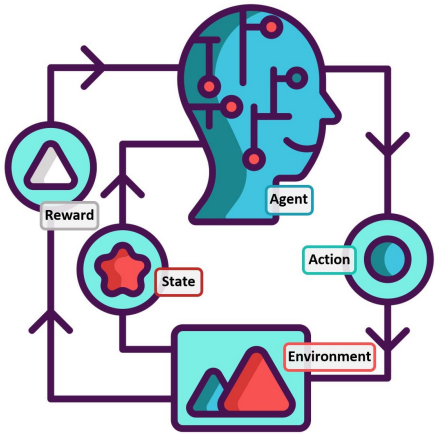
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #22
DeepRL: Continuous Action Space

Riccardo De Monte
Alberto Sinigaglia
Gian Antonio Susto



Typo about AC: lecture 19

Initialize policy parameter θ, v

Algorithm parameter : step-size α $\beta > 0$

Repeat (for each episode) :

Initialize s

Loop for each step of the episode $t = 0, \ldots, T-1$

take an action $a \sim \pi(a|s, \theta)$

Observe reward r and new state s'

$$\delta_{\mathbf{v}} = r + \gamma \hat{v}(s', \mathbf{v}) - \hat{v}(s, \mathbf{v})$$

$$\theta = \theta + \alpha \, \delta_{\mathbf{v}} \, \nabla_\theta \ln \pi(a|s, \theta)$$

$$\mathbf{v} = \mathbf{v} + \beta \delta_{\mathbf{v}} \nabla \hat{v}(s, \mathbf{v})$$

$$s \leftarrow s'$$

Now the critic is simplified! We get only 1 step of parameters

DQN and PPO

1. We have seen that one important assumption for many RL and DRL algorithms is that the action space is discrete and finite.

DQN and PPO

1. We have seen that one important assumption for many RL and DRL algorithms is that the action space is discrete and finite.

2. For value-based approaches, as DQN, this allows us to derive a greedy action:

$$\pi(s) = \text{argmax}_a Q(s, a)$$

3. For PPO (or any policy-gradient based approach):

$$\pi(a|s) = \frac{e^{h_\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{h_\theta(s,a')}}, \qquad |\mathcal{A}| < \infty$$

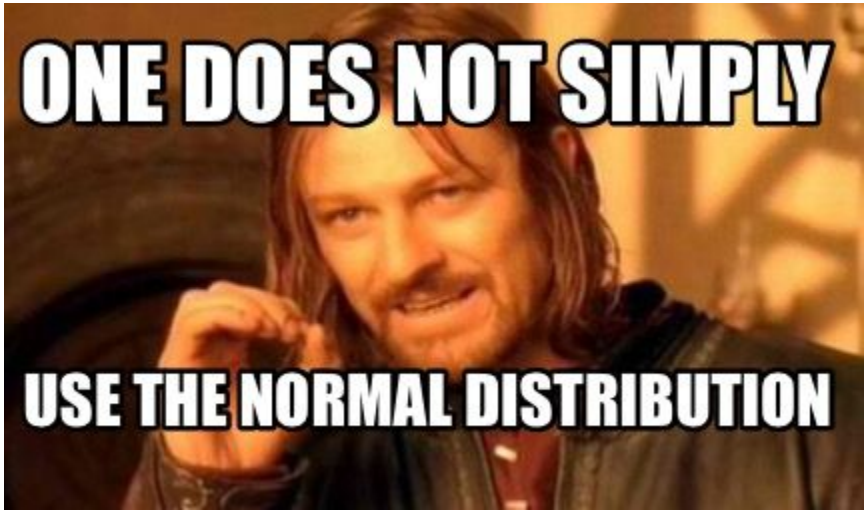
What about continuous action spaces?

What about continuous action spaces?

```
BoundedArray(shape=(6,), dtype=dtype('float64'), name=None, minimum=[-1. -1. -1. -1. -1. -1.], maximum=[1. 1. 1. 1. 1. 1.])
```

Can we use A2C/PPO with continuous action spaces?

Yes: use Gaussians!



Simple case: 1d continuous action

1. We assume 1d continuous action: one real number in the interval [–1,1].

Simple case: 1d continuous action

1. We assume 1d continuous action: one real number in the interval [-1,1].

2. Recall that a Gaussian has two parameters: mean and standard deviation

$$a \sim \mathcal{N}(\mu, \sigma^2), \quad \mu \in \mathbb{R}, \sigma \in \mathbb{R}^+$$

Simple case: 1d continuous action

1. We assume 1d continuous action: one real number in the interval [-1,1].

2. Recall that a Gaussian has two parameters: mean and standard deviation

$$a \sim \mathcal{N}(\mu, \sigma^2), \quad \mu \in \mathbb{R}, \sigma \in \mathbb{R}^+$$

