Elaborato valido per il Corso di Fondamenti di Intelligenza Artificiale

A.A. 2021/2022

Benchmark di algoritmi di Reinforcement Learning utilizzando il phisics-engine MuJoCo

Università Degli Studi Di Verona

Studenti:

Elisa Acciari VR478828

Giulio Cappelletti VR478827

Professore:

Alessandro Farinelli

Background

Reinforcement Learning

Il **Reinforcement Learning(RL)** può esser definito come lo studio degli agenti e di come essi imparano per tentativi ed errori.

Alcune notazioni largamente usate in problemi di RL:

- **Policy π:** Mapping stato-azione, in cui per ogni stato viene descritta l'azione da effettuare
- **Reward Atteso J(π):** Ricompense attese nel seguire π

Il **problema centrale del RL**, selezionare **π** che massimizza **J(π)** quando l'agente segue π, è dato da

$$\pi^* = \arg\max_{(\pi)} J(\pi)$$

Nel **Deep RL** trattiamo **policy parametriche**, i cui output sono funzioni che dipendono da un insieme di **parametri.**

Tassonomia

Una prima distinzione tra gli approcci al Reinforcement Learning è tra problemi:

- **Model-Based:** Si ha un modello proprio come nei Markov Decision process
- **Model-Free:** I reward ed il modello di transizione sono sconosciuti, vengono quindi imparate delle funzioni:
  - **Value-Function *V(s)***: Reward atteso di partire da ***s*** e seguire una policy da lì in poi.
  - **Q-Function *Q(s,a):*** Reward atteso di effettuare un'azione **a** in un dato stato **s** e seguire una policy da lì in poi.
  - **Policy stocastiche o deterministiche**

Di solito si preferisce l'approccio **Model-Free.** Vediamo quindi come addestrare un'agente:

- **Policy Optimization**: I parametri della policy sono stimati in base alla versione della policy più recente.

$$\theta_{k+1} = \theta_k + \alpha \, \nabla_\theta J(\pi_\theta)|_{\theta_k} .$$

  Definiamo il **Policy Gradient**, in cui le azioni vengono aggiornate solamente in base ai reward ottenuti dopo l'esecuzione dell'azione

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) \right].$$

  Un importante algoritmo che tratteremo di seguito che fa parte di questa classe è **PPO**

- **Q-Learning**: Si basa sull'approssimare la Q-Function ottima tramite un approssimatore. Viene spesso usata la Bellman Equation per effettuare l'update della Q-function:

$$Q^*(s,a) = \underset{s' \sim P}{\mathrm{E}} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$

  Nel **Deep RL** uno degli approcci per andare a stimare la Q-Function è quello di utilizzare una rete neurale Q_θ (Dove con θ indichiamo i parametri). In tal caso possiamo definire:
  - Una **loss function,** detta **funzione di errore di Bellman al quadrato medio (MSBE),** che ci dice quanto Q_θ si avvicina a soddisfare la Bellman Equation.
  - Delle **reti target, in quanto nella riduzione del MSBE vogliamo** che la Q-Function che stiamo stimando sia il più possibile simile al target.

- **Compromesso tra Policy Optimization e Q-Learning:** Vi sono algoritmi che implementano entrambi gli approcci in modo da combinare l'**efficienza** degli algoritmi basati su Q-Learning, in quanto ad ogni aggiornamento si tiene conto dei dati raccolti dall'inizio del training e la **stabilità** degli algoritmi basati

sulla Policy Optimization, in quanto sono ottimizzati direttamente sulla policy. In questo elaborato tratteremo più nel dettaglio i seguenti algoritmi che fanno parte di questa classe: **TD3** & **SAC**

Algoritmi trattati

Di seguito esponiamo brevemente gli algoritmi che siamo andati ad approfondire

Proximal Policy Optimization (PPO)

Caratteristiche

- È **on policy**, in quanto nell'update dei parametri/funzioni si tiene conto dell'azione consigliata dalla policy più recente.
- Si basa sulla policy optimization
- Può essere usato per environment con spazio delle azioni continuo o discreto

Idea

Si basa sull'idea di effettuare il miglior aggiornamento dei parametri della policy possibile in modo che la nuova policy non differisca troppo dalla policy precedente.

Approfondimento

Vi sono due versioni di PPO, **PPO-Penalty** & **PPO-Clip**, in questo elaborato trattiamo solamente PPO-Clip, in quanto è anche la variante usata in OpenAI Gym.
PPO-Clip si basa sull'utilizzo di una ***clip*** per far sì che non vi sia vantaggio per la nuova policy di discostarsi troppo da quella vecchia.

