Lez25_ReinforcementLearning4

December 22, 2023

1 Recap

Nell'ultima lezione abbiamo visto che, invece di puntare sulla value function per ottenere la politica ottima, cerchiamo di ottenere direttamente la policy, ovvero $\pi(a|s,\theta)$. Un modo è mediante la funzione *softmax*, con cui otteniamo le probabilità cercate.

Come ottimizziamo θ, per migliorare l'agente, e quindi i reward che ottiene?

Mediante il **Policy Gradient Theorem**. Ciò ci ha portato all'algoritmo di **REINFORCE**, che funziona per task episodici.

1 Initialize θ (e.g., to 0)
2 **Loop**
3 | generate episode $s_0, a_0, r_1, s_1, \ldots, r_T$ following π
4 | **for** t=0,1,...,T **do**
5 | | $G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
6 | | $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_\theta \ln \pi(a_t|s_t,\theta)$
7 | **end**
8 **EndLoop**

Dove necessitiamo del *gradiente della policy*.

2 REINFORCE con baseline

E' una piccola generalizzazione, realizzata mediante *baseline* $b(s)$, che per ciascuno stato ci dà una stima del *ritorno* per quello stato, ovvero "quanto è buono quello stato". Il *teorema policy gradient* continua a valere, e ci dice che "invece di imparare un valore $Q(S,a)$ per ogni stato ed azione, l'algoritmo apprende la differenza tra $Q(s,a)$ e la *baseline*, il tutto è espresso tramite:

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a \left(Q(s,a) - b(s)\right) \nabla \pi(a|s,\theta)$$

A cosa serve? Per la riduzione della varianza, e quindi incrementare la velocità di apprendimento. Posso avere stato che, qualsiasi azione scelta mi dà reward elevato, invece di apprendere che tale

valore è un certo numero, dico che il valore sarà "maggiore di un certo valore" (come una soglia minima). La baseline non viene appresa, bensì è un qualcosa che conosco. Metterlo a caso non ci dà miglioramenti.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t - b(s_t) \right) \nabla \ln \pi(a_t | s_t, \boldsymbol{\theta})$$

In realtà, questa conoscenza appena ottenuta è alla base di un altro concetto, l'**Actor Critic**.

2.1 Actor-Critic

L'obiettivo è sempre ridurre l'alta varanzia ottenuta dal REINFORCE. Invece di usare il ritorno G_t, si usa il **one-step-return**, che è sempre stima a partire dall'istante t partendo da stato s_t, ma invece di essere somma di tutti i reward fino a fine episodio, prendo solo il reward al momento r_t, mentre il resto è lasciato alla **value function** al tempo s_{t+1}.

$$G_{t:t+1} = r_t + \gamma V(s_{t+1})$$

L'approccio ha questo nome perchè abbiamo una **policy** (attore principale, cioè *actor*) che ci dice come agire, e una **value function** che ci dice quanto sono buone le azioni prese (quindi è un *critico*, ovvero *critic*)

REINFORCE era usabile solo a fine episodio con tutti i reward. Questo approccio, con singolo reward, mi permete con one-step-return di aggiornare i pesi:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - \hat{V}(s_t, \boldsymbol{w}) \right) \nabla \ln \pi(a_t | s_t, \boldsymbol{\theta}) = \\ &= \boldsymbol{\theta}_t + \alpha \left(r_{t+1} + \gamma \hat{V}(s_{t+1}, w) - \hat{V}(s_t, w) \right) \nabla \ln \pi(a_t | s_t, \boldsymbol{\theta}) \end{aligned}$$

Se apprendo value function, uso come *baseline* proprio l'ultima value function appresa. La value function dipende da w, con cui la approssimo e θ, che è quella che sto stimando.

L'algoritmo è il seguente:

1 Initialize θ and w (e.g., to 0)
2 **Loop**
3 | Initialize s as first state of the episode
4 | $I \leftarrow 1$
5 | **while** s *not terminal* **do**
6 | | choose action a according to $\pi(\cdot|s, \theta)$
7 | | observe s' and r
8 | | $\delta \leftarrow r + \gamma \hat{V}(s', w) - \hat{V}(s, w)$
9 | | $w \leftarrow w + \alpha^{w} \delta \nabla \hat{V}(s, w)$
10 | | $\theta \leftarrow \theta + I \alpha^{\theta} \delta \nabla \ln \pi(a|s, \theta)$
11 | | $s \leftarrow s'$
12 | **end**
13 **EndLoop**

