AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

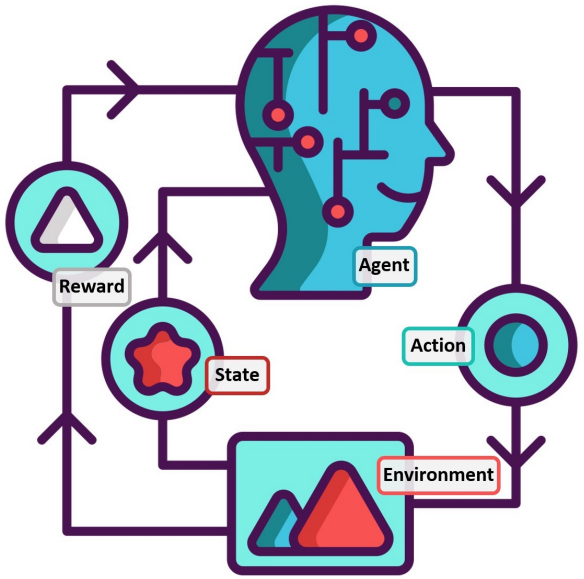
Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #21
DeepRL: the Discrete Case

Gian Antonio Susto

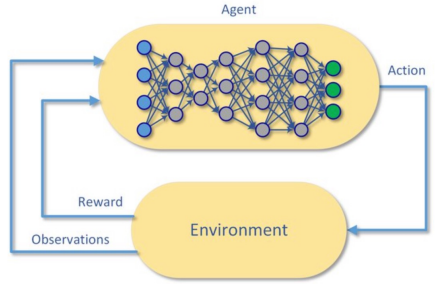
Special thanks to Alberto Sinigaglia!



Announcements before start

- This lecture is not second partial material, and it will not be part of the ‘regular’ exams
- However, the material is particularly relevant for real world application (and for the exam projects!)
- Today it will be ‘advanced material’!

Possible approaches

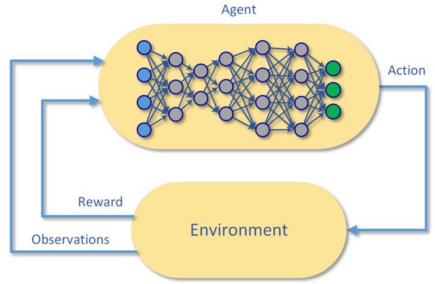


1. Value-based algorithms:
    a. SARSA
    b. Q-Learning
2. Policy Gradient approaches:
    a. Reinforce
    b. Actor-Critic

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s,\theta]$$

Possible approaches



1. Value-based algorithms:
    a. SARSA
    b. Q-Learning
2. Policy Gradient approaches:
    a. Reinforce
    b. Actor-Critic

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

Today we'll focus on this!

