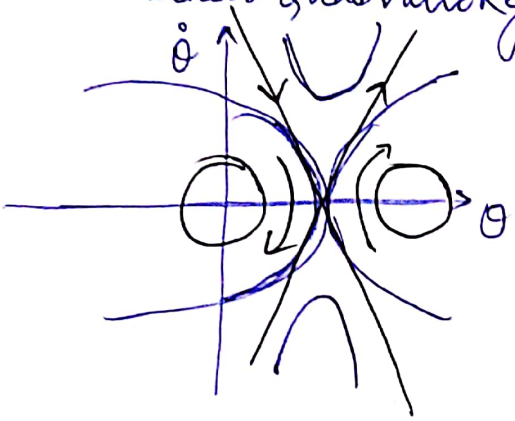
Lecture - 6 (Dynamic Programming - III)

(Using Function Approximators like NNs)

Recap

- Tabular (Discrete State, action, time) → Computationally expensive (High dimensionality issue)
  optimization over graph.
- Linear Quadratic Regulator (LQR) (Continuous State, action, Continuous time or discrete time)



LQR works well around the vicinity of some fixed point. But, it doesn't solve the problem globally. Linearization isn't valid across enough of the state space for that to be good.

Value Iteration w/ (with) Function Approximators (eg. NNs)

$\hat{J}(x)$ → approximate cost-to-go

$$\begin{bmatrix} \hat{J}(s_0) \\ \hat{J}(s_1) \\ \hat{J}(s_2) \\ \vdots \end{bmatrix}$$

Fully discrete case

$\hat{J}(x) = x^T S x$ (quadratic function) (LQR case)

In General

$\hat{J}_\alpha(x)$ → parametrized by vector α (α would be the weights & biases of the NNs (parameters))

Fully discrete

$$\hat{J}(s_i) \leftarrow \min_a \left[l(s,a) + \hat{J}(f(s_i,a)) \right]$$

to be the current estimate of $\hat{J}$ and compute for all of the points and treat it as desired target

Function Approximator

At sample x_i,

$$\hat{J}_\alpha(x_i) \approx \min_a \left[l(x_i,a) + \hat{J}_\alpha(f(x_i,a)) \right]$$

(Minimize Cost function using Gradient descent)

$J^d(x_i)$

Minimize over parameter α (desired)

Sample points x_i

$$\min_\alpha \sum_i \left[J_\alpha(x_i) - J^d(x_i) \right]^2$$

System dynamics

Applying Gradient descent on above loss

$$\Delta\alpha = \eta \left[J_\alpha(x_i) - J^d(x_i) \right] \frac{\partial \hat{J}}{\partial \alpha}$$

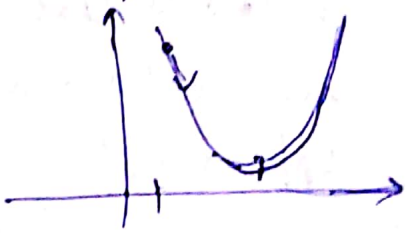
$$\alpha = \alpha + \Delta\alpha$$

When can we expect a Gradient Descent algorithm on a function approximator to actually converge to the true cost-to-go?

→ Fitted Value Iteration Algorithm

$$\min_x \|Ax-b\|^2$$

$$\frac{\partial R}{\partial x} = A^T(Ax-b) = 0$$

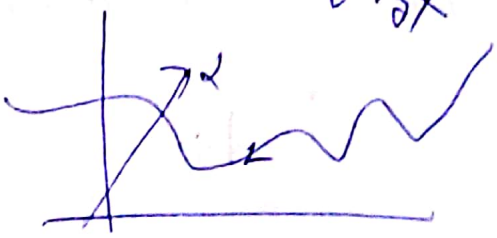


$$\hat{x}[n+1] = \hat{x} - \eta \frac{\partial Loss}{\partial x}$$

Really important special case –

Linear function approximators

$$\hat{J}_\alpha(x) = \underbrace{\alpha^T}_{\text{parameter}} \underbrace{\phi(\alpha)}_{\text{basis func}} = \sum_i \alpha_i \phi_i(x)$$



If the parameters only enter linearly even if the basis functions are non-linear.

Radial basis functions

Barycentric interpolations

Deep neural networks — in the limit of ultra wide networks (Neural Tangent Kernel)

→ Maybe the reason that big neural networks work even though they are non-linear in their parameters because they are so big and so wide that they start looking linear in the parameters again.

→ Explicitly, if we write a Multilayer perceptron, it is non linear in the parameters but the theoretical models that look at limits of arbitrarily big networks start to look like linear function approximators.

→ Some people say that's why RL works when it works.

Fitted Value Iteration Converges. ← proof

Tsitsiklis + Van Roy '97

LQR

$\hat{J} = x^T \hat{S} x$ is a linear function approximator.

- scalar
- linear in the parameters of $\hat{S}$ but quadratic in x

$= tr(x^T \hat{S} x) = tr(S x x^T)$ → property of trace (cyclic the elements)

$\phi(x)$

quadratic in x & coefficients on S

Algorithm

Given $\hat{S}$, $\langle x_i \rangle$, $\langle u_i \rangle$ → discrete u's

(Applied on linear dynamics)

① compute → $x_{i,j}^{next} = f(x_i, u_j) = Ax_i + Bu_j$

next state; (state, action) pair

② $J_{i,j}^{next} = (x_i^{next})^T \hat{S} (x_i^{next})$

(for every state, action pair)

③ $J_i^d = \min_j \left[L(x_i, u_j) + J_{i,j}^{next} \right]$

④ Loss(L) $= \sum_i \left[x_i^T \hat{S} x_i - J_i^d \right]^2$

⑤ $\Delta \hat{S} = -\eta \frac{\partial L}{\partial S}$ (gradient of loss (a scalar) w.r.t. a matrix (S))

Loop

Is the infinite horizon cost finite?

→ What we really need is at least one the $J_{i,j}^{next}$ to be 0. in order for the algorithm to eventually terminate

If it can't get directly to the origin, it's just going to be incurring cost forever and everytime we run the algorithm, it's going to add cost to the cost-to-go & eventually blow up.

→ for every X if we could find a u that would get us directly to the origin but if we discretize u, that's not gonna work.

Alternative version

Continuous u's

Given $\hat{S}$, $\{x_i\}$ = bunch of x_i's, all u available

$$J_i^d = \min_u \left[x_i^T Q x_i + u^T R u + (Ax_i + Bu)^T \hat{S} (Ax_i + Bu) \right]$$

Even when we have the wrong $\hat{S}$, we can still find the minimizing u exactly.

$$u_i^* = -R^{-1} B^T \hat{S} x_i$$

optimal u

In the particular case of a linear function approximator when an infinite horizon solution exists, above linear function approximator is guaranteed to converge.

Note – In the first algorithm, we expected the LQR – that it would work. It didn't quite work because the sampled version of LQR can't obtain zero finite cost but for the particular case of LQR

(Algorithm) we can do better and if we learn the samples but only change the actions to continuous actions (from discrete) then it resolves the problem & it converges.

Discounted version

$$= \left[X[0]^T Q X[0] + u^T[0] R u[0] + J(0)\right]$$

$$\min_{u[\,]} \sum_{n=0}^{\infty} \gamma^n \left(x[n]^T Q_x[n] + u^T[n] R u[n]\right)$$

Soln to this can be written as $x^T S x$ (unrolled version)

$0 < \gamma \leq 1$

If γ becomes small, that means we are not worried about future, but only worried about today.

discount factor

even if this doesn't go to 0, thus that sum will converge to a finite no.

LQR solution: DARE (A, B, Q, R) ⟸ soln to the undiscounted discrete-time LQR problem

(Algebraic Riccatic eqn; discrete)

DARE $(\sqrt{\gamma}A, B, Q, \frac{1}{\gamma}R)$ ⟸ soln to discounted LQR.

→ Discounted LQR has a closed form solution and the closed form solution is equivalent to solving the original LQR problem but with a smaller A ($\sqrt{\gamma}A$) which makes it more stable and is also modulating the cost ($\frac{1}{\gamma}R$).

→ So, Discounted LQR is like assuming A is more stable than it actually is.

Neural Fitted value iteration

$\hat{J}_2(x)$ as a neural network

Multi Layer Perceptron (ReLU - for activation)
3 layers ~ 255 hidden units per layer.

Discrete time:

$$\min_u \left[l_2(x) + u^T R u + \hat{J}^0\left(\left(f(x,u)\right)\right) \right]$$

result of f is inside our NNs (func approximator)

$\hat{J}^0$: function approximator

Cont. time:

$$J^d(x_i) \leftarrow \min_u \left[l(x,u) + \frac{\partial \hat{J}^0}{\partial x} f(x,u) \right]$$

$$l_2(x) + u^T R u + \frac{\partial \hat{J}}{\partial x}\left[f_1(x) + f_2(x)u\right]$$

In Cont. time, at least we don't have to run through the NNs (function approx.), we just have to go through f.
We only have to depend on the partial derivative (slope of the cost-to-go function) not the value of the cost-to-go func. in some of the future place. That's the benefit of being infinitesimal.

$f_1(x) + f_2(x)u$ (a lot of robotic can about are control Affine).

there have a close form sol^n for u

Summary

3 Approaches to Value-Iteration

(1) On a Grid

→ everything works — it's beautiful, ~~it's~~ fast

→ But, grid gets really really big in higher state spaces.

(2) LQR

→ solves to arbitrary dimension basically but is restricted to the linearization of the system.

(3) NN

→ don't have the same ~~guarantees~~ guarantees but they can often be made to work. We have to make decisions about discount factor, initialize weights choice of cost, etc.