UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA
DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Intelligenza Artificiale

Multi-Agent Reinforcement Learning con PPO: l'esempio hide-and-seek di OpenAI

Con la supervisione di:
Prof. E. A. Gerevini
Dott. N. Rossetti

Autore:
Claudio Metelli

Matricola n.
745246

Anno Accademico 2023/2024

Intelligenza Artificiale - Hide and Seek

Claudio Metelli

April 2024

1 Introduzione

L'ambito del reinforcement learning, all'interno di quello ben più ampio dell'intelligenza artificiale, offre numerose sottocategorie, tra le quali il multi-agent reinforcement learning, di cui l'articolo "Emergent Tool Use From Multi-Agent Autocurricula" [2] di OpenAI[1] è un noto esempio.

Esso vuole evidenziare come attraverso la multi-agent competition, algoritmi standard di reinforcement learning, ed un esempio semplice come il gioco del nascondino, gli agenti creino un autocurriculum, ovvero dei cambiamenti netti nell'interazione ambientale da parte degli agenti, tali da distinguere diverse fasi, che portano a miglioramenti radicali. In questo esempio particolare l'autocurriculum degli agenti si divide in 6 fasi diverse, che verranno illustrate successivamente.

Lo scopo di questo articolo è di illustrare maggiormente nel dettaglio la fase di apprendimento e le tecniche utilizzate, al fine di una più chiara comprensione dei modelli di ottimizzazione in ambito RL (Reinforcement Learning) ed in particolare di quelli utilizzati nel suddetto esperimento.

[1] https://openai.com

2 Policy Optimization

La policy optimization è una parte fondamentale del processo di apprendimento nel reinforcement learning.

Partendo dal presupposto che l'obiettivo sia quello di trovare una policy ottimale alla risoluzione di un determinato task, è necessario trovare un modo per raggiungere tale obiettivo: uno degli approcci più comuni è infatti l'utilizzo di un algoritmo di policy optimization.
Questa tipologia di metodi prova a definire quale sia la migliore azione da compiere in un dato stato attraverso una policy $\pi_\theta(a|s)$, la quale è definita da una serie di parametri θ (ad esempio i pesi di una rete neurale).

Successivamente l'obiettivo diventa quello di massimizzare i reward attesi di tale policy π_θ attraverso un problema di ottimizzazione, che massimizza una funzione $J(\theta)$; tale ottimizzazione viene quasi sempre eseguita in base alla policy, ovvero ogni epoca utilizza solo i dati raccolti mentre si agisce secondo la versione più recente della politica. Vediamo alcuni metodi nello specifico.

2.1 Policy Gradient

In un algoritmo RL, affinchè si trovi una policy ottimale, è necessario cercare di massimizzare i reward attesi.

La funzione obiettivo del problema di massimizzazione è dunque:

$$\max J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(a_t|s_t)\right] \quad (1)$$

dove R è la funzione di reward di una certa azione a ed un certo stato s al tempo t, mentre T è l'orizzonte temporale.

Per arrivare al set di parametri ottimale θ^*, un approccio comune in ambito machine learning, è il gradient descent, che attraverso il calcolo del gradiente, aggiorna continuamente il valore θ fino a convergenza. In particolare:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \quad (2)$$

con α parametro di learning rate.

In questa operazione, naturalmente, la parte più complessa è il ricavo del gradiente. Come illustrato nell'articolo [3] esso è ricavabile attraverso una serie di passaggi, ottenendo il risultato:

$$\nabla J(\theta_t) = \mathbb{E}_{\pi_\theta}\left[R(\tau)\sum_{t=0}^{T-1} \nabla \log \pi_\theta(a_t|s_t)\right] \quad (3)$$

con τ traiettoria della policy. Il continuo aggiornamento dei parametri θ porta dunque ad una, almeno teorica, convergenza.

La metodologia policy gradient è una delle più grezze, che porta tra i vantaggi una semplicità di comprensione ed implementazione ma, al tempo stesso, annovera tra i problemi una eccessiva sensibilità. Infatti, essa è molto sensibile ai cambiamenti di direzione, senza alcun controllo su di essi, e ciò può portare adaggiornamenti eccessivi della policy che dunque ne implicano la correttezza. Per questa ragione, vengono utilizzate metodologie più sofisticate, come la Proximal Policy Optimization (PPO) [8], la quale necessita però di alcuni prerequisiti per essere chiara.