$$\pi_\theta(s,a) = \mathbb{P}[a \mid s,\theta]$$

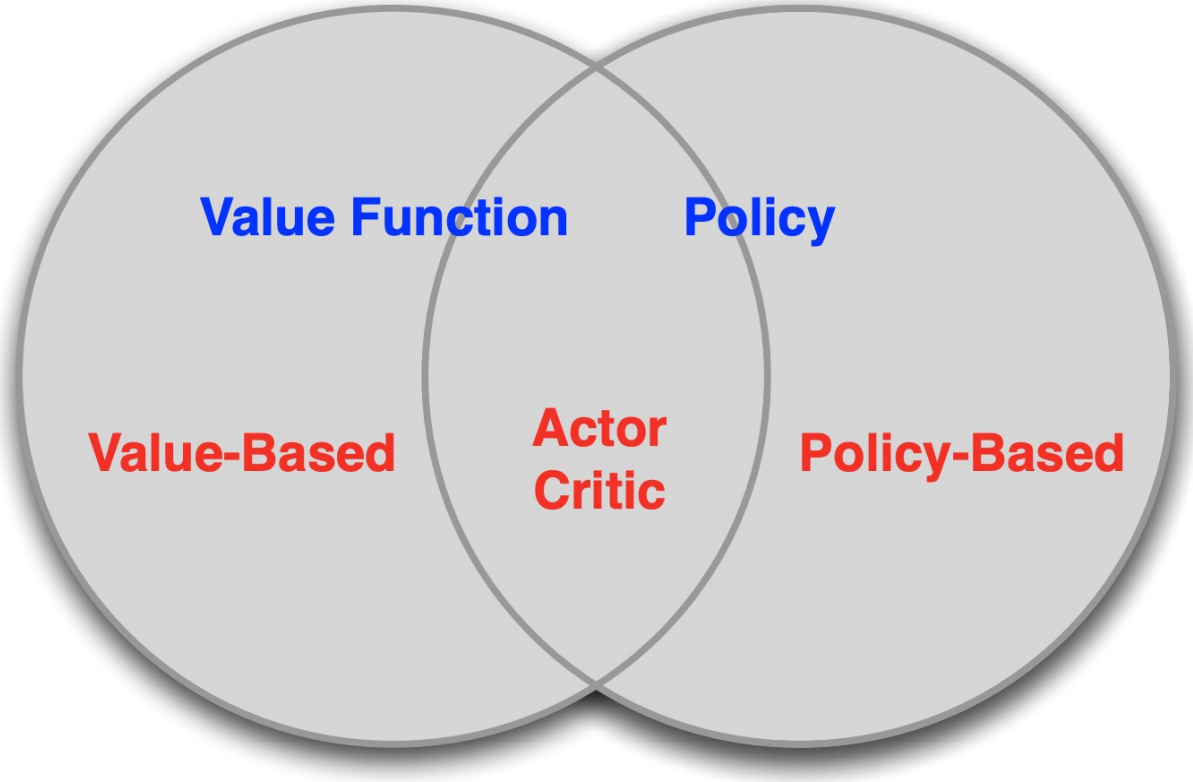
Recap – Policy-based Approaches

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance



D. Silver 'RL course' @ UCL (2015)

Recap – Policy Gradient / REINFORCE

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a)$$

We now have a parametrized policy (for example a soft-max in action preferences – 13.2 of the book)

Parametrizations can be linear functions of the states, but tipycally we resort to Neural Networks/Deep Learning architectures!

Recap – Policy Gradient / REINFORCE

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

Recap – Policy Gradient / REINFORCE

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \boxed{\frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}}$$

(score function)

$$\boxed{\nabla \ln \pi(a|s, \boldsymbol{\theta})}$$

Recap – Policy Gradient / REINFORCE

- we select an action;
- we “reinforce it” (proportionally to the return)

Assuming all $G_t > 0$, then all actions will be reinforced, but the one with the highest return will prevail, pushing the other ones closer to 0.

$$\nabla J(\boldsymbol{\theta}) \propto \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$= \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right]$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}$$

(score function)

$$\nabla \ln \pi(a|s, \boldsymbol{\theta})$$

Can’t we just use a table?

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

Can’t we just use a table?

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

No, dimensionality gets to |S|x|A|! (Policy gradient still works: nothing stops us from representing the $h(s, a, \theta)$ as just entries in a table, and use Policy gradient to learn such entries)

Can't we just use a table?

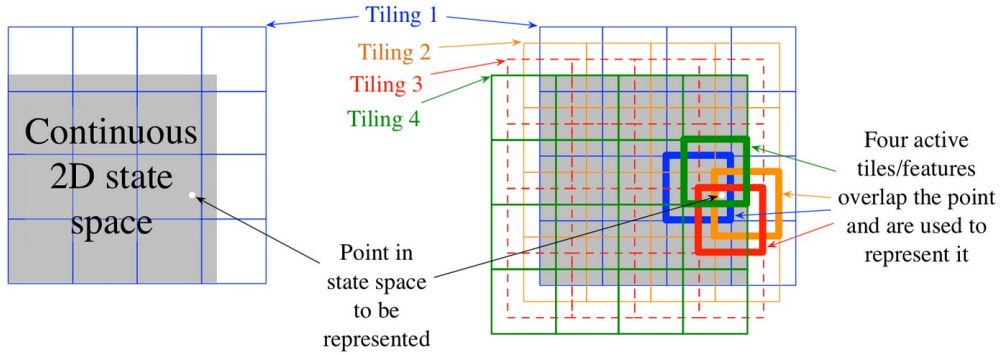
$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

No, dimensionality gets to |S|x|A|! (Policy gradient still works: nothing stops us from representing the $h(s,a,\theta)$ as just entries in a table, and use Policy gradient to learn such entries)

However, states usually are similar in some ways, and we can exploit this property! This will allow us to "fill the table" without actually having to sample multiple samples per entry.

We can generalize to extrapolate that information (similar states will have similar consequences, if the similarity is learnt in the "consequence" space)

