Reinforcement Learning

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

Recap

- Sometimes, it is better to learn a stochastic policy than a deterministic policy
- Policy gradient methods suffer from poor sample efficiency
- Importance sampling can help alleviate this problem
- However, we need to make sure that current policy is not far from policy with which we have collected trajectories
- Small changes in the policy parameters can unexpectedly lead to big changes in the policy.
- TRPO uses importance sampling to take multiple gradient steps and constrains the optimization objective in the policy space
- PPO does the same but approximately enforces KL constraint without computing natural gradients.

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model

Model-Based and Model-Free RL

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model
- So far, we have only looked at model-free reinforcement learning

What is a Model?

- A model $\mathcal{M}$ is a representation of an MDP $< S, A, P, R >$
- We will assume state space S and action space A are known
- So a model $M =< P_\phi, R_\phi >$ represents state transitions $P_\phi \approx P$ and rewards $R_\phi \approx R$

$$s_{t+1} \sim P_\phi(s_{t+1}|s_t, a_t)$$

$$r_{t+1} = R_\phi(r_{t+1}|s_t, a_t)$$

What is a Model?

- A model $\mathcal{M}$ is a representation of an MDP $< S, A, P, R >$
- We will assume state space S and action space A are known
- So a model $M =< P_\phi, R_\phi >$ represents state transitions $P_\phi \approx P$ and rewards $R_\phi \approx R$

$$s_{t+1} \sim P_\phi(s_{t+1}|s_t, a_t)$$

$$r_{t+1} = R_\phi(r_{t+1}|s_t, a_t)$$

- We can also only learn one of state transitions and reward model

Model Learning

- **Goal:** Estimate model M_ϕ from experience $s_1, a_1, r_1, \cdots, s_T$

Model Learning

- **Goal:** Estimate model M_ϕ from experience $s_1, a_1, r_1, \cdots, s_T$
- This is a supervised learning problem

Model Learning

- **Goal:** Estimate model M_ϕ from experience $s_1, a_1, r_1, \cdots, s_T$
- This is a supervised learning problem
- Learning $s, a \rightarrow r$ is a regression problem
- Learning $s, a \rightarrow s'$ is a density estimation problem

Model Learning

- **Goal:** Estimate model M_ϕ from experience $s_1, a_1, r_1, \cdots, s_T$
- This is a supervised learning problem
- Learning $s, a \rightarrow r$ is a regression problem
- Learning $s, a \rightarrow s'$ is a density estimation problem
- We pick a loss function like MSE, KL Divergence etc., and then optimize

Model-Based RL

- Key Idea:
  - It would be useful if we could approximately simulate the world! i.e. if we could predict the consequences of our actions

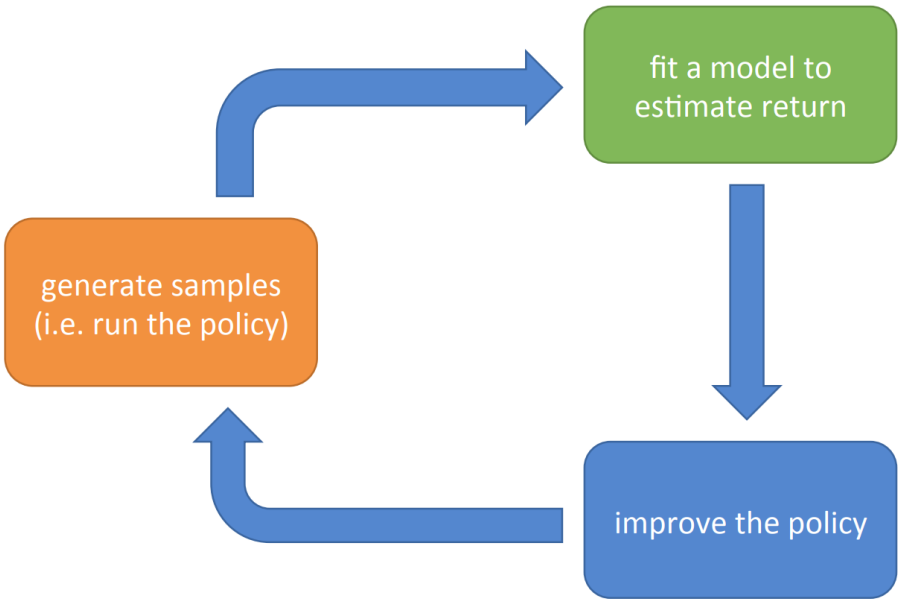
Model-Based RL

- Key Idea:
  - It would be useful if we could approximately simulate the world! i.e. if we could predict the consequences of our actions
- Advantages:
  - Can efficiently learn model by supervised learning method
  - Can reason about model uncertainty

