Progetto per Machine Learning su VizDoom

Andrea Iskander Belkhir 511089

Url GitHub https://github.com/andreaIskanderBelkhir/VizDoom_ML_Project

Introduzione su VizDoom e sul progetto

Il progetto si è basato sul mettere le mani su un "piattaforma" di Reinforcement Learning partendo dalla creazione di una gym da adattare al modello per poi usare un algoritmo di PPO per dei scenari semplici o un algoritmo di Curriculum Learning per scenari piu complessi. Per fare cio è stato scelto VizDoom, principalmente per usare un mio hobby come soggetto del progetto.

VizDoom è una piattaforma sviluppata dal instituto di computer science dell'univerità Poznan in Polonia per la ricerca di machine visual learning. VizDoom è una piattaforma molto utilizzata nel settore della ricerca in quanto il videogioco su cui è basato "Doom" ha diversi punti di forza che si sposano alla perfezione con cio che molti ricercatori cercano, ovvero: Personalizzazione in quanto creare scenari è molto facile.

Risorse utilizzate infatti Doom è rinomato per poter girare su praticamente qualsiasi device(come un test di gravidanza).

Il progetto si è strutturato(come discusso in chiamata) usando codice trovato online per poi usarlo come base capendo cio che è stato fatto e per quale motivo per poi modificare iperparametri utili all'addestramento per vedere e capire come l'agente compierà azioni diverse in modelli diversi.

Framework e tecnologie usate

Il progetto è stato svolto su macchina Windows 10 avente processore Intel i5 8th Gen senza una GPU, la mancanza di GPU ha influito sul processo di learning dei vari modelli in maniera negativa (training piu lenti e meno frame da analizzare)soprattutto nello scenario piu complesso. Come Framework è stato usato PyTorch, un framework open source basato sulla libreria Torch e usato principalmente per applicazioni di computer vision, ma contiene anche implementazioni di altri algoritmi come nel nostro caso il set denominato Stable-Baselines3 contiene algoritmi di Reinforment Learning.

Proximal Policy Optimization Algorithms

Fra gli algoritmi disponibili è stato usato l'algoritmo Proximal Policy Optimization o PPO, questo algoritmo è un ottimo bilanciamento fra la facilità di implementazione,complessità del campione e un tuning non complesso; Il PPO cerca di elaborare ogni step minimizzando la funzione di costo ma anche tenendo la deviazione dalla scorsa policy bassa.

LCLIP(o)=Et[min(rt(o)At,clip(rt(o),1-Q,1+Q)At)]

* o is the policy parameter
* Et denotes the empirical expectation over timesteps
* rt is the ratio of the probability under the new and old policies, respectively
* At is the estimated advantage at time ttt
* Q is a hyperparameter, usually 0.1 or 0.2

Fonte

Creazione Enviroment

Per addestrare un modello di reinforment learning si fa uso di Gym ovvero librerie python open source che permettono attraverso l'uso di API la comunicazione fra l'algoritmo di learning e l'enviroment, ma nel nostro caso non avevamo un enviroment gia predisposto dalla libreria quindi si è dovuto realizzare, un env per potersi chiamare tale ha la necessita di 5 funzioni basi :

- init : ovvero la funzione che crea l'enviroment dove dobbiamo far partire il gioco ma anche creare lo spazio di azione(le azioni che l'agente puo effetuare) e lo spazio di osservazione (lo spazio dove avviene il gioco),inolte è qui che possiamo effetuare delle modiche ad esempio nel progetto è stata messa l'opportunita di non rendirizzare la finestra di gioco per avere meno peso sulla macchina nel momento di addestramento.
- step : ovvero la funzione che data un'azione la fara effetuare all'agente(da notare che è presente un iperparametro atto al frameskip per non avere ricompense e informazioni prima che l'azione abbia effetivamente effetto) restituendo la reward e l'osservazione dell'enviroment dopo la mossa.
- render : che nel nostro caso possiamo non usare in quanto ci pensera il gioco a specificare come renderizzare
- close e reset : sono funzioni abbastanza autoesplicative, ovvero la prima chiude l'enviroment e la seconda fa partire una nuova partita.

La libreria baseline3 ci fornisce anche uno strumento utile alla creazione di un enviroment ovvero env_checker che controllera il nostro enviroment appena creato e ci dira se è mancante di informazioni o se abbiamo commesso degli errori(quindi è come se effetuasse una spece di debug).

