



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea in Ingegneria Informatica

Deep learning Application

Reinforcement Learning

Edoardo Sarri

7173337

Giugno 2025

Indice

1	Introduzione	4
1.1	Main	4
1.2	Moduli Python	4
2	Analisi ambienti	5
2.1	Cart Pole	5
2.1.1	Descrizione	5
2.1.2	Obiettivo	5
2.2	Lunar Lander	5
2.2.1	Descrizione	6
2.2.2	Obiettivo	6
3	Cart Pole	7
3.1	Baseline	7
3.1.1	Nessuna baseline	7
3.1.2	Standardizzazione	8
3.1.3	Baseline state-value function	8
3.2	Iperparametri	9
4	Lunar Lander	12
4.1	Baseline	12
4.2	Architettura rete neurale	13
4.3	Step scheduler	13
4.4	Gradient clipping	13

Elenco delle figure

3.1	Eseperimenti senza baseline.	7
3.2	Eseperimenti con standardizzazione.	8
3.3	Somma dei reward con state-value function come baseline.	9
3.4	Confronti tra esperimenti.	9
3.5	Loss della policy con baseline state-value function.	10
3.6	Learning rate di circa 1e-5.	11
3.7	Sweep con gamma e lr in un ampio range.	11
3.8	Confronti tra esperimenti.	11
4.2	Varie architetture.	13
4.3	Scheduler StepLR.	14
4.4	Risultati con gradient clipping.	14

1 Introduzione

In questo primo capitolo viene spiegata l'organizzazione del progetto.

1.1 Main

Il main del progetto è rappresentato dal file `main.py`. Inizialmente era stato valutato l'uso di un Jupyter notebook, ma il kernel di Jupyter non funziona bene con gli ambienti di Gymnasium, in particolare con quelli che usano ambienti grafici come Pygame. L'addestramento dell'agente procedeva senza troppi problemi, ma quanto si andava a visualizzare l'agente, Pygame andava in crash.

Questo documento è pesato per non far eseguire il main a chi è interessato solo ai risultati. Ogni risultato loggato su W&B [4] (framework usato durante tutto il progetto per tracciare gli esperimenti) o esperimento eseguito è riassunto in questa relazione. Con questa idea i vari file `.py` sono utili non per capire cosa è stato fatto, ma come è stato fatto.

1.2 Moduli Python

Le funzioni chiamate in `main.py` sono importate da più moduli:

- `reinforce.py`
Contiene la funzione che implementa l'algoritmo REIFORCE.
- `myModel.py`
Contiene le reti neurali usate nel processo di addestramento dell'agente.
- `core.py`
Contiene le funzioni fondamentali richiamate dall'algoritmo REIFORCE
- `sweep.py`
Contiene il codice usato per implementare lo sweep per ricercare i migliori iperparametri in Cart Pole.
- `utility.py`
Contiene le funzioni ausiliarie utili nel progetto.

2 Analisi ambienti

Gli ambienti in cui si muove l'agente sono definiti in Gymnasium [2], una libreria Python open source ideata per sviluppare algoritmi di Reinforcement learning.

2.1 Cart Pole

Il primo ambiente utilizzato è stato Cart Pole [1]. L'obiettivo dell'agente in questo contesto è quello di quello di tenere in equilibrio una barra posta su un carrello e soggetta alla forza di gravità.

Ci sono due versioni di Cart Pole. Quella utilizzata in questo progetto è CartPole-v1.

2.1.1 Descrizione

L'action space dell'ambiente è composto da due azioni: muovere il carrello verso sinistra è rappresentato un uno zero; muoverlo verso destra con un 1.

L'observation space è definito da quattro variabili: la posizione e la velocità del carrello, l'angolo e la velocità angolare della barra.

Per ogni azione eseguita all'interno dell'episodio, l'agente ottiene una reward di +1.

2.1.2 Obiettivo

L'episodio termina quando la barra supera i $\pm 12^\circ$ di inclinazione o se la posizione del carrello supera un range di $\pm 2,4$. Inoltre l'episodio finisce in maniera automatica quando vengono superate le 500 iterazioni.

