Deep RL

Deep RL

Deep RL = RL + neural networks as function approximators

The methods described in this chapter
find an optimal policy for an
unknown, continuous MDP

- Unknown: states/actions/transitions are not known a-priori
  → agent must rely on trial-and-error (run episodes and see how well it goes
- Continuous: Too many states/actions to list them in tabular form

Four methods are presented

- Deep Sarsa, Deep Q-learning
- DQN
- Actor-critic

Deep RL

Different types of unknown, discrete/continous MDPs
(with corresponding algorithm and example)

	discrete action space	continuous action space
discrete state space	SARSA / Q-learning / MC (robot maze)	actor-critic
continuous state space	Deep Sarsa, Deep Q-Learning, DQN (Breakout, Space Invaders)	actor-critic (autonomous driving)

Deep RL

Motivation/explanation 1 for Deep Sarsa, Deep Q-learning

- Recap: SARSA update rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(R + \gamma Q(s',a') - Q(s,a)\big)$$

- Recap: Q-learning update rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

- What happens after we calculated the optimal Q-function?
→ There is a unique optimal Q-function, so it will not change
→ SARSA / Q-learning update rules will be of the form

$$Q(s,a) \leftarrow Q(s,a)$$

→ the other terms in the SARSA/Q-learning update rule must be zero, e.g. it is for SARSA

$$R + \gamma Q(s',a') - Q(s,a) = 0$$

Deep RL

Motivation/explanation 2 for Deep Sarsa, Deep Q-learning

- Approximate the true Q-function $Q(s,a)$ with a neural network $\hat{Q}(s,a,\boldsymbol{w})$
- Minimize loss L measuring the difference between true and approximated Q-function through gradient descent
  - Define loss function

$$L = \left(Q(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)^2$$

  - Calculate gradient

$$\nabla_{\mathbf{w}} L = -2\left(Q(s,a) - \hat{Q}(s,a,\boldsymbol{w})\right)\nabla_{\mathbf{w}}\hat{Q}(s,a,\boldsymbol{w})$$

  - Use approximation of the true Q-function, e.g. for SARSA

$$\nabla_{\mathbf{w}} L = -2\left(R + \gamma\hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)\nabla_{\mathbf{w}}\hat{Q}(s,a,\boldsymbol{w})$$

  - Update weights $\boldsymbol{w}$ through (stochastic) gradient descent

Deep RL

Deep Sarsa

- Train a neural network to approximate the true Q-function $Q(s,a)$ with $\hat{Q}(s,a,\boldsymbol{w})$
- Loss L to be minimized is of the form

$$L = \left(R + \gamma\hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)^2$$

- If the loss becomes zero for all states/actions, the optimal Q-function is obtained (Q-function won't be updated anymore)

$$\hat{Q}(s,a,\boldsymbol{w}) \leftarrow \hat{Q}(s,a,\boldsymbol{w}) + \alpha \xcancel{\left(R + \gamma\hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)}$$

- Resulting gradient becomes

$$\nabla_{\mathbf{w}} L = -2\left(R + \gamma\hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)\nabla_{\mathbf{w}}\hat{Q}(s,a,\boldsymbol{w})$$

Deep RL

Deep Q-learning

- Train a neural network to approximate the true Q-function $Q(s,a)$ with $\hat{Q}(s,a,\boldsymbol{w})$
- Loss L to be minimized is of the form

$$L = \left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)^2$$

- If the loss becomes zero for all states/actions, the optimal Q-function is obtained (Q-function won't be updated anymore)

$$\hat{Q}(s,a,\boldsymbol{w}) \leftarrow \hat{Q}(s,a,\boldsymbol{w}) + \alpha \xcancel{\left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)}$$

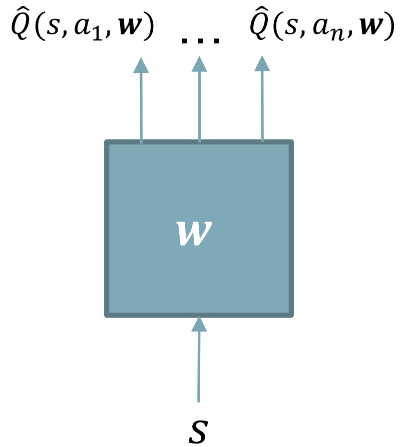
- Resulting gradient becomes

$$\nabla_{\mathbf{w}} L = -2\left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right) \nabla_{\mathbf{w}} \hat{Q}(s,a,\boldsymbol{w})$$

