2009-2010

Universidad Complutense
de Madrid

SISTEMA PARA LA PREDICCIÓN DE POSICIÓN Y SEGUIMIENTO DE UN CONJUNTO DE NÁUFRAGOS BASADO EN REDES NEURONALES

Proyecto Fin de Máster en Ingeniería Informática para la Industria.

Máster en Investigación en Informática,
Facultad de Informática, Universidad Complutense de Madrid

Fernández Ramírez, Francisco
Dirección: López Orozco, José Antonio
E-mail: *francisco.fernandez.ramirez@gmail.com*

El/la abajo firmante, matriculado/a en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: **"Sistema para la predicción de posición y seguimiento de un conjunto de naufragos basado en redes neuronales",** realizado durante el curso académico 2009-2010 bajo la dirección de **José Antonio López Orozco,** en el Departamento de **"Arquitectura de Computadores y Automática"**, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Firmado: **Francisco Fernández Ramírez**

RESUMEN

El objetivo del trabajo es aplicar las redes neuronales al problema de rescate de naufragos en alta mar. En un instante determinado se produce un naufragio y es necesario enviar uno o varios UAVs para la localización y rescate de los naufragos en el menor tiempo posible.
Para conseguir el rescate de los naufragos se utilizarán las redes neuronales de modo que indiquen a los UAVs donde deben buscar. Dos usos típicos de las redes son aplicados para resolver el problema: la predicción y la toma de decisiones.
La predicción realizada por las redes neuronales permite que se estimen de forma contínua las corrientes y vientos que desplazán a los naufragos, de modo que se pueda realizar una estima de la posición de éstos e indicar al UAV donde comenzar a buscar.
La red neuronal, en modo toma de decisiones, estima a partir de la información que se va adquiriendo, hacia donde debe dirigirse el UAV con el fin de localizar a todos los naufragos en el menor tiempo posible.
Una vez entrenadas las redes neuronales y verificado su funcionamiento, se ha construido en Matlab un entorno de simulación que permite provocar un naufragio y observar el comportamiento de un UAVs sencillo en la localización de los naufragos.

SUMMARY

The goal of this paper is to apply the neural networks to solve the rescue problems of shipwreckeds at high seas. Once given moment, a shipwreck is happend and if need be to send one or more UAVs to locate and rescue the shipwreckeds in a shortest time as possible.
To get the rescue of the shipwrecked, we'll use neural networks so as to show where the UAVs should be seeking. Two typical uses of the networks are applied to solve the problem: prediction and make a decision.
The prediction performed by neural networks allows us that the wind and current are estimated continuously which move the shipwrecked, so that you can make an estimate the position of the shipwrecked and tell the UAV where it searchs.
The neural network, in make decision mode, estimate from the information that have been gathered, where the UAV should be gone to locate every shipwrecked in the shortest time as possible.
Once trained and tested neural networks in operation, it has built in Matlab a simulation environment that allows cause a shipwreck and watch the behavior of a plain UAVs in the shipwrecked locations.

LISTA DE PALABRAS CLAVES

UAV (Vehículo aéreo no tripulado), Predicción, Red Neuronal Artificial, viento, marea, red Backpropagation, entrenamiento de la red, simulación de la red, sistema inteligente, predicción de mareas, resolución de la malla.

KEY WORDS

UAV (Unmanned Aerial), Prediction, Artificial Neural Network, wind, tide, Backpropagation network, network training, network simulation, Expert System, Wave forecast, Grid resolution.

CONTENIDO

ILUSTRACIONES

CAPÍTULO 1. INTRODUCCIÓN

En este capítulo plantearemos el problema de la predicción del movimiento y localización de los náufragos durante un naufragio, concretaremos nuestros objetivos y buscaremos una solución al problema de localización de dichos náufragos con las tecnologías que existen, hoy en día, a nuestro alcance.

Problema a resolver

A lo largo de los tiempos, los seres humanos hemos tenido el ansia de descubrir, de viajar cada vez más lejos, hasta que en la actualidad prácticamente hemos colonizado todo el planeta. Prácticamente hemos colonizado y estructurado las zonas en las que vivimos. Antiguamente si un barco salía a la mar y alguno de sus componentes se caía al agua, o el barco naufragaba con todos sus componentes lo único que podíamos hacer era rezar por el alma de estos.