L'update dei parametri in PPO-Clip è il seguente:

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s, a, \theta_k, \theta) \right],$$

In cui ***L*** rappresenta un sample, e può essere così calcolato

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), \quad g(\epsilon, A^{\pi_{\theta_k}}(s, a)) \right),$$

- ***A*** è la funzione di vantaggio, ovvero è il rendimento atteso di effettuare una data azione in un dato stato sottratto al rendimento atteso di uno stato
- Con il termine **clip** si intende che (1-∈) ed (1+∈) fungono, rispettivamente, da limite inferiore e superiore per evitare che la nuova policy si discosti troppo dalla precedente. Vediamo il perché:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

- Se $A \geq 0$ allora g(∈, A) = (1+∈) A

Se A è positivo implica che l'azione consigliata dalla nuova policy diventa più probabile rispetto alla policy precedente, ma il tetto massimo è:

$$(1+\epsilon)A^{\pi_{\theta_k}}(s, a)$$

Quindi la nuova policy non trae vantaggio dal distaccarsi troppo dalla policy precedente.

- Se $A \leq 0$ allora g(∈, A) = (1-∈) A

Se A è positivo implica che l'azione consigliata dalla nuova policy diventa meno probabile rispetto alla policy precedente, ma il tetto massimo è:

$$(1-\epsilon)A^{\pi_{\theta_k}}(s, a)$$

Quindi la nuova policy non trae vantaggio dal distaccarsi troppo dalla policy precedente.

Twin Delayed DDPG (TD3)

Background: DDPG

TD3 si basa sull'algoritmo Deep Deterministic Policy Gradient (**DDPG**).

DDPG utilizza un approccio combinato tra Policy optimization e Q-Learning, in quanto apprende contemporaneamente una Q-function ed una Policy. La Q-Function viene stimata tramite un approccio off-policy, e ne viene fatto l'update tramite la Bellman Equation

La policy viene appresa proprio in base alla Q-Function, infatti la Policy e la Q-Function sono legate, infatti l'azione presa dal Q-Learning è:

$$a^*(s) = \arg\max_a Q^*(s,a).$$

DDPG, come riportato in OpenAI Spinning Guide, presenta un errore, secondo cui la funzione Q appresa inizia a sovrastimare notevolmente i valori Q, il che porta ad errori nelle policy stimate, in quanto vengono usati gli errori nella funzione Q.

Caratteristiche

- È **off policy**, in quanto nell'update dei parametri/funzioni non si tiene sempre conto dell'azione consigliata dalla policy, ma a volte si fa **esplorazione**, così da avere maggiori probabilità di non cadere in ottimi locali.
- È definito per environment con spazio delle azioni continuo

Idea

TD3 è il successore di DDPG, va a risolvere il problema spiegato precedentemente di DDPG, andando ad utilizzare una **clipped double Q-Learning,** imparando quindi due Q-Function, utilizzando la più piccola per aggiornare il target, a loro volta i sample delle Q-Function vengono aggiornate in base al target.

Approfondimento

TD3 nel calcolo dell'azione target aggiunge del **rumore** in modo che essa si trovi nell'intervallo tra tutte le azioni valide $a_{Low} \leq a \leq a_{High}$ evitando che l'approssimatore della Q-function sviluppi un picco errato, poiché comporterebbe errori nella Policy.

In TD3 le azioni che formano il target sono basate sulla policy deterministica $\mu_{\theta_{\text{targ}}}$, con l'aggiunta di rumore ritagliato su ogni dimensione dell'azione. Viene poi ritagliato tutto una seconda volta per avere un intervallo di azione valido.

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

Vengono apprese **due Q-Function**. entrambe utilizzano **un solo target**, calcolato in base alla Q-function più piccola, questo meccanismo è detto **clipped double Q-Learning**.

$$y(r, s', d) = r + \gamma(1-d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')),$$

Entrambe le Q-Function sono aggiornate in base al target.

$$L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\text{E}}\left[\left(Q_{\phi_2}(s,a) - y(r,s',d)\right)^2\right]. \qquad L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\text{E}}\left[\left(Q_{\phi_1}(s,a) - y(r,s',d)\right)^2\right],$$

Infine, rispetto a DDPG l'update della policy viene posticipato; è consigliato effettuare l'update della policy ogni due update della q-function, quindi meno frequentemente, e viene calcolata in base alla seguente equazione

$$\max_\theta \underset{s\sim\mathcal{D}}{\text{E}}\left[Q_{\phi_1}(s, \mu_\theta(s))\right],$$

Soft Actor Critic (SAC)

Background: RL basato sull'Entropia

SAC si basa sul **Reinforcement Learning basato sull'Entropia**. Possiamo definire **l'entropia** di una variabile casuale x in base alla seguente formula, calcolata secondo la funzione di densità di probabilità P di x.

$$H(P) = \mathop{\mathrm{E}}_{x\sim P}\left[-\log P(x)\right].$$

Questo approccio al RL differisce da quello classico in quanto la policy ottima non è più la policy che presenta il massimo reward atteso, ma diventa la policy che per ogni passo temporale ***t*** massimizza il reward atteso sommato all'entropia della policy in quel tempo t.

