# General videogame Learning with Neural-evolution

Leonardo Quiñonez, Jonatan Gomez

***Abstract*—This paper describes the development of a general learning test, in which an agent's ability to learn to play different games is tested. We used a neuro-evolved agent, which main feature is the use of raw pixels as input, in contrast with common approaches that require some feature extraction defined by an expert. To evaluate the agents we used two games: *Pong* and *Breakout*. With these games a cross learning test is used to visualize the knowledge transfer ability of the agents.**

***Keywords*—intelligent systems; general game learning; genetic algorithms**

## I. Introducción

El desarrollo de agentes con la capacidad de actuar de forma inteligente en diversos juegos siempre ha despertado interés. Un ejemplo de este interés se ve en el ajedrez, que al ser relacionado con la capacidad intelectual del individuo, ha sido uno de los juegos que constantemente se ha seleccionado como hito para el desarrollo de agentes inteligentes, buscando superar al ser humano. Una muestra de esto es El Turco [1] construido por Wolfgang von Kempelen en el siglo XIX. El falso autómata era un conjunto mecánico de piezas que supuestamente se articulaban para jugar y vencer en ajedrez, pero en realidad una persona en el interior realizaba el movimiento de las piezas mediante dichos mecanismos. Solo hasta 1997 una versión del agente computacional Deep Blue [2] le ganó a Garry Kasparov, el campeón mundial de ajedrez de ese momento.

Para diseñar un agente como Deep Blue, normalmente, un experto en el dominio del juego extrae las características más importantes y diseña las heurísticas que finalmente producirán un agente con alto desempeño en dicho juego, pero que es incapaz de cambiar de dominio; Deep Blue no es capaz de jugar damas sin un cambio en su código.

Para motivar el desarrollo de agentes, más generales, se han propuesto juegos como *Metagame* [3] y competencias como la *General Game Playing Competition* [4]. En esta última los agentes se enfrentan a múltiples juegos (desconocidos para estos agentes) concentrándose en el desarrollo de estrategias y no en el aprendizaje del juego. En general, los agentes que participan en dicha competencia reciben las reglas de los juegos por medio de un lenguaje con principios de lógica [5]. Esto presenta una limitación en el aprendizaje general, ya que la exploración de las reglas del juego es una fase importante para un aprendizaje más cercano a un comportamiento humano.

Adicionalmente, Bellemare [6] presenta una plataforma para evaluar agentes generales con múltiples juegos, llamada ALE, en la que usando un emulador de la consola Atari 2600, se permite la competencia y el aprendizaje de los agentes sobre los múltiples juegos desarrollados para dicha consola. El desarrollo de agentes para jugar videojuegos, como los de la consola Atari 2600, no es nuevo. Se han desarrollado varios agentes [7] [8], con la habilidad de actuar de forma eficiente en un juego en particular, teniendo la misma limitación de Deep Blue, la falta de generalidad. A pesar de esto, hay algunos agentes que están diseñados para actuar en múltiples contextos. Hausknecht [9] presenta una prueba y comparación de algunos de estos agentes, en particular, se comparan un agente aleatorio, un agente que usa aprendizaje por refuerzo (Sarsa) y un agente que usa neuro-evolución. A dichos agentes se les presentan tres tipos de entradas diferentes: los píxeles del juego, ruido determinístico y características extraídas de los objetos o figuras de los juegos. Las diferentes combinaciones de agentes y tipos de entradas son probadas, entre estas se obtiene un mejor desempeño con los agentes cuyas entradas son las características definidas para cada juego, es decir, los que no son generales; el mejor de todos los agentes utiliza estas entradas y neuro-evolución.

