

Universidad Nacional Autónoma de México.

Aprendizaje Profundo

Neuroevolución profunda: PSO es una alternativa competitiva para entrenar redes neuronales profundas para aprendizaje por refuerzo.

Proyecto final de semestre

Yoshio Ismael Martínez Arévalo

Profesores:

Dr. Gibran Fuentes Pineda

Dra. Berenice Montalvo Lezama.

Dr. Ricardo Montalvo Lezama.

Diciembre 2019



Contenido

1. Problema

2. Reproducción del artículo.

3. Modificaciones.

- 3.1. PSO
- 3.2. Biblioteca GYM
- 3.3. Implementaciones
 - 3.3.1. GPU
 - 3.3.2. Paralelizada en GPU y CPU
 - 3.3.3. Paralelizada en CPU



4. Resultados y discusión.

- 4.1 Tiempos de entrenamiento.
- 4.2 Comparaciones.
- 4.3 Discusión de políticas encontradas por la red.

5. Conclusiones.

1. Problema

Recientemente se demostró que en algunos dominios las redes neuronales optimizadas con el algoritmo del descenso del gradiente no tienen buenos resultados. El grupo de investigación de Uber IA labs, publico en el año 2018 un artículo llamado “Neuroevolución profunda: los algoritmos genéticos son una alternativa competitiva para entrenar redes neuronales profundas para aprendizaje por refuerzo” donde se compara el rendimiento de una red neuronal profunda que fue entrenada con diferentes métodos (algoritmos genéticos, DQN, estrategias evolutivas etcétera) para jugar juegos de Atari. En el artículo se proponen los algoritmos genéticos para entrenar redes neuronales con el objetivo de jugar juegos de Atari y tareas de locomoción bípeda, sorprendentemente obtuvieron buenos resultados en algunos juegos y peores en otros, por lo tanto, los algoritmos genéticos pueden considerarse como una herramienta más para el entrenamiento de redes neuronales para aprendizaje por refuerzo. Con esto nos damos cuenta que es necesario tener más herramientas además de seguir el gradiente ya que ninguna es una herramienta general.

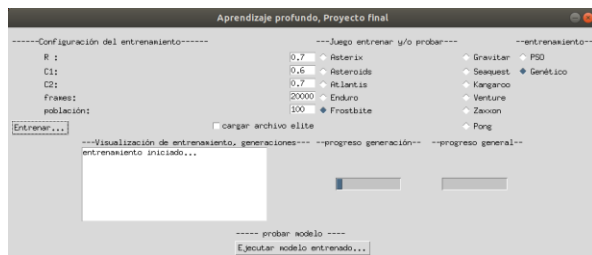
2. Reproducción del artículo.

Se realizaron dos reproducciones del artículo, una implementada desde cero y la otra con el código de los autores del artículo.

Se redujo el tamaño de la fotografía de entrada (64 x 64) con el fin de reducir la longitud de los vectores individuos. Se implemento una versión más simplificada del artículo con las siguientes características:

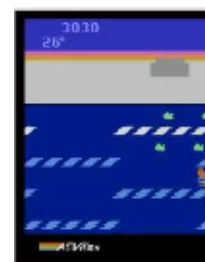
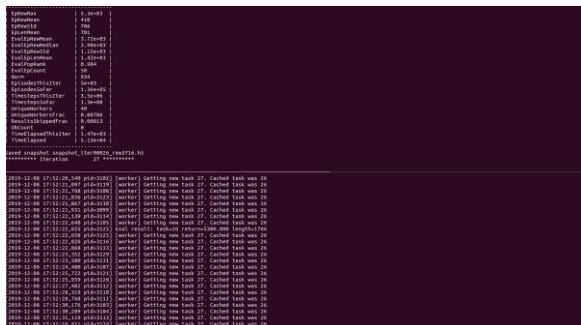
- **Frames totales:** 20M
- **Frames por individuo:** 20,000
- **Población inicial:** 100

La prueba se realizo con el juego de “Frostbite”



Se obtuvo un puntaje de 160 en promedio.

De igual manera se reprodujo el articulo con su código original (<https://github.com/uber-research/deep-neuroevolution>).



Obteniendo un puntaje promedio de 3000.

Se obtuvieron mejores resultados con el código del articulo ya que se incremento la población y los subprocesos de trabajo así como también el código del articulo trabaja con un servidor llamado “redis” (<https://redis.io/>). Estas versiones ejecutan subprocesos independientes

3. Modificaciones

En esta sección se muestran las modificaciones que se realizaron en el artículo con el motivo de ver el comportamiento durante el entrenamiento con otro optimizador.

