Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Trabajo Final de Graduación
Proyecto: Método basado en aprendizaje reforzado para el control automático de una planta no lineal.
Estudiante: Oscar Andrés Rojas Fonseca
I Semestre 2024

Firma del asesor

Bitácora de trabajo

Fecha	Actividad	Anotaciones	Horas dedicadas
15/04/2024	**1**. Redefinición de la conversión del código para valores discretos ($CartPole$) a valores continuos ($Pendulum$).	*a*) El error en $select\_action()$ se corrigió pero desconfiguró parte de la función $optimize\_model()$. Corrección del error. *b*) Persisten los problemas de indexado y proceso.	6 horas
15/04/2024	**2**. Pruebas de entrenamiento del modelo ($Pendulum$).	*a*) Se entrenaron cuatro modelos diferentes a 600 episodios para comparar el efecto de cuatro propuestas de redes neuronales artificiales (ANN).	4 horas
16/04/2024	**3**. Búsqueda de la teoría de los métodos PPO y $actor-critic$ dada la necesidad del manejo del *action space* con valores continuos.	*a*) Revisión general de los conceptos para ambos casos. Fuente importante en publicación de PPO [1]. *a*) La fuente [1] facilita un enlace al repositorio de GitHub respectivo con ejemplos debidamente documentados.	4 horas
17/04/2024	**4**. Reunión de seguimiento con el asesor del proyecto.	*a*) Revisión de avance en el código y errores de forma. *b*) Se acordó continuar con el interés en los métodos como PPO como opción.	2 horas

17/04/2024	**5**. Búsqueda de métodos para el manejo de valores continuos en DQN.	*a*) Las opciones popularmente mencionadas son el *actor-critic* y el $DDPG$. Se procede a investigar $DDPG$ *b*) La opción de discretizar el *action space* también se menciona.	4 horas
17/04/2024	**6**. Prueba de discretización del *action space* del env $Pendulum$.	*a*) Se logró adaptar el código del $PendulumDQN$ a una versión discretizada $PendulumDQN\_discrete$, depende principalmente de la resolución seleccionada ($n\_actions$). *b*) Pruebas de entrenamiento de hasta 100 episodios.	6 horas
19/04/2024	**7**. Revisión de implementación del $DDPG$ y el PPO.	*a*) Se buscaron ejemplos para aplicar $DDPG$ mediante PyTorch para evitar ajustes innecesarios. *b*) Estudio de otros repositorios para aplicar PPO.	4 horas
20/04/2024	**8**. Montaje y primera prueba del código PPO para $Pendulum$.	*a*) Revisión del error por cambio de versión Gym a $Gymnasium$. *b*) Implementación del procesamiento $CUDA$ en el código.	8 horas
21/04/2024	**9**. Prueba de entrenamiento con versión base del PPO para $Pendulum$.	*a*) Entrenamiento del modelo con $render(\text{"}human\text{"})$ de $200k$ episodios. Mal desempeño. *b*) Entrenamiento del modelo con $render(\text{"}rgb\_array\text{"})$ de $200M$ de episodios. Interrumpido por tiempo.	6 horas

21/04/2024	**10**. Adición de la función *calculate_reward*() al código base *PPO* de *Pendulum*.	*a*) Ajuste de la función para retornar valores *float*, no *torch.tensor*(). *b*) Ajuste de las funciones *log_summary*() para una mejor apreciación del proceso de entrenamiento y prueba. *c*) Estudio del uso de *Tensorboard* para procesos de entrenamiento de *Machine Learning*.	6 horas
22/04/2024	**11**. Pruebas de implementación del método *DDPG* para *Pendulum*.	*a*) Montaje del código en carpeta *PruebasDDPG* y se realizaron las primeras pruebas de entrenamiento. Errores de forma en el proceso.	4 horas
22/04/2024	**12**. Pruebas de entrenamiento de *PendulumPPO*.	*a*) Entrenamiento de modelo con $1M$ de episodios. Se muestra con avance prometedor en cuanto a las *rewards*.	6 horas
23/04/2024	**13**. Pruebas montaje e incorporación del código *pahm_model.py* proporcionado por el profesor asesor.	*a*) Primeras pruebas de implementación y adición del proceso método *PPO* al entrenamiento.	4 horas
		Total de horas de trabajo:	64 horas

Contenidos de actividades

Primera prueba de entrenamiento *PPO*

El código base utilizado [1] cuentaba con un valor de *timesteps* por defecto de docientos millones ($200M$), por lo que luego de adaptar el código para su funcionamiento mediante el procesamiento *CUDA*, se probó cumplir con esta cantidad, como se observa en la Figura 1.

Sin embargo y apesar de observarse un avance en el entrenamiento, luego de un aproximado de tiempo de cuatro horas transcurridas se proyectó un faltante importante, por lo que se decidió interrumpir el proceso para lograr mejoras en otros puntos del proyecto.

```
        main(args)
[5]  150m 29.7s
...  Iteration took: 6.17 secs
     ---------------------------------------------------


     -------------------- Iteration #1220 --------------------
     Average Episodic Length: 200.0
     Average Episodic Return: -145.06
     Average Loss: -0.00434
     Timesteps So Far: 2684000
     Iteration took: 6.98 secs
     ---------------------------------------------------


     -------------------- Iteration #1221 --------------------
     Average Episodic Length: 200.0
     Average Episodic Return: -157.11
     Average Loss: -0.00127
     Timesteps So Far: 2686200
     Iteration took: 8.02 secs
     ---------------------------------------------------
```

Figure 1: Proceso de entrenamiento del modelo para $PendulumPPO$ con $200M$ de episodios.

