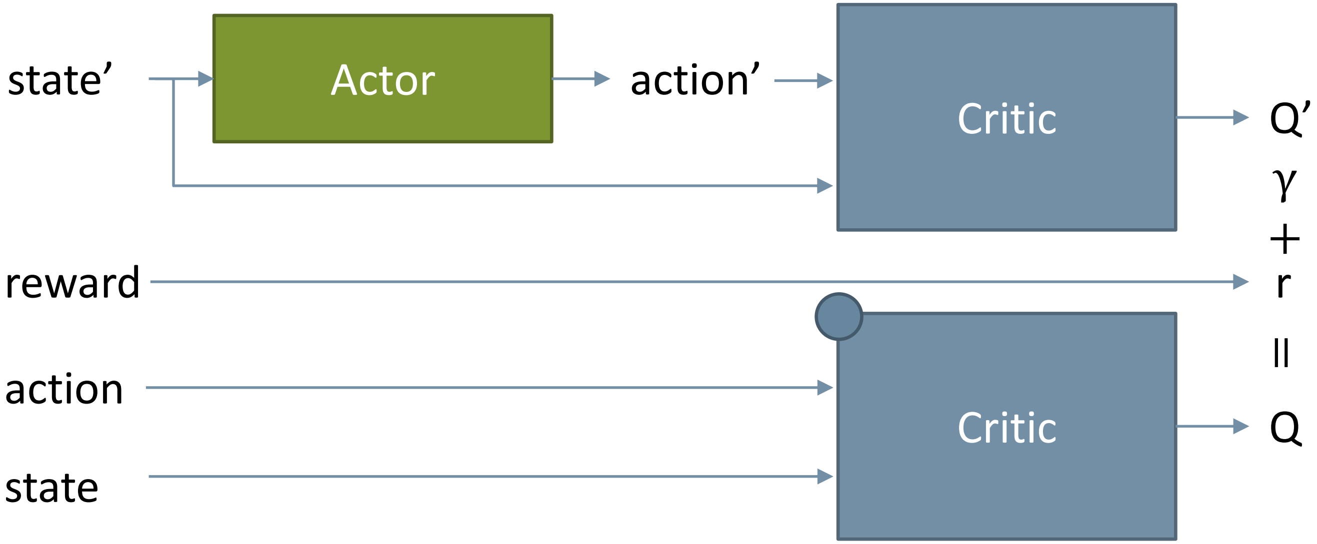


    - maximize predicted Q

- Weights of actor updated to follow the gradient of the Q value wrt the actions
    - actions are inputs of critic, but roles are symmetric for weights and parameters

DDPG : Training the Critic

- Assuming we have an already trained actor
- Using observed state, action, reward, and future state from the replay buffer :



  - MSE of temporal difference error $\delta = (r + \gamma\, Q') - Q$ to be minimized

DDPG : Target network

- Tabular case : each Q-value is updated separately and independently
- Continuous state and action setting : interdependencies exist between target updates (everything is a function of Q)

- Critic learning : minimize MSE between target and predicted value (i.e. temporal difference (TD) error)
- Target $y_t = r_t + \gamma \max_a (Q^{\pi}_{\phi}(s_{t+1}, a)|\phi)$ is itself a function of Q^{π}_{ϕ}
  - leads to unstable behavior

  $\Rightarrow$ Key idea : "periods of supervised learning"
  - Compute loss function from separate *target critic* $\mathrm{Q'}^{\pi}_{\phi'}$ (loss used to update actual critic)
  - Soft update target network $\phi' \leftarrow (1-\tau)\phi' + \tau\phi$ with gain $\tau \ll 1$