2.2 Advantage Function

Nella sottosezione precedente abbiamo fornito una stima del gradiente (3), che chiamiamo ora g. In particolare, una più generale formulazione di tale stima è:

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla \log \pi_\theta(a_t|s_t)\right]. \quad (4)$$

Tale formulazione può sostituire Ψ_t con differenti espressioni che forniscono diverse stime, tra cui $R(\tau)$ come già visto, oppure:

- $Q^\pi(s_t, a_t)$: state-action value function
- $A^\pi(s_t, a_t)$: advantage function

Utilizzando le definizioni:

$\mathrm{V}^\pi(s_t) := \mathbb{E}s_{t+1:\infty}, a_{t:\infty}\left[\sum_{l=0}^{\infty} r_{t+l}\right]$

$Q^\pi(s_t, a_t) := \mathbb{E}s_{t+1:\infty}, a_{t+1:\infty}\left[\sum_{l=0}^{\infty} r_{t+l}\right]$

$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t).$

Generalmente, la scelta $\Psi_t = A^\pi(s_t, a_t)$ porta a una varianza possibile minima (anche se, nella pratica, la funzione di vantaggio non è conosciuta e deve essere stimata). Quanto affermato si può intuire attraverso il seguente ragionamento: un passo nella direzione del gradiente della politica dovrebbe aumentare la probabilità delle azioni migliori della media e diminuire la probabilità delle azioni peggiori della media; la funzione di vantaggio, definita come: $A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t)$, misura per l'appunto se l'azione è migliore o peggiore rispetto al comportamento predefinito della policy.

2.3 Generalized Advantage Estimation

Vista l'intuizione precedente, risulta ora necessaria la stima della advantage function. Uno dei metodi più efficienti in questo caso è Generalized Advantage Estimation (GAE) [7]. Nell'articolo citato viene stimata introducendo il termine:

$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$

il quale introduce anche il valore di discount γ; esso è un estimatore del vantaggio della singola azione a_t. Attraverso ulteriori passaggi illustrati nello stesso articolo, si arriva ad un estimatore della funzione di vantaggio GAE:

$$\hat{A}_t^{GAE} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V. \tag{5}$$

Si arriva infine alla stima del gradiente finale (con discount):

$$g^\gamma \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V\right] \tag{6}$$

2.4 Proximal Policy Optimization

Definiti ora alcuni concetti e strumenti chiave, è possibile introdurre la Proximal Policy Optimization (PPO). Introduciamo innanzitutto un rateo di probabilità:

$\mathrm{r}_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\,old}(a_t|s_t)}$.

Abbiamo poi una nuova funzione obiettivo:

$$\mathrm{L}^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

con ϵ iperparametro, solitamente $\epsilon = 0.2$. Il primo termine del minimo deriva dalla funzione obiettivo utilizzata in TRPO [6], un altro metodo di policy optimization, mentre il secondo termine modifica la funzione obiettivo tagliando il rateo di probabilità, evitando di spostare r_t al di fuori dell'intervallo $[1-\epsilon, 1+\epsilon]$.

Utilizzare il minimo tra i due termini porta ad avere un vincolo pessimistico, ignorando la variazione del rateo di probabilità solo quando migliora l'obiettivo, ed includendola quando peggiora l'obiettivo. Questa operazione vuole assicurarsi che l'agente compia un "piccolo passo", cioè la mossa più sicura possibile.

La PPO presenta numerosi vantaggi, ad esempio non è eccessivamente costosa ed è efficiente anche su problemi di larga scala, è molto stabile, non presenta grandi difficoltà nel tuning dei parametri; infine, non ha necessità di grandi quantità di dati per l'addestramento, in quanto non avendo la tendenza a seguirli eccessivamente, grazie alla clipping function, l'addestramento risulta più lineare portando comunque a buoni risultati.

Questo strumento risulta di vitale importanza in esperimenti come quello di hide-and-seek che viene illustrato nella sezione successiva

3 Hide-and-seek

L'articolo alla base di questa relazione, riguarda il reinforcement learning attraverso la competizione tra agenti, prendendo come esempio il gioco del nascondino. Questa simulazione vuole evidenziare l'autocurriculum creatosi grazie al processo di apprendimento, e le sue varie fasi. Iniziamo definendo l'ambiente all'interno del quale si svolge l'attività.

Nel gioco di hide-and-seek abbiamo due tipi di agenti in team da 1 a 3 agenti ciascuno: gli *hiders*, coloro che si nascondono, ed i *seekers*, coloro che cercano. Il task principale dei primi è evitare il campo visivo dei secondi, che hanno l'obiettivo di tenere gli *hiders* nel loro campo visivo più a lungo possibile. I reward per questi task sono dati al team e non al singolo agente, e consistono per coloro che si nascondono in $+1$ per ogni timestep (una epoca sono 240 timesteps) per cui non sono nel campo visivo di un cercatore e in -1 se sono all'interno di tale campo visivo; per i *seekers* vale il contrario.
Sparsi nell'ambiente vi sono inoltre oggetti che gli agenti possono spostare oppure bloccare, come rampe o scatole. Infine, prima che inizi la fase di ricerca dei *seekers*, gli altri agenti hanno a disposizione un lasso di tempo per interagire con l'ambiente, dunque nascondendosi oppure muovendo oggetti.