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\right],$$

Di conseguenza aggiorniamo anche la **Value-Function V(s)** e la **Q-Function Q(s, a)**

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(R(s_t,a_t,s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right)\Big)\,\middle|\, s_0 = s\right] \qquad Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1}) + \alpha\sum_{t=1}^{\infty}\gamma^t H\left(\pi(\cdot|s_t)\right)\,\middle|\, s_0 = s, a_0 = a\right]$$

Caratteristiche

- È **off policy**
- È definito per environment con spazio delle azioni continuo

Idea

L'idea dietro SAC è che non cerca solo di massimizzare i reward, ma anche l'entropia della policy.

Approfondimento

SAC presenta due varianti: con coefficiente di regolarizzazione dell'entropia e con vincolo di entropia e variazione di $\propto$, durante il training. Tratteremo la prima variante, in quanto è anche la versione utilizzata in OpenAI GYM.

SAC come TD3 apprende due Q-Function, che vengono aggiornate in base al MSBE, e convergono ad un unico target condiviso, che viene calcolato con il meccanismo del **clipped double Q-Learning.**
Tuttavia, vi sono anche delle differenze:

- Nel target viene considerato un **termine per la regolarizzazione dell'entropia.**
- Nel target le azioni dello stato successivo sono della policy corrente.

Inoltre, la presenza del **coefficiente di regolarizzazione dell'entropia** può essere vista come un'aspettativa sugli stati successivi e le azioni successive, poiché è un valore atteso può essere approssimata tramite i samples. In SAC possiamo ridefinire quindi:

- il calcolo del **target**
  $y(r,s',d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',\tilde{a}') - \alpha\log\pi_\theta(\tilde{a}'|s')\right), \qquad \tilde{a}'\sim\pi_\theta(\cdot|s').$
- il calcolo della **MSBE**
  $L(\phi_i,\mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\left(Q_{\phi_i}(s,a) - y(r,s',d)\right)^2\right],$

Di conseguenza varia l'update della policy, data dal reward atteso futuro sommato alla futura entropia prevista:

$$\begin{aligned} V^{\pi}(s) &= \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^{\pi}(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^{\pi}(s,a) - \alpha\log\pi(a|s)\right]. \end{aligned}$$

Tool, librerie ed ambienti

Per questo progetto abbiamo utilizzato ed approfondito lo studio dei seguenti:

- phisics-engine ***MuJoCo***, in particolare abbiamo testato e confrontato le prestazioni degli algoritmi scelti sull'environment ***Ant_v2***
- Librerie ***OpenAI Gym*** e ***Stable-Baselines3 (SB3).***

Stable-Baselines3 (SB3)

Caratteristiche

- È una libreria Python
- Fornisce implementazioni testate ed affidabili di algoritmi di RL in PyTorch
- Presenta un'interfaccia molto semplice, è completamente documentata, e permette di addestrare un'agente in poche righe di codice

SB3 implementa i principali algoritmi utilizzati nel RL, tra questi noi abbiamo approfondito ed utilizzato PPO, SAC e TD3.

OpenAI Gym

Caratteristiche

- È una libreria Python, Open-source
- È di supporto per semplificare lo sviluppo ed il benchmark di algoritmi, per chi ha vuole entrare nel mondo del RL
- Fornisce un'API standard per comunicare tra algoritmi ed environment
- Mantiene un sistema di version control, tutti gli environment terminano con "_vX" dove X sta per il numero di versione: es "***ant_v2***"

Nel corso di questo progetto abbiamo approfondito l'utilizzo di OpenAI Gym, già utilizzato nel corso delle lezioni di laboratorio per il corso di Fondamenti di Intelligenza Artificiale.

Al seguente link sono riportati i principali metodi da conoscere:

https://www.gymlibrary.ml/content/api/

La principale caratteristica di Gym è la semplicità con cui permette di interfacciarsi, ed interagire con gli environment, nella sezione di questo elaborato dedicata al codice andremo ad esaminare come inizializzare, renderizzare ed eseguire azioni in un environment, utilizzando **Ant-v2** come esempio.

MuJoCo

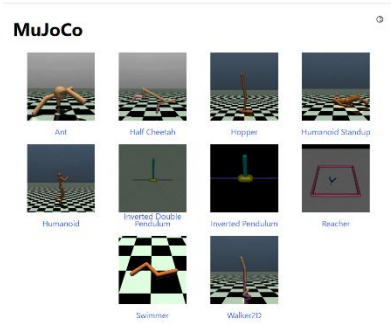
Caratteristiche

- E' un physics-engine, Open-source
- Fornisce environment per effettuare simulazioni accurate, quindi molto utilizzato in tutte le aree che richiedono una simulazione accurata e veloce

Si rimanda a questo link per chi volesse approfondire le funzionalità chiave di MuJoCo:

https://mujoco.readthedocs.io/en/latest/overview.html#key-features

MuJoCo - OpenAI Gym



Il motore MuJoCo è supportato da Gym, MuJoCo, e fornisce i seguenti environment

Ant

Ant è l'environment che abbiamo utilizzato nel nostro progetto.

