Contents

PART I

DataStructures

CHAPTER 1

Lecture I

1.1 The Static Predecessor problem

Data structure is represented by *Set* S of items $\{x_1, .., x_n\}$
   Query : $\text{pred}(z) = \max\{\, x \in S : x \leftarrow z \,\}$
   Example Solution : $\mathcal{O}(n \log n)$ using a balanced BST

Work RAM Model

- Item are integers $\{0, 1, \ldots, 2^w - 1\}$

- "w" word size is $u = 2^w - 1$

- Pointers fit in a word

- Space $\geq n$ and $w \geq \log(\text{Space}) \geq \log(n)$

Two Data Structures

- Van Emde Boas Tree FOCS '75

    - Update/Query $\mathcal{O}(\log w)$ time
    - Update/Query $\mathcal{O}(\log u)$ space

- y Fast Tries Willard '83

Fusion Trees Fredman,Willard JCCS '93

- Query is $\mathcal{O}(\log w^n)$ time

- Can achieve $\min(\lg w, \log w^n) \leq$

3

- Dynamic Fusion Trees $\mathcal{O}(n \lg n)$ sorting

- Dynamic Fusion Trees can also achieve $\mathcal{O}(n \lg \lg n)$ sorting (Han STOC '02) and $\mathcal{O}(n \sqrt{\lg \lg n})$ FOCS '02

Van Emde Boas

- Fields of $VEB_u(V)$

  - $\sqrt{u}$ size array V.cluster[0] ... V.cluster$\sqrt{u}$
  - V.summary is a VEB $\sqrt{u}$
  - V.min and V.max are integers in {0, ..., u-1}

- Represention of values

  - X ∈ {0,1, ..., u-1}
  -

$$x = \underbrace{1011}_{c} \overbrace{0011}^{i}$$

Predecessor Algorithm $(V, x = \langle c, i \rangle)$

$$\text{if } x > V.\text{max} :$$
$$\quad \text{return } V.\text{max}$$
$$\text{else if } V.\text{cluster}[c].\text{min} < x :$$
$$\quad \text{return } \text{pred}(V.\text{cluster}[c], i)$$
$$\text{else } c' = \text{pred}(V.\text{summary}, c) :$$
$$\quad \text{return } V.\text{cluster}[c'], \text{max}$$

Insertion Algorithm $(V, x = \langle c, i \rangle)$

$$
\begin{aligned}
&\text{if } V = \varnothing : \\
&\quad V.\text{min} \leftarrow x^{prime}; \\
&\text{if } x < V.\text{min} : \\
&\quad swap(x, V.\text{min}); \\
&\text{else if } V.\text{cluster}[c].\text{min} = \varnothing : \\
&\quad insert(V.summary, c); \\
&\text{insert}(V.\text{cluster}[c], i)
\end{aligned}
$$

Algorithm analysis $(V, x = \langle c, i \rangle)$

- Prediction time

  - T (u) = T $(\sqrt{u})$ + $\mathcal{O}(1)$
  - T (u) = $\mathcal{O}(\lg \lg u)$ = $\mathcal{O}(\lg w)$

- Insertion time

  - $T(u) \leq 2T(\sqrt{u}) + \mathcal{O}(1)$
  - $T(w) \leq 2T\mathcal{O}\left(\frac{w}{2}\right) + \mathcal{O}(1)$

