TJHSST AI/ML CLUB SEMINAR SERIES

Introduction and Recent Advances in Deep Reinforcement Learning

DR. RAJ DASGUPTA

NAVAL RESEARCH LABORATORY, WASHINGTON D. C.

EMAIL: RAJ.DASGUPTA@NRL.NAVY.MIL

Outline

- Review of Reinforcement Learning (RL)
  - Action-Value Methods
    - Q-learning
    - Deep Q-learning
  - Policy Based Methods
    - REINFORCE
    - Actor-Critic Learning
      - A2C (Advantage Actor-Critic) and A3C (Asynchronous A2C) Learning algorithm
    - Proximal Region Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO) algorithm
- Background Required:
  - Convolutional Neural Network (CNN) – for deep RL
  - Markov Decision Processes (MDP) – for mathematical framework underlying RL

Reinforcement Learning

- Recall: Supervised (and unsupervised) learning algorithms learn a hypothesis that is consistent with the distribution of data used to train the algorithm
- Reinforcement learning (RL) uses a reward function to learn a policy
  - Policy: mapping from state to action – what to do in which situation
  - Policy must maximize the reward that the agent gets (from solving the problem)

RL Framework

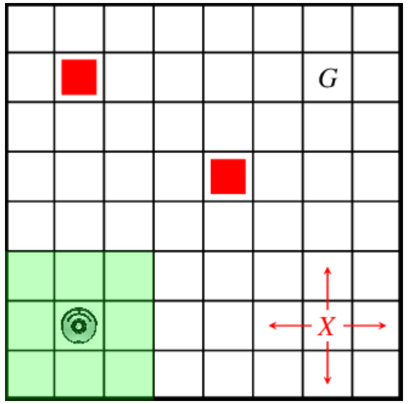
- Representation of the problem being solved by the RL algo
- Mostly represented as a stochastic process called Markov Decision Process (MDP)
- MDP has 4 attributes: (S, A, T, R)
  - S: State space: what are the states of the problem
  - A: Action space: what are the actions that the agent can take
  - T: Transition Function, also called model of the problem: if the agent takes a certain action at a certain state, what *next* state does it end up in ?
  - R: Reward Function: what is the reward that the agent gets when it reaches a state?
- MDP output is a policy, denoted by π
  - π is a mapping from state space to action space: what (is the best) action that the agent should take for every state in S

Example of MDP: Gridworld

- Objective: Robot has to reach G from its start location (green)
- MDP formulation:
  - S: all the cells of the grid, e.g., (0, 0) (0, 1)…
  - A: North, South, East, West (red arrows on bottom right)
  - T: e.g., if robot does action N at (1, 1) it reaches (1, 2)
    - Written as a function: T((1, 1,), N) = (1, 2)
    - T could be probabilistic too: Doing N at a cell takes the robot one cell north 90% of the time, but takes it one cell east or west 5% of the time resp.
      - Written as: T ((1, 1), N, (1, 2)) = 0.9; T ((1, 1), N, (0, 1)) = 0.05; T ((1, 1), N, (2, 1)) = 0.05…for each cell and for each action
  - R: e.g., +10 for reaching a cell 1-hop from G, +8 for cells 2-hops from G and so on (e.g., R(6, 5) = 10, R (7, 4) = 8, and so on…defined for every cell)
- Remember - Output is policy π – what action robot should take at each state