Deep RL

How to train Deep Sarsa / Deep Q-learning

1. Run certain number of episodes with neural network (type shown below)
2. Store data of the form (s, a, r, s', a') for every time step in a buffer
3. Train neural network to minimize the loss L, then clear the buffer and go to 1.
   (Neural network has one output per action, representing its probability)



Note that finding a ϵ-greedy/optimal policy based on the obtained values $\hat{Q}(s, a_1, \boldsymbol{w}) \ldots \hat{Q}(s, a_n, \boldsymbol{w})$ is easy, just pick the action with the largest value

Deep RL

DQN (Deep Q Network)

- Deep Q-learning with improvements:
  - experience replay
  - target network
- first "proper" deep RL approach
  (deep SARSA and deep Q-learning are just for didactic purpose)

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b

Deep RL

Experience replay

Idea: Use all data obtained so far for training

Training steps

1. Run certain number of episodes with neural network
2. Store data of the form (s, a, r, s', a') for every time step in a buffer
3. Train neural network to minimize the loss L, t~~hen clear the buffer~~ and go to 1.

Implementation

- buffer often has finite size, delete oldest data if buffer is full
- training often happens only on a (random) subset of all buffer data
  (faster if buffer is large)

Deep RL

Effect of experience replay:

- Is possible, because Q-learning is off-policy
- Each observed transition (s, a, r, s', a') can be used multiple times for training
  → better data efficiency
- History contains data from many different policies
  → lots of different samples seen

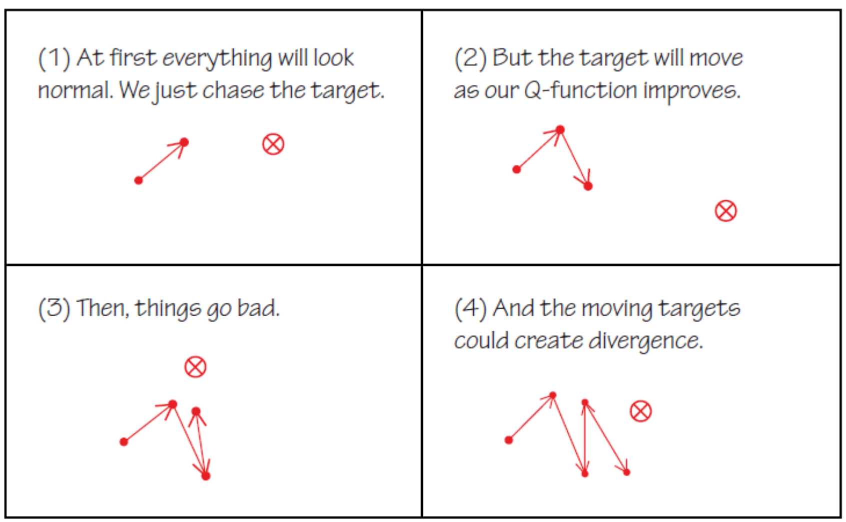
Task: What does off-policy mean?

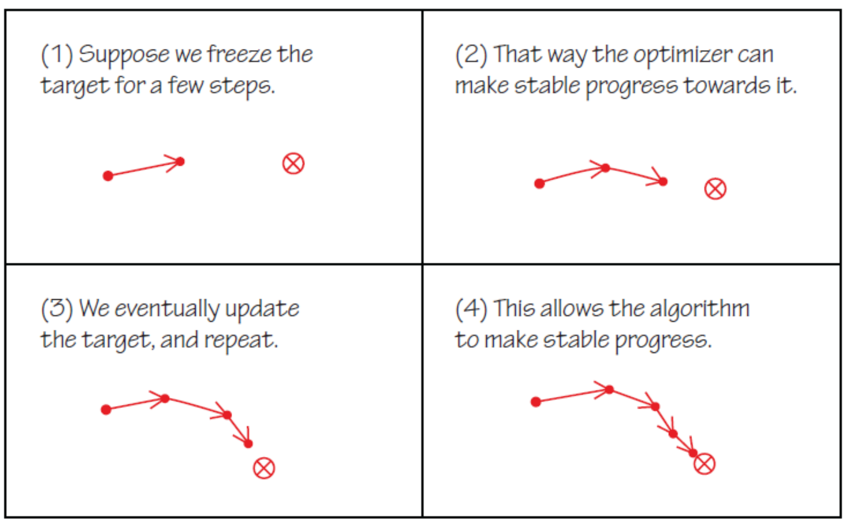
Deep RL

Target network

Idea: Fix the orange term (target network) for a certain number of training steps to avoid unstable behaviour

