Universidad de las Ciencias Informáticas

Facultad 5 “Entornos Virtuales”

Título: Definición del comportamiento de carros autónomos en un videojuego de carreras empleando redes neuronales artificiales

Trabajo de Diploma para optar por el título de
Ingeniero en ciencias Informáticas

Autores:
Andy Trujillo Rivero
Marvyn Amado Márquez Rodríguez

Tutor:
MsC. Yuniesky Coca Bergolla

Ciudad de la Habana
10 de junio de 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor
Andy Trujillo Rivero	Marvyn Amado Márquez Rodríguez	MsC. Yuniesky Coca Bergolla

AGRADECIMIENTOS

Quisiera agradecer a la Revolución por haberme dejado participar en este proyecto maravilloso. Agradecer a mis padres y a mi hermano que siempre me han alentado en toda mi trayectoria; a mi amiga de siempre Yusmilaidi por su incondicionalidad, a mi novia Diannys por resistirme en estos momentos complicados y a todos aquellos que de una forma u otra me han apoyado en el desarrollo de mi tesis.

"La soberanía del hombre está oculta en la dimensión de sus conocimientos."

Francis Bacon.

Marvyn Amado Márquez Rodríguez.

En este momento tan especial no puedo hacer otra cosa que agradecer a todos los que me han apoyado siempre. A mis padres, que se gradúan conmigo; a mi hermano querido; a mi novia, que supo comprenderme y apoyarme; a mis amigos de toda la vida; al team del proyecto; y a todos aquellos que el agotamiento causado por la confección de un documento como este no me permite recordar, pero están siempre presentes.

Andy Trujillo Rivero.

DEDICATORIA

A mi padre el de mayor talento.

A esa persona que no hay meta que no alcance, la perseverante, mi madre.

A mi hermano mi inspiración.

Marvyn Amado Márquez Rodríguez.

A mi familia y mi novia…

Andy Trujillo Rivero.

RESUMEN

En este trabajo se emplea una de las técnicas no determinista de la inteligencia artificial, las redes neuronales artificiales, para definir el comportamiento de carros autónomos en un videojuego de carreras. Se incorpora capacidad de aprendizaje a los carros controlados por la computadora para que puedan evolucionar y adaptarse al perfil de los jugadores, planteándoles así un desafío aceptable. El desempeño alcanzado convierte al juego en un producto novedoso y realista, alargando así su vida útil. Se comprobó la viabilidad de la investigación en el videojuego *"Rápido y Curioso"* del proyecto Juegos Consola de la Universidad de las Ciencias Informáticas.

PALABRAS CLAVE

Inteligencia artificial, IA, técnica no determinista, red neuronal artificial, RNA, perceptrón, perceptrón multicapa, PMC, algoritmo de entrenamiento, propagación del error hacia atrás, agente autónomo, carro autónomo, videojuego de carreras, aprendizaje, entorno virtual.

KEYWORDS

Artificial intelligence, AI, non deterministic technique, artificial neural network, ANN, perceptron, multilayer perceptron, MLP, training algorithm, back-propagation, autonomous agent, autonomous car, racing videogame, machine learning, virtual environment.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Una sencilla búsqueda en internet sobre los negocios más rentables arrojará un resultado que pudiera asombrar a muchos: la industria de los videojuegos genera ingresos similares o superiores a las de la música y el cine. Según datos publicados (1), en el año 2007 se obtuvieron 43 mil millones de dólares en ventas, y se espera (2) que en el 2008 alcancen 55 mil millones.

Con el incremento del poder computacional y la experiencia acumulada, los juegos han evolucionado considerablemente en los últimos años. Han incorporado detallados modelos de miles de polígonos que interactúan entre ellos, respondiendo a complejas leyes físicas. Los agentes controlados por computadora en los ambientes que recrean los videojuegos poseen personalidades adaptativas y son capaces de crear sus propias estrategias de juego.

Diferentes géneros han sido creados para satisfacer los diversos gustos de los jugadores. Se destacan los de deportes, estrategias, combates, juegos de roles, de mesa y los de motor. Dentro de esta última categoría encontramos los juegos de carreras de automóviles, que cuentan con gran aceptación del público. En ellos los jugadores adoptan el rol de pilotos, y tratan de sobrepasar los límites de velocidad para mejorar sus marcas de tiempo en carreras contrarreloj, o para vencer a sus oponentes humanos y artificiales.

Los desarrolladores buscan satisfacer las expectativas de los usuarios ofreciéndole una experiencia única con cada nuevo juego. Para lograr este objetivo es sumamente importante el desafío ofrecido por los elementos inteligentes.

Situación problémica:

En la mayoría de los videojuegos de carreras de automóviles, los carros autónomos son controlados por acciones y reacciones predeterminadas por el desarrollador y no poseen capacidad de aprendizaje o adaptación ante situaciones desconocidas, por lo que su comportamiento es restringido, esquemático y se torna predecible; cuando esto ocurre, los jugadores vencen con facilidad a los oponentes artificiales, perdiendo el interés por un juego que deja de ser entretenido y útil.

Problema científico:

¿Cómo lograr que los carros autónomos aprendan a desempeñar un comportamiento inteligente en videojuegos de carreras?

Objeto de investigación:

Inteligencia artificial en videojuegos.

Campo de acción:

Redes neuronales artificiales en videojuegos de carreras de automóviles.

Objetivo general:

Definir el comportamiento de carros autónomos en videojuegos de carreras empleando redes neuronales artificiales.

Tareas de investigación:

- Identificar las técnicas de inteligencia artificial empleadas para controlar los agentes autónomos[1] en videojuegos.
- Caracterizar el funcionamiento de las redes neuronales artificiales.
- Seleccionar la red neuronal artificial que más se ajuste a los requerimientos del proyecto.
- Modelar vectorialmente el estado del juego.
- Entrenar una red neuronal artificial para controlar los carros autónomos en un videojuego.
- Mostrar el comportamiento de los carros autónomos controlados por la red neuronal artificial en un entorno virtual.

Idea a defender:

Los carros autónomos de un videojuego de carreras que son controlados por redes neuronales artificiales desempeñarán un comportamiento inteligente.

Métodos de investigación empleados:

Para desarrollar esta investigación se utilizaron diferentes métodos científicos, entre los que se destacan el histórico-lógico, analítico-sintético, modelación, observación y experimento.

[1] **Agente autónomo:** Sistema situado en un entorno, que tiene la capacidad de sentirlo y actuar sobre él, a través del tiempo, persiguiendo sus propios objetivos de forma que afecte lo que siente en el futuro.

Métodos teóricos:

- **Histórico – Lógico:** Permite estudiar la evolución y desarrollo de las técnicas de inteligencia artificial (específicamente de las redes neuronales artificiales) para conocer su estado actual.
- **Analítico – Sintético:** Permite concretar y resumir el conocimiento reflejado en los materiales consultados sobre el tema de redes neuronales artificiales para utilizarlo en el desarrollo de esta investigación.
- **Modelación:** Para emplear redes neuronales artificiales, es muy importante realizar una correcta modelación del entorno.

Métodos Empíricos:

- **Observación:** Hay que observar el comportamiento desempeñado por los carros controlados por la red neuronal artificial para identificar errores que permitan mejorar su accionar.
- **Experimento:** Es importante probar con un grupo de variables para lograr mayor precisión en la emulación del pensamiento humano.

El desarrollo de esta investigación reviste especial importancia para el juego *"Rápido y Curioso"* porque le permitirá incorporar a su entorno elementos inteligentes, que controlados por redes neuronales artificiales son capaces de adaptar su comportamiento. El juego se convertiría en uno de los primeros de su tipo en emplear técnicas no deterministas de la inteligencia artificial para este propósito.

La bibliografía existente sobre el empleo de redes neuronales artificiales en videojuegos no es muy amplia, puesto que es una técnica poco usada en este ámbito. Además, los trabajos que se han realizado son propiedad de empresas que no hacen públicas sus soluciones.

Precisamente uno de los objetivos de este trabajo es disponer de una actualizada bibliografía de consulta sobre el tema de redes neuronales artificiales, que permita sentar las bases para difundir esta técnica en la universidad, y específicamente en la facultad de *"Entornos Virtuales"*.

Este documento está conformado por cuatro capítulos, en el primero se caracterizan las técnicas de inteligencia artificial empleadas en videojuegos y se enfatiza en las redes neuronales artificiales. En el segundo se expone el funcionamiento del perceptrón multicapa y el algoritmo de entrenamiento propagación del error hacia atrás. En el capítulo tres se explica la solución propuesta para controlar los carros autónomos en el videojuego *"Rápido y Curioso"*. Por último, en el capítulo cuatro se analizan los resultados obtenidos.

1

CAPÍTULO

FUNDAMENTACIÓN TEÓRICA

Las máquinas están especialmente capacitadas para realizar cálculos y operaciones programadas a velocidades impresionantes. Pero cuando los problemas son más complejos, que requieren capacidad de análisis, comprensión, razonamiento, innovación y adaptabilidad, las personas cuentan con grandes ventajas.

La inteligencia artificial es una rama de la ciencia de la computación dedicada a la creación de hardware y software capaz de imitar el pensamiento humano. Permite a las máquinas hacer cosas que de ser hechas por personas, requerirían inteligencia.

Se ocupa de la representación, adquisición y procesamiento de conocimientos de forma automatizada, de la arquitectura de los programas para estas actividades y de los lenguajes en los que se expresan tales programas. La modelación computacional de los procesos cognoscitivos es también un área de interés de la inteligencia artificial. Además se incluyen la percepción, la comprensión y síntesis del lenguaje natural, la robótica inteligente, la modelación del razonamiento, la programación automática y otras.

El desarrollo de la inteligencia artificial ha seguido dos líneas principales: la simbólica y la sub-simbólica. La primera se caracteriza por desarrollar modelos que describen, formalizan e implementan aspectos sistematizables del conocimiento en forma explícita (sistemas expertos, razonamiento basado en casos, etc). La otra se basa en los enfoques no representacionales de la inteligencia artificial (redes neuronales artificiales, algoritmos genéticos y sistemas difusos). El cálculo sub-simbólico se basa en el uso de representaciones analógicas, el conocimiento se reparte entre diversos componentes del sistema que están enlazados y que pueden funcionar en paralelo.

La inteligencia artificial generalmente se utiliza para resolver problemas difíciles, en los que no se conoce de antemano el mejor método para resolverlo. Tiene aplicaciones en diversos campos, como el militar, la medicina, los negocios, los servicios, la ciencia, la industria del entretenimiento, entre otros.

1.1 Inteligencia artificial en videojuegos

La industria de los videojuegos ha servido como base para probar y desarrollar muchas técnicas de inteligencia artificial; beneficiándose a la vez con realistas modelos de inteligencia en sus productos de entretenimiento, que los hacen más atractivos para los usuarios.

Con la gran cantidad de juegos que existen hoy en el mundo, es posible afirmar que la clave del éxito entre dos juegos de similares características visuales, se encuentra precisamente en la capacidad que

tenga de entretener al usuario, presentándole rivales capaces de retar sus habilidades con un adecuado nivel de inteligencia.

La principal diferencia entre la inteligencia artificial académica y la que es desarrollada para videojuegos radica en la optimización de los resultados. La primera trata de hallar la mejor solución a un problema, sin importar el costo computacional; mientras que la segunda, limitada por la disponibilidad de recursos para realizar sus cálculos en un juego en tiempo real, muchas veces opta por obtener una solución aceptable pero no óptima. Por otra parte, los juegos deben ser interesantes y divertidos, no imposibles de ganar, por lo que hay que mantener un balance adecuado.

En la industria de los videojuegos, el término Non-Player Character (NPC) se refiere a agentes autónomos controlados por la computadora. Para que estos agentes se comporten de manera inteligente, se utilizan varias técnicas de la inteligencia artificial que se pueden clasificar en:

- **Deterministas:** El comportamiento del agente inteligente es especificado por completo; o sea, los programadores tienen que codificar todas las acciones explícitamente, dificultando y retrasando el proceso de desarrollo del juego. Las más usadas son las máquinas de estado finito, los árboles de decisión, y los sistemas de reglas de producción.
- **No deterministas:** Es todo lo contrario, el grado de incertidumbre es mayor, por lo que no se puede predecir el comportamiento que seguirá el agente. Generalmente usan técnicas como redes neuronales, algoritmos evolutivos o redes bayesianas, que facilitan el aprendizaje y la adaptación al entorno de los elementos inteligentes. No hay que codificar explícitamente todas las posibles situaciones en el juego, ya que los elementos inteligentes pueden incluso extrapolar sus comportamientos y desarrollar otros nuevos o emergentes.

1.1.1 Técnicas deterministas

Los desarrolladores de videojuegos han experimentado con la mayoría de las técnicas de inteligencia artificial, pero sin dudas las más sencillas, eficientes, fáciles de implementar, entender y depurar son las deterministas, por lo que han sido las más usadas en este campo.

Además, el tiempo del que disponen las compañías para producir un juego es insuficiente para que los desarrolladores (más enfocados en la calidad de los gráficos) se decidan a experimentar con técnicas de inteligencia artificial no deterministas que son muy difíciles de entender, implementar y probar.

El empleo de trampas (del inglés cheating) es común en los videojuegos. Esta técnica le atribuye a los elementos controlados por la computadora capacidades o información extra que les permiten hacer

mayor resistencia a los jugadores. Por ejemplo, pueden tener información de la ubicación, cantidad y el tipo de las unidades del usuario en un juego de estrategia; pueden incluso tener mayor resistencia al daño, ser más veloces o tener armas más potentes. Sin embargo estas técnicas deben utilizarse con medida, ya que si son muy evidentes o el jugador las descubre puede perder el interés en un juego que "le hace trampas".

También se utilizan frecuentemente las máquinas de estado finito, que definen una serie de estados en los que puede permanecer un elemento, así como las condiciones para que se produzcan transiciones entre ellos. Muchas veces se combinan con lógica difusa, para representar en lenguaje computacional conceptos imprecisos o nociones subjetivas como "*cercano/lejano*".

En el juego "*The Sims*", la inteligencia artificial es implementada con máquinas de estado finito difusas, además utiliza técnicas específicas de A-Life[2] para simular el comportamiento de organismos vivos (en este caso, personas que viven en familia). La base de la inteligencia en este juego es su motor de comportamiento, el cual asocia acciones posibles a cada objeto. Por ejemplo, el archivo para el modelo de un televisor contiene todas las instrucciones para verlo, encenderlo, apagarlo, las condiciones en que un *Sim* querrá o no verlo, cómo debe animarse un *Sim* al verlo, etc.

Un NPC debe ser capaz de reconocer y comprender el espacio que le rodea, empleando para ello técnicas de percepción. Además debe ser capaz de buscar caminos para navegar, por lo que se le presta una especial atención a los algoritmos de búsqueda como Dijkstra o A*. Los juegos de tablero como el ajedrez y el backgammon, han usado árboles de búsqueda heurística con formidables aciertos.

Tradicionalmente en los videojuegos se han utilizado técnicas de planificación o guiones (del inglés script) para definir la conducta de los agentes, al igual que los sistemas basados en reglas y los comportamientos grupales o de manadas. Los sistemas expertos son un tipo de sistema basado en reglas que definen el conocimiento para que los agentes autónomos se comporten de manera similar a un jugador experto.

---

[2] **A-Life (Vida Artificial):** Se refiere a sistemas multi-agentes que intentan aplicar algunas de las propiedades universales de sistemas vivientes a agentes inteligentes en un entorno virtual.