Fa parte degli environment di MuJoCo disponibili in Gym. È un robot 3D costituito da un busto a rotazione libera con quattro gambe attaccate, ogni gamba presenta due collegamenti.

Goal

L'**obiettivo** è coordinare le gambe per muoversi in avanti (a destra).

Versioni

Vi sono differenti versioni di Ant, dalla v0 alla v4, la versione utilizzata da noi in questo progetto è **Ant-v2**, che supporta mujoco >= 1.50.

Una spiegazione più esaustiva di Ant, con un elenco completo dello spazio delle azioni ed osservazioni:

https://www.gymlibrary.ml/environments/mujoco/ant/

Stato iniziale

Come riportato in OpenAI Gym, le osservazioni iniziano nello stato (0,0, 0,0, 0,75, 1,0, 0,0 … 0,0)
Inizialmente la coordinata z è settata volontariamente con valori leggermente più alti in modo che Ant sia in piedi, ed anche l'orientamento iniziale è settato per far sì che Ant sia posizionato in avanti.

Fine episodio

Un episodio termina quando si verifica uno dei seguenti eventi:

1. La durata dell'episodio raggiunge i timestep
2. Nessuno dei valori dello spazio degli stati non è più finito
3. L'orientamento y nello stato non è nell'intervallo [0.2, 1.0]

Spazio delle azioni

Ant presenta un vettore di 8 elementi per le azioni, dove il dominio di valori è [-1, 1], ogni azione rappresenta un movimento applicato ai collegamenti tra il busto e la prima parte di gamba, o ai collegamenti tra la prima e seconda parte di gamba.

Spazio delle osservazioni

Un'osservazione è un array di forma (111,) in cui i valori che indicano la posizione delle diverse parti del corpo dell'agente.

Reward

Nell'environment Ant vi sono differenti tipologie di reward:

- **survive_reward**: Il Reward è di 1 ogni volta che Ant è ancora in vita
- **forward_reward**: il Reward è pari a (x (dopo l'azione) – x(prima dell'azione)) / dt
  Dove con **dt** indichiamo il tempo tra le azioni, e con **x** la coordinata x.
  Il Reward è quindi positivo se Ant va in avanti (a destra)
- **ctrl_cost:** Il Reward è negativo se Ant compie azioni troppo grandi
- ***contact_cost***: Il Reward è negativo se la forza di contatto esterna è troppo grande, con forza di contatto esterna si intendono delle forze esterne applicate al centro di massa di ognuno dei collegamenti

Installazioni necessarie

Prerequisiti

1. Sistema Linux, Ubuntu 20.04 LTS
2. Pip, per installare i pacchetti.
3. Python = 3.7+
   Nota:
   Stable-Baselines3 richiede tensorflow da 1.8.0 a 1.15.0, e tensorflow 1.15.0 è supportato da Python 3.7, in questa guida abbiamo utilizzato Python 3.7
4. Anaconda, si consiglia infatti di utilizzare un'ambiente virtuale Anaconda/Miniconda o un ambiente virtuale Python.
   Controllare se conda è installato

   ```
   >> conda --version
   ```

   Se conda non è installato, Installare seguendo le istruzioni riportate nel link di seguito:
   https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html
   Creare un ambiente virtuale con conda è Python = 3.7

   ```
   >> conda create -n env python=3.7
   ```

5. PyTorch >= 1.11

OpenAI GYM

https://github.com/openai/gym

OpenAI Gym supporta da Python 3.7 a Python 3.10 su sistemi Linux e MacOS, ed anche se non supportato ufficialmente Gym è disponibile anche su Windows

Per installare la libreria Gym, senza includere le dipendenze per tutti gli environment (in quanto alcune potrebbero riportare degli errori in alcuni sistemi operativi), utilizzare:

```
>> pip install gym
```

Altrimenti

```
>> pip install gym[all]
```

Come già accennato, Gym supporta alcuni environment MuJoCo, dopo aver installato il motore MuJoCo, vanno installate le dipendenze in Gym per questo tipo di environment

```
>> pip install gym[mujoco]
```

Installazione Stable-Baselines3 & Tensorboard

Per installare il pacchetto base di Stable-Baselines3 digitare

```
>> pip install stable-baselines3
```

Se invece si vogliono includere le dipendenze: Tensorboard, OpenCV o atary-py, digitare

```
>> pip install stable-baselines3[extra]
```

Per visualizzare i grafici ci siamo serviti del tool Tensorboard, che va installato con il seguente codice:

```
>> pip install tensorboard
```

Implementazione

In questa sezione tratteremo il codice sviluppato, approfondendo l'utilizzo di gym, mujoco e stable_baselines3.