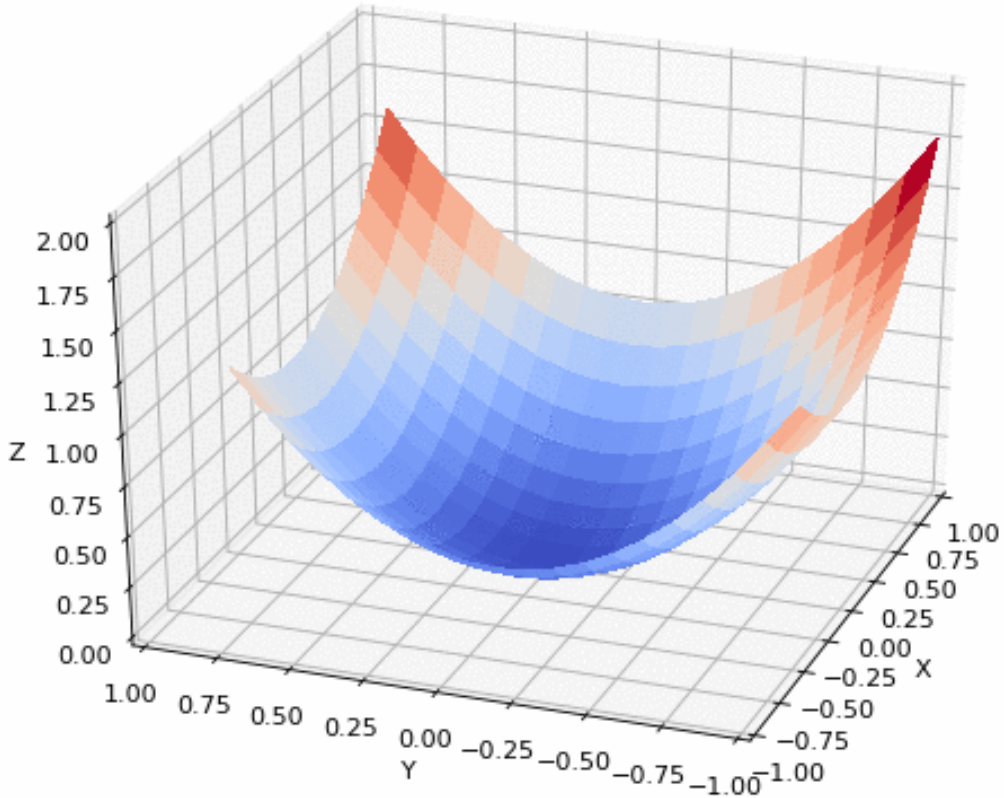
Maybe linear is the way...



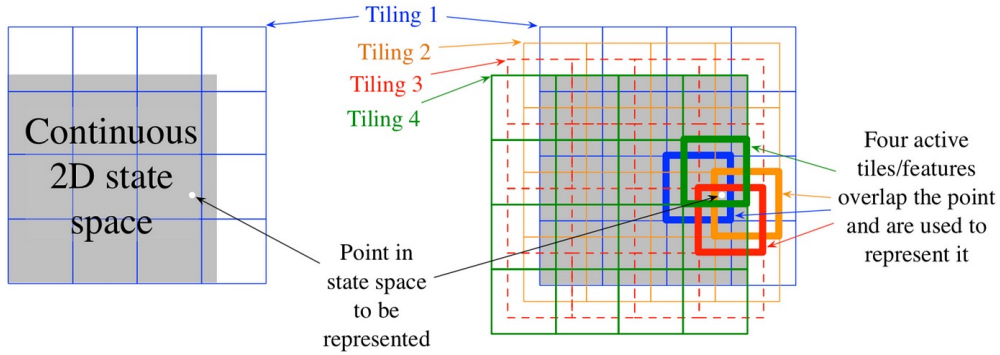
We have seen that we can parametrize the state in different ways (ie. tile coding), and assume there is some sort of linear correlation between the π and s

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a)$$



Maybe linear is the way...



We have seen that we can parametrize the state in different ways (ie. tile coding), and assume there is some sort of linear correlation between the π and s