- Similar idea for Actor

DDPG : Exploration

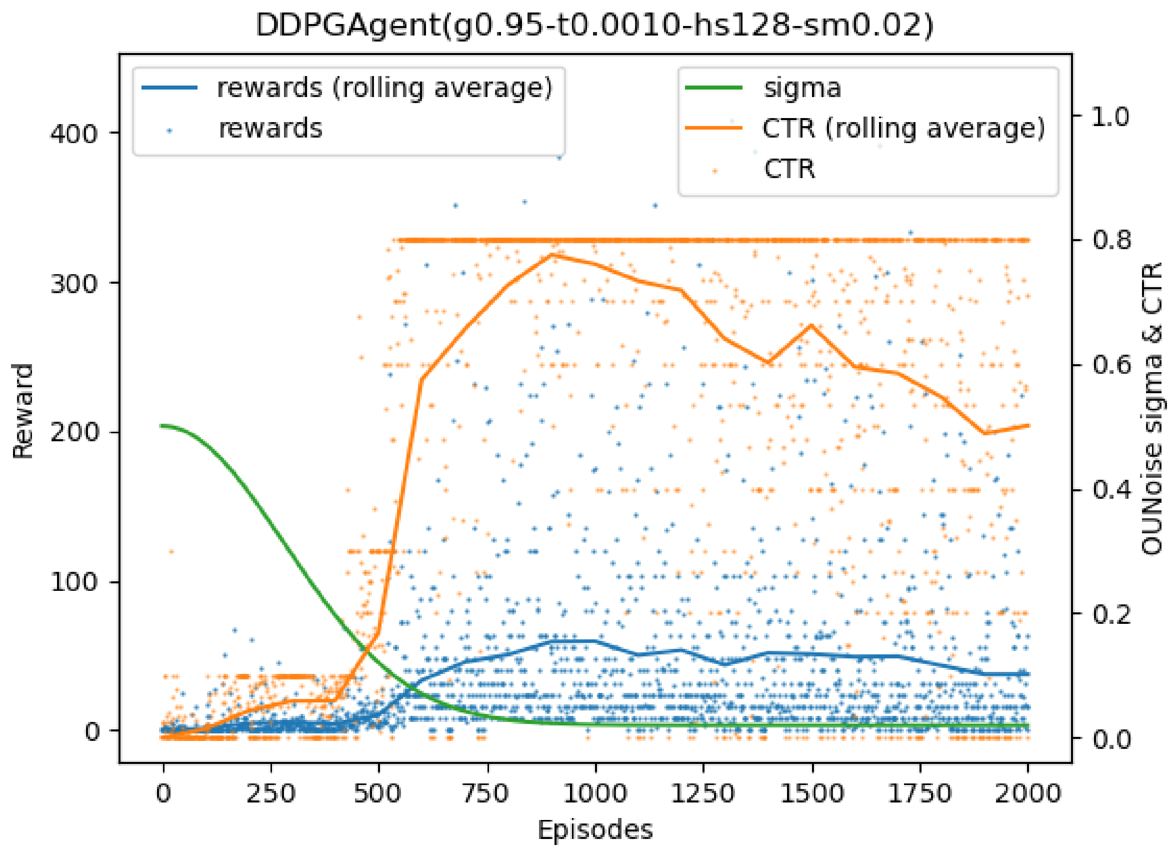
- To allow the agent to understand the observation space and action space, exploration must be allowed
- At initialization : random actions (due to noisy init)

- Adding noise
  - to action : variability in selected action
    - Ornstein–Uhlenbeck process to generate temporally correlated noise with inertia
  - to observation (state) : parameter noise