3. Assume to fix sigma to a constant (I will clarify this later) e.g. 0.2.

Simple case: 1d continuous action

1. We assume 1d continuous action: one real number in the interval [-1,1].

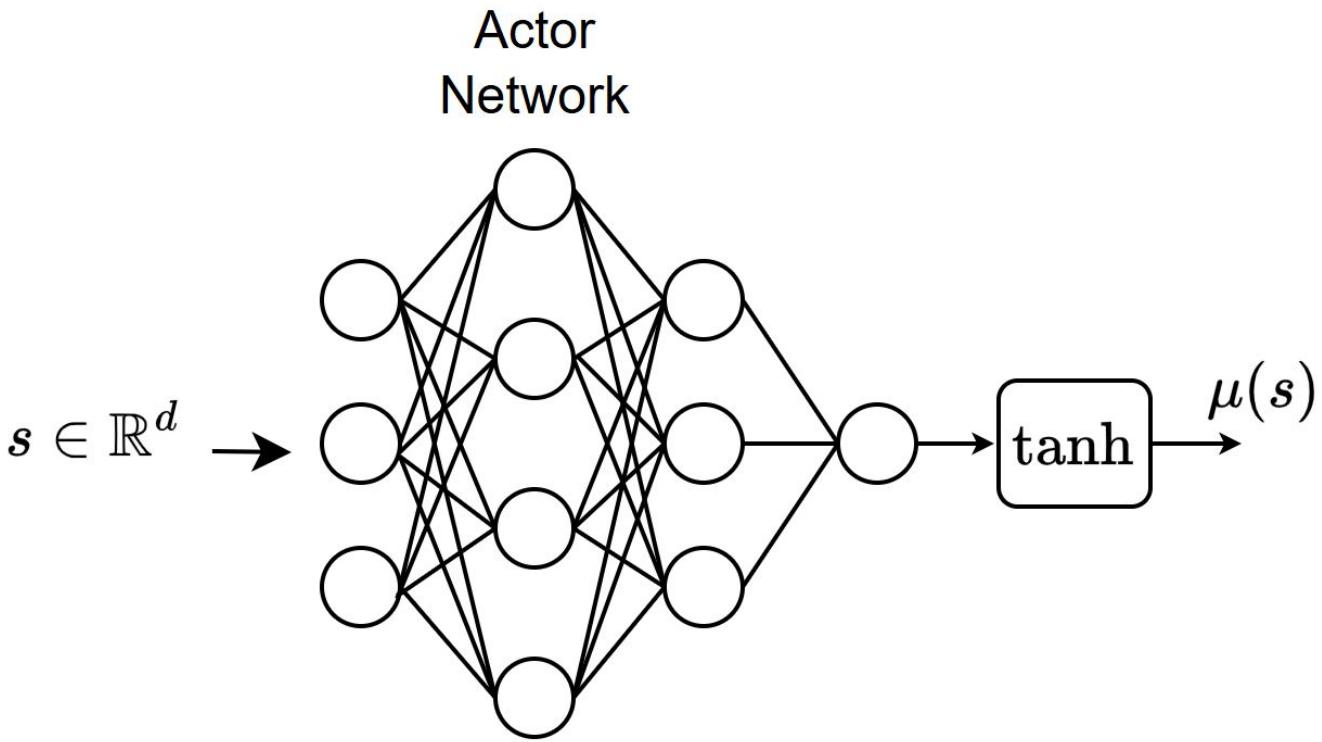
2. Recall that a Gaussian has two parameters: mean and standard deviation

$$a \sim \mathcal{N}(\mu, \sigma^2), \quad \mu \in \mathbb{R}, \sigma \in \mathbb{R}^+$$

3. Assume to fix sigma to a constant (I will clarify this later) e.g. 0.2.

4. What about the mean?

Simple case: 1d continuous action



Simple case: 1d continuous action

By doing so, we have a distribution over actions conditioned by the state: to condition such a distribution based on the state we compute the expected value of the gaussian with the output of the actor network.

$$\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma^2), \qquad \mu_\theta(s) = \text{ActorNN}(s)$$