Given as input in MDP

Solved using dynamic programming, e.g., Bellman update equations



8 x 8 grid world

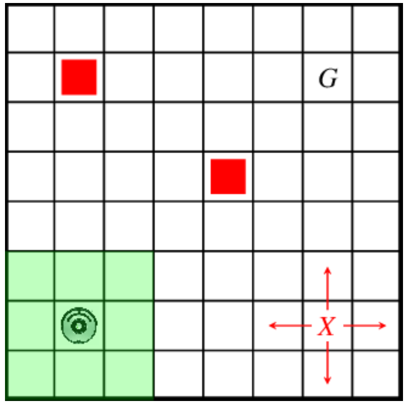
Example of MDP as RL Model: Gridworld

- Objective: Robot has to reach G from its start location (green)
- MDP formulation:
  - S: all the cells of the grid, e.g., (0, 0) (0, 1)…
  - A: North, South, East, West (red arrows on bottom right)
  - T: e.g., if robot does action N at (1, 1) it reaches (1, 2)
    - Written as a function: T((1, 1,), N) = (1, 2)
    - T could be probabilistic too: Doing N at a cell takes the robot one cell north 90% of the time, but takes it one cell east or west 5% of the time resp.
      - Written as: T ((1, 1), N, (1, 2)) = 0.9; T ((1, 1), N, (0, 1)) = 0.05; T ((1, 1), N, (2, 1)) = 0.05…for each cell and for each action
  - R: e.g., +10 for reaching a cell 1-hop from G, +8 for cells 2-hops from G and so on (e.g., R(6, 5) = 10, R (7, 4) = 8, and so on…defined for every cell)
- Remember - Output is policy π – what action robot should take at each state

Given as input in RL

X

Solved using dynamic programming + Q-learning or DQN (value function based), REINFORCE, A3C, PPO (policy based)



8 x 8 grid world

Value Based Method for RL

Q-LEARNING

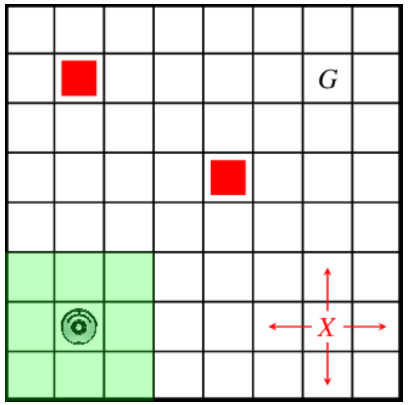
Q-Learning

- Recall input to RL is (S, A, ~~T~~, R) and output is policy π
- Recall T is called the *model* of the problem
- Q-learning does not try to re-construct T (some other RL algorithms do)
  - Q-learning is a model-free RL algorithm
- Main idea of Q-learning:
  - Build a table called q-table
  - Table is indexed by S X A: cross product of state and action spaces of problem
    - E.g., for the 8 x 8 grid world with a deterministic robot (one that moves only north when it does N)
    - 64 states (cells) and 4 actions of the robot – gives 64 * 4 = 256 entries of Q-table
  - Content of q-table gives the value of each state, action pair (initialized to reward of each state)

Index	Q
(0, 0), N	0.2
(0, 0), S	0.2
(0, 0), E	0.2
(0, 0), W	0.2
(1, 1), N	0.6
…	…
(256 rows)	

Q-table of grid world showing Q-values initialized to (example) reward of each state

- Main operation in Q-learning: Update the Q-table by letting the agent visit different states of the problem and taking different actions at each state
  - Called exploration
- Takes the form of state-action sequence $s_1, a_1, s_2, a_2, \ldots, s_T, a_T$
  - At state s_1 take action a_1 which gets the agent to state s_2; take action a_2 in s_2 which takes agent to state s_3 and so on
  - Sequence is called an episode or a trial or a sample
  - Each (s_i, a_i) pair inside sequence is called a step
- At each step, update the Q(s, a) value of the state agent is in
  - Using a Q-update function, given by $\Delta Q(S_t, A_t) = \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$
- Repeat for multiple episodes
- Stopping criterion of Q-learning algo: When Q(s, a) values inside the Q-table are not changing (significantly) over successive episodes - called convergence



For our grid world example, a trial would be a path that the robot takes; the Q-table would be updated by letting the robot explore different paths inside the grid world

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal

Pseudo-code for Q-learning algorithm

Index	Q
(0, 0), N	0.2
(0, 0), S	0.2
(0, 0), E	0.2
(0, 0), W	0.2
(1, 1), N	0.6
…	…

Initial Q-table

Index	Q
(0, 0), N	0.12
(0, 0), S	0.005
(0, 0), E	0.87
(0, 0), W	0.005
(1, 1), N	0.6
…	…

Final Q-table

Replaces the Policy

Deep Q-learning

DEEP Q-NETWORKS

DQN Objective

- Conventional Q-learning suitable for
  - Features are handcrafted
  - Fully observable, low dimension state spaces
- DQN allows Q-learning to handle high dimensional sensor inputs
- Q(s,a) function (action-value function) can be estimated with a function approximator parameter $Q(s, a; \theta)$
- DQN Idea: Function approximator implemented as deep neural network called Q-network