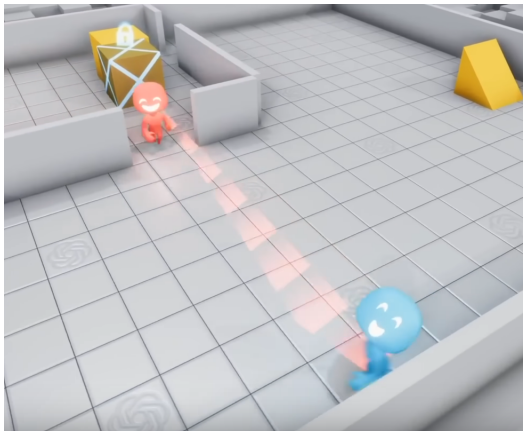

Figura 1: *Seeker* in rosso e *hider* in blu, in giallo gli oggetti movibili e /o bloccabili

Un esempio di nascondino con questi elementi è visibile in figura 1.
Ogni agente può scegliere se *muoversi* lungo l'asse x e/o y, girandosi lungo l'asse z. Vi sono inoltre due azioni binarie per *bloccare* e *spostare* oggetti.

3.1 Policy Optimization

Le policy degli agenti si compongono di due diverse reti neurali (LSTM), ognuna avente i propri parametri: una per la predizione dei reward ed una per decidere le azioni da compiere. La policy optimization è effettuata tramite gli strumenti già descritti: GAE e PPO.

In particolare, si che la stima della advantage function tramite GAE è:

$\hat{A}_t^H = \sum_{l=0}^{H} (\gamma\lambda)^l \delta_{t+l}^V$.

Rispetto alla formula 5, invece di ∞, computazionalmente impraticabile, si ha una H, ovvero la lunghezza dell'orizzonte, pari a 160 (su un totale di 240 timesteps per epoca), ed aggiungendo un ulteriore fattore di discount λ per equilibrare il bias-variance tradeoff.

Per quanto riguarda la PPO, la funzione obiettivo rimane quella descritta nella sezione 2.4.

3.2 Autocurriculum

Questa struttura matematica, insieme al contesto nel quale è applicata, porta ad un autocurriculum, ovvero cambiamenti netti nel- l'interazione ambientale da parte degli agenti, tali da distinguere diverse fasi, che portano a miglioramenti radicali. In particolare, si evidenziano sei fasi principali, che dividiamo anche per numero di episodi.

(a)≈0: Inizialmente, si ha una fase in cui gli inseguitori inseguono, mentre coloro che si nascondono scappano.