Código del proyecto:

<https://github.com/Yoshiolsmael/aprendizaje profundo>

3.1. Optimización por enjambre de partículas (particle swarm optimization, PSO).

Este método fue descrito alrededor de 1995 por James Kennedy y Russell C. Eberhart. La optimización por enjambre de partículas es un método de optimización que modela el comportamiento social, como por ejemplo el movimiento de organismos vivos en un banco de peces o en una parvada de aves.

Cada partícula tiene una posición, p , en el espacio de búsqueda y una velocidad, v , que determina su movimiento a través del espacio.

La velocidad es actualizada para el tiempo $t+1$ con la siguiente formula:

$$v_i(t+1) = v_i(t) + c_1 \cdot r_1 \cdot (p_i^{best} - p_i(t)) + c_2 \cdot r_2 \cdot (p_{gbest} - p_i(t))$$

Una vez calculada la velocidad se procede a calcular la posición con la siguiente formula:

$$p_i(t+1) = p_i(t) + v_i(t)$$

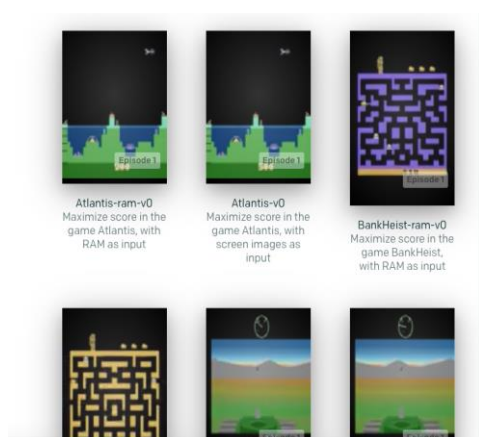
El algoritmo es:

```
Asignar posiciones y velocidades aleatorias iniciales a las partículas
Repetir
  Cada partícula:
    Actualizar su velocidad considerando:
      Inercia de la partícula (la hace seguir con la misma velocidad)
      Atracción al mejor personal
      Atracción al mejor global
    Actualizar la posición de la partícula
    Calcular el valor de fitness en la nueva posición
    Actualizar su mejor personal
    Actualizar el mejor global del sistema
  Devolver el mejor global
```

3. Modificaciones.

3.2. Biblioteca GYM.

GYM es un kit de herramientas para desarrollar y comparar algoritmos de aprendizaje por refuerzo. Incluye ambientes para enseñar a jugar juegos de Atari a un agente y también entornos para enseñar a caminar a humanoides.





3.3. Implementaciones.

A diferencia del algoritmo que se utilizó en el artículo de Uber (el algoritmo genético), aquí se utilizó PSO para optimizar los pesos de la red neuronal, así como también se requirió de la biblioteca GYM que nos permitirá probar en un entorno nuestra red neuronal.

Para ello se crearon distintas clases y módulos que se mencionan a continuación:

Clases:

-  **agente (dentro de juego.py):**
Contiene los métodos que permiten entrenar o probar la red neuronal en un juego.
-  **IndividuoPSO:**
Clase que representa una partícula, contiene sus métodos para el movimiento y sus vectores de posición y velocidad.

3. Modificaciones

+ **RedNeuronal:**

Clase que contiene la implementación de la red neuronal.

+ **Subproceso:**

Clase que crea un hilo de trabajo con el objetivo de evaluar un agente.

Módulos:

+ **Ventana:**

Este es el archivo principal, crea una ventana grafica con elementos visuales que permiten configurar el entrenamiento o probar algún juego.

+ **ServiciosPoblacion:**

Contiene todos los métodos que nos permiten convertir la estructura de los pesos de una red neuronal en un vector de pesos y de igual manera de forma inversa.

+ **archivos:**

Contiene los métodos que nos permiten guardar y recuperar soluciones encontradas durante el entrenamiento, y recuperar la mejor solución para las pruebas.

+ **Evaluador:**

Contiene métodos que permiten evaluar el rendimiento de un individuo.

Los parámetros utilizados fueron los siguientes:

$R = 0.9$

$C1 = 1.6$

$C2 = 1.7$

Población inicial = 200

Frames totales = 20M

Frames por individuo = 20,000

Vector de velocidad inicial con valor de 1 (todos los elementos del vector)

3. Modificaciones

La red neuronal que se utilizo fue la misma que la del articulo original.

