DeepHarvest

Categoria: Studenti di laurea triennale

Simone Marullo
simone.marullo@student.unisi.it

5 settembre 2018

Introduzione

Per l'esame di Programmazione e Progettazione Software (Ing. informatica e dell'informazione, II anno, Università di Siena) ho sviluppato un videogioco e un agente [1] che apprende a giocare tramite *reinforcement learning*.

1 Descrizione del gioco

DeepHarvest è un gioco in stile arcade il cui ambiente è costituito da una griglia 9x9 in cui si può muovere (di un passo, in direzione parallela agli assi) la pedina del giocatore. Sulla griglia sono presenti, in numero fissato, due categorie di item: i *bonus*, rappresentati come fragole e i *malus*, rappresentati come buche. Quando il giocatore si sposta su una cella in cui era precedentemente presente un item, il punteggio del giocatore si modifica conseguentemente e l'item viene trasferito in un'altra cella, estratta casualmente. In particolare, se si tratta di una fragola lo score aumenta di una quantità proporzionale alla dimensione visibile dell'item; se si tratta di una buca lo score subisce una diminuzione proporzionale al punteggio accumulato fino a quel momento dal giocatore.

Il giocatore ha a disposizione non più di 15 secondi e 50 mosse per accumulare il punteggio più alto possibile; il gioco termina non appena uno dei due vincoli venga superato. A quel punto viene ripristinata la condizione iniziale e data la possibilità ad un agente AI di giocare; al termine vengono confrontati i punteggi.

[1] È possibile vedere una demo all'indirizzo http://sailab.diism.unisi.it/deepharvest/play/

2 Descrizione dell'agente

La struttura dell'agente AI è direttamente ispirata alle tecniche presentate in [1], in cui sono state usate deep Q-network per ottenere performance comparabili o superiori a quelle umane nei classichi giochi Atari 2600. Tali architetture sfruttano i molteplici strati di nodi (nello specifico, strati gerarchicamente organizzati di filtri convolutivi) per ottenere rappresentazioni progressivamente più astratte di input sensoriali di grandi dimensioni e apprendono con successo politiche efficaci in una varietà di ambienti.
L'agente interagisce con l'ambiente tramite osservazioni, azioni e ricompense (eventualmente anche negative) e il suo obiettivo è massimizzare il ritorno, cioè la somma delle ricompense durante l'esecuzione della policy. Si definisce lo stato $s_t = \{x_1, a_1, x_2, a_2, ..., x_t\}$, dove a_t è la mossa, all'interno del set di mosse legali $\mathcal{A}$, intrapresa all'istante t e x_t è lo stato osservato (la griglia); ad ogni istante temporale l'agente effettua una mossa, la quale modifica lo stato interno del gioco, che in generale può non coincidere con lo stato osservabile. L'ambiente in generale può essere stocastico.

Al tempo t il ritorno atteso fino all'istante temporale finale T è definito come

$$R_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_t$$

in cui si è assunto un discounting factor $\gamma \in [0, 1)$ fissato (solitamente molto vicino a 1); tale scelta permette la convergenza della somma anche in giochi di durata non limitata. Il ruolo del deep Q-network è approssimare la *action-value function* ottimale al fine di valutare l'opportunità di prendere una decisione piuttosto che un'altra:

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = \max_\pi \mathbb{E}\big[R_t | s_t = s, a_t = a, \pi\big]$$

dove π è una policy, la quale mappa gli stati s con una distribuzione di probabilità delle azioni a da intraprendere.
In reinforcement learning è di fondamentale importanza l'equazione di Bellman, che fornisce il seguente risultato, valido per la action-value function:

$$Q^*(s,a) = \mathbb{E}_{s'}\big[r + \gamma \max_{a'} Q^*(s',a') | s, a\big]$$

Intuitivamente, se è noto $Q^*(s',a')$ per lo stato successivo s' e per tutte le possibili azioni a' emerge che la strategia ottima è quella di selezionare l'azione a' che massimizza il valore atteso di $r + \gamma \max_{a'} Q^*(s',a')$. Al posto di $Q^*(s,a)$, si utilizza tipicamente un approssimatore di funzioni $Q(s,a,\theta)$, che in questo caso è una *deep neural network* con pesi θ.
Ad ogni passo di ottimizzazione, l'obiettivo è minimizzare, agendo sui pesi θ_i del network all'iterazione i, lo scarto quadratico medio nell'equazione di Bellman, dove al posto di $Q^*(s,a)$ abbiamo $Q(s,a,\theta)$. Si ottiene ad ogni passo una *loss function*:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r}\Big[\Big(\mathbb{E}_{s'}\big[r + \gamma \max_{a'} Q(s',a',\theta_i^-) | s, a\big] - Q(s,a,\theta_i)\Big)^2\Big]$$

