UCLA DRL: https://www.youtube.com/playlist?list=PLkFD6_40KJIxJMR-j5A1mkxK26gh_qg37
https://www.youtube.com/playlist?list=PL_iWQOsE6TfX7MaC6C3HcdOf1g337dlC9
DRL course website: https://rail.eecs.berkeley.edu/deeprlcourse/
The learning order would be the following: realizing each day you have two hours… one hour for NLP and another hour for DRL would be the best possible choice you could have. Realizing to improve your efficiency of learning, you should consider the following measures: x2.0 speed is only useful when you are trying to skip some known contents.

UCLA DRL:

https://gymnasium.farama.org/ : interactive environment package, that works (at least) on windows system with relatively easier setups.
https://www.youtube.com/playlist?list=PL_iWQOsE6TfX7MaC6C3HcdOf1g337dlC9

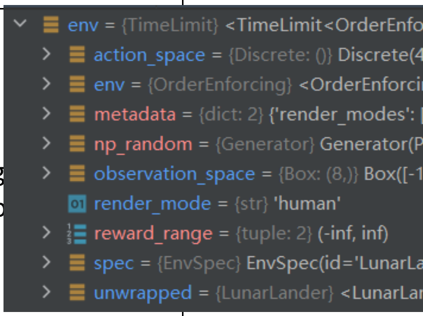
gynasium package:

# cannot use any environment of “mujoco_py”, due to possibly leading to error.

```
import gymnasium as gym
env = gym.make("LunarLander-v2", render_mode="human")
          # image shown right gives the class properties of created environment.
gymnasium.envs.registry.keys().make # this gives all possible set of environments
#
https://github.com/openai/gym/blob/e689f93a425d97489e590bba0a7d4518de0dcc03/gym/envs/__init__.py#L53-L58 provides a list of available environments, for gym.make()
```

```
# most commonly used methods:
observation, info = env.reset(seed) # gives initial states
          # this must be performed before any environment exploration!
action = env.action_space.sample()
          # better use python console to check the type of “action” before moving
          # action_space of “env” could also be strings? Or using indexing for actio
observation, reward, terminated, truncated, info = env.step(action)
          # this displays returned variables

if terminated or truncated:
          observation, info = env.reset()
```



```
env = {TimeLimit} <TimeLimit<OrderEnfo
  action_space = {Discrete: ()} Discrete(4
  env = {OrderEnforcing} <OrderEnforci
  metadata = {dict: 2} {'render_modes': [
  np_random = {Generator} Generator(P
  observation_space = {Box: (8,)} Box([-1
  render_mode = {str} 'human'
  reward_range = {tuple: 2} (-inf, inf)
  spec = {EnvSpec} EnvSpec(id='LunarLa
  unwrapped = {LunarLander} <LunarLar
```

```
# configurations for setting up
num_actions = env.action_space.n
observation_dim = env.observation_space.shape[0]
          # usually this is a tuple, representing a 1D vector.
# These two parameters are critical for defining the input and output shape of model.
```

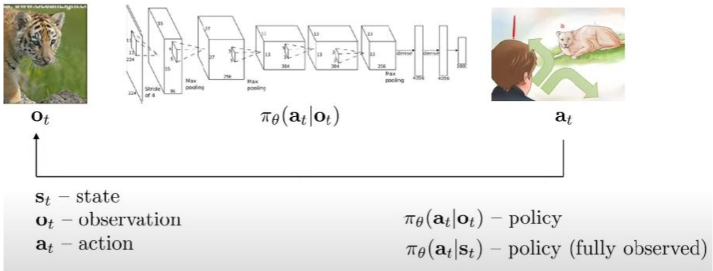
Lec1:

Challenges of DRL: (41015)

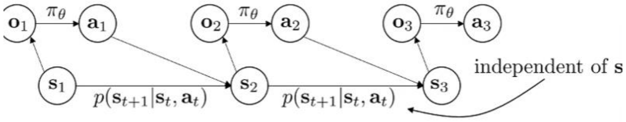
1. Slow learning
2. Cannot use part knowledge, transfer learning
3. Not clear of what reward function should be defined
4. Not clear the role of prediction(model based?) should be

Lec2: supervised learning for policies -> imitation learning

(10120)terminologies:
Observation and action: o,a; policy: \pi; weight: theta; time: t; state: s;
“a_t” has effect on “o_{t+1}”

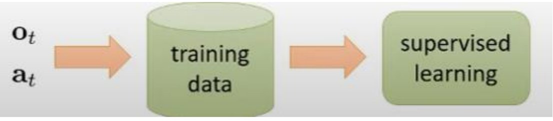

$\mathbf{o}_t$
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$
$\mathbf{a}_t$
$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action
$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

Graphical models(10548)
Dependencies analysis from graphical models



Future is not conditioned on past, given present; single markov chain; can forget about past states, only focus on current would be enough(markov property)
But in general, markov property is not satisfied; (past observation gives additional information)

Imitation learning(11027)



basically, given observation-action pairs from HUMAN demonstrations, use supervised learning that take observation as input, and then action as output -> policy is learned in this manner.
BUT: NOT WORK!!! In general;
When NN makes mistakes in this case, the error will accumulate and finally significantly deviate from true model’s behavior
However when applied with mistake-correcting techniques with supervised learning (Nvidia example), the problem would be resolved(11654)
Policies need to learn not only what’s correct, but also what’s wrong and learn to adjust -> feedbacks;

Dataset aggregation/Dagger(12012)
A training process:

goal: collect training data from $p_{\pi_\theta}(\mathbf{o}_t)$ instead of $p_{\mathrm{data}}(\mathbf{o}_t)$

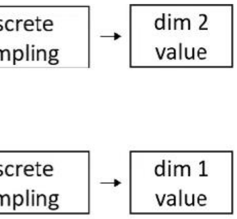
1. Use human demo to pretrain policy;

2. Run policy and gather another set of observation
3. Human label those observations with actions, and train again
4. Aggregate dataset and start from step 1

Non0Markovian behavior
Step 3 can be difficult: real scenes can still be quite various, and labeling using only one action might not match with real scenarios
(20250): markov property: if you do this at this state, you always do this at this state -> no changing, compared with human demonstrators…
Feeding historical data: RNN, stack information(20420)

Multimodal behavior(20800)
Solutions:

1. Using linear combination on mixture of gaussians to represent weighted action mixtures
   Pay attention to dimensional issues -> space
2. Latent variable: for representing noise of behaviour; recall reparameterization and variational autoencoder from csc412
3. Autoregressive discretization: discretize continuous action space dimension after dimension(recall hash map after hash map) -> leads to only “n” many dimension exploration instead of “n^2”; but applied too much pressure on accuracy of prior dimensions



(30000): case study, adopting Nvidia method: 3 cameras for error avoidance feedbacks

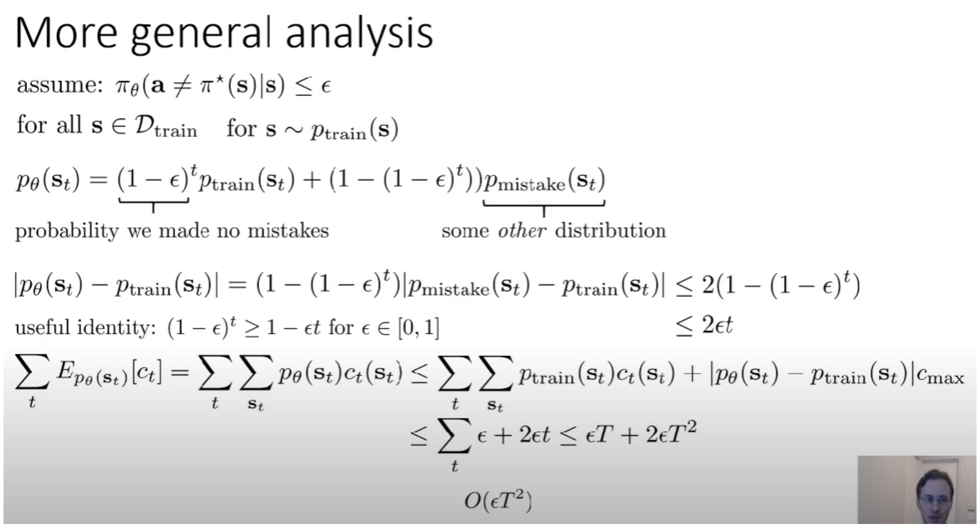
(40100)
Problems of imitation learning using supervised techniques:

- Data is finite -> not sufficient for DL
- Humans provides varied actions, which might not be globally optimal
- Machines can learn autonomously?