En este trabajo se propone realizar una prueba de aprendizaje general con dos juegos simples; *Breakout* y *Pong*, usando ALE. La prueba se realiza con agentes cuyas acciones están determinadas por una red neuronal *feed-forward* con una capa oculta; las entradas de la red son los píxeles en blanco y negro de los juegos y las salidas se reducen a dos posibles acciones, que corresponden a moverse hacia la izquierda o hacia la derecha en el juego de breakout y hacia arriba o hacia abajo en el juego de *Pong*. Para entrenar la red neuronal, se utiliza un algoritmo evolutivo, donde la función objetivo está representada por los puntajes del agente en el juego. De esta manera, se observa como las redes pueden usar el conocimiento aprendido en uno de los juegos, para mejorar sus resultados y su tasa de aprendizaje en otro juego. Esto tiene en cuenta la transferencia de conocimiento realizada por los humanos, en donde el aprendizaje de una habilidad permite un aprendizaje más rápido de otra habilidad cercana. [10]

El trabajo se encuentra organizado de la siguiente forma: en la sección II se presentan los antecedentes relevantes, donde se revisan los agentes de videojuegos y generales. Se continúa, en la sección III, con la descripción del modelo utilizado y en la sección IV se describen las pruebas realizadas, exponiendo los detalles de estas. Luego en la sección V se muestran los resultados obtenidos, seguidos del análisis de estos y finalmente en las secciones VI y VII se encuentran las conclusiones y el trabajo futuro.

## II. Antecedentes

### A. Agentes en Videojuegos

Los videojuegos son una fuente para motivar el desarrollo de agentes, ya sean para que estos hagan el papel de oponentes

Grupo de investigación Alife, Universidad Nacional de Colombia {laquinonezp,jgomezpe}@unal.edu.co

978-1-4799-6717-9/14$31.00 

o para contrastar las habilidades de un agente contra las de un ser humano. El primer caso es muy común, donde los primeros oponentes de videojuegos estaban basados en un conjunto de reglas que se ejecutaban de forma secuencial, luego para obtener mejores agentes se han usado técnicas de aprendizaje por refuerzo, que llevan al desarrollo de reglas dinámicas [11]. El segundo caso es utilizado constantemente como herramienta para probar agentes inteligentes, un ejemplo de esto se tiene con el juego de *Mario*, como lo utiliza Mohan en [8]. En dicho artículo, se presentan diferentes formas de diseñar agentes inteligentes usando aprendizaje por refuerzo. Los diferentes agentes usan heurísticas específicas para el juego, obteniendo así agentes capaces de jugar *Infinite Mario*, una variante del juego *Super Mario* de Nintendo.

Múltiples juegos se han empleado con este propósito de motivar el desarrollo de agentes, por ejemplo *Pong* es usado por Nadh en [7], donde un agente, utilizando memorias asociativas, aprende a actuar en dicho juego. En el artículo mencionado los autores obtienen un resultado de 90% de eficiencia al golpear la bola, pero no realizan comparaciones con otros juegos. Robles en [12] utiliza un árbol de búsquedas para jugar Ms. Pac-Man y Buro en [13] menciona una competencia para desarrollar agentes que jueguen Starcraft, siendo estos algunos ejemplos de los juegos para los cuales se han desarrollado agentes inteligentes.

### *B. Agentes generales*

A partir del 2005, pero basado en ideas anteriores [14], existe una corriente de trabajos orientada al desarrollo de agentes generales. Uno de los mejores ejemplos es la *General Game Playing Competition* [4] organizada por Stanford, en ésta, agentes generales se enfrentan entre ellos en juegos desconocidos [15]. Partiendo de ésta competencia se han desarrollado múltiples agentes generales [16] [17] [18], donde la mayoría utilizan técnicas de exploración del árbol de jugadas (conocidas como búsqueda min-max) basadas en la propuesta por John McCarthy en 1956 [19].

### *C. Agentes en ALE*

El ambiente de aprendizaje de Arcade (ALE por sus siglas en inglés) [6], es una plataforma basada en *Atari 2600*, una popular consola de videojuegos con múltiples juegos ya implementados. Esto permite tener una cantidad importante de juegos sencillos y variados para probar agentes que busquen inteligencia artificial general, donde además se evitan sesgos de los investigadores al tener juegos desarrollados por terceros. Adicionalmente, el procesador del Atari 2600, tenía una velocidad inferior a los computadores actuales, lo cual permite ejecutar simulaciones a una velocidad elevada, facilitando la realización de pruebas.