1.1.2 Técnicas no deterministas

En la actualidad el poder computacional se ha incrementado notablemente, manteniéndose fiel a los preceptos de la ley de Moore. Existen potentes tarjetas gráficas que liberan al procesador de la máquina para que pueda encargarse de otras tareas dentro del juego, por ejemplo la física y la inteligencia artificial. Esto, unido a la demanda de los usuarios de juegos más desafiantes, el deseo de los desarrolladores de lanzar un juego que marque la diferencia, así como el mayor interés de los "académicos" de la inteligencia artificial en los videojuegos como área de experimentación relativamente barata y sin riesgos, ha posibilitado que se comiencen a aplicar más frecuentemente técnicas no deterministas.

El reto actual de los desarrolladores es crear juegos novedosos, divertidos, realistas y con mayor vida útil. Un buen paso en ese sentido es lograr que los NPC aprendan, evolucionen, se adapten a nuevas situaciones y exhiban un comportamiento genuino. Estos resultados son posibles de alcanzar con métodos no deterministas, que algunos desarrolladores investigan con gran interés a pesar de su complejidad.

Se han alcanzado excelentes resultados en juegos que usan redes neuronales, algoritmos genéticos o métodos probabilísticos como *Dirt Track Racing*, *Creatures*, *Black & White*, *Battlecruiser 3000AD*, *Fields of Battle*, y *Heavy Gear*. Generalmente se combinan con métodos deterministas más tradicionales y estudiados, conformando una especie de sistemas híbridos.

Unreal Tournament es conocido y elogiado por su inteligencia artificial. El jugador puede elegir un nivel de dificultad (desde "novato" hasta "semidiós"), y más adelante el juego implementa una opción que ajusta automáticamente la dificultad al nivel del jugador humano.

Muchos expertos consideran el aprendizaje como uno de los requisitos fundamentales para definir el concepto de inteligencia, pero ¿qué es aprendizaje?

Aprendizaje implica conocimiento y experiencia, de forma que permita modificar el comportamiento y adaptarlo a las nuevas condiciones.

Tom Mitchell plateó: *"el aprendizaje automatizado implica la búsqueda en un gran espacio de posibles hipótesis para determinar una que sea la que mejor satisfaga los datos observados y algún conocimiento previo del aprendiz".*

Mientras Oliver G. Selfridge considera el aprendizaje como *"la capacidad de los sistemas de integrar y adquirir conocimiento a partir de la experiencia y la observación analítica. Esta capacidad permite que los sistemas estén continuamente en un proceso de mejora incrementando su eficiencia y efectividad"*.

El aprendizaje automatizado investiga los mecanismos de adquisición del conocimiento por las máquinas a través de la experiencia. Teóricamente es mucho más fácil que las computadoras aprendan de hechos, a tener que enseñárselo explícitamente mediante instrucciones. Las redes neuronales artificiales han sido utilizadas con acierto en este campo.

1.2 Redes Neuronales Artificiales

1.2.1 Evolución histórica

En el desarrollo histórico de las redes neuronales artificiales se distinguen tres períodos principales:

- El período de surgimiento, que se inicia a mediados de la década del 40 y termina a mediados de la década del 60.
- El período oscuro se inicia en los años finales de la década del 60 y llega a inicios de los 80.
- El período del renacimiento comienza a mediados de los años 80 y llega hasta el presente.

En 1943 McCulloch y Pitts proponen un modelo de neurona artificial que marcó el inicio del primer período. Otros resultados relevantes de la primera etapa fueron el modelo Perceptron y el modelo Adaline, ambos desarrollados alrededor del año 60 y con características muy similares. El primero fue propuesto por Frank Rosenblatt y el segundo por Bernard Widrow.

En 1969 Minsky y Papert publicaron el libro *Perceptrons*, donde se analizan las capacidades computacionales y limitaciones claves de este modelo. Quedó demostrado que tales redes neuronales artificiales no constituían un modelo computacional de propósito general. Esta fue una de las principales causas que conllevó al segundo período, denominado oscuro; lo que no impidió el desarrollo de la actividad investigativa, que sentó las bases para el tercer período.

En 1982 John Hopfield presentó un modelo de cálculo neuronal que se basa en la interacción de las neuronas (red de Hopfield). En su publicación *"Neural Networks and Physical Systems with Emergent collective computational Abilities"* argumentó que hay capacidades computacionales emergentes a nivel de la red que no existen a nivel de neurona. En la mitad de los años 80 David Rumelhart desarrolló el algoritmo de aprendizaje propagación del error hacia atrás para redes neuronales artificiales multicapas. Estos descubrimientos marcaron el inicio del tercer período.

1.2.2 Generalidades

Las computadoras fueron creadas para realizar tareas de alto nivel como el razonamiento o el cálculo, que pueden ser resueltas mediante el trabajo con símbolos; en este tipo de tareas nuestro cerebro no es excesivamente diestro y actúa con clara desventaja frente a la electrónica. Sin embargo para tareas de procesamiento de bajo nivel como las de reconocimiento de patrones, percepción o control, las computadoras se desenvuelven torpemente, pues en origen no fueron ideadas para ello. Para esta clase de problemas, esenciales para la supervivencia de un organismo vivo, la naturaleza encontró una solución: el procesamiento auto-organizado que emerge de la interacción de numerosos procesadores elementales.

En nuestro cerebro cohabitan unos cien mil millones de neuronas operando en paralelo. Aunque cada neurona individual es capaz de realizar procesamientos muy pequeños, al estar conectada con otras miles, puede trabajar en paralelo y alcanzar una actividad global de procesamiento enorme. Por otra parte las neuronas se auto-organizan, aprenden del entorno y se adaptan a él. De éstas características emergen ricas propiedades como nuestra capacidad de percepción y pensamiento.

Con el objetivo de crear un sistema capaz de ejecutar las tareas que más eficazmente resuelve el cerebro, se crearon las redes neuronales artificiales, que intentan emular el cerebro humano.

1.2.3 Composición de las neuronas biológicas

Se mostrará una breve introducción sobre las redes neuronales biológicas para establecer analogías entre ambos sistemas.

Desde el punto de vista funcional las neuronas constituyen procesadores de información sencillos. Como todo sistema de este tipo poseen un canal de entrada de información, un órgano de cómputo, y un canal de salida, llamados dendritas, soma y axón respectivamente. Existen tres tipos de neuronas: las que reciben información del exterior a través de las dendritas, las que reciben la información procesada desde otra neurona y las que dan la respuesta directamente al músculo

El cerebro está organizado en capas horizontales en las que radican millones de neuronas. Existen además áreas especializadas en ciertas tareas como la visual o la auditiva que están en constante intercambio de información.

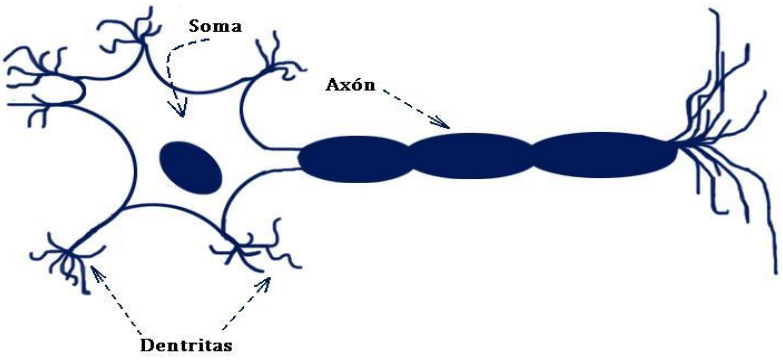


Figura 1: Modelo de neurona biológica

La unión entre dos neuronas se denomina sinapsis. Se denomina neurona pre-sináptica a la que envía señales y post-sináptica a la que las recibe.

1.2.4 Modelo general de una neurona artificial

Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que a partir de un vector de entrada procedente del exterior o de otras neuronas, da una única respuesta o salida.

Los elementos que componen una neurona artificial son:

- Conjunto de entradas X_j
- Pesos sinápticos W_{ij} de la neurona i, que representan la intensidad de interacción entre cada neurona presináptica j y la neurona postsináptica i.
- Regla de propagación $\delta\ (W_{ij},\ X_j)$ que proporciona el valor postsináptico $h_i = \delta\ (W_{ij},\ X_j)$ de la neurona i en función de sus pesos y entradas.
- Función de activación $fi\ (a_i,\ h_i)$, que proporciona el estado de activación actual $A_i = f_i\ (a_i,\ h_i)$ de la neurona i en función de su estado anterior a_i y de su potencial postsináptico h_i.
- Función de salida $f_i\ (A_i)$, que proporciona la salida actual $y_i = f_i\ (A_i)$ de la neurona i en función de su estado de activación.

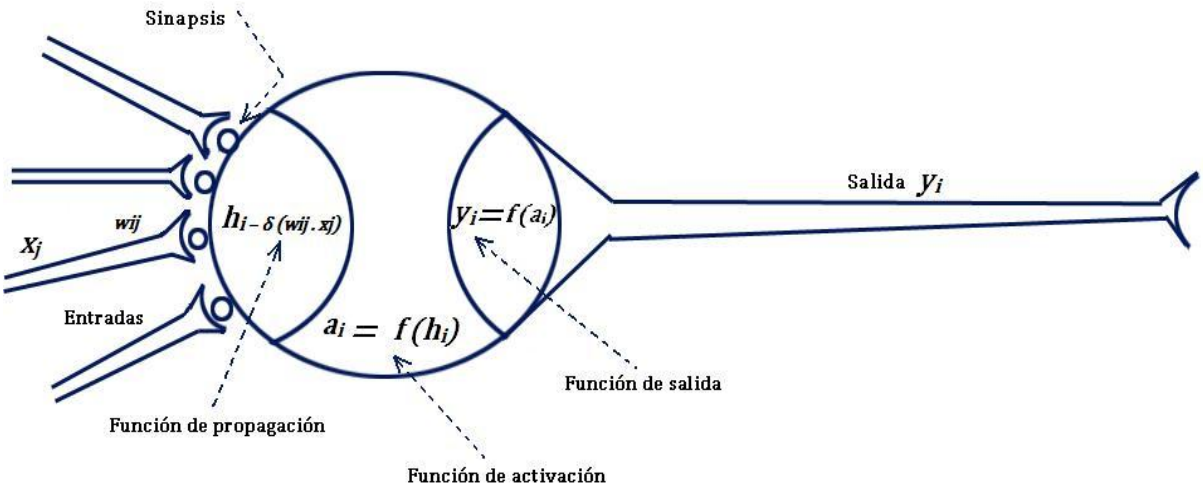


Figura 2: Modelo general de una neurona artificial

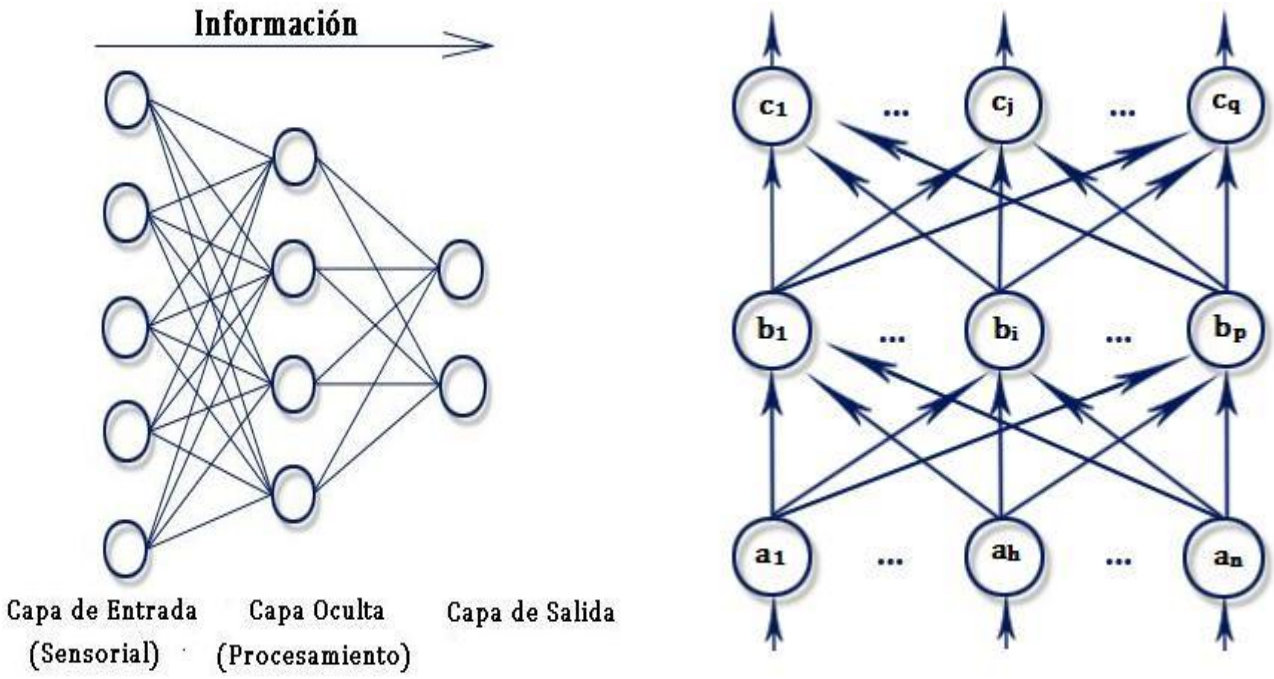
1.2.5 Arquitectura de las redes neuronales artificiales

Se denomina topología o arquitectura, a la estructura de conexión de una red neuronal. Las neuronas se agrupan en unidades estructurales denominadas capas.

En una red neuronal artificial pueden aparecer tres tipos de capas: de entrada, oculta o de salida. Una capa de entrada (sensorial) está compuesta por neuronas que reciben datos o señales procedentes del entorno. Una capa de salida es aquella cuyas neuronas proporcionan la respuesta de la red. Una capa oculta es aquella que no tiene una conexión directa con el entorno, es decir, que no se conecta directamente ni a órganos sensores ni a efectores, este tipo de capa proporciona a la red neuronal una mayor riqueza computacional.

Existe dos tipos de conexiones: la intra-capa y la inter-capa. Una conexión intra-capa (también denominada lateral) tiene lugar entre neuronas pertenecientes a una misma capa, mientras que una conexión inter-capa se produce entre las neuronas de diferentes capas. Existen además conexiones realimentadas que tienen un sentido contrario al de entrada-salida. En algunos casos puede existir la realimentación incluso de una neurona consigo misma.

Red monocapa y realimentada



Red unidireccional de tres capas **Red multicapa y unidireccional**

Figura 3: Ejemplos de arquitecturas neuronales

Pueden establecerse diferentes tipos de arquitecturas neuronales (algunas de las cuales se muestran en la Figura 3). En relación a la estructura en capas, se habla de redes mono-capa y multi-capa.

Atendiendo al flujo de información podemos clasificar las redes neuronales artificiales en unidireccionales (del inglés feedforward) y redes recurrentes (del inglés feedback). En las redes unidireccionales, la información circula en un solo sentido, desde las neuronas de entrada hacia las neuronas de salida. En las redes recurrentes o realimentadas la información entre las capas puede fluir en cualquier sentido.

También se habla de redes auto-asociativas y hetero-asociativas. Si se entrena una red para que asocie un patrón *A* consigo mismo, se dice que es auto-asociativa. Si se entrena una red neuronal con un patrón *A* para que dé como respuesta un patrón *B* se dice que es hetero-asociativa.

1.2.6 Aprendizaje en redes neuronales artificiales

Se distinguen dos modos de operación en los sistemas neuronales: el modo recuerdo o ejecución, y el modo aprendizaje o entrenamiento. En el modo de aprendizaje y entrenamiento la red neuronal es capaz de asimilar el conocimiento a partir de un conjunto de patrones o ejemplos.

Para construir un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura de red, se establecen pesos sinápticos iniciale aleatorios. Para que la red resulte operativa es necesario entrenarla, lo que constituye el modo aprendizaje.

Existe varios tipos de aprendizaje: el supervisado, el no supervisado, el híbrido y el reforzado.