Convolutacional con 32 filtros de 8 x 8 con stride de 4 con rectificador lineal de salida.

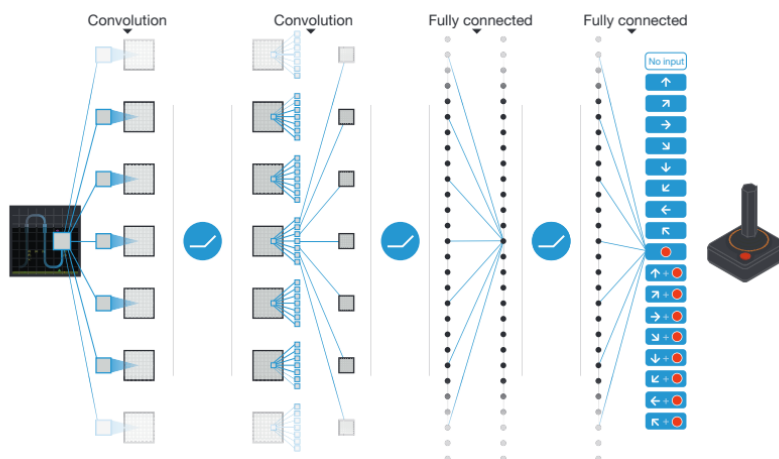
Convolutacional con 64 filtros de 4 x 4 con stride de 2 con rectificador lineal de salida.

Convolutacional con 64 filtros de 3 x 3 con stride de 1 con rectificador lineal de salida.

Capa oculta final, completamente conectada con 512 unidades rectificadoras.

Capa de salida es una completamente conectada con una salida para cada acción.

El número de acciones varía entre [4-18].



3. Modificaciones.

3.3.1 GPU.

Se implemento una versión donde se hizo uso de una tarjeta de video con el objetivo de acelerar el entrenamiento, sin embargo, el tiempo de entrenamiento de esta fue de 23 horas. La idea de esta implementación fue tener un agente jugando el juego para obtener el puntaje del individuo actual (el vector de pesos), esto es secuencial para cada uno de los individuos. De igual manera se realizo una modificación extra donde cada individuo se evaluaba en un subproceso separado, se crearon tantos subprocesos como individuos tenia la población, esto incremento un poco más el tiempo de entrenamiento llevándolo hasta las 25 horas. Esto puede deberse a las características del equipo.

3.3.2 Paralelizada en GPU y CPU

Se realiza una implementación dividiendo la población, donde una tercera parte de la población es evaluada en la GPU y el resto en la CPU, cada uno de los individuos en un hilo por separado dando como resultado un mejor rendimiento que el uso del GPU únicamente, reduciendo las horas a 20. Esto puede deberse a las características del equipo.

3.3.3 Paralelizada en CPU.

Se realizo una implementación más, en donde se hizo uso exclusivo del CPU, se crearon tantos subprocesos como individuos se tenían en la población, esto redujo el tiempo a 18 horas y se decidió usar esta implementación. Esto puede deberse a las características del equipo.

4. Resultados y discusión

En esta sección mostraremos los resultados que se obtuvieron al implementar la red neuronal y entrenarla con PSO, de igual manera se hace una discusión sobre la calidad de las políticas encontradas por la red.

4.1 Tiempos de entrenamiento.

Como se comentaba en la sección anterior, antes de realizar el entrenamiento completo de los juegos se decidió probar distintas implementaciones de los algoritmos con el objetivo de ver si existían mejores entre ellas, la diferencia de todas ellas fue únicamente el consumo de recursos y la velocidad de entrenamiento, sin embargo, lo que se busca es velocidad en el entrenamiento por lo que se notó que el uso de una GPU demoraba el entrenamiento en algunas horas, tanto de manera síncrona y asíncrona. La implementación del uso exclusivo de una CPU paralelizada resultó en buenos resultados, aunque no se tiene una explicación respecto a lo que sucedió.

El tiempo promedio de entrenamiento fue de 18 horas para cada uno de los juegos, con una media de 80 generaciones por juego.

Cabe destacar que, para realizar las apropiadas comparaciones con el artículo, cada juego fue corrido 3 veces independientemente y su mejor elite fue evaluado 200 veces donde el promedio es el puntaje final que mostramos en la siguiente sección. **Esto da poco más 2 días por juego.**

4. Resultados y discusión.

4.2 Comparaciones.

Comparado con los resultados del artículo aquí se muestran los resultados obtenidos.