(b)≈25 milioni: Successivamente, gli hiders imparano ad usare gli oggetti

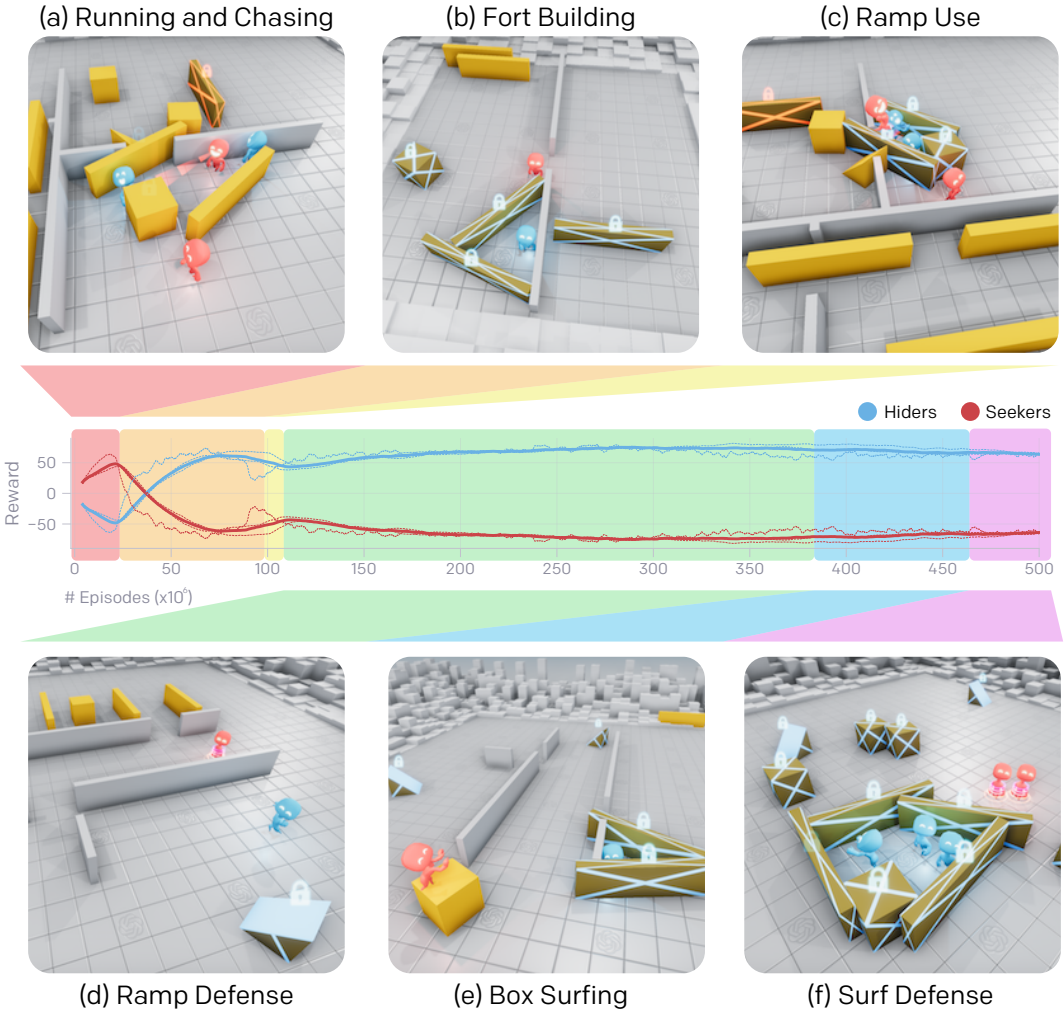

Figura 2: Descrizione grafica dell'autocurriculum

nel periodo di conta iniziale: li spostano così da chiudere stanze o creare ripari sicuri nei quali non si può essere visti; essi hanno imparato ad utilizzare strumenti.

(c)≈100 milioni: Anche i seekers imparano ad utilizzare strumenti: adoperano le rampe per scavalcare i ripari creati.

(d)≈110 milioni: Gli hiders imparano a privare i seekers delle rampe, portandole all'interno dei loro ripari e bloccandole.

(e)≈380 milioni: Una delle fasi più sorprendenti è questa: i cercatori imparano a portare una scatola fino al bordo dell'area di gioco dove i nasconditori hanno bloccato le rampe; essi usano quindi la rampa per spostarsi sopra la scatola e farla navigare fino al rifugio dei nasconditori. Questo è possibile in quanto l'a-

zione di movimento degli agenti consente loro di applicare una forza su se stessi, indipendentemente dal fatto che siano a terra o meno; se lo fanno afferrando la scatola sotto di loro, la scatola si muoverà con loro mentre sono sopra di essa, e ciò non era stato inizialmente previsto dagli sviluppatori.

(f)≈460 milioni: Di conseguenza, gli hiders imparano a bloccare le scatole per evitare che i seekers possano utilizzarle

Si può notare come gli agenti imparino a collaborare tra di essi in pressochè ogni fase: ciò è dovuto ai reward, che sono dati al team e non al singolo agente; proprio per questo, i team di hiders più numerosi (due oppure tre agenti, rispetto al singolo) riescono a bloccare molti più oggetti, indicando inoltre come team più numerosi riescano in una migliore divisione dei compiti.

Un aspetto importante per la convergenza verso il risultato dell'autocurriculum è inoltre la dimensione del batch: batch da 64000 episodi riescono a raggiungere il quarto stadio in circa 34 ore e 132 milioni di episodi circa; mentre con un batch size di 128 si raggiunge lo stesso risultato in 155 milioni di episodi e 20 ore circa. La differenza temporale è dovuta alla signifcativa quantità di step di ottimizzazione. Con batch di minore dimensione (8000, 16000) non si ha convergenza.

3.3 Valutazione dei risultati

Per la valutazione dei risultati si è scelto di utilizzare il trasferimento a una serie di compiti specifici del dominio per

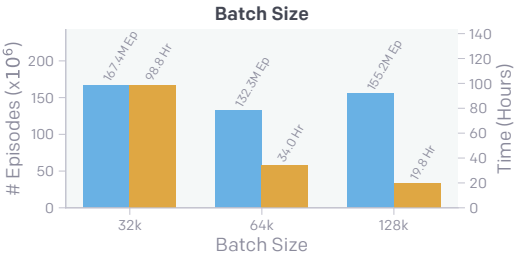

Figura 3: Numero di episodi per batch (blu) ed ore per il raggiungimento dello stage 4(arancione) in base al batch (32k, 64k, 128k).

valutare le capacità acquisite dall'agente, utilizzando 5 suite di benchmark. Esse utilizzano lo stesso spazio di azione e tipi di oggetti dell'ambiente del nascondino, e si dividono in due categorie: task cognitivi e di memorizzazione, e task di manipolazione.