1.2.6.1 Aprendizaje supervisado

En el aprendizaje supervisado se le presenta a la red un conjunto de patrones, junto con la salida deseada u objetivo, e iterativamente ajusta sus pesos hasta que la salida tiende a ser la deseada, utilizando para ello información detallada del error que comete en cada paso. De este modo, la red es capaz de estimar relaciones entrada/salida sin necesidad de proveerle una cierta forma funcional de partida.

Se consideran tres formas de poder llevar a cabo este aprendizaje:

- **Aprendizaje por corrección de error**: Consiste en ajustar los pesos en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error.
- **Aprendizaje por refuerzo**: Se basa en la idea de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada. La función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito= 1 o fracaso= -1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.
- **Aprendizaje estocástico**: Este tipo de aprendizaje consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad.

1.2.6.2 Aprendizaje no supervisado

En este tipo de aprendizaje se presentan a la red multitud de patrones sin indicarle la respuesta deseada. La red, por medio de la regla de aprendizaje estima *p(x)* (función densidad de probabilidad), a partir de lo cual pueden reconocerse regularidades en el conjunto de entradas, extraer rasgos o agrupar patrones según su similitud (clustering).

Por lo general en este aprendizaje se consideran dos tipos:

- **Aprendizaje Hebbiano:** Consiste básicamente en el ajuste de los pesos de las conexiones de acuerdo con la correlación, así si las dos unidades son activas (positivas), se produce un reforzamiento de la conexión. Por el contrario cuando una es activa y la otra pasiva (negativa), se produce un debilitamiento de la conexión.
- **Aprendizaje competitivo y cooperativo:** Las neuronas compiten y cooperan unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada, solo una de las neuronas de salida se active (alcance su valor de respuesta máximo). Por tanto las neuronas compiten por activarse, quedando finalmente una (o una por grupo) como neurona vencedora.

1.2.6.3 Aprendizaje híbrido

En este caso, coexisten en la red los dos tipos básicos de aprendizaje, el supervisado y el no supervisado, los cuales tienen lugar normalmente en distintas capas de neuronas.

1.2.6.4 Aprendizaje reforzado

Se sitúa a medio camino entre el supervisado y el no supervisado. Se emplea información sobre el error cometido, pero existe una única señal de error, que representa un índice global del rendimiento de la red (solamente le indicamos lo bien o lo mal que está actuando, pero sin proporcionar más detalles). No se suministra explícitamente la salida deseada. En ocasiones se denomina aprendizaje por premio-castigo.

Muchos de los algoritmos de aprendizaje se basan en métodos numéricos iterativos que tratan de minimizar una función coste, lo que puede dar lugar en ocasiones a problemas en la convergencia del algoritmo, pero esto depende de sus características y el entorno en que se desempeña. En un sentido riguroso, la convergencia es una manera de comprobar si una determinada arquitectura, junto a su regla de aprendizaje, es capaz de resolver un problema, pues el grado de error que se mide durante el proceso de aprendizaje describe la precisión del ajuste del mapping[3].

En el proceso de entrenamiento es importante distinguir entre el nivel de error alcanzado al final de la fase de aprendizaje y el error que la red entrenada comete ante patrones no utilizados en el

[3] **Ajuste del Mapping:** grado de convergencia del error.

aprendizaje, lo cual mide la capacidad de generalización de la red. Interesa más una buena generalización que un error muy pequeño en el entrenamiento, pues ello indicará que la red ha aprendido correctamente.

1.2.7 Clasificación de las redes neuronales artificiales

Las redes neuronales artificiales se clasifican en dependencia del aprendizaje y la arquitectura. Según el tipo de aprendizaje se clasifican en modelos supervisados, no supervisados, híbridos y reforzados. Dentro de cada uno de estos grupos se hace una nueva distinción entre redes realimentadas y redes unidireccionales, como se muestra en la Figura 4.

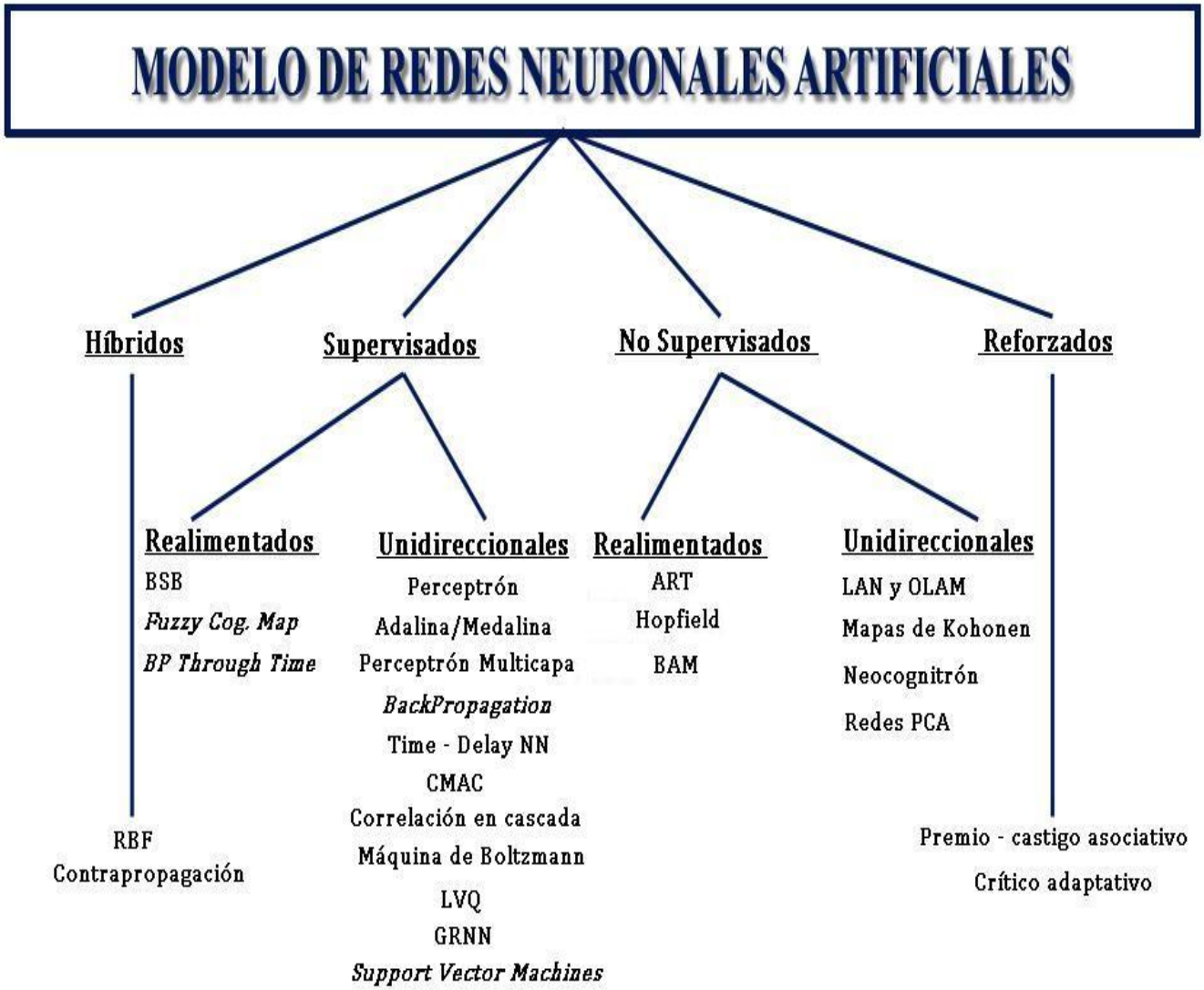


Figura 4: Clasificación de las redes neuronales artificiales según tipo de aprendizaje y arquitectura

1.2.8 Características de las redes neuronales artificiales

- **Aprendizaje Adaptativo:** Es una de las características más atractivas de las redes neuronales. Es la capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.
- **Auto-organización:** Las redes neuronales artificiales usan su capacidad de aprendizaje adaptativo para organizar la información que reciben durante el aprendizaje. Esto les permite responder apropiadamente cuando se les presentan datos o situaciones a los que no habían sido expuestas anteriormente.
- **Tolerancia a Fallos**: Hay muchos elementos de procesamiento independientes. Cada uno es responsable de una parte pequeña del procesamiento, de modo que si alguna unidad falla el efecto en el resultado total del sistema no es apreciable.
- **Paralelismo**: Poseen una gran cantidad de elementos de procesamiento que trabajan simultáneamente.
- **Fácil inserción dentro de la tecnología existente:** Debido a que una red puede ser rápidamente entrenada, comprobada, verificada y trasladada a una implementación de hardware de bajo costo, es fácil insertar una red neuronal artificial en aplicaciones específicas dentro de sistemas existentes (chips, por ejemplo). De esta manera se pueden utilizar para mejorar sistemas de forma incremental, y cada paso puede ser evaluado antes de acometer un desarrollo más amplio.

1.2.9 Aplicaciones de las redes neuronales artificiales

Las redes neuronales artificiales son capaces de aprender de la experiencia, de generalizar el conocimiento, de abstraer características esenciales de un conjunto de entradas que pueden contener información irrelevante.

Estas ventajas han posibilitado su utilización en problemas prácticos concretos, que normalmente no pueden ser resueltos con sistemas tradicionales, como los relacionados con la clasificación, la estimación funcional, optimización y el reconocimiento de patrones en general.

Algunos campos donde han sido utilizadas son: reconocimiento del habla, reconocimiento de caracteres, visión, robótica, control, procesamiento de señales, bibliometría, predicción, economía,

defensa, bioingeniería; así mismo se utilizan para incorporar aprendizaje en los sistemas borrosos y en la confección de sistemas expertos conexionistas.

El reconocimiento de caracteres es uno de los campos donde han alcanzado mayor éxito; se estima que aproximadamente el 50% de los sistemas de OCR (Optical Caracter Recognition) se basan en redes neuronales artificiales; por ejemplo Sharp ha desarrollado un sistema de reconocimiento de caracteres para el alfabeto Kanji (japonés) mediante una red jerárquica. Synaptics, empresa del Silicon Valley, ha desarrollado un chip neuronal para el reconocimiento de direcciones escritas en los sobres de las cartas.

Otra de las áreas donde se han obtenido grandes resultados es en la industria. Fujitsu, Kawasaki y Nippon Stee emplean sistemas de redes neuronales artificiales en el control de procesos industriales, como por ejemplo en plantas de producción de acero. Siemens aplica redes neuronales artificiales y sistemas difusos en la fabricación de celulosa en laminadoras y en galvanizadoras. Citroen emplea redes neuronales artificiales en la determinación de la calidad del material detectado en los asientos de los vehículos, Ford en reducción de contaminantes y Renault para detectar averías en el encendido de los automóviles.

Se utilizó también un sistema de redes neuronales artificiales en un avión de combate F-15 para ayudar al piloto en caso de alcance de fuego de un enemigo.

1.3 Utilización de redes neuronales artificiales en videojuegos

Las redes neuronales artificiales son modelos conexionistas que han sido empleados exitosamente en videojuegos para lograr que los agentes aprendan.

Existen sistemas que han utilizado redes neuronales artificiales para aprender a jugar Backgammon. Gerald Tesauro del centro de investigación de la IBM creó el sistema TD-Gammon, cuya red neuronal es capaz de entrenarse jugando Backgammon en contra de ella misma. Este sistema superó las expectativas del desarrollador ya que jugaba a tal nivel que derrotó sin problemas al campeón mundial, e incluso desarrolló nuevas estrategias que nadie había previsto.

Sin embargo, este tipo de juego no se desarrolla en "tiempo real" y por lo tanto es más sencilla la aplicación de las técnicas de aprendizaje.

Los juegos de carreras de autos también han experimentado con estas técnicas. El Colin McRae Rally 2.0 es posiblemente el que primero las incorporó a juegos de este tipo, y ha servido como referencia a otros desarrolladores. La red neuronal que controla la inteligencia de los autos logra que estos tomen

las curvas en superficies de arena o nieve con el nivel de precisión que lo hacen los jugadores; se mueven por toda la pista inteligentemente, bloquean a los contrarios, toman desvíos para acortar el camino, en fin, son dignos oponentes.

Figura 5: Imágenes del videojuego Colin McRae Rally 2.0

Por otra parte, Julian Togelius ha estado trabajando en un proyecto que vincula redes neuronales artificiales y algoritmos evolutivos para “hacer los juegos de carreras más divertidos” como él mismo dice en su artículo (3). La idea es modelar el estilo de conducción de un jugador con una red neuronal y a partir de ahí, generar con un algoritmo evolutivo una pista de carreras que se adapte a las habilidades personales de dicho jugador.

1.4 Tecnologías y herramientas de desarrollo utilizadas

Las herramientas y metodologías de desarrollo utilizados para desarrollar el demo que acompaña esta investigación están sujetos al marco de trabajo impuesto por el juego donde se insertará.

Se utilizó el entorno integrado de desarrollo Visual Studio 2005, que permite disminuir el tiempo de implementación al incrementar la comunicación y colaboración entre los desarrolladores. Incorpora además generación y completamiento de código, que mejoran considerablemente la calidad y la eficiencia en el trabajo.

C++ fue el lenguaje de desarrollo escogido por soportar el paradigma de programación orientado a objetos, además de ser libre, portable, veloz y muy robusto.

Se empleó la biblioteca gráfica de código abierto G3D, que provee una abstracción altamente optimizada y fácil de usar para numerosas funciones de OpenGL.

La biblioteca de clases de código abierto Flood implementa el perceptrón multicapa en lenguaje C++ e incorpora un conjunto de funcionalidades y algoritmos de entrenamiento que facilitan el empleo de redes neuronales.

2

CAPÍTULO

FUNCIONAMIENTO DEL PERCEPTRÓN MULTICAPA

El perceptrón multicapa (MLP por sus siglas en inglés) es una de las redes neuronales artificiales más estudiadas y utilizadas. Esto se debe fundamentalmente a su eficacia para resolver problemas complejos, principalmente de clasificación, en los que las variables no son linealmente separables.

El Dr Rafael Bello Pérez demuestra la capacidad de clasificación de los MLP al concluir que con al menos dos capas ocultas, son capaces de identificar y separar cualquier región de decisión, por compleja que sea (4).

Una de las principales virtudes de las redes clasificadoras como los MLP es su habilidad para generalizar, que le permite reconocer patrones en el conjunto de entrenamiento. Una vez entrenada la red, si se le presentara una situación nueva, podrá reaccionar satisfactoriamente, aunque no haya sido *"adiestrada"* específicamente para ello.

Otro aspecto a destacar de los MLP es la rapidez en el tiempo de respuesta, condición que lo ratifica como el candidato más apropiado para ser empleado en un videojuego, donde la eficiencia es fundamental.

2.1 Unidad básica: Perceptrón

El perceptrón es el elemento básico de un MLP que procesa un conjunto de datos y los transforma en una señal de salida. Está compuesto por entradas, asociadas a un peso sináptico, un valor umbral y una función de activación.

El valor umbral es determinante en el grado de activación de la neurona. Su objetivo es darle una mayor flexibilidad a la red neuronal artificial en el proceso de aprendizaje.

