# *Pong training*

Víctor de la Torre Rosa
Universidad Rey Juan Carlos
Madrid, España
v.delatorre.2019@alumnos.urjc.es

Mario Esteban Fernández
Universidad Rey Juan Carlos
Madrid, España
m.estebanf.2018@alumnos.urjc.es

Juan Fernández Eva
Universidad Rey Juan Carlos
Madrid, España
j.fernandezev.2018@alumnos.urjc.es

## I. Introducción

El pong es un juego de consolas basado en el tenis de mesa, en el que el jugador controla en el juego una paleta moviéndola verticalmente en la parte izquierda de la pantalla, y puede competir tanto contra un oponente controlado por computadora, como con otro jugador humano que controla una segunda paleta en la parte opuesta. Los jugadores pueden usar las paletas para pegarle a la pelota hacia un lado u otro.

En este caso, se tomarán un total de doscientas muestras, ya que a partir de ese valor la red neuronal entrenará de la misma manera y serán más que suficientes.

La pelota inicialmente partirá de una velocidad aleatoria, ya que debe funcionar en cualquier circunstancia e independientemente de la dirección con la que parta al principio.

Para la recogida de datos, primero se realizará un control sobre la velocidad en ambos ejes y la posición en el eje y de la pelota, dependiendo también si la bola está por encima o por debajo de cierto punto, el cual se irá actualizando de acuerdo a cómo se mueva la bola, tomando como referencia siempre el eje y, la paleta se desplazará verticalmente siguiendo el movimiento de la misma y se recogerán estos datos que serán importantes para su posterior uso en el entrenamiento de la red neuronal para indicar a la paleta si se debe desplazar hacia arriba o abajo.

También hay que tener en cuenta la posición de la paleta constantemente y con los datos anteriores, cuando se alcance los datos máximos a los que se pretende llegar y actualizando constantemente la posición de la pelota y la paleta, ya será más que suficiente para poder empezar a entrenar la red neuronal con un algoritmo el cual deberá de ser capaz de clasificar las entradas que serán los datos objetivo formados por vectores de la forma más óptima, rápida y eficiente posible.

La red neuronal entrenará ella sola, simulando que el otro jugador sea la pared del escenario. Siendo el objetivo que la pelota nunca pase de la paleta de la red neuronal.

## II. Algoritmos de aprendizaje

En segundo lugar, se utilizará la función de alto nivel de Matlab ‘patternnet’, la cual devuelve una red neuronal de reconocimiento de patrones con un tamaño establecido de un número determinado de neuronas de capa oculta.

También se podrá especificar una función de entrenamiento en base a las soluciones más apropiadas que se buscan para resolver este problema.

Se probaron diferentes algoritmos que fueran capaces de jugar a este juego obteniendo el menor error posible en el menor tiempo y con menos iteraciones necesitadas para entrenar la red neuronal.

Uno de ellos fue ‘trainlm’, la cual es una función de entrenamiento que actualiza los valores de peso y sesgo de acuerdo con la optimización Levenberg-Marquardt. Este es generalmente el algoritmo de retro propagación más rápido de la toolbox de Matlab, por lo que será el que necesite menos tiempo para entrenar la red y conseguirá reducir el error hasta uno considerablemente aceptable y se recomienda como el algoritmo de supervisado óptimo, a pesar de que requiere más memoria que otros.

Por estas razones se usó primero esta función de entrenamiento y no supuso en ningún problema a la hora de entrenar la red.

La siguiente función utilizada fue la de ‘trainbr’, que al igual que la usada anteriormente, realiza el mismo proceso de actualización de los pesos y sesgo, pero con la novedad

de que minimiza una combinación de errores cuadráticos y pesos por lo que conseguirá reducir más el error inicial del que parte del que lo que lo reducía el 'trainlm' pero será un poco más lento necesitando este así más iteraciones y tiempo para entrenar la red. Después determina la combinación correcta para producir una red final entrenada que la generalice correctamente, cuyo proceso se denomina regularización bayesiana.

Otras de las funciones menos óptimas e ineficientes que se usaron fueron las de 'trainrp' y 'trainbfg', actualizando la primera los valores de ponderación y sesgo de acuerdo con el algoritmo de retro propagación resiliente (Rprop) y actualizando el segundo los valores de peso y sesgo de acuerdo con el método cuasi-Newton de BFGS.