Per spiegare step by step cosa andiamo a fare utilizziamo il codice che implementa TD3.

Nelle prossime sezioni riporteremo anche il codice per l'implementazione di SAC e PPO, si ricorda infatti che lo scopo di questo elaborato è di approfondire questi 3 algoritmi valutandone le prestazioni tramite l'environment Ant-v2.

Esempio: step by step

Importare librerie necessarie

Andiamo ad importare la libreria **gym**, che ci permetterà di interagire con l'environment, e dalla libreria **stable-baselines3** andiamo ad importare l'implementazione dell'algoritmo **SAC**

```
import gym
from stable_baselines3 import TD3
```

Inizializzare environment con Gym

La caratteristica principale di OpenAI Gym è la semplicità con cui si possono gestire gli environment.

Vediamo come inizializzare un l'environment "Ant-v2" in Gym:

```
env = gym.make("Ant-v2")
```

Addestrare ed eseguire un modello con SB3

Con la libreria Stable-baselines3 è possibile creare ed addestrare un modello in poche righe di codice.

Con "`TD3("MlpPolicy", env)`" andiamo a definire il modello.

Con ". learn(n)" andiamo ad addestrare il modello per n timestep

```
model = TD3(policy, env).learn(n)
```

Per migliorare il training del modello siamo andati a manipolare i parametri di TD3.

I valori dei parametri saranno discussi in seguito, per ora ne riportiamo solo alcuni che ci serviranno per stampare gli output del modello e visualizzare dei grafici che ci danno indicazioni sulle prestazioni.

Output & grafici con Tensorboard

Per visualizzare l'output di ogni episodio andiamo a ad utilizzare un parametro messo a disposizione dalle implementazioni in SB3, ovvero, **verbose.** Verbose può assumere valore 0 o 1, se 0 indica che non si visualizzano gli output, se invece viene settato a 1 gli output vengono visualizzati.

In questo elaborato al fine di stampare dei grafici che permettessero di valutare le prestazioni dei vari algoritmi ci siamo serviti del tool **Tensorboard**. Infatti, un altro parametro che è possibile specificare è proprio **tensorboard_log.**

tensorboard_log = "path/folder_name/{}_tensorboard/"

Tramite questo i grafici saranno salvati nel percorso specificato. Per attivare il tool è necessario:

- digitare:
  tensorboard dev upload --logdir \ > 'path/folder_name/{}_tensorboard/'
- per visualizzare i grafici, aprire il link stampato come output:
  View your TensorBoard at https://tensorboard.dev/

Se invece il tool per quello specifico file log è stato già attivato con i passi precedenti, bisognerà digitare i seguenti comandi:

- digitare:
  tensorboard --logdir path/folder_name/{}_tensorboard/
- per visualizzare i grafici, aprire il link stampato come output :
  TensorBoard 2.9.0 at http://localhost:6007/

Reset dell 'environment

Con questo comando andiamo semplicemente a resettare l'ambiente al punto iniziale.

```
obs = env.reset()
```

Predict del modello

Andiamo ad iterare per un numero n di episodi

```
for i in range(n):
```

Successivamente andiamo ad utilizzare il metodo ".predict()" per ottenere l'azione e gli stati dati dal modello attuale

```
    action, _states = model.predict(obs)
```

Andiamo poi ad eseguire effettivamente l'azione con il metodo ".step()", questo metodo che prende come parametro l'azione da eseguire ci ritorna:

- **obs** (Object) – questo sarà un elemento dell'ambiente observation space.
- **reward** (float) – rappresenta la ricompensa restituita a seguito dell'esecuzione dell'azione.
- **done** (bool) – indica se il prossimo stato sarà uno stato terminale.
- **info** (*dizionario*) – contiene informazioni diagnostiche ausiliarie.

```
    obs, reward, done, info = env.step(action)
```

In modo opzionale possiamo aggiungere il comando ".render" per andare a renderizzare, quindi visualizzare l'environment

```
    env.render()
```

Effettuiamo nuovamente il reset dell'environment quando il prossimo stato è terminale

```
if done:
      obs = env.reset()
```

Infine con ".close()" terminiamo l'esecuzione del programma e dell'environment

```
env.close()
```

Qui di seguito riportiamo il codice completo:

```
import gym
from stable_baselines3 import TD3

env = gym.make("Ant-v2")

model = TD3("MlpPolicy", env, tensorboard_log="./antTD3/{}_tensorboard/",
verbose=1).learn(35000)

obs = env.reset()

for i in range(5000):
```

```
    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    env.render()

    if done:
      obs = env.reset()

env.close()
```

Codice sviluppato

L'utilizzo di Gym e Stable-baselines3 permettono, come già visto di semplificare di molto il codice, infatti il codice necessario per andare a inizializzare e ad allenare con il modello è composto da poche righe.