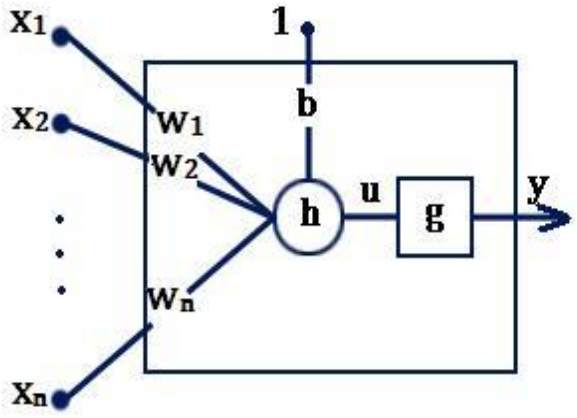


Figura 6: Representación de un perceptrón

- Señales de entrada (x_1, x_2 ..., x_n)
- Pesos sinápticos (w_1, w_2 ..., w_n)
- Valor umbral (b)
- Entrada neta (h)
- Función de activación (g)
- Señal de salida (y)

Para determinar el estado de excitación o señal de salida de una neurona se le aplica una función de activación a la sumatoria del producto de cada entrada por su peso asociado *(Ecuación 1)*, a lo que se adiciona el valor umbral.

$$\boldsymbol{h(x; b, w) = b + \sum_{i=1}^{n} w_i x_i}$$

Ecuación 1: Entrada neta de un perceptrón

2.2 Funciones de activación

Existen diversas funciones de activación como son: sigmoid, escalón, tangente hiperbólica, lineal.

Función Sigmoid:

Es una función no-lineal. Devuelve valores continuos en el intervalo *(0, 1)* con asíntotas horizontales $f(x) = 0$ y $f(x) = 1$. Su derivada es continua, fácil de calcular y de evaluar. Una salida cercana al mínimo valor significa que la neurona no está activada y una salida cercana al mayor valor implica que la neurona está activada.

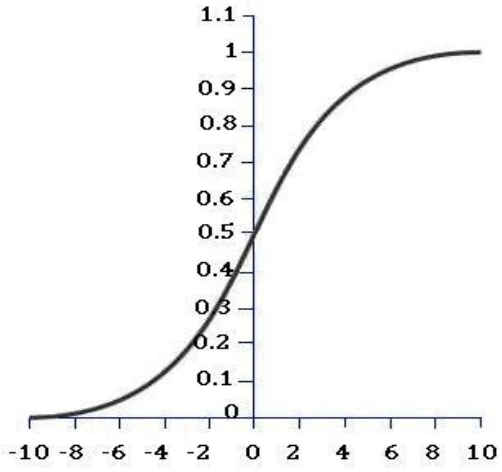


Figura 7: Función Sigmoid

$$f(x) = \frac{1}{(1 + e^{-x/c})}$$

Ecuación 2: Función Sigmoid

Función Escalón:

La función escalón fue utilizada en los comienzos de las redes neuronales, pero era muy complejo evaluar su derivada.

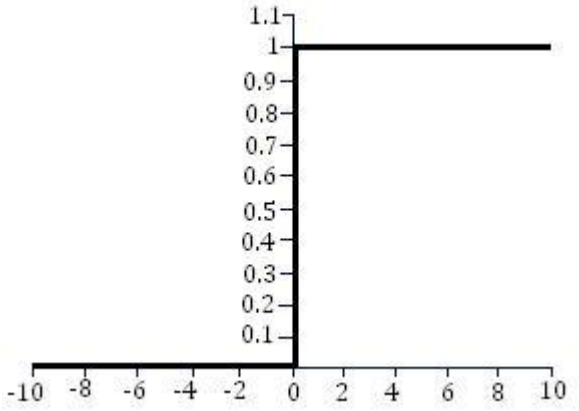


Figura 8: Función Escalón

$$f(x) = \begin{cases} 0 \,; x \leq 0 \\ 1 \,; x > 0 \end{cases}$$

Ecuación 3: Función Escalón

Función Lineal:

La función lineal es muy simple, su salida es igual a la entrada neta de la neurona. Es muy usual que se use en la capa de salida de un MLP.

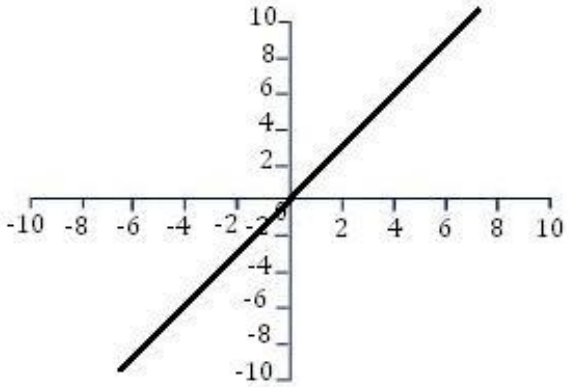


Figura 9: Función lineal

$$f(x) = x$$

Ecuación 4: Función lineal

2.3 Arquitectura del MLP

Como su nombre lo indica, esta red está formada por varias capas de perceptrones interconectados: una de entrada con tantas neuronas como elementos tenga el vector de información que se va a

introducir, una o varias capas ocultas que realizan la mayor parte del procesamiento de los datos y una capa de salida que arroja los resultados finales.

Los pesos sinápticos asociados a las conexiones entre los perceptrones de capas adyacentes representan el conocimiento de la red.

2.4 Proceso de entrenamiento

Cuando comienza el proceso de entrenamiento a la red se le asignan pesos aleatorios que son ajustados para alcanzar un estado que le permita generalizar su comportamiento ante patrones desconocidos. Para lograrlo se comparan los resultados calculados por la red con los valores de salida proporcionados durante la fase de entrenamiento para los distintos conjuntos de entrada; el error cometido se trata de minimizar ajustando los pesos de las conexiones. Esto se realiza de manera iterativa hasta que se satisfaga una condición de parada especificada por el programador.

Durante el entrenamiento la red aprende a reconocer rasgos generales que le permiten responder bien ante patrones no introducidos, lo que es fundamental para su buen desempeño, cuando se ha llegado a este punto se puede decir que la red ha aprendido y está lista para ejecutar las acciones del entorno en que se encuentre.

2.4.1 Algoritmo de aprendizaje back-propagation

Una manera de minimizar el error es mediante el método del gradiente descendente, que a pesar de conducir en ocasiones a mínimos locales, es preferido a otros (como el método de Montecarlo) por razones eficiencia computacional.

El algoritmo de aprendizaje por propagación del error hacia atrás (del inglés back-propagation) ofrece una variante de solución para dicho método, siendo uno de los más reconocidos y utilizados para entrenar redes neuronales.

En el back-propagation los pesos de las neuronas de cada capa son ajustados a partir del error cometido en la capa posterior, multiplicado por la derivada de la función de activación usada; por esta razón es importante que la función de activación sea fácil de derivar y evaluar.

El back-propagation tiene dos pasos fundamentales: Una pasada hacia adelante (del inglés feed-forward) donde se aplica un patrón de entrada a la red con sus pesos actuales (inicialmente pesos aleatorios pequeños). Las salidas de todas las unidades en cada nivel se calculan comenzando a partir de la capa de entrada y trabajando adelante en dirección a la capa de salida. La salida real de la

red se compara con la salida deseada y se calcula el error. Seguidamente se ejecuta una pasada hacia atrás en la cual la derivada del error se propaga a través de la red y todos los pesos son ajustados en proporción a su responsabilidad en el error de la salida.

Pasos del algoritmo back-propagation:

I. Inicializar todos los pesos (W_{ij}) con valores aleatorios pequeños (es usual seleccionar números aleatorios en el rango de -0.1 a 0.1)

II. Presentar una entrada como patrón y especificar la salida deseada.

III. Calcular las entradas netas de todas las neuronas.

IV. Calcular las salidas reales de todas las unidades con los valores de las entradas netas usando la función de activación correspondiente.

V. Encontrar el error para todas las neuronas. D_j es el valor de la salida deseado y Y_j es el valor de salida real para la j-ésima neurona, los errores se calculan según la capa:

   a. Para las neuronas de salida:

$$E_j = \left(Y_j - D_j\right) * (1 - Y_j)$$

   b. Para las neuronas en las capas ocultas:

$$E_j = Y_j * (1 - Y_j) * \sum Z_k * W_{kj}$$

Donde k recorre todas las neuronas de la capa siguiente conectadas a la neurona j y Z_k representa los errores calculados para las salidas de estas neuronas en dependencia de la capa en que se encuentren.

VI. Ajustar los pesos usando la ecuación

$$W_{ij}(n+1) = W_{ij}(n) + t * E_j * Y_i + h * (W_{ij}(n) - W_{ij}(n-1))$$

Donde:

W_{ij} $(n+1)$, W_{ij} (n) y W_{ij} $(n-1)$ indican el peso nuevo, presente y anterior respectivamente, t es un número real que denota la velocidad de aprendizaje y h determina el efecto de los cambios de pesos previos sobre la dirección actual de movimiento en el espacio de los pesos (su valor es constante y está entre 0 y 1).

VII. Presentar otra entrada e ir nuevamente al *paso II*.

Este procedimiento se repite hasta que el error tenga un valor cercano 0. Si el número de ejemplos excede la cantidad de conexiones de la red, puede no ser posible hacer 0 el error.

2.4.2 Técnicas para mejorar el back-propagation

Una vez entrenada la red, puede que esta no generalice bien. Esto puede ser causado por la caída del algoritmo en un mínimo local, o por la utilización de patrones de entrenamiento incorrectos o insuficientes.

Para resolver estos problemas se le pueden cambiar los valores de pesos iniciales a la red y entrenarla nuevamente. Es necesario hacer una buena elección de las variables de entrada a la red, evitando diferencias marcadas entre los valores numéricos. Con el fin de mejorar la convergencia del algoritmo y acelerar la velocidad del entrenamiento se utiliza la técnica de adición de impulso (del inglés momentum).

Técnica de adición de impulso

Esta técnica tiene como objetivo mantener la convergencia de la red en dirección del mínimo global.

Como se muestra en la *Ecuación 5*, *s*e le adiciona un factor impulso a la ecuación de ajuste de los pesos empleada en el back-propagation con el fin de que si el algoritmo cae en un mínimo local logre rebasarlo y continúe hacia el mínimo global.

$$W_{ac} = W_{vj} + \alpha * \Delta W$$

Ecuación 5: Adición de impulso

Donde α es un número aleatorio pequeño entre 0.1 y 0.9 que representa el factor de impulso. De acuerdo al desempeño del algoritmo, el desarrollador debe ajustar este valor.

Si la red está convergiendo hacia el mínimo global y se lo salta, la inercia de la actualización hará que los pesos siguientes sean menores, intentando atraerla nuevamente hacia este mínimo.

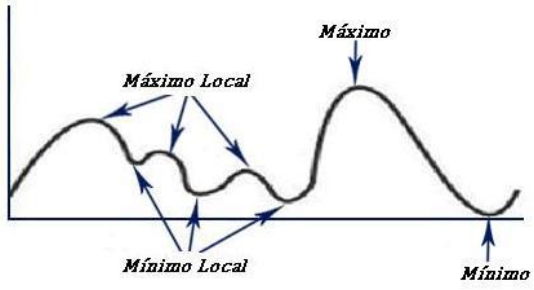


Figura 10: Máximos y mínimos locales

2.4.3 Calculando el error

Existen diversos métodos para calcular el error cometido durante el proceso de entrenamiento, entre los cuales se encuentran: la suma de los errores al cuadrado, el error cuadrado medio y la raíz del error cuadrado medio.

La función del error cuadrado medio calcula el error cometido por la red para un conjunto de datos de entrenamiento. Se utiliza como criterio de parada en el algoritmo back-propagation.

$$\varepsilon = \frac{\sum(n_c - n_d)^2}{m}$$

Ecuación 6: Error cuadrado medio

Nc representa la salida real y Nd la salida deseada para cada conjunto de datos de entrenamiento y m es la cantidad de neuronas de la capa de salida.

Para ajustar los pesos sinápticos de las conexiones, se requiere conocer cuanto influye cada neurona en el error total cometido por la red; por lo que se calcula el error asociado a cada una de ellas en las capas de salida y oculta.

Cálculo del error de las neuronas de la capa de salida:

$$\Omega_i^s = \Delta n_i^s * f(nc_i^s)$$

Donde Ω_i^S es el error cometido por la neurona i-ésima, Δn_i^S es la diferencia entre la salida real y la salida deseada, $f(nc_i^S)$ es la derivada de la función de activación evaluada en el valor de la neurona i-ésima.

Cálculo del error de las neuronas de la capa oculta:

$$\Omega_i^o = \left(\sum \Omega_i^s * w_{ij}\right) * f(nc_i^o)$$

Ω_i^o es el error cometido por cada neurona en la capa oculta. Esta ecuación es una función asociada con el error (Ω_i^S) cometido en la capa de salida a la que se conectan estas neuronas multiplicada por los pesos de cada conexión, por la derivada de la función de activación utilizada, evaluada en el valor de la neurona en la capa oculta.

Primero se calcula el error en la capa de salida y se propaga hacia la capa oculta ajustando los pesos de forma iterativa.

Para calcular este error se necesita la derivada de la función de activación, por lo que generalmente se utiliza la Sigmoid.

Los pesos en la capa de entrada no se actualizan porque sus valores son dados como parte del proceso de entrenamiento.

2.4.4 Ajuste de los pesos

A partir del error se puede calcular la variación que es necesaria hacer en los pesos actuales:

$$\Delta \boldsymbol{w} = \boldsymbol{\rho} * \boldsymbol{\Omega_i} * \boldsymbol{nc_i}$$

Donde Ω_i es el error cometido, n_{ci} es el valor de la neurona i-ésima y (ρ) es la tasa de aprendizaje, un multiplicador que afecta la cantidad de ajuste de cada peso.

La tasa de aprendizaje es un valor aleatorio pequeño entre 0.1 y 0.9. Un valor demasiado alto puede provocar que se rebase el peso óptimo; mientras que uno bajo, puede demorar el proceso de entrenamiento.

Al peso de la iteración anterior (W_{vj}) se le adiciona la variación (ΔW) para obtener el nuevo peso (W_{ac}):

$$\boldsymbol{W_{ac}} = \boldsymbol{W_{vj}} + \boldsymbol{\Delta W}$$

2.4.5 Criterio de parada

En general el algoritmo de entrenamiento se detiene cuando se ha alcanzado un error menor que una cantidad especificada; pero en ocasiones este criterio no es suficiente pues al detenerse el proceso de entrenamiento no se sabe con exactitud qué cantidad o proporción del conjunto de entrenamiento fue asimilada, por lo que se pueden incluir criterios adicionales:

- Detener el entrenamiento cuando se han aprendido todos los ejemplos.
- Detener el entrenamiento cuando se ha aprendido una proporción (por ciento) determinada de los ejemplos.
- Detener el entrenamiento cuando se ha excedido un número máximo de iteraciones[4].

[4] **Iteración:** ciclo en el que se ha recorrido el conjunto de datos de entrenamiento completo.

3

CAPÍTULO

SOLUCIÓN PROPUESTA

3.1 Marco de trabajo

En el proyecto productivo Juegos Consola, perteneciente a la facultad *"Entornos Virtuales"* de la Universidad de las Ciencias Informáticas (UCI) se está desarrollando un juego de carreras de automóviles para niños que lleva por nombre *"Rápido y Curioso"*.

Se trabaja en el entorno de desarrollo Visual Studio Team System 2005 y se programa en lenguaje C++.

El proyecto presenta una arquitectura modular: existe un módulo para las colisiones y la simulación física en general que se apoya en la librería ODE; otro para controlar los eventos del teclado o Joystick que emplean SDL; OpenAL se usa para la reproducción de sonido en un módulo con ese fin; el render se controla con el motor gráfico G3D, que brinda además soporte para el trabajo con las texturas, imágenes, modelos 3D, ventanas, efectos especiales, matemática, redes, así como acceso a funciones de OpenGL.

Los mapas son diseñados y compilados con la herramienta GTK-Radiant, que exporta ficheros de extensión *bsp*. El módulo de edición de pistas permite cargar estos ficheros para ubicar nodos interconectados por aristas, que indican la dirección y sentido de los circuitos de carreras.

El grafo obtenido después de este proceso, se emplea por el módulo que controla la lógica del juego, para saber cuándo un competidor va en sentido correcto, cuántas vueltas ha completado, etc.