L'obiettivo è raggiungere una reward stabile di almeno 475 punti per episodio.

2.2 Lunar Lander

Il secondo ambiente utilizzato è stato Cart Pole [3]. L'agente in questo caso è una navicella, dotata di motore principale e laterale, che deve atterrare in un preciso spazio nell'ambiente.

Ci sono due versioni di Lunar Lander, una discreta e una continua: nella prima i motori sono sempre accesi alla massima potenza; nella seconda abbiamo un valore che ci indica la potenza emessa. La versione utilizzata in questo progetto è quella discreta.

2.2.1 Descrizione

L'action space dell'ambiente è composto da quattro azioni: non fare niente, attivare il motore centrale, quello destro o quello sinistro.

L'oservation space è definito da otto variabili: le coordinate della navicella in due dimensioni, la sua velocità lineare in due dimensioni, il suo angolo, la velocità angolare e due booleani che indicano l'appoggio di una gamba sul terreno.

Le reward possono diminuire e aumentare all'interno di un episodio a seconda se l'agente è più o meno vicino rispetto alla zona di atterraggio, alla velocità dell'agente e alla sua inclinazione. Inoltre ottiene 10 punti per ogni gamba che poggia a terra, diminuisce di 0,03 punti per ogni frame dove è in azione un motore laterale e di 0,3 per ogni frame in cui è in azione quello principale; riceve ancora ± 100 punti per un atterraggio o un crash.

È interessante notare che quando si importa l'ambiente si possono settare la velocità del vento, al forza di gravità e il rumore sulla potenza dei motori.

2.2.2 Obiettivo

L'episodio termina quando la navicella ha un crush o se esce dai limiti del box che la contiene.

L'obiettivo è raggiungere una reward stabile di almeno 200 punti per episodio.

3 Cart Pole

In questo capitolo cercheremo di risolvere l'ambiente Cart Pole analizzato nella Sezione 2.1.

Come criterio per verificare l'obiettivo si è considerato il raggiungimento in modo consistente di una returns per episodio di 475 punti. Il grafico che ci farà osservare questo comportamento è quello che mette in evidenza la somma delle reward per ogni epoca; vista l'elevato rumore è stato rappresentato con un Time Weighted EMA con un fattore di 0,95. Questa verifica poteva essere atta anche osservando la lunghezza dell'episodio, ma in questo modo si è generalizzato anche per altri ambienti, come quello nel prossimo capitolo.

3.1 Baseline

In questa sezione si sono valutate più baseline. Siamo partiti da un'implementazione di REINFORCE senza baseline, poi usando una standizzazione delle returns e infinite con una stima della state-value function.

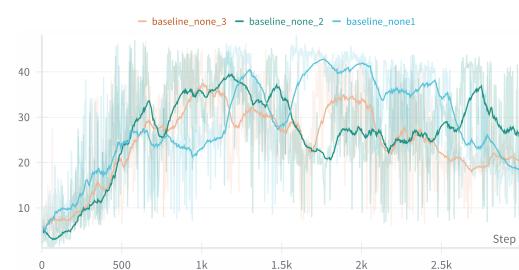
3.1.1 Nessuna baseline

I primi esperimenti che sono stati eseguiti sono senza una baseline.

Come si può osservare dalla Figura 3.1a, senza l'utilizzo di una baseline l'apprendimento è molto irregolare e non inoltre con soli 3000 episodi non si riesce a risolvere l'ambiente; l'unico esperimento che si avvicina a risolvere l'ambiente è stato il primo. Inoltre tre iterazioni portano a tre risultati abbastanza diversi.



(a) Somma dei reward



(b) Loss

Figura 3.1: Esempi di esperimenti senza baseline.

Un'osservazione interessante è quella relativa alla loss. Osservando la Figura 3.1b, notiamo che non è una classica loss durante un apprendimento: siamo infatti abituati

a vedere una loss che diminuisce nel tempo; in questo caso abbiamo una loss molto irregolare e che oscilla molto.