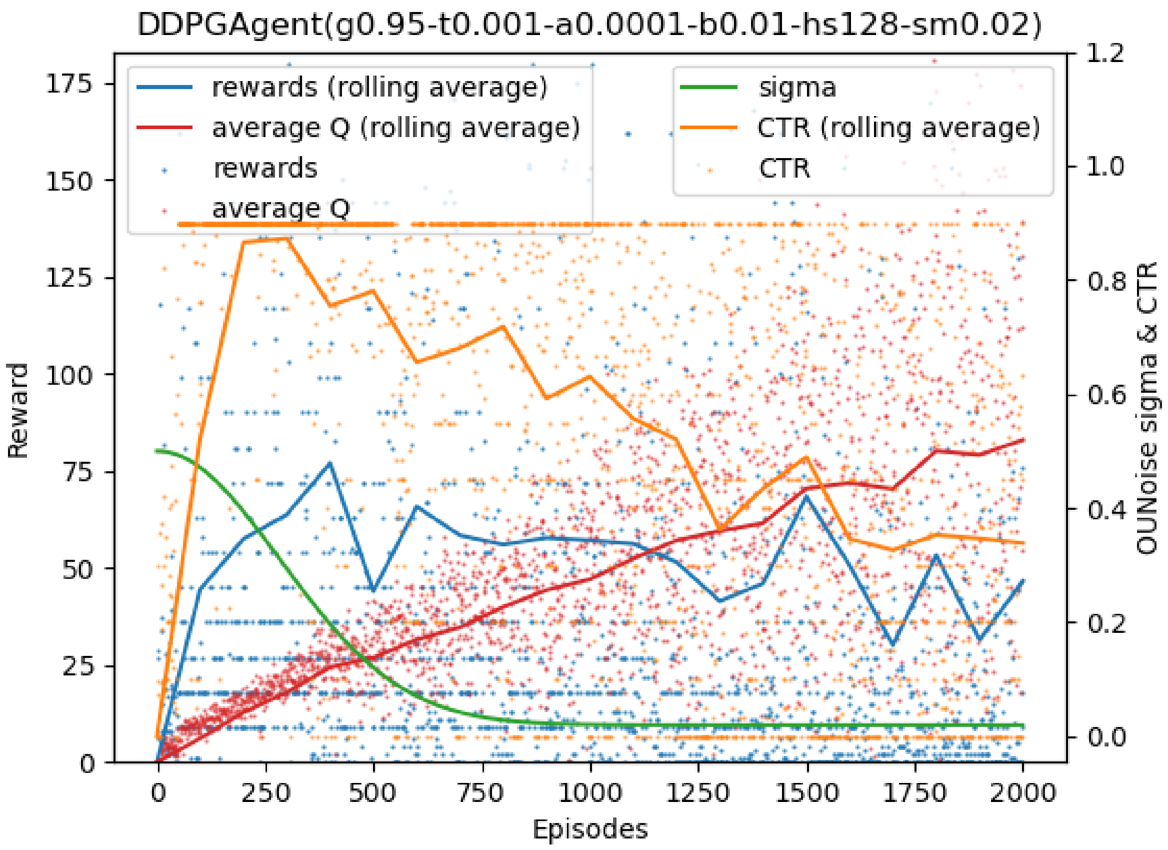
DDPG : Limitations

- As descendant of Q-learning : suffers from over-estimation bias
  - slows down learning as Q needs to converge after having over-estimated performances
- Estimation errors build up over time :
  - falling into a local optima
  - experience catastrophic forgetting
- Sensitive to hyper-parameter tuning

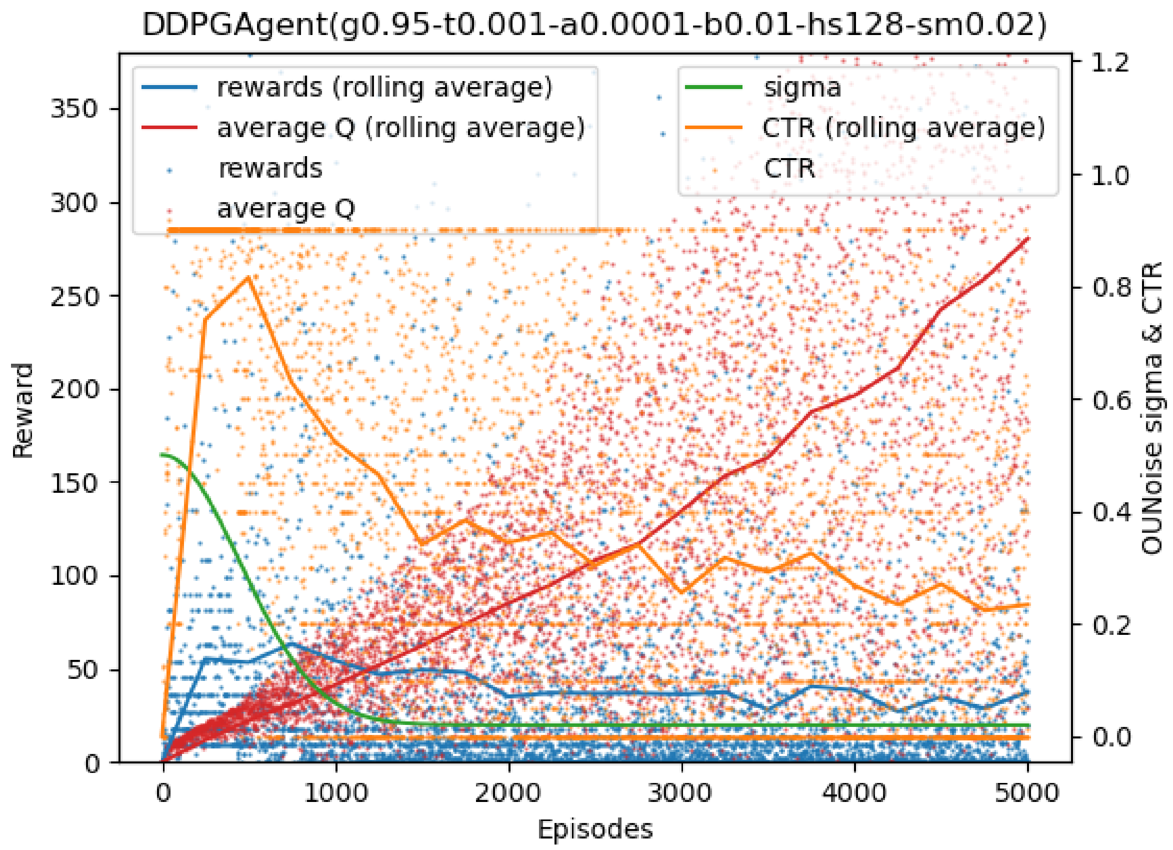
Experimental results of application to Recommender System