dove θ_i^- sono i pesi ad una precedente iterazione. Si può notare che il target dipende dai pesi attuali del network (non è infatti un task di supervised learning).
Approcciare task di reinforcement learning con un approssimatore nonlineare come una rete neurale tipicamente implica dover affrontare problemi di instabilità, causati da una molteplicità di fattori. Tra questi c'è la correlazione presente nella sequenza di osservazioni e il fatto che piccoli aggiornamenti dei *Q-values* possono modificare significativamente la policy seguita dall'agente e quindi la distribuzione dei dati progressivamente osservati. In [1] viene proposta a questo proposito l'idea di utilizzare un target network separato: esistono due deep Q network strutturalmente identici e θ_i^- sono i pesi del *target network*, mentre θ_i sono i pesi del *primary network*; ogni C passi si ha l'aggiornamento $\theta_i^- \leftarrow \theta_i$. Questa tecnica è stata raffinata in [3], dove l'aggiornamento si ha ad ogni passo ma è molto lento (in modo da evitare potenziali instabilità causate da improvvisi aggiornamenti):

$$\theta_i^- \leftarrow \eta\theta_i + (1-\eta)\theta_i^-$$

con $\eta << 1$.
In [1] viene anche utilizzata una tecnica nota come *experience replay*: si costituisce un buffer contenente tuple costituite da stato — azione — ricompensa — stato successivo e l'ottimizzazione viene condotta su campioni estratti uniformemente da tale buffer. Il vantaggio principale rispetto al Q-learning online standard è una maggiore efficienza nell'apprendimento: il learning di esperienze consecutive è infatti inficiato dalla forte correlazione. Inoltre si verifica che tale approccio mitiga il rischio di loop di retroazione e quindi il blocco dei pesi in un minimo locale di scarsa qualità o la loro divergenza.
In [4] si affronta il problema della sovrastima del valore delle azioni subottime, che può effettivamente impedire all'agente di apprendere la politica ottima. Per mitigare il problema si usa il primary network per selezionare l'azione da intraprendere e si utilizza il target network per valutare il Q-value; in pratica disaccoppiando i due passaggi su due set di pesi si riduce sostanzialmente la sovrastima e si ottengono performance migliori in tempi minori.
Per facilitare una approfondita esplorazione dello spazio degli stati durante l'apprendimento si utilizza una ϵ-greedy policy: con probabilità ϵ viene eseguita una mossa casuale; con probabilità $1-\epsilon$ viene invece scelta la mossa $a = \arg\max_{a'} Q(s,a',\theta)$.

3 Descrizione dell'architettura

Nel caso dell'architettura DRQN-a, l'input della rete neurale è costituito da un'immagine 11 x 11 x 3. Il primo strato nascosto convolve 32 filtri con kernel 1x1 e stride 1, il secondo 64 filtri con kernel 2x2 e stride 1, il terzo 512 filtri con kernel 10x10 e stride 1.
Nel caso dell'architettura DRQN-b, l'input della rete neurale è costituito da un'immagine 84 x 84 x 3. Il primo strato nascosto convolve 32 filtri con kernel 8x8 e stride 4, il secondo 64 filtri con kernel 4x4 e stride 2, il terzo 64 filtri con kernel 3x3 e stride 1, il quarto strato da 512 filtri con kernel 7x7 e stride 1.
Le funzioni di attivazione utilizzate sono Rectified Linear Units (ReLU).