Estas funciones de entrenamiento con sus respectivos algoritmos completaban el proceso de entrenamiento de la red en muchas más iteraciones y con más error que las dos anteriores citadas.

## III. Hipótesis

Para empezar, se realizaron diferentes pruebas con los diferentes algoritmos mencionados anteriormente cambiando el número de neuronas y estos fueron los resultados:

Utilizando la primera función mencionada 'trainlm' con su respectivo algoritmo Levenberg-Marquardt con una capa oculta de la red, se consigue el resultado esperado ya que la paleta es capaz de seguir la pelota correctamente, pero se puede minimizar el error, el tiempo que necesita la red para ser entrenada y el número de iteraciones incrementando simplemente la longitud del vector que determina el número de capas ocultas de la red.

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 2 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 0.486 | 0.0488 | 0.05 |

Fig. 1. Resultado del entrenamiento de la red con una neurona (trainlm)

Con cinco capas ocultas, se consigue esta reducción en todos los aspectos mencionados anteriormente, como se puede apreciar en la siguiente imagen, se entrena en una iteración y segundo menos y con un error de 0.0315 siendo este menor al que se tenía de 0.0488.

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 1 | 1000 |
| Elapsed Time | - | 00:00:01 | - |
| Performance | 0.344 | 0.0315 | 0.05 |

Fig. 2. Resultado del entrenamiento de la red con cinco neuronas (trainlm)

Si se utiliza la función 'trainbr' con el proceso de regularización bayesiana, la red consiguió entrenarse correctamente tanto para una capa oculta como para cinco y con los mismos parámetros de salida en cuanto a número total de iteraciones necesarias para entrenar la red y tiempo transcurrido, con la diferencia que con una capa se obtuvo un mayor error que con cinco.

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 3 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 0.25 | 0.0401 | 0.05 |

Fig. 3. Resultado del entrenamiento de la red con una neurona (trainbr)

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 3 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 0.371 | 0.0303 | 0.05 |

Fig. 4. Resultado del entrenamiento de la red con cinco neuronas (trainbr)

Con la función 'trainrp' y su respectivo algoritmo de retro propagación resiliente (Rprop), se apreciaban prácticamente los mismos resultados cambiando el número de capas ocultas de una a cinco igual que con la función 'trainbr', habiendo una reducción significante en el error y necesitando una iteración menos para entrenar la red.

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 42 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 1.9 | 0.106 | 0.05 |

Fig. 5. Resultado del entrenamiento de la red con una neurona (trainrp)

**Training Progress**

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 41 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 2.73 | 0.0496 | 0.05 |

Fig. 6. Resultado del entrenamiento de la red con cinco neuronas (trainrp)

Por último, haciendo uso de la función 'trainbfg' con el método cuasi-Newton de BFGS, con una capa oculta aumenta el número de iteraciones que usando cinco como se probaba con algoritmos anteriores siendo el error final prácticamente el mismo.

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 17 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 2.23 | 0.0481 | 0.05 |

Fig. 7. Resultado del entrenamiento de la red con una neurona (trainbfg)

| Unit | Initial Value | Stopped Value | Target Value |
|---|---|---|---|
| Epoch | 0 | 11 | 1000 |
| Elapsed Time | - | 00:00:02 | - |
| Performance | 0.422 | 0.0471 | 0.05 |

Fig. 8. Resultado del entrenamiento de la red con cinco neuronas (trainbfg)

Con todas las pruebas realizadas con cualquier número de capas ocultas con los diferentes algoritmos mencionados, se consiguió el funcionamiento esperado de la red al seguir la dirección de la pelota correctamente en cualquier circunstancia y llegar a los parámetros deseados ajustando el número de capas.

También se comprobó que, a partir de un elevado número de capas ocultas en cada algoritmo, los parámetros ya no cambiaban demasiado.

## IV. Análisis de los resultados y Experimentación

En cuanto a los resultados de la experimentación obtenidos, se analizarán ahora las matrices de confusion viendo la diagonal principal para confirmar que se clasifican correctamente.

Se analizarán los datos y porcentajes de los resultados obtenidos de la función 'trainlm' y 'trainbr', ya que se han obtenido mejores resultados con estas últimas.

