Lez24_ReinforcementLearning3

December 20, 2023

1 Recap

Riprendiamo l'algoritmo di **Deep Q-Learning**. A riga 7, si prende un minibatch delle tuple. A riga 8 vengono creati i target (etichette) y che la rete usa, a riga 8, come confronto con l'errore quadratico medio.

1 **Initialize** w
2 **Initialize** empty buffer $\mathcal{B}$
3 $i \leftarrow 0$
4 **Loop**
5   choose action a_i
6   gather experience $\langle s_i, a_i, r_i, s_{i+1} \rangle$ and add to $\mathcal{B}$
7   sample minibatch of b $\langle s_j, a_j, r_j, s_{j+1} \rangle$ tuples from $\mathcal{B}$
8     $y^{(j)} \leftarrow r_j + \gamma \max_{a'} \hat{Q}(s_{i+1}, a', \boldsymbol{w}), j = 1, \ldots, b$
9     $\mathcal{L}^{(j)} = (y^{(j)} - \hat{Q}(s_j, a_j, \boldsymbol{w}))^2$ /* Loss */
10   update w using, e.g., SGD on the minibatch
11     $i \leftarrow i + 1$
12 **EndLoop**

Per tutto il vettore di uscita della rete, sto calcolando solo **una** etichetta/target, a fronte di **n** uscite. La Q è predetta per tutte le azioni. Io predico il reward in funzione di una singola azione, non di tutte le azioni. Come risolvo questa problematica? La soluzione è semplice: metto come target il valore prodotto dalla rete (quindi se la rete produce x, il valore target sarà proprio x, che porterà la Loss al valore 0) e non apprenderà nulla da questi altri termini.

2 Policy Based RL

La politica viene appresa in modo autonoma rispetto all'apprendimento di una value function. Invece di approssimare la value function, cerchiamo di approssimare la policy ottima, parametrizzata così: Probabilità di scegliere azione a, sapendo di essere in un certo stato s con certi parametri θ (come la w nella value function).

$$\pi(a|s,\boldsymbol{\theta}) = P(A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$$

$\boldsymbol{\theta} \in \mathbb{R}^m$ is the vector of policy parameters

dove $\pi(a|s,\theta)$ è qualsiasi funzione differenziabile rispetto a θ, perchè per ottimizzare la scelta useremo il ben noto **gradiente**, che richiede differenziabilità.

Notiamo che θ (policy) viene usato come w nella value function, ma tuttavia possiamo incorrere in casi in cui verranno usati entrambi.

Anche qui abbiamo performance da migliorare, supponiamo metrica $J(\theta)$, che ci dice quanto è bravo il nostro agente, la vogliamo massimizzare. Quindi non usiamo discesa del gradiente (quindi direzione opposta del gradiente), bensì **gradient ascent**, ovvero in direzione del gradiente:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta}_t)$$

Parleremo quindi di **gradient policy** per tutti i metodi che richiedono queste implementazioni.

2.1 Perchè approssimare la policy?

- La policy, tendenzialmente, è più facile (ovvero addestrare l'agente per farlo convergere alla politica ottima) da approssimare rispetto la value function.
- La politica ottenuta è più regolare, mentre cambiare la *value function* comporta cambiamenti più radicali nella politica. Abbiamo quindi garanzie di convergenza migliori.
- Possiamo apprendere *politiche stocastiche*, mentre prima eravamo limitati a quelle *deterministiche*.
- Possiamo lavorare con spazi di azione *continui*, ovvero nulla vieta di dire che a sia azione reale (come angolo dello sterzo durante una curva).

Supponiamo di lavorare comunque in uno *spazio di azione discreto*. Una scelta di *approssimazione per la policy* potrebbe ricadere nella **softmax**, in cui $h(s,a,\theta)$ ci dice quanto preferiamo una certa azione in un certo stato, ed è un valore arbitrario. La usiamo in:

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_{a'} e^{h(s,a',\boldsymbol{\theta})}}$$

Non ci serve capire come calcolare questi valori, bensì è un "modo" per parametrizzare meglio $\pi(a|s,\theta)$

Cioè la softmax li usa per generare la probabilità. Non è un iperparametro, in quanto dipende da altri parametri, ma non dobbiamo saperlo "prima del training". I valori che hanno non sono di interesse. Allora, perchè sono utili?

Questi *theta* ricordano molto $Q(s, a)$. Tra due azioni, la differenza tra due $Q(s, a)$ può essere minima (perchè magari, in un percorso lungo, una singola scelta può essere poco significativa), e viene associata con un parametro detto *temperatura* per aumentare questo divario. Le **preferenze**, invece, non necessitano di convergere a specifici valori, ma semplicemente al miglior valore per la policy che stiamo apprendendo.