Di seguito riportiamo solamente i codici con gli iperparametri migliori che abbiamo trovato per Ant.

PPO

```
import gym
from stable_baselines3 import PPO

env = gym.make("Ant-v2")

model = PPO("MlpPolicy", env, learning_rate = 0.0003, n_steps=200, batch_size = 128, gamma=0.99,
        gae_lambda = 0.95, verbose=1, clip_range = 0.2,  tensorboard_log="./ppo/{}_tensorboard/").learn(35000)

obs = env.reset()

for i in range(10000):

    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    env.render()

    if done:
      obs = env.reset()

env.close()
```

SAC

```
import gym
from stable_baselines3 import SAC

env = gym.make("Ant-v2")

model = SAC("MlpPolicy", env, buffer_size=2000, learning_starts=1000, batch_size=128, tau=0.005,
        ent_coef = 'auto_0.1',  verbose=1, target_update_interval=2, tensorboard_log="./sac/{}_tensorboard/").learn(30000)

obs = env.reset()

for i in range(10000):

    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    env.render()

    if done:
      obs = env.reset()

env.close()
```

TD3

```
import gym
from stable_baselines3 import TD3

env = gym.make("Ant-v2")

model = TD3("MlpPolicy", env, buffer_size=900000, learning_starts=8000, batch_size=128,
        train_freq=1024, tensorboard_log="./td3AntDef/{}_tensorboard/", verbose=1).learn(28000)

obs = env.reset()

for i in range(5000):

    action, _states = model.predict(obs)
    obs, reward, done, info = env.step(action)
    env.render()

    if done:
      obs = env.reset()

env.close()
```

Iperparametri: PPO, SAC & TD3

Nel Corso del Progetto siamo andati ad approfondire alcuni iperparametri, di cui alcuni tipici per più algoritmi di Reinforcement Learning, ed altri specifici per le implementazioni di PPO, SAC e TD3.

Iper parametri generali

- **learning_rate**(int): Rappresenta il tasso di apprendimento
  (Valori consigliati: tra 0 ed 1)
- **batch_size**(`int`): Rappresenta le dimensioni del mini batch
  (Valori consigliati: da 32 a 5000)
- **gamma**(`float`): Rappresenta il discount factor

Iper parametri PPO

https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

- **n_steps**(**int**)*:* Rappresenta Il numero di step da eseguire per ogni ambiente per aggiornamento
  (valori consigliati: da 32 a 5000)
- **n_epochs**(**int**): Rappresenta il numero di iterazioni del training set.
  (valori consigliati: da 3 a 30)

Iper parametri per SAC & TD3

https://stable-baselines3.readthedocs.io/en/master/modules/sac.html

- **buffer_size**(**int**)*:* Rappresenta la dimensione del buffer di riproduzione
- **tau**(**float**)**:** Rappresenta il coefficiente di update
- **learning_starts**(**int**)*:* Rappresenta il numero di passaggi del modello in cui raccogliere le transazioni prima dell'inizio dell'apprendimento.
- **train_freq** (**Union**[**int**, **Tuple**[**int**, **str**]])**:** Aggiorna il modello ad ogni "train_freq" passaggio.
- **gradient_steps** (**int**) – Quanti passaggi del gradiente eseguire dopo ogni rollout

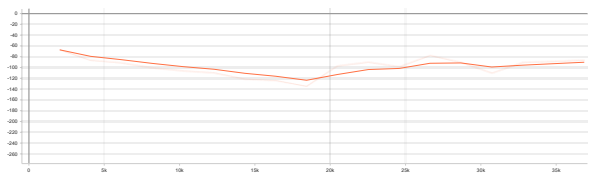
Nelle prossime sezioni tratteremo la configurazione e manipolazione degli iperparametri di PPO, SAC e TD3. Per ogni algoritmo al fine di trovare i valori degli iperparametri più performanti, abbiamo effettuato molte combinazioni, riporteremo solamente alcune combinazioni e gli iperparametri che siamo andati a manipolare.

In ordine avremo avremo:
- Modello SB3, con i parametri base implementati da SB3.
- Modello 2, con i parametri modificati, ma non ottimali.
- Modello 3, con i parametri modificati, con i quali abbiamo le migliori prestazioni.