Neural Network-based Function Approximator Issues

- Neural network is a non-linear function
- Issues with approximating Q() using non-linear function:
    1. Correlations in sequence of observations
    2. Small updates to Q() could significantly change policy and therefore, the data distribution
    3. Correlations between Q and target values $r + \gamma \, max_a \, Q(s, a)$

Solutions

Experience Replay (addresses 1 and 2)
- Randomizes over data
- Removes correlations in observation sequences
- Smooths over changes in data distribution

Iterative update (addresses 3)
- Reduces correlations between Q-values and target value

DQN: Experience Replay

- The state is a sequence of actions and observations $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$
- $e_t = (s_t, a_t, r_t, s_{t+1})$, called experience
- $D = e_1, \ldots, e_n$, called replay memory
- Difficult to give the neural network a sequence of arbitrary length as input
  - Use fixed length representation of sequence/history produced by a function $\varphi(s_t)$

DQN Architecture

- Input: 84 X 84 X 4
- Conv layer 1: 32 filters, 8 X 8, stride = 4
- Activation layer: ReLU
- Conv layer 2: 64 filters, 8 X 8, stride = 2
- Activation layer: ReLU
- Conv layer 3: 64 filters, 3 X 3, stride = 1
- Activation layer: ReLU
- Fully connected layer, 512 ReLUs
- Output layer: Fully connected, no. of outputs = no. of actions

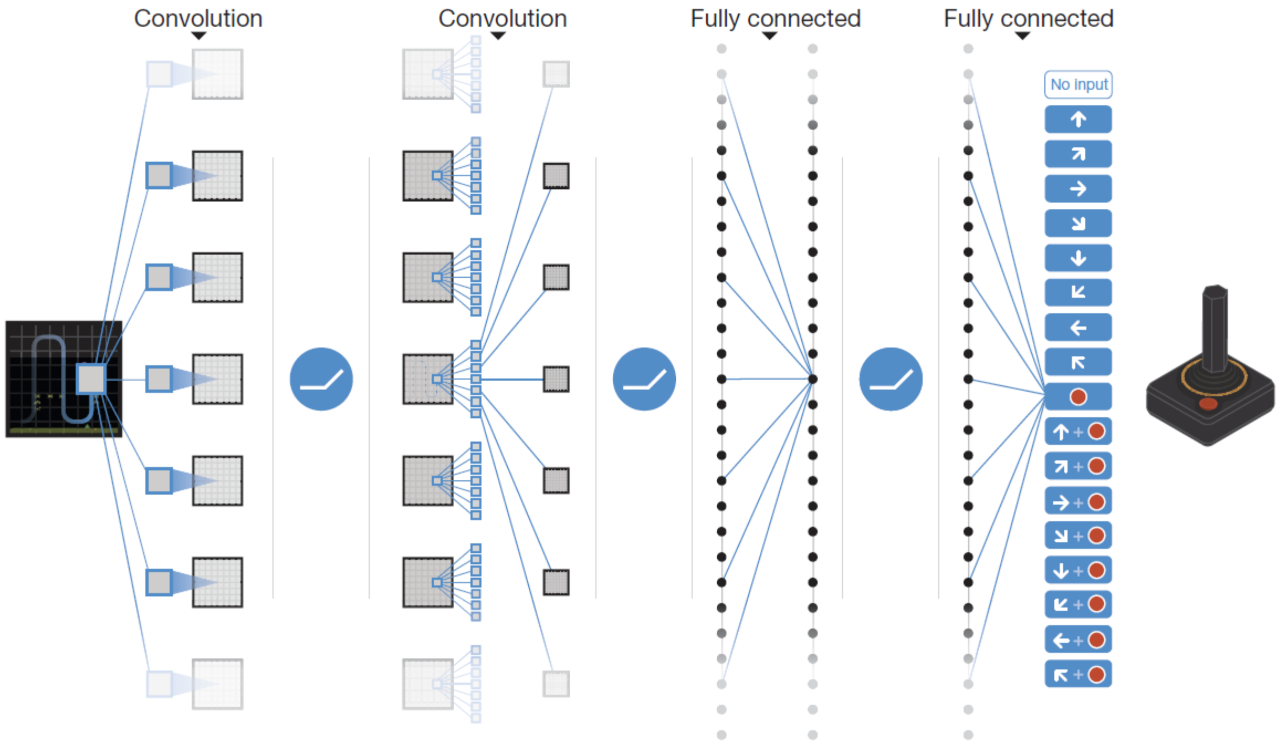


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).