Ajuste en la función $log\_summary()$ de PPO

Se buscó una mejora en la forma de monitoreo del avance del entrenamiento del modelo, por lo que agregó la lógica de la librería $Matplotlib$ y se probó con un entrenamiento inicial de $100k$ episodios como se muestra en la Figura 2.

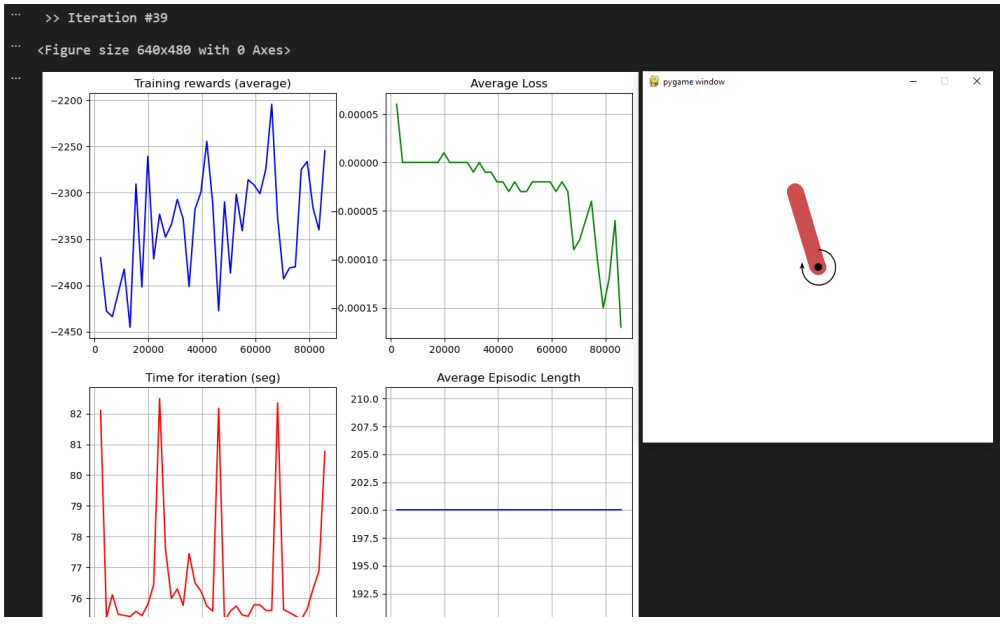

Figure 2: Proceso de entrenamiento del modelo para $PendulumPPO$ con $100k$ episodios.

También se empezó con pruebas simples de monitoreo mediante la librería $Tensorboard$, las cuales se encuentran en proceso.

Entrenamiento de modelo PPO de $1M$ de episodios

Luego de ajustar el monitoreo y agregar la lógica del $target\_angle$ al código, se procedió a entrenar un modelo con 1 millon de episodios y fijar el ángulo en 135^o, Figura 3, mostrando un desempeño prometedor.

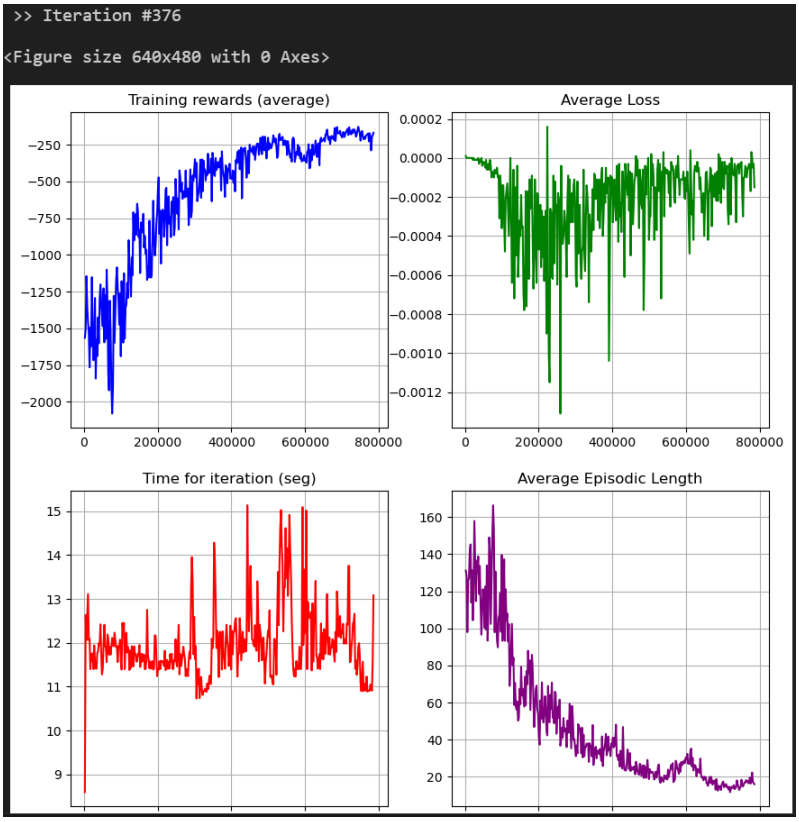

Figure 3: Proceso de entrenamiento del modelo para $PendulumPPO$ con $1M$ de episodios y $target\_angle = 135$.

Se encuentra en proceso de implementación la variación constante del ángulo y más entrenamientos, dada la necesidad de periodos largos de tiempo para entrenamientos de modelos prometedores.

Referencias

[1] E. Yang-Yu, "Coding ppo from scratch with pytorch (part 1/4)," *Medium*, 2020.