Según han ido pasando siglos, nos hemos ido preocupando más y más de este problema, primero construyendo barcos con los que ir a buscar a dichos náufragos, con realmente pocas posibilidades de encontrarlos. En esa época según llegaba la noticia al pueblo de que había habido un naufragio, salían de forma inmediata un conjunto de barcos, cuantos más mejor para ocupar el mayor espacio de búsqueda, que se dirigían hacia la zona donde se había producido dicho naufragio. Cuando llegaban allí, al cabo de unas horas o quizá días, normalmente el marinero o marineros con más experiencia intentaban adivinar la dirección del viento y las corrientes, normalmente por la experiencia pasada, con la idea de saber hacia dónde habían sido arrastrados nuestros náufragos.

Se ven claramente algunos indicios de la necesidad de ocupar el mayor espacio y de intentar predecir la dirección hacia donde el viento y las mareas [SebasGue06] podrían haber llevado a nuestros náufragos.

Hoy en día la cosa se ha tecnificado bastante, y cuando existe un problema de estos, inmediatamente se llama al servicio de guardacostas que pone en funcionamiento un sistema con aviones pilotados y barcos que se dirigen a la zona del naufragio [BusRes]. El arte de adivinación de la dirección de los náufragos también ha cambiado bastante ya que hoy en día normalmente existen planos de las zonas marítimas donde más o menos están especificadas las velocidades de los vientos, direcciones de estos e igual con las mareas y corrientes marinas.

Con estos datos se puede intentar predecir y dirigir un equipo de rescate realizando simulaciones numéricas que nos permitan saber la zona hacia la que se dirigen dichos náufragos.

Actualmente, el desarrollo de la tecnología ha permitido la construcción de UAV [UAV], que son prácticamente automáticos, con un cerebro que toma las decisiones necesarias para mantener el vuelo y la dirección a la que el aparato quiere dirigirse según las instrucciones que le demos sobre lo que queremos que haga.

En este trabajo lo que pretendemos es desarrollar un método para predecir y dirigir esos UAV de forma automática con la idea de que no sea necesaria la intervención humana ni en la predicción del movimiento de los náufragos ni en la toma de decisiones a la hora de buscar a dichos

náufragos. El sistema debe de ser capaz de predecir la dirección de los náufragos y también con los datos obtenidos durante el vuelo, poder tomar las decisiones necesarias para localizar a los náufragos restantes [Fossen94].

Objetivos

El proyecto nació con la idea de construir un modelo que demostrara, lo más posible, la potencia de las redes neuronales a la hora de realizar predicciones y toma de decisiones. Para ello se definieron un conjunto de objetivos a alcanzar usando como base una red neuronal dentro de la solución.

Los objetivos que nos hemos propuesto alcanzar dentro de esta investigación no son nuevos, existen otros modelos matemáticos [HorPag], [QLEARNING] y [NNWPREDICTION] que se pueden usar para conseguir el mismo objetivo y a los que haremos referencia más adelante. Veremos que aunque la existencia de dichos modelos para predecir el movimiento de los náufragos, como las series temporales o los modelos dinámicos, no tienen la misma flexibilidad que nos aportan los modelos de redes neuronales. Lo que vamos a ver dentro de nuestro desarrollo es la gran capacidad adaptativa de las redes neuronales. El modelo de redes neuronales nos permite, que ante un cambio de las condiciones iniciales, simplemente con reentrenar la red, el comportamiento de nuestro modelo cambie y se adapte a las nuevas condiciones [HilMart95]. Aquí es donde podríamos hacernos la pregunta, ¿Pueden las redes neuronales resolver el problema planteado?, creemos que sí y basándonos en ello se ha desarrollado este trabajo.

Por lo tanto este trabajo tiene una doble vertiente por un lado probar las capacidades ya conocidas de las redes neuronales y ver si aquí son aplicables, estas son, aprendizaje adaptativo, auto-organización, tolerancia a fallos, operación en tiempo real y fácil inserción dentro de la tecnología existente en un proyecto real [Patterson96]. Y por otro lado modelización el comportamiento de los náufragos y construir un sistema inteligente que aconseje la dirección que permita optimizar la localización de los náufragos [Fossen02].