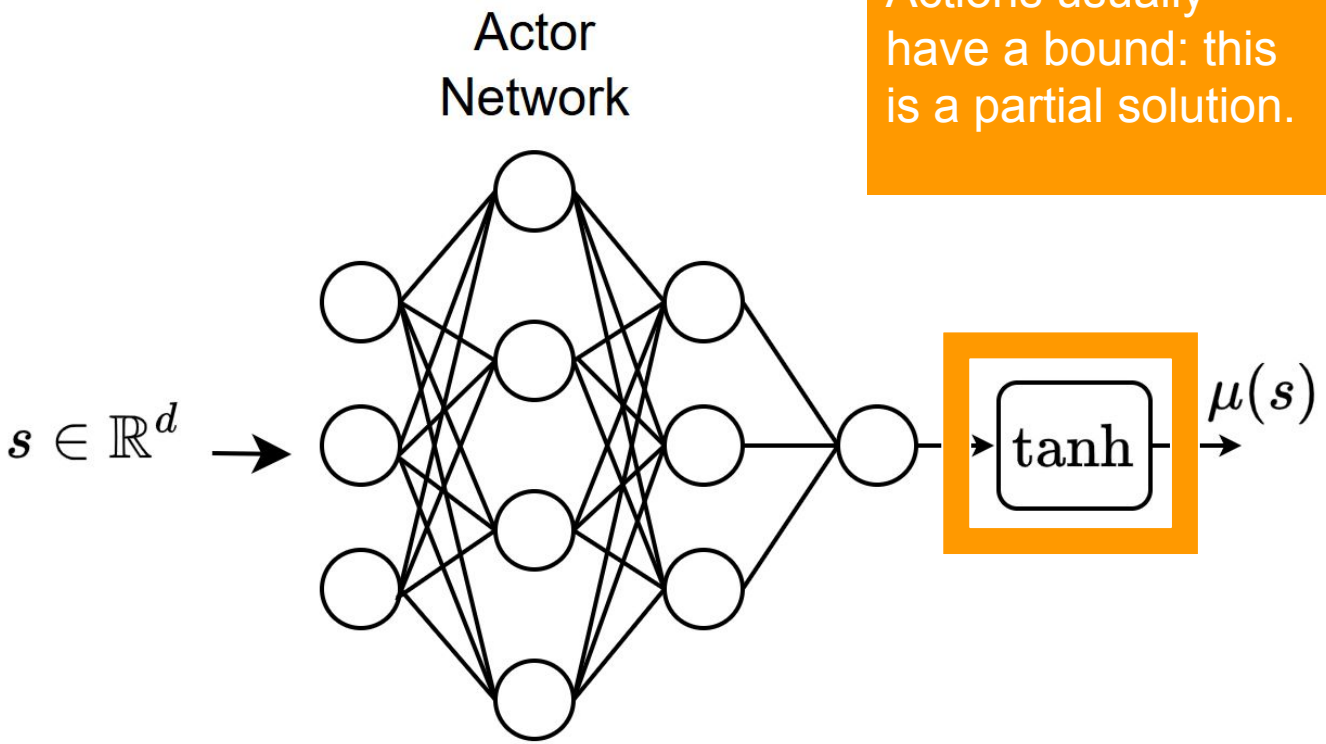
Simple case: 1d continuous action

By doing so, we have a distribution over actions conditioned by the state: to condition such a distribution based on the state we compute the expected value of the gaussian with the output of the actor network.

$$\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma^2), \qquad \mu_\theta(s) = \text{ActorNN}(s)$$

Once we get the output of the NN (given the current state), we can sample the corresponding action. Numpy,PyTorch, … allow to sample from a gaussian!

Simple case: 1d continuous action



Actions usually have a bound: this is a partial solution.

Simple case: 1d continuous action

Ok... but how to update the actor network with PPO???

Simple case: 1d continuous action

Ok... but how to update the actor network with PPO???

Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):

$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t | s_t)$$

We need to update the parameters of the actor network by computing that gradient.

Simple case: 1d continuous action

Ok... but how to update the actor network with PPO???

Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):

$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t | s_t)$$

We need to update the parameters of the actor network by computing that gradient.

With Gaussians, life is easy!

Recap about A2C

1. Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):

$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Recap about A2C

1. Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):
$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

2. Recall what the advantage is: $A_t = Q_\pi(s_t, a_t) - V\pi(s_t) \approx r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$

Recap about A2C

1. Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):
$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

2. Recall what the advantage is: $A_t = Q_\pi(s_t, a_t) - V\pi(s_t) \approx r_{t+1} + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)$

3. If $A_t > 0$, the action sampled a_t for the current state s_t is “better” than the others (at least, there are some that are worse) . We should increase its likelihood.

4. If $A_t < 0$, opposite case.

Recap about A2C

1. Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):

$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

2. So, the sign of the advantage tells you if we should increase or decrease the likelihood of a certain action given a state.

Recap about A2C

1. Recall for simplicity A2C for one sample (PPO is a bit more complicated, but it is the same):
$$\nabla_\theta L(\theta) = A_t \nabla_\theta \log \pi_\theta(a_t|s_t)$$

2. So, the sign of the advantage tells you if we should increase or decrease the likelihood of a certain action given a state.

3. What about the other term?
$$\nabla_\theta \log \pi_\theta(a_t|s_t)$$

With Gaussian, $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is easy to compute

1. Recall the PDF of a Gaussian:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2\right)$$

With Gaussian, $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is easy to compute

1. Recall the PDF of a Gaussian:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2\right)$$

2. The log:

$$\log \pi_\theta(a_t|s_t) = -\log\sqrt{2\pi\sigma^2} - \frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2$$

With Gaussian, $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is easy to compute

1. Recall the PDF of a Gaussian:

$$\pi_\theta(a_t|s_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2\right)$$

2. The log:

$$\log \pi_\theta(a_t|s_t) = -\log\sqrt{2\pi\sigma^2} - \frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2$$

3. The gradient:

$$\nabla_\theta \log \pi_\theta(a_t|s_t) = \nabla_\theta\left(-\frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))^2\right) = \frac{1}{2\sigma^2}(a_t - \mu_\theta(s_t))\nabla_\theta \mu_\theta(s_t)$$

So, in practice, we might do

1. The NN computes the expected value.

So, in practice, we might do

1. The NN computes the expected value.

2. During exploration, we set this output as expected value of a gaussian and we sample (e.g. use np.random.normal(loc=output_nn, scale=0.2).

So, in practice, we might do

1. The NN computes the expected value.

2. During exploration, we set this output as expected value of a gaussian and we sample (e.g. use np.random.normal(loc=output_nn, scale=0.2).

3. To update the policy, we define the following loss function (for A2C):

$$L(\theta) = A_t \cdot \frac{1}{2\sigma^2}(a_t - \mathrm{ActorNN}_\theta(s_t))^2$$

So, in practice, we might do

1. The NN computes the expected value.

2. During exploration, we set this output as expected value of a gaussian and we sample (e.g. use np.random.normal(loc=output_nn, scale=0.2).

3. To update the policy, we define the following loss function (for A2C):

$$L(\theta) = A_t \cdot \frac{1}{2\sigma^2}(a_t - \text{ActorNN}_\theta(s_t))^2$$

When we train NNs, we need a loss function to minimize. In RL we maximize -> we need to change sign!

Nice, but don’t do that!!!

