Sequential Decision Making and Reinforcement Learning

(SDMRL)

Model-Based RL

Sarah Keren

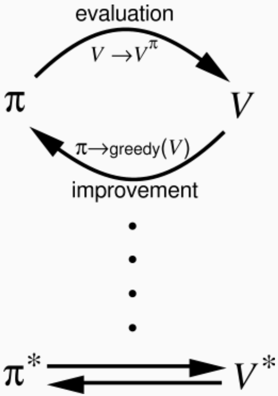
The Taub Faculty of Computer Science
Technion - Israel Institute of Technology

Acknowledgments

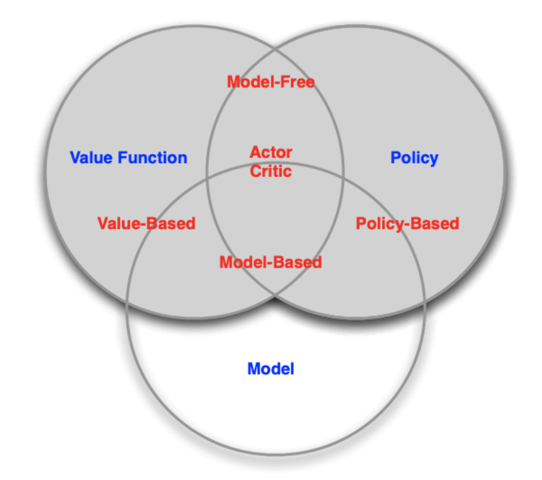
- David Sliver's course on RL: *https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silve*
- Slides by Malte Helmert, Carmel Domshlak, Erez Karpas and Alexander Shleyfman.

Recap

Anatomy of RL Algorithms



RL Approaches



Model-Free vs. Model-Based RL

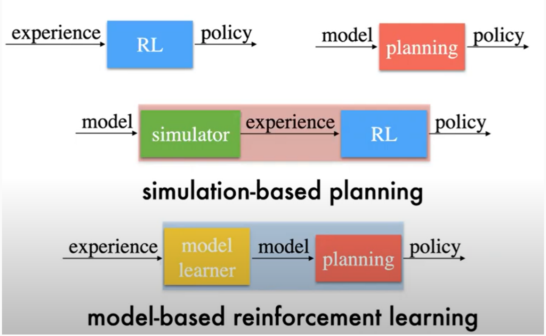


Figure 1: By Michael Littman

https://www.youtube.com/watch?v=45FKxa3qPHo

Model-Free vs. Model-Based RL

There is no clear distinction. Can be viewed as a spectrum.

From Model-Free to Model-Based RL

- We have so far explored two **model-free value-based** approaches:
  - Monte-Carlo methods:

$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(G_t - V(S_t))$$

  - Temporal Difference methods:

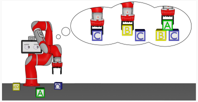
$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha(\mathcal{R}(S_t) + \gamma V_\pi(S_{t+1}) - V_\pi(S))$$

- We have also explored policy based methods.
- In model-based RL the agent uses a transition and reward model to make decisions about how to act.
  - The model may be initially known (e.g., chess) or unknown (e.g., robot manipulator).

Model-Based RL Intro

Model-Based RL

- Reinforcement learning systems can make decisions in one of two ways.
  - In the model-based approach, a system uses a predictive model of the world to ask questions of the form **“what will happen if I do x?”** to choose the best action.
  - In the alternative model-free approach, the modeling step is bypassed altogether in favor of **learning a control policy directly**. **“how much reward will I get if I do x?”**
- Although in practice the line between these two techniques can become blurred, as a coarse guide it is useful for dividing up the space of algorithmic possibilities.



BAIR blog by Michael Janner
https://bair.berkeley.edu/blog/2019/12/12/mbpo/

Advantages and Disadvantages of Model-Based RL

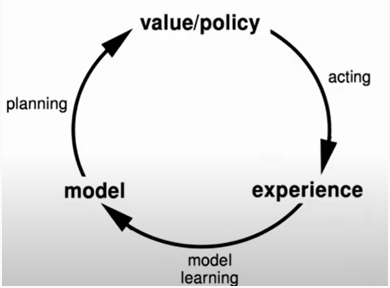
Advantages and Disadvantages of Model-Based RL

Advantages:

- Can efficiently learn model by supervised learning methods
- Can reason about model uncertainty
- Explainability

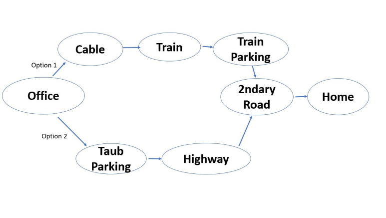
Disadvantages:

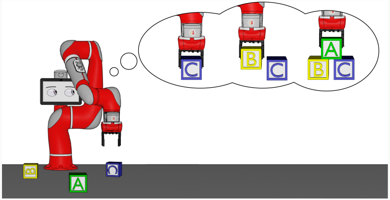
- First learn a model, then construct a value function $\Rightarrow$ two sources of approximation error



By Emma Brunskill
https://www.youtube.com/watch?v=vDF1BYWhqL8

Model-Based RL for the Going Home Example ?