Experimental results of application to Recommender System



Experimental results of application to Recommender System



Twin Delayed DDPG (TD3)

TD3 : Fujimoto, S., Hoof, H. & Meger, D.. (2018). Addressing Function Approximation Error in Actor-Critic Methods. Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research 80:1587-1596

- Improvement over DDPG tackling some of its limitations
- Inspired by Double Q-Learning
  - pair of independently trained critics : smallest value estimation is used
  - suggest to clip value estimate : upper-bound favors underestimation (which is not propagated)
    - need to have problem-specific knowledge
  - based on empirical evaluation
- Introduction of delayed policy updates
  - policy less frequently updated in comparison to value network
    - updates on more stable value prediction
  - allows for better (lower variance) value estimation and prevents actor fooling

Double Q-Learning : Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

TD3 : Addressing overestimation bias

- Overestimated values will propagate to neighboring states (through Bellman equation)
  - due to max operator in Q-learning
    - less clear in actor-critic setting : due to gradient direction being a local maximizer
  - tend to exploit over-estimated states until value lowers
    - agent will lose a lot of time exploring these over-estimated states
  - Not the case for under-estimation
    - agent won't propagate as other states are more valuable

- Inaccurate value estimate may lead to poop policy updates
  - feedback loop is created
  - suboptimal actions will be higher rated by suboptimal critic

- Solution suggested by Double Q-Learning :
  - lowest of two independent value estimates will be "lesser of two evils"

TD3 : Addressing variance build-up

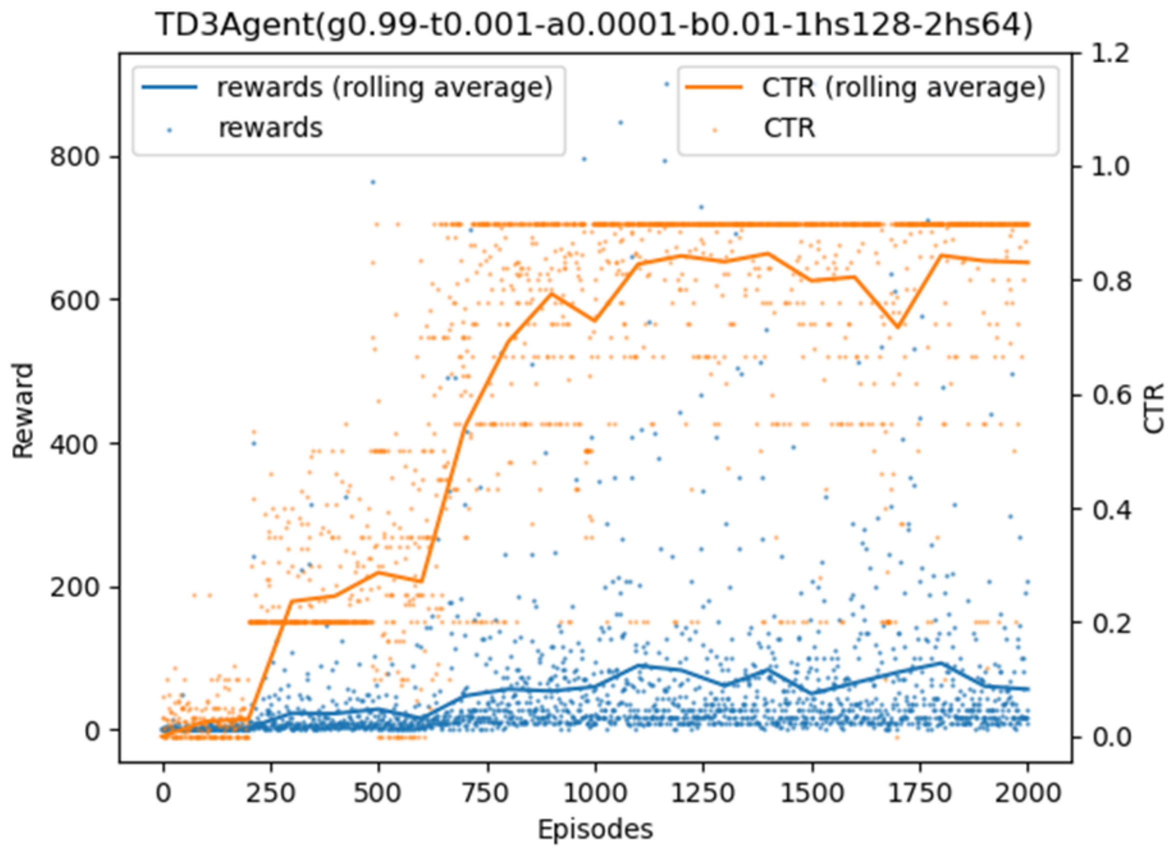
- Introduction of delayed policy updates (discussed above)