En la matriz de entrenamiento, hay 133 datos del conjunto de datos final, clasificando 69 datos (49.3%) en la clase 0 correctamente y 2 datos (1.4%) en la clase 0 que deberían haber sido clasificados en la clase 1. También ha clasificado correctamente 64 datos (45.7%) en la clase 1 y 5 datos (3.6%) en la clase 1 que deberían haber sido clasificados en la clase 0, siendo el porcentaje de acierto final para los datos de entrenamiento del 95% y un 5% de error de clasificación.

En la matriz de validación, hay 30 datos del conjunto de datos final, clasificando correctamente 14 datos (46.7%) en la clase 0 y 16 datos (53.3%) en la clase 1, siendo el porcentaje de acierto final para este caso del 100% sin ningún error.

En la matriz de test, al igual que la matriz de validación hay 30 datos del conjunto de datos final, también clasifica correctamente 20 datos (66.7%) en la clase 0 y 10 datos (33.3%) en la clase 1, sin ningún error en la clasificación por lo que se tendrá un acierto del 100%.

En la última matriz donde se recogen todos los datos, se comprueba que la suma de todos es de 200 datos como se indicaba al principio. En este caso se clasifican con acierto 103 datos (51.5%) y 90 datos (45%) en las clases 0 y 1 respectivamente, y 2 datos (1%) y 5 (2.5%) con error que deberían haber sido clasificados en la clase 1 y 0 respectivamente, porque se tendría finalmente un acierto final del 96.5 % y un error del 3.5%.

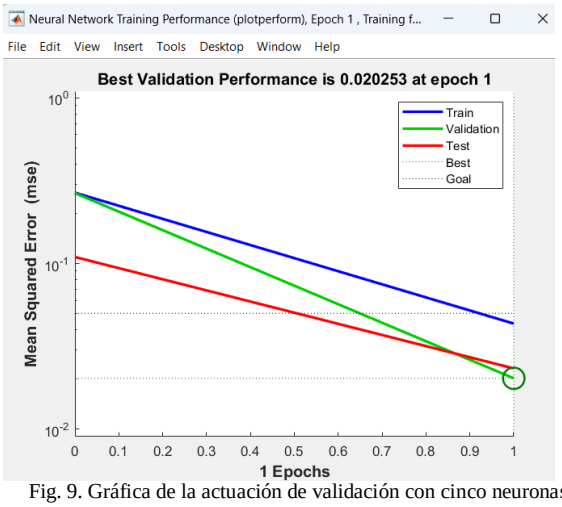


Fig. 9. Gráfica de la actuación de validación con cinco neuronas (trainlm)


Fig. 10. Matrices de confusión con cinco neuronas (trainlm)

Usando la función ‘trainbr’, se obtienen resultados parecidos en las matrices como se comprueba a continuación:

En la matriz de entrenamiento, hay 133 datos del conjunto de datos final, clasificando 68 datos (48.6%) en la clase 0 correctamente y 4 datos (2.9%) en la clase 0 que deberían haber sido clasificados en la clase 1. También ha clasificado correctamente 65 datos (46.4%) en la clase 1 y 3 datos (3.6%) en la clase 1 que deberían haber sido clasificados en la clase 0, siendo el porcentaje de acierto final para los datos de entrenamiento del 95% y un 5% de error de clasificación.

En la matriz de validación, hay 30 datos del conjunto de datos final, clasificando correctamente 17 datos (56.7%) en la clase 0 y 13 datos (43.3%) en la clase 1, siendo el porcentaje de acierto final para este caso del 100% sin ningún error.

En la matriz de los datos de test, al igual que en la de validación se clasifica bien el 100% de los datos (30), 11 en clase 0(36.7%) y 19 en la clase 1(63.3%).


Fig. 12. Matrices de confusión con cinco neuronas (trainbr)

En la última matriz se comprueba que se obtienen prácticamente los mismos resultados que con el algoritmo Levenberg-Marquardt, siendo el porcentaje de acierto final del 96.5% y un error del 3.5 %, clasificando de manera correcta 96 datos (48%) en la clase 0 y 97 en la clase 1(48.5%). En cambio, clasifica erróneamente 3 datos (1.5%) en la clase 1 que deberían ir en la 0 y 4 datos (2%) en la clase 0 que deberían clasificarse en la clase 1.

En este caso se comprueba como se había mencionado inicialmente, que se consigue reducir más el error (0.015) que con la función ‘trainlm’ (0.02) gracias al proceso de minimizar una combinación de errores cuadráticos y pesos, pero será un poco más lento necesitando este así 2 iteraciones más que con el anterior algoritmo utilizado Levenberg-Marquardt.