Task cognitivi e di memorizzazione

Nel compito di *conteggio degli oggetti*, si vuole misurare se gli agenti hanno un senso di permanenza degli oggetti; l'agente viene bloccato in una posizione e osserva come sei scatole si spostano a caso verso destra o verso sinistra, dove alla fine vengono oscurate da un muro. Gli viene poi chiesto di prevedere quante scatole sono andate da una parte e dall'altra per molti istanti dopo che tutte le scatole sono scomparse.

In *blocca e ritorna* vogliamo misurare se l'agente è in grado di ricordare la sua posizione originale mentre esegue un nuovo compito. L'agente deve navigare in un ambiente con 6 stanze casuali e 1 scatola, bloccare la scatola e tornare alla posizione di partenza.

In *blocco sequenziale* ci sono 4 scatole posizionate casualmente in 3 stanze ca-

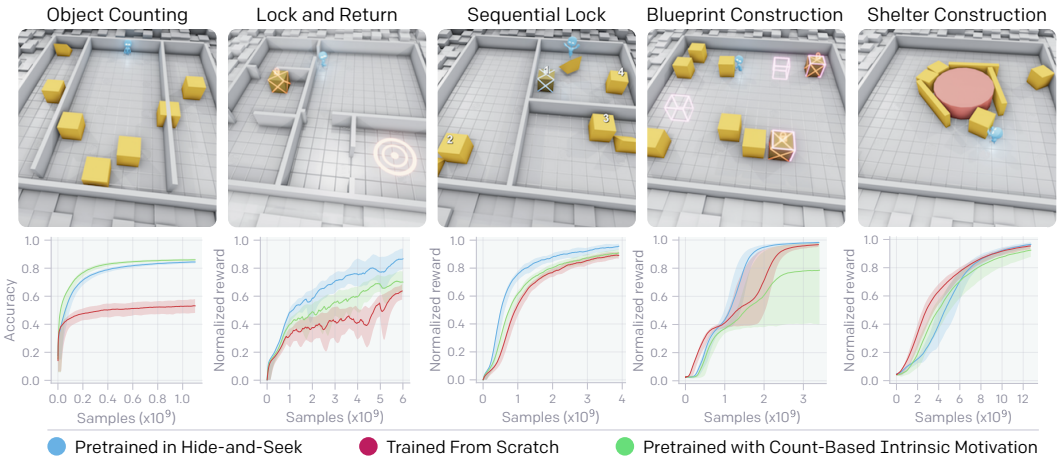

Figura 4: Risultati della valutazione

suali senza porte ma con una rampa in ogni stanza. L'agente deve bloccare tutte le scatole in un ordine particolare (una scatola è bloccabile solo quando è chiusa nell'ordine corretto) che non è osservato dall'agente. L'agente deve scoprire l'ordine, ricordare la posizione e lo stato delle scatole visitate e utilizzare le rampe per spostarsi tra le stanze per portare a termine il compito in modo efficiente.

Task di manipolazione

Nell'attività *costruzione con progetto*, ci sono 8 scatole in una stanza aperta e da 1 a 4 destinazioni obiettivo. L'agente ha il compito di posizionare una scatola in ogni destinazione obiettivo.

Nell'attività *costruzione del riparo* ci sono 3 scatole allungate, 5 scatole cubiche e un cilindro statico. L'agente ha il compito di costruire un riparo intorno al cilindro.

Risultati dei task

I risultati ottenuti sono visibili in figura 4, attraverso un confronto con agenti non allenati e agenti allenati con diverse tecniche. Si osserva che la policy preaddestrata dell'esperimento ha prestazioni leggermente migliori rispetto alle linee di base basate sul conteggio e a quelle inizializzate casualmente in *blocca e ritorna*, *blocco sequenziale* e *costruzione con progetto*; tuttavia, ha prestazioni leggermente inferiori rispetto alla media in *conteggio degli oggetti*, e raggiunge la stessa ricompensa finale ma apprende in modo leggermente più lento in *costruzione del riparo*.

I risultati ottenuti sono spiegabili dalla natura dei task: essi infatti possono essere più o meno complessi da apprendere a seconda della base di partenza. Infatti è possibile che i task nei quali gli agenti di hide-and-seek permormino meglio sia dovuto ad un riutilizzo di feature già imparate (per esempio, in *blocca e ritorna* si ha che la skill è molto simile alla fase (d) di difesa con rampe), mentre nei rimanenti le feature imparate hanno avuto la necessità di essere reinterpretate, che è un processo più lungo e complesso.

4 Applicazioni di RL rilevanti

In generale, è visibile come gli agenti abbiano imparato in maniera più che efficace il gioco del nasccondino, convergendo fino ad un punto in cui non è possibile trovare ulteriori tecniche. Questo esempio è in maniera evidente un esempio di successo di applicazione di reinforcement learning, che però non porta a riscontri pratici applicabili e visibile anche ad un pubblico più ampio, contrariamente a quanto successo in passato ad esempio con Deep Blue [1].