Space of VEB

$S(u) = (\sqrt{u} + 1)S(\sqrt{u}) + \mathcal{O}(1) \rightarrow S(u) = \theta u$
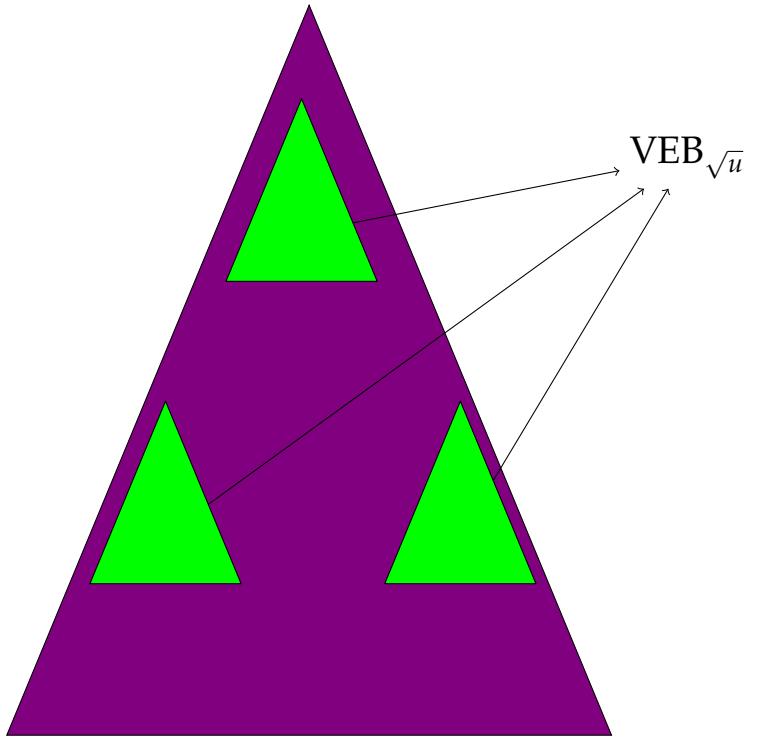
Figure 1.1: Van Ende Boas

Binary trie

Search time is $\mathcal{O}(\log n)$

- Red dashed lines are jump Pointers

- Jump pointers are used when children are missing to point to the successor
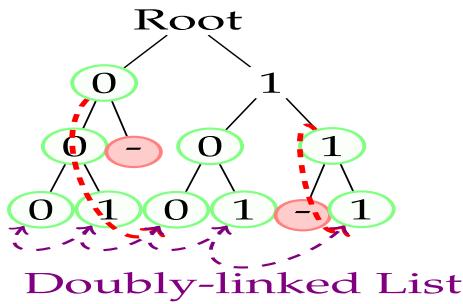
Figure 1.2: Binary Trie

Add node to Binary trie

Insertion time is $\mathcal{O}(\log n)$
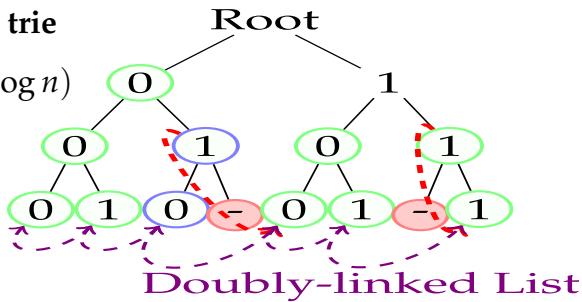
Figure 1.3: Binary Trie; Add Node

This shows an example of addition of a new node. The New node and its left child are in blue. The jump pointers are updated. And new node is added to the doubly-linked list at the appropriate position

The integer inserted above is

$$0\ 1\ 0$$

Remove node from Binary trie

Removal time is $\mathcal{O}(\log n)$

- Remove node (It will be a leaf)

- Walk back up and remove any node without a child and The jump pointers are updated. The doubly-linked list is updated.

7
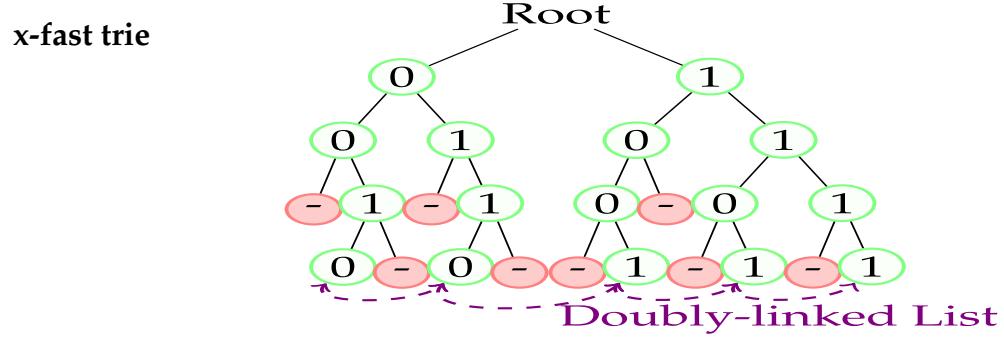
x-fast trie

Figure 1.4: xfasttrie

- An x-Fast Trie is a threaded binary trie where leaves are stored in a doubly-linked list and where all nodes in each level are stored in a hash table.