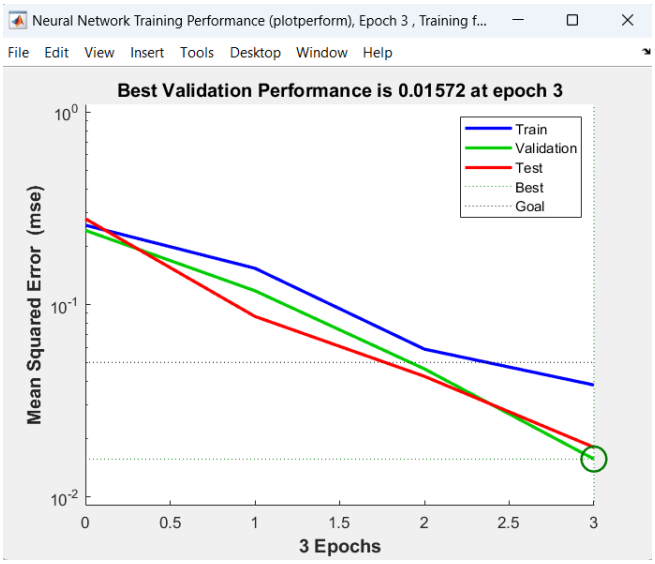


Fig. 11. Gráfica de la actuación de validación con cinco neuronas (trainbr)

## V. Conclusiones

En conclusión, como ya se había comentado al principio, una de las ventajas de utilizar el algoritmo Levenberg-Marquardt es que es de los más rápidos de los que ofrece Matlab y con una tasa de acierto en cuanto a clasificación de los datos alta, pero en cambio como desventaja necesita más memoria que otros. Por otra parte, se podría utilizar también el algoritmo de regularización bayesiana, ya que como ventaja presenta que puede minimizar más el error inicial pero como inconveniente, requiere más tiempo e iteraciones para entrenar la red siendo así este algoritmo más lento que el anterior, aunque presenta también una tasa de acierto elevada.

Después de conseguir los datos necesarios y entrenar la red neuronal, se procedió a ponerla en práctica con una simulación del juego pong en Matlab, con una sola pala controlada por la red neuronal para analizar los resultados. Se realizaron dos versiones. Una con la pala en el lado derecho y otra en el lado izquierdo, para comprobar que funcionaba en ambos casos.

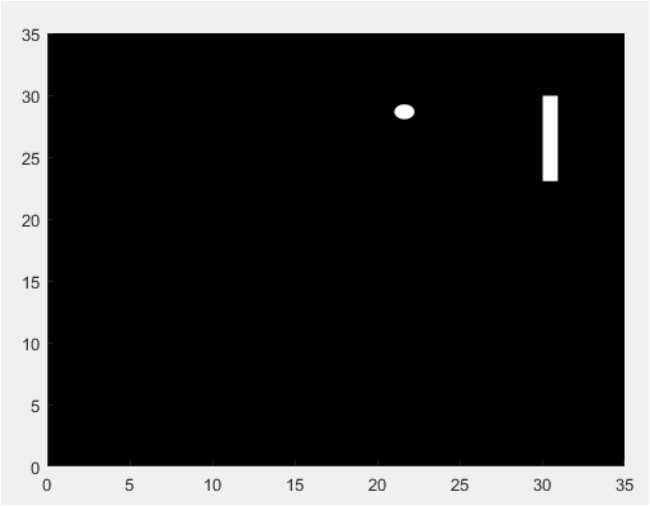


Fig. 13. Simulación del juego pong con una pala

Los resultados son satisfactorios la mayor parte del tiempo, pero la red puede mover la pala de manera incorrecta de tal manera que la bola puede pasar de la pala y chocar con la pared, aunque en raras ocasiones. La frecuencia en la que ocurre esto disminuye cuantos más datos de entrenamiento tenía la red para entrenar.

También se realizó una simulación con dos palas. Una en el lado derecho y otra en el lado izquierdo. Los resultados son similares a las anteriores dos simulaciones, dependiendo del número de datos de entrenamiento, el rendimiento será mejor o peor.

