Autonomous Systems: Deep Learning

Deep Reinforcement Learning Introduction

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf

Heilbronn University of Applied Sciences

Outline

1. Deep Q-Learning
2. Task 1: 2D Pole Cart
3. DQN Improvements
4. Task 2: DQN Improvements
5. Policy Gradient
6. Actor Critic Methods
7. Task 3: Deep Deterministic Policy Gradient (DDPG)

1. Deep Q-Learning

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf

Heilbronn University of Applied Sciences

From Tabular Methods to Deep Methods

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

250 states

https://spiele.rtl.de/kartenspiele/black-jack.html

10^{70802} states

https://www.retrogames.cz/play_222-Atari2600.php

$7.7 * 10^{45}$ states

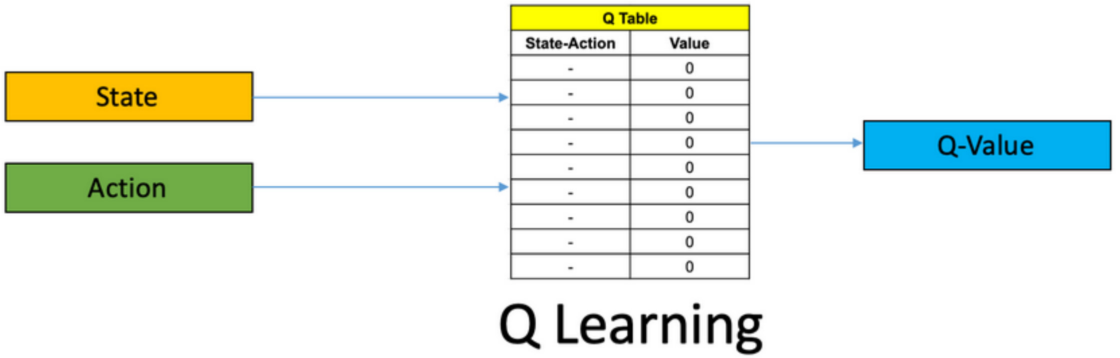
https://i.ytimg.com/vi/VHqCAaFXpbc/maxresdefault.jpg

From Tabular Methods to Deep Methods

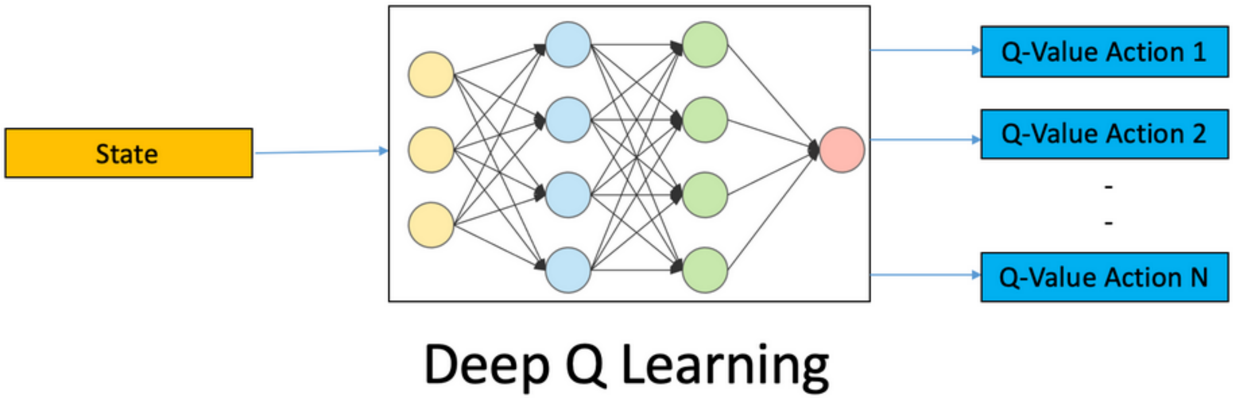
HEILBRONN UNIVERSITY OF APPLIED SCIENCES

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

$$y = R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \qquad L = (Q(S_t, A_t) - y)^2 \qquad (2)$$



State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0



https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

Naive Algorithm

HEILBRONN UNIVERSITY
OF APPLIED SCIENCES

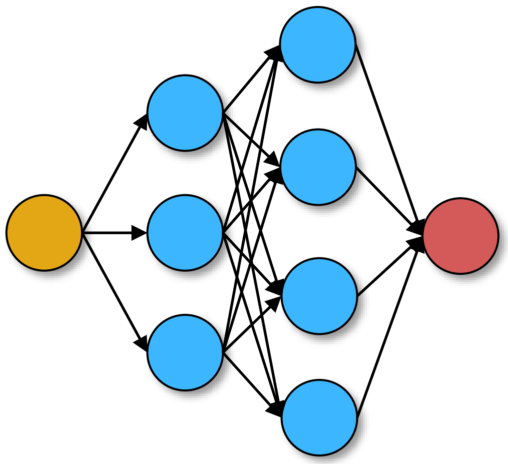
1. Initialize $Q(s, a)$ with some initial approximation.
2. By interacting with the environment, obtain the tuple (s, a, r, s').
3. Calculate loss: $\mathcal{L} = (Q(s,a) - r)^2$ if the episode has ended, or
   $\mathcal{L} = \left(Q(s,a) - \left(r + \gamma \max_{a' \in A} Q_{s',a'}\right)\right)^2$ otherwise.
4. Update $Q(s, a)$ using the **stochastic gradient descent** (**SGD**) algorithm, by minimizing the loss with respect to the model parameters.
5. Repeat from step 2 until converged.

