Relazione

Obiettivo

L'obiettivo principale di questo progetto è di apprendere come costruire e strutturare diversi tipi di reti neurali e di testarle su problemi di Reinforcement Learning, al fine di sviluppare una comprensione più approfondita di come funziona questa tecnologia e di come può essere applicata in diversi contesti.

Inoltre, durante il corso del progetto, verranno affrontati una serie di sotto-obiettivi, tra cui la comprensione dei principi fondamentali del Reinforcement Learning, l'analisi di diversi algoritmi di apprendimento automatico utilizzati nelle reti neurali, lo studio delle tecniche di elaborazione dei dati utilizzate per preparare i dati di input per le reti neurali, e l'implementazione di un'architettura di rete neurale completa per risolvere problemi di Reinforcement Learning.

Tools Usati

Gym

Gym è una libreria open-source di OpenAI che permette di creare e testare algoritmi di apprendimento automatico su problemi di Reinforcement Learning. La libreria include una vasta gamma di ambienti, come il classico gioco del CartPole e altri giochi Atari, e offre un'interfaccia standardizzata per l'interazione con questi ambienti di gioco. In questo caso specifico Gym è stato usato sia per essere dato direttamente come input a una rete, che come creatore di un dataset di sequenze.

Stable-Baselines3

Stable-Baselines3 è una libreria open-source di apprendimento automatico che si concentra su algoritmi di reinforcement learning. È costruita sulla libreria di machine learning PyTorch e offre una vasta gamma di algoritmi di reinforcement learning, come DQN, A2C e PPO. È stato progettato per essere facile da usare e configurare soprattutto vista la presenza di modelli già trainati quindi un notevole risparmio di tempo..

Keras-RL

Keras-RL è una libreria open-source di apprendimento automatico che fornisce un set di strumenti per implementare algoritmi di reinforcement learning in Keras. Keras-RL offre una vasta gamma di algoritmi di reinforcement learning, tra cui Deep

Q-Networks (DQN), Actor-Critic, e Proximal Policy Optimization (PPO). Keras-RL è stato usato per testare diversi modelli di reti neurali con diversi pesi presi da modelli pre-addestrati.

Ambiente Virtuale Gym

Ogni ambiente di Gym è costruito con la stessa struttura

- **Action Space**

  L'Action Space in un ambiente di Reinforcement Learning rappresenta l'insieme di tutte le azioni che l'agente può compiere in ogni stato dell'ambiente.

- **Observation Low-High**

  L’observation è una descrizione di tutti i parametri che compongono un ambiente virtuale. In questo caso Low-High perchè ci sono i valori minimi e massimi di ogni variabile

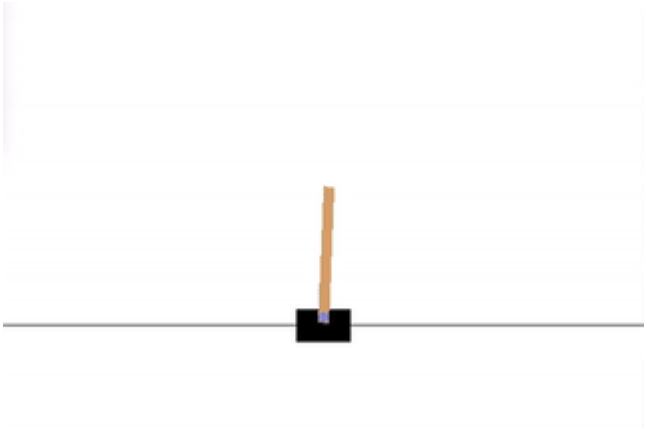
- **Reward**

  Il **reward** in Reinforcement Learning è una misura numerica che rappresenta la qualità dell'azione scelta dall'agente in uno stato specifico dell'ambiente. L'obiettivo dell'agente è massimizzare la somma dei reward accumulati nel tempo, ovvero massimizzare la sua "ricompensa".

Cart Pole

Tipo di problema

Il problema CartPole è un problema con una struttura semplice. Un palo posto su un carrello che può muoversi solo a sinistra e a destra, e l’obiettivo è non farlo cadere ma farlo rimanere in equilibrio



- Action Space

  Nel caso del CartPole l’action space è un ndarray (array multi-dimensionale) di shape (2,) che indica la direzione dove imprimere la forza

0	Spingi il cart a sinistra
1	Spingi il cart a destra

- Observation Low-High

  In questo caso è sempre un ndarray ma con shape (4,) che descrive tutti i valori che può raggiungere l’ambiente. Però non sono per forza sinonimi dei valori accettati da questo, nello specifico

  La posizione può andare da -4.8, 4.8 però l’episodio termina se il cart lascia il range -2.4, 2.4

  E l’angolo che è accettato a ±24° ma il range di accettabilità è solo ±12°

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

- Reward

  Il reward in questo caso va da 0 a 500 e inviene incrementato di 1 ogni volta che fa uno step il cart con il pendolo compreso nel range

Pendulum

Tipo di problema

Il problema Pendulum è un problema più complesso rispetto a CartPole. In questo caso dobbiamo bilanciare un pendolo su un asse orizzontale.