ALE usa un simulador de código abierto llamado Stella [20], partiendo de este simulador se realiza la comunicación de las acciones, los puntajes y los píxeles de la pantalla (160x210). Esta información se transfiere por medio de tuberías (pipes en inglés), donde se tiene una tubería para enviar información al agente y otra para recibir información. Esta estructura permite implementar los agentes en cualquier lenguaje de programación y comunicarlos con el simulador; para este trabajo, se parte del ejemplo de agente (en Java), presentado por los autores del artículo anterior.

Alrededor de ALE, se han realizado varios experimentos con neuro-evolución, por ejemplo, Hausnecht en [21] utiliza un algoritmo llamado HyperNEAT, que se prueba con dos juegos diferentes: Asterix y Freeway. Para ello se utiliza una extracción de objetos del juego y también se busca identificar el propio agente en el juego, por medio de la información sobre su movimiento. Esto último se logra al marcar como el agente a los conjuntos de puntos en pantalla cuyo movimiento tiene alta relación con las acciones tomadas. En dicho artículo se obtienen resultados de los agentes sobre los juegos mencionados, pero no se realizan pruebas de aprendizaje general, en donde se utilice el conocimiento de un juego para probar si esto aumenta su eficiencia en otro juego.

Hausnecht en [9] realiza una extensión del artículo anterior, en donde se utilizan diferentes métodos como aprendizaje por refuerzo y neuro-evolución, entre otros. Usando dichos métodos y con diferentes estrategias para manipular los píxeles de entrada, se realiza una comparación con más de 30 juegos.

## III. MODELO

Para realizar la experimentación se utiliza la plataforma ALE, por medio de la cual los agentes pueden obtener la información del juego: píxeles de la pantalla y valor del puntaje. Los agentes en Java, reciben dicha información a través tuberías y luego calculan la acción correspondiente, la cual se envía usando otra tubería.

Debido a la cantidad de píxeles de la pantalla (33,600), el procesamiento de estos requiere demasiado tiempo, haciendo a los agentes poco prácticos. Para simplificar esto, se reduce el espacio de entrada recibido por los agentes por medio de dos pasos; primero se eliminan los colores, dejando una representación en blanco y negro de la pantalla; luego se realizan grupos de 100 píxeles, 10 de ancho y 10 de largo, logrando una representación del espacio en 16 por 21 píxeles (ver figuras 2 y 3). Esta reducción de espacio es equivalente a la realizada en [9].

Para generar las acciones a ejecutar en los diferentes juegos, se utiliza una red neuronal feed-forward [22] totalmente conexa, con una función sigmoidea como función de activación. Dicha red tiene tantas neuronas de entrada como píxeles (16x21), una capa oculta de 30 neuronas y una capa de salida con dos neuronas que indican la acción a tomar (ver Figura 1). La cantidad de neuronas en la capa oculta se elige para mantener un balance entre la complejidad de la red y la eficiencia del agente. Para seleccionar este número se realizan pruebas con 20, 30 y 40 neuronas ocultas, en las cuales se encuentra que el mejor resultado se obtiene con 30 neuronas (ver tabla I); para mayores cantidades de neuronas ocultas se requiere demasiado tiempo de procesamiento y por esto no son consideradas.

Tabla I. PRUEBAS CON DIFERENTES CANTIDADES DE NEURONAS OCULTAS

| Num Ocultas | Puntaje |
|---|---|
| **20** | 5.6 |
| **30** | 6.5 |
| **40** | 6.1 |

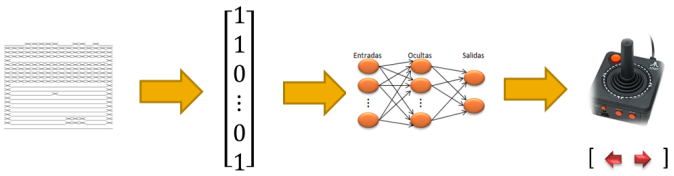


Fig. 1. Estructura de un Agente

### A. *Algoritmo Evolutivo*

El entrenamiento de la red neuronal se realiza por medio de un algoritmo evolutivo, donde los pesos de las redes se van modificando en busca de la red que obtenga la mayor cantidad de puntos en los diferentes juegos. Para ello los individuos están representados por redes neuronales, donde el genotipo de estas es representado por medio de un vector con los pesos de cada conexión, de tal forma que el valor real de la primera posición del vector, representa el peso de la conexión de la primera neurona de entrada con la primera neurona oculta.