Uno dei motivi di questo andamento può essere la natura molto stocastica dell'algoritmo: in ogni episodio i returns sono diversi, ma la loss dipende da questi ritorni. La loss inoltre non è una funzione fissa, ma dipende dalla policy; la policy viene aggiornata dopo ogni episodio e con essa cambia anche la loss.

3.1.2 Standardizzazione

Per migliorare la stabilità dell'apprendimento si è usata la standardizzazione dei returns all'interno di ogni episodio.

Come si può osservare dalla Figura 3.2a con la standardizzazione de returns l'apprendimento è molto più regolare; inoltre con circa 1500 episodi l'ambiente è risolto, infatti la somma delle rewards per epoca è abbastanza costantemente sopra i 475.

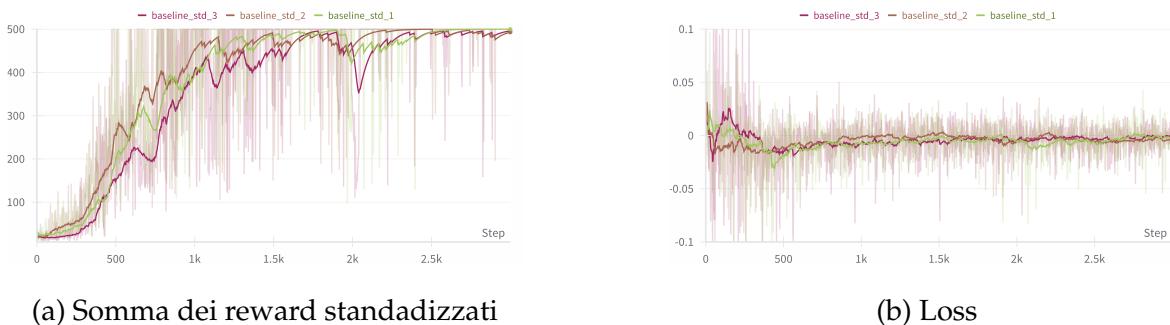


Figura 3.2: Esempi di esperimenti con standardizzazione.

Osservando la loss in Figura 3.2b notiamo qualcosa di molto particolare: il valore della loss è strettamente compreso in un intervallo ± 1 ; questo avviene appunto a causa della standardizzazione dei returns.

3.1.3 Baseline state-value function

In questa sezione è usata come baseline la state value function. Per stimare questa funzione si è usata una rete neurale, con la stessa architettura della rete neurale che stima la policy.

Grazie a questa soluzione i vari esperimenti sono più prevedibili, nel senso che c'è meno differenza l'uno dall'altro, come si vede in Figura 3.3.

Osserviamo invece la differenza tra uno di questi esperimenti e uno con la standardizzazione relativamente alla somma delle rewards per epoca. In Figura 3.4a, si nota come questo metodo sia più stabile e abbia una velocità di convergenza migliore.

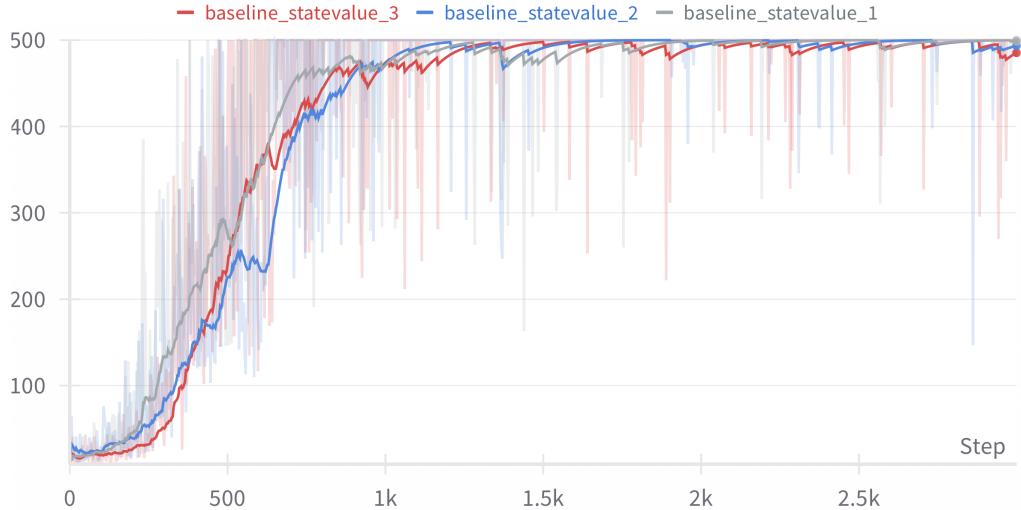


Figura 3.3: Somma dei reward con state-value function come baseline.