4.1 OpenAI Five

Un esempio di questo calibro, utilizzando strumenti simili a quelli descritti in questo articolo, è stato fornito dalla stessa OpenAI il 13 aprile 2019: essi hanno sviluppato OpenAI Five [5], un sistema di intelligenza artificiale per il videogioco Dota 2, che è riuscito a battere i campioni mondiali del gioco, sconfiggendo per la prima volta nella storia i vincitori in carica in ambito esports.

In questo esperimento gli sviluppatori hanno usato strumenti di reinforcement learning come quelli già descritti (PPO, GAE), ma trovandosi a fronteggiare una sfida di più ambio respiro, vista la natura del videogame. Esso viene giocato in una mappa aperta e quadrata con due team da cinque gioctaori, il cui scopo è difendere la base contenente una reliquia; la partita termina quando una delle due reliquie viene distrutta.

Le principali sfide che sono state affrontate sono:

- **Lunghezza degli orizzonti temporali:** una partita di Dota 2 viene eseguita a 30 fotogrammi al secondo e dura circa 45 minuti. OpenAI Five seleziona un'azione ogni quattro fotogrammi, per un totale di circa 20.000 timesteps per epoca, ovvero 125 volte in più rispetto a quelli di hide-and-seek.

- **Osservabilità parziale:** ogni squadra in gioco può vedere solo la porzione dello stato di gioco vicino alle proprie unità ed edifici; il resto della mappa è nascosto. Fare ottime giocate richiede di fare inferenze basate su dati incompleti e modellare il comportamento dell'avversario.

- **Alta dimensionalità dello spazio di azione ed osservazione:** il sistema osserva circa 16.000 valori totali (principalmente float e categorici con centinaia di possibilità) ad ogni timestep. Per quanto riguarda lo spazio delle spazio d'azione, in un timestep medio il modello sceglie tra 8.000 e 80.000 azioni circa (in hide-and-seek si avevano solamente le azioni di movimento e poche azioni binarie).

4.1.1 Architetture hardware e software

La policy di questo sistema è una distribuzione di probabilità delle azioni, definita da una rete neurale ricorrente (RNN), più in particolare una LSTM, i cui parametri sono circa 159 milioni. Lo schema generale è rappresentato in figura 4.1.1: come spiegato, ad ogni timestep vengono effettuate delle osservazioni, le quali vengono processate (ad esempio, vi è una barra della vita, che viene processata diventando un numero) e fornite in in-

put alla rete neurale (LSTM); infine essa fornisce in output probabilità per ogni azione, tra cui ne viene poi compiuta una.

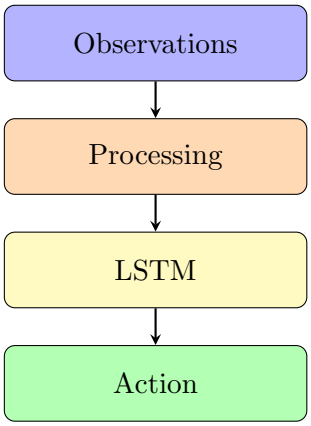

Figura 5: Schema dell'architettura di OpenAI Five

L'obiettivo è quello di trovare una policy che massimizzi la probabilità di vincere contro dei giocatori professionisti. Affinchè ciò avvenga, è stato necessario adattare alcuni strumenti, come la funzione di reward, che invece di concentrarsi esclusivamente sulla vittoria, include input aggiuntivi come la morte dei personaggi, la raccolta di risorse, ecc. Essa è stata progettata per essere a somma zero, il che significa che ogni guadagno per una squadra è una perdita per l'altra.

La policy invece, è allenata con PPO ed ottimizzata con GAE, già descritte. Differentemente però dall'esempio del nascondino, anche queste tecniche sono state adattate: esse hanno necessità di essere scalate su una scala molto più ampia, in particolare la dimensione dei vari batch ed il tempo di training (10 mesi per OpenAI Five), oltre che la significativamente maggiore mole di dati. Questo implica che le suddette tecniche abbiano avuto la necessità di un maggiormente ampio ambiente per essere allenate (figura 6, utilizzata nello stesso articolo).

Un pool centralizzato (*optimizer*) con GPU riceve dati dai giochi che vengono eseguiti e li salva in buffer locali (*expericence buffer*) in maniera asincrona.

L'optimizer, come intuibile dal nome, si occupa della fase di ottimizzazione, cioè cerca valori più adeguati possibile allo scopo nella rete neurale e più in generale in tutte le fasi dell'apprendimento tramite backpropagation, utilizzando l'ottimizzatore Adam [4]. I gradienti derivati dai calcoli delle GPU vengono poi aggregati. Il batch size in questo problema è molto ampio: ogni GPU ha un batch size da 120 sample di 16 timesteps, che per un totale di 1532 GPU a picco, portano a 2949120 timesteps elaborati.