$$\hat{Q}(s,a,\boldsymbol{w}) \leftarrow \hat{Q}(s,a,\boldsymbol{w}) + \alpha\left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}) - \hat{Q}(s,a,\boldsymbol{w})\right)$$





https://livebook.manning.com/concept/reinforcement-learning/target-network

Deep RL

Effect of target networks:

- By using target networks, the goal $R + \gamma \max_{a\prime} \hat{Q}(s', a', \boldsymbol{w})$ that should be learned by $\hat{Q}(s, a, \boldsymbol{w})$ in order to minimize

$$L = \left(R + \gamma \max_{a\prime} \hat{Q}(s', a', \boldsymbol{w}') - \hat{Q}(s, a, \boldsymbol{w}) \right)^2$$

  changes only slowly → more stable learning

Deep RL

Training steps with experience replay and target network

1. Initialization: Create two identical neural networks $\hat{Q}(s, a, \boldsymbol{w})$ and $\hat{Q}(s, a, \boldsymbol{w}')$
2. Repeat for a certain number of times
   a) Run certain number of episodes (policy based on neural network $\hat{Q}(s, a, \boldsymbol{w})$)
   b) Store data of the form (s, a, r, s', a') for every time step in a buffer
   c) Train neural network $\hat{Q}(s, a, \boldsymbol{w})$ to minimize the loss L

   $$L = \left(R + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}') - \hat{Q}(s, a, \boldsymbol{w})\right)^2$$

   This corresponds to the update rule

   $$\hat{Q}(s, a, \boldsymbol{w}) \leftarrow \hat{Q}(s, a, \boldsymbol{w}) + \alpha \left(R + \gamma \max_{a'} \hat{Q}(s', a', \boldsymbol{w}') - \hat{Q}(s, a, \boldsymbol{w})\right)$$

3. Copy $\hat{Q}(s, a, \boldsymbol{w})$ to obtain $\hat{Q}(s, a, \boldsymbol{w}')$ and go to 2.

Deep RL

Task: Why is there no Deep SARSA with experience replay and target networks?

Deep RL

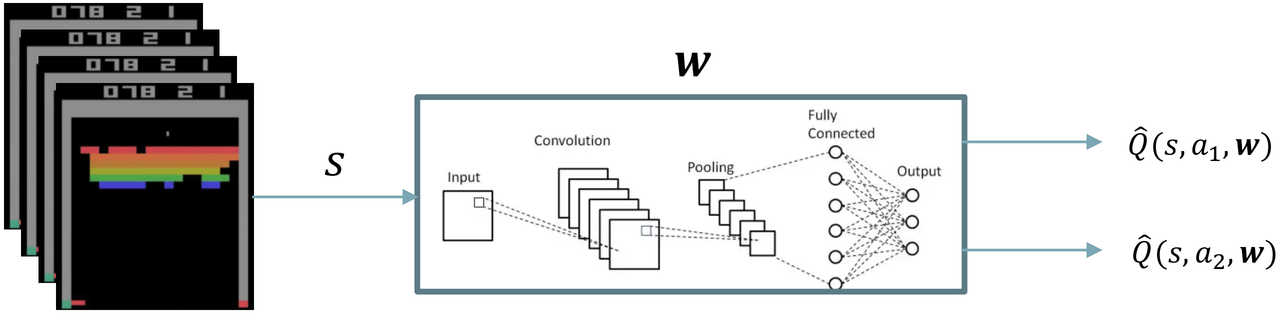
Example: Playing Atari with Deep RL (2013)

- "inventor" of DQN
- breakthrough as it uses raw images as input
- no domain knowledge
  → generalizes to other tasks
- 2015: presented in Nature
- paved the road for many modern deep RL techniques
- successor: DDQN

https://www.youtube.com/watch?v=TmPfTpjtdgg

Deep RL

- function approximator is a convolutional neural network
- state is encoded by the last four images



Task: What do the two outputs $\hat{Q}(s, a_1, \boldsymbol{w})$ and $\hat{Q}(s, a_2, \boldsymbol{w})$ represent?