2.2 Policy Gradient in Episodic Task

Consideriamo *task episodici*, partendo da s_0. Quindi stiamo valutando:

$$J(\boldsymbol{\theta}) = V_{\pi_{\boldsymbol{\theta}}}(s_0)$$

Il theta cambia le azioni scelte, ma ciò cambia la distrubuzione degli stati durante l'episodio, che dipende sia dalla politica sia dall'ambiente. Questa "dipendenza ricorsiva" è un bel problema.

Ci aiuta il **Policy gradient theorem**:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a Q_\pi(s, a) \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta})$$

Come osservabile, si parla di **proporzionalità**, non uguaglianza.

Secondo il prof, questo teorema è bellissimo, perchè noi vorremmo la parte a sinistra della proporzionalità, e questo teorema ci dice che possiamo usare il gradiente rispetto a π, che invece sappiamo calcolare.

Ciò ci è di aiuto perchè:

- Abbiamo proporzionalità rispetto alla lunghezza dell'episodio.
- Non serve la derivata di μ, che era il problema principale.

2.2.1 Dimostrazione (purtroppo)

To simplify notation, we leave it implicit that π is a function of $\boldsymbol{\theta}$, and that gradients are w.r.t. $\boldsymbol{\theta}$

$$\nabla V_\pi(s) = \nabla\left[\sum_a \pi(a|s)Q_\pi(s,a)\right] =$$

$$= \sum_a \nabla[\pi(a|s)Q_\pi(s,a)] =$$

$$= \sum_a [\nabla\pi(a|s)Q_\pi(s,a) + \pi(a|s)\nabla Q_\pi(s,a)] =$$

$$= \sum_a \left[\nabla\pi(a|s)Q_\pi(s,a) + \pi(a|s)\nabla\sum_{s',r'} p(s',r'|s,a)(r + V_\pi(s'))\right] =$$

Nella seconda riga, mettiamo la derivata nella sommatoria. Nella terza riga, trattandosi di una derivata di un prodotto, la esplicitiamo. Nella quarta riga, esplicitiamo $Q_{pi}(s,a)$.

Soffermiamoci su:

$$= \sum_a \left[\nabla\pi(a|s)Q_\pi(s,a) + \pi(a|s)\nabla\sum_{s',r'} p(s',r'|s,a)(r + V_\pi(s'))\right] =$$

Ci stiamo concentrando sul *reward*, ovvero ci concentriamo sulla politica non sull'azione. Il reward non dipende da θ (ma solo rispetto r) allora la derivata rispetto a θ scompare.

Successivamente, scomponiamo la *sommatoria* in s', r', come due sommatorie. Come vediamo, non c'è nulla in funzione di r', quindi questa seconda sommatoria non ha alcun impatto, come si può vedere nel punto 2).

2) $\sum_{s'} \sum_{r'} p(s',r'|s,a)V_\pi(s') = \sum_{s'} p(s'|s,a)V_\pi(s')$

$$= \sum_a \left[\nabla\pi(a|s)Q_\pi(s,a) + \pi(a|s)\sum_{s'} p(s'|s,a)\nabla V_\pi(s')\right] =$$

Note: we are computing $\nabla V_\pi(s)$ and now we have a recursive term $\nabla V_\pi(s')$! Let's unroll the recursion...

Come scritto in inglese, c'è ricorsività.

$$= \sum_a \Bigg[\nabla \pi(a|s) Q_\pi(s,a) + \pi(a|s) \sum_{s'} p(s'|s,a) \cdot$$

$$\sum_{a'} \Bigg(\nabla \pi(a'|s') Q_\pi(s',a') + \pi(a'|s') \sum_{s''} p(s''|s',a') \nabla V_\pi(s'') \Bigg) \Bigg] =$$

Esplodiamo $\nabla V_\pi(s')$, ottenendo la parte rossa moltiplicata per $\nabla V_\pi(s'')$, quindi ciò che cambia è che siamo passati da s' ad s''. Ma non basta, perchè dobbiamo continuare a srotolare:

$$= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} P(s \to x, k, \pi) \sum_a [\nabla \pi(a|x) Q_\pi(x,a)]$$

where $P(s \to x, k, \pi)$ is the probability of transitioning from s to x in k steps under policy π.

Adesso possiamo scrivere un'espressione per il gradiente di J:

$$\nabla J(\boldsymbol{\theta}) = \nabla V_\pi(s_0) = \sum_s \sum_{k=0}^{\infty} P(s_0 \to s, k, \pi) \sum_a [\nabla \pi(a|s) Q_\pi(s,a)] =$$

$$= \sum_s \eta(s) \sum_a [\nabla \pi(a|s) Q_\pi(s,a)] =$$

$\eta(s)$: avg. number of steps spent in s within an episode

$$= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a [\nabla \pi(a|s) Q_\pi(s,a)] =$$

Nell'ultima riga, le parti in rosso sono *costanti*, abbiamo moltiplicato e diviso per la stessa quantità, perchè?

$$= \sum_{s'} \eta(s') \sum_{s} \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_{a} [\nabla \pi(a|s) Q_\pi(s,a)] =$$

$$= \sum_{s'} \eta(s') \sum_{s} \mu(s) \sum_{a} [\nabla \pi(a|s) Q_\pi(s,a)]$$

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} [\nabla \pi(a|s) Q_\pi(s,a)]$$

Perchè ci possiamo ricondurre a μ, arrivando finalmente alla formula finale, visibile in colorazione blue misto lacrime dopo aver letto questa dimostrazione.