Este vector de valores reales se modifica por medio de dos operadores genéticos, una mutación y un cruce, obteniendo con estos un balance entre exploración y explotación. Se elige una mutación gaussiana, donde el peso a mutar se modifica con una distribución de Gauss con sigma igual a 0.01. En la tabla II se muestran los puntajes obtenidos, que llevaron a esta elección de sigma.

Tabla II. PRUEBA CON DIFERENTES VALORES DE SIGMA

| Sigma | Puntaje |
|---|---|
| **0.1** | 7 |
| **0.01** | 11 |
| **0.001** | 4 |

El cruce realizado es equivalente al clásico, donde se elige de forma aleatoria un punto de cruce y luego se intercambian los genes de los dos padres [23]; de esta forma se generan dos hijos con los genes (pesos en la red neuronal) combinados de ambos padres.

La función objetivo del algoritmo evolutivo está representada por el desempeño de la red en el juego sobre el que se realice el aprendizaje, para evaluar esta función se coloca a cada red a interactuar con dicho juego. Debido a que un agente puede obtener puntajes diferentes cada vez que juega, se toma como valor de la función objetivo el promedio de puntaje en tres rondas de juego, con esto se busca que el puntaje de cada agente sea menos variable. Cada ronda de juego está limitada por el número de vidas o por un número de fotogramas, cuando se acaban las vidas o se llega al límite de fotogramas, se toma el puntaje actual como puntaje de la ronda.

## IV. EXPERIMENTOS

Para probar la capacidad de inteligencia general de los agentes, se prueba como se comportan y aprenden, al ser entrenados con un juego y luego se mide su desempeño en el otro juego, buscando con esto visualizar como se transfiere el conocimiento entre los juegos. Para realizar la experimentación se utilizan dos juegos con estructuras similares: *Pong* y *Breakout*. Como parámetros del algoritmo evolutivo, se utiliza una población inicial de 20 individuos y 80 generaciones, donde todos los juegos tienen un límite de 4000 fotogramas lo cual toma 10 horas en promedio usando un equipo con 2.2Ghz de velocidad de procesamiento. A través de las 80 generaciones se cambia el juego sobre el cual son probadas las redes, este cambio se realiza en el 60% de las generaciones. Luego de estas 48 generaciones (60%), las redes entrenadas con un juego, son enfrentadas al otro. De esta forma, se busca comparar la diferencia en el puntaje y aprendizaje de los agentes, cuando ya están entrenados con un juego y cuando no es así. Por último, cada experimento se repite 5 veces y se obtiene el promedio, la desviación estándar y el puntaje máximo obtenido en la última generación.

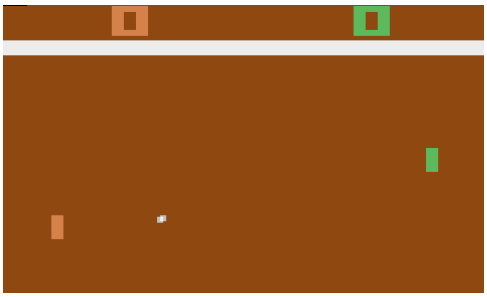

Fig. 2. Juego de *Pong*

### A. *Pong*

*Pong* (Figura 2) es uno de los primeros juegos de video, disponible desde 1977 para la consola Atari 2600. Es un juego simple donde dos jugadores, representados por líneas verticales, deben evitar que la bola entre en su área de gol por medio de movimientos para arriba o para abajo. *Pong* puede ser jugado por uno o dos jugadores, para las pruebas realizadas se utiliza la versión de un solo jugador, donde el oponente es un agente incluido en el juego. Es importante resaltar que, a pesar de que los movimientos son verticales: arriba y abajo, también se pueden realizar con los comandos de derecha e izquierda respectivamente, esto permite que nuestros agentes puedan jugar ambos juegos, generando solo dos acciones. El puntaje en el juego son los goles realizados por cada agente, terminando el juego cuando alguno de estos llega a los 21 goles, para manejar dicho puntaje como un solo número se calcula la diferencia de goles entre los jugadores, de forma que varía entre -21 y 21, donde un valor positivo representa que el agente ganó, y uno negativo que la computadora lo hizo. Por ejemplo, un resultado de 5 goles del agente vs 8 goles del oponente resulta en un puntaje de -3.