La primera versión del juego solo cuenta con un modo de carreras contrarreloj, en la que los competidores luchan por rebajar sus marcas de tiempo en cada carrera. Para la segunda versión se planea incluir otras funcionalidades, destacándose el módulo de red que permitirá la opción multijugador, y el módulo de inteligencia artificial, que controlará los elementos inteligentes que se incorporarán a las carreras y que transitarán por las calles de la ciudad en que se desenvuelve el juego.

Como se discutió en el *Capítulo 1*, las técnicas de inteligencia artificial no deterministas pueden, y de hecho se han empleado con éxito para controlar agentes autónomos en videojuegos.

Varios autores han trabajado sobre esta línea, en busca de comportamientos más reales, naturales, menos predecibles, y que brinden una mayor sensación de inteligencia por parte de los agentes controlados por la computadora.

Se destaca por sus contribuciones Jeff Hannan, quien empleó un una red neuronal artificial para que los carros autónomos aprendieran a conducir en cada una de las pistas del videojuego de rally *"Colin*

McRae 2.0". Este juego fue muy bien aceptado por el público por su capacidad de entretenimiento y el desafío que planteaban sus competidores artificiales. Tuvo además un notable impacto en la comunidad mundial de desarrollo de elementos inteligentes para juegos.

Julian Togelius también ha experimentado con redes neuronales artificiales multicapas para controlar carros autónomos en videojuegos con excelentes resultados.

Se ha comprobado la utilidad de las redes neuronales artificiales para ejecutar habilidades y manejar variables estrechamente relacionadas entre sí, como las que influyen sobre un conductor para controlar un auto. Este tipo de relaciones es muy difícil de modelar matemáticamente. Sin embargo, a partir un conjunto de ejemplos una red neuronal puede sistematizar patrones que le permitan desempeñar una conducta semejante a la de un chofer convencional.

3.2 Descripción de la solución propuesta

La propuesta concreta de esta investigación es emplear redes neuronales artificiales para definir el comportamiento de los carros autónomos en el videojuego *"Rápido y Curioso"*, teniendo en cuenta los resultados y experiencias mencionadas.

Por las virtudes explicadas en el capítulo anterior, se decidió emplear un perceptrón multicapa con flujo de datos hacia adelante y algoritmo de aprendizaje back-propagation. Las conexiones son inter-capas, en las que cada neurona se conecta con la capa adyacente.

Básicamente la red funciona como el "cerebro" de uno de los conductores a derrotar. Se puede definir como un neuro-controlador para el auto, que recibe información del entorno para determinar qué acción debe ejecutar en cada instante de la simulación.

El objetivo es que desempeñen un comportamiento capaz de emular un jugador humano. El competidor y el agente autónomo estarán en igualdad de condiciones en cuanto al acceso a información y acciones para maniobrar se refiere. Se evita el empleo de "trampas", una efectiva fórmula utilizada a menudo por los desarrolladores de inteligencia artificial para videojuegos pero a la vez aborrecida por los jugadores, que pueden quedar muy decepcionados al descubrirlas.

3.2.1 Entorno del videojuego

Para que un auto recorra la pista rápidamente evidenciando un comportamiento inteligente, debe poder reconocer su entorno.

Utilizando el editor de pistas, se ubican nodos circulares en el mapa del juego, procurando que su centro coincida con el de la pista y que su diámetro sea similar al ancho de la misma. Los nodos se asocian mediante aristas de un solo sentido, conformando un grafo dirigido que representa el circuito de carreras.

Esta tarea es engorrosa, porque la versión actual del editor brinda pocas facilidades de diseño, atentando contra la precisión en la ubicación física de los nodos. Por otra parte, cada nodo solo contiene información sobre su posición, radio y la lista de sus vecinos.

Al comenzar la simulación, el agente autónomo conoce cuales son sus nodos actual, siguiente y anterior. Cuando el carro pasa por el nodo siguiente, éste se convierte en el actual, y el primero de su lista de vecinos pasa a ser el siguiente. El nodo anterior es el que era actual hasta ese momento.

A partir de la posición (centro) de los nodos se obtienen diferentes segmentos del circuito que son de interés para el agente autónomo: los segmentos actual, siguiente y anterior:

$$Segmento\ actual = Nodo\_Siguiente - Nodo\_Actual$$

$$Segmento\ siguiente = Nodo\_Siguiente\_al\_Siguiente - Nodo\_Siguiente$$

$$Segmento\ anterior = Nodo\_Actual - Nodo\_Anterior$$

3.2.2 Sistema Sensorial

Cada auto tiene acceso a parámetros que son utilizados intuitivamente por las personas cuando conducen, como su ubicación espacial, a qué velocidad y en qué dirección se mueve, qué obstáculos tiene en el campo de visión, cuan próximo está del borde de la carretera, qué grado de peligrosidad presenta la próxima curva y cual es el trazado del circuito.

3.2.2.1 Distancia al centro de la pista

La distancia al centro de la pista fue la variante escogida para saber cuando el carro se está alejando de su centro. Su utilización en lugar de la distancia cada uno de los bordes se debe en gran medida a la limitada información contenida en los modelos del juego, que no permite diferenciar los objetos sólidos, impidiendo conocer por ejemplo, cuales son vallas, cuales son señales de tránsito o cuál es el piso.

Curiosamente la solución empleada resultó ser más eficiente, ya que evita realizar continuos chequeos de colisión con cada objeto sólido en la escena. En su lugar se calcula la distancia del carro al

segmento de pista más cercano, valiéndose de la ecuación de la distancia de un punto a una recta (*Ecuación 7*).

El segmento más cercano al carro se determina a partir de los tres segmentos de interés para el agente autónomo (actual, siguiente y anterior).

$$D(P,S) = \frac{|\mathrm{Ax} + \mathrm{By} + \mathrm{C}|}{\sqrt{A^2 + B^2}}$$

Ecuación 7: Distancia de un punto a una recta

Donde *P* es la posición del carro y *S* el segmento definido por dos nodos consecutivos, que al estar ubicados aproximadamente en el medio de la pista, hacen que el segmento coincida con su centro.

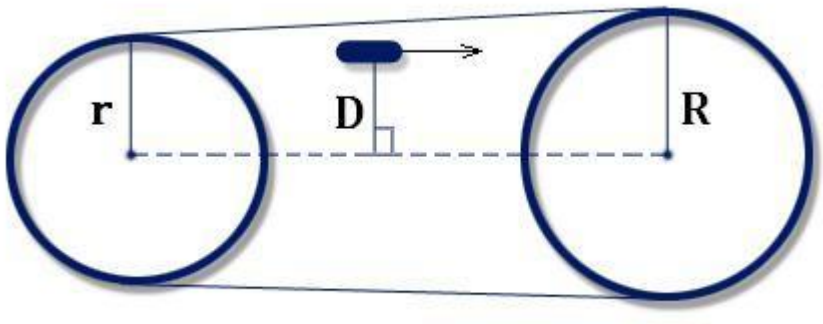


Figura 11: Distancia del carro al centro de la pista

3.2.2.2 Ángulo entre la dirección del carro y la pista

Cuando conducimos, debemos conocer el sentido del movimiento del carro para compararlo con la trayectoria que queremos seguir, y corregirlo en caso necesario.

Mediante la *Ecuación 8* se obtiene el coseno del ángulo formado por el vector dirección del carro (V_1) y un vector (V_2) que parte del centro del carro y que tiene la misma dirección de su segmento más cercano, por lo que es paralelo al mismo.

$$\cos\alpha = \frac{\mathrm{V}_1 * \mathrm{V}_2}{|\mathrm{V}_1| * |\mathrm{V}_2|}$$

Ecuación 8: Coseno del ángulo entre dos vectores

Si el ángulo tiene valores cercanos a cero, significa que el carro se mueve en la dirección correcta por todo el trazado.

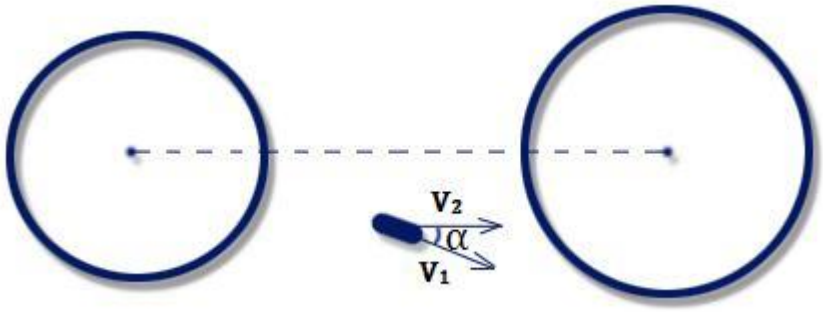


Figura 12: Ángulo entre la dirección del carro y la pista

3.2.2.3 Ángulo de la próxima curva

Utilizando la *Ecuación 8* se puede conocer la curvatura de la pista, que influye en el modo de controlar el carro. La amplitud de la próxima curva coincide con el ángulo formado por los segmentos actual (V_1) y siguiente (V_2).

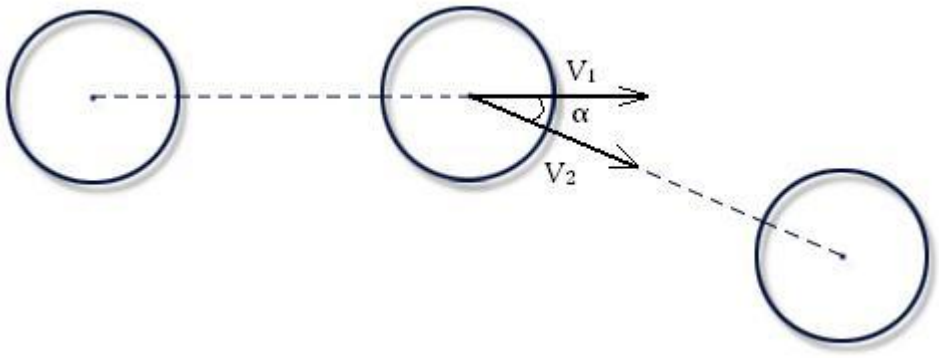


Figura 13: Ángulo de la próxima curva

3.2.2.4 Distancia del carro a la próxima curva

Otro aspecto que se tiene en cuenta a la hora de conducir es la distancia a la que se encuentra el carro de una curva. Las curvas en el juego se identifican con un nodo, o sea: dondequiera que haya un nodo existe una variación en el sentido de la pista, aunque sea mínima. Por lo tanto, si se halla la distancia al próximo nodo se podrá conocer la distancia a la próxima curva:

$$D = |próximo\ nodo - posición\ carro|$$

Ecuación 9: Distancia del carro al próximo nodo

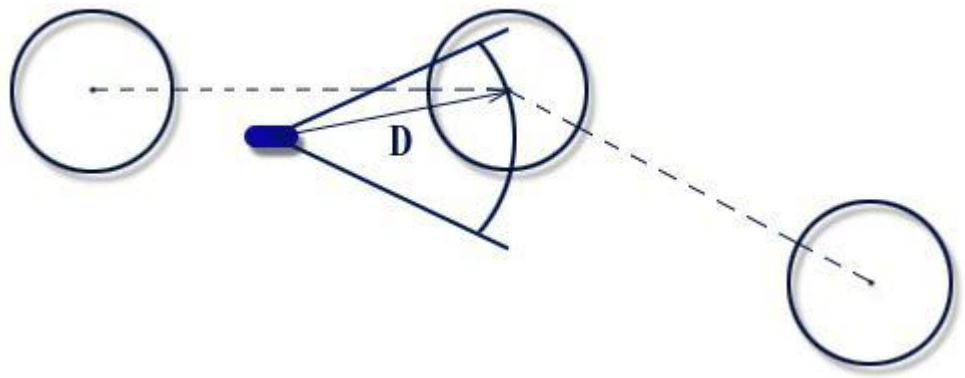


Figura 14: Distancia del carro a la próxima curva

3.2.3 Interfaz para controlar el auto

Para controlar un auto en el juego, se necesitan las posiciones del acelerador y el freno así como el ángulo de giro del timón.

A partir de las posiciones del acelerador y el freno, se calculan las revoluciones por minuto del motor, que generan un torque que actúa sobre los ejes del carro para alcanzar una velocidad deseada.

Abstrayendo este proceso, se puede concluir que el objetivo de la aceleración y el frenado es alcanzar una velocidad deseada (que puede ser baja en las curvas y alta en las rectas).

3.3 Diseño de la red

Comoquiera que la arquitectura de la red depende del problema, y que además no existe un consenso entre los investigadores referente a la cantidad óptima de capas y neuronas por cada una de ellas, se experimentará con varias arquitecturas, comenzando por la más simple: una capa entrada, que recibirá los parámetros de retroalimentación del entorno virtual; una oculta, en la que tendrá lugar la mayor parte del procesamiento de la red; y otra de salida, que decidirá cómo controlar el auto.

Es necesario determinar cuántas neuronas habrá en cada capa y qué representa cada una de ellas. Se trata de minimizar su cantidad para evitar que la red reciba datos incorrectos, ineficaces o

redundantes; reduciendo además el procesamiento y aumentando la eficiencia. Esto se logra a partir de un análisis cuidadoso del contexto del problema en que se utilizará la red.

Las neuronas de la capa de entrada se deducen del sistema sensorial descrito en el epígrafe 3.2.2 Sistema Sensorial, que contiene la información necesaria y suficiente para producir valores correctos en las neuronas de la capa de salida, con los que se controla el auto. Las neuronas de salida representan el ángulo de giro del timón y la velocidad deseada.

Tratando de minimizar el número de variables de la red, se prefirió emplear la velocidad deseada a las posiciones del freno y el acelerador, porque como se explicó anteriormente, sintetiza su función.

El número de neuronas en la capa oculta, está sujeto a la experimentación (al igual que el número de capas ocultas). Aunque algunos autores (5) sugieren una aproximación inicial:

$$\text{Nodos\_O} = \sqrt{\text{Nodos\_E} * \text{Nodos\_S}}$$

Ecuación 10: Cantidad aproximada de nodos de la capa oculta

Nodos_O: Cantidad de nodos en la capa oculta.

Nodos_E: Cantidad de nodos en la capa de entrada.

Nodos_S: Cantidad de nodos en la capa de salida.

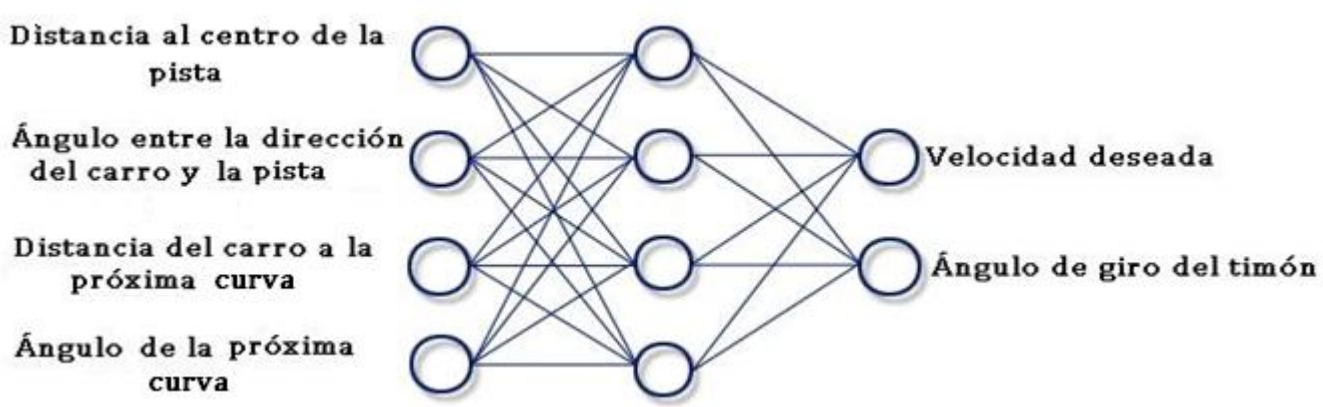


Figura 15: Modelo de red utilizado

3.3.1 Interpretación matemática de las variables empleadas.

Las redes neuronales artificiales trabajan con números reales, por lo que hay que traducir a este tipo de dato los valores del domino de las variables seleccionadas (booleanos, enumerados, valores continuos que pueden representar formatos de imágenes, música, etc), para que puedan ser comprendidas por la red.