Q-Network Training

$$\Delta Q(S_t, A_t) = \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$$

- θ_i: set of network weights in iteration i
- Sample random set of experiences uniformly at random from D (replay memory), called mini-batch
- Similar to Q-learning update rule but:
  - Use mini-batch stochastic gradient updates
  - Calculate gradient of loss function, L
    - The gradient of the loss function for a given iteration with respect to the parameter θ_i is the difference between the target value and the actual value is multiplied by the gradient of the Q function approximator Q(s, a; θ) with respect to that specific parameter
- Use the gradient of the loss function to update the Q function approximator

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$$

$$\nabla_{\theta_i} L(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

- θ_i^-: weights of target network
- θ_i: weights of Q-network
- Target network weights are updated (copied from Q network weights) every C steps (iterative update)

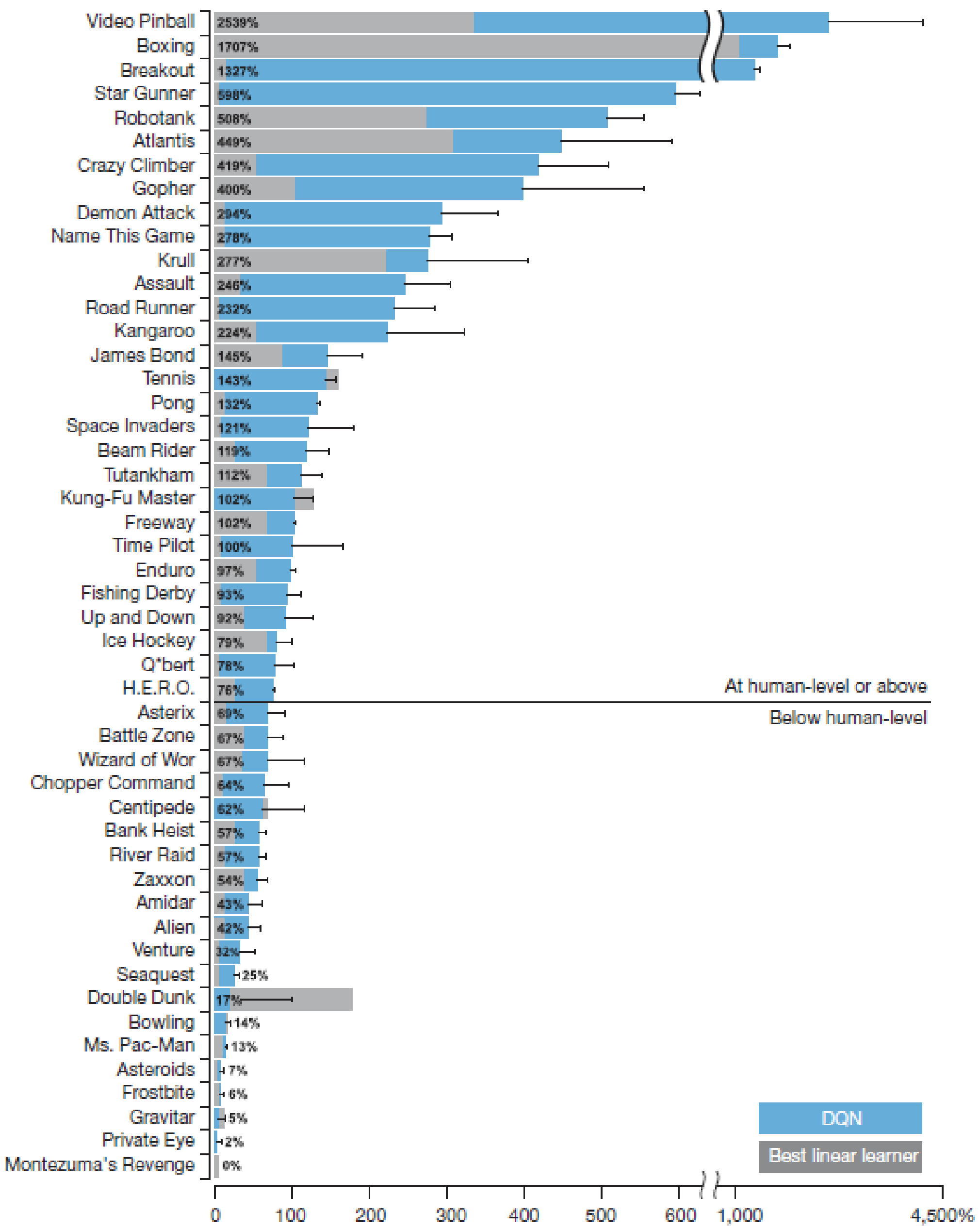
DQN Training Algorithm

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DQN Performance Comparison

For playing different Atari games



Two Main Concepts that make DQN work well

- Experience buffer: stores the agent's data so that it can be randomly sampled from different time-steps
  - Requires more memory and computation per real interaction than online updates
  - Requires off-policy learning algorithms that can update from data generated by an older policy