Ideas for model-based evaluation ?

Ideas for model-based control ?

Learning a Model

What is a Model?

- A model $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ parameterized by η
- Typically, $\mathcal{S}$ and $\mathcal{A}$ are assumed to be known
- So a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}\eta \approx \mathcal{R}$

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1} | S_t, A_t)$$

$$R_{t+1} \sim \mathcal{R}_\eta(R_{t+1} | S_t, A_t)$$

- Typically assume conditional independence between state transitions and rewards

$$\mathbb{P}[S_{t+1}, R_{t+1} | S_t, A_t] = \mathbb{P}[S_{t+1} | S_t, A_t] \mathbb{P}[R_{t+1} | S_t, A_t]$$

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience collected by interacting with the environment $\{S_1, A_1, R_2, \dots, S_T\}$
- This is a supervised learning problem

$$S_1, A_1 \to R_2, S_2$$

$$S_2, A_2 \to R_3, S_3$$

$$\vdots$$

$$S_{T-1}, A_{T-1} \to R_T, S_T$$

- Learning $s, a \to r$ is a regression problem
- Learning $s, a \to s'$ is a density estimation problem
- Pick loss function, e.g. mean-squared error, KL divergence, ...
- Find parameters η that minimise empirical loss

Examples of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- ...

Table Lookup Model

- Model is an explicit MDP with estimated transition function $\hat{\mathcal{P}}$ and estimated reward function $\hat{\mathcal{R}}$
- Count visits $N(s, a)$ to each state action pair

$$\hat{\mathcal{P}}^a_{s,s'} = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, S_{t+1} = s, a, s')$$

$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t . A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

Model-Based Evaluation

Adaptive Dynamic Programming (ADP)

- Follow the policy for a while
- Estimate transition model based on observations
- Learn reward function
- Use estimated model to compute utility of policy

$$V_\pi(s) = \mathcal{R}(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s)) \cdot V_\pi(s')$$

How can we estimate transition model $\mathcal{P}$ and $\mathcal{R}$?

Adaptive Dynamic Programming (ADP)

- Follow the policy for a while
- Estimate transition model based on observations
- Learn reward function
- Use estimated model to compute utility of policy

$$V_\pi(s) = \mathcal{R}(s) + \gamma \sum_{s'} \mathcal{P}(s'|s, \pi(s)) \cdot V_\pi(s')$$

How can we estimate transition model $\mathcal{P}$ and $\mathcal{R}$?

Compute the fraction of times we see s' after taking a in state s (similarly for the reward function).

Chernoff bounds can be used to quantify the confidence in these estimates by providing high-probability bounds on the deviation of the estimated probabilities from the true probabilities.

Model-Based Control

Sample-Based Planning

- A simple but powerful approach to planning
- Use the model only to generate samples
- **Sample** experience from model

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t)$$

$$R_{t+1} \sim \mathcal{R}_\eta(R_{t+1}|S_t, A_t)$$

- Apply model-free RL to samples, e.g.: Monte-Carlo control, Sarsa, Q-learning
- Sample-based planning methods are often more efficient

Sample-Based Planning: Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sampled experience

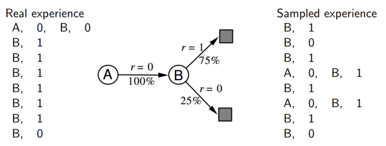


Figure 2: e.g., Monte-Carlo learning: $\mathcal{V}(A) = 1, \mathcal{V}(B) = 0.75$

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A} \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ using favourite planning algorithm
  - Value iteration
  - Policy iteration
  - Tree search
  - Heuristic search

Naïve Approach To Model-Based Control

Model-Based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Solve resulting MDP using value iteration, policy iteration, LAO*, MCTS etc.
4. Follow resulting policy thereafter.

Will this work ?

Naïve Approach To Model-Based Control

Model-Based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Solve resulting MDP using value iteration, policy iteration, LAO*, MCTS etc.
4. Follow resulting policy thereafter.