La ultima columna de la derecha (PSO) muestra en color azul aquellos juegos en donde PSO obtuvo un mejor puntaje que algunas de las técnicas que usan gradiente, el color verde en GA indica lo mismo.

Estos resultados se obtuvieron con una menor cantidad de Frames a diferencia de otras técnicas, lo que sugiere que PSO también puede ser útil para entrenar redes neuronales para este tipo de problemas.

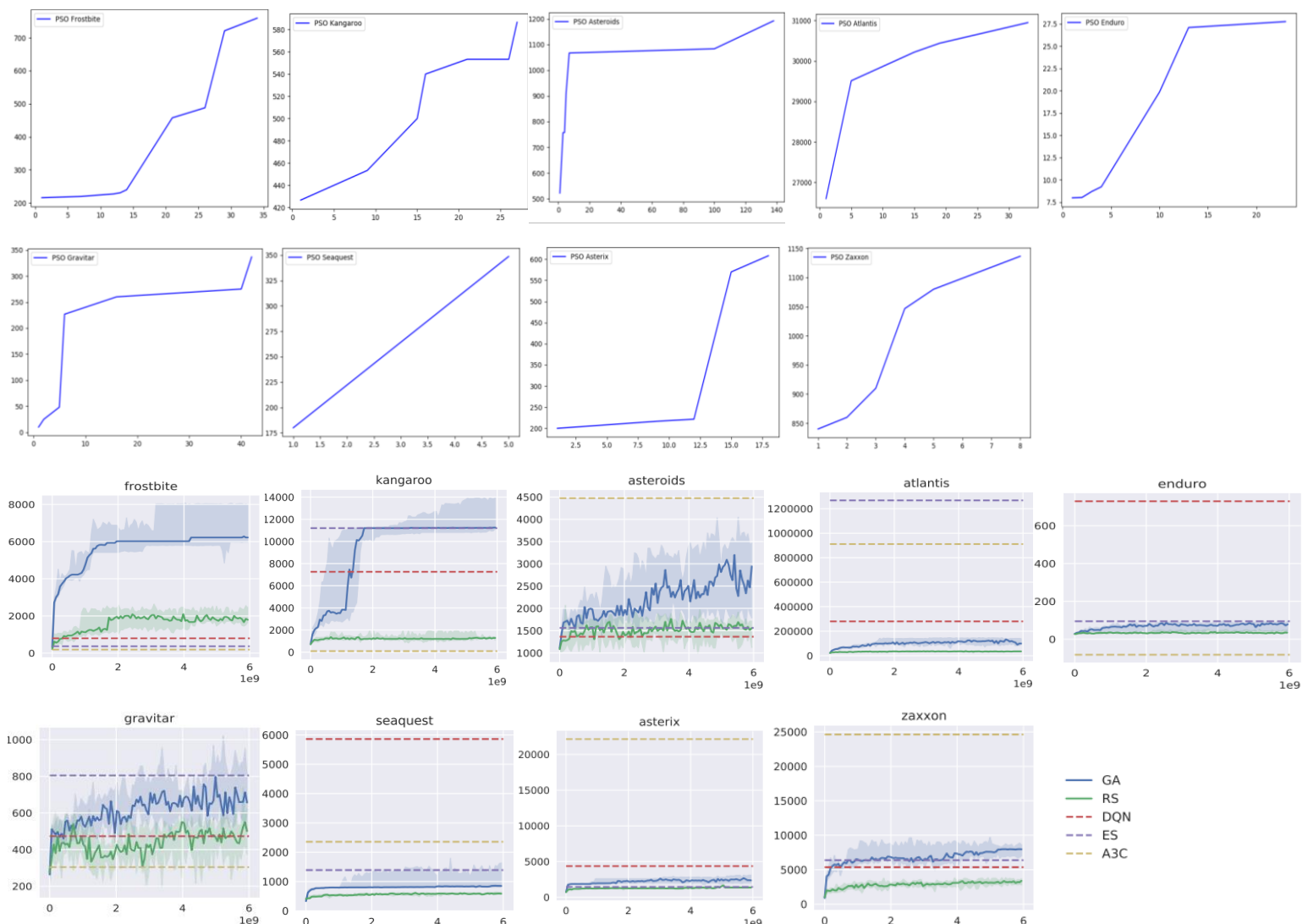
El procedimiento fue el siguiente:

Cada una de las 200 partículas se evalúan en un subproceso independiente, es decir, se ponen a jugar un juego, después se ordenan y se obtienen las mejores 5 partículas (las de mejor puntaje) y cada una de ellas es evaluada por 3 veces en un proceso independiente, una vez terminado se ordenan y se obtiene la mejor partícula de todas, después esa partícula es evaluada en 200 procesos independientes, el promedio final es el registrado a continuación.

