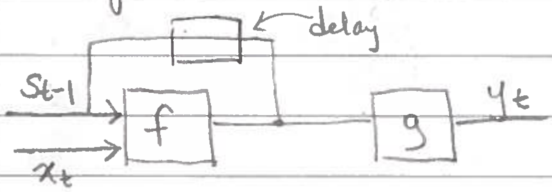
L8-1

Sequential models or
"There's more to the world than pure functions."

State machines describe a computational process
- S_0 : initial state
- f : state transition function
- g : output function
- $\mathcal{S}$: set of possible states
- $\mathcal{X}$: set of possible inputs
- $\mathcal{Y}$: set of outputs

} May not be strictly necessary to define



Start with s_0, then:

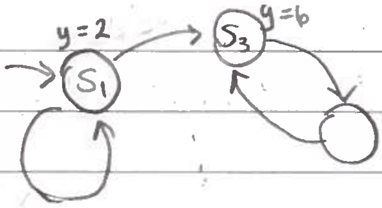
$$s_t = f(x_t, s_{t-1})$$
$$y_t = g(s_t)$$

So, given a sequence of inputs: $x_1, x_2, \dots$
we generate a sequence of outputs:

"transducer" $\underbrace{g(f(x_1, s_0))}_{y_1}, \underbrace{g(f(x_2, f(x_1, s_0)))}_{y_2}, \dots$

Output at time t can have a dependence on inputs from steps 1 to t.

Two common forms of state machine:
finite state machine: $\mathcal{S}, \mathcal{X}, \mathcal{Y}$ all finite sets
often drawn like this:



nodes are states
- arcs labeled by inputs
- states labeled by output

many variations

LTI (linear time-invariant systems) ← discrete time

- defined by a linear difference equation, like
  $$y[t] = 3 \cdot y[t-1] + 6 \cdot y[t-2] + 5x[t] + 3x[t-2]$$
- can be implemented using state to store relevant previous inputs/outputs

We will study recurrent neural networks, which are a kind of state machine.

$\mathcal{X} = \mathbb{R}^l$ $\quad W^{sx} : m \times l$

$\mathcal{S} = \mathbb{R}^m$ $\quad W^{ss} : m \times m \quad W_0^{ss} : m \times 1$

$\mathcal{Y} = \mathbb{R}^n$ $\quad W^{o} : n \times m \quad W_0^{o} : n \times 1$

$$f(s,x) = f_1(W^{sx}x + W^{ss}s + W_0^{ss})$$

$$g(s) = f_2(W^{o}s + W_0^{o})$$

f_1, f_2: activation functions

Markov Decision Processes (MDPs)

- state transition function is stochastic (probabilistic)
- no separate output function (state is observable)
- some states/actions are more desirable than others
- can be used to model interaction with an outside "world" like some single-player games (direct extension to two-player zero-sum games)
- we will focus on discrete, finite $\mathcal{S}$ and $\mathcal{X}$
- typical to call $\mathcal{X}$, instead $\mathcal{A}$, for action
- try to pick actions to drive the world into high-scoring states.

Formally, an MDP is $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ ← optional

T: transition model $\quad T(s,a,s') = P(S_t = s' \mid S_{t-1} = s, A_{t-1} = a)$

conditional prob. distribution $\quad$ ↑ sums to 1 over s'

R: ~~reward function~~ $\cancel{R(s,a) \in \mathbb{R}}$

A policy $\pi : S \to A$ specifies what action to take in each state.

Policy evaluation: given an MDP, a policy, and a horizon $\equiv$ number of actions we can take

What is the expected sum of rewards we will get?

for all s $\quad V_{\pi}^{\overset{\text{horizon}}{0}}(s) = 0$ $\qquad$ With 0 steps to go, we can't earn any reward ⍨

$\boxed{V \equiv \text{value}}$

$$V_{\pi}^{1}(s) = R(s, \underbrace{\pi(s)}_{\text{action}}) + 0$$

$$V_{\pi}^{2}(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') \cdot R(s', \pi(s'))$$

$$V_{\pi}^{h}(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') V_{\pi}^{h-1}(s')$$

How to find the best policy?

Could enumerate all possible policies and pick the best one!

Note that, in finite horizon problems, the best choice of action depends on how many steps you have left!

Define $Q^{h}(s, a) \equiv$ expected value of
- starting in state s
- executing action a
- continuing for $h-1$ more steps executing the optimal policies

dynamic programming

(1) $\quad Q^{0}(s, a) = 0$ $\qquad$ no steps, no reward!

$$Q^{1}(s, a) = R(s, a) + 0$$

$$Q^{2}(s, a) = R(s, a) + \sum_{s'} T(s, a, s') \max_{a'} R(s', a')$$

(h) $$Q^{h}(s, a) = R(s, a) + \sum_{s'} T(s, a, s') \max_{a'} Q^{h-1}(s', a')$$

Cases 1 and h define simple recursive algorithm $\boxed{\text{value iteration}}$

Given Q, optimal policy is easy to find:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\ Q(s,a)$$

Infinite horizon: when you don't know when the game will be over!

$0 < \gamma < 1$: discount factor

$\boxed{\text{probability you'll live to play the next step}}$

Instead of maximizing expected finite horizon undiscounted value:

$$E\left[\sum_{t=0}^{h} r_t \,\middle|\, \pi, s_0\right]$$

maximize expected infinite horizon discounted value

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, \pi, s_0\right]$$

$$E\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \,\middle|\, \pi, s_0\right] \equiv V_\pi(s_0)$$

$$= E\left[r_0 + \gamma(r_1 + \gamma(r_2 + \gamma \ldots)) \,\middle|\, \pi, s_0\right]$$

policy evaluation

for all s: $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi(s')$

You could write down one of these equations for each of $n = |S|$ states. There are n unknowns ($V_\pi(s)$). Equations are linear. So, easy to solve for the values.

optimal policy

The best way of behaving in an infinite-horizon discounted MDP is not time dependent: at every step, your expected future lifetime, given that you have survived until now, is $1/(1-\gamma)$.

Thm: there exists a stationary optimal policy π^* such that, for all s, and all other policies π,

May be more than one.

$$V_{\pi^*}(s) \geq V_\pi(s)$$

$n = |S| \quad m = |A|$

Algorithms for finding π^*

- enumerate and test $\quad O(m^n)$
- linear programming $\quad O(\text{poly}(n, m, b))$ — # bits per element of T, R
- policy iteration $\quad \sim O(\text{poly}(n, m, \text{bits}(\gamma)))$
  requires solving lots of $n \times n$ systems.
- value iteration $\quad O(\text{poly}(n, m, b, 1/1-\gamma))$

easy to implement, foundation for many reinforcement-learning methods

$Q(s,a)^* \equiv$ expected infinite horizon discounted value of being in state s, executing action a, executing π^* thereafter

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^*(s',a')$$

Theorem: these equations have a unique solution.

But not linear so not easy to solve.

Can derive π^*:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

Value iteration algorithm $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \varepsilon)$

$Q_{old}(s, a) = 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$

loop:

for $s \in \mathcal{S}$, for $a \in \mathcal{A}$:

$$Q_{new}(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q_{old}(s', a')$$

if $\max_{s,a} \| Q_{old}(s, a) - Q_{new}(s, a) \| < \varepsilon$:

return Q_{new}

$Q_{old} := Q_{new}$ (be sure to copy!)

Theory

Define $\pi_Q(s) = \arg\max_a Q(s, a)$

- $|V_{\pi_{Q_{new}}} - V_{\pi^*}|_{max} < \varepsilon$

- There is a value of ε such that

  $|Q_{old} - Q_{new}|_{max} < \varepsilon \Rightarrow \pi_{Q_{new}} = \pi^*$

- $|V_{\pi_{Q_{new}}} - V_{\pi^*}|_{max}$ decreases monotonically

Robustness

Can be executed asynchronously, in parallel: as long as all (s, a) pairs are updated infinitely often in an infinite run, it still converges to optimal values.