Task: Why are the last four images used to represent the state (and not just a single one)?

Deep RL

Ostbayerische Technische Hochschule Amberg-Weiden

DDQN (Double Deep Q Networks)

- Problem of DQN (similar to Q-Learning): max-operator in

$$\hat{Q}(s,a,\boldsymbol{w}) \leftarrow \hat{Q}(s,a,\boldsymbol{w}) + \alpha\left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}') - \hat{Q}(s,a,\boldsymbol{w})\right)$$

  may overestimate the true Q-value if $\hat{Q}$ is inaccurate

- There are two situations where this has an influence
  - selecting an action during episodes
  - rating an action during training

- Idea of DDQN: Decouple these two steps in to reduce the effect of a wrong estimate (use two different networks)

Deep RL

- Original update rule (target)

$$\hat{Q}(s,a,\boldsymbol{w}) \leftarrow \hat{Q}(s,a,\boldsymbol{w}) + \alpha\left(R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}') - \hat{Q}(s,a,\boldsymbol{w})\right)$$

- DQN has the target

$$R + \gamma \max_{a'} \hat{Q}(s',a',\boldsymbol{w}')$$

$$R + \gamma \hat{Q}\left(s', \operatorname*{argmax}_{a'} \hat{Q}(s',a',\boldsymbol{w}'), \boldsymbol{w}'\right)$$

- DDQN has the target

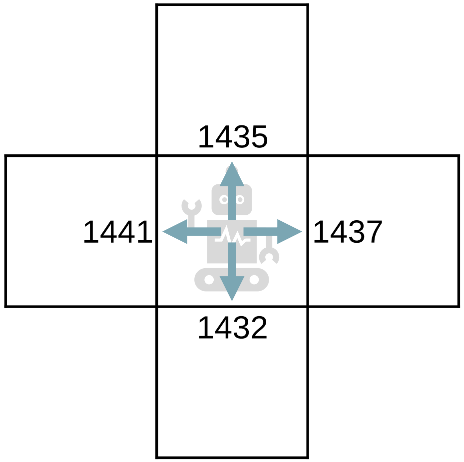
$$R + \gamma \hat{Q}\left(s', \operatorname*{argmax}_{a'} \hat{Q}(s',a',\boldsymbol{w}'), \boldsymbol{w}\right)$$

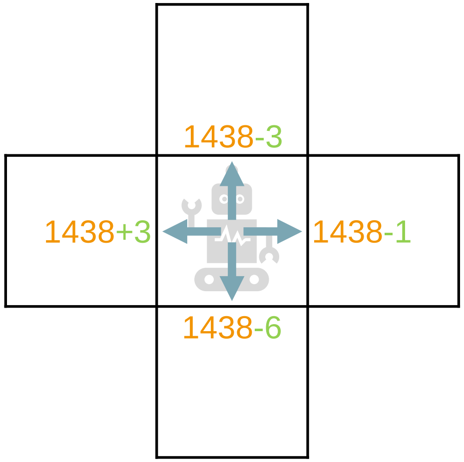
Deep RL

Dueling (D)DQN

- Problem of DQN/DDQN: The absolute Q-values are not that important when choosing an action, it is more their relative value w.r.t. each other
- Idea: Split up the learned Q-function into two different networks:
  - One network can learn the absolute value
  - The other network can learn the relative value for different actions
- Formula used in paper

$$\hat{Q}(s,a,\boldsymbol{w}) = V(s,\boldsymbol{w}) + \left(A(s,a,\boldsymbol{w}') - \max_a A(s,a,\boldsymbol{w}')\right)$$