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}$$

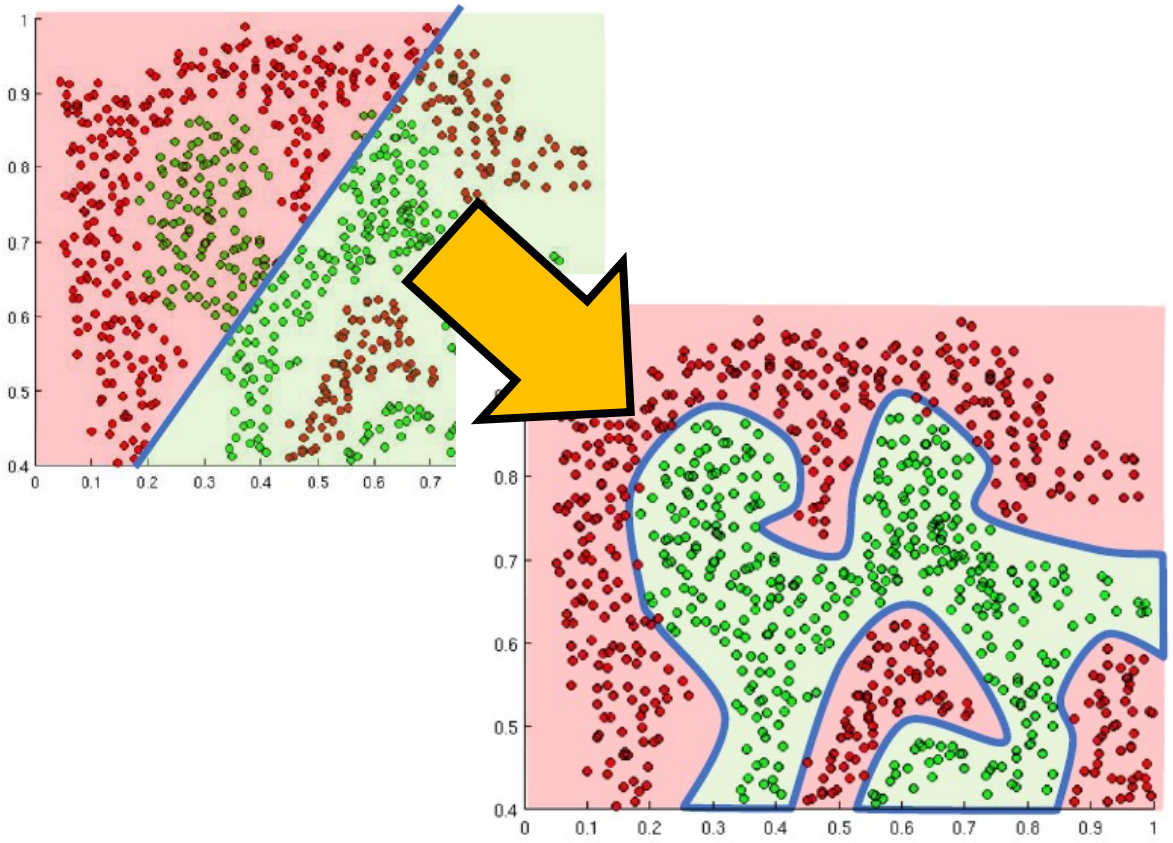
$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a)$$

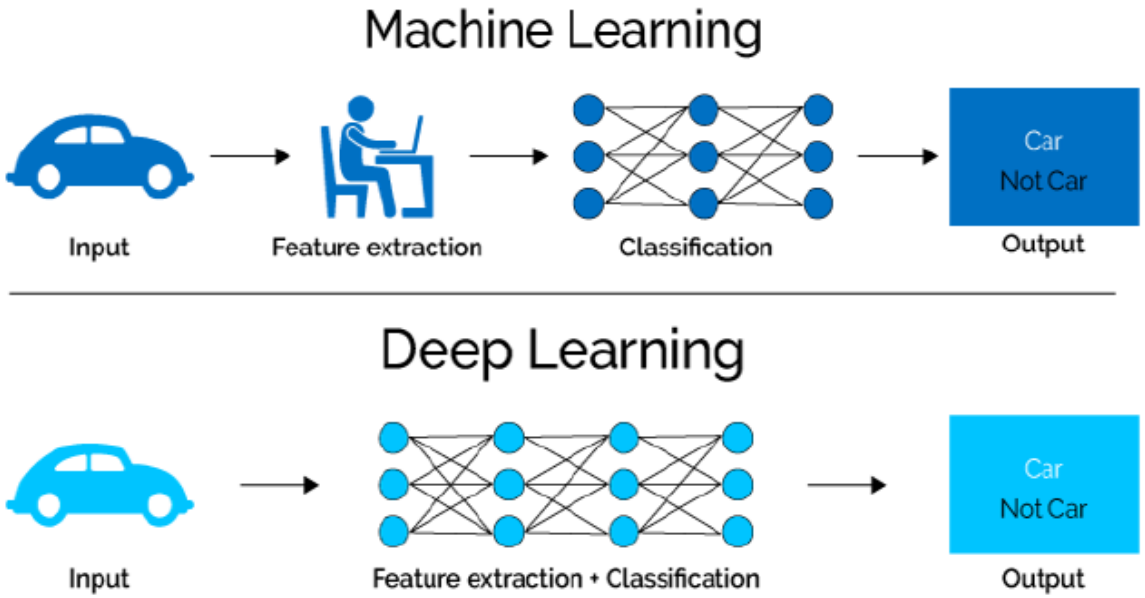
- s is something that we can build
- we can choose to build *s* such that being close in the state representation space, you are also close in the "consequence space"

Example: in chess, very similar boards might be completely different, but very different boards, might at the end be pretty much equivalent, if the differences are irrelevant)

Maybe linear is the way… or maybe not!

- Goal: learn a good representation of *s*, that we can learn
- Neural Networks are amazing generalization machine





Long life to non-convex function approximators and SGD! However... (1/2)

Given that, now we are left with the non-trivial problem to make the Reinforcement Learning (RL) and Deep Learning (DL) worlds work together.

However, it’s easier to be said than to be done, as there is a very big friction between the two approaches (RL and DL)... which one?