Policy optimization(40230)
Minimize cost, or maximize reward(r)
r=-c

$$\min_{\theta} E_{\mathbf{s}_{1:T},\mathbf{a}_{1:T}}\left[\sum_t c(\mathbf{s}_t,\mathbf{a}_t)\right]$$

imitation learning theory (50200)
bound on the final learning cost analysis



More general analysis

assume: $\pi_\theta(\mathbf{a} \neq \pi^\star(\mathbf{s})|\mathbf{s}) \leq \epsilon$

for all $\mathbf{s} \in \mathcal{D}_{\text{train}}$ for $\mathbf{s} \sim p_{\text{train}}(\mathbf{s})$

$$p_\theta(\mathbf{s}_t) = \underbrace{(1-\epsilon)^t}_{\text{probability we made no mistakes}} p_{\text{train}}(\mathbf{s}_t) + (1-(1-\epsilon)^t))\underbrace{p_{\text{mistake}}(\mathbf{s}_t)}_{\text{some \textit{other} distribution}}$$

$$|p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| = (1-(1-\epsilon)^t)|p_{\text{mistake}}(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| \leq 2(1-(1-\epsilon)^t)$$

useful identity: $(1-\epsilon)^t \geq 1-\epsilon t$ for $\epsilon \in [0,1]$ $\qquad \leq 2\epsilon t$

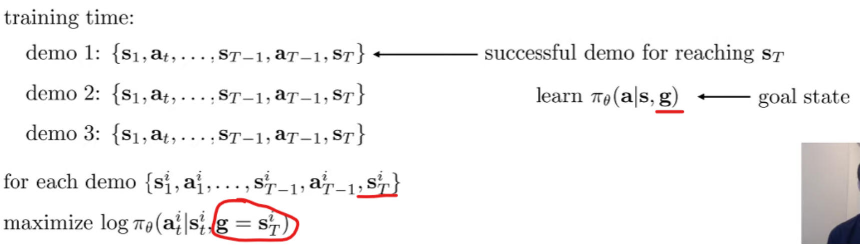
$$\sum_t E_{p_\theta(\mathbf{s}_t)}[c_t] = \sum_t \sum_{\mathbf{s}_t} p_\theta(\mathbf{s}_t) c_t(\mathbf{s}_t) \leq \sum_t \sum_{\mathbf{s}_t} p_{\text{train}}(\mathbf{s}_t) c_t(\mathbf{s}_t) + |p_\theta(\mathbf{s}_t) - p_{\text{train}}(\mathbf{s}_t)| c_{\max}$$

$$\leq \sum_t \epsilon + 2\epsilon t \leq \epsilon T + 2\epsilon T^2$$

$$O(\epsilon T^2)$$

Another imitation idea(60000)

- Facing multiple end points: policy learning will also be conditioned on the final point want to reach, no longer states only



training time:

demo 1: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$ ← successful demo for reaching $\mathbf{s}_T$

demo 2: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$ learn $\pi_\theta(\mathbf{a}|\mathbf{s}, \underline{\mathbf{g}})$ ← goal state

demo 3: $\{\mathbf{s}_1, \mathbf{a}_t, \ldots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, \mathbf{s}_T\}$

for each demo $\{\mathbf{s}_1^i, \mathbf{a}_1^i, \ldots, \mathbf{s}_{T-1}^i, \mathbf{a}_{T-1}^i, \underline{\mathbf{s}_T^i}\}$

maximize $\log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i, \boxed{\mathbf{g} = \mathbf{s}_T^i})$

Can also use random goals and random policies to sufficiently explore at early stages(60408)

Lec4: introduction to reinforcement learning

Reward (10313)

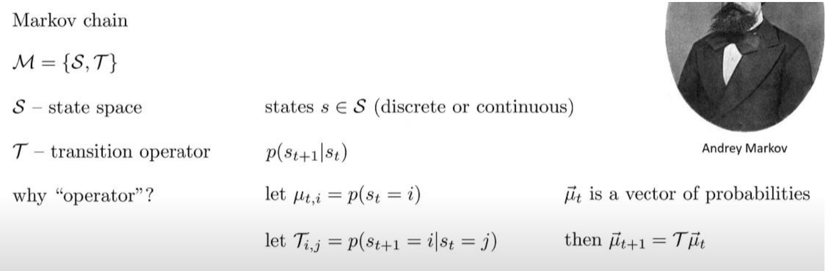
- r(s, a): which states and actions are better.
- Choosing long-term high reward

Markov decision process

Markov chain(10500)

M = {S, T}: States, Transition operator (probabilistic or deterministic state assignment)

\mu: set of probabilities: \mu(t, i)=p(s_t=i)



Markov chain

$\mathcal{M} = \{\mathcal{S}, \mathcal{T}\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{T}$ – transition operator $p(s_{t+1}|s_t)$

why "operator"? let $\mu_{t,i} = p(s_t = i)$ $\vec{\mu}_t$ is a vector of probabilities

let $\mathcal{T}_{i,j} = p(s_{t+1} = i|s_t = j)$ then $\vec{\mu}_{t+1} = \mathcal{T}\vec{\mu}_t$

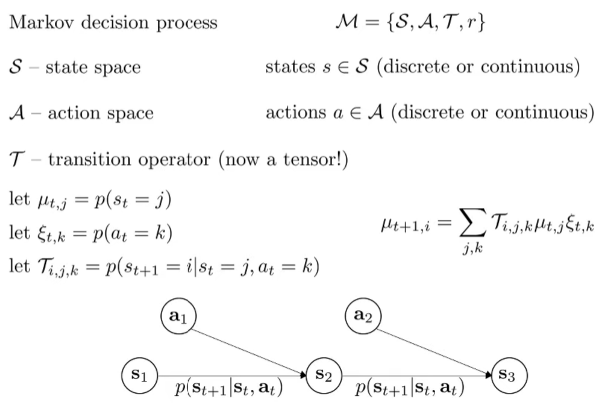
Andrey Markov

Graphical model(10720)

Markov decision process:

M={S(tates), A(ctions), T(ransitions), r(eward)}

s \in S; a \in A; transition: \mu, \Xi (probability of action at time t); reward: r(s, a) -> R



Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{T}$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

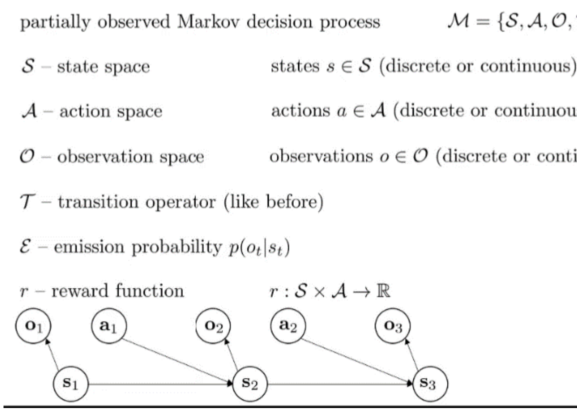
let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k}\mu_{t,j}\xi_{t,k}$$

$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$\mathbf{s}_2$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$\mathbf{s}_3$

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward



partially observed Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O},$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuou

$\mathcal{O}$ – observation space observations $o \in \mathcal{O}$ (discrete or conti

$\mathcal{T}$ – transition operator (like before)

$\mathcal{E}$ – emission probability $p(o_t|s_t)$

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

Partially observed markov decision process: (11003)
O: observation states: o \in O; \epsilon: emission probability: p(o_t | s_t)

Reinforcement learning objective(11220)

- Expectation of rewards, over all possible trajectories; use the expectation of reward to do updates of weights
- Ultimate goal: maximize the expectation of rewards

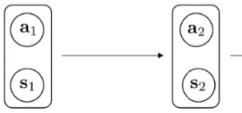
\tao: a sequence of state-action pairs(trajectory)

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

the product term is a Markov chain on (s, a); group states and actions as one state;

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$
$$= \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t)\sim p_\theta(\mathbf{s}_t, \mathbf{a}_t)}[r(\mathbf{s}_t, \mathbf{a}_t)]$$



infinite horizon: stationary distribution(12030)

- Definition of stationary distribution: \mu = T(\mu) after a certain period of time: same before and after transition; -> convergence

Expectations and stochastic systems(12343)

- Infinite STATIONARY horizon case and finite horizon case: finite as a sum of finitely many terms. Infinite stationary simply removes summation.
- (12623): non-differentiable functions under a differentiable distribution becomes differentiable -> reparameterization trick

Algorithms: (20000)
General procedure: (20030)
Generate samples -> fit a model/estimate return -> improve policy;
Stage 1 -> 2 -> 3
Under DRL: (perhaps)model would use current state and action to approximate next state(stage 2), and improving policy involves backpropagation(state 3)(20440)
Sampling(20530): usually using simulators to generate samples.