- Inizializzo vettori θ (reinforce) e w (value function).
- Entro in loop infinito, in realtà dipende da quanto tempo voglio dedicare al training (quindi quanti episodi voglio simulare…)
- Inizializzo s come stato iniziale dell'episodio, ed $I = 1$.
- Vado avanti finchè s non arriva a stato terminale (con il REINFORCE non era così).
- In ogni stato scelgo azione, in accordo con $\pi(\cdot|s, \theta)$, ed osserviamo s' ed r.
- In riga 8, calcolo la differenza tra *one-step-return* e $\bar{V}$, che uso per aggiornare entrambi i vettori, mediante metodo gradiente. Nel primo caso (riga 9) lo calcolo rispetto a $\bar{V}$, nel secondo rispetto π. Nella value function uso Q-deep, per la policy qualsiasi rete con uscita softmax. Ad ogni step aggiorno entrambe le reti usando valori target.

Questo algoritmo, visto così, è per task episodici. Possiamo estenderlo nel continuo, Se $J(\theta)$ non è calcolabile, perchè le misure sarebbero infinite, si usa invece il **reward medio**, che mantiene lo stesso nome.

3 Spazi di azione larghi o continui

Come facciamo a gestire questo problema? Usare REINFORCE con softmax, con un milione di azioni, richiede un milione di uscite nell'ultimo livello della rete.

Ciò che si fa è:

Invece di apprendere $\pi(a|s, \theta)$, si usa una distrubuzione di probabilità nota. Assusiamo che la policy sia *distribuzione normale*:

$$\pi(a|s,\boldsymbol{\theta}) = \frac{1}{\sigma(s,\boldsymbol{\theta})\sqrt{2\pi}}\exp\left(-\frac{(a-\mu(s,\boldsymbol{\theta})^2)}{2\sigma(s,\boldsymbol{\theta})^2}\right)$$

Media e varianza dipendo da s e θ, quindi calcolo per ogni stato media e varianza opportune per produrre policy stocastica. La scelta dell'azione verrà campionata dalla distribuzione di probabilità.

3.1 Approcci derivati

Abbiamo:

- **Deterministic Policy Gradient (DPG)**: Introdotto da Silver et al. nel 2014 nel loro articolo intitolato "Deterministic Policy Gradient Algorithm". Questo algoritmo è progettato per politiche deterministiche, che sarà quella ottima.
- **Deep Deterministic Policy Gradient (DDPG)**: Un'estensione del DPG che utilizza reti neurali profonde (DNNs) introdotte da Lillicrap et al. nel 2016. Il loro articolo "Controllo continuo con apprendimento profondo per rinforzo" discute questo approccio.
- **Proximal Policy Optimization (PPO)**: Considerato un algoritmo all'avanguardia, introdotto da Schulman et al. Il loro articolo "Algoritmi di ottimizzazione delle politiche prossimali" presenta questo metodo. Oggi è uno *standard*.

Questi algoritmi rappresentano importanti progressi nell'apprendimento per rinforzo, con il DDPG che integra reti neurali nel framework del gradiente di politica deterministica e il PPO che si distingue come tecnica all'avanguardia.

3.1.1 Curiosità ChatGPT

ChatGpt è composto da:

- *Gpt* : language model, data porzione di testo genera token associato a prossima parola
- *Chat*: per l'interazione con gli utenti.

Per l'addrestramento è stato usato *Reinforcement Learning with Human Feedback*, ovvero gli utenti giudicano le risposte prodotte.

3.1.2 Topic RL avanzati

Le tematiche nel campo RL non si concludono qui, abbiamo altri rami:

- *Multi agent RL*: si hanno più agenti che cooperano (o sono antagonisti, come in una partita a ping-pong).
- *Hierarchical RL*: task suddivisi in task di alto livello, composte da task di basso livello. Ad esempio, un task di alto livello è "Vai all'uscita", composto dai task di basso livello "Vai avanti, Gira a destra".
- *RL + Heuristic Tree Search*: negli scacchi si ha reward immediato (come la cattura di un pezzo), a cui corrisponde una risposta dell'avversario, a cui risponderò io a mia volta. Ciò genera un *albero*, che ci porta in tantissime ramificazioni. L'Heuristic Tree Search, in maniera

euristica, è usato negli scacchi. Euristica perchè, se il giocatore è "scarso", non perdo tempo a fare troppe previsioni. Con spazio di stati grandi, apprendere unica policy è faticosa, ma è meglio del vedere solo *cosa posso fare ora*, in cui non ho una visione di insieme. Qui cerchiamo di mettere questi due aspetti insieme.

- *Transfer RL*: L'agente ha appreso un qualcosa da un certo task, posso riusare questo apprendimento per altri task?