L'output dell'ultimo strato convolutivo viene usato come input di uno strato ricorrente (cella LSTM a 512 unità): questo elemento architetturale permette di facilitare l'apprendimento di politiche che tengano in considerazione l'evoluzione temporale del gioco. Questa tecnica viene descritta in [2]: una struttura ricorrente fornisce una valida alternativa allo stacking dei frame di gioco che viene effettuato tipicamente nel deep Q network standard (infatti s_t include tutti gli stati osservabili agli istanti $\tau < t$; nella pratica si utilizza come input al network lo stacking dei soli frame più recenti). Nonostante non fornisca vantaggi sistematici nel processo di learning, tuttavia risulta meno esoso in termini di potenza di calcolo richiesta e si dimostra particolarmente adatto a fornire stime di buona qualità anche in caso di incompletezza delle osservazioni (problemi ad osservabilità parziale).
All'output dello strato ricorrente sono connessi due *stream* di strati fully-connected: questa tecnica è nota come *dueling Q-network* (fig. 1) ed è stata presentata in [5]. L'obiettivo è stimare separatamente la Value function V^π e la Advantage function A^π (il cui ruolo è dare una stima del valore relativo delle varie azioni, svincolato dal valore dello stato corrente), dove:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s)}\big[Q^\pi(s,a)\big]$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Intuitivamente la Value function indica quanto l'essere in uno specifico stato è favorevole e il motivo per cui un'architettura dueling può avere prestazioni migliori è che possono esistere situazioni all'interno dell'ambiente di gioco altamente favorevoli, a prescindere da qualsiasi azione si prenda; disaccoppiando quindi la valutazione dello stato dalla scelta dell'azione è possibile ottenere stime più robuste. Infine le stime di Advantage e Value vengono combinate per produrre i Q-values. Nell'architettura utilizzata, indicando con α i parametri dell'advantage stream e con β i parametri del value stream:

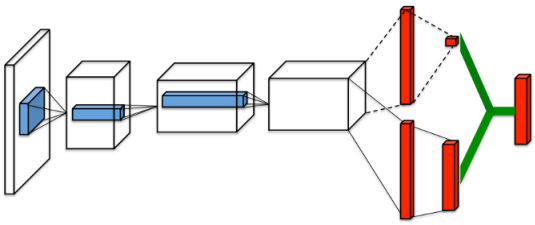

Figura 1: Architettura dueling

$$Q(s,a,\theta,\alpha,\beta) = V(s,\theta,\beta) + A(s,a,\theta,\alpha)$$

Si nota che non è possibile recuperare univocamente V e A dalla Q function: infatti, posto di aver identificato A e V è possibile trovare nuovi candidati $A' = A + c$ e $V' = V - c$ che rispettano il vicolo dell'equazione precedente; questo fatto (la *non identificabilità* di A e V) si traduce in scarse performance nell'ottimizzazione. Tale problema è risolto forzando, ad esempio, lo stimatore $A(s,a,\theta,\alpha)$ ad essere nullo in corrispondenza dell'azione effettivamente scelta:

$$A(s, \arg\max_{a'} A(s,a',\theta,\alpha),\theta,\alpha) = 0$$

Questa scelta è giustificata dal fatto che

$$\mathbb{E}_{a\sim\pi(s)}\big[A^\pi(s,a)\big] = 0$$

e nel caso di policy deterministica, $Q(s,a^*) = V(s)$ quindi $A(s,a^*) = 0$. Allora l'ultimo modulo della rete effettuerà l'associazione:

$$\begin{aligned} Q(s,a,\theta,\alpha,\beta) =& V(s,\theta,\beta) \\ &+ \big(A(s,a,\theta,\alpha) - \max_{a'\in\mathcal{A}} A(s,a',\theta,\alpha)\big) \end{aligned}$$

Nella pratica si verifica che è possibile aumentare la stabilità del processo di ottimizzazione imponendo invece:

$$\begin{aligned} Q(s,a,\theta,\alpha,\beta) =& V(s,\theta,\beta) \\ &+ \Big(A(s,a,\theta,\alpha) \\ &- \frac{1}{|\mathcal{A}|}\sum_{a'\in\mathcal{A}} A(s,a',\theta,\alpha)\Big) \end{aligned}$$

Tale risultato non è un passo algoritmico distinto ma fa parte del deep Q network, il quale richiede quindi esclusivamente *backpropagation*. Come ottimizzatore è stato utilizzato Adam.