Vi è poi un *controller* centrale `redis` il quale riceve update dei gradienti computati e diversi metadati, così da salvare tutti i parametri e poterli rievocare in caso di necessità. Infine vi sono i *rollout worker* e le *forward pass GPU*. I primi sono le macchine sulle quali viene eseguito il gioco: i videogiochi sono eseguiti a circa metà del tempo reale, in quanto a questa velocità è possibile eseguire in parallelo un numero di giochi leggermente superiore al doppio, aumentando il throughput totale. Esse però eseguono solamente il gioco, non la policy. Essa è eseguita da un pool di GPU (*forward pass GPU*), che comunica con le macchine di rollout ed utilizza batch di dimensione circa 60; la policy viene aggiornata tramite dei pull periodici al controller, che permettono di utilizzare la versione di policy più recente.

Questi componenti, così descritti, portano ad un ciclo di continuo apprendi-

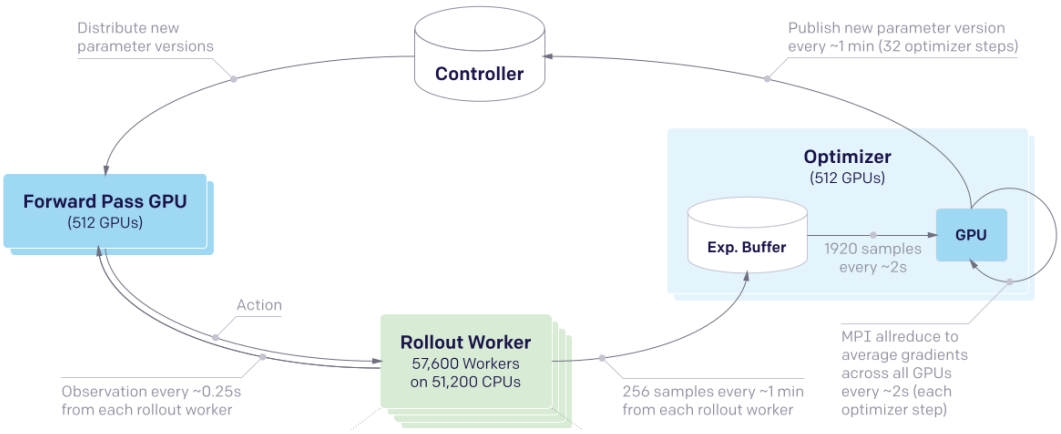

Figura 6: Architettura per l'addestramento di OpenAI Five

mento ed aggiornamento di parametri attraverso nuove partite, come visibile nella figura 6.

4.1.2 Comprensione dell'ambiente da parte degli agenti

Proprio come in hide-and-seek, si vuole provare a capire quali siano le abilità effettivamente apprese dagli agenti nell'addestramento, con particolare riferimento alla comprensione di questi ultime delle dinamiche di gioco, e quindi alla direzione che la partita sta intraprendendo. Non ci si vuole limitare a valutare le prestazioni tramite vittorie e sconfitte, ma si vuole avere, in questo caso, un focus maggiormente dettagliato sulla comprensione del processo decisionale dell'agente, cercando di scoprire come e perché l'agente compie determinate azioni; ad esempio: l'agente ha intenzione di attaccare la torre, o deve opportunisticamente infliggere il maggior numero di danni possibile nei secondi successivi?

Per questo, sono stati introdotti dei valori degli stati futuri da calcolare anticipatamente:

- **Probabilità di vittoria:** la probabilità, compresa tra 0 ed 1, di vittoria durante la partita. Per calcolare questo parametro è stata aggiunta una piccola rete totalmente connessa alla LSTM, avente come output la stessa probabilità. Come visibile in figura 7, con il proseguire delle versioni dell'agente (fino in particolare alla linea rossa, la versione che ha sconfitto i campioni mondiali) vi è una ottima comprensione dell'andamento della partita, e conseguentemente una migliore comprensione di chi vincerà. In figura 7, durante la partita finale, si può notare come dopo circa 5 minuti, la squadra umana ha ucciso diversi eroi di OpenAI Five, mettendone in dubbio il vantaggio. A circa 18 minuti, la squadra OpenAI Five ha ucciso tre eroi umani di fila, ed ha proseguito dichiarando il 95% di probabilità di vittoria.