Model-Based RL

- Key Idea:
  - It would be useful if we could approximately simulate the world! i.e. if we could predict the consequences of our actions
- Advantages:
  - Can efficiently learn model by supervised learning method
  - Can reason about model uncertainty
- Disadvantages:
  - First learn a model, then construct a value function. This means we will have two sources of approximation error

Model-Based RL



estimate $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ (model-based)

supervised learning

$$\min_{\phi} \sum_{i} ||f_{\phi}(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{s}'_i||^2$$

optimize $\pi_{\theta}(\mathbf{a}|\mathbf{s})$ (model-based)

Model-Based RL

- So, now we have the following algorithm
  1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
  2. Learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||$
  3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions

Model-Based RL

- So, now we have the following algorithm
  1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
  2. Learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||$
  3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions
- What can go wrong here?

How can this approach fail?

How can this approach fail?

- In the first step, when we sample using random policy we may get get trajectories like the red line.

How can this approach fail?

- In the first step, when we sample using random policy we may get get trajectories like the red line.
- The agent may learn that going right means that we can go higher!

How can this approach fail?

- In the first step, when we sample using random policy we may get get trajectories like the red line.
- The agent may learn that going right means that we can go higher!
- Thus, it will learn an incomplete policy and fall at the top.

How can this approach fail?

- In the first step, when we sample using random policy we may get get trajectories like the red line.
- The agent may learn that going right means that we can go higher!
- Thus, it will learn an incomplete policy and fall at the top.
- Data distribution mismatch $p_{\pi_0}(s) \neq p_{\pi_f}(s)$

Model-Based RL

- How might we alleviate this issue?

Model-Based RL

- How might we alleviate this issue?
- Let's modify the algorithm a bit

1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||$
3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions

Model-Based RL

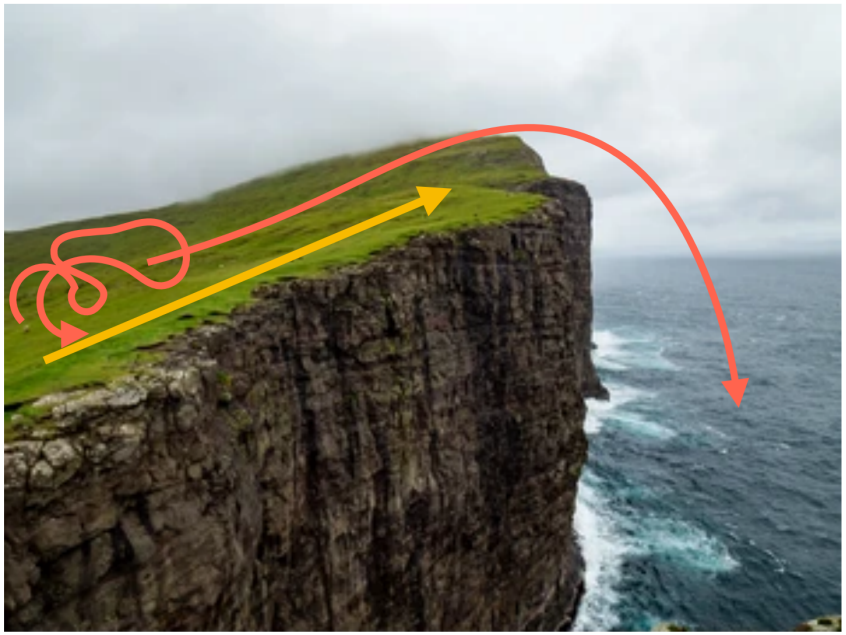
- How might we alleviate this issue?
- Let's modify the algorithm a bit

1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
2. Learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||$
3. Iteratively sample action sequences, run through model$f_\phi(s, a)$ to choose actions
4. Execute planned actions, appending visiting tuples (s, a, s') to $\mathcal{D}$
5. Go to step 2