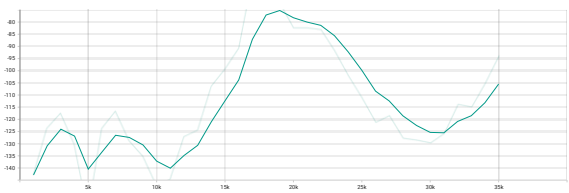
Iperparametri: PPO

Iperparametri	PPO_ SB3	PPO_2	PPO_3
n_steps	2048	1000	200
batch_size	64	128	128
clip_range	0.2	0.1	0.2

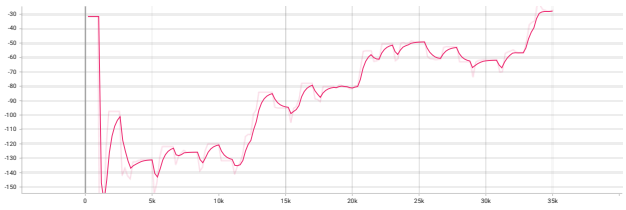
PPO_ SB3



PPO_2



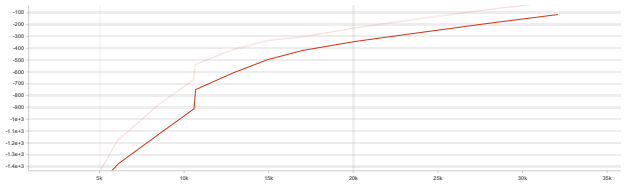
PPO_3



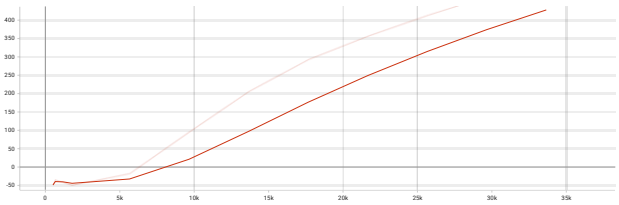
Iperparametri: SAC

Iperparametri	**SAC_ SB3**	**SAC_2**	**SAC_3**
buffer_size	1000000	2000	2000
learning_starts	100	1000	1000
batch_size	256	256	128
tau	0.005	0.001	0.005

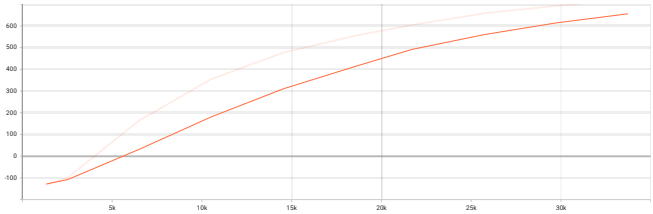
SAC_ SB3



SAC_2



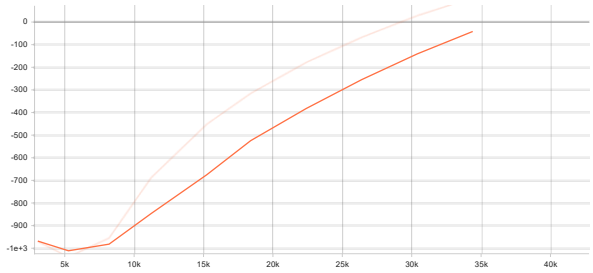
SAC_3



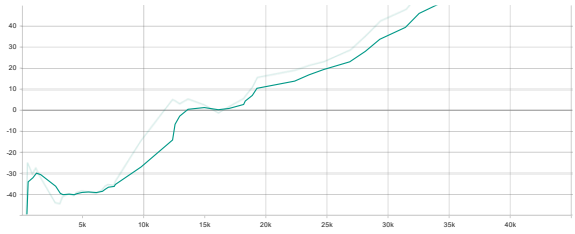
Iperparametri: TD3

Iperparametri	**TD3_ SB3**	**TD3_2**	**TD3_2**
buffer_size	1000000	90000	90000
learning_starts	100		8000
batch_size	100		128
train_freq	(1,'episode')		1024