- Target policy smoothing regularization : “similar actions should have similar values”
  - TD3 suggests to fit the value of small area around the target action

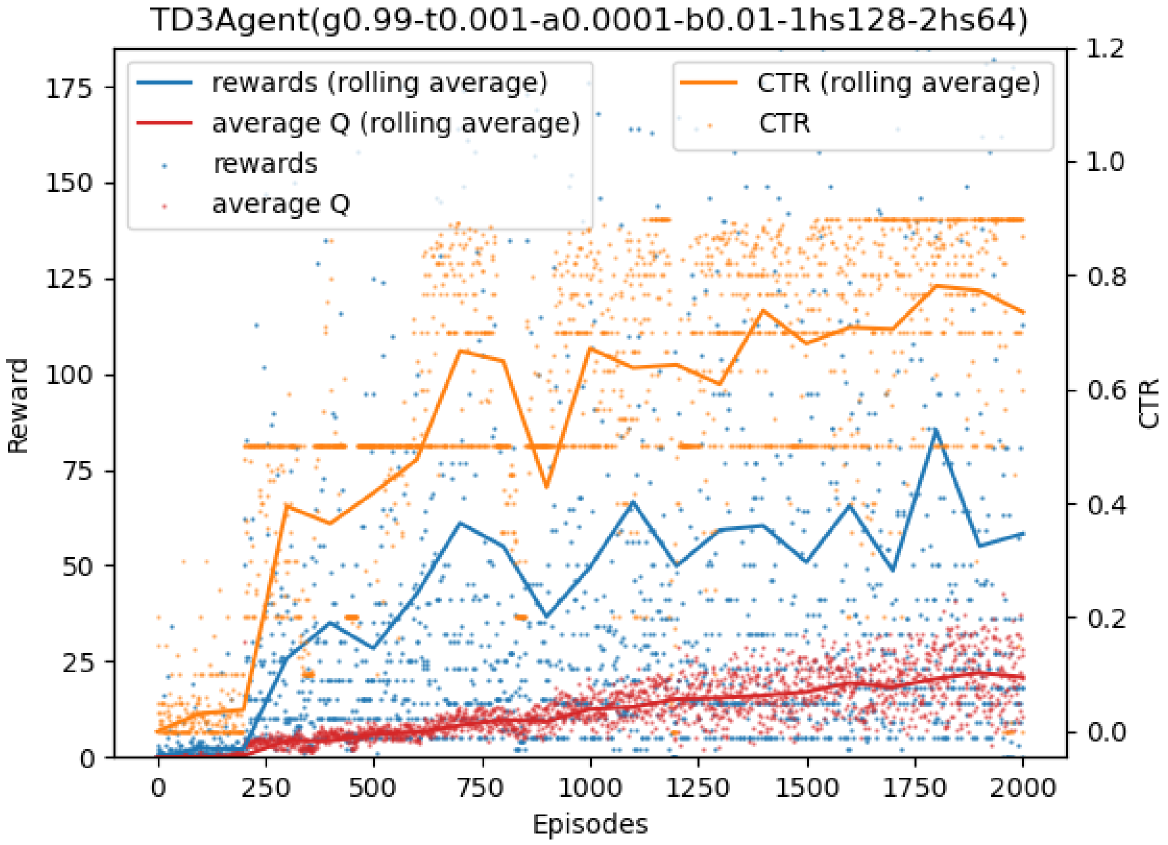
$$y = r + \mathbb{E}_{\varepsilon}[Q_{\phi}(s', \pi(s') + \varepsilon)]$$

    - in practice : approximate expectation over actions by adding noise to the target policy

Experimental results of application to Recommender System



Experimental results of application to Recommender System



Experimental results of application to Recommender System

- Both DDPG and TD3 algorithms are applied to a recommender problem
- Virtual-Taobao : real-world online retail environment
- Metric used : click-trough-rate (CTR)
  - good indication of suggesting appropriate and relevant recommendations
  - rewards alone can be misleading : do not represent how many suggestions had to be made