Long life to non-convex function approximators and SGD! However... (1/2)

Given that, now we are left with the non-trivial problem to make the Reinforcement Learning (RL) and Deep Learning (DL) worlds work together.

However, it's easier to be said than to be done, as there is a very big friction between the two approaches (RL and DL)... which one?

I.I.D. data

Indeed, RL until now didn't have to make any big assumption on the distribution of the data, as tabular cases are immune to correlation between samples... well, not SGD (and thus not neural networks)
(This is the reason why you saw Experience Replay being used in DQN)

Long life to non-convex function approximators and SGD! However... (2/2)

However, we cannot use experience reply in policy gradient, as that would imply using off-policy data, which is not really the best... for another reason:

Long life to non-convex function approximators and SGD! However... (2/2)

However, we cannot use experience reply in policy gradient, as that would imply using off-policy data, which is not really the best... for another reason:

VARIANCE

(for example with importance sampling coeff)

$$\prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

If $b \ll \pi$ one step can overshadow the gradient from all other samples!

Long life to non-convex function approximators and SGD! However... (2/2)

However, we cannot use experience reply in policy gradient, as that would imply using off-policy data, which is not really the best... for another reason:

VARIANCE

(for example with importance sampling coeff)

$$\prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

If $b \ll \pi$ one step can overshadow the gradient from all other samples!

This is a problem also for value function data, but now is worse since we not messing up with a single entry on the table, but with the whole rapresentation of the policy!

So, can’t we just use the policy gradient we studied?

Yes and no. You have to be very careful with the two problems just stated. The rule of thumb is: if there is a way to do the same with less variance, then do it that way

How can we introduce variance?

So, can't we just use the policy gradient we studied?

Yes and no. You have to be very careful with the two problems just stated. The rule of thumb is: if there is a way to do the same with less variance, then do it that way

How can we introduce variance?

1. Reward function: indeed G_t might have high variance, and when you do $G_t \nabla \ln(\pi(a|s))$, the gradient for that sample is weighted according to that scalar... if you have 100 samples in your minibatch, and one has a return 100x bigger, the gradient of that sample will overshadow the gradient produced by the others

Solution #01: Define the reward function accurately!

So, can’t we just use the policy gradient we studied?

Yes and no. You have to be very careful with the two problems just stated. The rule of thumb is: if there is a way to do the same with less variance, then do it that way

How can we introduce variance?

1. Reward function

2. Stochastic optimization: NN are just differentiable models, and being non-convex, means to having to deal with a lot of bad scenarios… in addition to poor conditioning, you might have cliffs, exploding gradients, flat regions, ill-conditioning

Solution #02: Use advanced optimizers (that use adaptive steps and momentum)

So, can’t we just use the policy gradient we studied?

Yes and no. You have to be very careful with the two problems just stated. The rule of thumb is: if there is a way to do the same with less variance, then do it that way

How can we introduce variance?

1. Reward function
2. Stochastic optimization
3. Importance Sampling ratio: if using off-policy data is nice, but the importance sampling ratio might explode if the denominator is much smaller than the numerator

(Solution #03): Pay attention to the algorithm that you use, if it handles this case or not

So, can’t we just use the policy gradient we studied?

Summarizing:

1. Reward function: redefine the reward function to be well behaved, normalize the returns, or use more advanced approaches proposed in literature for DeepRL (PopArt, Value Normalization)

2. Stochastic optimization: the family of momentum based optimizers and adaptive gradient optimizers tries to solve this problem since 20 years, Adam is always your best friend (of if you really cannot converge, maaaybe consider K-fac, if your library has it available)

3. Importance Sampling ratio: can we do anything about it? More in the following...

Getting rid of Importance Sampling: on-policy approaches (for example Actor Critic)

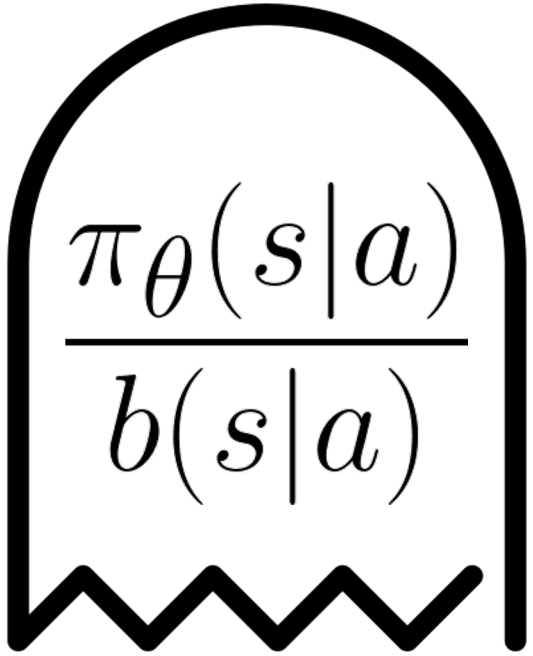
Actor critic/Advantage Actor critic (aka A2C), is still a very good algorithm for DeepRL, however, the fact that we have to discard the data after one step of learning, is kind of bummer.