- Action Space

  Nel caso del Pendulum l’action space è un ndarray (array multi-dimensionale) di shape (1,) che indica la forza da applicare all'asse orizzontale del pendolo.

- Observation Low-High

  In questo caso l'observation è un ndarray con shape (3,) che descrive tutti i valori che può raggiungere l’ambiente.

Num	Observation	Min	Max
0	cos(theta)	-1.0	1.0
1	sin(theta)	-1.0	1.0
2	theta dot	-8.0	8.0

- Reward

  Il reward in questo caso va da -16.27 a 0.0 e viene calcolato con una funzione

$$r = -(theta^2 + 0.1 * thetadt^2 + 0.001 * torque^2)$$

DQN Policy

Una **policy** nel Reinforcement Learning è una funzione che mappa uno stato dell'ambiente ad un'azione da eseguire. La **Deep Q-Network** (DQN) è un algoritmo di apprendimento di Reinforcement Learning che utilizza una rete neurale per approssimare la funzione Q, ovvero la funzione che indica il valore di una coppia stato-azione. La policy DQN viene quindi definita come la scelta dell'azione che massimizza il valore di Q per lo stato corrente. La rete neurale viene addestrata utilizzando un algoritmo di ottimizzazione per minimizzare la differenza tra il valore di Q stimato e il valore di Q atteso, calcolato utilizzando la regola di Bellman.

Creazione del dataset

Abbiamo utilizzato le librerie **Keras-rl** e **Stable-Baselines3** per costruire modelli pre-addestrati.

Questi modelli sono stati usati per costruire un dataset di osservazioni e azioni che hanno ottenuto un buon reward.

Per rendere il dataset più variegato abbiamo caricato pesi diversi al modello, scartando comunque le osservazioni con un basso reward.

Il dataset è stato salvato su un file .csv con la seguente struttura:

- Per CartPole:

observation	action
[0.030446894466876984, 0.043070994317531586, 0.03923819586634636, 0.011520277708768845]	[0.0, 1.0]

- Per Pendulum:

Observation	Action
[0.585658073425293, -0.8105581998825073, 0.885906994342804]	[-0.341]

Sul git sono presenti i seguenti file csv :

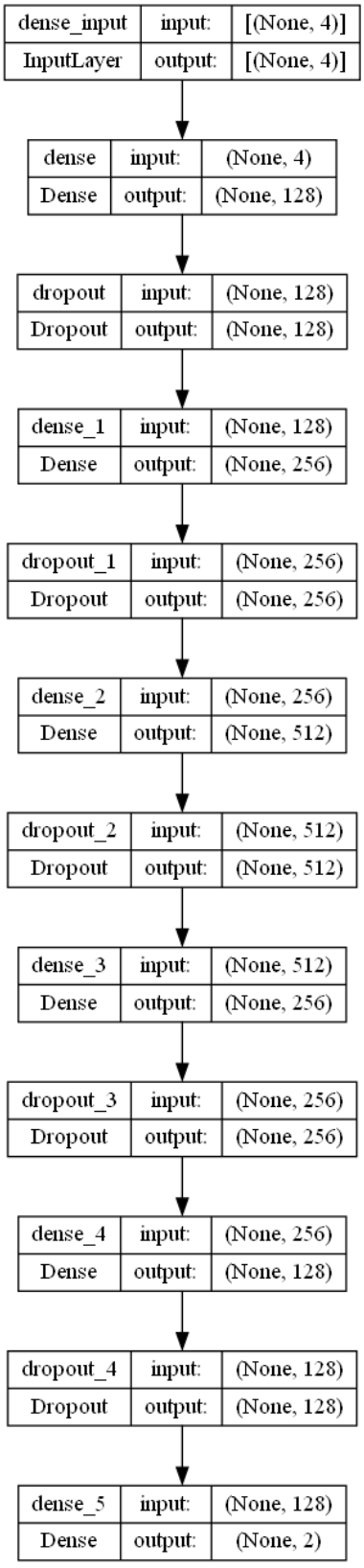
- dataset.csv: costruito utilizzando un modello pre-addestrato di Stable-Baselines3 con una lunghezza di circa 2000 entry
- dataset_keras.cvs: costruito utilizzando un modello pre-addestrato di Keras-rl con una lunghezza di circa 5000 entry
- dataset_random.csv: costruito eseguendo azioni in maniera casuale, salvando azioni e osservazioni con un buon reward con una lunghezza di circa 20000 entry
- …