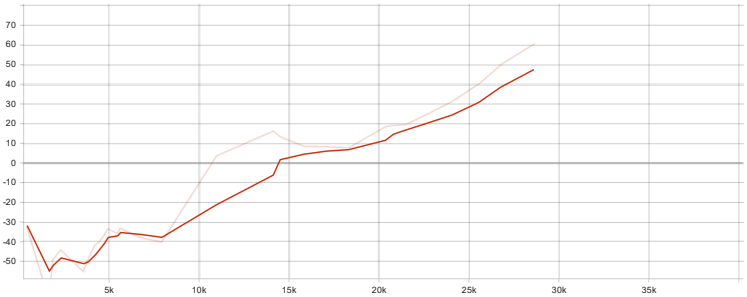
TD3_SB3



TD3_2



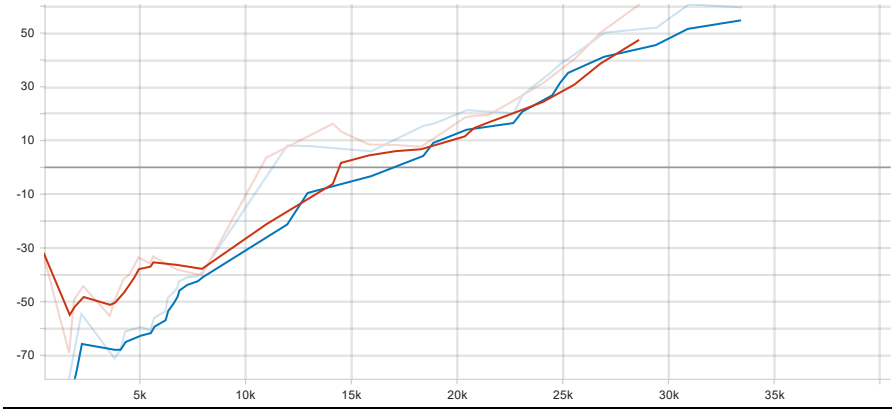
TD3_3



Durante lo svolgimento del progetto abbiamo potuto notare come non solo la manipolazione dei parametri causa cambi di prestazioni, ma anche la variazione del valore dei timesteps (ovvero il parametro n del metodo ".learn(n)")

Di seguito vediamo un esempio di ciò tramite TD3, andando ad eseguire TD3_3 con timesteps = 35000 anziché timesteps = 28000.

- TD3_3 con timesteps = 28000
- TD3_3 con timesteps = 35000



Si può notare come nel 2° caso il reward cresca in modo sempre continuo e maggiormente avendo più timesteps a disposizione, tuttavia si parte da un reward minore (-100) rispetto al 1° caso (-30) .

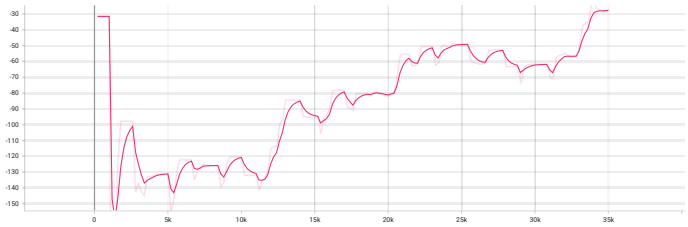
Benchmark: PPO, SAC & TD3

In questa sezione finale andiamo a confrontare le prestazioni dei tre algoritmi trattati, ovvero PPO, SAC e TD3, riportando le configurazioni per i rispettivi iperparametri.

Iperparametri	**PPO**	**SAC**	**TD3**
learning_starts	/	1000	8000
batch_size	128	128	128
n_steps	200	/	/
buffer_size	/	2000	90000
train_freq	/	1	1024
clip_range	0.2	/	/
tau	/	0.005	0.005

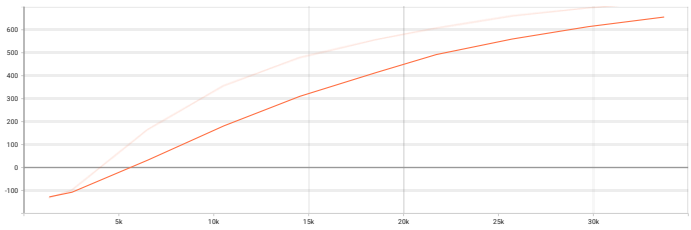
PPO

Eseguendo PPO_3 vediamo che l'agente Ant tende a muoversi nella direzione giusta, ma effettua movimenti scattosi e avviene frequentemente il reset dell'ambiente.



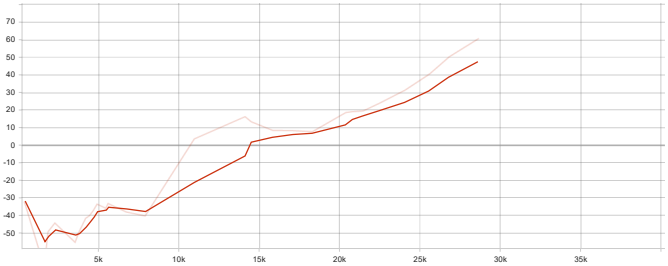
SAC

Eseguendo SAC_3 vediamo che l'agente Ant si muove in direzione corretta verso destra come dovrebbe risultare con una traiettoria rettilinea.



TD3

Eseguendo TD3_3 vediamo che l'agente Ant tende a muoversi nella direzione corretta, tuttavia si blocca frequentemente.



In conclusione possiamo notare che, considerando le implementazioni con i parametri settati in SB3 per tutti e tre gli algoritmi, TD3 risulta più performante sull'environment Ant rispetto a PPO e SAC.

Mentre andando a confrontare la versione con i parametri da noi trovati (PPO_3, SAC_3, TD3_3), SAC cresce in modo più continuo rispetto a PPO e TD3, inoltre comporta reward più alti.

Di seguito riportiamo alcuni frame dell'esecuzione di Ant.