Can we do better?

Getting rid of Importance Sampling: on-policy approaches (for example Actor Critic)

Actor critic/Advantage Actor critic (aka A2C), is still a very good algorithm for DeepRL, however, the fact that we have to discard the data after one step of learning, is kind of bummer.

Can we do better? Why not reusing the data? Or not?



BOHOHOOOHOO

(if b<<π)

Special Thanks to Alberto Sinigaglia!

Tackling Importance Sampling Weights

How can reuse the data?

1. sample data from the environment using $\boldsymbol{\pi}$

Critic

2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1-d)\gamma V_\phi(s'))])^2$$

Actor

3. update Policy function using A2C

$$L(\theta) = \sum (V_\phi(s) - (r + (1-d)\gamma V_\phi(s')))) \nabla \ln(\pi_\theta(a|s))$$

$$= \sum A_t \nabla \ln(\pi_\theta(a|s))$$

Tackling Importance Sampling Weights

How can reuse the data?
1. sample data from the environment using $\boldsymbol{\pi}$
2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1-d)\gamma V_\phi(s'))])^2$$

3. update Policy function using A2C

$$L(\theta) = \sum (V_\phi(s) - (r + (1-d)\gamma V_\phi(s'))))\nabla \ln(\pi_\theta(a|s))$$
$$= \sum A_t \nabla \ln(\pi_\theta(a|s))$$

By reusing data, we are summing over different data points

Tackling Importance Sampling Weights

How can reuse the data?
1. sample data from the environment using $\boldsymbol{\pi}$
2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1-d)\gamma V_\phi(s')])^2$$

3. update Policy function using A2C

'sg' = 'stop gradient'
We want to update the weights only on the term on the left to avoid a sort of recursion. We freeze the weights on the right term

$$L(\theta) = \sum (V_\phi(s) - (r + (1-d)\gamma V_\phi(s')))) \nabla \ln(\pi_\theta(a|s))$$
$$= \sum A_t \nabla \ln(\pi_\theta(a|s))$$

Tackling Importance Sampling Weights

How can reuse the data?
1. sample data from the environment using $\boldsymbol{\pi}$
2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1 - d)\gamma V_\phi(s'))])^2$$

3. update Policy function using A2C

$$L(\theta) = \sum (V_\phi(s) - (r + (1 - d)\gamma V_\phi(s'))))\nabla \ln(\pi_\theta(a|s))$$
$$= \sum A_t \nabla \ln(\pi_\theta(a|s))$$

'd' = 'done'
Equal to 1 if the state is terminal (so we get a zero on the right)

Tackling Importance Sampling Weights

How can reuse the data?
1. sample data from the environment using $\boldsymbol{\pi}$
2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1-d)\gamma V_\phi(s'))])^2$$

3. update Policy function using A2C

Here we need to use the data multiple times, however, after the first update, the policy will not be anymore the same

$$L(\theta) = \sum (V_\phi(s) - (r + (1-d)\gamma V_\phi(s')))) \nabla \ln(\pi_\theta(a|s))$$
$$= \sum A_t \nabla \ln(\pi_\theta(a|s))$$

Tackling Importance Sampling Weights

How can reuse the data?
1. sample data from the environment using $\boldsymbol{\pi}$
2. update Value function using TD learning ('sg' = stop gradient, 'd' = done)

$$L(\phi) = \sum (V_\phi(s) - sg[r + (1-d)\gamma V_\phi(s'))])^2$$

3. update Policy function using A2C (N steps of gradient descent)

$$L(\theta) = \sum \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_t \nabla \ln(\pi_\theta(a|s))$$

$$= \sum \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A_t \frac{\nabla \pi_\theta(a|s)}{\pi_\theta(a|s)} = \sum \frac{A_t}{\pi_{\theta_{\text{old}}}(a|s)} \nabla \pi_\theta(a|s)$$

We introduced importance sampling.
We didn't solve the problem, we just hided it in the simplification... if we set a N too large, we still have the variance

Trust Region Policy Optimization (TRPO)

TRPO was introduced to tackle this problem. Instead of just optimizing that loss, it wants to constrain the problem in a safe area of search (in the parameter space), thus solving the same problem:

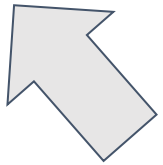
$$\underset{s.t.||\theta-\theta_{\text{old}}||_2^2 \leq \epsilon}{minimize} \sum A_t \frac{\nabla \pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

The problem is that staying close in the parameter space, doesn't mean to stay close in the policy space

Trust Region Policy Optimization (TRPO)

