Offline vs. Online/ Batch Reinforcement Learning | Yessmine

Deep Reinforcement Learning agents **learn with batches of experience**. The question is, how do they collect it?

Offline vs. Online Reinforcement Learning

Online RL

• In *online reinforcement learning*, the agent **gathers data directly**: it collects a batch of experience by **interacting with the environment**. Then, it uses this experience immediately (or via some replay buffer) to learn from it (update its policy).

Implementation: **train your agent directly in the real world or have a simulator**

> Disadvantage: If you don't have a simulator, you need to build it, which can be very complex **(how to reflect the complex reality of the real world in an environment?)**

Offline RL

• On the other hand, in *offline reinforcement learning*, the agent only **uses data collected from other agents or human demonstrations**. It does **not interact with the environment**.

Implementation:

1. **Create a dataset** using one or more policies and/or human interactions.
2. Run **offline RL on this dataset** to learn a policy

   Disadvantage: What do we do if our agent **decides to do something for which we don't have the data?** For instance, turning right on an intersection but we don't have this trajectory. → **watch this video**

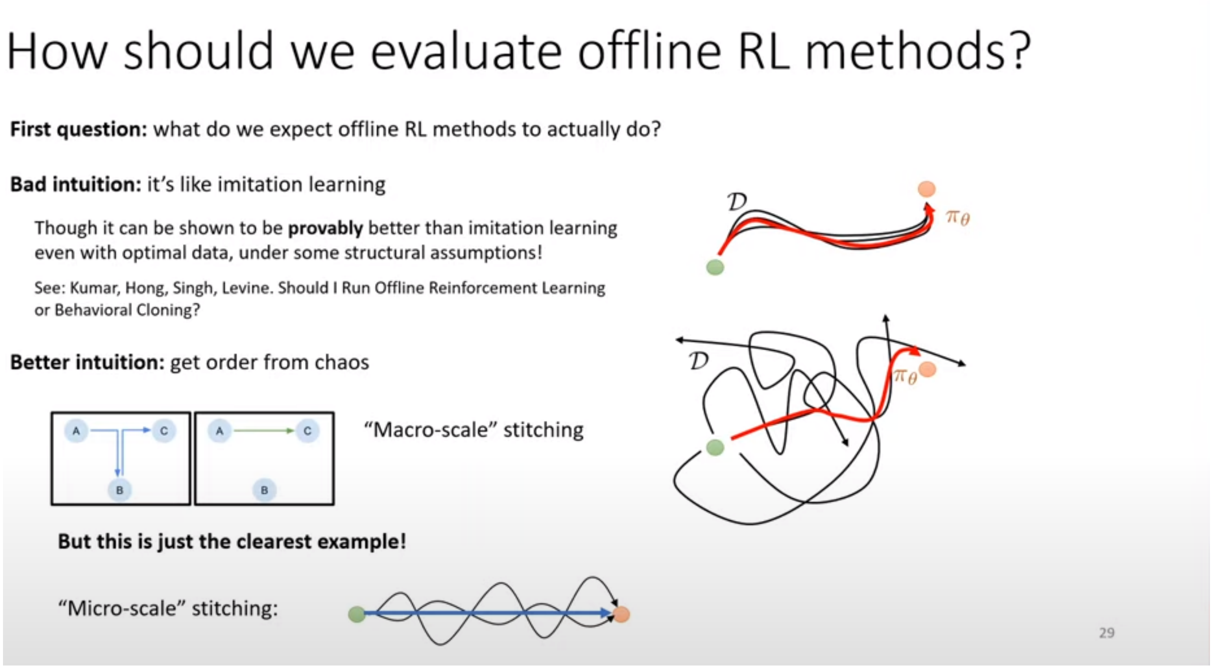
   Challenges:

   - Online → Try and Error vs Offline → data set must cover edge cases

- Which Policy to start with?