Use PyTorch, TensorFlow, JAX. Example with PyTorch:

```
import torch

# actor net output
mu = actor(s)  # ActorNN(s_t)
std = 0.2

# create our policy (distribution of a given s)
dist = torch.distributions.Normal(mu, std)

# For sampling:
a = dist.sample()

# For policy update (actor update)
logp = dist.log_prob(a)  #log pi(a|s)

# a2c loss
loss = - A * logp
```

Additional stuffs

The standard deviation should be “learnable”:

1. We can set it as not-state dependent, but learnable in the sense the optimizer will change it.

2. We can consider it as state dependent: the actor network also computes a the standard deviation:
    a. Use softplus for that output.
    b. Or, consider that output of the NN as the log of the standard deviation and apply torch.exp() (**use this**)

Additional stuffs

The standard deviation should be “learnable”:

1. We can set it as not-state dependent, but learnable in the sense the optimizer will change it.

2. We can consider it as state dependent: the actor network also computes a the standard deviation:
   a. Use softplus for that output.
   b. Or, consider that output of the NN as the log of the standard deviation and apply torch.exp() (**use this**)

For multi dimensional action space, we assume diagonal covariance matrix.


THATS ALL YOU GOT? REALLY ?
memecrunch.com


THATS ALL YOU GOT? REALLY ?
BUT WAIT..
THERE'S MORE..


THATS ALL YOU GOT? REALLY ?
BUT WAIT..
WELL, WE CAN DO BETTER THAN THAT.

Can we instead obtain a reasonable deterministic policy?

Yes! How?????

1. Recall usual Q-learning approach with greedy policy:

$$\pi(s) = \text{argmax}_a Q(s, a)$$

Yes! How?????

1. Recall usual Q-learning approach with greedy policy:

$$\pi(s) = \text{argmax}_a Q(s,a)$$

2. For continuous action space, this is an optimization problem: Q(s,a) might be output of the critic NN that takes as inputs both the state and the action.

Yes! How?????

1. Recall usual Q-learning approach with greedy policy:

$$\pi(s) = \text{argmax}_a Q(s,a)$$

2. For continuous action space, this is an optimization problem: Q(s,a) might be output of the critic NN that takes as inputs both the state and the action.

3. A NN is a function of a -> how to solve an optimization problem w.r.t. a?

Yes! How?????

1. Recall usual Q-learning approach with greedy policy:

$$\pi(s) = \text{argmax}_a Q(s,a)$$

2. For continuous action space, this is an optimization problem: Q(s,a) might be output of the critic NN that takes as inputs both the state and the action.

3. A NN is a function of a -> how to solve an optimization problem w.r.t. a?

4. What about following the gradient direction?

What did we do with gaussians?

1. We sample one action a.

What did we do with gaussians?

1. We sample one action a.

2. We evaluate how good is that action by observing the advantage.

What did we do with gaussians?

1. We sample one action a.

2. We evaluate how good is that action by observing the advantage.

3. Accordingly to the advantage, we decide to either increase or decrease the output of the actor NN (the expected value of the gaussian).

What did we do with gaussians?

1. We sample one action a.

2. We evaluate how good is that action by observing the advantage.

3. Accordingly to the advantage, we decide to either increase or decrease the output of the actor NN (the expected value of the gaussian).

4. It’s like a procedure where sampling is used to perturbe the output of the actor NN and observe if that perturbation actually improves or not the return.

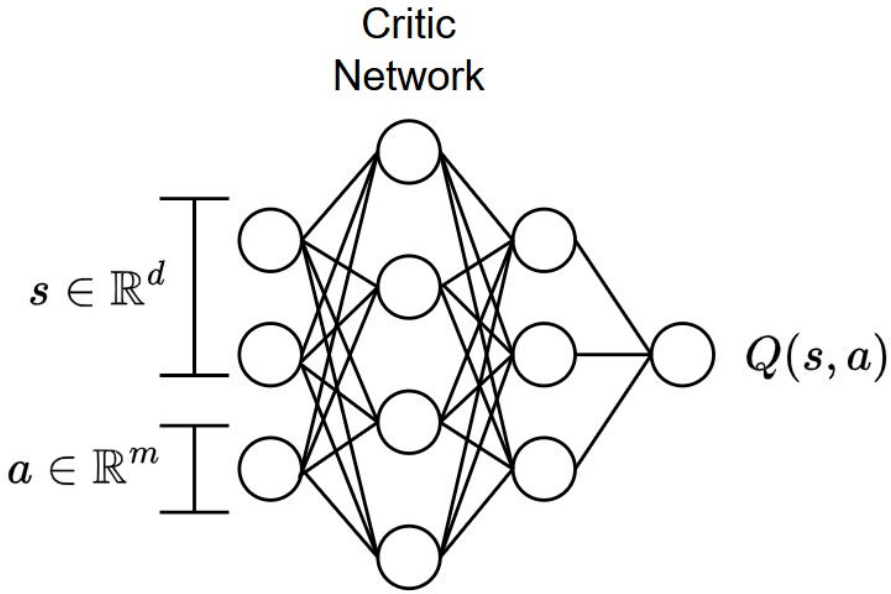
A different approach for continuous action space

