

# Confronto delle baselines di OpenAI su Street Fighter II

Filippo Soncini, Gianluca Lutero

Luglio 2019

## Introduzione

In questo lavoro verranno confrontate le prestazioni di alcuni algoritmi baseline di OpenAI nel creare un agente in grado di giocare o completare il picchiaduro Street Fighter II - Special Champion Edition (per Sega Genesis). Lo stato attuale dell'arte non fornisce strategie di sviluppo efficaci per i picchiaduro in generale, ciò che si può trovare sono tentativi di utilizzo di alcuni algoritmi, presentati anche in questo lavoro, in "ambienti" personalizzati dove vengono fornite direttamente tutte le informazioni sul gioco all'agente, mentre nel nostro caso forniamo solo una ristretta quantità di informazioni e i frame di gioco.

## 1 Street Fighter II

Street Fighter II: Champion Edition è un gioco del genere picchiaduro rilasciato da Capcom nel 1992. L'obiettivo del giocatore, una volta selezionato un personaggio, è di combattere e sconfiggere in scontri uno-conto-uno a difficoltà crescente tutti gli altri personaggi (compreso se stesso) fino a completamento del torneo.






Il gioco è composto da 12 personaggi giocabili:











Figura 1: Tutti i 12 personaggi giocabili

Necessaria una breve descrizione delle meccaniche di combattimento particolari del gioco per poter comprendere a pieno alcune scelte fatte successivamente per questo lavoro. Il gioco, oltre alle normali azioni che possono essere fatte, prevede anche la possibilità di eseguire mosse speciali, tramite la combinazione di alcuni tasti, specifiche per ogni personaggio utilizzabile. Esistono essenzialmente due tipi di mosse speciali in questo gioco:

1. Mosse speciali **Standard** che utilizzano i tasti movimento in sequenza circolare seguiti da pugni e/o calci [Fig.2a].
2. Mosse speciali **Caricate** che utilizzano una sequenza di tasti movimento con il requisito che il primo tasto debba essere tenuto premuto per almeno 2 secondi seguiti da pugni e/o calci [Fig. 2b].

Move Name	Input
Hadoken	 + 
Red Hadoken	 + 
Fake Hadoken	 + 

(a) Standard

Move Name	Input
Sonic Boom	 HOLD  + 
Flash Kick	 HOLD + 
Super Combo: Double Flash	 HOLD  + 

(b) Caricate

Figura 2: Esempi di mosse speciali

## 1.1 Street Fighter AI

L'”AI” del computer é gestita da un meccanismo di script e vantaggi in gioco che gli permette di confrontarsi con i giocatori reali, in particolare:

- Il computer legge l'input dell'avversario, permettendo così diversi frame d'anticipo per scegliere cosa fare.
- Il computer non necessita di premere tutta la serie di tasti per eseguire le mosse speciali, queste vengono eseguite istantaneamente.

Le mosse fatte dal computer sono raggruppate in piccoli script. Ogni personaggio ha un suo repertorio di script diversi per ogni avversario che potrebbero affrontare nel gioco, e un insieme di circostanze in cui eseguirle. Gli script vengono scelti in modo casuale tra quelli a disposizione del personaggio per quella determinata situazione.

## 1.2 Impostazioni difficoltà

Il gioco include 8 livelli di difficoltà all'aumentare del quale il computer farà maggior danno all'avversario, ne subirà meno e soprattutto sceglierà (modificando le probabilità) script per contrattaccare l'avversario sempre più elaborati.



Figura 3: Sistema di gestione difficoltà

## 2 Setup dell'ambiente di sviluppo

Per poter far comunicare l'agente con il gioco e testare i vari algoritmi abbiamo utilizzato la piattaforma Gym Retro (un branch dedicato di OpenAI) che consente di creare per videogiochi classici ambienti Gym per la sperimentazione di tecniche di reinforcement learning, inoltre contiene diversi supporti per l'integrazione di nuovi giochi e la ricerca di nuove variabili di gioco da usare nella sperimentazione.

Per Street Fighter II sono state estratte ed utilizzate le seguenti informazioni di gioco, non tutte le informazioni sono state usate in tutti gli algoritmi [vedi cap. 3]:

- HP dell'avversario
- HP del giocatore
- Score

- Vittoria dell'avversario
- Vittoria del giocatore
- Fine del gioco

Per il training è stata usata la piattaforma Google Colab che fornisce un notebook Jupyter e una macchina virtuale con GPU NVIDIA Tesla T4, unico limite è il tempo per il quale è attiva la macchina virtuale oltre il quale viene resettato l'ambiente con conseguente perdita dei dati relativi alla sessione, quindi non è possibile effettuare sessioni di training molto lunghe.