▼ Unsupervised ml vs Offline RL

1. I**n reinforcement learning the agent receives what is termed "evaluative feedback" in terms of a scalar reward, which gives the agent some feedback of the quality of the action that was taken but it does not tell the agent if this action is the optimal action or not. Contrast this with supervised learning where the agent receives what is termed "instructive feedback": for each prediction that the learner makes, it receives a feedback (a label) that says what the optimal action/prediction was.** The differences between instructive and evaluative feedback is detailed in Rich Sutton's book in the first chapters. Essentially reinforcement learning is optimization with sparse labels, for some actions you may not get any feedback at all, and in other cases the feedback may be delayed, which creates the credit-assignment problem.
2. In reinforcement learning you have a temporal aspect where the goal is to find an optimal policy that maps states to actions over some horizon (number of time-steps). If the horizon T=1, then it is just a one-off prediction problem like in supervised learning, but if T>1 then it is a sequential optimization problem where you have to find the optimal action not just in a single state but in multiple states and this is further complicated by the fact that the actions taken in one state can influence which actions should be taken in future states (i.e. it is dynamic).
3. In supervised learning there is a fixed i.i.d distribution from which the data points are drawn (this is the common assumption at least). In RL there is no fixed distribution, rather this distribution depends on the policy that is followed and often this distribution is not i.i.d but rather correlated.


How should we evaluate offline RL methods?

First question: what do we expect offline RL methods to actually do?

Bad intuition: it's like imitation learning

Though it can be shown to be **provably** better than imitation learning even with optimal data, under some structural assumptions!

See: Kumar, Hong, Singh, Levine. Should I Run Offline Reinforcement Learning or Behavioral Cloning?

Better intuition: get order from chaos

"Macro-scale" stitching

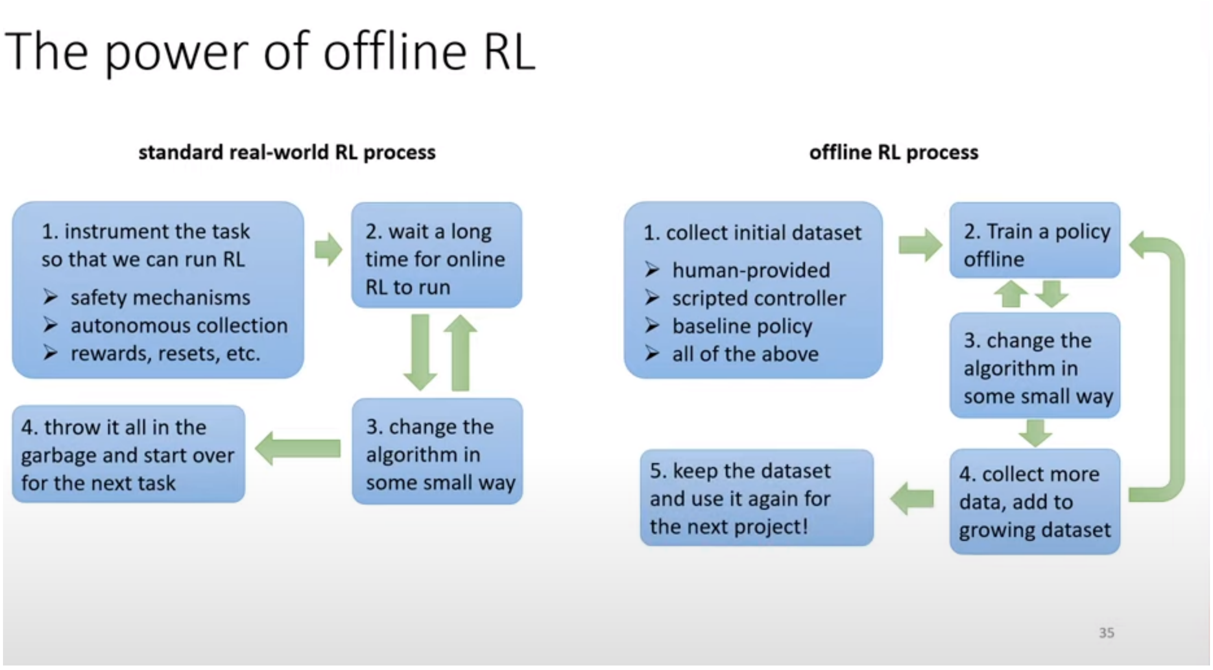
But this is just the clearest example!