1. We still consider an Actor-Critic approach.

A different approach for continuous action space

1. We still consider an Actor-Critic approach.

2. Instead of V, the critic network is used to estimate Q:



A different approach for continuous action space

1. So, the critic NN allows to evaluate how good is one action.

A different approach for continuous action space

1. So, the critic NN allows to evaluate how good is one action.

2. Solving a maximization problem like $\pi(s) = \text{argmax}_a Q(s, a)$ is still quite hard.

A different approach for continuous action space

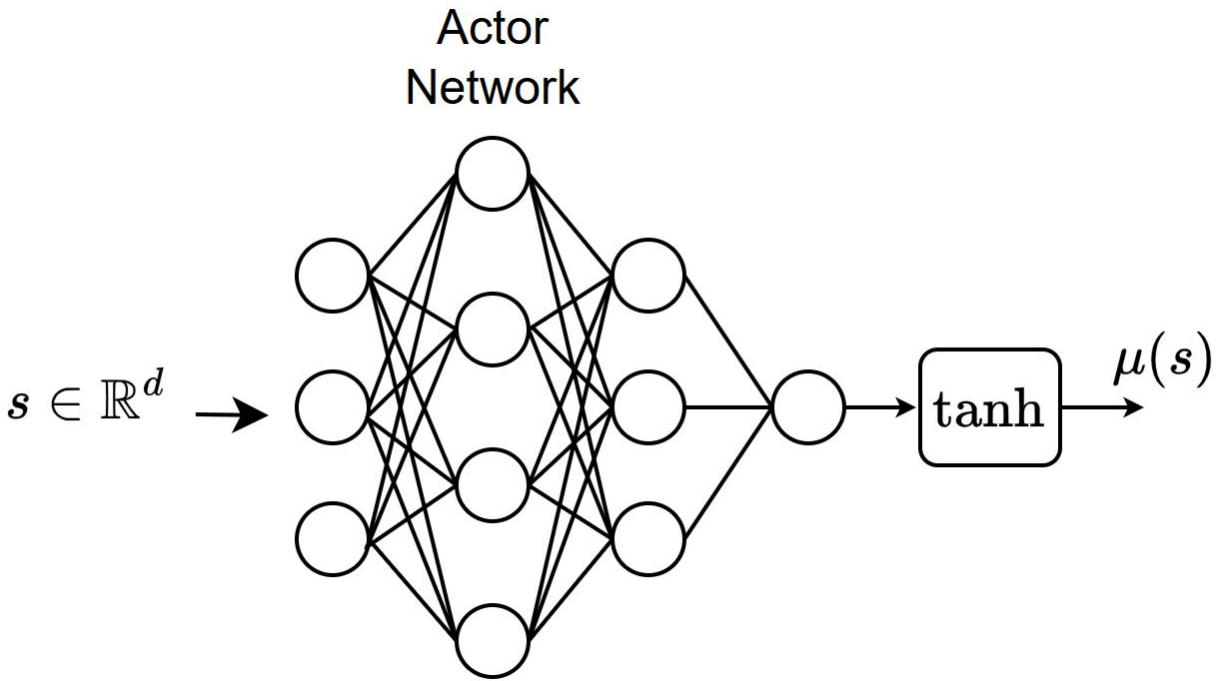
1. So, the critic NN allows to evaluate how good is one action.

2. Solving a maximization problem like $\pi(s) = \text{argmax}_a Q(s, a)$ is still quite hard.

3. But, we take inspiration by the perturbation procedure, similar to Gaussian policy + A2C .

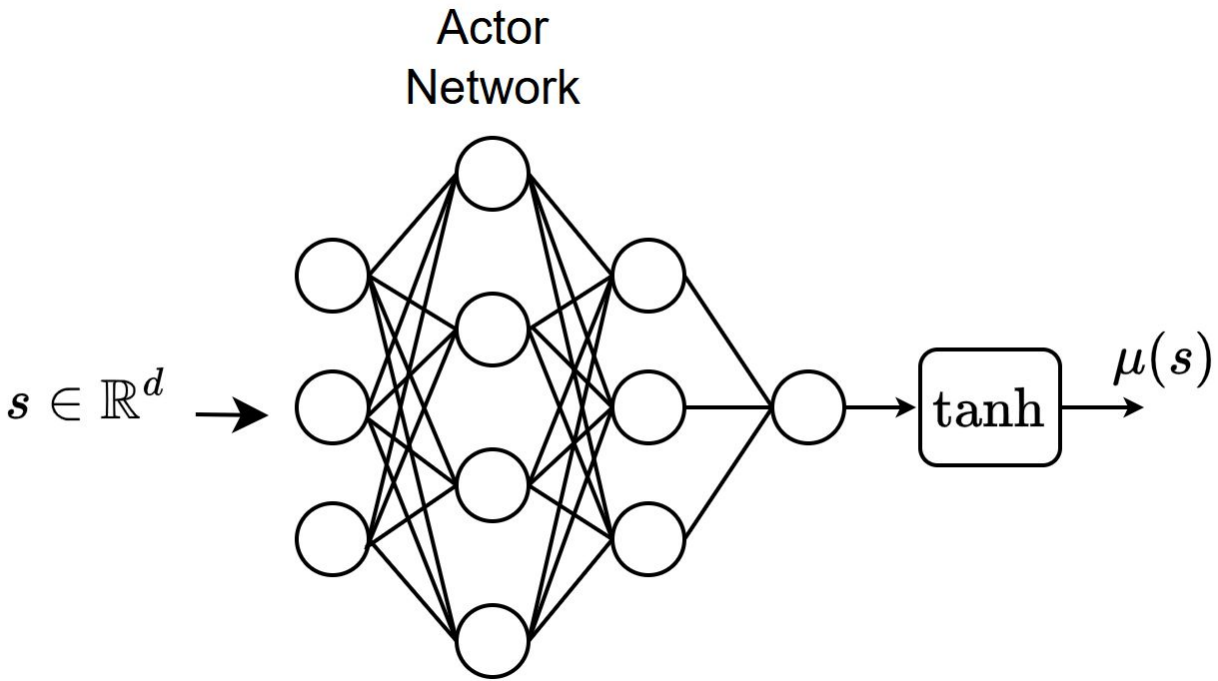
A different approach for continuous action space

We define the actor network similarly to the PPO/A2C



A different approach for continuous action space

We define the actor network similarly to the PPO/A2C



The output of the NN is no longer the expected value of a gaussian, but the actual output of a deterministic policy!