### B. *Breakout*

*Breakout* (Figura 4) es un juego influenciado por *Pong*, en donde se tiene una línea de ladrillos en la parte de arriba de la pantalla y una bola que es capaz de romper dichos ladrillos al golpearlos. La bola rebota por la pantalla hasta caer a la parte baja, donde debe rebotar sobre una línea horizontal corta manipulada por el jugador. De esta forma el jugador dirige la bola para romper los diferentes ladrillos; en caso de no golpear la bola, esta caerá y el jugador perderá una vida (se inicia con cinco vidas). El puntaje del juego es proporcional a la cantidad de ladrillos rotos, por lo cual es un entero positivo, siendo el mínimo cero.

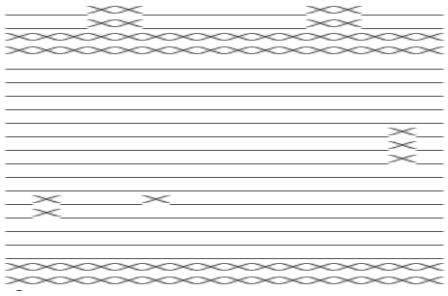

Fig. 3. Juego de *Pong* con el espacio reducido, con unos y ceros representados como 'X' y '_' respectivamente

Fig. 4. Juego de *Breakout*

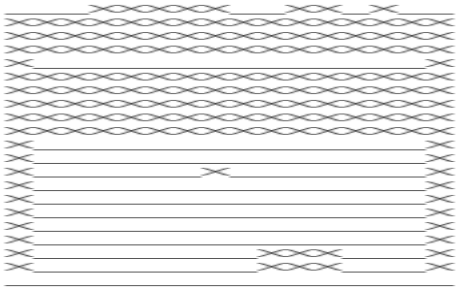

Fig. 5. Juego de *Breakout* con el espacio reducido, con unos y ceros representados como 'X' y '_' respectivamente

## V. Resultados y Análisis

En la tabla III se observan los resultados obtenidos, en las filas se encuentran diferentes agentes y en las columnas su puntaje obtenido en cada juego. La primera columna identifica a cada agente y las columnas restantes muestran el promedio de puntaje y la desviación estándar de cada juego. Para mantener la tabla compacta se utiliza la notación *Br* para referirse al juego *Breakout* y *Po* para referirse al juego *Pong*.

En las dos primeras filas se muestran los resultados obtenidos con el agente desarrollado en este trabajo; en la primera fila el agente fue entrenado inicialmente con *Breakout* y luego con *Pong* y en la segunda fila se intercambia el orden de los juegos. De la fila tercera a la sexta se presentan los resultados para establecer límites máximos y mínimos esperados, teniendo así los puntajes de un agente aleatorio y el máximo puntaje reportado por un humano. En la tercera fila se presentan los resultados de un agente aleatorio usando la misma arquitectura que nuestros agentes, para este agente se realizan diez repeticiones. En la cuarta línea se observan los puntajes máximos de un ser humano, para el juego de

Tabla III. Resultados

| Agente | Br Avg | Po Avg | Br Std | Po Std |
|---|---|---|---|---|
| **Neuro-Ev Br,Po** | **9.7** | **-19** | **3.07** | **1.11** |
| **Neuro-Ev Po,Br** | **7.7** | **-13** | **1.10** | **1.28** |
| **Aleatorio** | 0.7 | -20 | 1.25 | 1.89 |
| **Humano** | 864 | 21 | – | – |
| **Aleatorio [9]** | 0.8 | -20.7 | – | – |
| **Humano [9]** | 825 | – | – | – |
| **Pixel [9]** | 4 | -16 | – | – |
| **FT NEAT [9]** | 17 | -11 | – | – |

*Breakout*, el puntaje se toma de [24] y para el juego de *Pong*, el máximo puntaje posible es 21. Las dos filas siguientes, quinta y sexta presentan estos mismos resultados según son reportados por Hausnecht en [21]. Respecto a esto se observa, como los puntajes de nuestro agente aleatorio son similares a los puntajes que obtuvo Hausnecht y el máximo puntaje reportado por un humano en *Breakout* se incrementó en 40 puntos desde que Hausnecht reportó este puntaje.