"Micro-scale" stitching:

29

Take the good part of each trajectory and combine it

Comparing


The power of offline RL

standard real-world RL process

1. instrument the task so that we can run RL
   - safety mechanisms
   - autonomous collection
   - rewards, resets, etc.
2. wait a long time for online RL to run
3. change the algorithm in some small way
4. throw it all in the garbage and start over for the next task

offline RL process

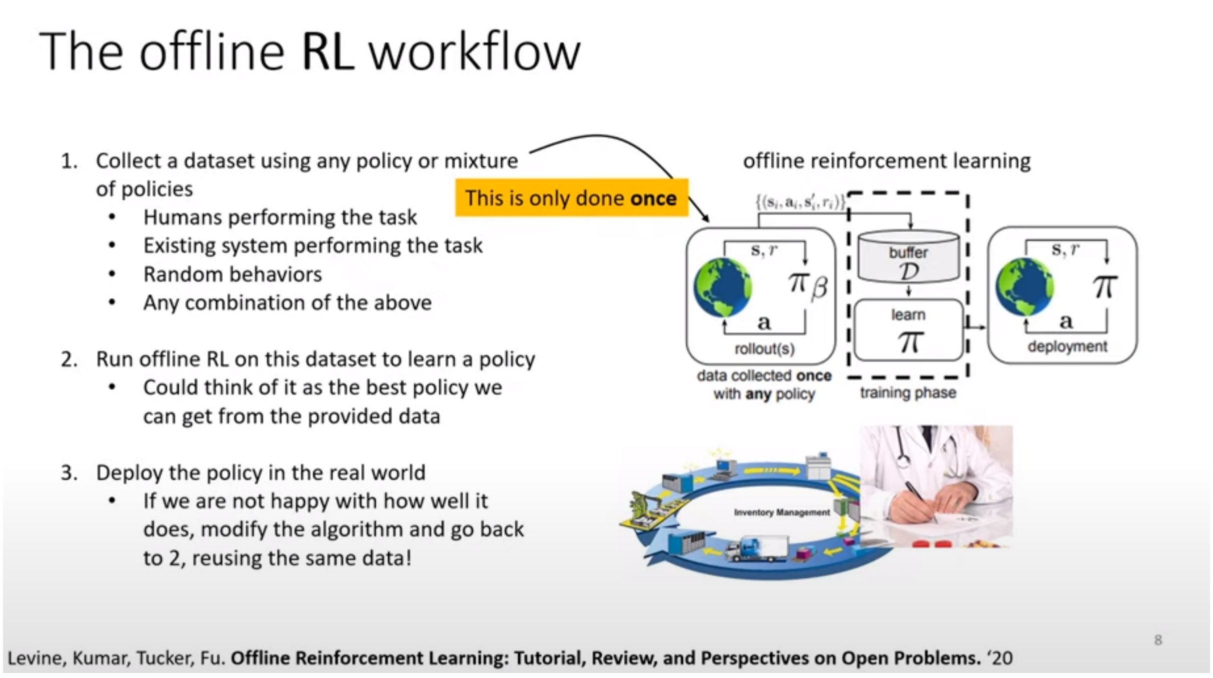
1. collect initial dataset
   - human-provided
   - scripted controller
   - baseline policy
   - all of the above
2. Train a policy offline
3. change the algorithm in some small way
4. collect more data, add to growing dataset
5. keep the dataset and use it again for the next project!

35

Offline RL: Recollect data but only what's missing and add it to the dataset
Offline → only do 1. once*

Aspect	Batch Learning	Online Learning
Learning Approach	Trains on fixed dataset offline	Adapts to incoming data online
Resource Requirements	High (CPU, memory, storage)	Low (fast and efficient)
Real-time Adaptation	No	Yes
Handling Large Datasets	Challenging for huge datasets	Handles large datasets efficiently
Learning Rate Control	Not applicable	Important for adaptation speed
Anomaly Handling	Rarely handles anomalies	Requires monitoring and handling