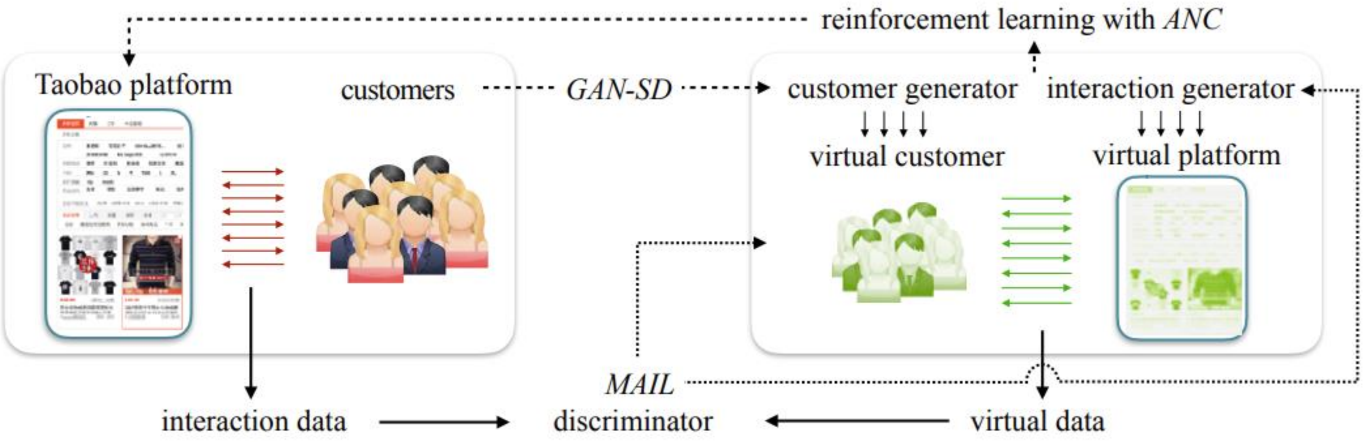


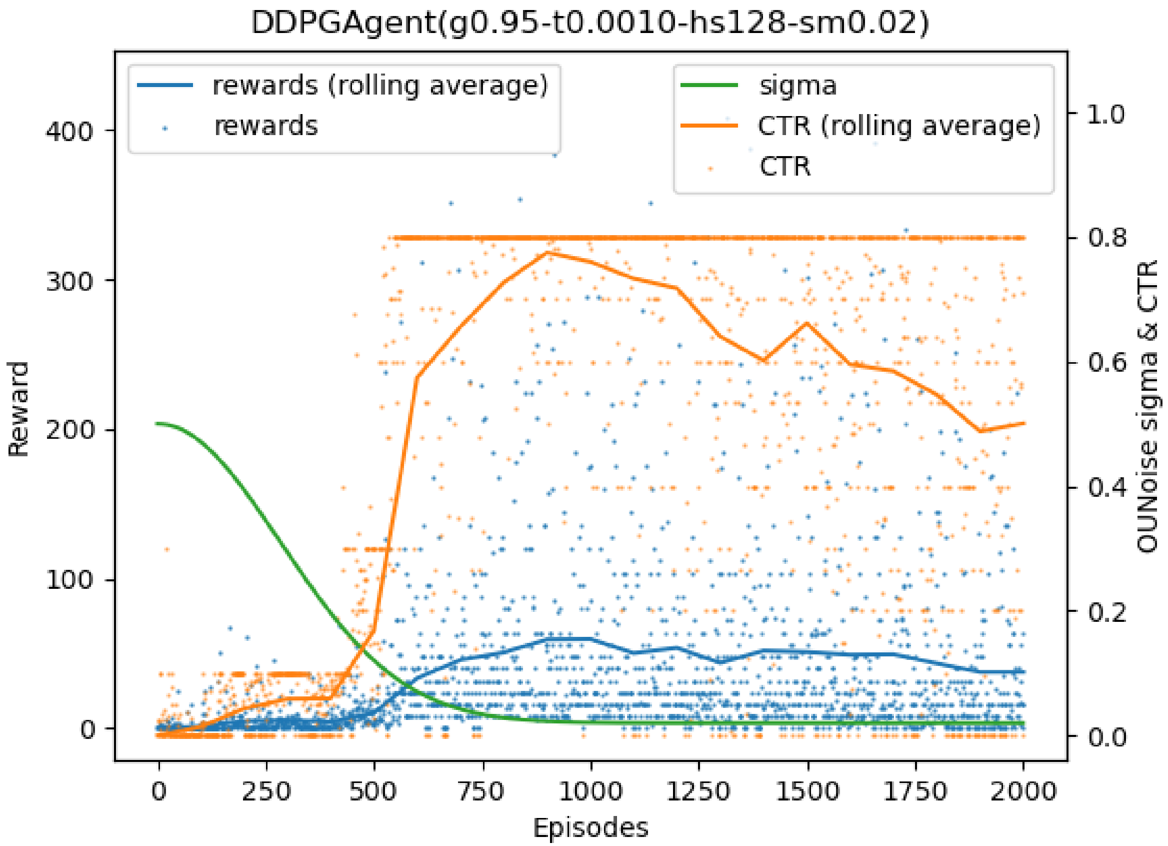
Figure 1: Virtual-Taobao architecture for reinforcement learning.