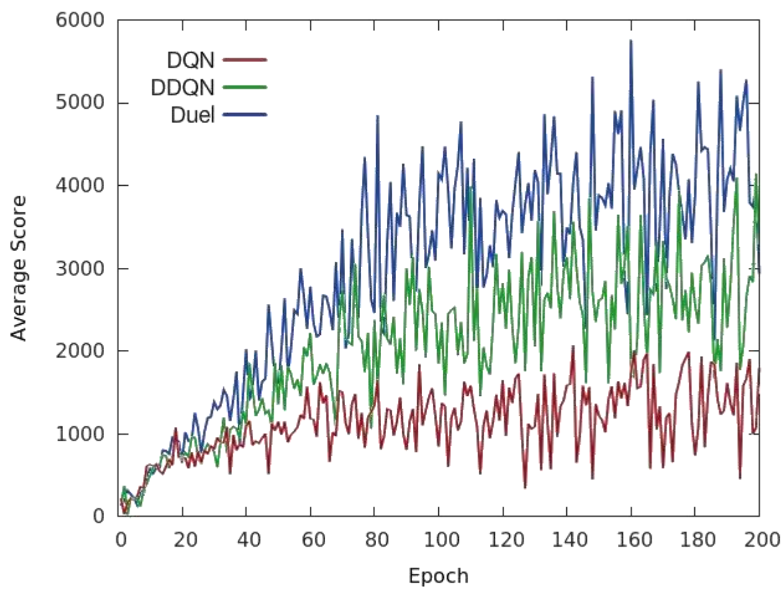




Deep RL

Example: Space Invaders

- performance of DQN / DDQN / Dueling DQN over trained epoch

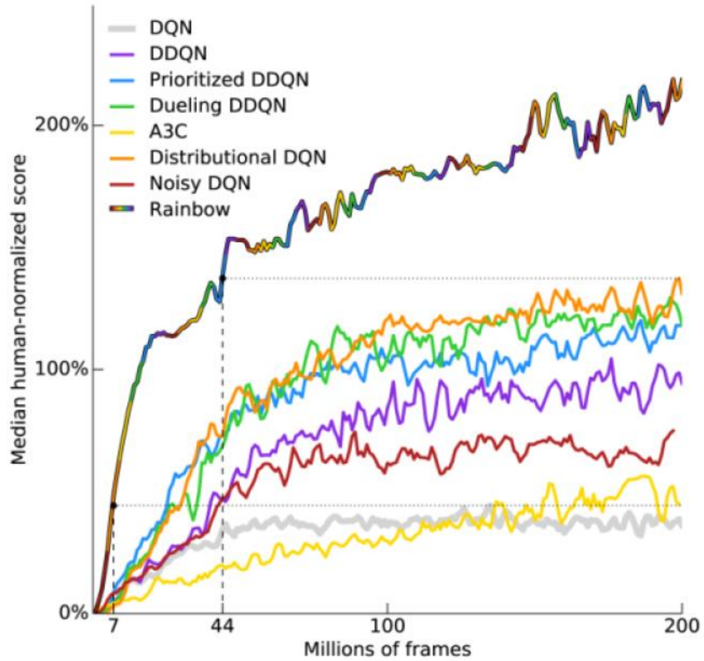


http://torch.ch/blog/2016/04/30/dueling_dqn.html

Deep RL

Rainbow

- Idea: Because different DQN variants excel for certain problems, iterate over all of them during training



https://www.arxiv-vanity.com/papers/1710.02298/

Deep RL

Evolution of RL algorithms

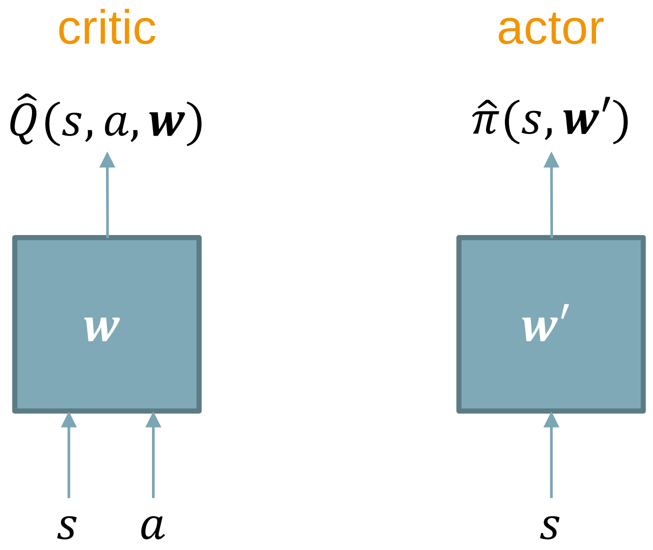
	Bellman optimality equation		Neural networks		Experience replay / target networks		Split up target network		Split up Q-function		Mixture of algorithms	
MC evaluation	→	Q-Learning	→	Deep Q-Learning	→	DQN	→	DDQN	→	Dueling DDQN	→	Rainbow

Task: What are the improvements for each RL algorithm?