Will this work ?

Yes, if we do step 1 long enough and there are no “dead-ends”.

Any problems?

Naïve Approach To Model-Based Control

Model-Based approach to RL

1. Act randomly for a (long) time
2. Learn transition function and reward function
3. Solve resulting MDP using value iteration, policy iteration, LAO*, MCTS etc.
4. Follow resulting policy thereafter.

Will this work ?

Yes, if we do step 1 long enough and there are no "dead-ends".

Any problems?

We will act randomly for a long time before exploiting what we know

Revision of Naïve Approach

Model-Based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

Model-Based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work ?

Revision of Naïve Approach

Model-Based approach to RL

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work ? No. Can get stuck in local optimum

What can be done ?

Reminder: Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation:** We exploit our current knowledge to get a payoff.
  - **Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL.
- Basic **intuition** behind most approaches:

Reminder: Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation:** We exploit our current knowledge to get a payoff.
  - **Exploration:** Get more information about the world. How do we know if there is not a pot of gold around the corner?
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL.
- Basic **intuition** behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as agent gains knowledge

ADP-based RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work?

ADP-based RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy

Any ideas?

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

$$Q(s,a) = \mathcal{R}(s) + \gamma \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

  - where V is current optimal value function estimate (based on current model), and $\mathcal{R}$ and $\mathcal{P}$ are current estimates of the model
  - $Q(s,a)$ is the expected value of taking action a in state s and then getting the estimated value $\mathcal{V}(s')$ of the next state s'.
- We want an exploration policy that is GLIE (greedy in the limit of infinite exploration).

Explore/Exploit Policies

- GLIE policy 1:
  - On time step t select random action with probability $p(t)$ (i.e., ϵ) and greedy action with probability $1 - p(t)$
  - $p(t) = \frac{1}{t}$ (will lead to convergence, but is slow.)
  - Greedy action is the one that maximizes the Q value.
- GLIE policy 2: Boltzmann Exploration [1]
  - Select action a with probability $\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$
  - T is the **"temperature"**: Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically: start with large T and decrease with time.

[1] according to wikipedia - a Boltzmann distribution (also called Gibbs distribution) is a probability distribution or probability measure that gives the probability that a system will be in a certain state as a function of that state's energy and the temperature of the system.

$$\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$$

Suppose we have just two actions and that $Q(s, a_1) = 1$, and $Q(s, a_2) = 2$. **Which action will have a higher probability?**

1. $T = 10$:

$$\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$$

Suppose we have just two actions and that $Q(s, a_1) = 1$, and $Q(s, a_2) = 2$. **Which action will have a higher probability?**

1. $T = 10$: $\mathcal{P}(a_1|s) = 0.48$, $\mathcal{P}(a_1|s) = 0.52$
   Almost equal probability, and so explore
2. $T = 1$:

Impact of Temperature

$$\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$$

Suppose we have just two actions and that $Q(s, a_1) = 1$, and $Q(s, a_2) = 2$. **Which action will have a higher probability?**

1. $T = 10$: $\mathcal{P}(a_1|s) = 0.48$, $\mathcal{P}(a_1|s) = 0.52$
   Almost equal probability, and so explore
2. $T = 1$: $\mathcal{P}(a_1|s) = 0.27$, $\mathcal{P}(a_1|s) = 0.73$
   Probabilities more skewed, so explore a_1 less
3. $T = .25$:

Impact of Temperature

$$\mathcal{P}(a|s) = \frac{\exp(Q(s,a)/T)}{\sum_{a' \in \mathcal{A}} \exp(Q(s,a')/T)}$$

Suppose we have just two actions and that $Q(s, a_1) = 1$, and $Q(s, a_2) = 2$. **Which action will have a higher probability?**

1. $T = 10$: $\mathcal{P}(a_1|s) = 0.48$, $\mathcal{P}(a_1|s) = 0.52$
   Almost equal probability, and so explore
2. $T = 1$: $\mathcal{P}(a_1|s) = 0.27$, $\mathcal{P}(a_1|s) = 0.73$
   Probabilities more skewed, so explore a_1 less
3. $T = .25$: $\mathcal{P}(a_1|s) = 0.02$, $\mathcal{P}(a_1|s) = 0.98$
   Almost always exploit a_2

Alternative Approach: Optimistic Exploration

Model-Based approach to RL

1. Start with initial model
2. Solve for optimistic policy given current model using optimistic value iteration - that inflates value of actions leading to unexplored regions.
3. Execute greedy action suggested by the optimistic policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

Basically act as if all “unexplored” state-action pairs are maximally rewarding

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := \mathcal{R}(s) + \gamma \max_a \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

  - Optimistic variant -

What do we mean by "explored enough"?