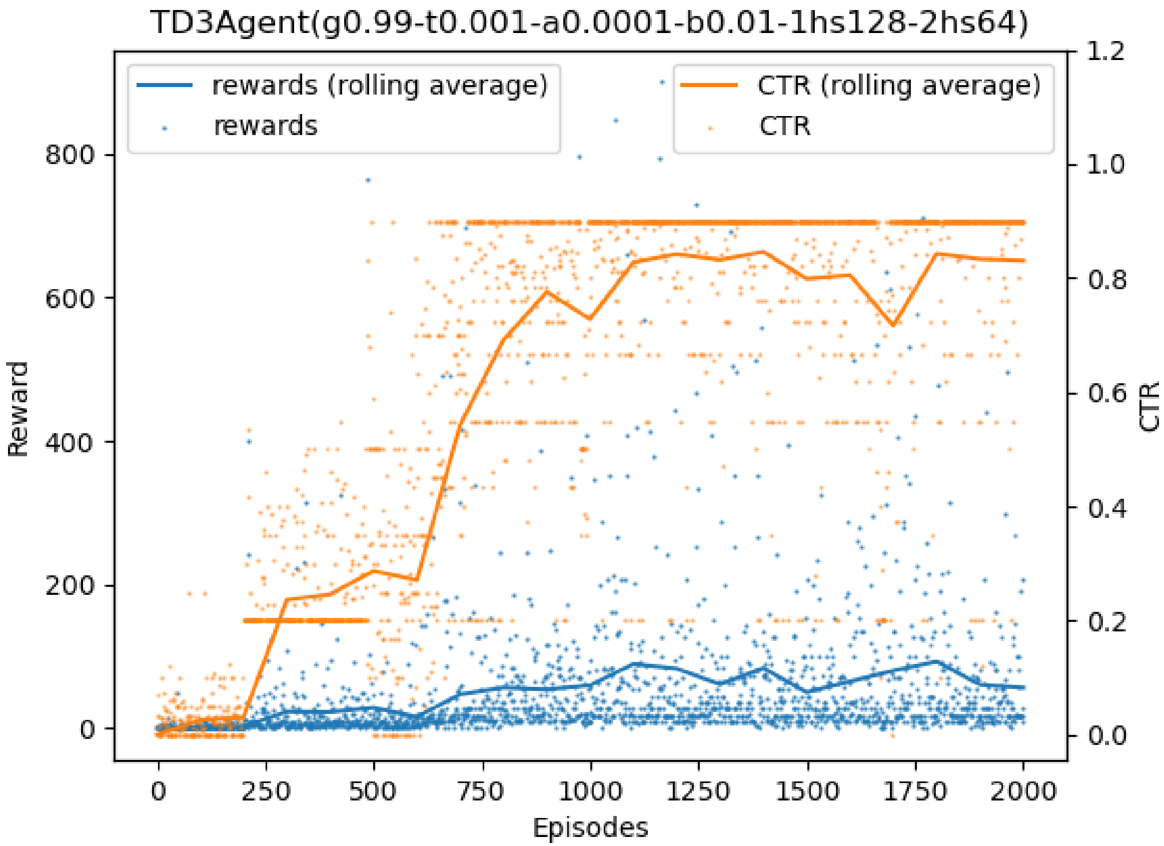
Experimental results : Setup & Hyper-parameters

- DDPG
  - actor lr : 0.0001
  - critic lr : 0.01
  - gamma (discount factor) : 0.95
  - tau : 0.001
  - hidden layer size : 128
  - memory sample size : 100
  - OU-noise sigma : 0.5 w/ decay over 1000 timesteps
- TD3
  - actor lr : 0.0001
  - critic lr : 0.01
  - gamma (discount factor) : 0.99
  - tau : 0.001
  - hidden layer size : 128, 64
  - memory sample size : 400
  - Gaussian noise $\mathcal{N}(0,0.1)$
- Trained over 2000 episodes
- Implementation in PyTorch, trained on local machine

Virtual-Taobao : Shi, J. C., Yu, Y., Da, Q., Chen, S. Y., & Zeng, A. X. (2019, July). Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, No. 01, pp. 4902-4909).