Offline RL Workflow



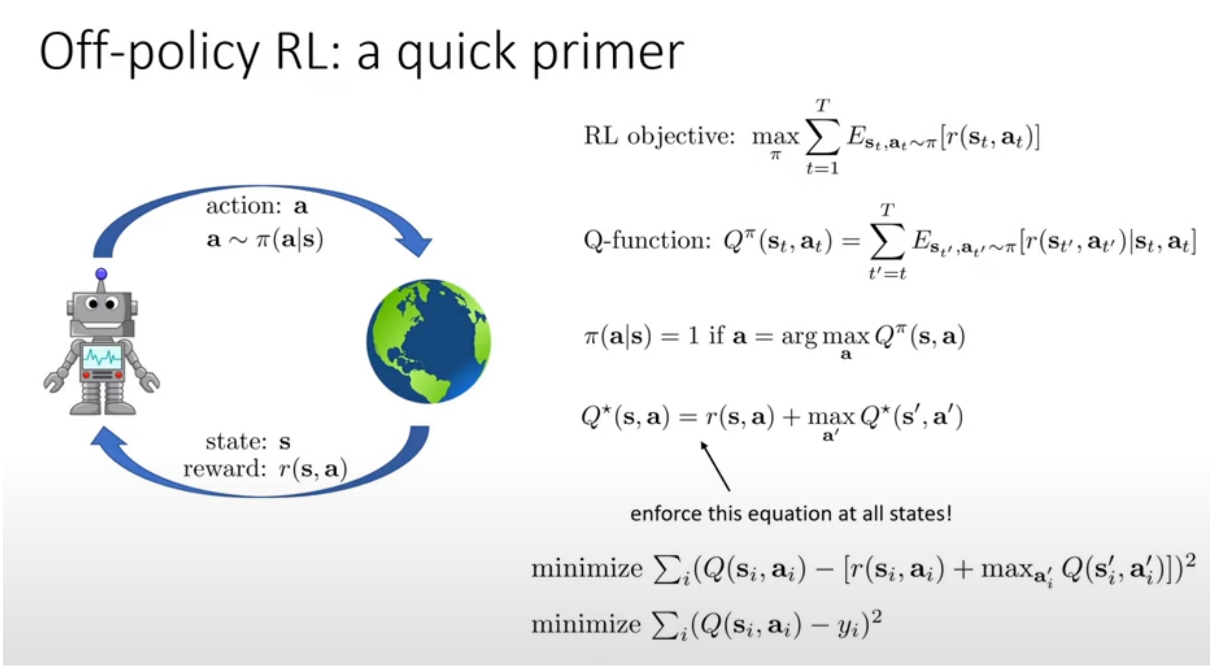
The offline RL workflow

1. Collect a dataset using any policy or mixture of policies
   - Humans performing the task
   - Existing system performing the task
   - Random behaviors
   - Any combination of the above
2. Run offline RL on this dataset to learn a policy
   - Could think of it as the best policy we can get from the provided data
3. Deploy the policy in the real world
   - If we are not happy with how well it does, modify the algorithm and go back to 2, reusing the same data!

This is only done **once**

8

Levine, Kumar, Tucker, Fu. **Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.** '20


action: $\mathbf{a}$
$\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$
state: $\mathbf{s}$
reward: $r(\mathbf{s}, \mathbf{a})$

Off-policy RL: a quick primer

RL objective: $\max_\pi \sum_{t=1}^{T} E_{\mathbf{s}_t, \mathbf{a}_t \sim \pi}[r(\mathbf{s}_t, \mathbf{a}_t)]$

Q-function: $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\mathbf{s}_{t'}, \mathbf{a}_{t'} \sim \pi}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]$

$\pi(\mathbf{a}|\mathbf{s}) = 1 \text{ if } \mathbf{a} = \arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$

$Q^\star(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \max_{\mathbf{a}'} Q^\star(\mathbf{s}', \mathbf{a}')$

enforce this equation at all states!

minimize $\sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \max_{\mathbf{a}'_i} Q(\mathbf{s}'_i, \mathbf{a}'_i)])^2$