3.3.1.1 Entradas

Es recomendable que los valores de entrada a la red estén aproximadamente en el mismo orden de magnitud, para evitar que algunos tengan mayor impacto que otros sobre los pesos de la red. En caso contrario, durante el proceso de entrenamiento se escalan a un mismo rango, comúnmente [0; 1] ó [-1; 1].

Desde luego, una vez que esté funcionando, los datos que recibe como entrada tienen que ser escalados en la misma forma que se hizo durante el proceso de entrenamiento.

Variables utilizadas:

I. Distancia al centro de la pista

La información que brinda esta variable es empleada por el neuro-controlador para evitar que el carro se salga de la pista.

Se calcula la distancia al centro de la pista como se explicó en el epígrafe 3.2.2.1 Distancia al centro de la pista. Su valor puede ser positivo si está a la derecha del segmento más cercano, o negativo si está a la izquierda.

Para conocer en qué lado de la pista se encuentra el carro, se emplea la Ecuación 11 donde V1 y V2 son las posiciones de los nodos que definen el segmento más cercano a V3 (posición del carro).

$$\alpha = (V_1.x - V_0.x) * (V_3.z - V_0.z) - (V_1.z - V_0.z) * (V_3.x - V_0.x)$$

Ecuación 11: Orientación relativa de tres vectores

Si α es menor que cero, V3 está a la izquierda del segmento $\overline{V_1V_2}$, en caso contrario está a la derecha.

Los valores de esta variable se escalan al rango [-1, 1] que recibe como entrada el neuro-controlador. Para lograrlo, se compara la distancia al centro de la pista (DCP) con el menor de los radios de los nodos que conforman el segmento más cercano (r):

- Si es mayor que el 80% del radio, se considera el auto está próximo al borde: $DCP = \pm 1$
- Si es menor que el 30% del radio, se considera el auto está en el centro: $DCP = 0$
- Si no, toma un valor proporcional al radio: $DCP = \pm \frac{DCP - (0.3r)}{(0.8r) - (0.3r)}$

Figura 16: Clasificación de un segmento de pista

II. Ángulo entre la dirección del carro y la pista

Se calcula el ángulo entre la dirección del carro y la pista como se explicó en el epígrafe 3.2.2.2 Ángulo entre la dirección del carro y la pista. Su valor puede ser positivo o negativo, para lo que se utiliza la Ecuación 11, donde $\overline{V_1V_2}$ es el segmento que parte de la posición del carro y es paralelo al segmento más cercano; y V3 es el vector dirección del carro.

En dependencia del ángulo y el signo, el neuro-controlador determinará en qué medida hay que girar y hacia donde, para mantener la dirección del movimiento del carro en el sentido de la pista.

III. Ángulo de la próxima curva

Se calcula el ángulo de la próxima curva como se explicó en el epígrafe 3.2.2.3 Ángulo de la próxima curva. Su valor puede ser positivo o negativo, para lo que se utiliza la Ecuación 11, donde $\overline{V_1V_2}$ es el segmento actual; y V_3 es el vector posición del nodo siguiente al siguiente.

La curva puede ser de riesgo, media o mínima; lo que es interpretado por el neuro-controlador a partir del ángulo, permitiéndole determinar la velocidad óptima deseada para tomar la curva.

IV. Distancia del carro a la próxima curva

Se especifica un rango de visión para el agente autónomo, representado por un valor entero (40 en este caso). Si la distancia del carro a la próxima curva (calculada como se explica en el epígrafe 3.2.2.4 Distancia del carro a la próxima curva) es menor que dicho valor, se considera que la curva definida por los segmentos actual y siguiente se encuentra en el campo de visión del agente, por lo que la puede “ver”.

Esta información es empleada por el neuro-controlador para determinar en qué momento comenzar a reducir o aumentar la velocidad.

3.3.1.2 Salidas

Un aspecto crucial en el empleo de redes neuronales, es la interpretación que se hace de las salidas, cuyos valores están regidos por la función de activación. En esta investigación se emplea una función Sigmoid, por lo que los valores salida de cada neurona están en el rango [0 - 1].

I. Ángulo de giro del timón

Los valores arrojados por esta variable deben ser escalados al rango [-1, 1], que son los admitidos por la interfaz que controla la dirección del auto en el juego. Esto se puede lograr mediante la ecuación

$$x = 2 * (y - 0.5)$$

Un valor negativo implica girar a la izquierda, uno positivo girar a la derecha y cero no girar.

II. Velocidad deseada

Es necesario escalar la salida de esta variable al rango de valores de velocidad admisible por el carro [0, Velocidad Máxima], por lo que se multiplica por el valor máximo de velocidad.

3.4 Modelo de clases empleado

Los conceptos mencionados hasta el momento pueden ser representados en el dominio del software orientado a objetos mediante el lenguaje de modelado unificado (UML).

En la Figura 17 se muestra el modelo conceptual simplificado de la biblioteca de clases para redes neuronales artificiales Flood, para que se ajuste a los requisitos de este proyecto.

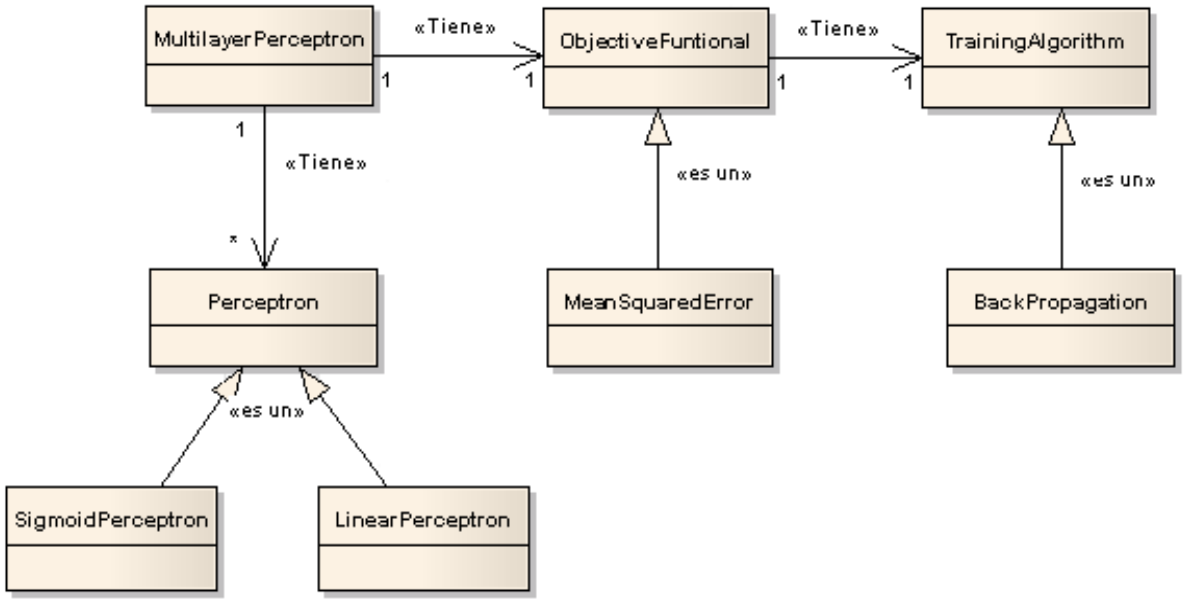


Figura 17: Modelo conceptual de un MLP (biblioteca de clases Flood).

Un *MultilayerPerceptron* está compuesto por varias capas de neuronas (*Perceptron*), tiene asignada una función objetivo (*ObjectiveFunctional*) que es mejorada por un algoritmo de entrenamiento (*TrainingAlgorithm*).

Perceptron, *ObjectiveFunctional* y *TrainingAlgorithm* son clases abstractas, ya que no representan conceptos concretos.

Las clases *SigmoidPerceptron* y *LinearPreceptron* representan el concepto de una neurona *Perceptron*, que se diferencian por la función de activación que utilizan: *Sigmoid* o *Lineal*.

Debido al problema de regresión funcional en cuestión, se utiliza el concepto del error cuadrado medio para aproximar la distribución de las variables de salida, condicionadas por las variables de entrada. Esto se representa con la clase concreta *MeanSquaredError*.

El *back-propagation* es la clase concreta que representa el algoritmo de entrenamiento utilizado.

En la Figura 18 se muestra el modelo conceptual de la solución propuesta para controlar los carros autónomos en el videojuego "*Rápido y Curioso"*.

Demo es la clase controladora, regida por la arquitectura G3D, donde el ciclo principal se compone de entrada de teclado, red, simulación, lógica y por último los gráficos.

Automobile es una clase abstracta que representa los carros que participan en el videojuego. *PlayerCar* y *Carbot* son tipos de *Automobile*, que representan a los carros que son controlados por el usuario y a los agentes autónomos respectivamente.

SensorialSystem es la clase que brinda información sobre el estado del juego, que es empleada por *NeuroCarController* para encuestar el *MultilayerPerceptron*, cuya respuesta es utilizada para controlar el *Carbot*.

Este proceso es ejecutado en el método de la clase *Demo* que controla la lógica del juego en cada ciclo de la simulación.

A medida que se desarrolla una carrera se salvan en un fichero los datos relevantes del comportamiento de un jugador. Para este fin se utilizan las clases *TrainingDataColector* y *GameSnapshot*. El fichero salvado es utilizado como conjunto de entrenamiento por el *MultilayerPerceptron* para adaptar su comportamiento al perfil del jugador.

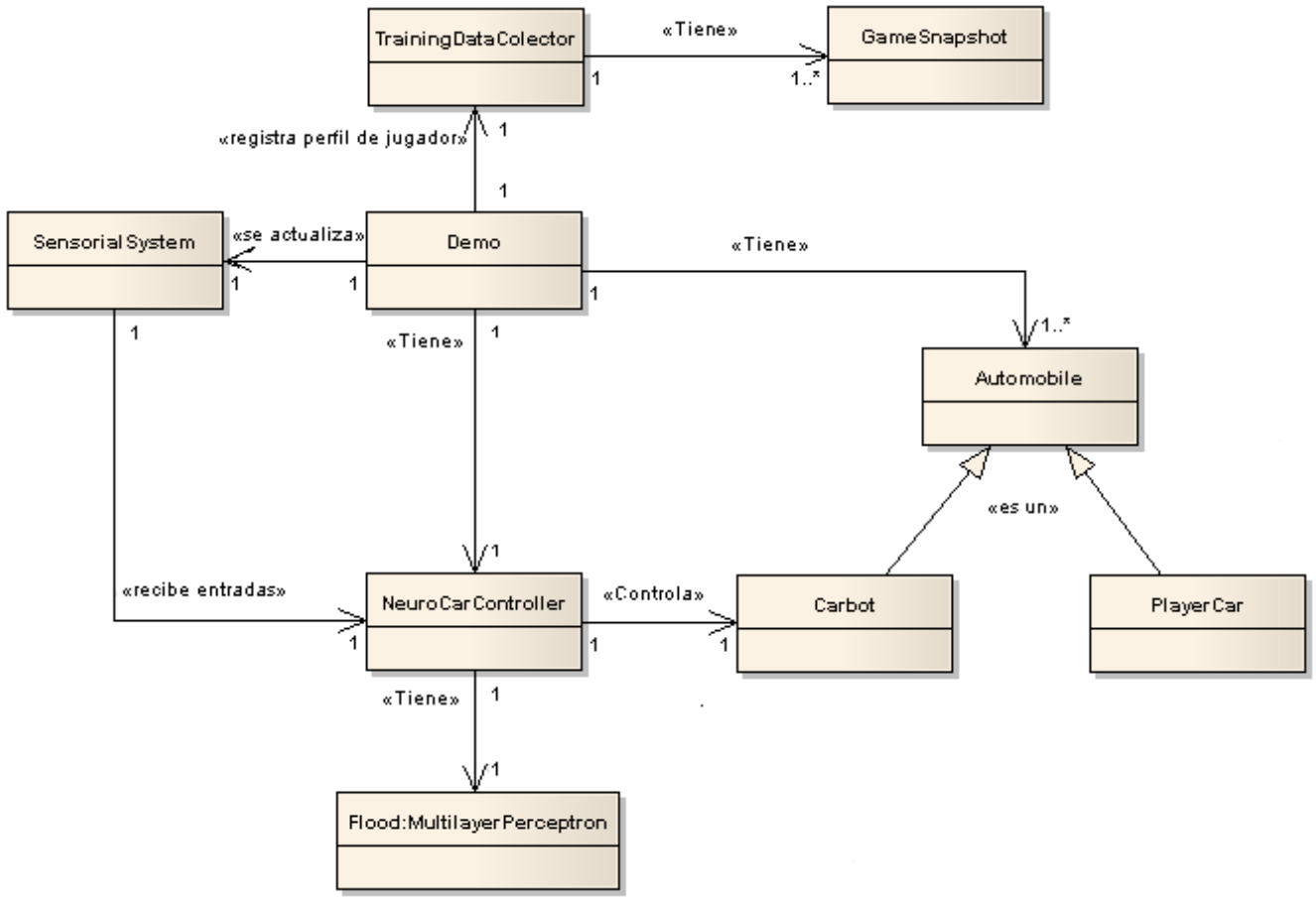


Figura 18: Modelo conceptual de la aplicación que emplea redes neuronales artificiales.

3.5 Proceso de entrenamiento

El aprendizaje supervisado se ajusta perfectamente a los requerimientos de este proyecto, ya que se pueden utilizar ejemplos del modo de juego de una persona para que el neuro-controlador aprenda a desempeñarse. Se empleó una estrategia de entrenamiento offline.

Es básicamente un problema de regresión funcional, en el cual a partir de una correspondencia de valores de entrada y salida, se actualizan los pesos de la red mediante el algoritmo back-propagation. Como resultado se obtiene una red que es capaz de responder correctamente ante patrones de entrada desconocidos.

El conjunto de entrenamiento está compuesto por dos vectores: uno contiene los valores de las cuatro variables de entrada, que describen una instantánea del juego; y el otro está formado por los valores de velocidad y ángulo de timón deseados para reaccionar ante dicha situación.

Se utilizaron dos variantes para obtener los datos de adiestramiento: en una se generan automáticamente las entradas y luego se le ofrece la salida deseada correspondiente mediante una aplicación diseñada con ese objetivo; mientras que en la otra se obtienen los datos del comportamiento desempeñado por el jugador en una carrera.

Ambas formas de entrenar reportan utilidad, mediante la primera la red recibe ejemplos que cubren la mayor parte del dominio de cada variable, aumentando las probabilidades de generalización y dotando al neuro-controlador de un conocimiento básico. Cuando se entrena usando la segunda variante, el comportamiento estará más adaptado a las características del jugador. Combinando ambas, el agente autónomo comienza la carrera con el conocimiento básico que le proveen los datos generados automáticamente. A medida que avanza la competencia, se va personalizando su comportamiento para las condiciones específicas de cada pista según el perfil del jugador; aumentando de esta manera el nivel de dificultad y planteándole siempre un reto aceptable al usuario y con él la vida útil del sistema.