TRPO was introduced to tackle this problem. Instead of just optimizing that loss, it wants to constrain the problem in a safe area of search (in the parameter space), thus solving the same problem:

$$\underset{s.t. D_{KL}(\pi_\theta || \pi_{\theta_{\text{old}}}) \leq \epsilon}{minimize} \sum A_t \frac{\nabla \pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$



Ideally we want this or some other distance measure in the distribution space, however, it's definitely not clear how to enforce such constraint

Trust Region Policy Optimization (TRPO)

- This method will solve this problem by using several relaxation

- For example, will linearize the KL via 2 order taylor expansion, will use the Fisher information matrix instead of the hessian, and use Conjugate gradients to calculate the inverse of the conditioning matrix... for more informations, check the paper “Natural policy gradient” [1] and then the paper of TRPO [2]

- We have a method that is very robust, however we also have to pay the price for such robustness, and that price is that it’s very computational expensive, however, some times the overhead of this method is lower then the price of generating new data, thus might be very convenient.

[1] Kakade, Sham M. "A natural policy gradient." Advances in neural information processing systems 14 (2001).
[2] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

Trust Region Policy Optimization (TRPO)

'Classic' approximation based on data availability

Since estimating that KL term is too complicated, they first approximate it heuristically:

$$D_{KL}(\pi_\theta||\pi_{\theta'}) = \mathbb{E}_{\mu_{\pi_\theta}(s)}[D_{KL}(\pi_\theta(a|s)||\pi_{\theta'}(a|s))]$$

At this point, we further need to simplify it, considering an approximation:

$$\mathbb{E}_{\mu_{\pi_\theta}(s)}[D_{KL}(\pi_\theta(a|s)||\pi_{\theta'}(a|s))] \approx (\theta - \theta')^T F(\theta - \theta')$$

With F being the Fisher information matrix:

$$F = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a|s) \nabla_\theta \log \pi_\theta(a|s)^T]$$

Trust Region Policy Optimization (TRPO)

We start with a quadratic form in weight space, and now we have a quadratic form in policy space:

$$||\theta - \theta'||^2 = (\theta - \theta')I(\theta - \theta')^T \quad \Longrightarrow \quad \mathbb{E}_{\mu_{\pi_\theta}(s)}[D_{KL}(\pi_\theta(a|s)||\pi_{\theta'}(a|s))] \approx (\theta - \theta')^T F(\theta - \theta')$$

And the solution is given by the following equation (with some caution for the stepsize to stay in the trust region):

$$\theta' = \theta + \alpha F^{-1} \nabla_\theta J(\theta)$$

(This is also done by Natural Gradient, but TRPO uses conjugate gradient to calculate the inverse, and in the meantime also obtaining the stepsize)

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Algorithm 1 Trust Region Policy Optimization

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, \ldots K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

'Doable'... but a real mess!

TRPO Relaxed

One way of solving constrained optimization (at least with equalities) is to use Lagrange multipliers, and indeed we can do the same here:

$$L(\theta) = \sum A_t \frac{\nabla \pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} - \lambda D_{KL}(\pi_\theta || \pi_{\theta_{\text{old}}})$$

However, **λ** it's quite tricky to pick... too low and you have variance problem, too high and you don't learn at all. What we can do is to use dual gradient descent:

1. we optimize the loss **L**
2. we adaptively set **λ**: if the KL is too high, we increase **λ**, if too low, we relax it

TRPO Relaxed

Similarly to what you may have seen in regularization...

One way of solving constrained optimization (at least with equalities) is to use Lagrange multipliers, and indeed we can do the same here:

$$L(\theta) = \sum A_t \frac{\nabla \pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} - \lambda D_{KL}(\pi_\theta || \pi_{\theta_{\text{old}}})$$

However, **λ** it's quite tricky to pick... too low and you have variance problem, too high and you don't learn at all. What we can do is to use dual gradient descent:

1. we optimize the loss **L**
2. we adaptively set **λ**: if the KL is too high, we increase **λ**, if too low, we relax it

Proximal Policy Optimization (PPO)

Let’s consider again what’s the problem… the source of the variance are the samples where we are going too far from the old policy. Can we do better than staying close to the reference?

Proximal Policy Optimization (PPO)

Let’s consider again what’s the problem… the source of the variance are the samples where we are going too far from the old policy. Can we do better than staying close to the reference?

What about neglecting the data sample we went too far away from?

Proximal Policy Optimization (PPO)

Let's consider again what's the problem... the source of the variance are the samples where we are going too far from the old policy. Can we do better than staying close to the reference?

What about neglecting the data sample we went too far away from?