Un aspetto interessante è invece il confronto delle varie tecniche relativamente al tempo di esecuzione. I risultati in Figura 3.4b sono prevedibili: la tecnica senza baseline è mediamente la più veloce, mentre usare un'approssimazione della state-value function è il metodo più lento.

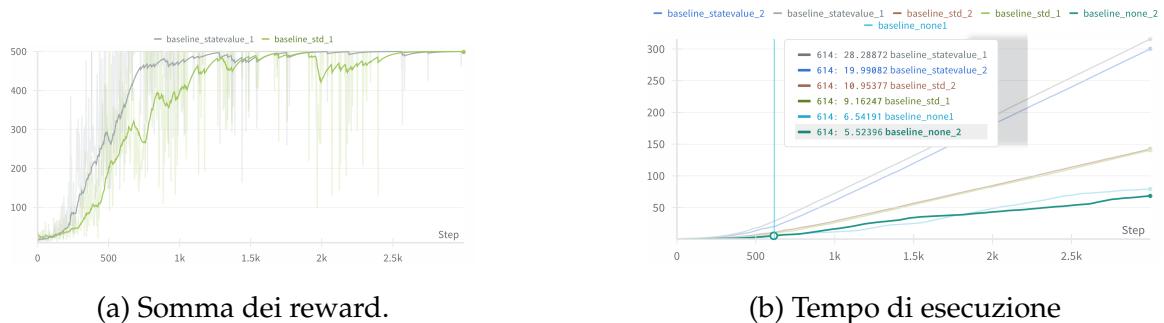


Figura 3.4: Confronti tra esperimenti.

Possiamo osservare la Figura 3.5 per capire cosa succede alla loss della rete neurale che approssima la policy. Possiamo osservare che all'inizio il comportamento è lo stesso della loss senza l'utilizzo di una baseline, ma con l'avanzare degli episodi la loss ha una convergenza.

3.2 Iperparametri

Fino ad adesso il fattore di sconto γ e il learning rate sono stati impostati di default a 0,99 e a 1e-3. In questo apitolo cerchiamo di capire cosa cambia se li variamo: il learning rate può variare tra 1e-2 e 1e-5; gamma varia tra 0,9 e 0,999. Gli altri apraemtri

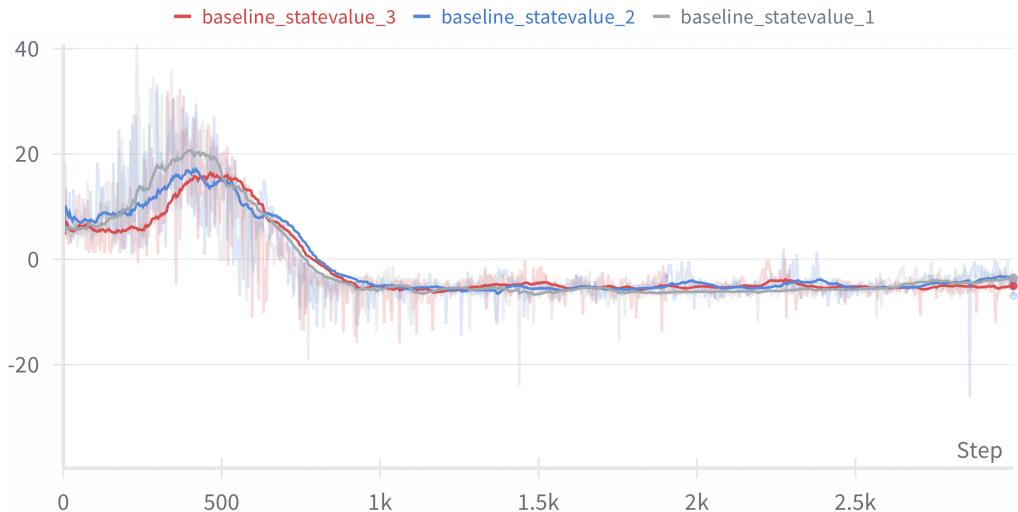


Figura 3.5: Loss della policy con baseline state-value function.

utilizzati sono stati: 3000 episodi, la baseline è stata l'approssimazione della state-value function.