Value functions (30000)
Write out rewards recursively, using conditioning from previous states -> write out recursively

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}\left[E_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1|\mathbf{s}_1)}\left[r(\mathbf{s}_1,\mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2|\mathbf{s}_1,\mathbf{a}_1)}\left[E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2|\mathbf{s}_2)}\left[r(\mathbf{s}_2,\mathbf{a}_2) + ...|\mathbf{s}_2\right]|\mathbf{s}_1,\mathbf{a}_1\right]|\mathbf{s}_1\right]\right]$$

Q-value(30249)

$$Q(\mathbf{s}_1,\mathbf{a}_1) = r(\mathbf{s}_1,\mathbf{a}_1) + E_{\mathbf{s}_2 \sim p(\mathbf{s}_2|\mathbf{s}_1,\mathbf{a}_1)}\left[E_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2|\mathbf{s}_2)}\left[r(\mathbf{s}_2,\mathbf{a}_2) + ...|\mathbf{s}_2\right]|\mathbf{s}_1,\mathbf{a}_1\right]$$

Applying Q-function on a policy \pi gives:

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right]\text{: total reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$$

Calculating from terminal state all the way to starting state, and use properties of recursive value memorization

Value function(30530)

$$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t\right]\text{: total reward from } \mathbf{s}_t$$

$$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t,\mathbf{a}_t)]$$

$$E_{\mathbf{s}_1 \sim p(\mathbf{s}_1)}[V^\pi(\mathbf{s}_1)] \text{ is the RL objective!}$$

compared with Q, only has states. Allows taking expectations of all actions to integrate the impact of all actions.

Ideas of performing updates (30755)

If Q(s,a) > V(s) given an action “a”: this action is better than average actions -> increase likelihood of this action!

Types of RL algorithms: (40000)

Policy gradient, value-based, actor-critic, model-based RL

- Policy gradient: differentiate “expectation of rewards” w.r.t parameter theta
- Value-based: estimate optimal policy Q function
- Actor-critic: estimate Q/V of current policy and improve
- Model-based: also include a model for making predictions, useful for planning

Model-based(40210)

- Stage 2: predict next state using current state and current action; NN or search
- Stage 3: backpropagate; just use the model, no improvements; or use model to learn value function

Value-based: (40430)

- Stage 2: fit V/Q using a model (simpler model, using KL divergence to check how good estimation is)
- Stage 3: take argmax for policy, that maximizes V/Q function

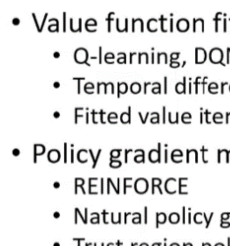
Direct policy gradients (40512)

- Stage2: evaluate total reward
- Stage 3: using gradient descent to improve parameter theta for policy

Actor-critic:

- 2: fit V, Q
- 3: gradient descent on parameter theta for policy;

Tradeoffs between algorithms: (50000)



- Value function fi
    - Q-learning, DQN
    - Temporal differe
    - Fitted value iter
- Policy gradient m
    - REINFORCE
    - Natural policy g
    - Trust region pol

- Sample efficiency(50230)
  Some algorithms need new samples for each update (on policy, w.r.t off-policy);
- Stability/ease of use(50510)
  Convergence? RL usually don't use gradient descent -> not converge; policy gradient uses gradient descent, but is not efficient; maximizing policy might not maximize reward

- Different assumptions -> stochastic? Continuous? Infinite horizon?(50700)
  Full observation; episodic learning(can repeatedly learning?); continuity/smoothness;
- Model/policy tradeoff

Examples of algorithms(60000)
Just an overview, will be introduced later.

Lec5: Policy Gradient RL algorithm

Direct policy differentiation(10936)
Realizing log is a monotonic function; thus taking the log over reward function(which is an integration or summation, when considering samples) and finding derivatives would become easier

Final optimization objective: (11156)

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

second equation takes sampling into account. (N is the number of sampled actions and results)
realizing Q\hat_{i, t} might replace the reward summation term in product

mechanism of training: (11329)
- Sample rewards by running the policy
- Calculate gradient w.r.t above two equations
- Perform gradient descent on parameter theta;

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

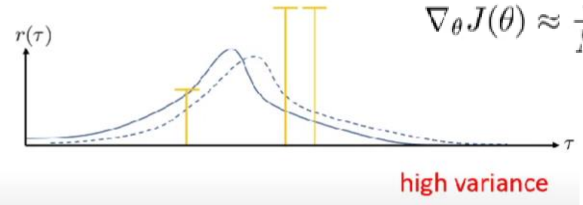
(20000)
Comparison of PG and MLE: (20200)
- Using reward (which can be negative) which is not part of MLE estimation makes the gradient adjustment fits with optimizing only the desired actions, instead of all the actions as MLE(since MLE doesn't have negative value). Thus PG can be treated as a weighed MLE

Partial observation(20600)
- Observations might no longer follow Markov properties; however the derivation of policy gradient objective is not being affected. When making updates, simply replace the “state/s” in objectives into “observations/o” and continue doing the update. -> observations here acts like sampling

Variance of policy gradient(30000)
Problems with policy gradient(20740)

- High variance problem(21133)

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



Shown in above image: realizing the optimization of policy might lead to giving all probabilities to the left one action, or all policies will be designed for right two actions only, which can be extreme cases. This modification is possible according to the reward gradient formula, which only has \pi as the only changeable parameter.

- Causality: future cannot affect past(reward) (30100)
  Can be utilized to reduce variance
- Method of reducing variance: modify the reward summation: instead of starting from “t=1”, start from “t=t’ ”, utilizing causality property; (30430)

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)}_{\text{“reward to go”}}$$

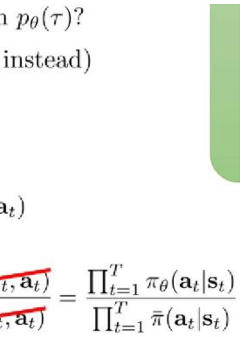
$$\hat{Q}_{i,t}$$

- “Baselines” mechanism for reducing variance (30703)
  If all rewards are positive, take the average (as “b”) and use “r-b” as the reward, so only actions with higher than average rewards will be increased.
  Proof of unbiased estimation(30800)

Analysis of policy gradient variance(31100)

(40000) Off-policy setting

- On-policy: actively engaging with environments, generating new samples every time the parameter is updated
- Gradient descent of neural networks is sometimes tiny, but every iteration of RL policy update requires generating new samples -> inefficient (40200)
- Importance sampling(40330)
  Evaluate current model using samples from a different model
  Derivation involving a new distribution, for calculating current distribution’s expectation
  Image on the left applies importance sampling, using a different distribution(usually taken as previously derived policy)

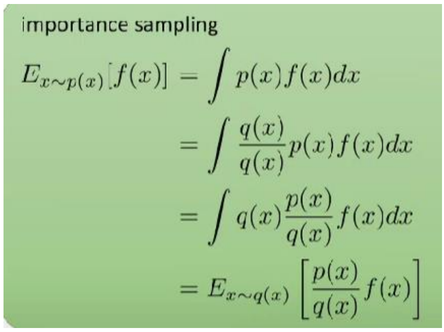


n $p_\theta(\tau)$?

instead)

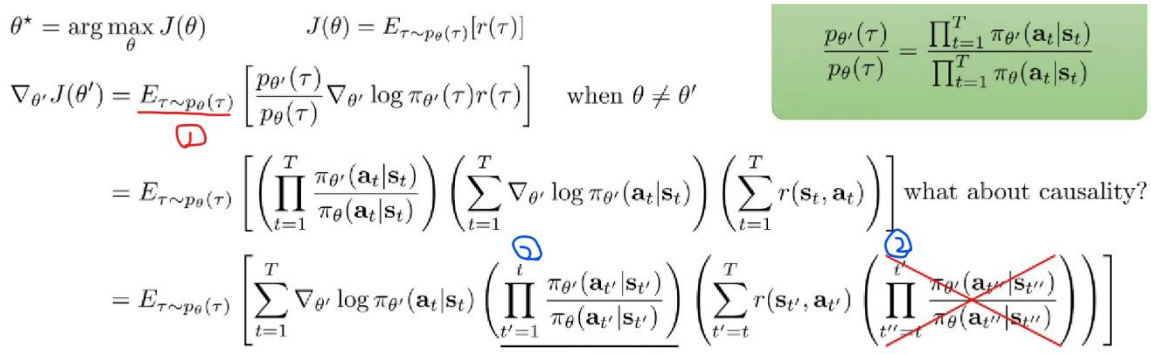
$\mathbf{a}_t)$

$$\frac{\cancel{t,\mathbf{a}_t)}}{\cancel{t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$



importance sampling

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$



$$\theta^\star = \arg\max_\theta J(\theta) \qquad J(\theta) = E_{\tau\sim p_\theta(\tau)}[r(\tau)]$$

$$\frac{p_{\theta'}(\tau)}{p_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$\nabla_{\theta'} J(\theta') = \underbrace{E_{\tau\sim p_\theta(\tau)}}_{1}\left[\frac{p_{\theta'}(\tau)}{p_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right] \quad \text{when } \theta\neq\theta'$$

$$= E_{\tau\sim p_\theta(\tau)}\left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\right)\left(\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^T r(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \text{what about causality?}$$

$$= E_{\tau\sim p_\theta(\tau)}\left[\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\underline{\overset{1}{\left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)}}\left(\sum_{t'=t}^T r(\mathbf{s}_{t'},\mathbf{a}_{t'})\overset{2}{\cancel{\left(\prod_{t''=t}^{t'} \frac{\pi_{\theta'}(\mathbf{a}_{t''}|\mathbf{s}_{t''})}{\pi_\theta(\mathbf{a}_{t''}|\mathbf{s}_{t''})}\right)}}\right)\right]$$