- Iterative Update: Aggregating over memory reduces non-stationarity and de-correlates updates but limits methods to off-policy RL algorithms

DQN Resources

- DQN/Deepmind https://deepmind.com/research/dqn/
- Dopamine - Google Tensorflow Deep RL framework https://github.com/google/dopamine/tree/master/docs#downloads

Additional Resource

- R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, 2018. (open source pdf: http://www.incompleteideas.net/book/the-book-2nd.html)
- Slides: Richard Suttons RL Tutorial at NIPS 2015 http://media.nips.cc/Conferences/2015/tutorialslides/SuttonIntroRL-nips-2015-tutorial.pdf
- Video Tutorial on RL, Q-learning ~1 hr https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT
- Video DeepMind Course on RL (10 lectures, 1.5 hrs each) https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT

- 8 weeks of research internships high school students to participate at Department of Navy Laboratories including NRL
- Major criteria:
  - Completed Grade 9
  - Graduating seniors can apply
  - Must be 16 years or older at time of application
  - U S Citizenship (for NRL)
- NRL research areas: AI/ML, computer science, engineering, space sciences, radar, remote sensing, plasma physics, chemistry, bio-sciences, material sciences, acoustics
- Application deadline: November 30, 2020
- Website: https://seap.asee.org/

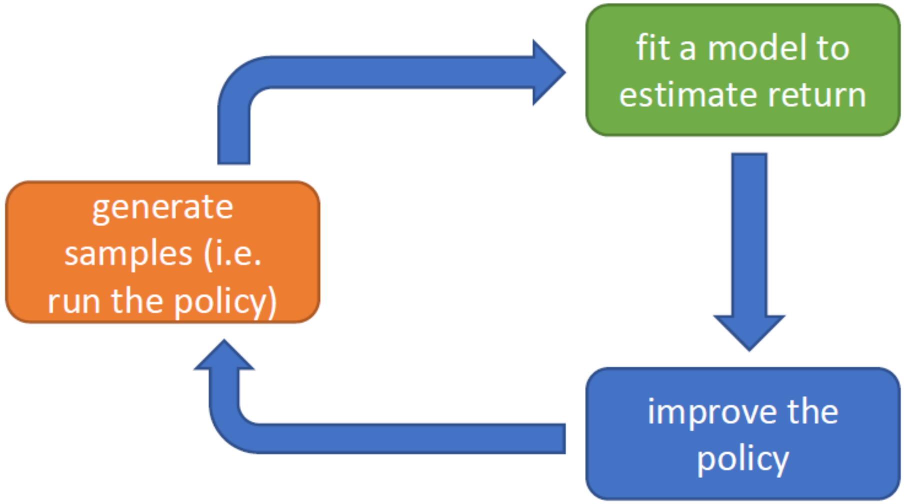
Policy Based Methods for RL

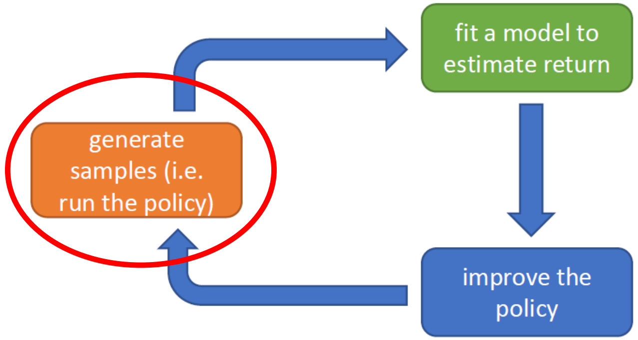
Why Policy Based?