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) := \mathcal{R}(s) + \gamma \max_a \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s')$$

    - Optimistic variant - adjusts update to make actions that lead to unexplored regions look promising
- Implement variant of VI that assigns the highest possible value V^{max} to any state-action pair that has not been explored enough
    - Maximum value is when we get maximum reward forever

$$V^{max} = \sum_{t=0}^{\infty} \gamma^t \cdot R^{max} = \frac{R^{max}}{1-\gamma}$$

What do we mean by “explored enough”?

Optimistic Exploration

- What do we mean by "explored enough"?
  - $N(s,a) > N_e$, where $N(s,a)$ is number of times action a has been tried in state s and N_e is a user selected parameter.
  - While a standard initialization is:

$$V^{max} := \sum_{t=0}^{\infty} \gamma^t \cdot R^{max} = \frac{R^{max}}{1-\gamma}$$

  - Optimistic value iteration computes an optimistic value function V^+ using updates:

$$V^+ := \mathcal{R}(s) + \gamma \max_a \begin{cases} V^{max}, & N(s,a) < N_e \\ \sum_{s'} \mathcal{P}(s'|s,a) \cdot V_\pi(s'), & \text{otherwise} \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around (-> optimistic)
- But after actions are tried enough times, we will perform standard "non-optimistic" value updates

For more details: R-max – A General Polynomial Time Algorithm for Near-Optimal

Planning with an Inaccurate Model

- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A} \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. Just like model-free RL is only as good as the data that is collected, model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
- Ideas?

Planning with an Inaccurate Model

- Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- Performance of model-based RL is limited to optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}\mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- i.e. Just like model-free RL is only as good as the data that is collected, model-based RL is only as good as the estimated model
- When the model is inaccurate, planning process will compute a suboptimal policy
- Ideas?
    - Solution 1: when the model is wrong, use model-free RL
    - Solution 2: reason explicitly about model uncertainty

Integrated Architectures

Real and Simulated Experience

We consider two sources of experience:

- Real experience sampled from environment (true MDP)

$$S' \sim \mathcal{P}^a_{s,s'}$$

$$R' \sim \mathcal{R}^a_s$$

- Simulated experience sampled from model (approximate MDP)

$$S' \sim \mathcal{P}_\eta(S'|S,A)$$

$$R \sim \mathcal{R}_\eta(R|S,A)$$

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

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL
  - Learn a model from real experience
    - Using Sample-Based Planning: Plan value function (and/or policy) from simulated experience
    - Plan using estimated model

Integrating Learning and Planning

- Model-Free RL
  - No model
  - **Learn** value function (and/or policy) from real experience
- Model-Based RL
  - Learn a model from real experience
    - Using Sample-Based Planning: Plan value function (and/or policy) from simulated experience
    - **Plan** using estimated model
- Dyna
  - Learn a model from real experience
  - **Learn and plan** value function (and/or policy) from real and simulated experience

Dyna Architecture

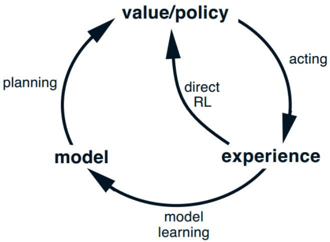


Figure 8.1: Relationships among learning, planning, and acting.

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Do forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \varepsilon$-greedy(S,Q)
  (c) Execute action A; observe resultant reward, R, and state, S'
  (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
  (e) $Model(S,A) \leftarrow R, S'$ (assuming deterministic environment)
  (f) Repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

SOTA(ish) Examples

Approaches to Models-Based RL

- Analytic gradient computation:
  - Based on assumptions about the form of the dynamics and cost function (.e.g., Gaussian processes)
  - Can yield locally optimal control (e.g.Linear–quadratic regulator).
  - Even when these assumptions are not valid, can account for small errors introduced by approximated dynamics.
  - Linear (simplified) models, can also be used to provide guiding samples for training more complex nonlinear policies.

- Sampling-based planning:
  - Typically, for nonlinear dynamics models we resort to sampling action sequences (e.g., via random shooting). More sophisticated variants iteratively adjust the sampling distribution.
  - In discrete-action settings, however we can search over tree structures than to iteratively refine a single trajectory of waypoints.
    - Monte-Carlo Tree Search (MCTS)- has underpinned recent impressive results in games playing, and iterated width search.
  - In both continuous and discrete domains, can be combined with structured physics-based, object-centric priors.