- Can do lookups in the HashTable in time $\mathcal{O}(1)$.

- Claim: Can determine successor(x) in time $\mathcal{O}(\log \log u)$.

- Start by binary searching for the longest prefix of x.

- If at a leaf node, follow the forward pointer to the successor.

- If at an internal node with a missing 1, follow the 1 thread.

- If at an internal node with a missing 0, follow the 0 thread and follow the forward pointer.

Fusion tries - Fredman, Willard JCSS '93

- Static Predecessor

- Made dynamic by Andersson, Thorup JACM ($\log_w n \log \log n$) deterministic updates

- Made dynamic by Raman ESA JACM '96 $\mathcal{O}(\log_w n)$ updates (Expected time since it is a randomized algorithm)

- Query for all $\mathcal{O}(\log_w n)$

Idea : B-Tree with branching factor $w^1/5$

- Fusion tree node stores k = $\mathcal{O}(w^{1/5})$ keys

- Item are integers $\{x_0, \ldots, x_{k-1}\}$

- $\mathcal{O}(1)$ Predecessor/Successor

- Polynomial time $k^{0(1)}$ Preprocessing
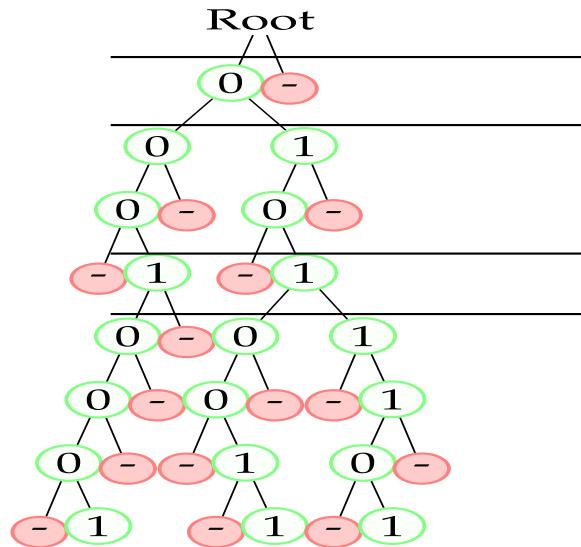
Figure 1.5: Fusion Trees Distinguishing Keys

9

The tree shown has

$$k\text{-}1$$

branching nodes and the height of the tree is
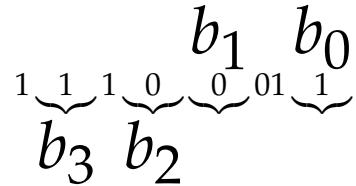
$$w$$

$\to\ \le$ k-1 levels containing a branching node. The levels starting from the top of the tree correspond to important bits the indices of which are $b_0, \ldots, b_r$ where $r < k = \mathcal{O}(w^{1/5})$ The leaves have the least significant bits.

The parallel lines show that we can check some branching nodes to find out if the lower bits can exist or not instead of traversing upto the leaf. Refer the section on Sketching.

What is sketching in this algorithm?

sketch(x) = Extract bits $\{b_0, \ldots, b_{r-1}\}$

$$1 \underbrace{1}\,1 \underbrace{0}\,\underbrace{0}^{b_1}\,01\,\underbrace{1}^{b_0}$$
$$\underbrace{}_{b_3}\,\underbrace{}_{b_2}$$

Sketching Example

- The important bits (violet circles) are used for sketching

- The labels of the leaf nodes also show these (violet) bits

The query is shown by the blue path.