- Value-based RL does not work in continuous spaces
- Recall:
  - Policy π is a mapping from the state space S to the action space A (of a problem), i.e., $\pi : S \rightarrow A$
  - Policy must maximize the reward that the agent gets (from solving the problem)
- Value-based RL
  - π is represented in the form of a table (works for discrete state, action spaces)
- Policy-based RL
  - π is represented in the form of (continuous) function parameterized by parameter θ
- $J(\theta)$: performance measure of the policy parameterized by θ
  - Requirement: $J(\theta)$ must be continuous and differentiable,
    - $\Delta_\theta J(\theta)$ represents the derivative of $J(\theta)$ w.r.t. θ

Policy Gradient Algorithm

- Three steps for policy gradient algorithm, run iteratively:
  - Generate samples with current policy (initial policy: random)
  - Determine the expected rewards from samples
  - Update policy parameter





- Trial (also called trajectory)
  - Sequence of state-action pairs executed by the agent while following policy parameterized by θ
  - Denoted by $s_1, a_1, s_2, a_2, \ldots, s_T, a_T$
  - Probability of selecting a trial denoted by $p_\theta(s_1, a_1, s_2, a_2 \ldots, S_T, a_T)$, or, in short as $\pi_\theta(\tau)$
- Let us evaluate this probability (in terms of values available from the model)
- Consider a two-sequence trial (s_1, a_1, s_2, a_2)
- Probability of doing trial (s_1, a_1, s_2, a_2), i.e., $p_\theta(s_1, a_1, s_2, a_2)$ is:
  - Prob. of starting in state s_1 → $p(s_1)$
  - Prob. of choosing action a_1 in state s_1 → $\pi_\theta(a_1 | s_1)$
  - Prob. of reaching state s_2 by doing action a_2 in state s_1 → $p(s_2 | s_1, a_1)$
  - Prob. of choosing action a_2 in state s_2 → $\pi_\theta(a_2 | s_2)$

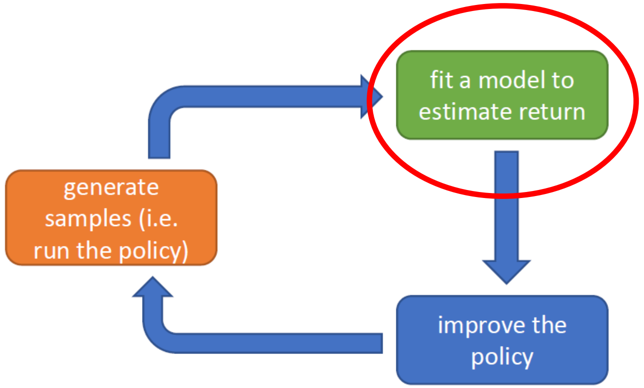
Comes from state transition model of problem (input): $p(s_1)$, $p(s_2 | s_1, a_1)$

Comes from current policy π_θ (input): $\pi_\theta(a_1 | s_1)$, $\pi_\theta(a_2 | s_2)$

$$p_\theta(s_1, a_1, s_2, a_2) = p(s_1)\,\pi_\theta(a_1 | s_1)\,p(s_2 | s_1, a_1)\,\pi_\theta(a_2 | s_2)$$

Objective of RL with Policy-based Method

- Extend trial by one more step: (s_1, a_1, s_2, a_2, s_3, a_3)
- For trial (s_1, a_1, s_2, a_2):
  - $p_\theta(s_1, a_1, s_2, a_2) = p(s_1)\ \pi_\theta(a_1 | s_1)\ p(s_2 | s_1, a_1)\ \pi_\theta(a_2 | s_2)$
- For trial (s_1, a_1, s_2, a_2, s_3, a_3):
  - $p_\theta(s_1, a_1, s_2, a_2, s_3, a_3) = p(s_1)\ \pi_\theta(a_1 | s_1)\ p(s_2 | s_1, a_1)\ \pi_\theta(a_2 | s_2)\ p(s_3 | s_2, a$
- We can continue doing this till step T (and write the resulting expression in closed form) to get:



Prob. of a trial under policy π_θ

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

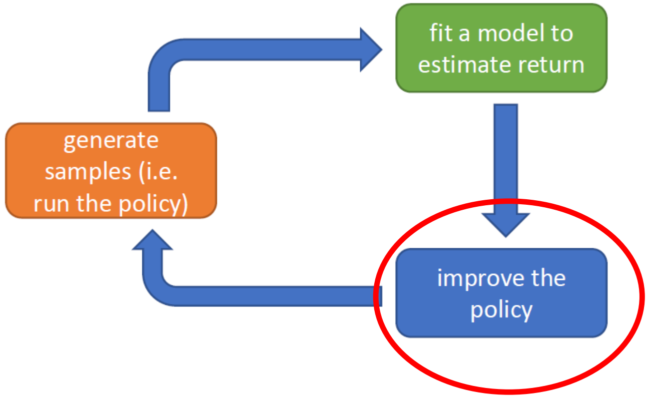
Goal of RL

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

Find policy parameter θ

that maximizes the reward from trials

averaged over trials drawn as per the trials' probabilities



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

Recall previous slide: J(θ): performance measure of the policy parameterized by θ

- Next we slightly simplify *J(θ)* so that we can calculate it from the trials or samples
- If there were N trials or samples, we could do this averaging over the N trials
  - Note: i is index for a trial, t is index for step inside a trial

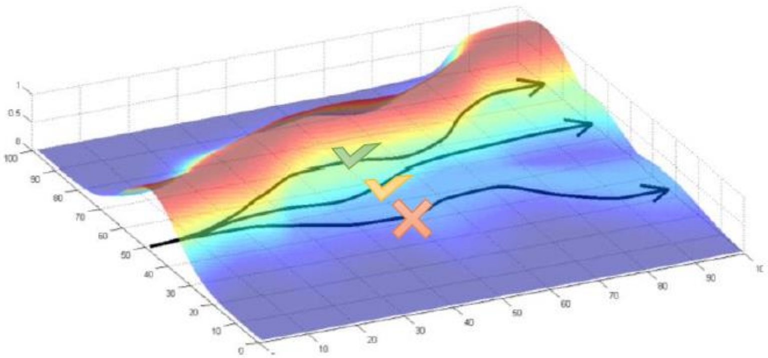
$$J(\theta) = E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i\sum_t r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

- Next, find the derivative of *J(θ)* w. r. t. *θ*:

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau)\nabla_\theta \log\pi_\theta(\tau) r(\tau) d\tau = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log\pi_\theta(\tau) r(\tau)]$$

$$\pi_\theta(\tau)\nabla_\theta \log\pi_\theta(\tau) = \pi_\theta(\tau)\frac{\nabla_\theta\pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta\pi_\theta(\tau)$$