A different approach for continuous action space

1. What about taking one action a for state s and compute Q(s,a+c), where c>0 is a small perturbation, and observe what happens?

A different approach for continuous action space

1. What about taking one action a for state s and compute Q(s,a+c), where c>0 is a small perturbation, and observe what happens?

2. What Q(s,a+c)>Q(s,a) means? It means, that, locally (w.r.t. a), Q(s,a) increases as a increases.

A different approach for continuous action space

1. What about taking one action a for state s and compute Q(s,a+c), where c>0 is a small perturbation, and observe what happens?

2. What Q(s,a+c)>Q(s,a) means? It means, that, locally (w.r.t. a), Q(s,a) increases as a increases.

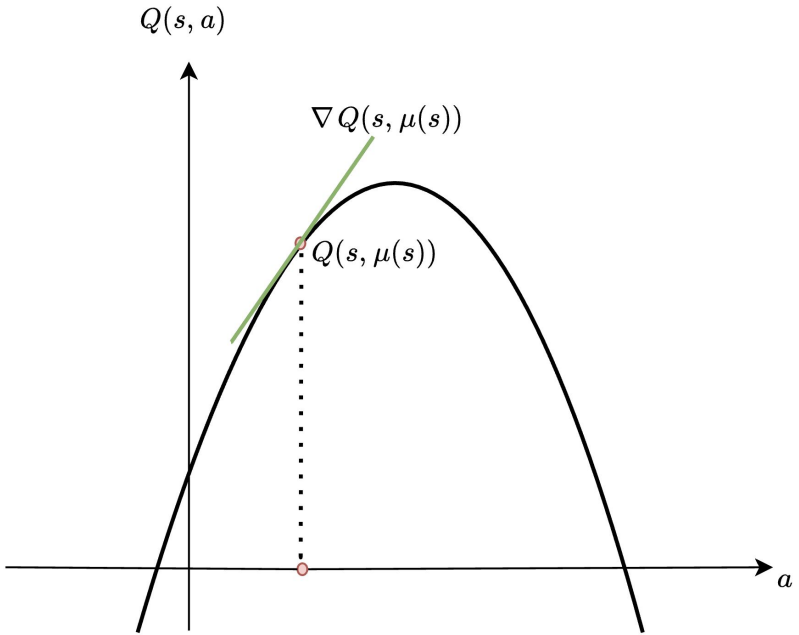
3. This is computing the derivative of Q w.r.t. a and observe if the derivative is either positive or negative.

A different approach for continuous action space

Note that the Q network is a NN and therefore it is differentiable (we can apply the chain rule)

A different approach for continuous action space

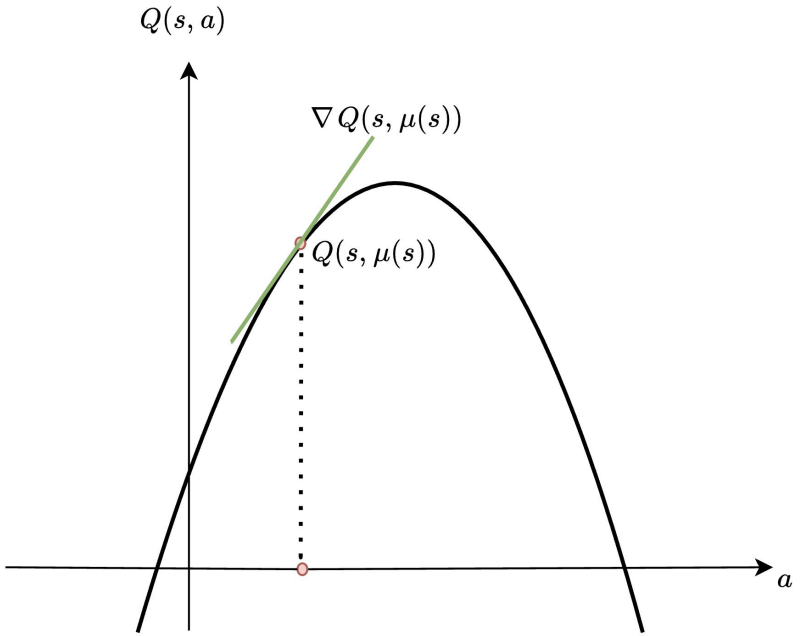
Note that the Q network is a NN and therefore it is differentiable (we can apply the chain rule)



Just for visualisation, assume that Q(s,a) is a paraboloid.

A different approach for continuous action space

Note that the Q network is a NN and therefore it is differentiable (we can apply the chain rule)



Just for visualisation, assume that Q(s,a) is a paraboloid.