Elaboration for above image: policy gradient using importance sampling: (30856)

1: direct application of importance sampling equation above, treating “p(x)” as “p\theta_prime”, and “q” as “p\theta”: evaluating new policy “theta_prime” using current policy “theta”

2: realizing this is just a division of product into two terms, where the later term could be eliminated (will introduce in later lectures: policy iteration)

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla$$

Final gradient term(41329)

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}$$

Implementing policy gradient (50000)

- (50200)Explicit calculation is inefficient; use maximum likelihood, and multiply with reward to form weighted loss
- Pseudocode as reference(50540)
- training tip(50600)
- Adam, larger batches, adaptive step-size rule

$$\tilde{J}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\log\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\hat{Q}_{i,t}$$

cross entropy (discrete) or squared error (Gaussian)

(60000) advanced policy gradients

Ensuring the update of policy is withing a threshold to avoid high variance in gradient updating(threshold is called “trust region”(61050))

Using KL divergence(60800) to measure the difference between previous state parameter and current state parameter

Approximate using Fisher-information matrix, which is derived using log-likelihood of policy.

usually KL-divergence: $D_{\text{KL}}(\pi_{\theta'}\|\pi_\theta) = E_{\pi_{\theta'}}[\log\pi_\theta - \log\pi_{\theta'}]$

$$D_{\text{KL}}(\pi_{\theta'}\|\pi_\theta) \approx (\theta'-\theta)^T\mathbf{F}(\theta'-\theta) \qquad \mathbf{F} = E_{\pi_\theta}[\nabla_\theta\log\pi_\theta(\mathbf{a}|\mathbf{s})\nabla_\theta\log\pi_\theta(\mathbf{a}|\mathbf{s})^T]$$

Fisher-information matrix   can estimate with samples

$$\text{s.t. } ||\theta'-\theta||^2_{\mathbf{F}} \leq \epsilon$$

trust-region gradient update: image on the left

Lec6: Actor-critic RL algorithm;

Idea of computing expectation of reward over actions at a given state (10400)
Policy gradient: small sample size means high variance(10430)

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$: total reward from taking $\mathbf{a}_t$ in $\mathbf{s}_t$

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total reward from $\mathbf{s}_t$

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better $\mathbf{a}_t$ is

Formulation (10730)
Advantage function (10900)

- Difference between Q and V
- Fit value function, improve policy parameter theta; (11120)

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

Fitting value function: Q/V/A:

- Q: (11200)
  Rewrite Q(T=t) as r(t) + E[V(t+1)](11250)
- Choose V! (only depends on states, not actions)

$$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1})$$

$$A^\pi(\mathbf{s}_t, \mathbf{a}_t) \approx r(\mathbf{s}_t, \mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

let's just fit $V^\pi(\mathbf{s})$!

Policy evaluation: (11620)

- Applied on value function.
- Loss is the expectation over value function

Monte Carlo evaluation with function aproximation

- Use neural networks that given states, approximate V function(183
- Training data & loss function

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t\right]$: to

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total rev

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how muc

training data: $\left\{\left(\mathbf{s}_{i,t}, \underbrace{\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})}_{y_{i,t}}\right)\right\}$

supervised regression: $\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\|\hat{V}_\phi^\pi(\mathbf{s}_i) - y_i\right\|^2$

"y" above is gained by Monte Carlo Sampling

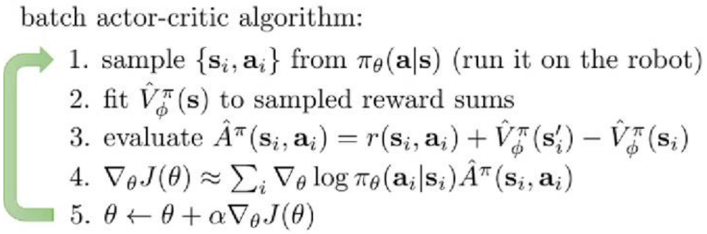
Update using previous model's output (Bootstrap evaluation, lower variance): (2322)
Similar to importance sampling?

$$: \; y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t}\right] \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + V^\pi(\mathbf{s}_{i,t+1}) \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \underbrace{\hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})}$$

- Adopt same loss function as above, by replacing 'y' using Boostrap sampling

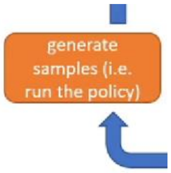
Evaluation to Actor-critic algorithm

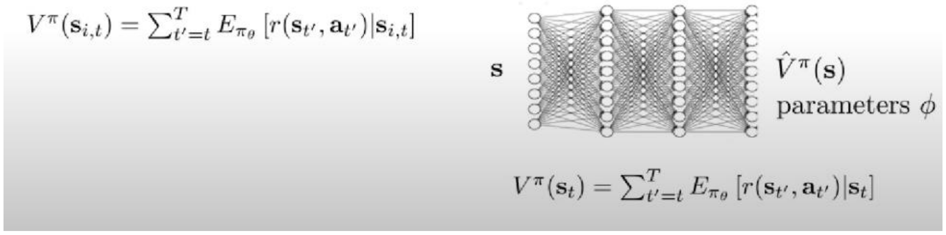
- Mechanism/procedure(20030)



batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$





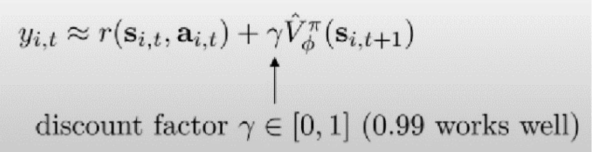
$V^\pi(\mathbf{s}_{i,t}) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_{i,t}\right]$

$V^\pi(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t\right]$

- Si' is the state s_{i+1}, refer to definition of "A" function
- Value function is approximated using neural network
- Loss is still gained as likelihood multiplied by reward

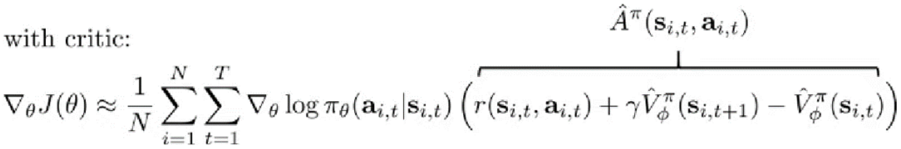
Discount factors(20240)

- If time step is not limited, value will be larg
- Getting reward sooner than later;
- discount factor \gamma
- Updated loss function



$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1})$

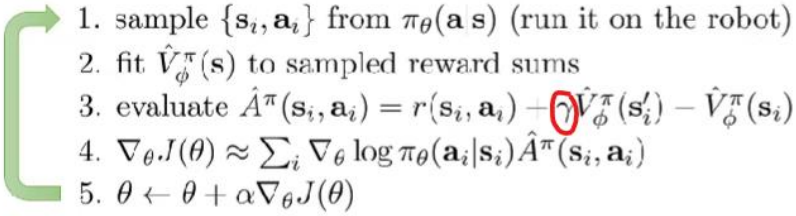
discount factor $\gamma \in [0, 1]$ (0.99 works well)



with critic:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \overbrace{\left(r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}_{i,t+1}) - \hat{V}^\pi_\phi(\mathbf{s}_{i,t})\right)}^{\hat{A}^\pi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}$$

- Choices of using discount factor: (21300)
  The ultimate goal is to help making calculations easier, instead of really decreasing the significance of future reward.
- Discount actor-critic algorithm



1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}\ \mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i)\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Design decisions (30000)

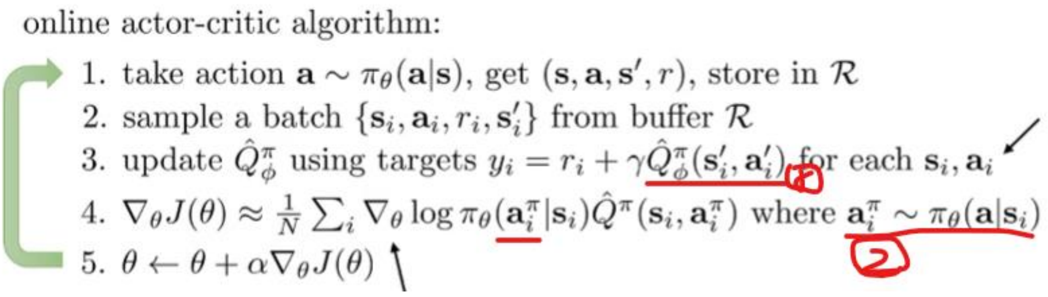
- Two networks: one maps state to value, another maps state to action. (30040)
  But: not efficient, common features not shared.
- Shared NN: take one state, and generate both value and policy.
  Difficult to train, can be unstable.