Juego	DQN (200M)	ES(1B)	A3C (1B)	RS (1B)	GA (1B)	GA(6B)	PSO (20M)
Asterix	4,349	1,440	22,140	1,197	1,850	2,255	617
asteroids	1,365	1,562	4,475	1,307	1,661	2,700	1072
atlantis	279,987	1,267,410	911,091	26,371	76,273	129,167	26,265
enduro	729	95	-82	36	60	80	30
frostbite	797	370	191	1,164	4,536	6,220	550
gravitar	473	805	304	431	476	764	296
kangaroo	7,259	11,200	94	1,099	3,790	11,254	579
seaquest	5,861	1,390	2,355	503	798	850	334
venture	163	760	23	488	969	1,422	0
zaxxon	5,363	6,380	24,622	2,538	6,180	7,864	975

4. Resultados y discusión.

De estas graficas podemos observar las de PSO aun pueden incrementar más, y las del algoritmo genetico se quedan “estancadas”, es decir que podemos llegar a tener el mismo rendimiento o aun mejor si incrementamos la población y el número de frames.



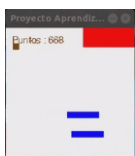
4. Resultados y discusión.

4.3. Discusión de políticas encontradas por la red.

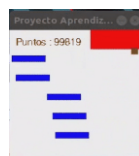
Para poder comprender que calidad de políticas se estaban encontrando se desarrollo un juego extra con ayuda de la biblioteca Pygame, un juego muy sencillo y limitado en las posibles acciones que puede tomar, es decir, acciones necesarias para jugarlo se omitieron con el fin de saber que hacia la red en esos casos y que políticas encontraba. Fue entrenado con la misma arquitectura de red y los mismos parámetros, con una población de 50 partículas, 5000 frames para cada agente y un total de 50000 frames,

El juego tiene 4 niveles, en el primer nivel aparecen dos barras azules moviéndose rápidamente de forma horizontal, el agente (cuadrado marrón) debe cruzar las barras sin que lo toquen ya que le restan puntos, y debe de llegar hasta el rectángulo rojo, donde recibe una enorme recompensa, una vez en la meta pasa al siguiente nivel (**Se anexa imagen GIF en repositorio**).

Nivel 1



Ultimo nivel



Las posibles acciones que puede realizar el agente son mover arriba, derecha e izquierda, sin embargo, nos damos cuenta que necesitamos detenernos para no chocar con las barras en los ultimos niveles y el agente “aprendio” la acción de detener haciendo movimientos de derecha a izquierda (oscilando) lo que le permitia detenerse, de igual manera con el paso de las generaciones “aprendio” a evitar las barras azules ya que ellas le quitaban puntos. Con este sencillo ejemplo se observo que las politicas generadas son muy buenas ya que permitian evitar obstaculos y crear acciones nuevas con ayuda de las que se tenian disponibles.

5. Conclusiones

Como lo hemos visto anteriormente, el uso del algoritmo PSO puede entrenar una red neuronal para aprender a jugar juegos de Atari, es decir, pequeños juegos en 2D. Sin embargo, a pesar de las limitaciones que se tuvieron se puede estimar que con un mejor equipo e incrementando la cantidad de frames y población se puede llegar a tener excelentes resultados en el entrenamiento. Lo que podemos concluir es que en estos dominios donde existe mucha incertidumbre porque son ambientes muy dinámicos, el uso de gradiente puede llegar a tardar mucho tiempo en converger en una buena política, sin embargo, PSO trabaja en ambiente dinámicos, ya que está inspirado en comportamientos antes mencionados (parvadas de aves, enjambres de abejas etcétera), entonces puede evitar (saltar) de una mejor manera estos mínimos locales en lo que se puede llegar a quedar el gradiente, pero no significa que puede llegar a ser mejor, ya que como se observó, también tuvo malos puntajes en varios juegos, por lo tanto podemos decir que de igual manera que los algoritmos genéticos, PSO puede ser considerado como una herramienta más para este tipo de problemas.