And indeed, it's what PPO will do, but let's see when to do it:

- if we have $A_t < 0$, the next learning step, most likely the probability of taking that action will be decreased, thus the IS ratio will be less than 1, not causing any problem
- if we have $A_t > 0$, the next learning step, most likely the probability of taking that action will be increased, thus the IS ratio will be greater than 1... here's the problem! We will use clipping to get rid of this

Why PPO use ‘clipping’ (saturation of ‘big steps’)

In PPO, the goal is to update the policy without making overly big steps. If the new policy changes too much relative to the old one, then:
- the agent’s behavior can become unstable,
- performance can crash,
- training may diverge.

To prevent this, PPO applies clipping to the probability ratio:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

This ratio tells us how much the new policy increases or decreases the probability of an action compared to the old policy. The purpose of clipping is to keep policy updates within a small, safe range (typically ±0.2).

Why the Problem Happens Mainly When the Advantage Is Positive

A: how much better/worse an action was compared to the average.
- A<0 → action was bad: we want to decrease its probability.
- A>0 → action was good: we want to increase its probability.

$$r(\theta)A$$

Case 1 – Negative Advantage (A < 0)
- The algorithm naturally tries to push r below 1.
- There is no rish the algorithm pushes to have big r: making r big will lead to worsening the loss

Case 2 – Positive Advantage (A > 0)
- The algorithm naturally tries to make r big!
- We can prevent this with clipping:

$$r(\theta) \leq 1 + \epsilon$$

Proximal Policy Optimization (PPO)

Since we just saw that $A_t > 0$ is the case were we might have problem, we just need to fix it, and indeed PPO does exactly that (r_t is just the IS ratio):

$$J(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \cdot \hat{A}_t \right) \right]$$

The reasoning above works out to be just the second term in the **min**, however we did not considered that even though we might have $A_t < 0$, the probability at the second step of learning might actually have increased... in that case the min is used to avoid clipping that sample, and to still consider it.

Keep in mind that then the **min** term is the **clip**, if you actually clip the loss, **the gradient will be 0** (in other words, if you are too far from the old policy in that state, you stop learning from it)

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Now, a question for all of you:
Can we use this approaches in continuous action spaces?

Or even infinite discrete actions, for example for a Poisson policy

Can we use this approaches in continuous action spaces?

Well, theoretically, yes, practically, hardly not.

Can we use this approaches in continuous action spaces?

Well, theoretically, yes, practically, hardly not.

- policy-based: well now we have a problem... if earlier we could just have used a categorical distribution, now it's hard to have a concept of continuous distribution that can just model any distribution, so we need to rely on a family of distributions (gaussians), and that's not obvious at all (maybe we are in a continuous bounded action space)

Can we use this approaches in continuous action spaces?

Well, theoretically, yes, practically, hardly not.

- policy-based: well now we have a problem… if earlier we could just have used a categorical distribution, now it's hard to have a concept of continuous distribution that can just model any distribution, so we need to rely on a family of distributions (gaussians), and that's not obvious at all (maybe we are in a continuous bounded action space)

Can we use this approaches in continuous action spaces?

Well, theoretically, yes, practically, hardly not.

- policy-based: well now we have a problem... if earlier we could just have used a categorical distribution, now it's hard to have a concept of continuous distribution that can just model any distribution, so we need to rely on a family of distributions (gaussians), and that's not obvious at all (maybe we are in a continuous bounded action space)
- value-base: well, even if learning **Q(s,a)** might theoretically work, the problem is that **Q** is learnt under a policy, but now we cannot have a ϵ-greedy policy anymore, so DQN like methods cannot be easily applied

Next lecture we will see how to actually solve this issue, by using the ‘nature’ of neural networks

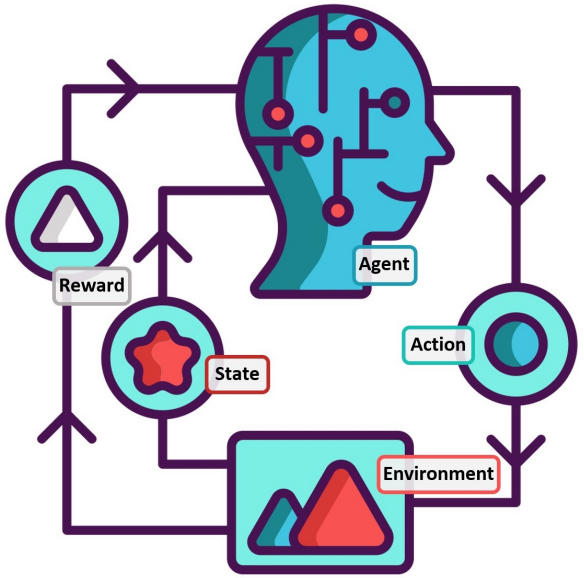
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning 2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Thank you!
Questions?

Lecture #21:
DeepRL: the Discrete Case



Gian Antonio Susto

Special thanks to Alberto Sinigaglia!