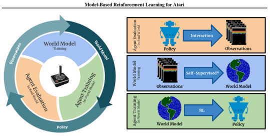
Approaches to Models-Based RL

- Model-Based data generation
  - Many machine learning success stories rely on artificially increasing the size of a training set.
  - It is difficult to define a manual data augmentation procedure for policy optimization, but we can view a predictive model analogously as a learned method of generating synthetic data.
  - The original proposal of such a combination comes from the Dyna algorithm by Sutton, which alternates between model learning, data generation under a model, and policy learning using the model data.

Model-Based RL: State-of-the-Art

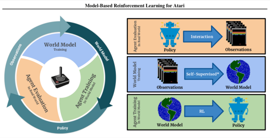
"In this paper, we explore how video prediction models can similarly enable agents to solve Atari games with orders of magnitude fewer interactions than model-free methods. We describe Simulated Policy Learning (SimPLe), a complete model-based deep RL algorithm based on video prediction models and present a comparison of several model architectures, including a novel architecture that yields the best results in our setting." (arXiv)

Paper by Kaiser et al 2020 *https://arxiv.org/abs/1903.00374*
https://medium.com/syncedreview/google-brain-simple-complete-model-based-reinforcement-learning-for-atari-b550a960



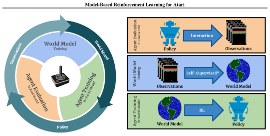
Model-Based RL: Example

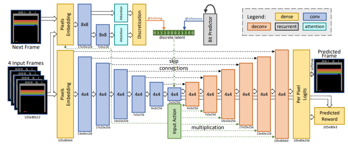
- SimPLe is a complete model-based deep RL algorithm that utilizes video prediction techniques and can train a policy to play a game within the learned model.
- SimPLe outperforms model-free algorithms in terms of learning speed on nearly all of the games, and in the case of a few games, does so by over an order of magnitude.
- The best model-free reinforcement learning algorithms require tens or hundreds of millions of time steps — equivalent to several weeks. SimPLe has obtained competitive results with only 100K interactions between the agent and the environment on Atari games, which corresponds to about two hours of real-time play.



Model-Based RL: Example

- Agent starts interacting with the real environment following the latest policy (initialized to random).
- Collected observations used to train (update) current world model.
- Agent updates the policy by acting inside the world model. The new policy will be evaluated to measure performance and collect more data (back to 1).





- Trust Region Policy Optimization (TRPO) is a policy optimization algorithm that optimizes the policy within a ”trust region” to prevent large updates.
- To ensure stability, TRPO adds a constraint on policy updates:

$$\text{KL-Divergence: } D_{KL}(\pi_{\text{new}} || \pi_{\text{old}}) \leq \delta$$

ModelTRPO Algorithm

1. Collect trajectories by interacting with the environment using the current policy.
2. Estimate advantage function $A^{\pi_{\text{old}}}(s, a)$.
3. Compute the surrogate objective $L(\pi)$.
4. Perform constrained optimization to find π_{new}.
5. Update policy: $\pi \leftarrow \pi_{\text{new}}$.

ME-TRPO Algorithm

- Model-Ensemble TRPO (ME-TRPO) incorporates an ensemble of learned dynamics models to generate synthetic data for improving sample efficiency, while retaining TRPO's policy optimization stability through trust region constraints.
- The ensemble mitigates bias by averaging predictions or accounting for uncertainty among the models.

Model-Based RL - Links

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

Benchmarking Model-Based Reinforcement Learning by Wang et al. 2019 *https://arxiv.org/abs/1907.02057* When to Trust Your Model: Model-Based Policy Optimization Janner et al. 2021 *https://arxiv.org/abs/1906.08253*

https://www.natolambert.com/writing/debugging-mbrl

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

- Pieter Abbeel on model-based RL: *https://youtu.be/2o1yrkbpcUk?feature=shared*

Conclusion

Model-Free vs. Model-Based RL

- Adaptive Dynamic Programming (model based)

- Monte-Carlo Direct Estimation (model free)

- Temporal Difference Learning (model free)

Model-Free vs. Model-Based RL

- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints—adjusts state to “agree” with observed successor (not all possible successors)
  - Convergence in between direct estimation and ADP

What Next ?

Large State Spaces

- When a problem has a large state space we can not longer represent the V or Q functions as explicit tables
- Even if we had enough memory
  - Never enough training data!
  - Learning takes too long

What to do?