$$\underbrace{0101}$$
Query

The sketch is
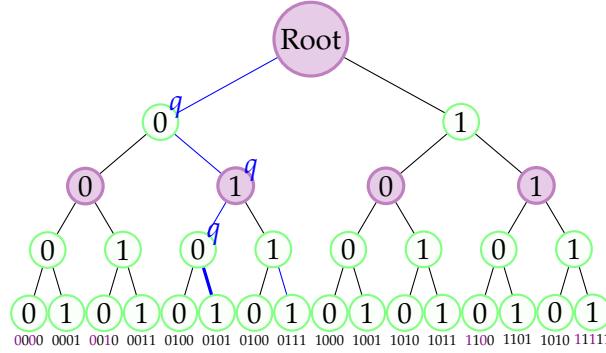
Figure 1.6: Example of Sketching

$$\underbrace{0}\ 0\ \underbrace{0}\ 0$$

$$\underbrace{0}\ 0\ \underbrace{1}\ 0$$

$$\underbrace{1}\ 1\ \underbrace{0}\ 0$$

$$\underbrace{1}\ 1\ \underbrace{1}\ 1$$

Since the sketch of the query 0101 matches the sketch 00 the answer will be the left-most node in the tree. This is wrong. It is not the predecessor 0010 or the successor 1100

We need to find a solution for this.

Let us assume sketch(x) $\leq$ sketch(g) $\leq$ sketch(x_{k+1}). We will look for the longest common prefix/ancestor of the query

$$q$$

and either x_i or x_{i+1}.

Now we can identify the wrong path we took. The correct path from the LCA(Least common Ancestor) would have lead to the left, to the tree whose nodes are shown in grey. Identify the maximum in that tree. This is e. And find sketch(e) among sketch(x_i). Now we can find the predecessor sketch(e) among sketch(x_i). Similarly we find the successor using the same algorithm.

11
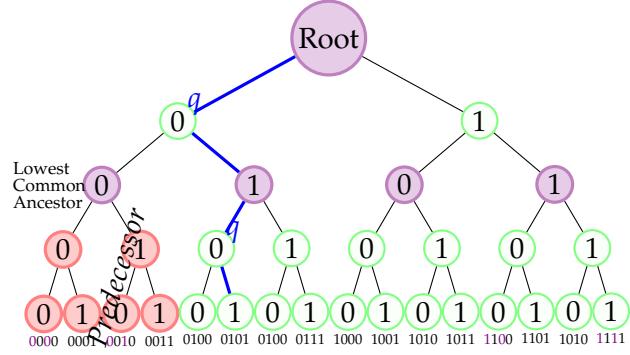
Figure 1.7: Example of Sketching

Approximate sketch(x)

We can spread the important bits in predictable pattern of $\text{len}(\mathcal{O}(w^{1/5}))$
Idea : Mask important bits

$\quad$ x′ = x and $(\sum_i^{r-1} 2^{b_i})$ and multiple x′.m = $(\sum_i^{r-1} x_{b_i} 2^{b_i})((\sum_j^{r-1} 2^{m_j}))$
= $(\sum_i^{r-1} \sum_j^{r-1} 2^{b_i + m_j}$

$\quad$ Claim : for any $b_0 \ldots b_{r-1}$ we can choose for any $m_0 \ldots m_{r-1}$ such that

1. $b_i + m_j$ are distinct $for\, all\, i, j$

2. $b_0 + m_0 < \cdots < b_{r-1} + m_{r-1}$

3. $(b_{r-1} + m_{r-1}) - (b_0 + m_0) = \mathcal{O}(r^4) = \mathcal{O}(w^{4/5})$

Proof

Choose $m'_0, \ldots, m'_{r-1} < r^3$ (Start with Stronger assumption) such that
$b_i + m'_j < \bmod r^3$
$\quad$ Property 1:
$\quad$ Pick $m'_0, \ldots, m'_{t-1}$ by induction. m'_t must avoid $m'_i + b_j + b_k \bmod r^3$,
$\forall\ \underbrace{i}_{t},\ \underbrace{j}_{r},\ \underbrace{k}_{r}$
$\leftarrow\ tr^2 < r^3$
$\quad$ Property 2:

12

$$b_{r-1} + m_r b_3 + m_3 b_2 + m_2 b_1 + m_1 b_0 + m_0$$
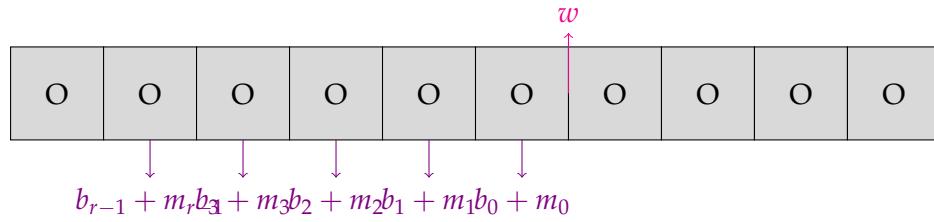
Figure 1.8: Description of Proof

Pick $m_i = m'_i + (w - b_i + i_{r^3})$ rounded down to r^3
We also need to mask with ($\sum_i^{r-1} 2^{b_i + m_i}$ and shift by $(b_0 + m_0$

CHAPTER 2

Lecture II

2.1   Order Statistics problem

Given k elements in an array, find kth smallest element (element of k rank)

Sorting will take $\mathcal{O}(n \log n)$ and we want to better than that, ideally better than linear time.