4 Risultati ottenuti

Per valutare il comportamento dell'agente nel corso del training, si è fissato un ambiente di gioco Env1, con 11 malus e 4 bonus di valore compreso tra 16 e 64 unità e somma complessiva pari a 100 unità. Sono state considerate le seguenti metriche: punteggio finale medio per episodio, numero medio di mosse *dumb* (sono così definite le mosse che non producono

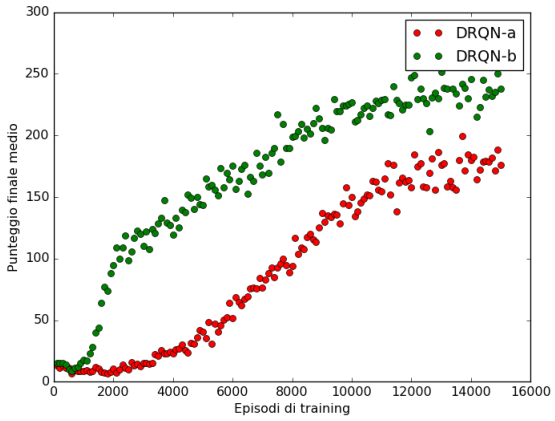

Figura 2: Punteggio per episodio

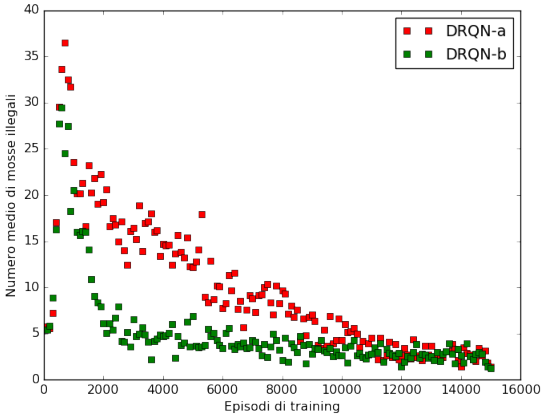

Figura 4: Mosse illegali

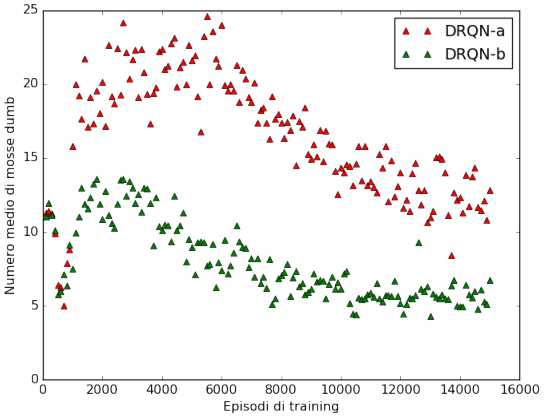

Figura 3: Mosse dumb

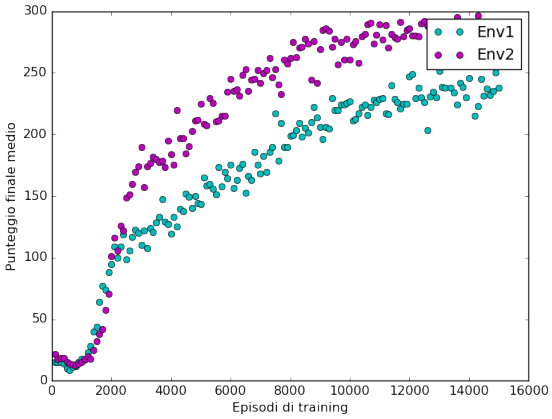

Figura 5: Confronto ambienti di gioco

la cattura di alcun item e il cui spostamento sulla griglia viene annullato dalla mossa successiva) e numero medio di mosse illegali (l'agente chiede di muoversi oltre i confini della griglia); si mostrano i valori medi di queste metriche ogni 100 episodi.

L'architettura DRQN-b si mostra superiore per quanto riguarda tutte le metriche considerate.

Fissata l'architettura DRQN-b, si è confrontato (fig. 5) l'andamento dei punteggi medi ottenuti in Env1 e quelli ottenuti in Env2, un ambiente analogo al primo che presenta però 7 malus sulla griglia. In accordo con quanto ci si aspetta, l'evoluzione dell'agente è analoga ma essendo il gioco diventato più facile i punteggi sono più alti.

Riferimenti bibliografici

[1] "Human-level control through deep reinforcement learning" (Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al.) Nature volume 518, pages 529–533 (26 February 2015)

[2] "Deep Recurrent Q-Learning for Partially Observable MDPs" (Matthew Hausknecht e Peter Stone) arXiv:1507.06527

[3] "Continuous control with deep reinforcement learning" (Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel et al.) arXiv:1509.02971

[4] "Deep Reinforcement Learning with Double Q-Learning" (Hado van Hasselt , Arthur Guez, and David Silver) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

[5] "Dueling Network Architectures for Deep Reinforcement Learning" (Ziyu Wang, Tom Schaul, Matteo Hessel et al.) arXiv:1511.06581