We can compute the derivative of Q(s,$\boldsymbol{\mu}$(s)) with respect to the parameters of the actor network by using the chain rule to determine how to change the actor parameters to increase Q:

$$\frac{dQ(s,a)}{da}\frac{da}{d\theta} = \frac{dQ(s,\mu_\theta(s))}{d\mu_\theta(s)}\frac{d\mu_\theta(s)}{d\theta}$$

Deep Deterministic Policy Gradient (DDPG)

1. An actor network for a deterministic policy: $a = \mu_\theta(s)$

Deep Deterministic Policy Gradient (DDPG)

1. An actor network for a deterministic policy: $a = \mu_\theta(s)$

2. Deterministic Policy Gradient Theorem [1] provides an update rule for the actor:

$$\nabla_\theta J(\theta) = \mathbb{E}_s[\nabla_\theta \mu_\theta(s) \nabla_a Q(s,a)|_{a=\mu_\theta(s)}]$$

[1] Deterministic Policy Gradient Algorithms, David Silver et all.

Deep Deterministic Policy Gradient (DDPG)

1. An actor network for a deterministic policy: $a = \mu_\theta(s)$
2. Deterministic Policy Gradient Theorem [1] provides an update rule for the actor:
$$\nabla_\theta J(\theta) = \mathbb{E}_s[\nabla_\theta \mu_\theta(s) \nabla_a Q(s,a)|_{a=\mu_\theta(s)}]$$
3. Since we deal with NNs, we use a replay buffer as for DQN.

[1] Deterministic Policy Gradient Algorithms, David Silver et all.

Deep Deterministic Policy Gradient (DDPG)

1. An actor network for a deterministic policy: $a = \mu_\theta(s)$

2. Deterministic Policy Gradient Theorem [1] provides an update rule for the actor:
$$\nabla_\theta J(\theta) = \mathbb{E}_s[\nabla_\theta \mu_\theta(s) \nabla_a Q(s,a)|_{a=\mu_\theta(s)}]$$

3. Since we deal with NNs, we use a replay buffer as for DQN.

4. In practice we use mini-batch SGD and we rely on PyTorch/TensorFlow/JAX:
$$L(\theta) = -\frac{1}{M}\sum_{j=1}^{M} Q(s_j, \mu_\theta(s_j)), \quad s_j \sim D$$

[1] Deterministic Policy Gradient Algorithms, David Silver et all.

Deep Deterministic Policy Gradient (DDPG)

1. For the critic we use a Q network $Q_w(s, a)$

Deep Deterministic Policy Gradient (DDPG)

1. For the critic we use a Q network $Q_w(s, a)$
2. Since the policy is deterministic, we need off-policy training (for exploration).

Deep Deterministic Policy Gradient (DDPG)

1. For the critic we use a Q network $Q_w(s, a)$
2. Since the policy is deterministic, we need off-policy training (for exploration).
3. Simple way to explore: adding gaussian noise.

$$a_t = \mu_\theta(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Deep Deterministic Policy Gradient (DDPG)

1. For the critic we use a Q network $Q_w(s, a)$
2. Since the policy is deterministic, we need off-policy training (for exploration).
3. Simple way to explore: adding gaussian noise.

$$a_t = \mu_\theta(s_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

4. We use Q-Learning to update the critic without Importance Sampling:

$$L(w) = \frac{1}{M} \sum_{j=1}^{M} (y_j - Q_w(s_j, a_j))^2, \quad y_j = r_j + \gamma Q_w(s_{j+1}, \mu_\theta(s_{j+1}))$$

DDPG: additional stuffs

1. In the original paper of DDPG, they use target networks for both the critic and the actor.

DDPG: additional stuffs

1. In the original paper of DDPG, they use target networks for both the critic and the actor.

2. DDPG has some problems:
   a. Overestimation as DQN, but here the problem cannot be solved as in Double DQN.

DDPG: additional stuffs

1. In the original paper of DDPG, they use target networks for both the critic and the actor.

2. DDPG has some problems:
   a. Overestimation as DQN, but here the problem cannot be solved as in Double DQN.
   b. Inaccurate Q, makes policy updates quite noisy.

DDPG: additional stuffs

1. In the original paper of DDPG, they use target networks for both the critic and the actor.

2. DDPG has some problems:
   a. Overestimation as DQN, but here the problem cannot be solved as in Double DQN.
   b. Inaccurate Q, makes policy updates quite noisy.

3. **TD3: T**win **D**elayed **D**eep **D**eterministic **P**olicy **G**radient, provides some modifications to tackle those problems. Paper: “Addressing Function Approximation Error in Actor-Critic Methods”, Scott Fujimoto et all.

Stochastic Policy Gradient vs Deterministic Policy Gradient

1. Recall PG (for stochastic policies):

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\int_a Q(s,a) \nabla_\theta \pi_\theta(a|s) \, da \right] = \mathbb{E}_s \left[\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [Q(s,a) \nabla_\theta \log \pi_\theta(a|s)] \right]$$

Stochastic Policy Gradient vs Deterministic Policy Gradient

1. Recall PG (for stochastic policies):

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\int_a Q(s,a) \nabla_\theta \pi_\theta(a|s) \, da \right] = \mathbb{E}_s \left[\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [Q(s,a) \nabla_\theta \log \pi_\theta(a|s)] \right]$$

2. While PG for deterministic policies:

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\nabla_\theta \, \mu_\theta(s) \nabla_a Q(s,a) \big|_{a=\mu_\theta(s)} \right]$$

Stochastic Policy Gradient vs Deterministic Policy Gradient

1. Recall PG (for stochastic policies):

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\int_a Q(s,a) \nabla_\theta \pi_\theta(a|s)\, da \right] = \mathbb{E}_s \left[\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [Q(s,a) \nabla_\theta \log \pi_\theta(a|s)] \right]$$

2. While PG for deterministic policies:

$$\nabla_\theta J(\theta) = \mathbb{E}_s \left[\nabla_\theta\, \mu_\theta(s) \nabla_a Q(s,a)|_{a=\mu_\theta(s)} \right]$$

3. For continuous action spaces, due to curse of dimensionality, DPG-based approaches has less variance problems. PPO might be ok, but for continuous action space problem I highly recommend TD3/SAC


Anything else


Anythi
BUT WAIT
THERE'S MORE


BUT
THERE
How exciting!