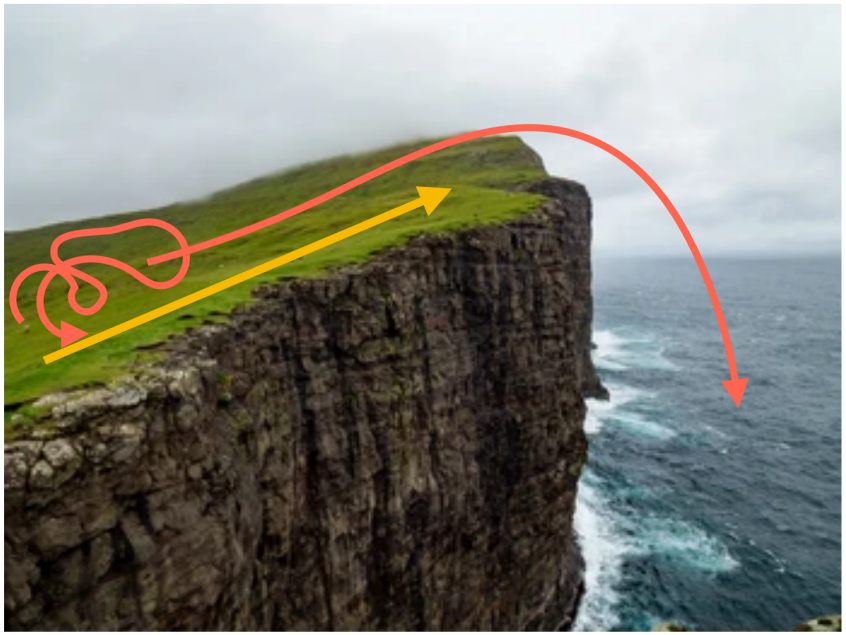
Model-Based RL

- How might we alleviate this issue?
- Let's modify the algorithm a bit
  1. Run some policy (e.g. random policy) to collect data $\mathcal{D} = \{(s, a, s')_i\}$
  2. Learn model $f_\phi(s, a)$ to minimize $\sum_i ||f_\phi(s_i, a_i) - s'_i||$
  3. Iteratively sample action sequences, run through model $f_\phi(s, a)$ to choose actions
  4. Execute planned actions, appending visiting tuples (s, a, s') to $\mathcal{D}$
  5. Go to step 2
- Now, we update the model using data collected with planning
- Then, replan periodically to help account for mistakes.

Revisiting the Cliff

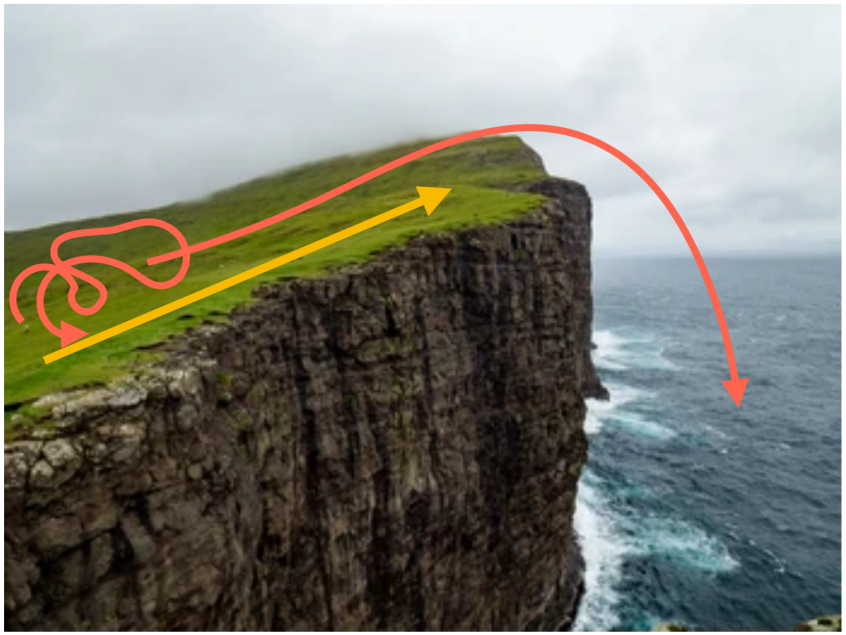


Revisiting the Cliff



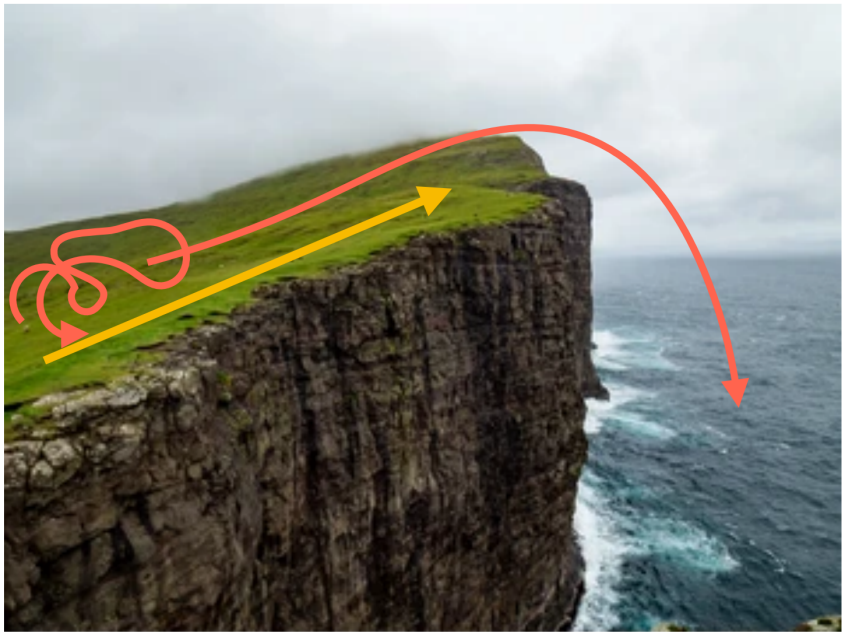
- Initially, its still the same. The agent learns that going right means that we can go higher!