### 3 Algoritmi usati e parametri degli esperimenti

Sono stati selezionati i seguenti algoritmi dalla libreria Stable Baselines di OpenAI per il confronto:

- Deep Q-Learning (DQN)
- Advantage Actor Critic (A2C)
- Proximal Policy Optimization (PPO)
- Sample Efficient Actor-Critic with Experience Replay (ACER)

La scelta è ricaduta su questi quattro in quanto anche in altri lavori sono risultati essere i più efficienti, ad esclusione di DQN.

Il training è stato effettuato sulla rete convoluzionale di default definita in OpenAI Baselines, strutturata su tre strati convoluzionali e uno totalmente connesso, per almeno 1000000 di step in una partita arcade e alla difficoltà minima (una stella). Per ogni algoritmo sono state usate come reward function il punteggio di gioco, ottenuto tramite la lettura di una variabile e la differenza tra la vita restante ai giocatori, calcolata per mezzo di un apposito script.

Il personaggio scelto per l'addestramento dell'agente é stato **Ryu**.



Figura 4: Personaggio scelto

Il motivo della scelta è stata la semplicità e varietà di mosse adatte contro tutti i personaggi del gioco e soprattutto per l'assenza di mosse *Caricate* che si è riscontrato essere difficile apprendere con le risorse a nostra disposizione. Gli **stati** su cui effettuare i training sono stati creati appositamente da noi, suddivisi in due categorie:

- Singoli personaggi: abbiamo salvato uno stato con il nostro personaggio contro ogni singolo avversario.
- Arcade: lo stato parte dal primo livello della modalità arcade.

Per ognuna di queste categorie ne è stata creata una versione per i primi 4 livelli di difficoltà.

## 4 Esperimenti

Di seguito verranno mostrati i vari esperimenti effettuati e la relativa miglior strategia trovata per il training della rete.

### 4.1 DQN

L'agente addestrato con DQN sembra non essere in grado di comprendere le meccaniche di base del rendendo il training eccessivamente lungo, inoltre vengono fuori le limitazioni di colab che superato il tempo limite della sessione si ha la perdita di tutte le informazioni utili relative alla rete neurale. Nonostante le svariate sperimentazioni fatte l'agente addestrato tramite DQN non è mai riuscito ad avere prestazioni anche solo paragonabili con agli altri algoritmi presentati in questo lavoro.

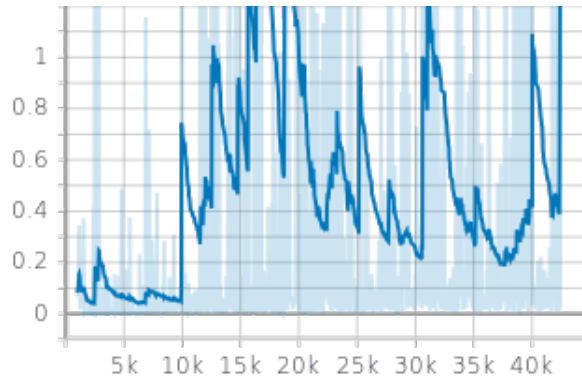


Figura 5: Loss function

## 4.2 A2C

A2C si è rivelato ottimo per questo problema, fornendo ottimi risultati in tempi relativamente brevi. L'agente è stato addestrato inizialmente utilizzando come reward function lo score ma, dopo diverse sperimentazioni, si notò che l'agente eseguiva azioni spesso casuali. Per risolvere il problema è stata usata come reward function la differenza tra la vita dei personaggi che, nel caso di A2C, ha migliorato fortemente le prestazioni e il tempo di training. L'agente addestrato tramite A2C è l'unico ad essere riuscito a completare la modalità arcade in 1, 2 stelle e a 3 stelle arrivando alla fine, ma senza completarlo. Siamo sicuri che con un ulteriore training sarebbe in grado di migliorare ulteriormente. La caratteristica principale che si nota nell'agente addestrato tramite A2C è l'utilizzo di **combo**, le combo sono serie di colpi e mosse speciali che devono essere eseguite con tempi molto stretti (intorno hai 2-3 frame), queste garantiscono un danno maggiore all'avversario e una volta subito il primo colpo si è incapaci di parare i rimanenti. Quindi l'agente ha imparato, quando possibile, a massimizzare il danno all'avversario tramite le combo.

## 4.3 ACER

In questo scenario l'agente viene addestrato per 1M di step al livello di difficoltà 1 e per 1M di step a livello di difficoltà 4 partendo sempre dal primo livello della modalità arcade. La reward function usata è il punteggio di gioco ottenuto dall'agente. La rete così addestrata è in grado di finire con difficoltà la modalità arcade ad una stella mentre a livelli superiori ha difficoltà nel superare i primi due combattimenti.