Deep RL

Actor-critic methods

- are used mainly for continuous state / continuous action MDPs
  ($\longleftrightarrow$ DQN: continuous state / discrete action)
- rely on two neural networks
  - critic: approximate optimal (V-)/Q-function based on current state/action
  - actor: approximate optimal policy based on current state

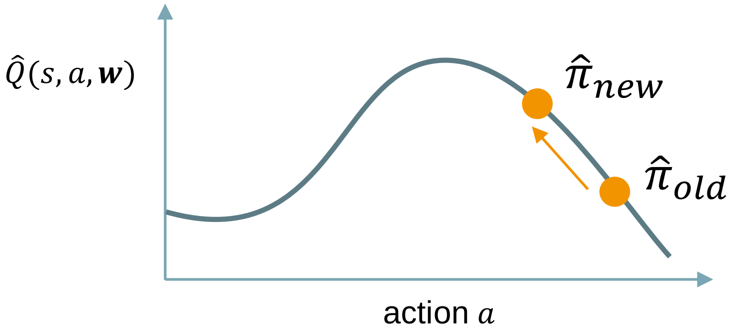


Deep RL

- the critic $\hat{Q}(s, a, \boldsymbol{w})$ is trained to learn the optimal (V-)/Q-function, e.g. by minimizing the loss L

$$L = \left(R + \gamma \hat{Q}(s', a', \boldsymbol{w}) - \hat{Q}(s, a, \boldsymbol{w})\right)^2$$

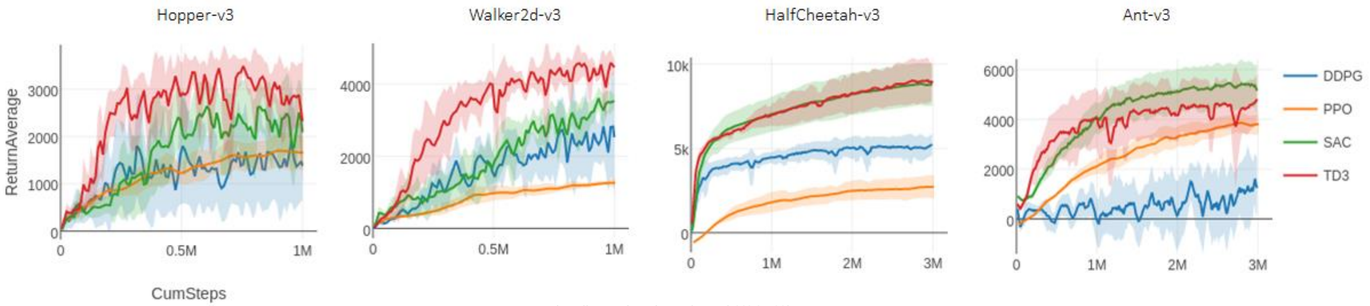
- the actor $\hat{\pi}(s, \boldsymbol{w}')$ is trained through gradient ascent on the critic to output the optimal action for a given state such that the critic output $\hat{Q}(s, \hat{\pi}(s, \boldsymbol{w}'), \boldsymbol{w})$ is maximized



Deep RL

State-of-the-art RL algorithms

- more additional improvements (similar to experience replay / target networks) lead to state-of-the-art algorithms (e.g. A2C, A3C, DDPG, PPO, TRPO, TD3, SAC, …)
- rule of thumb: The newer the algorithm is, the better it performs (learns faster)
- Overview of different algorithms:
  - https://lilianweng.github.io/posts/2018-04-08-policy-gradient/
  - https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1
  - https://www.ias.informatik.tu-darmstadt.de/uploads/Team/DavideTateo/felix_thesis.pdf



https://www.arxiv-vanity.com/papers/1909.01500/

Deep RL

Brief summary

- Deep RL is used whenever the state/action space is continuous
- Deep Sarsa, Deep Q-learning are not used in practice
- DQN is used for continuous state / discrete action spaces
- DQN approximates the Q-function with a neural network
- Actor-critic methods are used for continuous state / continuous action spaces
- Actor-critic methods approximate the Q-function and the policy with neural networks

https://www.freecodecamp.org/news/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d/

Kahoot!