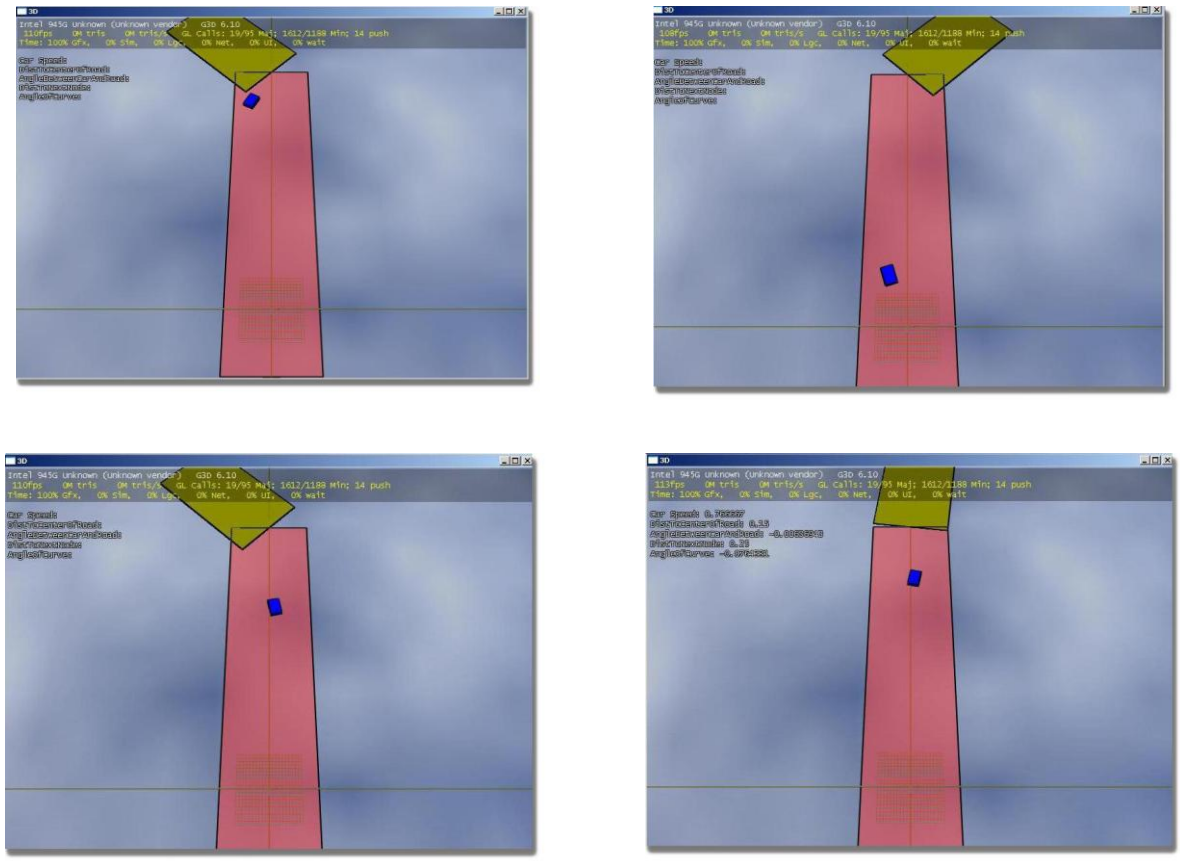


Figura 19: Imágenes del proceso de entrenamiento (entradas generadas automáticamente)

4

CAPÍTULO

ANÁLISIS DE LOS RESULTADOS

Como se ha comentado, para emplear una red neuronal artificial es necesario pasar por varias etapas: se selecciona el modelo de red, la función de activación y el algoritmo de aprendizaje; se definen las variables que aportan información relevante para entender y resolver el problema en cuestión (que es modelado matemáticamente); se obtiene un conjunto de datos de entrenamiento que permiten a la red adquirir el conocimiento necesario para responder ante diferentes situaciones.

Sin embargo, se deja buena parte del esfuerzo a la experimentación, donde se obtiene el número de capas ocultas y de neuronas por cada una de ellas, el valor del índice de aprendizaje, entre otras. Es fundamental la interpretación de los valores de salida de la red, lo que se debe mejorar durante este proceso.

Básicamente se prueban distintas arquitecturas, se entrena varias veces con diferentes pesos iniciales; también se modifica el índice de aprendizaje y el factor de corrección del error en busca de un comportamiento más óptimo.

La biblioteca de clases Flood provee cuatro criterios de parada para el algoritmo back-propagation, que pueden ser utilizados a conveniencia. En este proyecto se utilizaron dos de ellos: Cuando se alcanza un número máximo de 200 iteraciones; o cuando el error cuadrado medio sea menor que 0.01, este criterio garantiza que se ejecute al menos una iteración, ya que su valor es calculado al final de cada una de ellas.

Para asegurar que se tuvieran en cuenta todos los ejemplos al menos una vez, no se utilizó un tiempo de procesamiento máximo como criterio de parada; aunque esta opción debe ser considerada para reducir el tiempo de procesamiento cuando se entrene la red durante el juego.

Uno de los factores que ha frenado la difusión de las redes neuronales artificiales es que son difíciles de entender y depurar. Esto significa que cuando no se desempeñan correctamente, no existe un mecanismo preciso para determinar la causa del problema.

Para poder comprender a fondo el funcionamiento de la red, se realizaron experimentos en los que se modificó de forma gradual su arquitectura y parámetros principales.

4.1 Funcionamiento de algunas arquitecturas empleadas

Arquitectura 3-4-3

Una de las primeras aproximaciones empleadas fue una arquitectura 3-4-3 (tres neuronas en la capa de entrada, cuatro en la oculta y tres en la de salida).

Las variables de entrada empleadas fueron la velocidad del carro, la distancia al centro de la pista y el ángulo entre la dirección del carro y la posición del próximo nodo relativa al mismo. Las de salida fueron la posición del acelerador, la posición del freno y el ángulo de giro del timón.

El objetivo principal de esta versión fue comprobar la capacidad del neuro-controlador para seguir la trayectoria descrita por los nodos del grafo, y que además reconociera cuando el carro se aproxima al borde de la pista.

Para entrenarla se generaron automáticamente 72 estados, teniendo en cuenta las combinaciones de los valores que pueden tomar las variables de entrada. Para cubrir la mayor parte del dominio de estas variables, se dividieron en rangos con diferentes significados:

Los valores de velocidad que se alcanzan con el modelo físico empleado se encuentran en el rango $[0; 55]$, por lo se categorizó la velocidad en:

- ***Alta*:** por encima de 45.
- ***Media:*** entre 15 y 45.
- ***Baja:*** por debajo de 15

La distancia al centro de la pista, que se calcula como una proporción entre la distancia del carro al segmento actual y el menor de los radios entre los nodos actual y siguiente, se dividió en:

- ***Próximo al centro*** por la izquierda o por la derecha: valores modularmente menores que 0.5.
- ***Próximo a los bordes*** izquierdo o derecho: valores modularmente menores que 1 y mayores que 0.5.
- ***Fuera de la pista*** por la izquierda o por la derecha: valores modularmente mayores que 1.

El ángulo entre la dirección del carro y el próximo nodo puede ser clasificado en:

- ***Bueno***: valores modularmente menores que 5 grados.
- ***Malo***: valores modularmente superiores a 5 grados.

Tratando de minimizar el ángulo, se logra que el carro se dirija hacia el próximo nodo.

Los valores referidos a la izquierda son negativos y los referidos a la derecha son positivos.

Para completar el conjunto de entrenamiento, se le asocia a cada estado generado un vector que representa una serie de comandos sobre el carro para que reaccione ante dicha situación.

La posición del acelerador puede ser clasificada en:

- **Mínima:** valores entre 0 y 30.
- **Media:** valores entre 31 y 70.
- **Máxima:** valores entre 71 y 100.

La posición del freno puede ser clasificada en:

- **Mínima:** valores entre 0 y 10.
- **Media:** valores entre 11 y 30.
- **Máxima:** valores entre 31 y 60.

El ángulo de giro del timón puede ser:

- **Izquierda:** -1.
- **Derecha:** 1.
- **Centro:** 0.

Esta primera versión despejó las dudas existentes sobre la eficiencia de consultar al neuro-controlador en cada ciclo de simulación del juego, ya que el método donde se ejecuta consume solamente el 1% del procesamiento.

Una vez entrenada la red, se obtuvo un carro autónomo capaz de seguir la trayectoria del circuito y mantenerse alejado de los bordes de la pista, aunque con problemas visibles:

La velocidad variaba sin sentido aparente, provocado por la falta de información de cuando debe acelerar o desacelerar. En ocasiones el carro se detenía, ya que los valores del freno dominaban sobre los del acelerador. No mantenía una dirección estable, causado por variaciones demasiado bruscas del ángulo de giro del timón (presumiblemente por una mala interpretación de la salida de la red). Por otra parte la conducta mostrada no era del todo natural, ya que siempre intentaba buscar el centro de la carretera.

El comportamiento desempeñado por el carro con 4, 5 o 6 neuronas en la capa oculta fue similar. Se optó por mantener 4 ya que implicaba menor cantidad de procesamiento. Con 3 neuronas también se probó, pero la red no contaba con la capacidad suficientemente para adquirir el conocimiento necesario.

Arquitectura 5-4-2

Para mejorar el rendimiento del neuro-controlador se añadieron 2 nuevas variables en la capa de entrada: el ángulo de la próxima curva y la distancia a la que se encuentra el carro de la misma.

Además se sustituyó el ángulo entre la dirección del carro y el próximo nodo por el ángulo entre la dirección del carro y la pista.

El ángulo de la próxima curva se clasificó en:

- **Riesgo:** ángulo mayor de 75 grados.
- **Medio:** ángulo comprendido entre 15 y 75 grados.
- **Leve:** ángulo comprendido entre 0 y 15 grados.

Una curva a la izquierda se representa con ángulo negativo y una curva a la derecha se representa con ángulo positivo.

La distancia a la próxima curva se clasificó en:

- **Fuera del rango visión:** La distancia es mayor que el rango de visión del carro.
- **Lejos:** La distancia es mayor que la mitad del rango de visión del carro.
- **Cerca:** La distancia es menor que la mitad del rango de visión del carro.

El ángulo entre la dirección del carro y la pista puede ser clasificado en:

- **Bueno:** valores modularmente menores que 1 grado.
- **Malo:** valores modularmente superiores a 1 grado.

Se generaron automáticamente 1296 combinaciones según los estados posibles que pueden tomar las variables de entrada.

Por lo general los conductores no aceleran y frenan al mismo tiempo, por lo que es lógico que estas variables sean mutuamente excluyentes. A partir de esta conclusión, se decidió combinar las variables de salida posición del freno y posición del acelerador en una sola que representa la velocidad deseada.

Cuando se entrenó esta red, el carro fue capaz de acelerar o desacelerar en dependencia del ángulo de la curva y la distancia a la que estuviera de la misma. Su dirección fue más estable, ya que se mantenía paralelo a la trayectoria de la pista sin necesidad de buscar constantemente su centro y era capaz de alejarse de los bordes.

Sin embargo se pudo ver que cuando el segmento de la pista por donde transitaba el auto era estrecho, este tendía a dar giros bruscos; lo cual era causado porque la variación de la distancia al centro de la pista era mucho mayor en los tramos estrechos que en los anchos.

Arquitectura 4-4-2

Al tener como variable de salida la velocidad que se desea que alcance el carro, es innecesario y poco eficiente mantener como entrada la velocidad actual. De esta forma se evita el uso de información redundante, se agiliza el proceso de entrenamiento y el tiempo de respuesta de la red.

Se generaron automáticamente 432 combinaciones según los estados posibles que pueden tomar las cuatro variables de entrada.

Para resolver los giros bruscos del timón se modificó la interpretación que se daba a la variable de salida de la red ángulo de giro del timón.

Hasta este momento, valores menores que 0.3 se consideraban como un giro completo del timón a la izquierda, con valores superiores a 0.6 se giraba completamente a la derecha y con el resto se mantenía derecho el timón. Esta acción provocaba inconsistencia en la dirección del carro, sobre todo a altas velocidades.

Para alcanzar un comportamiento más estable se escaló el resultado de esta variable al rango [-1; 1] y se determinó girar el timón gradualmente a la izquierda o la derecha en dependencia de su valor numérico. O sea que para girar completamente el timón los valores deben ser modularmente cercanos a uno, mientras que valores menores significan girar en menor grado.

Cuando se entrenó esta nueva red, el carro fue capaz de recorrer la pista rápidamente, evitando choques contra las paredes, acelerando en las rectas y disminuyendo la velocidad en las curvas.

Hasta ahora se ha explicado el proceso de entrenamiento con los datos generados automáticamente. Su mayor problema es que solo se pueden generar las entradas. Las salidas hay que agregarlas manualmente, lo cual es muy incómodo y tedioso.

Por esta razón se prefiere recolectar los datos de entrenamiento durante la carrera de un jugador real. Mientras va compitiendo, se salvan en un fichero los valores correspondientes a cada una de las variables de interés para la red, que luego se entrena a partir de esos ejemplos.

Las redes entrenadas con datos generados automáticamente y las entrenadas con el perfil de un jugador presentan un comportamiento similar.

Se realizaron pruebas en situaciones críticas para las cuales la red no fue explícitamente entrenada, por ejemplo: pistas y trayectorias con ángulos de giro diferentes, se ubicó el carro fuera de la pista y en sentido contrario. En todas estas variantes su desempeño fue siempre adecuado, demostrando capacidad de generalización, una de las características esenciales de las redes neuronales artificiales.

El tiempo de entrenamiento varía según la cantidad de ejemplos y la distribución inicial de los pesos, ya que mientras más se acerquen a los valores óptimos, menor cantidad de iteraciones serán necesarias para completarlo.

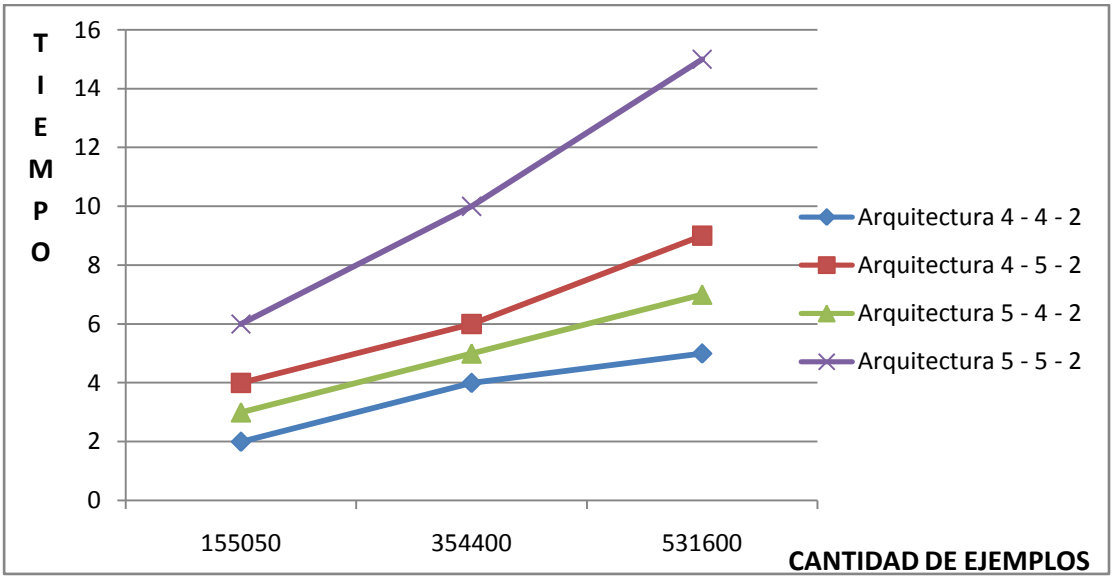


Figura 20: Tiempo de entrenamiento vs. cantidad de ejemplos.

Se realizó un estudio del tiempo de entrenamiento de algunas de las redes utilizadas. Los resultados se encuentran reflejados en la Figura 20, donde el eje de las X representa la cantidad de ejemplos y el eje de las Y el tiempo en minutos.