3 Reinforce

$\mu(s)$ è la distribuzione **on-policy** degli stati sotto π, se π viene seguito, lo stato occorrerà (cioè lo incontriamo) secondo questa proporzione:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto \sum_{s} \mu(s) \sum_{a} Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) =$$
$$= E_\pi \left[\sum_{a} Q_\pi(s_t,a) \nabla_{\boldsymbol{\theta}} \pi(a|s_t,\boldsymbol{\theta}) \right]$$

Ora, come abbiamo approssimato la *value function*, possiamo attuare un *gradiente stocastico ascendente* sugli stati s_t che incontriamo.

Introduciamo altri passaggi, in cui sostituiamo la *somma sugli stati* con un'*attesa rispetto* π, iniziamo pesando le azioni con $\pi(a|s,\theta)$:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \propto E_\pi \left[\sum_a Q_\pi(s_t, a) \nabla_{\boldsymbol{\theta}} \pi(a|s_t, \boldsymbol{\theta})\right] = \\
= E_\pi \left[\sum_a \pi(a|s_t, \boldsymbol{\theta}) Q_\pi(s_t, a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|s_t, \boldsymbol{\theta})}{\pi(a|s_t, \boldsymbol{\theta})}\right] = \\
= E_\pi \left[Q_\pi(s_t, a_t) \frac{\nabla_{\boldsymbol{\theta}} \pi(a_t|s_t, \boldsymbol{\theta})}{\pi(a_t|s_t, \boldsymbol{\theta})}\right] = \\
= E_\pi \left[G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(a_t|s_t, \boldsymbol{\theta})}{\pi(a_t|s_t, \boldsymbol{\theta})}\right]
\end{aligned}
$$

Invece di sommare su tutte le azioni, ad ogni istante considero solo a_t. Ciò ci permette di scrivere $Q_\pi(s_t, a_t)$ come G_t, ottenendo:

$$
\nabla J(\boldsymbol{\theta}) \propto E_\pi \left[G_t \frac{\nabla_{\boldsymbol{\theta}} \pi(a_t|s_t, \boldsymbol{\theta})}{\pi(a_t|s_t, \boldsymbol{\theta})}\right]
$$

Possiamo campionare il ritorno G_t per *ogni istante di tempo*, ottenendo una espressione *proporzionale* al gradiente. Ora possiamo usare *stochastic gradient* per aggiornare i parametri, ottenendo l'algoritmo **Reinforce**. Quindi possiamo aggiornare i parametri θ, che approssimano la politica in base alla fine dell'episodio, sapendo tutti gli stati e le azioni che ho scelto.

$$
\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha G_t \nabla_{\boldsymbol{\theta}} \ln \pi(a_t|s_t, \boldsymbol{\theta})
$$

$$
\frac{\nabla_\theta \pi(a_t|s_t, \theta)}{\pi(a_t|s_t, \theta)} = \nabla_\theta \ln \pi(a_t|s_t, \theta)
$$

3.1 Algoritmo

Notiamo l'iterazione interna da 0 a T, cioè fino alla fine dell'episodio: Non aggiorniamo i pesi man mano, perchè il **ritorno** G_t è noto solo a **fine episodio**, non durante.

```
Initialize θ (e.g., to 0)
Loop
    generate episode s_0, a_0, r_1, s_1, ..., r_T following π
    for t=0,1,...,T do
        G_t ← Σ_{k=t+1}^{T} γ^{k-t-1} r_k
        θ ← θ + α γ^t G_t ∇_θ ln π(a_t|s_t, θ)
    end
EndLoop
```

4 Policy Gradient: altra prospettiva

Prendiamo una rete neurale usata per classificazione multi-classe, avente *softmax* alla fine, che produce una probabilità y_c per ogni classe c.

Il gradiente della **loss cross-entropy** usata nel training è:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} \left[\sum_c \bar{y}_c \ln y_c \right]$$

Intuitivamente, il training incrementa y_c per la classe c se questa viene etichettata correttamente (si parla di *ground truth*).

Ora, se poniamo `classi == azioni`, l'output sarà $\pi(a|s,\theta)$, e l'aggiornamento del **REINFORCE** sarà proporzionale a $\nabla_\theta \ln \pi(a_t|s_t,\theta)$, che non presenta *ground truth*, quindi la probabilità è aumentata o diminuita unicamente considerando il ritorno.