Batch size(30200)

- Parallel workers: run multiple simulators to generate actions;
- Integrate all samples to update.
- Asynchronous parallel actor-critic: (30400)
  NEED FURTHER CLARIFICATION

Off policy actor-critic:

- Load from history/replay buffer
- Mechanism (31809) “R” is replay buffer

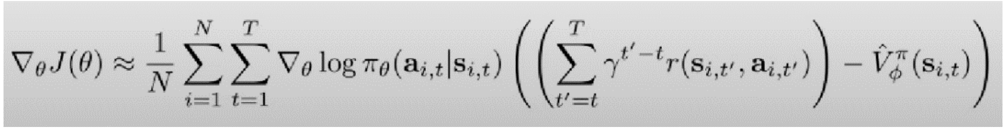


```
online actor-critic algorithm:
    1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$, store in $\mathcal{R}$
    2. sample a batch $\{\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i\}$ from buffer $\mathcal{R}$
    3. update $\hat{Q}^\pi_\phi$ using targets $y_i = r_i + \gamma\underline{\hat{Q}^\pi_\phi(\mathbf{s}'_i, \mathbf{a}'_i)}$ ① for each $\mathbf{s}_i, \mathbf{a}_i$
    4. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \nabla_\theta \log \pi_\theta(\underline{\mathbf{a}^\pi_i}|\mathbf{s}_i)\hat{Q}^\pi(\mathbf{s}_i, \mathbf{a}^\pi_i)$ where $\underline{\mathbf{a}^\pi_i \sim \pi_\theta(\mathbf{a}|\mathbf{s}_i)}$ ②
    5. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$
```

- Points to pay attention to:
  1. Value function comes from previous value function’s estimation, thus when being stored in replay buffer, doesn’t work -> use Q function instead (31120)
     Q is the expectation over reward given action and state, while V can also be interpreted as expectation over actions of Q (31200)
     Sampling a new action when computing Q instead of directly adopting the action in replay buffer(31323)
  2. Action is also old, since it’s from replay buffer.
     Sample a new action at this step using current policy, and use that action for calculating new reward, given “A” function

Critics as base line (40000)

- Actor-critic is not unbiased; trained on finite samples means can have bias, doesn’t generalize to all cases



$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t}r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'})\right) - \hat{V}^\pi_\phi(\mathbf{s}_{i,t})\right)$$

- To reduce the bias of policy gradient: subtract any function that depends only on states would be a method (here the future value function is adopted)(40200)
- Applying a baseline dependent on both actions and states of actor-critic algorithm(40500)

Controlling bias-variance tradeoff by combining the advantages from actor-critic “advantage estimation” and Monte Carlo sampling: (40900)

- Problem: Monte Carlo samples for theoretically large amount of steps, which is not preferred; on the other hand, advanced estimation also has high variance if the critic base line is chosen wrong.
- Using discount factor to reduce the impact:

Generalized advantage estimation(41500)

- Using discount to impose less effect on future states helps reducing the variance of Monte Carlo high variance problem, and allows the model to focus on only next few steps.

Summary (50000)

Lec7: Value Function RL algorithm

Omitting gradient completely and use values as only criterion of next state.

$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$:

- Realizing for state transitions, there is always a best action we could choose to reach the next state, which is irrevelant to likelihood of policy $\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$:
- Just choose the action maximizing value would be enough, and become the policy

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

- Step 2: fitting the model with values; step 3: update the policy following above argmax criterion(10400)

To acquire value function:

- Adopting dynamic programming to store state values(10700)
  However this requires states and actions to be discrete?
  (Can also use a neural network to make estimates for continuous space?)
- V-function: (11000) $V^\pi(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))}[V^\pi(\mathbf{s}')]$
- Using Q value for dynamic programming(11400)
  For each state, there will be an argmax on actions to take, based on the values listed in table.

$$A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$$

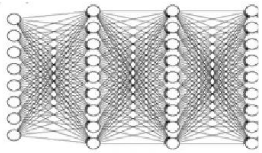
$$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$$

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] \text{ (a bit simpler)}$$

1. set
2. set

t, and use ximating value function

- The only purpose of sampling here is to acquire a broad range of actions could take



$$\mathcal{L}(\phi) = \frac{1}{2}\left\| V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a}) \right\|^2$$

$|\mathcal{S}| =$
(more th

fitted value iteration algorithm:

1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')])$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|V_\phi(\mathbf{s}_i) - \mathbf{y}_i\|^2$

fitted Q iteration algorithm:

1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')]$ ←
2. $\leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

Q iteration algorithm(20800)

- Replacing V function as maximum of Q function over actions "a"
- Works even for offline training
- But might not converge!!!
- Whole mechanism(21200)

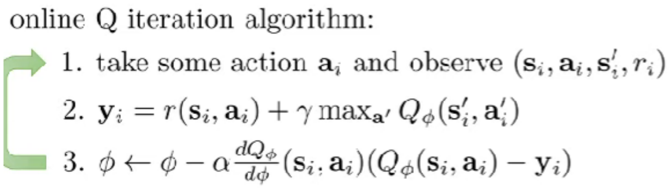
```
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i)\}$ using some policy
    K×
    2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}_i'} Q_\phi(\mathbf{s}_i', \mathbf{a}_i')$
    3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$
```

Q-learning (30000)

- Q-iteration is off-policy(30230)

Online Q-iteration(30600)

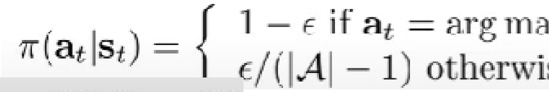
- Interaction with environment using an action, compute the observation of value function and update parameter using differentiation on difference between new value and model's estimated value.



online Q iteration algorithm:

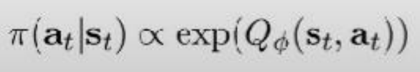
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$

- Exploration-exploitation tradeoff(30800)
  At beginning of action taking for Q-value: allow some randomness instead of always choosing the "optimal action" to achieve exploration
- Epsilon-greedy
  Allows progressively decrease "epsilon" as training progresses



$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1-\epsilon \text{ if } \mathbf{a}_t = \arg \text{ma} \\ \epsilon/(|\mathcal{A}|-1) \text{ otherwi} \end{cases}$

- Boltzmann exploration: using softmax w.r.t Q-value

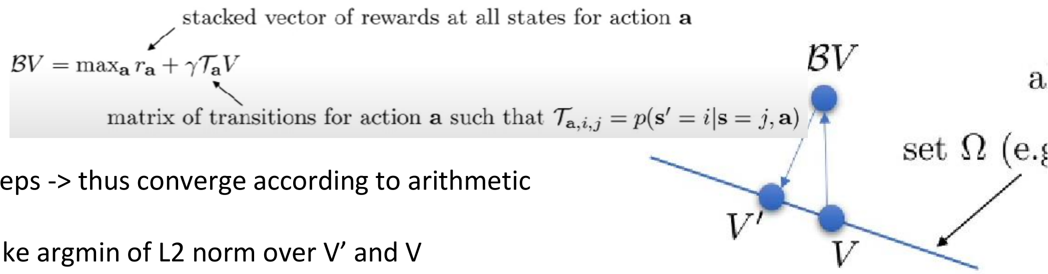


$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$

Summary(31000)

Value function learning theory(40000)

- Will value iteration converge, and whether will converge to optimal policy.
  Operator definition(40200)
  Selecting the action that maximizes the summation of reward and downstream value transitions.
  operator B is a contraction (meaning applying "B" will always lead to less changes than previous steps -> thus converge according to arithmetic series"
  operation "Capital PI": take argmin of L2 norm over V' and V
  "PI-B" is not a contraction of any kind -> doesn't converge.



stacked vector of rewards at all states for action $\mathbf{a}$

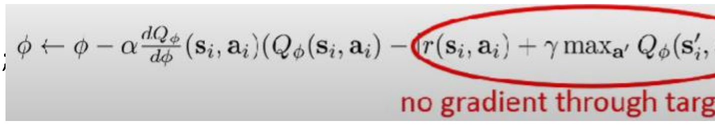
$\mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}} V$

matrix of transitions for action $\mathbf{a}$ such that $\mathcal{T}_{\mathbf{a},i,j} = p(\mathbf{s}' = i|\mathbf{s} = j, \mathbf{a})$

Q-iteration convergence analysis(41200)

- Q-learning(online) isn't gradient descent!!!
  The gradient isn't taking on target value "y"(41400)

In theory, value iterations might not lead to convergence; work quite well.