En las últimas filas se muestran los resultados obtenidos por los agentes presentados por Hausnecht. En la séptima fila se tiene un agente que no realiza extracción de características y en cambio usa los píxeles del juego como entradas, de la misma manera como se realizó con nuestros agentes, por ello nuestros resultados son comparables con este agente. En la última fila se encuentra el puntaje obtenido por Hausnecht usando extracción de características y una versión del algoritmo NEAT [21], llamada FT Neat, en donde los operadores genéticos mantienen estable la estructura de la red.

Basado en la tabla III se pueden comparar los resultados de nuestros agentes, con los de Hausnecht. El agente entrenado con *Breakout* y luego con *Pong* obtiene un resultado mayor que el Pixel de Hausnecht en el juego de *Breakout* (9.76 contra 4), de la misma manera como el agente entrenado con *Pong* y luego *Breakout* obtiene un resultado mejor en el juego de *Pong* (-13 contra -16). Luego, al revisar la transferencia de conocimiento, se observa que los resultados disminuyen y nuestros agentes, aunque logran superar al agente aleatorio (-19 contra -20 y 7.7 contra 0.7), no logran superar a Pixel, como si lo hacían en el caso anterior. Esto muestra la importancia de realizar pruebas no solo con múltiples juegos sino pruebas en donde se vea como la transferencia de conocimiento afecta a los agentes. El agente FT Neat obtiene mejores resultados que los agentes propuestos, pero hay que tener presente que este agente no es general, requiere de una extracción de características que le indique los objetos importantes en el juego. Finalmente se puede observar como los resultados de estos agentes que procuran ser generales, se encuentran bastante alejados de la capacidad humana.

En la tabla IV se muestran los resultados máximos obtenidos por nuestros agentes y el agente aleatorio. En dicha tabla se puede observar como el agente entrenado con *Breakout* puede llegar a obtener un resultado cercano al de FT Neat, de esta misma manera el entrenado con *Pong* puede llegar a obtener un resultado comparable con el de FT Neat en este juego.

En la figura 6 se puede ver el comportamiento del promedio de aprendizaje, cuando se entrena la red con *Breakout*. En la

Tabla IV. MÁXIMOS

| | Br Max | Po Max |
|---|---|---|
| **Aleatorio** | 3 | -15 |
| **Neuro-Ev Br,Po** | 16 | -17 |
| **Neuro-Ev Po,Br** | 9.33 | -11 |

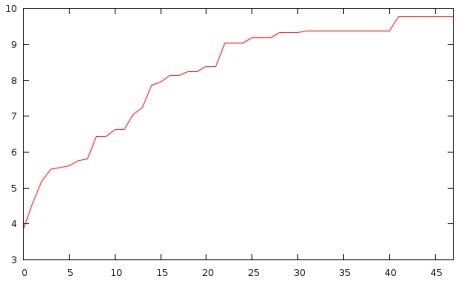


Fig. 6. Aprendizaje promedio con *Breakout*. En el eje X se tiene la generación, en el eje Y el puntaje obtenido.

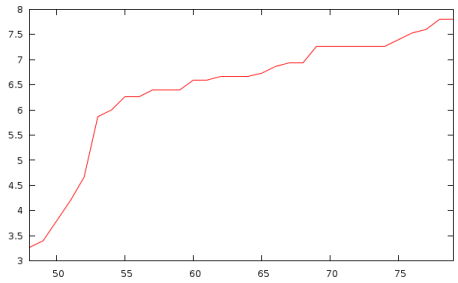


Fig. 7. Aprendizaje promedio con *Breakout* luego de ser entrenado con *Pong*. En el eje X se tiene la generación, en el eje Y el puntaje obtenido.