Como puede observarse, la red de arquitectura 4 - 4 - 2 fue la que menos tiempo de entrenamiento consumió, esto se debe a que está formada por menor cantidad de neuronas. Por otra parte, aunque hay dos redes con la misma cantidad de neuronas (11), la de arquitectura 5 - 4 - 2 se demora menos que la de arquitectura 4 - 5 -2 para un mismo número de ejemplos. Se puede concluir que un mayor número de neuronas en la capa oculta dificulta el proceso de entrenamiento, ya que en ellas se lleva a cabo la mayor parte del procesamiento de la red.

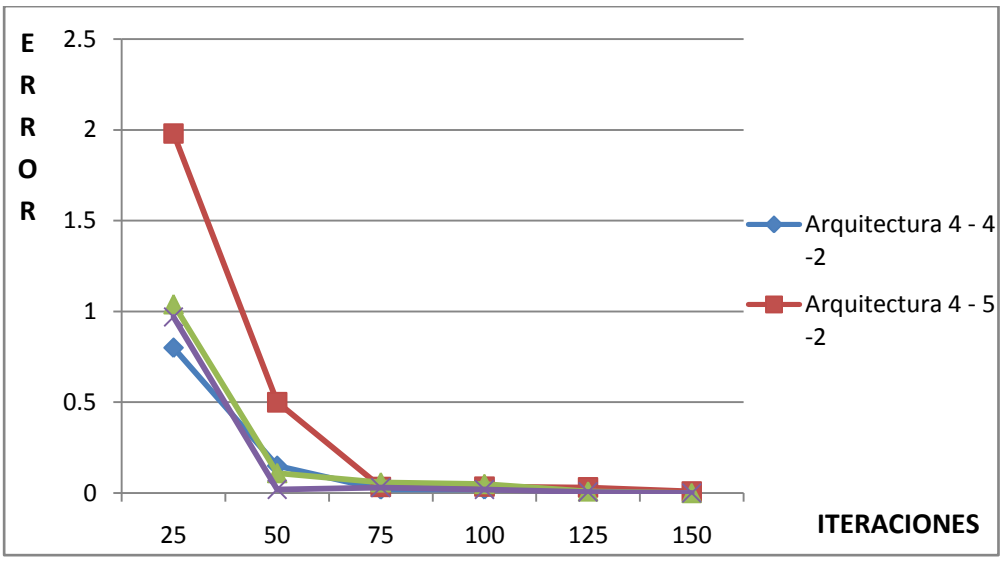


Figura 21: Error vs. Número de iteraciones

En la Figura 21 se muestra una gráfica que representa el error cometido por la red en las diferentes iteraciones durante el proceso de entrenamiento. A partir de la iteración 100 el error cometido al clasificar los ejemplos es bastante aceptable.

Se realizó un video que mostraba el desempeño de un carro controlado por la red neuronal en una pista de carreras. Se entrevistaron 350 personas entre estudiantes y profesores de la Universidad de las Ciencias Informáticas sobre el nivel de realismo observado en el video.

De los encuestados, 263 identificaron el comportamiento como el desempeñado por una persona (alto realismo), 33 estuvieron indecisos (realismo medio) mientras que 54 notaron que el auto era conducido por un robot (bajo realismo).

Figura 22: Encuesta sobre nivel de realismo del auto controlado por la red neuronal artificial.

CONCLUSIONES

En el demo que acompaña esta investigación se demuestra que es posible utilizar eficientemente redes neuronales artificiales para controlar agentes autónomos en un videojuego de carreras de automóviles. Esta técnica incorpora capacidad de aprendizaje a los elementos controlados por la computadora, permitiéndoles adaptarse a nuevas situaciones; además mejora el realismo del juego al conseguir oponentes artificiales que no exhiben un comportamiento rígido y esquemático.

Existen diversas redes neuronales artificiales que fueron creadas para resolver problemas específicos. Seleccionar una u otra depende de las características del entorno donde se va a emplear.

Para conformar la arquitectura de la red hay que transitar por un período de prueba y error en el que se evalúa su desempeño. En muchas ocasiones el comportamiento observado no es el que se desea, y al no existir un mecanismo determinista que ayude a identificar dónde radica específicamente el problema, hay que apelar entonces a la lógica, la intuición y la experimentación.

En esta investigación se consideró a la red como el cerebro de un competidor artificial, que recibe información de su entorno para determinar qué acción debe realizar. A partir de este enfoque se diseñó una arquitectura que responde a los requisitos del videojuego. Se trató de minimizar el número de capas y neuronas para obtener un mejor rendimiento y evitar ambigüedades en el análisis de los datos de entrenamiento.

Es sumamente importante el modelo matemático que representa la información empleada por la red para su desempeño.

Aún cuando se minimice el error durante el proceso de entrenamiento, la red puede desempeñarse incorrectamente. Esto puede ser originado por múltiples factores: insuficiencias en la arquitectura, datos de entrenamiento erróneos, caída en mínimos locales del algoritmo de aprendizaje, o que la red haya asimilado los ejemplos “de memoria” sin ser capaz de generalizar su comportamiento.

Se analizaron dos formas de obtener los datos de entrenamiento (generados automáticamente o del perfil de un jugador) y se demostró que la red aprende de forma similar con ambas variantes.

En esta investigación se integra la teoría sobre redes neuronales artificiales y su aplicación práctica en un videojuego de carreras, dos temas que normalmente son abordados por separado en la bibliografía existente; por lo que constituye un material de consulta para todo aquel que esté interesado en emplear redes neuronales artificiales en videojuegos.

RECOMENDACIONES

- Profundizar en el estudio de técnicas no deterministas de la inteligencia artificial para su empleo en videojuegos.
- Estudiar la forma de combinar técnicas deterministas y no deterministas de la inteligencia artificial para obtener videojuegos de mayor realismo y aceptación del público.
- Integrar los resultados de esta investigación (materializados en el demo que la acompaña) en el módulo que controla la inteligencia artificial del videojuego *"Rápido y Curioso"*.
- Utilizar la habilidad de generalización y aprendizaje de las redes neuronales artificiales para ajustar el comportamiento de los carros autónomos del videojuego al perfil de los competidores humanos:
    - Se recolecta información del modo de juego del competidor.
    - Se entrena la red al concluir la carrera solo si el tiempo logrado por el jugador es bueno.
    - Se comprueba si la red fue capaz de generalizar. Esto se puede lograr mediante un conjunto de control. Dicho conjunto no es más que una parte de los ejemplos que no se usan durante el entrenamiento. Una vez concluido este se emplea para calcular el error cometido por la red, obteniendo una medida de su rendimiento.
- Antes de personalizar el comportamiento de los agentes autónomos, se recomienda que el neuro-controlador posea un conocimiento básico, para minimizar las probabilidades de obtener una conducta inadecuada.

BIBLIOGRAFÍA

1. Megaconsolas.com. [online] [cited: 5 15, 2008.]

http://www.megaconsolas.com/noticias/2007/12/65636.laindustriadelosvideojueg.asp.

2. *joinforfree.* [Online] [Cited: 5 15, 2008.] http://joinforfree.nuskyway.com/es/market.cfm.

3. Flood Reference Manual. *cimne.* [Online] [Cited: 10 29, 2007.]

http://www.cimne.com/flood/docs/FloodA1ReferenceManual/index.html.

4. *G3D Engine.* [Online] [Cited: 10 18, 2007.] http://g3d-cpp.sourceforge.net/.

5. *AI Game Dev.* [Online] [Cited: 10 2, 2007.] http://aigamedev.com/.

6. *Blog Julian Togelius.* [Online] [Cited: 12 5, 2007.] http://togelius.blogspot.com/.

7. *AI Junkie.* [Online] [Cited: 12 1, 2007.] http://www.ai-junkie.com/misc/hannan/hannan.html.

8. *Generation 5.* [Online] [Cited: 12 1, 2007.] http://www.generation5.org/content/2001/hannan.asp.

9. **Togelius, Julian and Lucas, Simon.** *Evolving Controllers for Simulated Car Racing.* [pdf] 2005.

10. —. *Evolving robust and specialized car racing skills.* [pdf] 2006.

11. **Gallego, Francisco.** *Driving-Bots with a Neuroevolved Brain: Screaming Racers.* 2003.

12. **Alexandros, Agapitos, Togelius, Julian and Lucas, Simon.** *Evolving Controllers for Simulated Car Racing using Object Oriented Genetic Programming.* 2007.

13. **Anna, Booth.** *Using Neural Networks to Improve Behavioural Realism in Driving Simulation Scenarios.* 2007.

14. **Togelius, Julian and Simon, Lucas.** *Arms races and car races.* 2006.

15. **Togelius, Julian, Simon, Lucas and Burrow, Peter.** *Multi-population competitive co-evolution of car racing controllers.* 2007.

16. **Göktas, Haldun, Çavusoglu, Abdullah and Sen, Baha.** *THE USE OF ARTIFICIAL NEURAL NETWORKS IN SIMULATION OF MOBILE GROUND VEHICLES.* 2007.

17. **Bello Pérez, Rafael.** *Curso Introductorio a las Redes Neuronales Artificiales. Departamento de Ciencia de la Computación.* s.l. : Universidad Central de las Villas, 1993.

18. **Buckland, Mat.** *AI Techniques for games programming.* 2002.

19. —. *Programming Game AI by Example.* 2005.

20. **Rabuñal, Juan and Dorado, Julián.** *Artificial Neural Networks in Real Life Applications.* 2005.

21. **Charles, Darryl and al, et.** *Biologically Inspired Artificial Intelligence for Computer Games.* 2008.

22. **Munakata, Toshinori.** *Fundamentals of the New Artificial Intelligence. Neural, Evolutionary, Fuzzy and More.* 2008.

23. **McGonigal, Jane.** *All Game Play is Performance: The State of the Art Game.* 2005.

24. **Buckland, Mat.** *Programming Game AI by Example.* 2005.

25. **Autores, Colectivo de.** *AI game Programming Wisdom.* 2002.

26. **Martin, Bonifacio and Sanz, Alfredo.** *Redes Neuronales y Sistemas Difusos.* 2000.

27. **O'Reilly.** *AI for Game Developers.* 2004.

ANEXOS

Figura 23: Auto controlado con Redes Neuronales

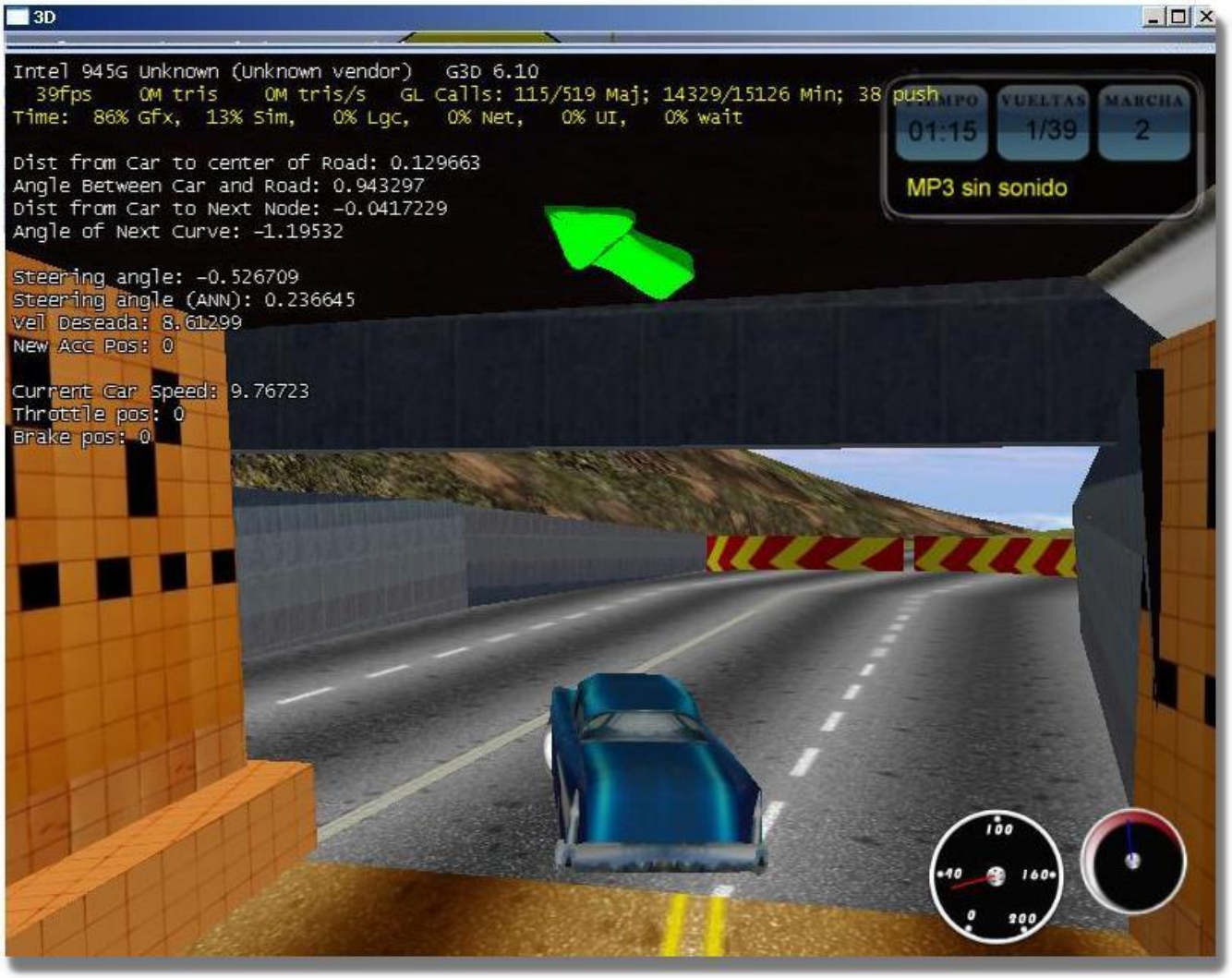


Figura 24: Auto controlado con Redes Neuronales

Figura 25: Auto controlado con Redes Neuronales

GLOSARIO

Agente autónomo: Sistema situado en un entorno, que tiene la capacidad de sentirlo y actuar sobre él, a través del tiempo, persiguiendo sus propios objetivos de forma que afecte lo que siente en el futuro.

Ajuste del Mapping: Grado de convergencia del error en el proceso de entrenamiento de redes neuronales.

Algoritmo de entrenamiento: Proceso que le permite a una red neuronal artificial aprender de un conjunto de patrones.

A-Life (Vida Artificial): Sistemas multi-agentes que intentan aplicar algunas de propiedades universales de los sistemas vivientes a agentes inteligentes en un entorno virtual.

Aprendizaje: Proceso mediante el cual un individuo adquiere conocimiento y experiencia, de forma que le permita modificar su comportamiento y adaptarlo a nuevas condiciones de su entorno.

Elemento inteligente: Individuo auto-controlado, capaz interactuar con su entorno y conseguir metas.

Inteligencia artificial: Rama de la informática que se encarga de crear software y hardware capaces de imitar la inteligencia humana.

Iteración: Ciclo en el que se ha recorrido el conjunto de datos de entrenamiento completo.

Perceptrón: Tipo de neurona artificial. Unidad básica de un perceptrón multicapa.

Perceptrón multicapa: Tipo de red neuronal artificial formada por múltiples capas de perceptrones.

Propagación del error hacia atrás: Algoritmo de aprendizaje que se puede utilizar para entrenar un perceptrón multicapa. Tiene como objetivo minimizar el error cometido por la red ante un conjunto de patrones de entrenamiento ajustando los pesos entre sus conexiones.

Red neuronal artificial: Técnica no determinista que emula el cerebro humano. Está compuesto por un conjunto de neuronas interconectadas que colaboran entre sí para obtener una salida.

Técnica no determinista: Técnica de la inteligencia artificial que incorpora un alto grado de incertidumbre.