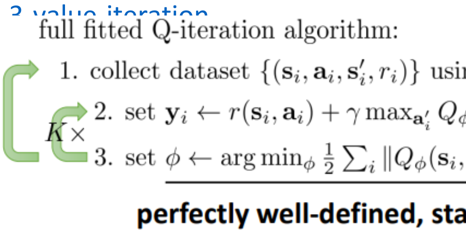


$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i,$

no gradient through targ

Lec8: Q-learning

Useful explanation of value iteration and Q-iteration: https://core-robotics.gatech.edu/2021/01/19/bootcamp-summer-2020-week-3-value-iteration-and-q-learning/

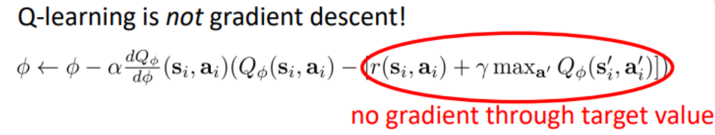


full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ usin
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i,$

perfectly well-defined, sta

- Q-learning is not gradient descent; reason: differentiation doesn't include
- Target netwo
  Fixing no-grac



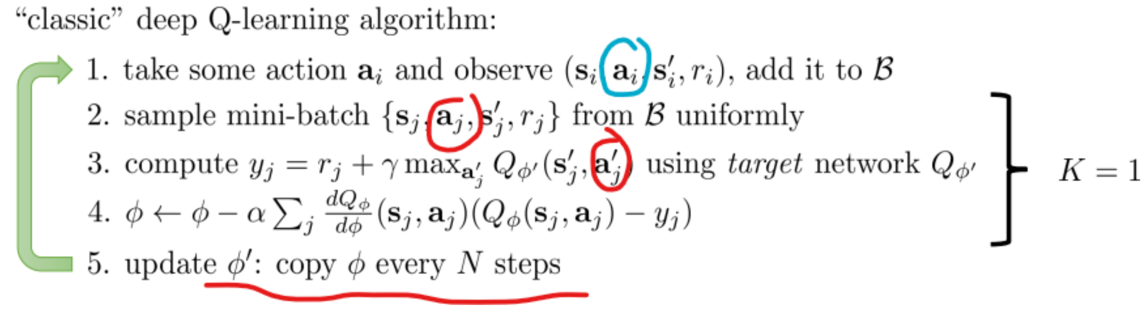
Q-learning is *not* gradient descent!

$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

no gradient through target value

- Deep Q netw

Idea of updating policy network and target/reference network separately, but updating reference network based on policy network

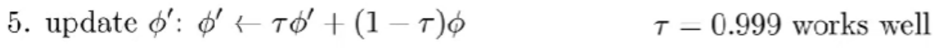


“classic” deep Q-learning algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

$K = 1$

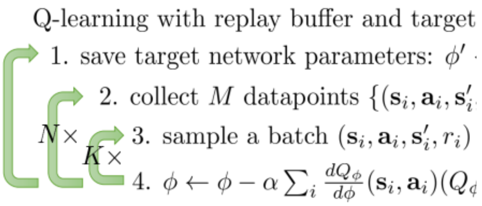
- Target network update intuition(20900)
  Problem of lag(target is updated only one step later)
  Polyak average: updating target network parameter as a combination of policy network and target network(21100)



5. update ϕ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ $\tau = 0.999$ works well

(30000)

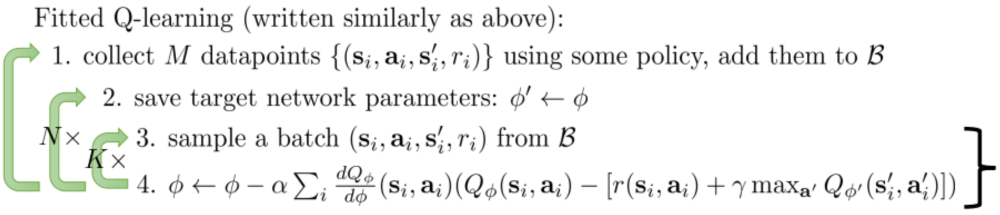
- Q learning with replay buffer (30500) and fitted Q-learning



Q-learning with replay buffer and target

1. save target network parameters: ϕ' ·
2. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi$

$N\times$ $K\times$



Fitted Q-learning (written similarly as above):

1. collect M datapoints $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add them to $\mathcal{B}$
2. save target network parameters: $\phi' \leftarrow \phi$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

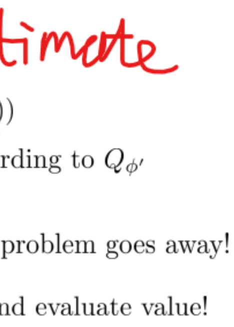
$N\times$ $K\times$

just SGD

Realize target/reference network is just the policy network of previous update.

- Process speed(30600)
  Online-Q learning: all processes run at same speed
  DQN updates parameter much slower compared with online Q-learning(30700)

Improving Q learning(40000)



timate

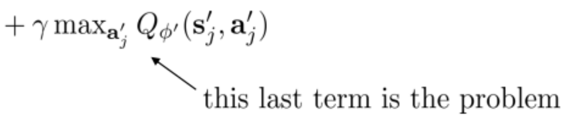
)

rding to $Q_{\phi'}$

problem goes away!

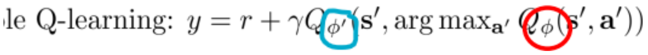
nd evaluate value!

- Problem of overestimation(40800) need further clarification
  The action maximizing the Q-value function leads to biased expectation of max function



$+\gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

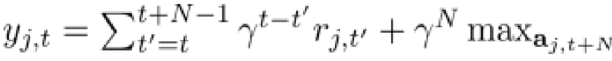
this last term is the problem



le Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}'))$

- Double Q-learning: resolving noise due to argmax (41600)
  Use current policy network for action evaluation, and target network for value.

- Multi step returns(41600)
  Maximize bias, minimize variance
  Predict “n” future steps instead of immediate next step.
  Less biased target values(when Q is inaccurate, and faster for training)(42000)

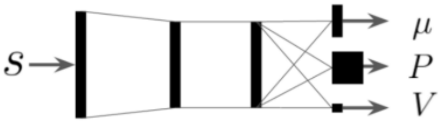


$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}}$

Continuous actions(50000)

- Estimate continuous actions by discrete approximations(50200)
  But not very accurate
  CMA-ES: complex but more reliable
- Use functions that are easy to optimize(50500), such as quadratic

$$Q_\phi(\mathbf{s},\mathbf{a}) = -\frac{1}{2}(\mathbf{a}-\mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s})(\mathbf{a}-\mu_\phi(\mathbf{s})) + V_\phi(\mathbf{s})$$

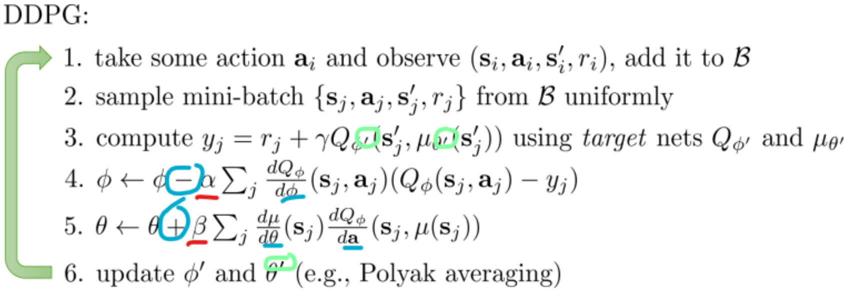


  Normalized advantage functions $\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = \mu_\phi(\mathbf{s})$ $\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = V_\phi(\mathbf{s})$
  Efficient but lose representation power for Q value.