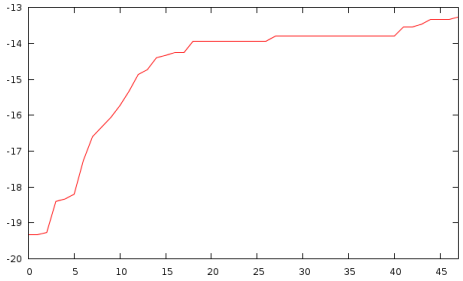


Fig. 8. Aprendizaje promedio con *Pong*. En el eje X se tiene la generación, en el eje Y el puntaje obtenido.

figura 7 se puede ver el mismo caso, cuando se entrena la red con *Breakout*, luego de aprender a jugar *Pong*. En estas gráficas se puede ver como el aprendizaje cruzado no ayuda a la red, al contrario genera un periodo de poco aprendizaje en las primeras etapas, comparado con el aprendizaje obtenido sin tener conocimiento previo. Esto posiblemente se debe a que el conocimiento previo que se traía limita el aprendizaje.

En la figura 8 se muestra como se comporta el promedio de aprendizaje, cuando se entrena con *Pong* y en la figura 9 se muestra este promedio cuando *Pong* se entrena después de *Breakout*. Como en el caso anterior, se evidencia como la red neuronal en vez de explotar el conocimiento adquirido anteriormente, muestra un aprendizaje más lento,

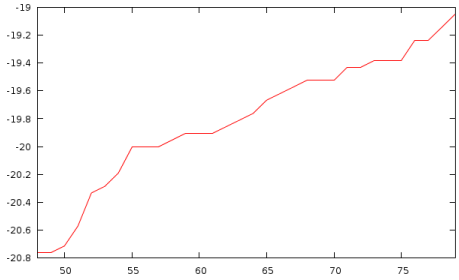


Fig. 9. Aprendizaje promedio con *Pong* luego de ser entrenado con *Breakout*. En el eje X se tiene la generación, en el eje Y el puntaje obtenido.

aunque aparentemente estable.

## VI. CONCLUSIONES

Los agentes entrenados obtuvieron puntajes que superan los reportados por Hausnecht para agentes que reciben los píxeles como entradas, lo que permite a estos considerarse como generales. Esto muestra además, como una red con arquitectura sencilla y usando operadores genéticos simples puede lograr buenos resultados luego de la evolución. El comportamiento de estos agentes muestra detalles sobre su capacidad de "entender" los juegos, por ejemplo al aprender en forma limitada, a actuar según los píxeles de la bola. Esto se ve al probar los agentes entrenados, en los cuales se visualiza, en algunos casos, el intento de estos agentes por seguir el movimiento de la bola.

La transferencia de conocimiento no se visualiza en los resultados obtenidos, en donde los agentes entrenados en un juego obtienen líneas de aprendizaje, en el juego contrario, similares a las obtenidas sin dicho aprendizaje previo. Esto permite resaltar la importancia de realizar este tipo de pruebas de aprendizaje cruzado, ya que el buen funcionamiento en un juego no implica el buen funcionamiento en otros juegos y el entrenamiento por separado aleja a los agentes de la generalidad.

En cuanto a la estructura de pruebas, utilizar una plataforma como ALE, que permite realizar pruebas con múltiples juegos resulta útil para el desarrollo de la experimentación. La arquitectura de dicha plataforma facilita la conexión de diferentes tipos de agentes con un simulador y los puntajes de dichos agentes, para lograr esto, se utilizan tuberías nombradas que permiten conectar agentes escritos en cualquier lenguaje; pero el uso de dichas tuberías presenta dificultades cuando se intentan ejecutar varias pruebas en paralelo.

Finalmente es importante resaltar que las pruebas con los juegos producen recompensas muy variables entre una ejecución y otra, para corregir esto sería útil realizar una aproximación estadísticamente correcta a la función objetivo de cada red, pero para lograrlo el tiempo de evolución crecería enormemente.