Soft Actor Critic (SAC)

I have not enough time to explain SAC, but...

1. SAC is quite important because it introduces a different RL objective: not only the reward matters, but also the entropy of the policy.

I have not enough time to explain SAC, but...

1. SAC is quite important because it introduces a different RL objective: not only the reward matters, but also the entropy of the policy.

2. Among the many cool stuffs about SAC, they exploit a famous trick used also in Generative AI e.g. Variational Autoencoder.

I have not enough time to explain SAC, but...

1. SAC is quite important because it introduces a different RL objective: not only the reward matters, but also the entropy of the policy.

2. Among the many cool stuffs about SAC, they exploit a famous trick used also in Generative AI e.g. Variational Autoencoder.

3. SAC considers stochastic policies as PPO, but they derived a different loss for the policy improvement (assume as usual a given Q network):

$$L(\theta) = \mathbb{E}_s \left[\mathbb{E}_{a \sim \pi_\theta(\cdot|s)} \left[\alpha \log \pi_\theta(a|s) - Q(s,a) \right] \right]$$

A general problem (not only in RL)

1. Forget about states and focus on the actions.

A general problem (not only in RL)

1. Forget about states and focus on the actions.

2. With PG we have seen the following trick (useful to estimate an integral using MC):

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)] = \int_a f(a)\nabla_\theta \pi_\theta(a)\, da = \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)\nabla_\theta \log \pi_\theta(a)]$$

A general problem (not only in RL)

1. Forget about states and focus on the actions.

2. With PG we have seen the following trick (useful to estimate an integral using MC):

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)] = \int_a f(a)\nabla_\theta \pi_\theta(a)\, da = \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)\nabla_\theta \log \pi_\theta(a)]$$

3. If we use MC, the resulting estimator has high variance and we should sample many a to reduce it.

Pathwise Derivative (Reparameterization Trick)

1. If we define:

$$a = g(\epsilon, \theta), \quad \epsilon \sim p(\epsilon), \quad g(\cdot, \cdot) \text{ differentiable and invertible}$$

Pathwise Derivative (Reparameterization Trick)

1. If we define:

$$a = g(\epsilon, \theta), \quad \epsilon \sim p(\epsilon), \quad g(\cdot, \cdot) \text{ differentiable and invertible}$$

2. ... we have:

$$\nabla_\theta \mathbb{E}_{a \sim \pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_\theta f(g(\epsilon, \theta))]$$

Pathwise Derivative (Reparameterization Trick)

1. If we define:

$$a = g(\epsilon, \theta), \quad \epsilon \sim p(\epsilon), \quad g(\cdot, \cdot) \text{ differentiable and invertible}$$

2. ... we have:

$$\nabla_\theta \mathbb{E}_{a \sim \pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_\theta f(g(\epsilon, \theta))]$$

3. Gaussian case: $a = \mu_\theta + \sigma_\theta \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1)$

Pathwise Derivative (Reparameterization Trick)

1. If we define:

$$a = g(\epsilon, \theta), \quad \epsilon \sim p(\epsilon), \quad g(\cdot, \cdot) \text{ differentiable and invertible}$$

2. ... we have:

$$\nabla_\theta \mathbb{E}_{a \sim \pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_\theta f(g(\epsilon, \theta))]$$

3. Gaussian case: $a = \mu_\theta + \sigma_\theta \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1)$

$$\pi_\theta(a) = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$$

Pathwise Derivative (Reparameterization Trick)

1. If we define:

$$a = g(\epsilon, \theta), \quad \epsilon \sim p(\epsilon), \quad g(\cdot, \cdot) \text{ differentiable and invertible}$$

2. ... we have:

$$\nabla_\theta \mathbb{E}_{a \sim \pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_\theta f(g(\epsilon, \theta))]$$

3. Gaussian case: $a = \mu_\theta + \sigma_\theta \epsilon, \qquad \epsilon \sim \mathcal{N}(0, 1)$

$$\pi_\theta(a) = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$$

$$\nabla_\theta \mathbb{E}_{a \sim \mathcal{N}(\mu_\theta, \sigma_\theta)}[f(a)] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[\nabla_a f(a)|_{a=\mu_\theta+\sigma_\theta\epsilon} \nabla_\theta(\mu_\theta + \sigma_\theta \epsilon)]$$

Pathwise Derivative vs Log-Likelihood based (Reinforce)

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{\epsilon\sim p(\epsilon)}[\nabla_\theta f(g(\epsilon,\theta))]$$

$$\nabla_\theta \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)] = \mathbb{E}_{a\sim\pi_\theta(\cdot)}[f(a)\nabla_\theta \log \pi_\theta(a)]$$

AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Questions?
Lecture #22
DeepRL: Continuous Action Space

Riccardo De Monte
Alberto Sinigaglia
Gian Antonio Susto