- **Obiettivi di team / edifici nemici** la previsione probabilistica di un agente nel partecipare o meno al-

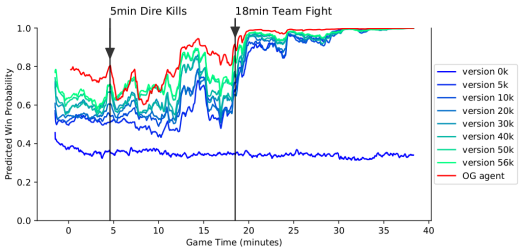

Figura 7: Andamento della probabilità di vittoria durante il match contro il Team OG.

la distruzione di un obiettivo nemico nel futuro prossimo. Nella figura possiamo vedere le previsioni dei diversi eroi per ciascuno degli obiettivi nella partita 1 delle OpenAI Five Finals. In molti casi tutti gli eroi prevedono di partecipare all'attacco, e lo fanno. In pochi casi uno o due eroi sono rimasti fuori:guardando il replay della partita si vede che tali eroi sono impegnati in un'altro punto della mappa. I risultati delle previsioni sono visibili in figura 8.

4.1.3 Obiettivi raggiunti

In generale, le sfide descritte sono state affrontate e vinte grazie ad un sistema che vanta molti punti in comune con alcuni già descritti, ad esempio hide-and-seek, ma che riesce a scalare le caratteristiche su una scala molto più ampia, in particolare la dimensione dei vari batch ed il tempo di training (10 mesi per OpenAI Five).

Nonostante il focus su Dota 2,è ipotizzabile che questi risultati si possano applicare in maniera più generale e che questi metodi possano risolvere svariati ambienti di gioco, in particolar modo se a somma zero, venendo eseguiti in parallelo su centinaia di migliaia di istanze. Inoltre, in futuro, gli ambienti ed i task continueranno a crescere in complessità. A questo proposito, la scalabilità diventerà ancora più importante con l'aumentare della complessità dei compiti. In questo ambito OpenAI Five dimostra come l'utilizzo di alcune tecniche di RL già ampiamente utilizzate, come la PPO, se adeguatamente scalate possono raggiungere "superhuman performances"[2] in ambito esports, tanto da battere i campioni mondiali.

Riferimenti bibliografici

[1] Deep blue (chess computer). https://en.wikipedia.org/wiki/Deep-Blue-(chess-computer), 2024. Accessed: 2024-09-24.

[2] Bowen Baker, Ingmar Kanitscheider, Todor M. Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *CoRR*, abs/1909.07528, 2019.

[3] Sanyam Kapoor. Policy gradients in a nutshell. https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d, 2018. Accessed: 2024-09-18.

[4] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2015.

[5] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin

[2] Citazione dallo stesso articolo

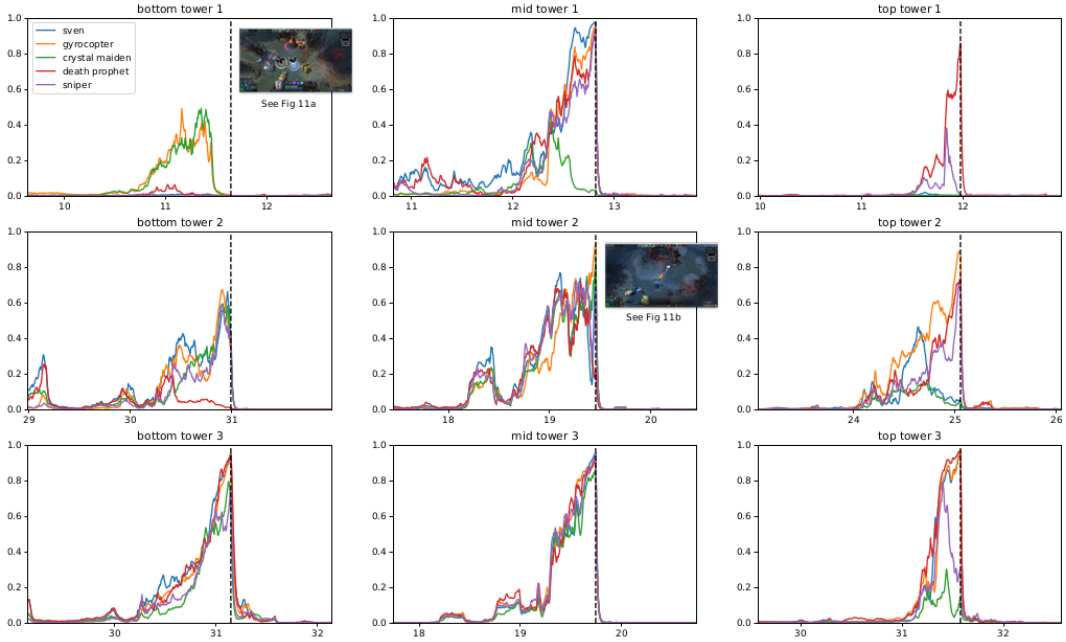

Figura 8: Probabilità di partecipazione per ogni eroe alla distruzione di varie torri.

Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[6] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[7] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[8] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.