Revisiting the Cliff



- Initially, its still the same. The agent learns that going right means that we can go higher!
- Then we re-train the model using the new observations

Revisiting the Cliff



- Initially, its still the same. The agent learns that going right means that we can go higher!
- Then we re-train the model using the new observations
- The final policy is different now. It learns to go to the top and stop.

Model-Based Policy Optimization

- We can also use model-based RL for policy optimization

Model-Based Policy Optimization

- We can also use model-based RL for policy optimization
- Key Idea:
  - Augment data with model-simulated roll-outs

Model-Based Policy Optimization

- We can also use model-based RL for policy optimization
- Key Idea:
  - Augment data with model-simulated roll-outs
- Algorithm:
  1. Collect data using current policy π_θ, add to D_{env}
  2. Update model $p_\phi(s'|s,a)$ using D_{env}
  3. Collect synthetic roll-outs using π_θ in model p_ϕ from states in D_{env}; add to D_{model}
  4. Update policy π_θ (and critic $\mathcal{Q}$) using D_{env}
  5. Go to step 1

Model-Based Policy Optimization

- We can also use model-based RL for policy optimization
- Key Idea:
  - Augment data with model-simulated roll-outs
- Algorithm:
  1. Collect data using current policy π_θ, add to D_{env}
  2. Update model $p_\phi(s'|s,a)$ using D_{env}
  3. Collect synthetic roll-outs using π_θ in model p_ϕ from states in D_{env}; add to D_{model}
  4. Update policy π_θ (and critic $\mathcal{Q}$) using D_{env}
  5. Go to step 1
- Compatible with variety of model-free RL methods
- Could additionally use D_{env} in policy update (step 4)

Model-based Policy Optimization

How to augment?

- Generate full trajectories from initial states?

Model-based Policy Optimization

How to augment?

- Generate full trajectories from initial states?
  - Model may not be accurate for long horizons

Model-based Policy Optimization

How to augment?

- Generate full trajectories from initial states?
  - Model may not be accurate for long horizons
- Generate partial trajectories from initial states?

Model-based Policy Optimization

How to augment?

- Generate full trajectories from initial states?
  - Model may not be accurate for long horizons
- Generate partial trajectories from initial states?
  - May not get good coverage of later states

Model-based Policy Optimization

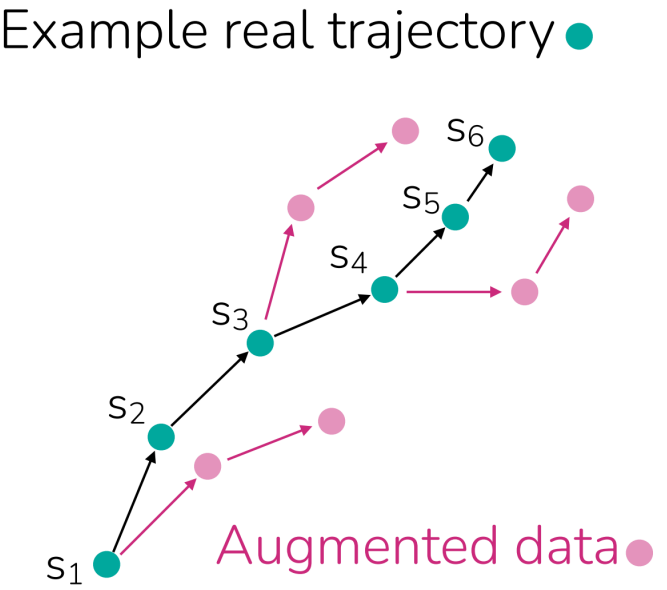
How to augment?