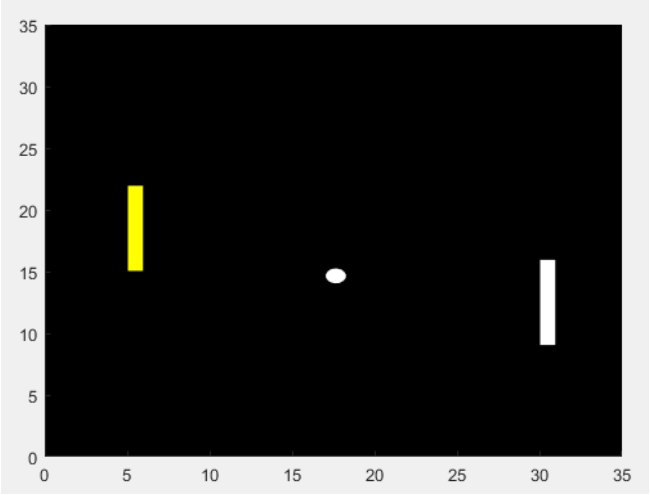


Fig. 14. Simulación del juego pong con ambas palas

No obstante, una red que pueda jugar al pong con una frecuencia de errores en sus decisiones también puede ser utilizada para jugar contra un jugador humano. Ya que si la inteligencia artificial fuera perfecta, el jugador humano nunca ganaría. Utilizando una red neuronal entrenada con menos datos de entrenamiento que comete errores puede dar la oportunidad al jugador de ganar la partida contra la IA.

Por último, con la red neuronal que se ha realizado anteriormente, se ha diseñado un minijuego del pong entre la IA mencionada y un jugador humano, en la que el jugador humano moverá la pala con las teclas de arriba y abajo del teclado, y la IA se moverá automáticamente.

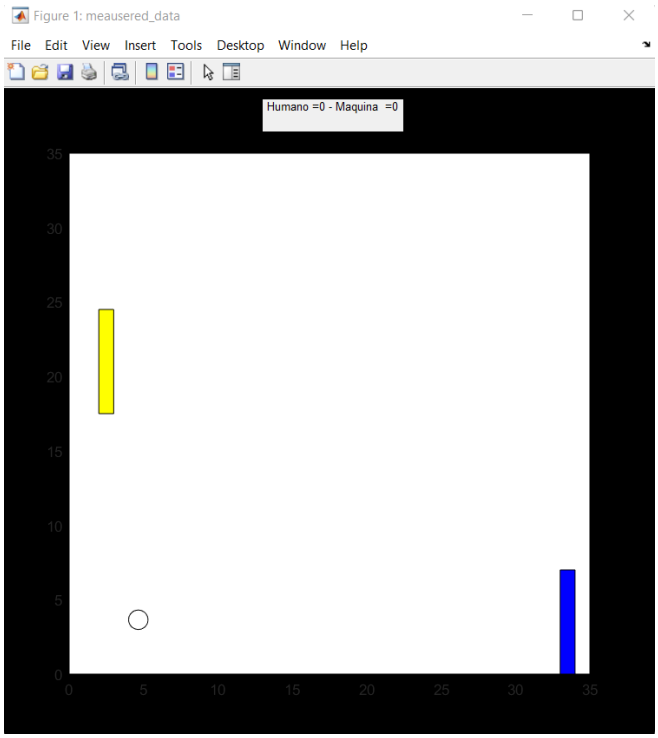


Fig 15. Humano vs IA.

## REFERENCIAS

[1] Pong: https://www.ponggame.org/

[2] Trainlm: https://es.mathworks.com/help/deeplearning/ref/trainlm.html?s_tid=doc_ta

[3] .Trainbr https://es.mathworks.com/help/deeplearning/ref/trainbr.html?searchHighlight=trainbr&s_tid=srchtitle_trainbr_1

[4] Trainrp https://es.mathworks.com/help/deeplearning/ref/trainrp.html?s_tid=doc_ta.

[5] Trainbfg: https://es.mathworks.com/help/deeplearning/ref/trainbfg.html?s_tid=doc_ta

[6] Patternnet https://es.mathworks.com/help/deeplearning/ref/patternnet.html;jsessionid=59245ae17be56c7acf681c7444f0

[7] Figure. https://es.mathworks.com/help/matlab/ref/figure.html

[8] Interruption/uicontrol. https://es.mathworks.com/help/matlab/ref/uicontrol.html?s_tid=doc_ta