## VII. TRABAJO FUTURO

Para obtener agentes con mejores desempeños en los juegos se propone realizar las pruebas con un periodo de evolución más prolongado, el cual permita observar la capacidad máxima que pueden obtener estos agentes. Por otro lado el uso de

técnicas de evolución multiobjetivo, como por ejemplo la creación de nichos, podría permitir mantener poblaciones con eficiencia en múltiples juegos, evitando el periodo de estabilidad en bajos puntajes visto en las gráficas.

## REFERENCIAS


[1] G. M. Levitt, *The Turk, Chess Automation*. McFarland & Company, Incorporated Publishers, 2000.

[2] M. Campbell, A. Hoane, and F.-h. Hsu, "Deep Blue," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 57–83, Jan. 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0004370201001291

[3] B. Pell, "METAGAME: A new challenge for games and learning," *Heuristic Programming in Artificial Intelligence*, vol. 3, pp. 237–251, 1992.

[4] M. Genesereth, N. Love, and B. Pell, "General game playing: Overview of the AAAI competition," *AI magazine*, vol. 26, no. 2, p. 62, 2005.

[5] M. Love, Nathaniel and Hinrichs, Timothy and Haley, David and Schkufza, Eric and Genesereth, "General game playing: Game description language specification," 2008.

[6] M. Bellemare, M.G. and Naddaf, Y. and Veness, J. and Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013. [Online]. Available: http://arxiv.org/abs/1207.4708

[7] K. Nadh and C. R. Huyck, "A Pong playing agent modelled with massively overlapping cell assemblies," *Neurocomputing*, vol. 73, no. 16-18, pp. 2928–2934, Oct. 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0925231210003413

[8] J. E. Mohan, Shiwali and Laird, "Learning to play Mario," *Center for Cognitive Architecture, University of Michigan, Tech. Rep. CCA-TR-2009-03*, 2009.

[9] M. Hausknecht, R. Miikkulainen, and P. Stone, "A Neuro-evolution Approach to General Atari Game Playing," pp. 1–15, 2013.

[10] L. Torrey, J. Shavlik, T. Walker, and R. Maclin, *Skill acquisition via transfer learning and advice taking*. Springer, 2006. [Online]. Available: http://link.springer.com/chapter/10.1007/11871842_41

[11] M. Ponsen, "Improving adaptive game AI with evolutionary learning," no. Manslow 2002, 2004.

[12] D. Robles and S. Lucas, "A simple tree search method for playing Ms. Pac-Man," *Intelligence and Games, 2009. CIG 2009*, pp. 249–255, 2009.

[13] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Magazine*, pp. 106–108, 2012. [Online]. Available: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2419

[14] R. Levinson, "General Game-Playing and Reinforcement Learning," *Computational Intelligence*, 1996.

[15] M. Thielscher, "General game playing in AI research and education," in *KI 2011: Advances in Artificial Intelligence*. Springer, 2011, pp. 26—-37. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-24455-1_3

[16] H. Finnsson, "Cadia-player: A general game playing agent," M.S. thesis, Reykjavik University, 2007.

[17] J. Méhat and T. Cazenave, "Ary, a general game playing program," in *Board Games Studies Colloquium*, 2010.

[18] H. Finnsson, Y. Björnsson, and Y. Bj, "Simulation-based approach to general game playing," in *The Twenty-Third AAAI Conference on Artificial Intelligence*, 2008, pp. 259–264.

[19] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Englewood Cliffs, 1995, vol. 74.

[20] S. A. Bradford W. Mott and T. S. Team, "Stella." [Online]. Available: http://stella.sourceforge.net/

[21] M. Hausknecht, P. Khandelwal, R. Miikkulainen, and P. Stone, "HyperNEAT-GGP: A HyperNEAT-based Atari General Game Player," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, 2012, pp. 217—-224.

[22] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent ...*, vol. 39, pp. 43–62, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743997000610

[23] D. E. and others Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley Reading Menlo Park, 1989.

[24] JVGS, "Atari 2600 High Scores." [Online]. Available: http://www.jvgs.net/2600/top50.htm