Ci concentriamo solo sul grafico che esprime la somma dei returns per epoca, visto che è quello che ci interessa maggiormente. Per una migliore visualizzazione in questo caso il fattore del Time Weighted EMA è stato impostato a 0,99

La Figura 3.6 rappresenta tre esperimenti con un learning rata basso, circa $1e-5$; γ invece varia tra 0,93 e 0,97. Si può osservare come l'agente non stia imparando nulla al passare delle epoche.

Se osserviamo in Figura 3.7 tutti gli altri esperimenti eseguiti vediamo che ci sono dei problemi: non abbiamo ottenuto gli stessi risultati degli esperimenti con learning rate egamma standard. I migliori sono comunque quelli con un gamma maggiore. Da questo possiamo capire che l'algoritmo REINFORCE è molto suscettibile a questi iperparametri.

Per cercare di migliorare la situazione ho diminuito il range di γ , tenendolo tra 0,95 e 0,99, e quello del learning rate, che varia tra $1e-2$ e $1e-4$.

Come si può osservare dalla Figura 3.8a avere un γ più basso può portare a non risolvere l'ambiente.

La situazione migliore è quando abbiamo un fatto di sconto γ molto alto. Quello che osserviamo dai grafici in Figura 3.8b è che avere un learning rate più alto porta a risolvere l'ambiente molto velocemente, ma dopo molti episodi i returns peggiorano nuovamente; avere invece un learning rate più basso porta a risolvere l'ambiente più lentamente ma in modo più stabile.

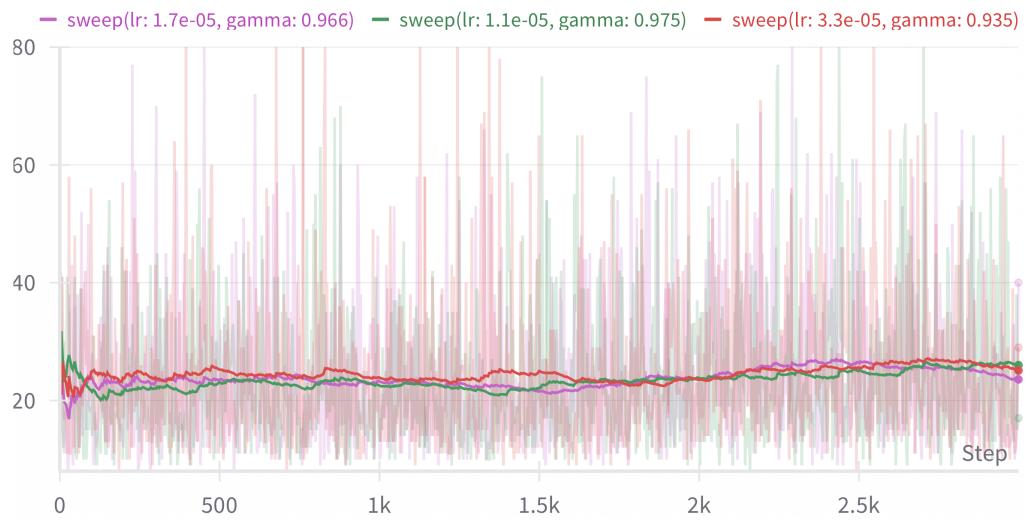


Figura 3.6: Learning rate di circa $1e-5$.