## 4.4 PPO

In questo scenario l'agente viene addestrato per 1M di step al livello di difficoltà 1 e per 1M di step a livello di difficoltà 4 partendo sempre dal primo livello della

modalità arcade. La reward function usata è il punteggio di gioco ottenuto dall'agente. La rete così addestrata é in grado di finire la modalità arcade ad una e due stelle mentre a livello di difficoltà 3 riesce a completare solo metà dei livelli della modalità arcade.

## 4.5 Risultati finali

Dai risultati ottenuti si osserva che utilizzando lo script come reward function il miglior algoritmo risulta essere A2C, mentre utilizzando lo score il migliore risulta essere PPO. Importante notare che l'agente, addestrato tramite A2C e script come reward function, è l'unico ad essere arrivato ai livelli finali della modalità arcade a livello di difficoltà 3 (e completato i precedenti livelli di difficoltà). Mentre agente addestrati con PPO e score come reward function si ferma a metà dell'arcade a 3 stelle di difficoltà. ACER non ha ottenuto buoni risultati sia con lo score come reward function, sia con lo script. Per quanto riguarda DQN non si è riuscito ad ottenere un agente in grado di completare alcun livello e pertanto neanche dei risultati confrontabili con gli altri algoritmi proposti.

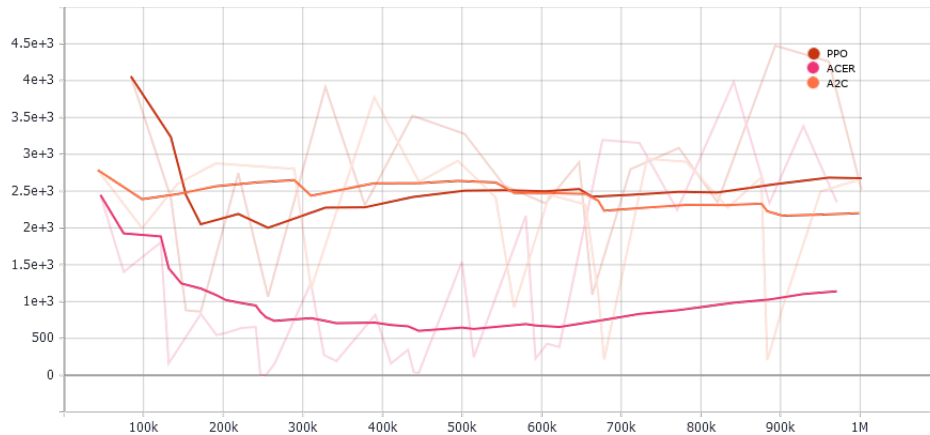


Figura 6: Reward ottenuti nella fase di addestramento ad una stella usando come reward function il punteggio di gioco

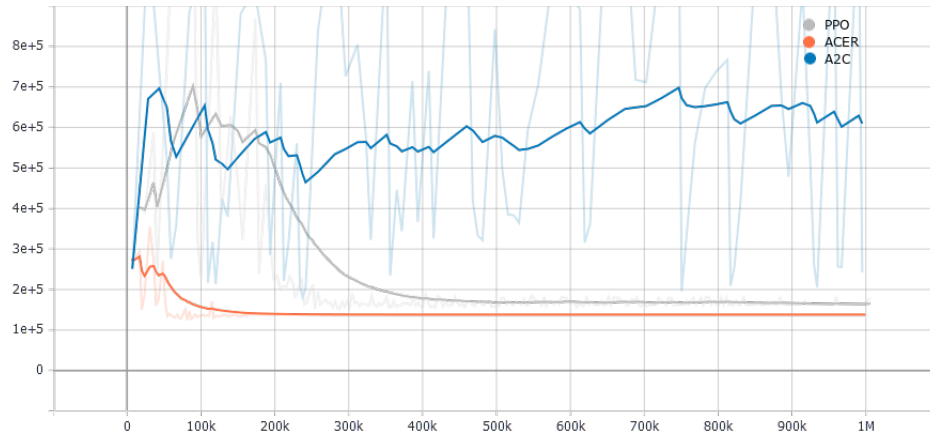


Figura 7: Reward ottenuti nella fase di addestramento ad una stella usando come reward function la differenza tra la vita dei personaggi