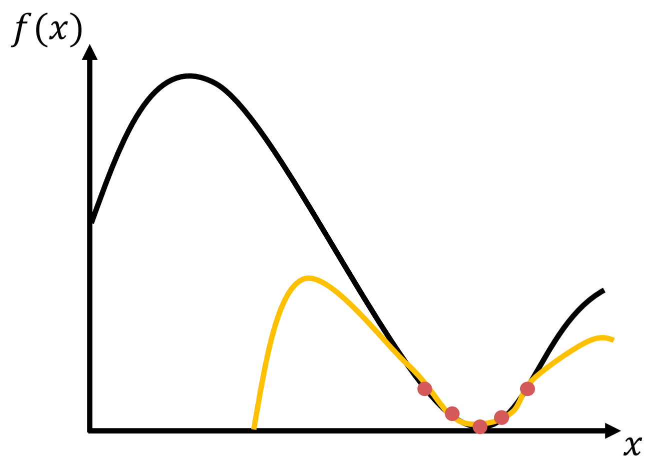
https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Naive Algorithm

PROBLEMS

- Exploration vs. Exploitation Dilemma:
  ➔ **Epsilon-Greedy Algorithm**
- Markov Property (**partially observable MDPs**)
  ➔ **State Stack**
- SGD optimization:
  Training data needs to be ***independent and identically distributed.***
  ➔ **Replay Buffer**





https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Naive Algorithm

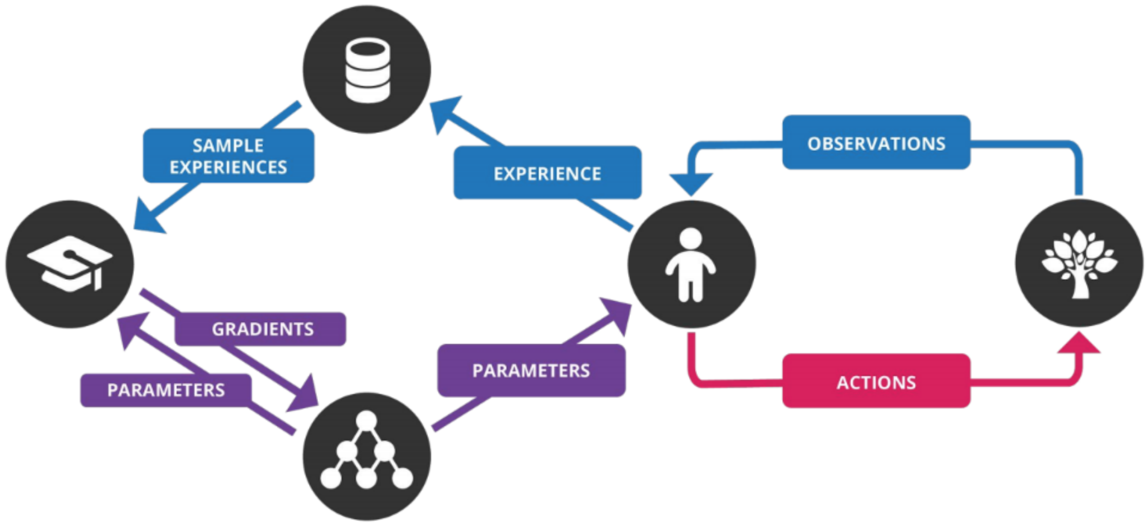
PROBLEMS

- Exploration vs. Exploitation Dilemma:
  ➔ **Epsilon-Greedy Algorithm**
- Markov Property (**partially observable MDPs**)
  ➔ **State Stack**
- SGD optimization:
  *Training data needs to be **independent and identically distributed.***
  ➔ **Replay Buffer**
- Correlation between steps:
  We're training $Q(s,a)$ via $Q(s',a')$ (bootstrapping). When we perform an update of our NN's parameters to make $Q(s,a)$, we can indirectly alter the value produced for $Q(s',a')$ and other states nearby.
  ➔ **Target Network**

https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Final Algorithm

$$y_i = r_i + \gamma \max_{a'} \hat{Q}(s_i', a') \qquad L = \frac{1}{N}\sum_i (Q(s_i, a_i) - y_i)^2 \tag{3}$$



https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

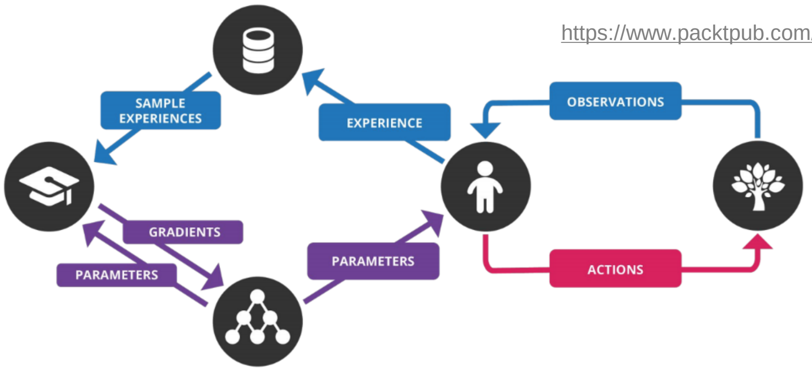
Final Algorithm

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

The algorithm for DQN from the preceding papers has the following steps:

1. Initialize the parameters for $Q(s,a)$ and $\hat{Q}(s,a)$ with random weights, $\varepsilon \leftarrow 1.0$, and empty the replay buffer.
2. With probability ε, select a random action, a; otherwise, $a = \arg\max_a Q(s,a)$.
3. Execute action a in an emulator and observe the reward, r, and the next state, s'.
4. Store transition (s, a, r, s') in the replay buffer.
5. Sample a random mini-batch of transitions from the replay buffer.
6. For every transition in the buffer, calculate target $y = r$ if the episode has ended at this step, or $y = r + \gamma \max_{a' \in A} \hat{Q}(s', a')$ otherwise.
7. Calculate loss: $\mathcal{L} = (Q(s,a) - y)^2$.
8. Update $Q(s, a)$ using the SGD algorithm by minimizing the loss in respect to the model parameters.
9. Every N steps, copy weights from Q to $\hat{Q}$.
10. Repeat from step 2 until converged.

https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition



https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

2. Task 1: 2D Pole Cart

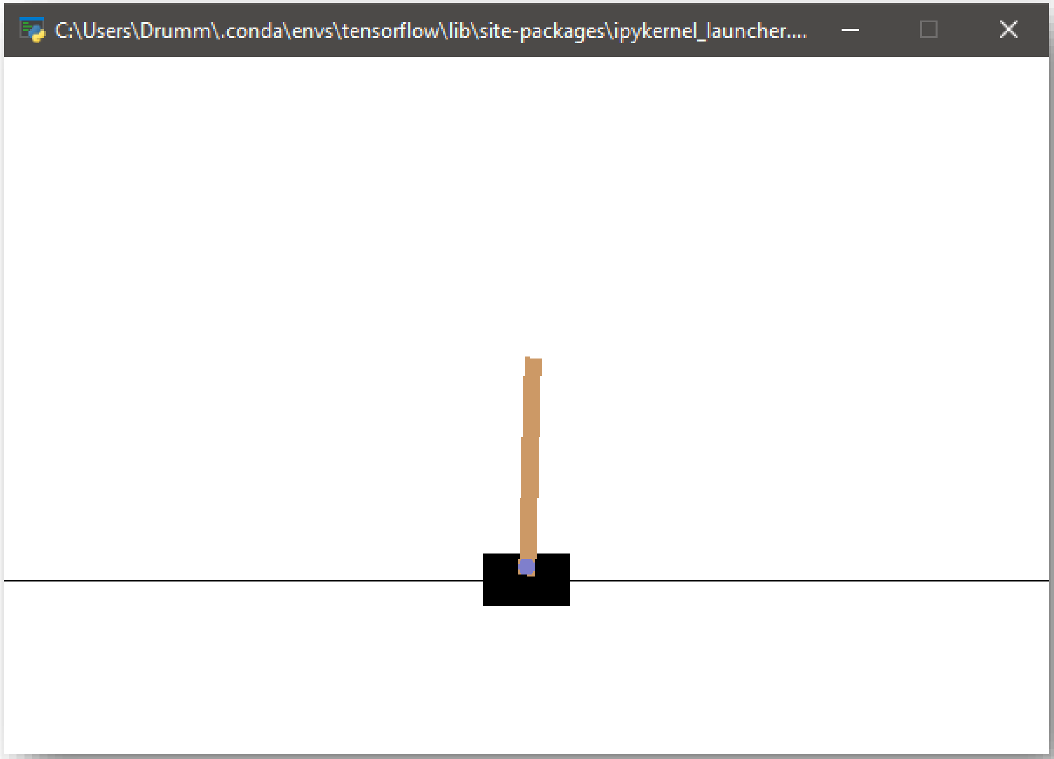
HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf
Heilbronn University of Applied Sciences

2D Pole Cart

Task:

Implement a Deep Q-Learning Algorithm (DQN) to balance a pole on a cart in two dimensions by moving the cart left and right. (Additionally, try to implement some of the improvement techniques discussed before and compare training performance.)



3. DQN Improvements

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf
Heilbronn University of Applied Sciences