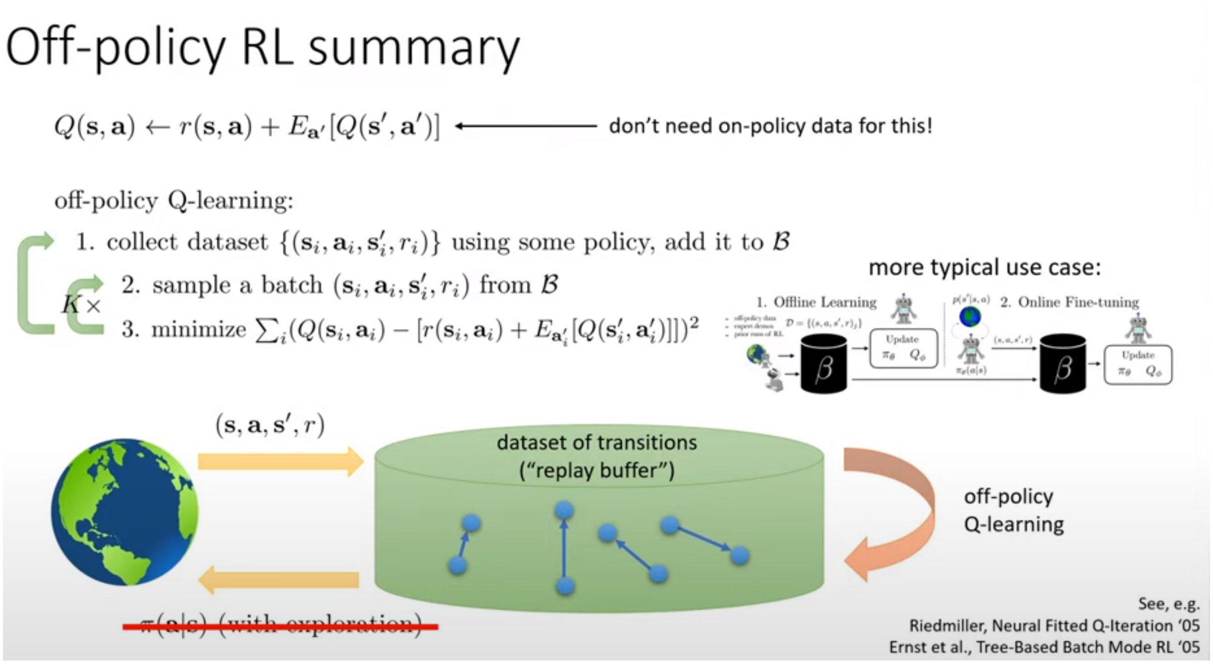
minimize $\sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - y_i)^2$

RL obj: Maximize the cummulative expected sum of rewards

How? Q-function (This function tells us how good it is to take action) = expected cummulative sum of rewards from now using policy (pi) → Optimal Q function(Q*) = Minimizing the difference between both sides (current Q and target value) = current Reward + Max(next q value) → enforce this in all states → Minimizing the difference between both sides = Minimizing square(current Q - (target value))

(si,ai) = transition 1

(s'i,a'i) = transition 2



Off-policy RL summary

$Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + E_{\mathbf{a}'}[Q(\mathbf{s}',\mathbf{a}')]$ ← don't need on-policy data for this!

off-policy Q-learning:

```
1. collect dataset $\{(\mathbf{s}_i,\mathbf{a}_i,\mathbf{s}'_i,r_i)\}$ using some policy, add it to $\mathcal{B}$
    $K\times$
    2. sample a batch $(\mathbf{s}_i,\mathbf{a}_i,\mathbf{s}'_i,r_i)$ from $\mathcal{B}$
    3. minimize $\sum_i (Q(\mathbf{s}_i,\mathbf{a}_i) - [r(\mathbf{s}_i,\mathbf{a}_i) + E_{\mathbf{a}'_i}[Q(\mathbf{s}'_i,\mathbf{a}'_i)]])^2$
```

more typical use case:

See, e.g.
Riedmiller, Neural Fitted Q-Iteration '05
Ernst et al., Tree-Based Batch Mode RL '05

B: buffer

s,a,s',r: transition state,action,next state, reward