Funzioni di supporto per l'addestramento

Passando alla parte di addestramento le funzioni che hanno supportato "esternamente" sono la funzione di callback e la funzione di evaluate_policy, la funzione di callback ci permette di salvare durante l'addestramento le informazioni e i modelli in questo modo possiamo vedere le informazioni attraverso la tensorboard di TensorFlow con cui possiamo trarre l'andamento dell'addestramento o testare

i modelli e vedere i miglioramenti graficamente, anche perche non è detto che un valore di reward piu alto significhi un agente piu ottiamale e questo è vero soprattutto in scenari piu complessi.

Invece la funzione evaluete_policy è stata usta nella fare di testing per valutare i modelli sviluppati (attraverso la media delle reward di un numero di episodi scelto a mano) insieme a una ciclo for che ci fa vedere graficamente gli episodi su cui testiamo.

Reward Shaping e Curriculm Learning

Nel progetto è stato sviluppato uno scenario piu complesso dove le sole caratteristiche di base non erano abbastanza, ed e per questo che si è usato il reward shaping, ovvero una tecnica spesso usata nel RL basata sull'addestramento degli animali dove le ricompense extra vengono usate per rendere il problema meno complesso.

Oltre alla necessita di reward extra questo scenario è pensato per una difficolta elevata ed è per questo che si è usata la tecnica del curriculum learning, ovvero addestrare il modello per un obbiettivo piu facile per poi usare lo stesso modello come base per un addestramento di un obbiettivo piu complesso. Nel nostro caso è stato necessario in quanto usare un difficolta minore permette al nostro agente di imparare mentre nella difficolta di base i nemici sono molto piu punitivi e questo impedirebbe un buon addestramento.

Scenario 1 Basic

Figure 1: basic.cfg

Il primo scenario ha solo 3 possibili azioni per l'agente ovvero spostarsi a destra

o a sinistra e sparare con reward(+101 per morte mostro,-1 se si è vivi e -5 se episodio finiesce per timeout) che incentivano ad uccidere un mostro che puo "nascere" una volta sola ad episodio in un punto casuale ma sempre alla stessa distanza.

Questo scenario essendo il piu semplice è lo scenario che si usa per prendere confidenza e capire bene come funziona VizDoom, infatti prima ancora di creare l'enviroment si puo provare a far girare il gioco prendendo azioni randomiche. Per il RL invece essendo uno scenario molto semplice un buon modello non richiedera troppo tempo di addestramento, quindi è possibile osservare come cambiano i grafici creati da TensorFlow al cambiare degli iperparametri e attraverso queste prove capire sia i valori dei grafici e gli iperparametri.

Per gli iperparametri abbiamo :

- learning _rate : parametro di tuning che determina la step size di ogni iterazione mentre si avvicina alla funzione minima di loss
- n_steps : il numero di step per ogni enviromentper aggiornamento, il valore deve essere una batch di 64,quindi valori classici da usare sono 2048,4096,8192,etc
- gamma : fattore di sconto
- gae_lamda : fattore del trade-off di bias vs varianza
- clip_range : quanto cambia ad un aggiormento

Mentre nei grafici :

- ep_lenght_mean: quanto dura in media un episodio,nel caso di questo scenario piu è breve piu l'agente arriva prima al goal
- ep_reward_mean: ricompensa media
- experience_variance: un parametro che vogliamo che migliori
- value_loss: parametro che vogliamo che vada verso lo zero
- policy_gradient_lost: se il parametro va troppo verso lo zerosignifica che il modello non sta imparando niente di nuovo
- approx_kl: quanto un agente prende azioni diverse dallo scorso episodio

Un buon modello che è stato addestrato ha un n_step basso di 2048 e un learning_rate di 0.0001 ed è stato addestrato per un totale di 40000 timesteps

Scenario 2 Defend the center

Figure 2: defende the center.cfg

Questo scenario è piu complesso del precedente è composto dal nostro agente,con tre azioni sparare o girarsi a destra o sinistra, situato al centro della mappa dove cinque nemici con la possibilità di rinascere dopo del tempo dalla morte corrono verso il nostro agente.

Questo scenario è stato pensato per far comprendere all'agente che deve uccidere i nemici e sopravvire piu a lungo possibile, per fare cio le ricompense sono +1 ogni volta che uccide un nemico e una penality di 1 quando muore da notare che lo scenario non ha un timeout come lo scenario precedente e questa caratteristica permette durante l'addestramento a far capire che per sopravvivere piu tempo deve sparare per uccidere quindi non deve sprecare colpi . L'addestramento di questo scenario è stato molto semplice in quanto lo scenario è si piu complesso del precendente ma non dobbiamo agire sull'enviroment ma solo sugli iperparametri dell'addestramento;

si è arrivati ad un modello addestrato con n_step pari a 4096 e un learning rate ancora di 0.0001 per un totale di 100000 time_steps.