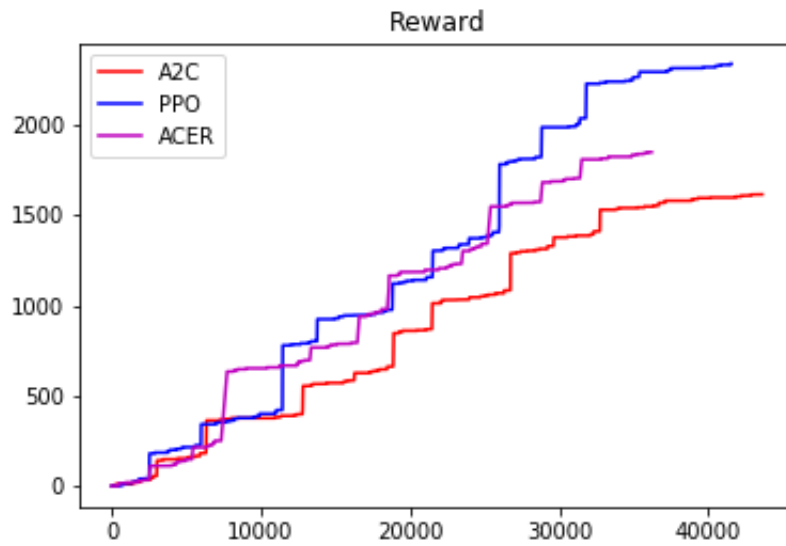


Figura 8: Reward ottenuti nella fase di test a tre stelle usando come reward function il punteggio di gioco, in ascisse il numero di azioni eseguite e nelle ordinate il reward



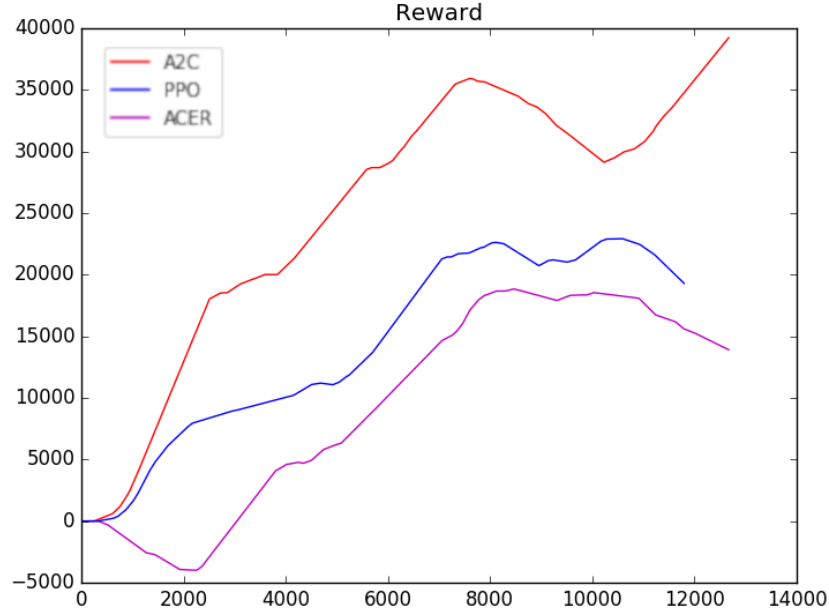


Figura 9: Reward ottenuti nella fase di test a tre stelle usando come reward function la differenza di vita, in ascisse il numero di azioni eseguite e nelle ordinate il reward

## 5 Osservazioni

Indipendentemente dal livello di difficoltà, in tutti gli scenari testati e step di addestramento, le prestazioni dell'agente giocando contro il suo stesso personaggio diminuivano sensibilmente portando spesso alla sua sconfitta. Si è pensato che questo fenomeno fosse dovuto oltre che al fatto che Ryu sia tra i personaggi più difficili da sconfiggere anche al fatto che l'agente si confonda, non riuscendo sempre a distinguere il personaggio che controlla da quello dell'avversario (i due personaggi si distinguono solo per i colori del vestiario [Fig. 10]).



Figura 10: Differenze dei costumi

## Riferimenti bibliografici

- [1] V Mnih, K Kavukcuoglu, D Silver, A Graves, I Antonoglou, D Wierstra, M Riedmiller. **Playing Atari with Deep Reinforcement Learning**. 2013
- [2] O Vinyals, D Silver, H van Hasselt, T Schaul, K Simonyan, S Petersen, S Gaffney, J Schrittwieser, J Agapiou, H Küttler, A Makhzani, M Yeo, A S Vezhnevets, P Georgiev, S Bartunov, T Ewalds, T Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, Rodney Tsing. **StarCraft II: A New Challenge for Reinforcement Learning**. 2017
- [3] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, Nando de Freitas. **Sample Efficient Actor-Critic with Experience Replay**. 2016
- [4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. **Proximal Policy Optimization Algorithms**. 2017
- [5] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. **Asynchronous Methods for Deep Reinforcement Learning**. 2016
- [6] Mendonça, Matheus and Bernardino, Heder and Fonseca Neto, Raul. **Simulating Human Behavior in Fighting Games Using Reinforcement Learning and Artificial Neural Networks**. 2015
- [7] OpenAI, <https://openai.com>
- [8] OpenAI Retro, <https://github.com/openai/retro>
- [9] Retro Contest, <https://openai.com/blog/retro-contest/>