- Generate full trajectories from initial states?
  - Model may not be accurate for long horizons
- Generate partial trajectories from initial states?
  - May not get good coverage of later states
- Generate partial trajectories from all states in the data

Model-based Policy Optimization

How to augment?

- Generate full trajectories from initial states?
  - Model may not be accurate for long horizons
- Generate partial trajectories from initial states?
  - May not get good coverage of later states
- Generate partial trajectories from all states in the data



When to use model-based RL?

- Advantages:
  - Models are immensely useful if easy to learn
  - Model can be trained without reward labels (self-supervised)
  - Model is somewhat task-agnostic (can sometimes be transferred across rewards)

When to use model-based RL?

- Advantages:
  - Models are immensely useful if easy to learn
  - Model can be trained without reward labels (self-supervised)
  - Model is somewhat task-agnostic (can sometimes be transferred across rewards)
- Disadvantages:
  - Models don't optimize for task performance
  - Sometimes harder to learn than a policy

When to use model-based RL?

- Advantages:
  - Models are immensely useful if easy to learn
  - Model can be trained without reward labels (self-supervised)
  - Model is somewhat task-agnostic (can sometimes be transferred across rewards)
- Disadvantages:
  - Models don't optimize for task performance
  - Sometimes harder to learn than a policy

Whether to use a model depends on how hard it is to learn!

Case study: Model-based RL for dexterous manipulation

أكاديمية كاوست
KAUST ACADEMY

Model-free methods:

SAC: actor-critic method

NPG: policy gradient method

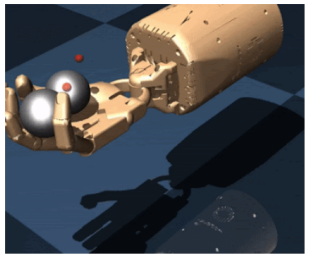
Model-based methods:

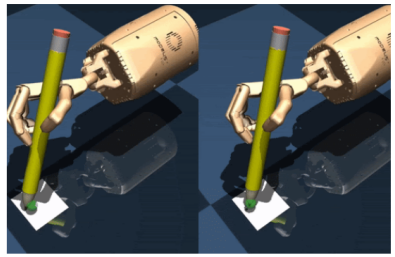
PDDM: proposed method

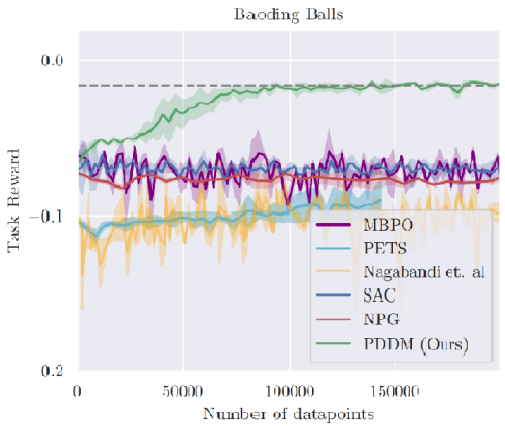
MBPO: RL with model-generated data

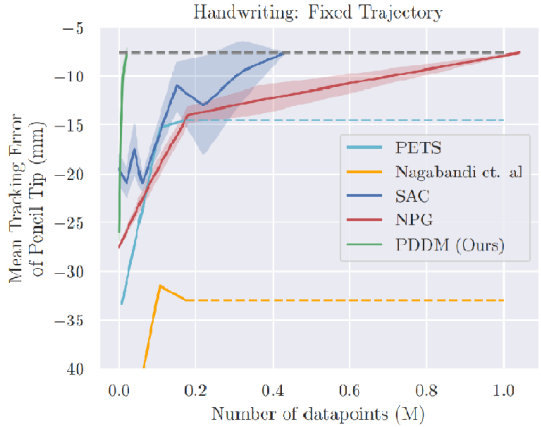
PETS: CEM-based planner

Nagabandi et al.: random shooting, no ensembles









[0]Nagabandi, et al. Deep Dynamics Models for Learning Dexterous Manipulation

Summary

- Reinforcement Learning can be categorised into Model-Based and Model-Free.
- In Model-Based RL, we also learn the model of the world i.e, the state transition model and/or the reward model
- Model-Based RL can be employed with both value and/or policy based methods
- We can use the model to augment the real environment data
- Models can be immensely useful if easy to learn but sometimes they can be harder to learn than a policy
- Whether to use a model depends on how hard it is to learn

References

These slides have been adapted from

- Chelsea Finn & Karol Hausman, Stanford CS224R: Deep Reinforcement Learning
- Sergey Levine, Berkeley CS285 Deep Reinforcement Learning
- David Silver, Deepmind: Introduction to Reinforcement Learning