We know that that median is $\left\lceil \dfrac{k+1}{2} \right\rceil$ or $\left\lfloor \dfrac{k+1}{2} \right\rfloor$

It is tricky to find the median in linear time. In addition we also find kth smallest element.

That is the focus of this lecture.

Randomized Divide and Conquer

Rand-select $(A - p - q - \underbrace{\qquad i \qquad}_{i\text{th smallest element in } A \text{ to}})$

if $p = q$ then return A[p] :
$r \leftarrow$ Rand-partition $A - p - q$
$k \leftarrow$ r - p - 1
else if $V.\text{cluster}[c].\text{min} < x$:
  return $\text{pred}(V.\text{cluster}[c], i)$
else $c' = \text{pred}(V.\text{summary}, c)$:
  return $V.\text{cluster}[c'], \text{max}$

Rand-partition A - p - q means that we pick a value in the range p-q and use it partition the array, so that all elements less than the *random* elements are the left and the greater elements are on the right

CHAPTER 3

Lecture I/II/III

3.1 Basic Definitions

- state s_t, is the state of the world at time t

- action a_t, the decision taken at time t

- trajectory τ, sequence of states/obserations and actions $D := (s_1, a_1, \ldots, s_T)$

- reward function $r(s, a)$

- Policy $\pi(a, s)$ or $\pi(a, o)$

$$\min_{\theta} \mathbb{E}_{\tau \sim P_\cdot(\varnothing)} \left[\sum_t^T r(s_t, a_t) \right]$$

3.2 Imitation Learning

What is the goal of imitation learning ?

- Data: Given trajectories collected by an expert "demonstrations" $D := (s_1, a_1, \ldots, s_T)$

- Sampled from some unknown policy π_{expert}

- Goal: Learn a policy π_θ that performs at the level of π_{expert} by mimicking

Imitation learning -Version 0

Deterministic policy

- Data: Given trajectories collected by an expert "demonstrations"$D := (s_1, a_1, \ldots, s_T)$

- Supervised regression to the expert's policy $\min_\theta \frac{1}{\|D\|}\sum_{(s,a)\in D}\|a - \hat{a}\|_2^2$

- Deploy learned policy π_{theta}

Evaluation of the RL objective

$$\theta^* = \text{argmax}_\theta \underbrace{\mathbb{E}_{\tau\sim P_\cdot(\emptyset)}\left[\sum_t^T r(s_t, a_t)\right]}_{J(\theta)}$$

$$\nabla_\theta J(\theta) = \nabla_\theta \int P_\theta(\tau)r(\tau)d\tau = \int \nabla_\theta P_\theta(\tau)r(\tau)d\tau$$

This gradient can be simplified using this trick.

$$= P_\theta(\tau)\nabla_\theta \log P_\theta(\tau)$$

$$= P_\theta(\tau)\frac{\nabla_\theta P_\theta(\tau)}{P_\theta(\tau)}$$

We apply the chain rule after evaluating the derivation of $\log P_\theta(\tau)$ which is $\frac{1}{P_\theta(\tau)}$

This means that $\int \nabla_\theta P_\theta(\tau)r(\tau)d\tau$ is $= \int \nabla_\theta \log P_\theta(\tau)r(\tau)d\tau$

We know that $P_\theta(\tau) = P(s_t)\Pi_t\pi_\theta(a_t, s_t)P(s_{t+1} \mid (s_t, a_t))$

So $\log P_\theta(\tau) = \log P(s_t) + \sum_t \log \pi_\theta(a_t, s_t) + \log P(s_{t+1} \mid (s_t, a_t))$

$\log P(s_t)$ and $\log P(s_{t+1} \mid (s_t, a_t))$ are constants.