Evaluating the policy gradient

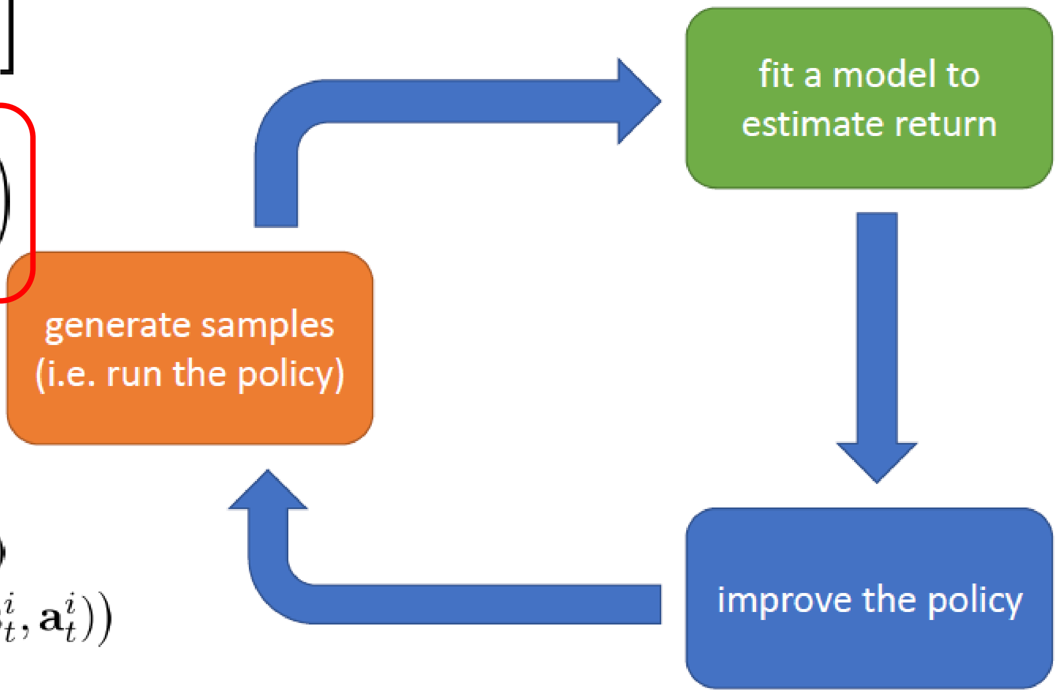


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

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

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

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$



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. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

REINFORCE Algorithm Pseudocode

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$

Note that te $\Sigma_t\, r(s_t, a_t)$ term is replaced by G_t in the last two lines of the pseudocode

Gradient formula described in previous slides

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

Actor Critic Learning

Actor-Critic Learning

- Methods that learn approximations to both policy and value functions
- Critic does value update portion
- Actor does policy update portion
- In simplest form:
  - Suppose within a trial action a_t is selected at state s_t giving next state as s_{t+1} and reward r_{t+1}
  - Critic does a value update using above values in equation: $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
  - Actor does a policy update using critic's update in equation: $p(s_t, a_t) \leftarrow p(s_t, a_t) + \beta\delta_t,$

Note this update eq. is very similar to Q-update. It is called Temporal Difference (TD) update

Similar to policy update eq.

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Variants of Actor-Critic Algorithm

- Recall that we update the policy (actor part) using gradient descent on the performance measure J(θ)
  - Derivative given by product of derivative of policy π (w. r. t. policy parameter θ)and the cumulative value term
- Different value terms given different algorithms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\, G_t\right] && \text{REINFORCE} && \text{(Recall } G_t = \Sigma_t\, r(s_t, a_t) \text{)}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; Q^w(s,a)\right] && \text{Q Actor-Critic} && \text{(uses Q-value)}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; A^w(s,a)\right] && \text{Advantage Actor-Critic} && \text{(} A^w(s, a)\text{: advantage function)}\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\; \delta\right] && \text{TD Actor-Critic} && \text{(uses TD value)}
\end{aligned}
$$

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

- Advantage actor critic
  - Single threaded (one worker): A2C
  - Multiple threads (workers) running in parallel working on different parts of input feature vector – called Asynchronous A2C or A3C

Pseudo-code of Q-Actor Critic

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Pseudo-code of TD Actor Critic

- On-policy method
- The state-value function update rule is the TD(0) update rule
- The policy function update rule is shown below.
- For n-step Actor-Critic, simply replace $G_t^{(1)}$ with $G_t^{(n)}$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha\left(G_t^{(1)} - \hat{v}(S_t,\mathbf{w})\right)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}$$
$$= \boldsymbol{\theta}_t + \alpha\left(R_{t+1} + \gamma\hat{v}(S_{t+1},\mathbf{w}) - \hat{v}(S_t,\mathbf{w})\right)\frac{\nabla_{\boldsymbol{\theta}}\pi(A_t|S_t,\boldsymbol{\theta})}{\pi(A_t|S_t,\boldsymbol{\theta})}.$$

One-step Actor-Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy weights $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
    Initialize S (first state of episode)
    $I \leftarrow 1$
    While S is not terminal:
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \beta\delta\nabla_{\mathbf{w}}\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I\delta\nabla_{\boldsymbol{\theta}}\log\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

Two Policy Gradient Algorithms (Overview)

- Probabilistic Policy Optimization (PPO)
- Trust-Region Policy Optimization (TRPO)
- Main problem addressed:
  - As trials are being done, the next state might end up in a low reward state (falling off the side of a cliff while climbing it)
  - Can be fixed by adjusting step size
- PPO and TRPO give methods to calculate the step size so that the states explored during a trial lead to improved rewards
- Non-technical overview of PPO: https://medium.com/@jonathan_hui/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

Resources

- R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, 2018. (open source pdf: http://www.incompleteideas.net/book/the-book-2nd.html)
- Online articles with github code:
  - REINFORCE: https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63
  - Actor Critic Learning (A2C): https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
- Deep RL course at UC Berkeley (videos and lecture slides) http://rail.eecs.berkeley.edu/deeprlcourse-fa17/index.html