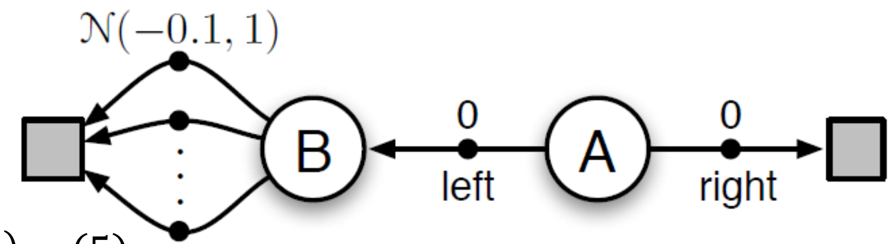
Improvements

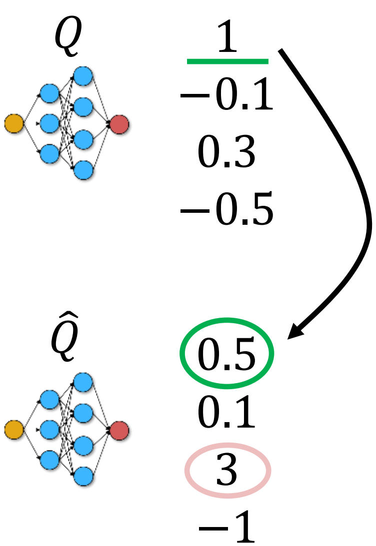
MAXIMIZATION BIAS AND DOUBLE LEARNING

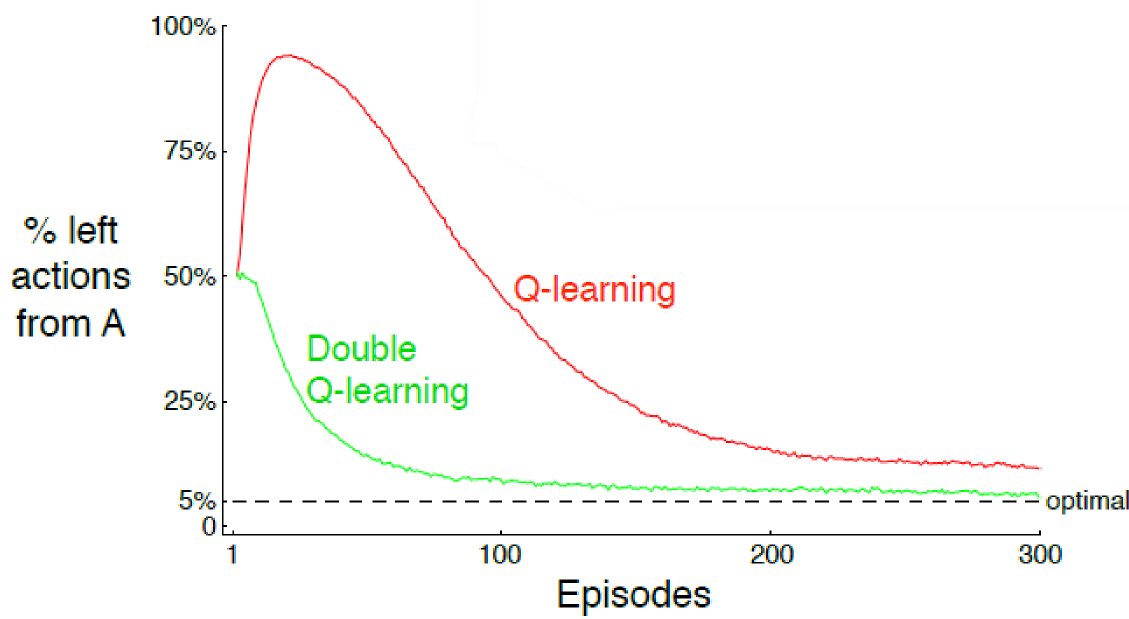
$$Target \leftarrow R_{t+1} + \gamma \boxed{\max_a} \hat{Q}(S_{t+1}, a) \qquad (4)$$

$$Target \leftarrow R_{t+1} + \gamma \hat{Q}\big(S_{t+1}, argmax_a Q(S_{t+1}, a)\big) \qquad (5)$$

https://arxiv.org/abs/1509.06461







http://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

Improvements

NOISY NETWORKS

- Classical DQN achieves exploration with the hyperparameter epsilon, which is slowly decreased over time.
- Instead, the authors add a noise to the weights of fully-connected layers of the network and adjust the parameters of this noise during training using backpropagation.
- For every weight in a fully-connected layer, we have a random value that we draw from the normal distribution. Parameters of the noise μ and σ are stored inside the layer and get trained using backpropagation, the same way that we train weights of the standard linear layer.

https://arxiv.org/abs/1706.10295

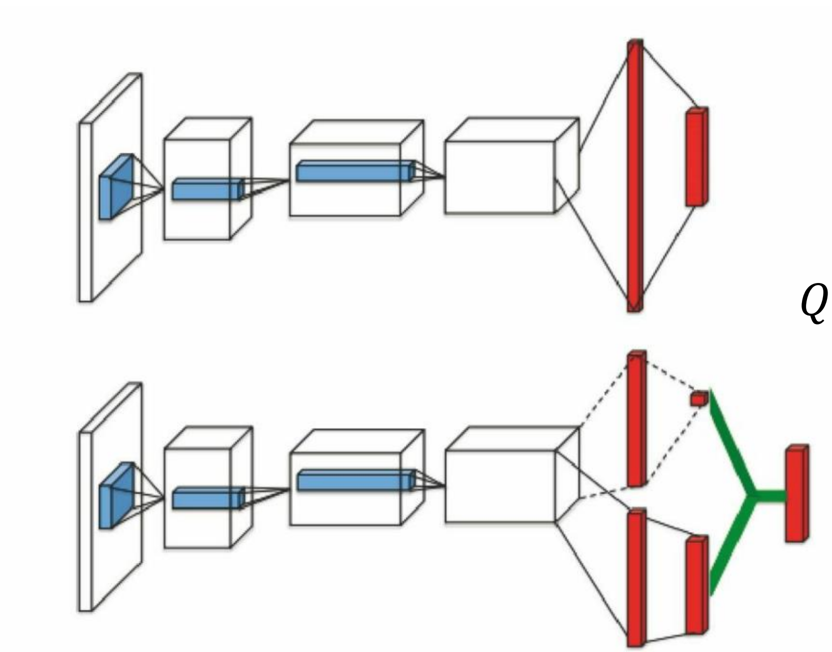
https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Improvements

DUELING DQN

$$Q(S_t, A_t) \quad = V(S_t) + \ A(S_t, A_t) \qquad (6)$$

Action-Value *State-Value* *Advantage*



- The key motivation behind this architecture is that for some games, it is unnecessary to know the value of each action at every timestep.
- By explicitly separating two estimators, the dueling architecture can learn which states are (or are not) valuable, without having to learn the effect of each action for each state.
- Problem: The naive sum of the two is "unidentifiable," in that given the Q value, we cannot recover the V and A uniquely.

$$Q(S_t, A_t) = V(S_t) + \ A(S_t, A_t) - \frac{1}{N}\sum_k A(S_t, k) \quad (7)$$

https://arxiv.org/abs/1511.06581

https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

4. Task 2: DQN Improvements

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf
Heilbronn University of Applied Sciences

5. Policy Gradient

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf

Heilbronn University of Applied Sciences

VALUE VS. POLICY GRADIENT

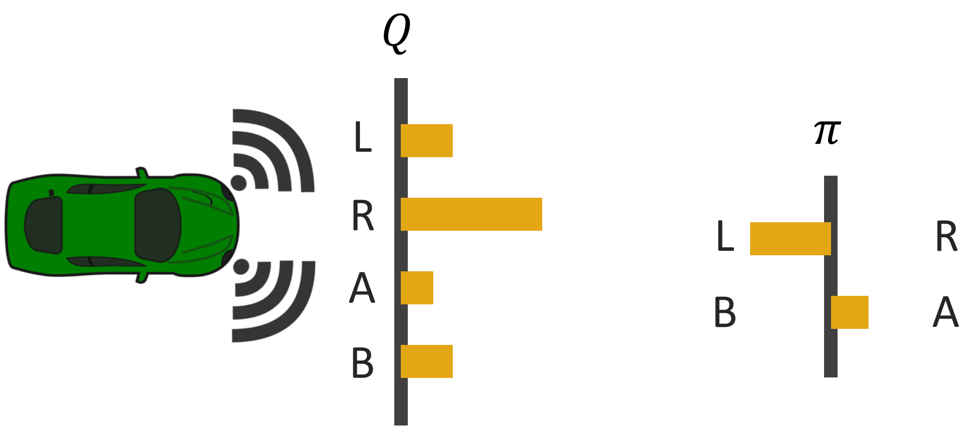
Previous Algorithms:

- Estimate **value** $\boldsymbol{V(s)}$ or $\boldsymbol{Q(s,a)}$
- Take the action with the highest estimated value in every state

$$\pi(s) = \text{argmax}_\text{a} Q(s,a) \tag{8}$$

Why Policy?

1. Environments with lots of actions or a **continuous action space**



https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

VALUE VS. POLICY GRADIENT

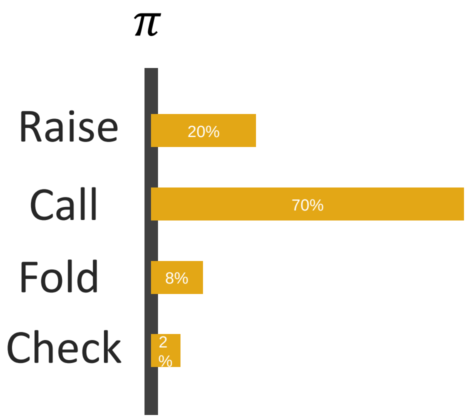
Previous Algorithms:

- Estimate **value** $\boldsymbol{V(s)}$ or $\boldsymbol{Q(s,a)}$
- Take the action with the highest estimated value in every state

$$\pi(s) = \mathrm{argmax}_{\mathrm{a}} Q(s,a) \tag{8}$$

Why Policy?

1. Environments with lots of actions or a **continuous action space**
2. Environments with **stochasticity** in them



https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Policy Gradient

VALUE VS. POLICY GRADIENT

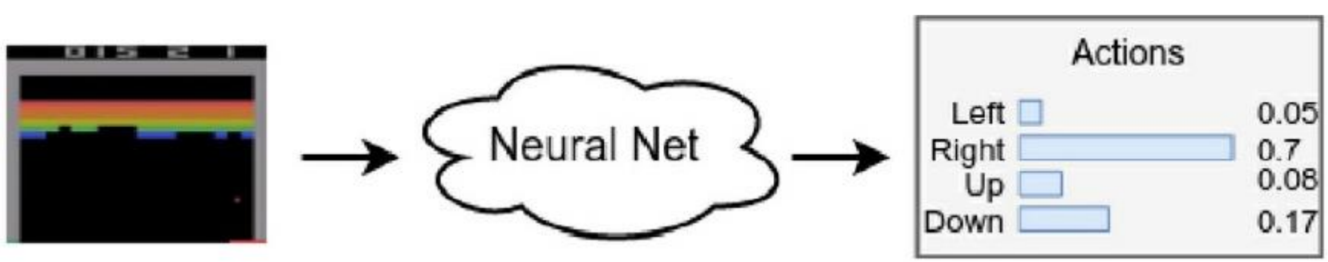
Previous Algorithms:

- Estimate **value** $\boldsymbol{V(s)}$ or $\boldsymbol{Q(s,a)}$
- Take the action with the highest estimated value in every state

$$\pi(s) = \text{argmax}_{\text{a}} Q(s,a) \tag{8}$$

Why Policy?

1. Environments with lots of actions or a **continuous action space**
2. Environments with **stochasticity** in them
3. Enables smooth representation



https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

POLICY GRADIENT

Key Idea

Push up the probabilities of actions that lead to higher return and push down the probabilities of actions that lead to lower return, until you arrive at the optimal policy.

Policy performance:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)] \tag{9}$$

Weight update for gradient ascent:

$$\theta_{k+1} = \theta_k + \alpha \underbrace{\nabla_\theta J(\pi_\theta)}_{\text{Policy Gradient}} \tag{10}$$

$$\boxed{\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right] \approx \frac{1}{|D|}\sum_{\tau \in D}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)} \tag{11}$$

https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146
https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html
https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Reinforce Algorithm

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

Pseudocode

1. Initialize the network with random weights.
2. Play D full episodes, saving their (s, a, r, s') transitions.
3. For every step t of every trajectory τ, calculate the discounted total reward for subsequent steps $G_{\tau,t} = \sum_{i=0} \gamma^i r_i$.
4. Calculate the loss function for all transitions.

$$L = -\frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) G(\tau) \qquad (12)$$

5. Perform SGD update of weights minimizing the loss.
6. Repeat from step 2 until converged.

$$\boxed{\nabla_\theta J(\pi_\theta) \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)} \qquad (11)$$

https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146
https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html
https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

6. Actor Critic Methods

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf
Heilbronn University of Applied Sciences

Policy Gradient Shortcomings

HEILBRONN UNIVERSITY
OF APPLIED SCIENCES

Shortcomings of Policy Gradient:

- Whole trajectories needed
- High variability in log probabilities and cumulative rewards:

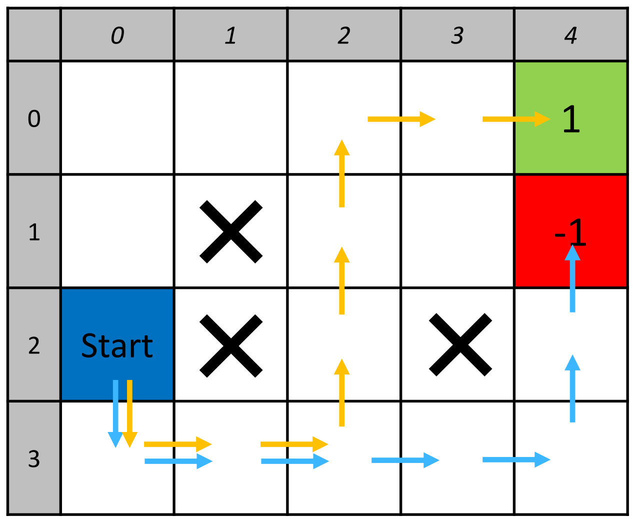
  ➔ **High variance gradients**

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau) \quad (11)$$

- Trajectories with cumulative reward of zero:

  ➔ **Zero gradients**

- On-Policy



$R_t = -0.1$
on all other transitions

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146

Actor Critic Idea

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right] \qquad (11)$$

$$= \mathbb{E}_{s_0,a_0,\ldots,s_t,a_t}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right]\mathbb{E}_{r_{t+1},\ldots,r_T}[G(\tau)]$$

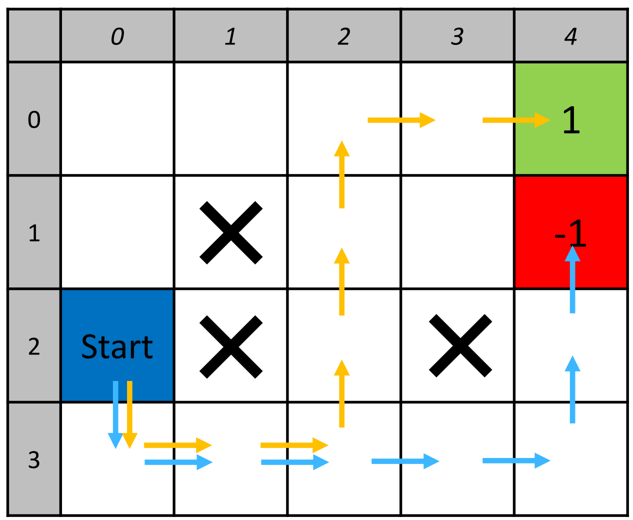
$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_\omega(s_t,a_t)\right] \qquad (12)$$

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Actor Critic Idea

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G(\tau)\right] \tag{11}$$

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_\omega(s_t, a_t)\right] \tag{12}$$

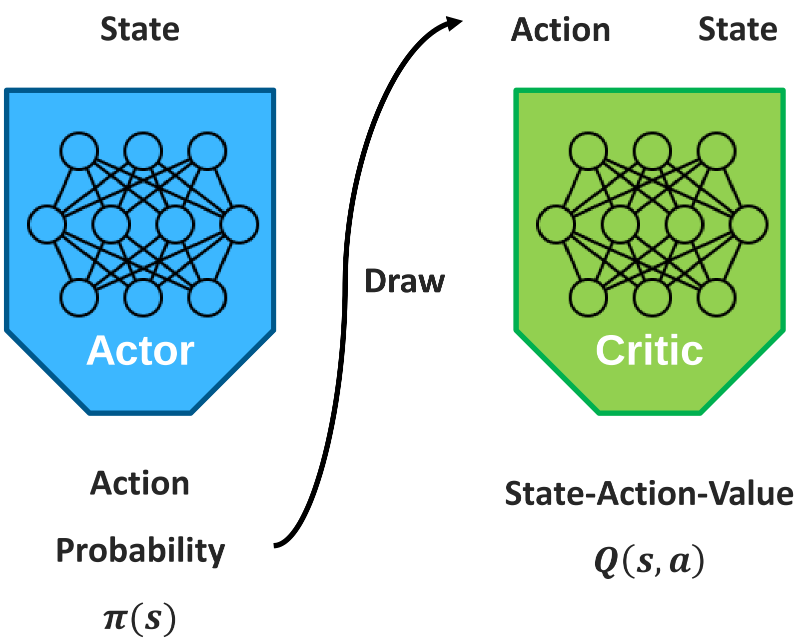


$R_t = -0.1$
on all other transitions

https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Q Actor Critic

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_\omega(s_t, a_t)\right] \quad (12)$$



https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Q Actor Critic

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_\omega(s_t, a_t)\right] \quad (12)$$

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Benefits of Q Actor Critic:

- Updates after one step of playing
- Lower variance in policy gradients

Shortcomings of Q Actor Critic:

- On-Policy algorithm

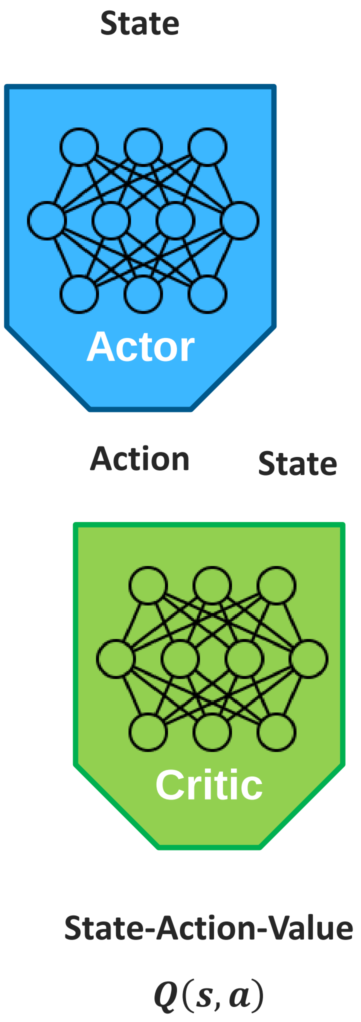
https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Deep Deterministic Policy Gradient (DDPG)

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

Ideas of DDPG:

- Continuous actions instead of probability distribution
- Off-Policy Learning (➔ Replay Buffer)
- Online learning



https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185
https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

Deep Deterministic Policy Gradient (DDPG)

Actor Critic Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\, Q_\omega(s_t, a_t)\right] \quad (12)$$

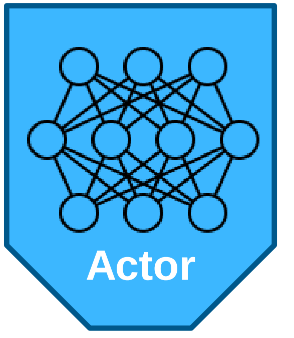
DDPG:

$$J(\mu_\theta) = \mathbb{E}\left[Q_\omega\big(s_t, \mu_\theta(s)\big)\right] \quad (13)$$

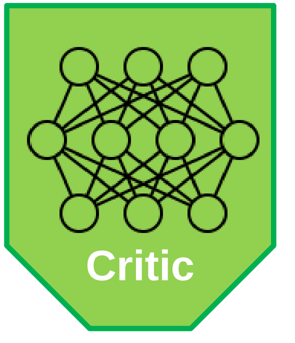
$$\nabla_\theta J(\mu_\theta) = \mathbb{E}\left[\nabla_{\mu_\theta} Q_\omega\big(s_t, \mu_\theta(s_t)\big)\nabla_\theta \mu_\theta(s_t)\right]$$

$$\boxed{\approx \frac{1}{N}\sum_i \nabla_{\mu_\theta} Q_\omega\big(s_i, \mu_\theta(s_i)\big)\nabla_\theta \mu_\theta(s_i)} \quad (14)$$

State



Action **State**



State-Action-Value

$\boldsymbol{Q(s,a)}$

https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185
https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

Deep Deterministic Policy Gradient (DDPG)

DDPG Actor Update:

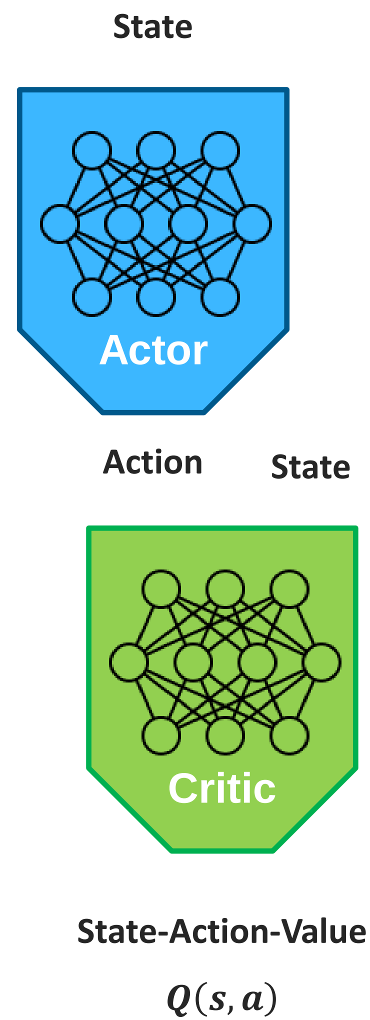
$$\boxed{\nabla_\theta J(\mu_\theta) \approx \frac{1}{N}\sum_i \nabla_{\mu_\theta} Q_\omega\big(s_i, \mu_\theta(s_i)\big)\nabla_\theta \mu_\theta(s_i)} \quad (14)$$

Q Learning Target:

$$y_i = r_i + \gamma \max_{a'} \hat{Q}(s_i', a') \qquad L = \frac{1}{N}\sum_i (Q(s_i, a_i) - y_i)^2 \quad (3)$$

DDPG Critic Target:

$$\boxed{y_i = r_i + \gamma \hat{Q}\big(s_i', \hat{\mu}(s_i')\big) \qquad L_Q = \frac{1}{N}\sum_i (Q(s_i, a_i) - y_i)^2} \quad (15)$$



https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185
https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

Deep Deterministic Policy Gradient (DDPG)

HEILBRONN UNIVERSITY OF APPLIED SCIENCES

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

https://towardsdatascience.com/deep-deterministic-and-twin-delayed-deep-deterministic-policy-gradient-with-tensorflow-2-x-43517b0e0185
https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b

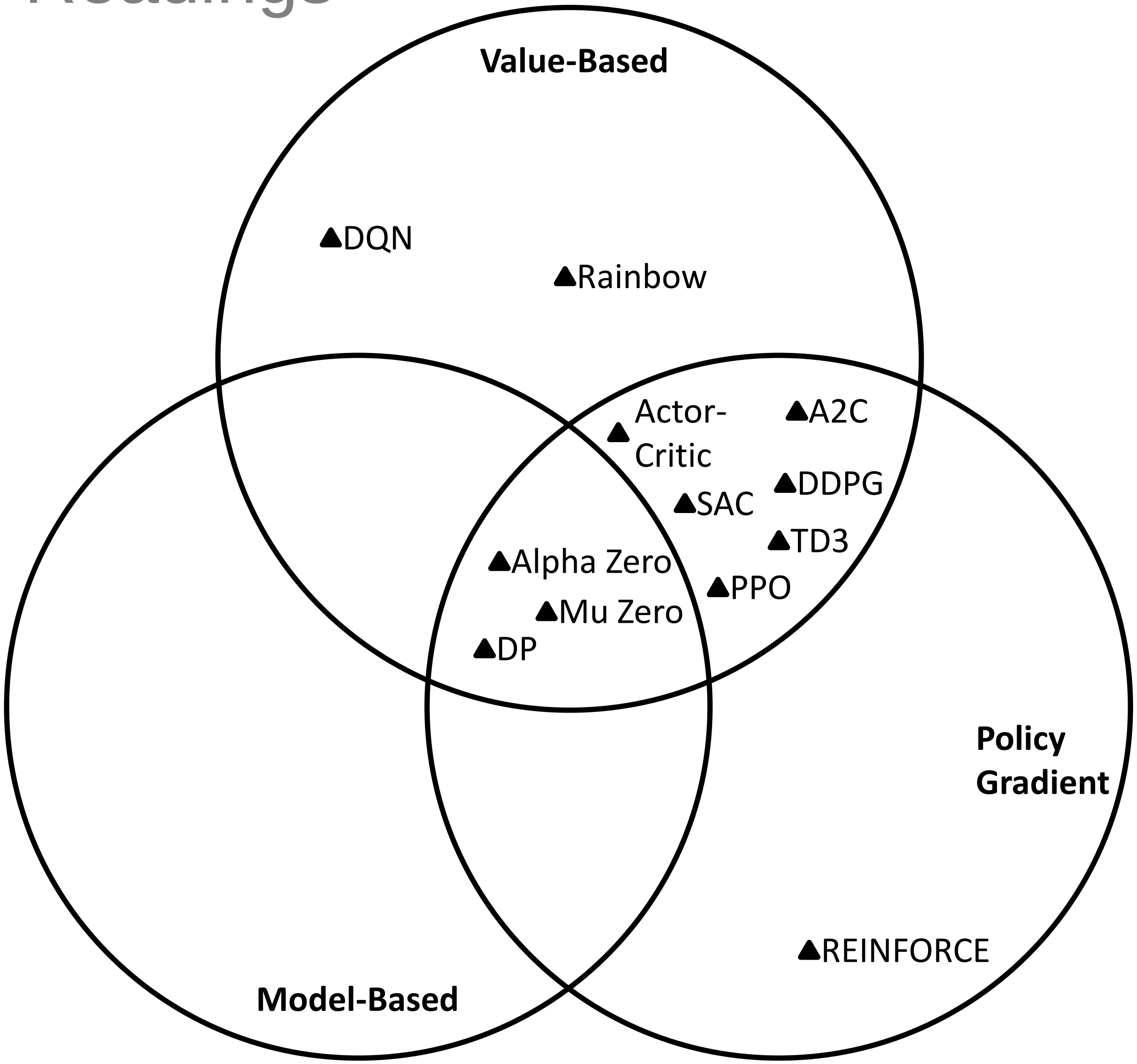
7. Task 3: Deep Deterministic Policy Gradient (DDPG)

HOCHSCHULE HEILBRONN

Prof. Dr.-Ing. Nicolaj Stache, Pascal Graf

Heilbronn University of Applied Sciences

Further Readings



https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146

https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

https://www.packtpub.com/data/deep-reinforcement-learning-hands-on-second-edition

Further Readings

HEILBRONN UNIVERSITY
OF APPLIED SCIENCES

- **TD3 (TWIN DELAYED DDPG)**
- **PPO (PROXIMAL POLICY OPTIMIZIATION)**
- **SAC (SOFT ACTOR-CRITIC)**