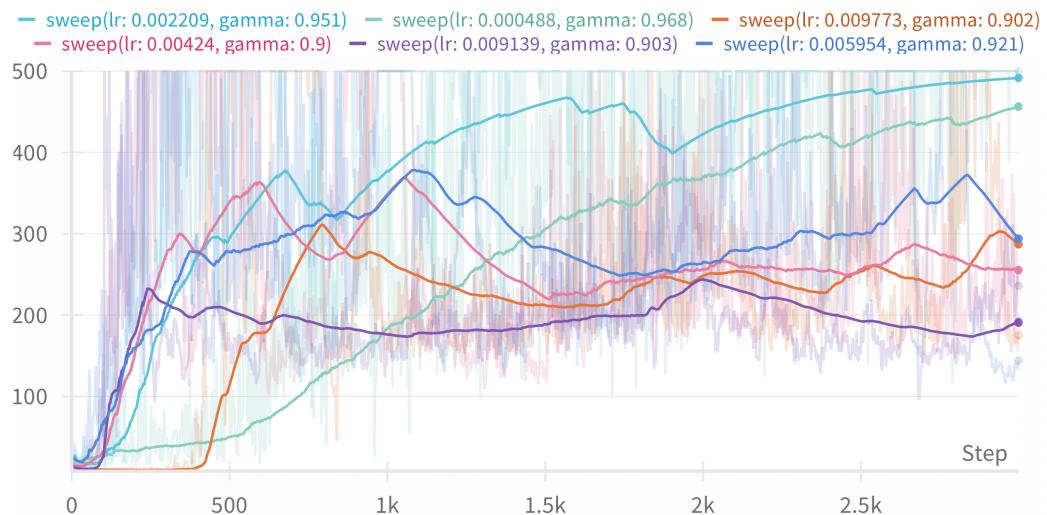


Figura 3.7: Sweep con gamma e lr in un ampio range.

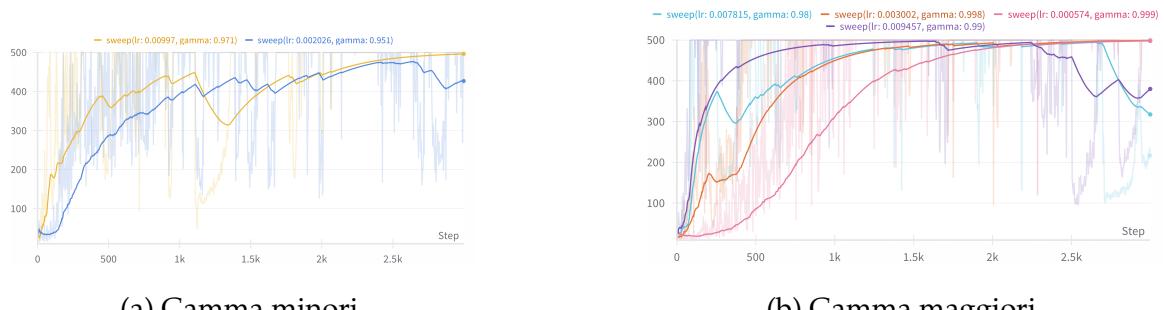


Figura 3.8: Confronti tra esperimenti.

4 Lunar Lander

In questo capitolo cercheremo di risolvere l'ambiente Lunar Lander analizzato nella Sezione 2.2.

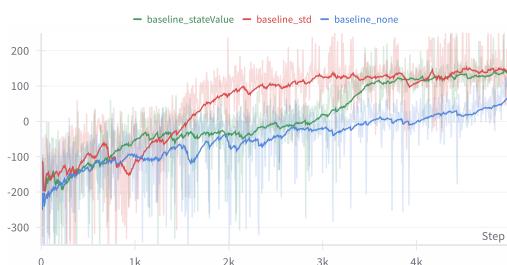
Come criterio per verificare l'obiettivo si è considerato il raggiungimento in modo consistente di una returns per episodio di 200 punti. Il grafico che ci farà osservare questo comportamento è quello che mette in evidenza la somma delle reward per ogni epoca; vista l'elevato rumore è stato rappresentato con un Time Weighted EMA con un fattore di 0,95.