Come possiamo vedere dalla gif l'agente effetua ancora spari casuali ma questo per ora va bene e nel caso si volesse migliorare basterebbe continuare con lo stesso modello per altri 100000 time_steps

Scenario 3 Deadly Corridor

Figure 3: deadly Corridor.cfg

Questo scenario è il piu complesso dei 3 ed è composto da piu azioni per l'agente infatti avra per la prima volta una vera possibilita di muoversi in un mondo tredimensionale (ovvero potra sia girarsi che muoversi a destra e sinistra)oltre che a sparare, l'obbiettivo di base dello scenario è di insegnare l'agente ad arrivare alla fine del corridoio per prendere un oggetto, per fare cio lo scenario ci dara solo le reward di movimento ovvero un delta positivo se ci si avvicina all'oggetto da raccogliero o negativo se ci si allontana e la penalita di morte, inoltre lo scenario è impostato a difficolta elevata doom_skill=5 ovvero il massimo.

E' in questo scenario che entrano in gioco le tecniche di reward shaping e curriculum learning.

Come gia accennato il curriculum learning ci serve per far addestrare l'agente a difficolta crescente e per fare cio basta creare delle copie dello scenario e modificare la difficolta quindi addestarre normalmente a difficolta 1 fino a trovare un buon modello e poi usare quel modello in un addestramento con la difficolta aumentata e ripetere fino al livello di difficolta iniziale data dallo scenario.

Invece il reward shaping è servito in quanto le semplici ricompense non permetto all'agente di raggiungere l'obbiettivo,questo perche le ricompense sono solo incentrare sul muoversi ignorando i nemici o la propria vita, quindi per implementare queste caratterische è stato necessario modificare il file dello scenario aggiungendo le segueti variabili (da notare queste variabili si trovano nei file Types di VizDoom) : *DAMAGE_TAKEN,HITCOUNT,SELECTED_WEAPON_AMMO* .

Dopo aver inserito queste variabili nello scenario dobbiamo anche inizializzarle nel nostro enviroment semplicemente settando le variabili nella funzione di

init(). A questo punto possiamo usare queste nuove variabili per creare la nuova ricompensa usando i delta di cio che è successo dopo lo step effetuato avendo una formula dove dobbiamo trovare i valori ottimi per le ricompense create.

Reward = Movement_rew + damage_taken_delta * a + hitcount_delta * b + ammo_delta * c . Dove movement reward è la reward gia presente nello scenario, il damage taken è un valore negati che punisce quando veniamo colpiti, hitcount invece ci premia se colpiamo un nemico e ammo in fine serve per non specare troppi proiettili sparando casualmente.

In questo scenario si è notata una neccessità di aumentare il n_steps, learing rate e il numero totale di time_steps per avere risultati migliori, questo ha portato a molti addestramenti di svariate ore(per arrivare sui 500k time_steps sono necesarie piu di 9 ore) per provare a trovare iperparametri migliori e trovare fattori di sconto migliori per le ricompense create. Infatti è da notare che i parametri trovati online non sono ottimale per la macchina su cui è stato realizzato il progetto(per l'addestarmento viene usata la cpu in mancanza della gpu), questo è dovuto molto probabilmente al numero di frame analizzati durante l'addestramento (60 contro i 15 della mia macchina).

Nel corso di giorni sono state provate diversi settaggi sia di iperparametri che di reward(documentati nel notebook) ma per tutti ad un certo punto del training (fra i 100k e i 200k time_steps) il tempo medio del modello cala sempre sotto i 100 che, se questo andava bene per lo scenario 1, per questo scenario è sinonimo di errore in quanto significa che il modello si sta overfittando ovvero in questo caso ignora i nemici e cerca di andare dritto risultando quindi magari ad una reward piu alta ma un comportamento sbagliato.

Dopo circa una settimana di tentativi si è trovata una combinazione di valori per i reward tali che dopo circa 700000 times_steps abbiamo un risultato non ottimale ma comunque otteniamo un agente in grado di finire lo scenario, anche se solo poche volte(circa una volta su cinque), visionando il modello possiamo anche vedere come capita che l'agente prenda strane decisioni quando uccidendo i nemici ne raccoglie l'arma come tornare indietro e stare fermo davanti un muro.

Figure 4: defend_the_cented.gif