En la primera parte del proyecto, se creará un modelo del comportamiento de las corrientes y vientos sobre el náufrago, a fin de poder ver el movimiento que este tendría a lo largo del tiempo [Fossen02]. Las características más importantes que vamos a estudiar sobre las redes neuronales son, su aprendizaje adaptativo, tolerancia a fallos y operaciones en tiempo real. Mediremos estos parámetros de las redes neuronales ya que por un lado aunque tengamos un mapa de las corrientes marinas y de los vientos de la zona, estos no son exactos y tampoco están completos, y si lo estuvieran, ocuparían un espacio prácticamente infinito y no tenemos memoria para tanto. Normalmente escogeríamos un conjunto de puntos más o menos significativos, e intentaríamos predecir lo que ocurre en el resto del espacio que hay alrededor de estos puntos, para eso la capacidad auto-adaptativa de las redes neuronales es bastante útil ya que son capaces de crear un modelo continuo a partir de un modelo discreto. Por otro lado su tolerancia a fallos y la capacidad para realizar las operaciones en tiempo real, una vez han sido entrenadas, nos permite crear un sistema sencillo de búsqueda a partir de las predicciones que va realizando la red.

Por otro lado las características que anteriormente hemos definido, nos permiten aprovechar los recursos de las redes neuronales para construir dentro del cerebro de nuestro UAV, un modelo de toma de decisiones que actuará una vez hemos llegado al punto donde el modelo de predicción ha dicho que se encontrarían los náufragos. La capacidad de la red neuronal para trabajar en tiempo real y su tolerancia a fallos la hace imprescindible dentro del cerebro del UAV [UAV],

uno de nuestros principales objetivos sería que ante cualquier problema el UAV no se perdiera, y por otro lado la capacidad auto-adaptativa hace que sea un recurso muy valioso a la hora de tomar decisiones de una forma flexible, sin la rigidez de otros modelos matemáticos donde, o lo tienes todo previsto, o el resultado probablemente no será correcto. Las redes neuronales tratan de tomar una decisión con los datos que se le suministra y tienen la gran ventaja, y de hecho uno de nuestros objetivos será aprovechar dicha ventaja, de ser auto-adaptativas y flexibles.

Planteamiento de la solución

Aproximación inicial a la solución

Inicialmente intentamos construir el recorrido a lo largo del tiempo de los náufragos, estos se desplazarán según las fórmulas que definen el modelo de viento *(v)* y de corriente *(c)*.

$$U_{wx} = V_{xc}\cos(\psi_{wc}) + V_{wv}\cos(\psi_{wv})$$
$$U_{wx} = V_{xc}\sin(\psi_{wc}) + V_{wv}\sin(\psi_{wv})$$

Donde V_w representa la celeridad con la que se mueve el objeto, y que será una cierta fracción de la velocidad del viento a, por ejemplo 10 metros de altura, y ψ_w es el rumbo que lleva el viento y las olas [Fossen94]. Para ello hemos añadido un pequeño ruido a la dirección y velocidad durante un periodo de tiempo y calculamos las posiciones de los náufragos que van ocupando en cada instante de tiempo. Podemos ver esta generación en la siguiente ilustración, cada mancha de náufragos está representada por un conjunto de náufragos, en un instante determinado, que se desplaza siguiendo las corrientes marinas, en la ilustración 1 la dirección de la corriente, sería desde la esquina superior derecha hacia la esquina inferior izquierda.

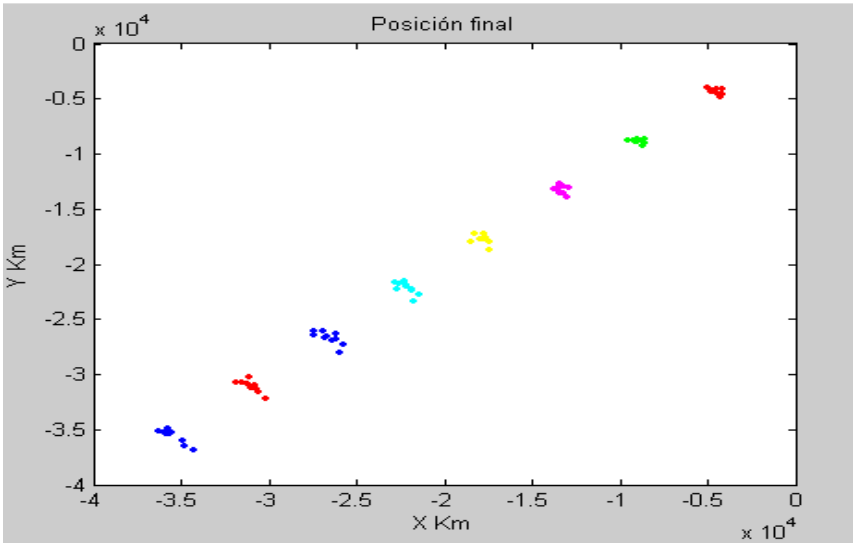


Ilustración 1: Desplazamiento de los náufragos.