$\nabla \log P_\theta(\tau) = \nabla_\theta \sum_t \log \pi_\theta(a_t, s_T)$

17

> Expectation over trajectories sampled by our policy is
> $\mathbb{E}_{\tau \sim P_\theta(\tau)}$
>
> So the policy gradient now is $\nabla \theta J(\theta) =$
> $\mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\sum_t^T \nabla_\theta \log \pi(a_t, s_t) \right] \left[\sum_t^T r(s_t, a_t) \right]$

The equation shown above is the gradient of the policy of the RL objective.

Estimating the Gradient

This is the full algorithm.

1. Sample r_i from $\pi_\theta(a_t \mid s_t)$

2. $\nabla_\theta J(\theta)$ from $\pi_\theta(a_t \mid s_t) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(a_t^i s_t^i) \right) \left(\sum_t (a_t^i s_t^i) \right)$

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

1. The equation in the first box is for imitation gradient, weighted by reward

2. The 2nd equation is for imitation learning

$$\nabla_\theta J(\theta) \approx \tfrac{1}{N} \sum_i^N \left(\sum_i^T \nabla_\theta \log \pi_\theta(a_{i,t} s_{i,t}) \right) \left(\sum_i^T r(s_{i,t}, a_{i,t}) \right)$$

$$\nabla_\theta J_{BC}(\theta) \approx \tfrac{1}{N} \sum_i^N \left(\sum_i^T \nabla_\theta \log \pi_\theta(a_{i,t} s_{i,t}) \right)$$

Figure 3.1: Imitation gradient and learning

This increases the likelihood of action when the rewards are high and decreases the likelihood of actions in trajectories with low reward.

Off-policy version of Policy Gradient

Importance Sampling - Using proposal distribution q, we get $\mathbb{E}_{x \sim P(x)}[f(x)] = \int P(x)f(x)dx = \int \frac{q(x)}{q(x)}P(x)f(x)dx = \int q(x)\frac{p(x)}{q(x)}f(x)dx = \mathbb{E}_{x \sim q(x)}\left[\frac{p(x)}{q(x)}\right]f(x)$ Please note: Important for q to have non-zero support for high probability P(X)

Figure 3.2: Off-policy Gradient

q(x) is considered the old policy.$\mathbb{E}_{x \sim q(x)}\left[\frac{p(x)}{q(x)}\right]f(x)$ and we are sampling from the old policy and weighting by dividing the new policy by the old policy.

What if we want to use samples from $\overline{P}(\tau)$?

$J(\theta) = \mathbb{E}_{\tau \sim \overline{P}_\theta(\tau)}\left[\frac{P_\theta(\tau)}{\overline{P}(\tau)}\right]r(\tau)$

Some of the probabilities cancel out.

$$\frac{P_\theta(\tau)}{\overline{P}(\tau)} = \frac{\cancel{P(s_1)}\;\Pi_{t=1}^{T}\pi_\theta(a_t,s_t)\;\cancel{P(s_{t+1}|(s_t,a_t))}}{\cancel{P(s_1)}\;\Pi_{t=1}^{T}\overline{\pi}_\theta(a_t,s_t)\;\cancel{P(s_{t+1}|(s_t,a_t))}}$$

Common final form $\nabla_{\theta'}J(\theta') \approx$
$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_{\theta'}(a_{i,t}|s_{i,t})}{i\pi_\theta(a_{i,t}|s_{i,t})}\nabla_{\theta'}\log\pi_{\theta'}(a_{i,t}\quad|$
$s_{i,t})\left(\left(\sum_{t'=t}^{T}r(s_{i,t'},a_{i,t'})\right) - b\right)$

Figure 3.3: Off-policy Gradient

CHAPTER 4

Lecture IV

4.1 Actor Critic methods

Value function

This is the future rewards starting at s and following policy π

$$V^\pi(s_t) = \sum_{t'=t}^T \mathbb{E}_{s_{t'}, a_{t'} \sim \pi} \left[r(s_{t'}, a_{t'}) \mid s_t \right]$$

Q-function $\mathbb{Q}^\pi$

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^T \mathbb{E}_\pi \left[r(s_t, a_t) \mid s_t, a_t \right]$$

Improving Policy Gradients

The gradient would be better if we increased the likelihood when Q is high and decreased the likelihood when Q is low

$$\nabla_\theta J(\theta) \sim \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_\theta \log \pi\theta(a_{i,t} s_{s,i}) Q(s_t^i a_t^i)$$

The explanation shown below is about this.