4.1 Baseline

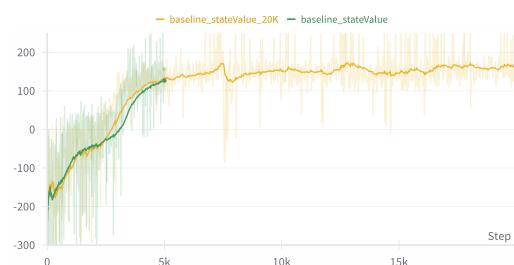
Per cercare di risolvere l'ambiente utilizzando gli stessi metodi di Cart Pole, sono stati eseguiti tre esperimenti usando le tre baseline definite nel Capitolo 3: senza baseline, con la standardizzazione e con l'approssimazione della state-value function tramite una rete neurale. Siccome l'ambiente è più complesso rispetto a Cart Pole, ho deciso di addestrare l'agente per 5000 episodi, contro le 3000 usate in precedenza.

Come si può osservare dai risultati in Figura 4.1a, mentre risolvere Cart Pole era banale, adesso il compito è più complesso. Si può notare che la soluzione senza baseline fa molta fatica, e in praticamente nessun episodio è stato raggiunta una somma dei rewards superiore a 200. Con la standardizzazione e la state-value function la situazione migliora: ci sono degli episodi in cui l'ambiente è risolto, ma questo succede in modo non stabile e quindi il risultato non ci soddisfa. Usare più episodi (20K) non migliora l'addestramento, come si vede in Figura 4.1b; dobbiamo quindi cercare di cambiare un po' le cose per cercare una soluzione più soddisfacente.

Come osservato nel capitolo precedente possiamo notare come la soluzione con la baseline che approssima la state-value function è più stabile.



(a) Tre baseline.



(b) Baseline con state-value function.

4.2 Architettura rete neaurale

Nel seguito i vari esperimenti sono stati eseguiti con la state-value function baseline. In questo capitolo vediamo come è stata cambiata la rete che approssima la policy e la state-value function; ricordiamo che queste due hanno la stessa architettura.

Sono state provate varie architetture: la profondità è di tre layers interni tutti con lo stesso numero di neuroni; in un caso 32, in un altro 64 e nell'ultimo 128. Visto il contesto molto stocastico, sono stati fatti tre esperimenti per ogni architettura, ognuno dei quali è stato eseguito per 10000 episodi.

Osserviamo i vari esperimenti fatti. Come si può notare dalle Figure 4.2, all'aumentare della complessità della rete aumenta anche la differenza di risultati da un esperimento all'altro. Si può anche osservare che una rete più complessa sembra essere più promettente rispetto a una più semplice nell'arrivare al risultato, cioè ai 200 punti per episodio.

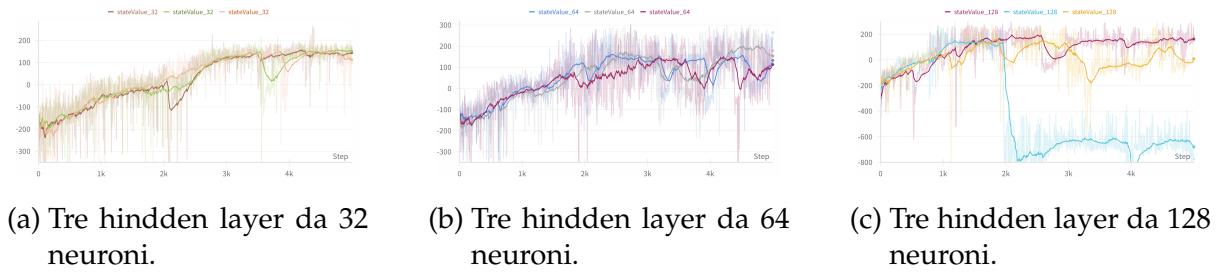


Figura 4.2: Varie architetture.