- Approximated maximizer(50600)
  Train a neural network that approximates the action yielding maximum Q value.

$$y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \boxed{\mu_\theta}(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j,\mathbf{a}'_j)) \qquad \mu_{\boxed{\theta}}(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a})$$

  DDPG: act and add to buffer -> sample from buffer -> acquire target value y using approximated function -> update policy parameter -> update action approximation parameter -> update target policy parameter and action approximation parameter



DDPG:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma Q_{\phi\boxed{'}}(\mathbf{s}'_j, \mu_{\theta\boxed{'}}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi \boxed{-} \underline{\alpha} \sum_j \frac{dQ_\phi}{\underline{d\phi}}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta \boxed{+} \underline{\beta} \sum_j \frac{d\mu}{\underline{d\theta}}(\mathbf{s}_j) \frac{dQ_\phi}{\underline{d\mathbf{a}}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$
6. update ϕ' and $\boxed{\theta'}$ (e.g., Polyak averaging)

Implementation tips(60000)

- Start with testing algorithms on easy, reliable tasks first; to see the implementation is correct
- Large replay buffer improves stability;
- Start with high exploration (epsilon greedy) and decrease gradually
- Huber loss(60300)

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$

- Double-Q-learning is always recommended
- Multiple random seeds for robustness

Examples(60500)

Lec9: Advanced policy gradient

eration

$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

$$\text{claim:} J(\theta') - J(\theta) = E_{\tau\sim p_{\theta'}(\tau)}\left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)\right]$$

Reinforcement learning objectives recap(check how theta_prime and theta are converted using importance sampling.)

- (10900)Trajectory of current policy subtract by trajectory of previous policy
  Derivation of above claim(11300)

$$= E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0} \gamma^t (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^{\pi_\theta}(\mathbf{s}_{t+1}) - V^{\pi_\theta}(\mathbf{s}_t)) \right]$$

$$= E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Second line of derivation: initial state is independent of any policy sampling trajectories, thus could replace the expectation of s_0 state into whole trajectory on theta_prime

- Using importance sampling to rewrite the difference between old and new policys' rewards:

$$E_{\tau \sim p_{\theta'}(\tau)} \left[\sum_t \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] = \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)} \left[\gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$= \sum_t E_{\mathbf{s}_t \sim p_{\theta'}(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

(20000): trying to replace the p_theta_prime into p_theta in above summation.

- Proof: try to show the difference between p_theta_prime and p_theta are small that could be bounded up to a constant. The condition is met when two policies (pi_theta_prime and pi_theta) are also quite close.
- (20200) proof started.
- (20900) generalized proof for any arbitrary policy.
  Set up a bound for "pi". (20100) gives the bound for two distributions given two policies.
- Final optimization objective for advanced policy gradient(21810)

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right]$$

$$\text{such that } D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$$

(30000) policy gradient with constraints

- KL divergence(30100)
  Convenient properties for easy approximations

- Lagrange multiplier optimization(30300)

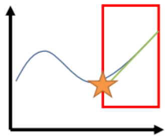
$$\mathcal{L}(\theta', \lambda) = \sum_t E_{\mathbf{s}_t \sim p_\theta(\mathbf{s}_t)} \left[E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} \gamma^t A^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \boxed{- \lambda (D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) - \epsilon)}$$

1. Maximize $\mathcal{L}(\theta', \lambda)$ with respect to θ' ← **can do this incompletely (for a few grad steps)**
2. $\lambda \leftarrow \underline{\lambda + \alpha(D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) - \epsilon)}$

Dual gradient descent

Natural gradient (40000)

- Using Taylor expansion along a point, and minimize the linear(1st order Taylor term)(40300)
  Trust region



$$\theta' \leftarrow \arg\max_{\theta'} \nabla_\theta \bar{A}(\theta)^T (\theta' - \theta)$$

$$\text{such that } D_{\mathrm{KL}}(\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \| \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)) \leq \epsilon$$

Use first order Taylor approximation for objective (a.k.a., linearization)

- Gradient ascent for policy optimization(40800)
- Fisher information, quadratic Taylor and KL(41200)

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

second order Taylor expansion

$$D_{\mathrm{KL}}(\pi_{\theta'} \| \pi_\theta) \approx \frac{1}{2} (\theta' - \theta)^T \mathbf{F} (\theta' - \theta) \qquad \mathbf{F} = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s})^T] \qquad \alpha = \sqrt{\frac{2\epsilon}{\nabla_\theta J(\theta)^T \mathbf{F} \nabla_\theta J(\theta)}}$$

Fisher-information matrix

Gradient update based on KL and fisher information(41500)
(Natural policy gradient)

- Practical notes(41700)

Review (41900)

- Old policy state distribution optimizes a bound, if the policies are close enough(epsilon bound)
- First order approximation objective: gradient ascent
- Second order approximation: Fisher information, KL divergence and gives natural policy gradient.

Lec10: Optimal control and planning

Model-based: knowing transition dynamics: **p(s’|s, a)**, idea of planning;
Known dynamics(10400): rules of games; easily modeled situations(usually involves mathematical formulas for deterministic transitions); simulated environments with known configurations

Learning dynamics(10600): using specifis model and fit the configurations; using general approximation model and fit the parameters(this general approximation model can be a completely different distribution or model compared with original, just like variational inferences).

(assume dynamics are given to us for this lecture. General cases in later lectures)

- Constraint problem for maximizing reward or minimizing cost(11000)

Optimization objectives(11300)
Suboptimal objective: if information could be revealed in future that leads to better policies.

$$p_\theta(\mathbf{s}_1, \ldots, \mathbf{s}_T | \mathbf{a}_1, \ldots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \ldots, \mathbf{a}_T\right]$$

Open-loop environment(11600):

- Only observe one state, and must commit all subsequent actions without observing other states. (this challenges model-based learning’s reliability)
- Closed loop allows making actions after observing new states.

Open-loop algorithms(20000)

Optimization objective:

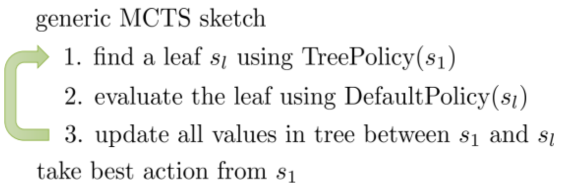
$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \ldots, \mathbf{a}_T)}_{} \qquad \mathbf{A} = \arg \max_{\mathbf{A}} J(\mathbf{A})$$

- Guess & check: sample actions, and take argmax from sampled models.
  Works on small models, and could be fast on small models
- Cross entropy method(20400)
  Use multiple distributions to sample actions, and choose the “best/elite” distribution based on evaluation of generated action samples, and optimize using more generated samples from only “elite” distributions; typically use Gaussian.

- Strengths and weakness: fast(if parallelized), simple/dimensionality limit, only applies to open-loop planning

- Monte Carlo Tree search(21000)
  Take multiple actions at each state as sequences of trajectories sampled; to avoid long sequences, using heuristics being described as DEFAULT policies(pi(a|s))
  Choosing priority on search paths; (21310)
    Prioritize depth for better rewards, but also take care of breadth by accounting unvisited nodes; update whenever after taking one action
  Generic MCTS: (21600)
    See image at right for full algorithm(update state value as described above)
  TreePolicy(s_t):
    if s_t is not fully expanded: choose new action; otherwise choose the child currently yielding best reward.
    (21800)
  Score for choosing child state:

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

take best action from s_1

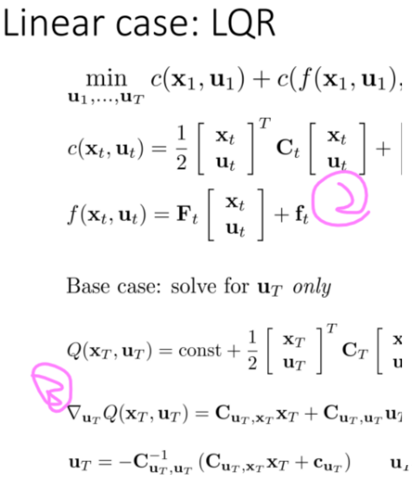
Trajectory optimization with derivatives(30000)

- Notation for optimal control problem: “x, u”: state & action; use cost as rewards(30100)
- Optimization objective:

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \dots + c(f(f(\dots)\dots),\mathbf{u}_T)$$

- Using derivatives: using 2nd order derivative would be better.
- Shooting method(30400): optimize over action only
  Collocation: (30600) optimize over both states and actions
- LQR(30700): linear case optimization;
  Basic settings:
  1: uT and xT are unknowns;
  2: using linear and quadratic approximation, where coefficient matrices indicates costs(4).
  Realizing quadratic forms are used for finding the local minima, thus lowering the cost.
  3: find gradient of cost approximation function

- Solving u{T-1}, x{T-1}(31600)
- General recursion of LQR(32100)
  (derivation see previous parts)