Introducing baselines

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\nabla_\theta \log P\theta(\tau) r(\tau) - \underbrace{b}_{} \right]$$

subtracting a baseline "constant"

which is the average reward given by

$$b \overset{!}{=} \frac{1}{N} \sum_{i=1}^{N} r(\tau)$$

If we subtract average reward we get negatie-gradients for below-average behaviour.

The following is the proof that the subtracted value doesn't have any effect on the gradient

$\mathbb{E}\left[\nabla_\theta \log \pi\theta(\tau) b\right] = \int p_\theta(\tau) \nabla_\theta \log \pi\theta(\tau) b d\tau$ Using this trick

$$P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) = \nabla_\theta P_\theta(\tau)$$

we get $= b \nabla_\theta \underbrace{\int P_\theta(\tau) d\tau}_{\text{This is 1}}$

Gradient is zero

21
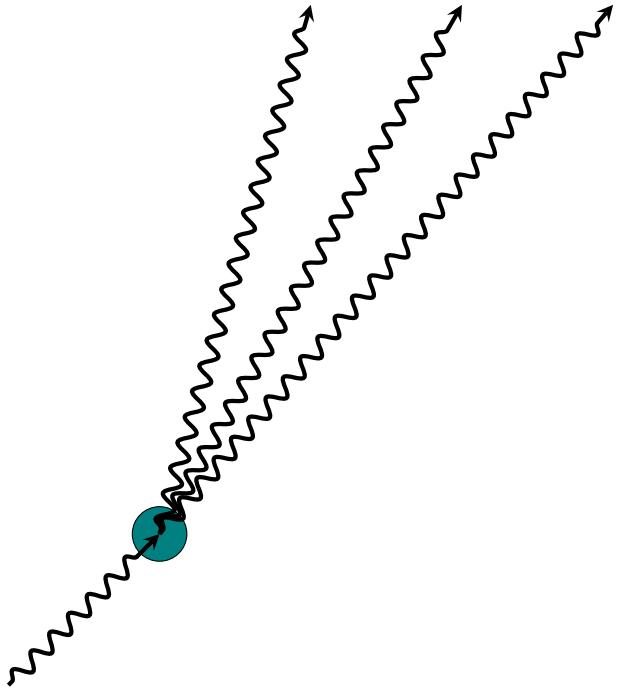
Figure 4.1: Improving policy Gradients

This will ameliorate the situation caused when the robot takes a small step forward and fall backwards.

Estimating future rewards using $\mathbb{Q}$ function

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi\theta(a_{i,t} \mid s_{i,t}) \boxed{r(s_{i,t'}, a_{i,t'})}$$

$$\underbrace{\phantom{r(s_{i,t'}, a_{i,t'})}}$$

Extimate future rewards if we take a_t at s_t

True future rewards can be obtained by using the $\mathbb{Q}$ function like this.

We will get much better results if the form of the gradient is like this.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi\theta(a_{i,t} \mid s_{i,t}) \mathbb{Q}(a_{i,t}, s_{i,t})$$

What about baselines ?

Once we start using the $\mathbb{Q}$ function we get this when we subtract the baseline.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi\theta(a_{i,t} \mid s_{i,t})(\mathbb{Q}(a_{i,t}, s_{i,t}) - \underbrace{b}_{\mathbb{V}^\pi(s_t)})$$

$$b = \text{ average reward } = \frac{1}{N} \sum_i \mathbb{Q}(s_{i,t}, a_{i,t})$$

$$V(s_t) = \mathbb{E}_{a_t \sim \pi_\theta(\bullet | s_t)} \left[\mathbb{Q}(s_t, a_t) \right]$$

Recall : $\mathbb{A}^\pi(a_t, s_t) = \mathbb{Q}^\pi(a_t, s_t) - \mathbb{V}^\pi(s_t)$
So with baseline we get
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi\theta(a_{i,t} \mid s_{i,t}) \mathbb{A}^\pi(a_t, s_t)$$

4.2   Offline RL outline

Better estimates of $\mathbb{A}$ will lead to less noisy gradients.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi\theta(a_{i,t} \mid s_{i,t}) \boxed{\mathbb{A}^\pi(a_t, s_t)}$$

First initialize the policy (randomly with imitation learning and heuristics)

4.3   Estimated expected return

Should we fit $\mathbb{V}^\pi, \mathbb{Q}^\pi, \mathbb{A}^\pi$?. We are going to fit $\mathbb{V}^\pi$.

Recall : $\mathbb{A}^\pi(s_t, a_t) = \mathbb{Q}^\pi(s_t, a_t) - \mathbb{V}^\pi(s_t)$. There is a way to turn this equation that only depends on $\mathbb{V}^\pi$. $\mathbb{Q}^\pi(a_t, s_t)$ starts at state s, takes action a and sums the rewards. So we can write the equation like this.

$$\underbrace{Q^\pi(s_t, a_t)}$$

Rewards at the current time step and the value starting from s_{t+1}

Run policy to collect batch of data

Fit model to estimate expected return

Improve policy
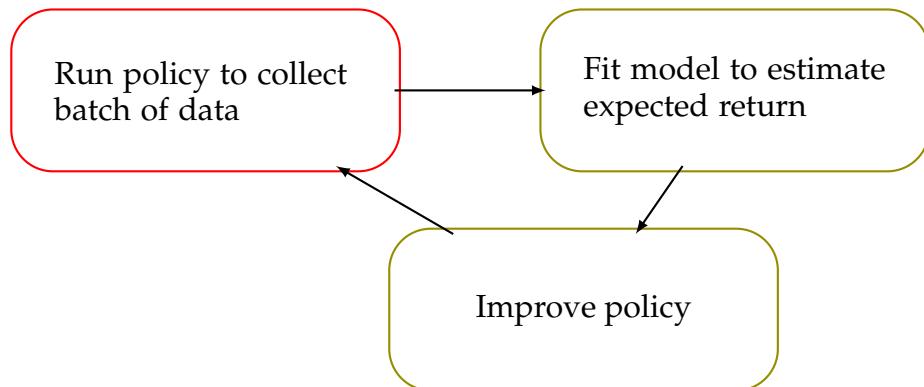
Figure 4.2: Online RL

So we can rewrite the equations like this.

$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta}\left[r(s_{t'}, a_{t'}) \mid (s_t, a_t)\right]$$

$$r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\bullet|(s_t,a_t))}\left[\mathbb{V}^\pi_{s_{t+1}}\right]$$

Use the sampled s_{t+1}

$$\mathbb{A}^\pi(s_t, a_t) = r(s_t, a_t) + \mathbb{V}^\pi(s_{t+1}) - \mathbb{V}^\pi(s_t)$$

Estimating $\mathbb{V}^\pi$

Version 1 : Monte carlo estimation

1. Original single sample estimate $\mathbb{V}^\pi(s_t) = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$

2. Multisample estimate (but can't reset the world in practice) $\mathbb{V}^\pi(s_t) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{t'=t}^{T} r(s_t$

Step 1 : Aggregate dataset of single sample estimates

$$\left(s_{i,t}, \underbrace{\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})}_{y_{i,t}} \right)$$

Step 2 : $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(s_i) - y_i \right\|_2^2 \; Laplace transform$

Version 2 : Bootstrapping

Ideal target is the true sum of rewards. This is equal to the reward at the current time step and the expected reward at the next time step.

$$\text{Monte carlo target} : y_{i,t} = \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$$

Ideal target : $\sum_{t'=t}^{T} \mathbb{E}_{\pi_\theta} \, r(s_{i,t'}, a_{i,t'}) \mid (s_{i,t}) \approx r(s_{i,t}, a_{i,t})$
$+ \sum_{t'=t+1}^{T} \mathbb{E}_{\pi_\theta} r(s_{t'}, a_{t'}) \mid (s_{i,t+!}) \; r(s_{i,t}, a_{i,t}) \mathbb{V}^\pi(s_{i,t+1}) \approx$
$r(s_{i,t}, a_{i,t}) \quad \underbrace{\hat{\mathbb{V}}_\phi^\pi(s_{i,t+1})}_{\text{Fitted value function}}$