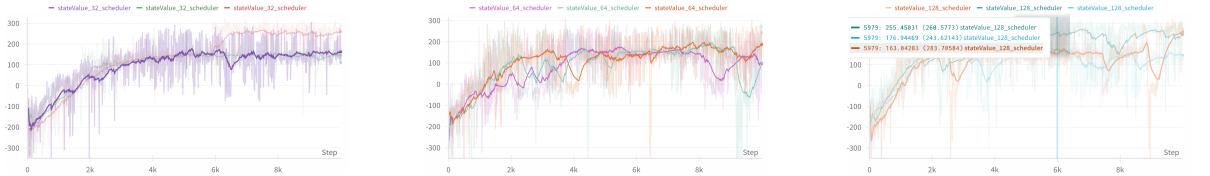
4.3 Step scheduler

Per cercare di migliorare la stabilità dell'addestramento è stato introdotto uno scheduler per i due ottimizzatori delle reti che approssimano la policy e la state-value function. Lo scheduler in questione è stato più semplici, cioè StepLR: ogni 1500 episodi il learning rate è decrementato di un fattore di 0,9.

I risultati ottenuti sono quelli mostrati in Figura 4.3. Come si può osservare la stabilità è migliorata leggermente, soprattutto per il modello più complesso. Nonostante questo i risultati non sono del tutto soddisfacenti per quanto riguarda le performance.

4.4 Gradient clipping

Il metodo che forse ha dato il miglioramento più evidente è stato quello di limitare la norma dei gradienti per evitare il fenomeno dell'exploding gradient, molto comune nel reinforcement learning.



(a) Tre hidden layer da 32 neuroni.

(b) Tre hidden layer da 64 neuroni.

(c) Tre hidden layer da 128 neuroni.

Figura 4.3: Scheduler StepLR.

Sono state testati tre valori per il massimo delle norme: 2, 5 e 10. Come si può notare dalla Figura 4.4, i risultati migliori li abbiamo quando cercavamo di limitare maggiormente la norma dei gradienti; questo ci fa capire quanto sia importante il fenomeno dell'exploding gradient in questo contesto.

Con questo risultato si può dire, anche osservando al visualizzazione, che l'ambiente è risolto.

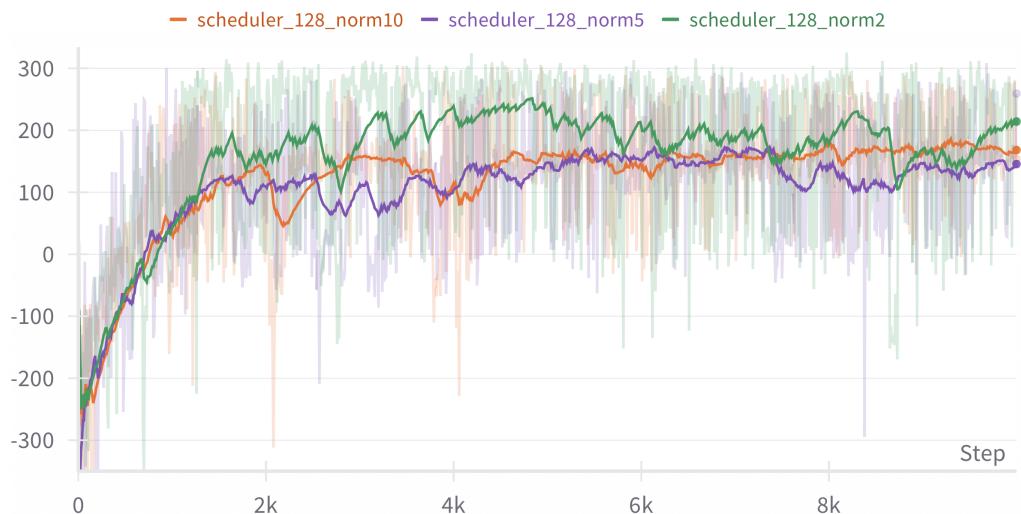


Figura 4.4: Risultati con gradient clipping.

Bibliografia

- [1] *Cart Pole.* https://www.gymlibrary.dev/environments/classic_control/cart_pole/.
- [2] *Gym.* <https://www.gymlibrary.dev/index.html>.
- [3] *Lunar Lander.* https://gymnasium.farama.org/environments/box2d/lunar_lander/.
- [4] *Weight & Biases.* <https://wandb.ai/site/>.