Linear case: LQR

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} +$$

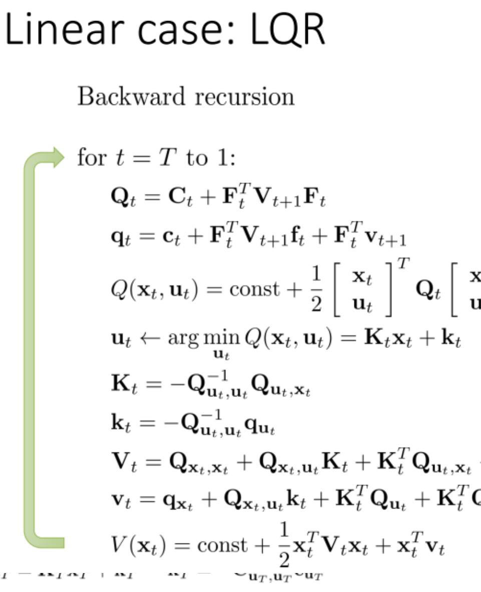
$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t$$ 2

Base case: solve for $\mathbf{u}_T$ *only*

$$Q(\mathbf{x}_T,\mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}\\ \mathbf{u}\end{bmatrix}$$

3 $$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T,\mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T$$

$$\mathbf{u}_T = -\mathbf{C}^{-1}_{\mathbf{u}_T,\mathbf{u}_T}(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T})$$



Linear case: LQR

Backward recursion

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T\mathbf{V}_{t+1}\mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T\mathbf{V}_{t+1}\mathbf{f}_t + \mathbf{F}_t^T\mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t,\mathbf{u}_t) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix}\mathbf{x}\\ \mathbf{u}\end{bmatrix}$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}\mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}^{-1}_{\mathbf{u}_t,\mathbf{u}_t}\mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t,\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t}\mathbf{K}_t + \mathbf{K}_t^T\mathbf{Q}_{\mathbf{u}_t,\mathbf{x}_t}$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t,\mathbf{u}_t}\mathbf{k}_t + \mathbf{K}_t^T\mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T\mathbf{C}$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2}\mathbf{x}_t^T\mathbf{V}_t\mathbf{x}_t + \mathbf{x}_t^T\mathbf{v}_t$$

LQR for stochastic and non linear systems(40000)

- Instead of giving deterministic points, a distribution with mean and variance is give. Follow the transition from autoencoder to variational autoencoder to incorporate LQR from previous part

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$$

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t, \Sigma_t\right)$$

Solution: choose actions according to $\mathbf{u}_t = \mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t$

- Handling non-linear case: using multivariable Taylor expansion, to use linear terms and quadratic terms as in LQR to approximate non linear functions at a specific point?(40400)

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

- Non linear LQR algorithm(40600)

until convergence:

$\mathbf{F}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$

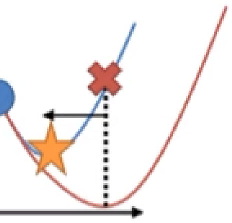
$\mathbf{c}_t = \nabla_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$

$\mathbf{C}_t = \nabla^2_{\mathbf{x}_t,\mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$

Run LQR backward pass on state $\delta\mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$ and action $\delta\mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with $\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \alpha\mathbf{k}_t + \hat{\mathbf{u}}_t$

Update $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ based on states and actions in forward pass



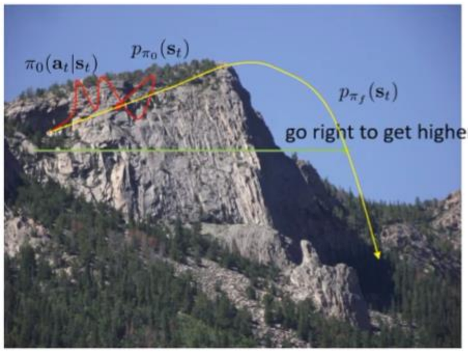
- Newton method is not a good idea?(41000) & comparison with iterative LQR(41100)
  Parameter "alpha" adjusts the terms, and offers some control for not deviating too much.

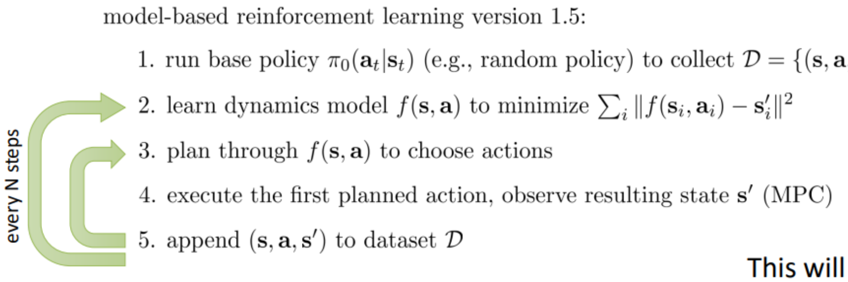
Case study and additional readings(50000)

Lec11: Model based reinforcement learning algorithm

Brief introduction to a toy example(10200)

- Learning a environment dynamics model
- Simple models might not work, due to: using the environment gained at beginning as generalized environment, yet the agent hasn't fully explored the environment yet, thus won't change actions accordingly if prior of environment is wrong.
  Confirmation bias; (10700)
- A qualified model-based RL should learn CONTINUOUSLY along the way (11219) model version 1.0
- Model version 1.5: apart from continously learn, the learning could be applied immediately by only executing the very first action, and adjust accordingly. (11500)


$\pi_0(\mathbf{a}_t|\mathbf{s}_t)$
$p_{\pi_0}(\mathbf{s}_t)$
$p_{\pi_f}(\mathbf{s}_t)$
go right to get higher!


model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

This will be on

Regarding re-planning(step 3)(11600)

- Replan more means less requirements on replan quality -> use shorter time horizons;

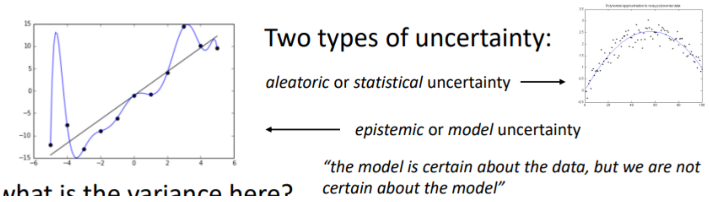
Uncertainty in model-based-RL(20000)

- Model based learning sometimes perform poorly than model-free learning
  I think this addresses the reason why model based learning is not preferred:
  Collecting data and training environment model impose unique challenges to training the model appropriately: beginning of training doesn't have much samples, and can lead to overfitting. Neural networks are data-hungry; (20200)
- Uncertainty estimation(20400)
  Confidence interval, expected region & danger detection (20600)

- Incorporating uncertainty into training requires modifying step 3: taking actions maximizing EXPECTATION of reward(20700)

Uncertainty-aware neural network models(30000)

- Using entropy(30100) softmax; not sufficient: require reviewing; (30300)
- Two types of uncertainty: model/statistical: not choosing the right model/data is noisy



Two types of uncertainty:

aleatoric or *statistical* uncertainty

epistemic or *model* uncertainty

"the model is certain about the data, but we are not certain about the model"

what is the variance here?

- Uncertainty-aware: instead of selecting a deterministic model, using a distribution to model the parameters; the integral right can be written as discrete summations when given samples of actions and states. (30700)

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta$$

- Bayesian neural network is one alternative(30800)
  Likelihood approximation is not adequate(30900)
  Using a Gaussian model as prior for weight initializations of bayesian neural network
- Bootstrap(31100)
  Training multiple models and see if they are consistent; (more like an averaging over emsembled models) (31300)
  Need to ensure independence of data sampling for each independent model. Idea: sampling with replacement from data(31500); but applying SGD and random initialization also gives independence. (31700)

$$p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i)$$

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

Planning with uncertainty(40000)

How to calculate rewards given bootstrapping of models: using averages/expectations

$$J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{H} r(\mathbf{s}_{t,i}, \mathbf{a}_t), \text{ where } \mathbf{s}_{t+1,i} = f_i(\mathbf{s}_{t,i}, \mathbf{a}_t)$$

- Taking into account the number of models (there are N many)(40200)

Step 1: sample $\theta \sim p(\theta|\mathcal{D})$

Step 2: at each time step t, sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$

Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

Step 4: repeat steps 1 to 3 and accumulate the average reward

- Examples of ensembles(40400)
- Model based learning with model free fine tuning;

Incorporate images (50000)

- Images are high dimensional, redundancy, can be observed only partially;
- Ising model(CSC412, message passing&image denoising)? Observations and hidden states?
- Knowns and objectives to measure(50300) transition dynamics, observations, latent state learning conditioned on observations(posterior required)
- Learn approximate posterior using neural network(50500)

Lec12:

Will introduce variational inference and reparameterization trick this week. Check the coincidences with CSC412 and STA414.

Lec13: