Deep Reinforcement Learning

(RL with function approximation)

Learning to make decisions ← DNN

Agents learn → policy: state → action
→ value: state → value
→ model: state → state

functions

Agents ↓ decisions affect → reward
→ agent state
→ environment state

Value-based (learn value function + implicit policy)

Learning Value functions

Go: 10^{170} states

lookup tables

基于值的算法会评估每个(s,a)元组的Q值 Q(s,a)，再根据Q值求最优策略.

state s : $V(s)$

state-action s, a : $q(s,a)$

$$\boxed{\begin{aligned} V_\theta(s) &\approx V_\pi(s) \\ q_\theta(s,a) &\approx q_\pi(s,a) \end{aligned}}$$

update θ using MC or TD learning

if the environment state is not fully observable:

→ use agent state

→ Learn a state update function

$$S_{t+1} = u(S_t, O_{t+1})$$

Function Approximator:

1. ANN (DNN)
2. Decision Tree
3. Nearest Neighbor
4. Fourier / wavelet bases
5. Coarse coding

RL Notice:

1. experience is not iid
2. update non-stationary
3. feedback delayed

Classes of Function Approximation

Tabular

State Aggregation

Linear function approximation

Stochastic Gradient Descent for Value function Approximation

$$J(w) = \mathbb{E}_{\pi}\left[\left(V_{\pi}(s) - \hat{V}(s,w)\right)^2\right]$$

$$\Delta w = -\frac{1}{2}\alpha \nabla_w J(w)$$

$$= \alpha \mathbb{E}_{\pi}\left[(V_{\pi}(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w) \right]$$

Sample:

$$\Delta w = \alpha (V_{\pi}(s) - \hat{v}(s,w)) \nabla_w \hat{v}(s,w)$$

Feature Vector

Represent state by a feature vector

$$\phi(s) = \begin{pmatrix} \phi_1(s) \\ \phi_2(s) \\ \vdots \\ \phi_n(s) \end{pmatrix}$$

$$\phi : S \to R^n$$

$$\phi_t = \phi(S_t)$$

Linear Approximation

$$V_\theta(s) = \theta^T \phi(s) = \sum_{j=1}^{n} \phi_j(s)\,\theta_j$$

$$J(\theta) = \mathbb{E}_\pi\left[\left(V_\pi(s) - \theta^T \phi(s) \right)^2 \right]$$

$$\nabla_\theta V_\theta(S_t) = \phi(S_t) = \phi_t$$

$$\longrightarrow \quad \Delta\theta = \alpha \left(V_\pi(S_t) - V_\theta(S_t) \right) \phi_t$$

update = step size × prediction error × feature value

Table lookup features

Incremental Prediction Algorithms

MC:

$$\Delta\theta_t = \alpha \left(G_t - V_\theta(s) \right) \nabla_\theta V_\theta(s)$$

Linear TD(0):

$$\Delta\theta_t = \alpha \underbrace{\left(\boxed{R_{t+1} + \gamma V_\theta(S_{t+1})} - V_\theta(S_t) \right)}_{\delta_t \ \text{"TD error"}} \nabla_\theta V_\theta(S_t)$$

TD(λ):

$$\Delta\theta_t = \alpha \left(G_t^\lambda - V_\theta(s) \right) \nabla_\theta V_\theta(S_t)$$

Linear MC:

$$\Delta\theta_t = \alpha \left(G_t - V_\theta(S_t) \right) \nabla_\theta V_\theta(S_t)$$

$$= \alpha \left(G_t - V_\theta(S_t) \right) \phi_t$$

Convergence of MC

$$\min_{\theta} \mathbb{E}[(G_t - V_\theta(S_t))^2]$$

$$= \mathbb{E}[\phi_t \phi_t^T]^{-1} \mathbb{E}[V_\pi(S_t)\phi_t]$$

Proof:

$$\nabla_\theta \mathbb{E}[(G_t - V_\theta(S_t))^2]$$

$$= \mathbb{E}[(G_t - V_\theta(S_t))\phi_t] = 0$$

$$\mathbb{E}[(G_t - \phi_t^T\theta)\phi_t] = 0$$

$$\mathbb{E}[G_t\phi_t - \phi_t\phi_t^T\theta] = 0$$

$$\mathbb{E}[\phi_t\phi_t^T]\theta = \mathbb{E}[G_t\phi_t]$$

$$\theta = \mathbb{E}[\phi_t\phi_t^T]^{-1}\mathbb{E}[V_\pi(S_t)\phi_t]$$

Convergence of TD

$$\min_{\theta} \mathbb{E}\left[\left(R_{t+1} + \gamma V_\theta(S_{t+1}) - V_\theta(S_t)\right)^2\right]$$

$$= \mathbb{E}\left[\phi_t\left(\phi_t - \gamma \phi_{t+1}\right)^T\right]^{-1} \mathbb{E}\left[R_{t+1}\phi_t\right]$$

TD method : faster , better

policy evaluation : prefer MC

Residual Bellman updates

TD:

$$\Delta\theta_t = \alpha \delta \nabla_\theta V_\theta(S_t)$$

$$\delta_t = R_{t+1} + \gamma V_\theta(S_{t+1}) - V_\theta(S_t)$$

Bellman residual gradient update:

loss: $\mathbb{E}[\delta_t^2]$

update: $\Delta\theta_t = \alpha \delta_t \nabla_\theta (V_\theta(S_t) - \gamma V_\theta(S_{t+1}))$

Work <u>WORSE</u> in practice

$$\underline{[R_{t+1} + \gamma V_\theta(S_{t+1})] - V_\theta(S_t)}$$

TensorFlow

Action-Value Function Approximation

$$q_\theta(s,a) \simeq q_\pi(s,a)$$

linear:

$$q_\theta(s,a) = \phi(s,a)^T \theta = \sum_{j=1}^{n} \phi_j(s,a)\,\theta_j$$

Stochastic Gradient Descent Update:

$$\Delta\theta = \alpha\big(q_\pi(s,a) - q_\theta(s,a)\big)\nabla_\theta q_\theta(s,a)$$

$$= \alpha\big(q_\pi(s,a) - q_\theta(s,a)\big)\phi(s,a)$$

Incremental prediction

bootstrapping

value-function approximation

off-policy learning

Tabular control learning algorithms

(Q-learning)

DQN (deep Q network)

Batch RL (Experience Replay)

Stochastic Gradient Descent with Experience Replay

Given experience consisting of $\langle \textit{state}, \textit{value} \rangle$ pairs

$$\mathcal{D} = \{\langle S_1, \hat{v}_1^\pi \rangle, \langle S_2, \hat{v}_2^\pi \rangle, ..., \langle S_T, \hat{v}_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, \hat{v}^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta\theta = \alpha(\hat{v}^\pi - v_\theta(s))\nabla_\theta v_\theta(s)$$

Converges to least squares solution

$$\theta^\pi = \underset{\theta}{\operatorname{argmin}}\ \mathrm{LS}(\theta) = \underset{\theta}{\operatorname{argmin}}\ \mathbb{E}_\mathcal{D}\left[(\hat{v}_i^\pi - v_\theta(S_i))^2\right]$$

Linear Least Squares Prediction (2)

- At minimum of $\mathrm{LS}(\theta)$, the expected update must be zero

$$\mathbb{E}_\mathcal{D}[\Delta\theta] = 0$$

$$\alpha \sum_{t=1}^{T} \phi_t(\hat{v}_t^\pi - \phi_t^\top \theta) = 0$$

$$\sum_{t=1}^{T} \phi_t \hat{v}_t^\pi = \sum_{t=1}^{T} \phi_t \phi_t^\top \theta$$

$$\theta_t = \left(\sum_{t=1}^{T} \phi_t \phi_t^\top\right)^{-1} \sum_{t=1}^{T} \phi_t \hat{v}_t^\pi$$

- For N features, direct solution time is $O(N^3)$
- Incremental solution time is $O(N^2)$ using Shermann-Morrison

MacBook Pro

Convergence of Linear Least Squares Prediction Algorithms

On/Off-Policy	Algorithm	Table Lookup	Linear	Non-Linear
On-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✓	✗
	LSTD	✓	✓	-
Off-Policy	MC	✓	✓	✓
	LSMC	✓	✓	-
	TD	✓	✗	✗
	LSTD	✓	✓	-

Policy case the cross turns into a check
mark for LS

TD / MC

Double Q-learning

Experience reply

Neural Q-learning

A network q_θ : $O_t \Rightarrow (q[1], \cdots, q[M])$

(M actions)

An ε-greedy exploration policy

$$q_t \Rightarrow \pi_t \Rightarrow A_t$$

Loss function :

Minimize

$$\boxed{J(\theta) = \frac{1}{2}\left(\boxed{R_{t+1} + \gamma\left[\max_a q_\theta(S_{t+1}, a)\right]} - q_\theta(S_t, A_t)\right)^2}$$

$$\nabla_\theta J(\theta) = \left(R_{t+1} + \gamma \max_a q_\theta(S_{t+1}, a) - q_\theta(S_t, A_t)\right) \nabla_\theta q_\theta(S_t, A_t)$$

Example: TF pseudo-code for Q-learning

```
# Compute Q values Q(S_t, .)
q = q_net(obs)

# Get action A_t
action = epsilon_greedy(q)

# Compute Q(S_t, A_t)
qa = q[action]

# Step in environment
reward, discount, next_obs = env.step(action)

# Get max of values at next state
max_q_next = tf.reduce_max(q_net(next_obs))

# Compute TD-error, do not to propagate into next state value
delta = reward + discount * tf.stop_gradient(max_q_next) - qa

# Define loss
q_loss = tf.square(delta)/2
```

Example: DQN

- DQN (Mnih et al. 2013, 2015) includes:
  - A network q_θ: $O_t \mapsto (q[1], \dots, q[m])$ (m actions)
  - An ϵ-greedy exploration policy: $q_t \mapsto \pi_t \implies A_t$
  - A replay buffer to store and sample past transitions
  - A target network q_{θ^-}: $O_t \mapsto (q^-[1], \dots, q^-[m])$
  - A Q-learning loss function on θ (uses replay and target network)

$$I(\theta) = \frac{1}{2}\left(R_{i+1} + \gamma [\![\max_a q_{\theta^-}(S_{i+1}, a)]\!] - q_\theta(S_i, A_i)\right)^2$$

  - An optimizer to minimize the loss (e.g., SGD, RMSprop, or Adam)
- Replay and target networks make RL look more like supervised learning
- It is unclear whether they are vital, but they help
- "DL-aware RL"

MacBook Pro

Multi-step updates

- When we bootstrap, updates use old estimates
- Information can propagate back quite slowly
- In MC information propagates faster, but the updates are noisier
- We can go in between TD and MC

case if you do TD with a single step you
actually only