Experimental results of application to Recommender System





Conclusion

- Application of Deep RL techniques on the Recommender Problem
  - DDPG
  - TD3
- Identified key problems of DDPG and addressed by TD3
  - experimental results demonstrate benefits of suggested improvements

Further improvements

- Session-based recommendation (history of previous recommendations)
- Improve ability to learn long-term rewards
- Explore different replay buffer management strategies (random sampling, prioritized experience replay, …)
  - used now : random sampling : samples do not correspond to an agent trajectory

Reinforcement Learning

- Goal : learn a policy π which maximizes expected reward

- Expected reward after taking action a_t in state s_t following policy π is described by action-value function
  $Q^{\pi}(s_t, a_t) = \mathbb{E}[R_t \mid s_t, a\_t]$

- Recursive relationship by the Bellman equation :

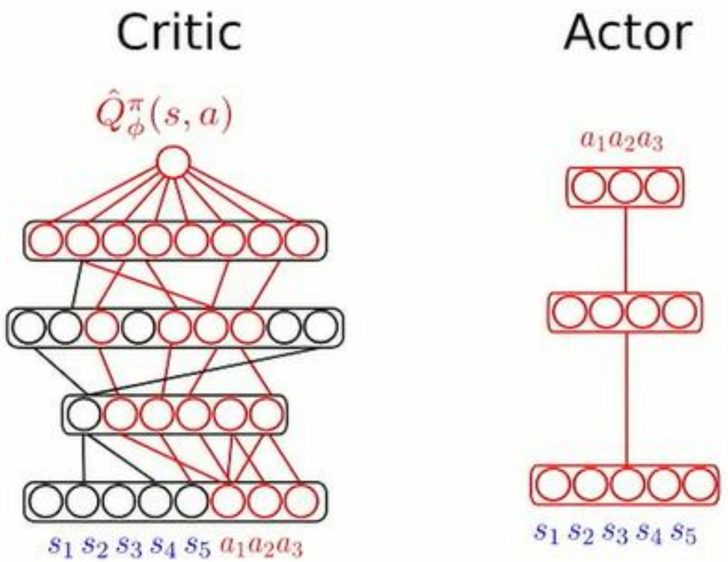
$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}}(Q^{\pi}(s_{t+1}, a_{t+1}))$$

- Q-Learning : Q table where each state has a value for all possible actions

state / action	a_0	a_1	a_2	a_3
$\mathbf{s}_0$	0.66	0.88*	0.81	0.73
$\mathbf{s}_1$	0.73	0.63	0.9*	0.43
$\mathbf{s}_2$	0.73	0.9	0.95*	0.73
$\mathbf{s}_3$	0.81	0.9	1.0*	0.81
$\mathbf{s}_4$	0.81	1.0*	0.81	0.9
$\mathbf{s}_5$	0.9	1.0*	0.0	0.9

- Off vs On policy learning :
  - off-policy : refers to learning about one way of behaving, called the target policy, from data generated by another way of selecting actions, called the behavior policy.
    - reusing old data (e.g. replay buffer) : sample efficiency
    - more freedom for exploration
    - able to learn from demonstration (imitation)
    - allow for transfer learning

DDPG : Training the Actor



- Deterministic policy gradient theorem: the true policy gradient is

$$\nabla_{\boldsymbol{\theta}}\pi(\mathbf{s}_t,\mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_t,\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}(.)}[\nabla_a \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t,\mathbf{a}_t)\nabla_{\boldsymbol{\theta}}\pi(s|\boldsymbol{\theta})]$$

- $\nabla_a \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t,\mathbf{a}_t)$ is used as error signal to update the actor weights.
- Comes from NFQCA
- $\nabla_a \hat{Q}^{\pi_{\boldsymbol{\theta}}}_{\boldsymbol{\phi}}(\mathbf{s}_t,\mathbf{a}_t)$ is a gradient over actions
- $y = f(w.x + b)$ (symmetric roles of weights and inputs)
- Gradient over actions $\sim$ gradient over weights