Test CNN

CartPole

La CNN che abbiamo testato è la seguente:

```
#file git CNN_test
def create_model(states, actions):
    model = Sequential()

    model.add(Dense(128, input_shape=(states,), activation="relu"))
    model.add(Dropout(0.6))

    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.6))

    model.add(Dense(512, activation="relu"))
    model.add(Dropout(0.6))

    model.add(Dense(256, activation="relu"))
    model.add(Dropout(0.6))

    model.add(Dense(128, activation="relu"))
    model.add(Dropout(0.6))
    model.add(Dense(actions, activation="softmax"))

    model.compile(
        loss="categorical_crossentropy",
        optimizer="adam",
        metrics=["accuracy"])
    return model
```



Attraverso il comando `model.fit(obs_data, act_data, epochs=5)` abbiamo addestrato la rete per 5 epoche su i dataset creati.

Con il comando `model.predict(observation.reshape(1, states))` abbiamo utilizzato il modello addestrato come policy per scegliere l'azione da eseguire in base all'osservazione ottenuta.

Test con Keras-Rl

```
#output CNN con input dataset_kesar.csv
Episode: 50/50

Average: 492.9
Median: 500.0
```

Dato un limite massimo di reward di 500 abbiamo visto che il dataset, comunque limitato in lunghezza (con solo 5k valori) ha fornito un ottimo risultato di 492 reward in media

Test senza modello pre-trained

```
#output CNN con input dataset_random.csv
Episode: 50/50

Average: 364.62
Median: 403.0
```

Nonostante la quantità maggiore di osservazioni i risultati sono stati peggiori, questo perché le osservazioni salvate hanno un reward medio inferiore rispetto al dataset_keras.csv

Pendulum

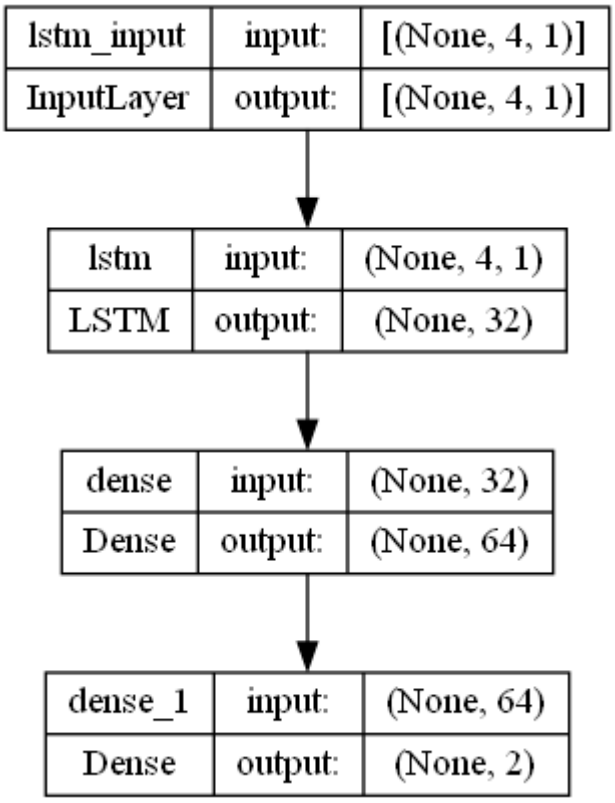
Test LSTM

CartPole

La LSTM che abbiamo testato è la seguente:

```
#file git LSTM_test
def create_model(states, actions):
    model = Sequential()

    model.add(LSTM(32, input_shape=(states, 1)))
    model.add(Dense(64, activation="relu"))
```

```
    model.add(Dense(actions, activation="softmax"))

    model.compile(
        loss="mse",
        optimizer="adam",
        metrics=["accuracy"])
    return model
```



Come per la CNN abbiamo addestrato la rete per 5 epoche usando gli stessi dataset.

Test con Keras-Rl

Dato in input databese_keras.csv alla rete i risultati:

```
#output LSTM con input dataset_keras.csv
Episode: 50/50

Average: 497.34
Median: 500.0
```

Dato in input dataset_random.csv alla rete i risultati:

```
#Run 1
#output LSTM con input dataset_random.csv
Episode: 50/50

Average: 500.0
Median: 500.0

#Run 2
#output LSTM con input dataset_random.csv

Episode: 50/50
Average: 423.64
Median: 440.0
```

In generale la LSTM ha ottenuto dei risultati migliori rispetto alla CNN, ma facendo numerose prove ci siamo accorti che i risultati ottenuti sono molto altalenanti e possono variare di molto ripetendo l’apprendimento della rete. Come si vede dalle 2 run

Pendulum