Como se ve, los náufragos se habrán desplazado mientras el avión se dirige a la zona del naufragio, por lo que es necesario predecir donde se encontrarán los náufragos cuando llegue el equipo de rescate y dirigirnos a ese lugar en vez de donde se produjo el naufragio.

Para ello, crearemos una red neuronal entrenada con el componente final de vientos y corrientes asociado al mapa de mareas y vientos disponibles para la zona e el instante del naufragio, ahora incluiremos esta red dentro de nuestro sistema. Con los datos que nos de la red neuronal respecto

a la velocidad y dirección del viento y las mareas en la posición que se encuentre el náufrago, podremos calcular la siguiente posición donde este se debería encontrar en un instante de tiempo dt. Es importante reseñar que nuestro UAV nunca conoce la posición real de los náufragos y que es esta red predictiva la que nos tiene que dar esos datos.

Calcular la siguiente posición de nuestro náufrago es bastante sencillo, bastará con aplicar la siguiente fórmula.

$$X = X' + dt \cdot v \cdot \cos(a)$$
$$Y = Y' + dt \cdot v \cdot \sin(a)$$

De modo que se obtiene la indicación de donde se debe dirigir el UAV para buscar. A esta red la denominaremos red de predicción y se verá en el Capítulo 2.

El siguiente paso a aclarar es cómo se puede tomar la decisión de las acciones que hay que realizar una vez hemos llegado a la zona del rescate. En otras palabras hay que dotar de un cerebro al UAV que le permita definir una trayectoria que persiga o al menos trate de perseguir a los náufragos hasta recogerlos a todos.

Una vez estamos en la posición indicada por la red de predicción, se comienza a localizar a los náufragos. Los náufragos descubiertos, nuestra posición respecto a los náufragos descubiertos, la media de los náufragos descubiertos y su dirección. Son datos que debemos tener en cuenta para construir una red neuronal que nos permita ser entrenada de forma independiente y que haga que el UAV con los datos disponibles en cada momento tome la decisión más adecuada a la hora de perseguir, localizar y recoger a los náufragos restantes que faltan. Añadiremos, a los datos anteriores, el porcentaje de náufragos descubiertos por nuestro sistema, este porcentaje será un número entre 0 y 1. El porcentaje es un valor que deberemos estimar, ya que a priori es posible que no conozcamos exactamente el número total de náufragos que hay, desde el principio, en el mar. Así como toda aquella información que sea necesaria, en la ilustración 2 se puede observar que todavía no está claro qué parámetros serán necesarios utilizar. En el capítulo 3 veremos qué datos se han utilizado para el entrenamiento y uso de la red que guíe al UAV. Esta red la denominaremos red de decisión.

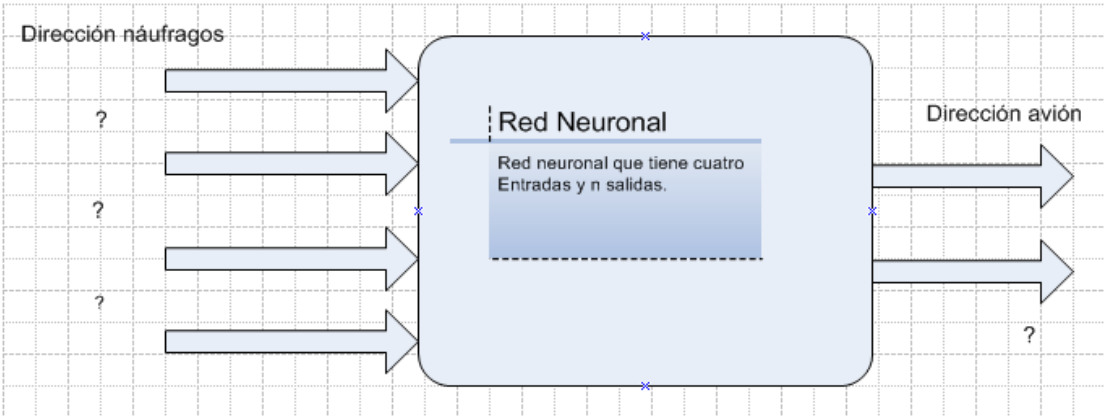


Ilustración 2: Esquema de la red neuronal de decisión.

Si el UAV no encontrara ningún valor debería aun así tomar una decisión, normalmente seguir la línea, que en teoría deberán de seguir nuestros náufragos, o retroceder si ve que no encuentra nada siguiendo dicha línea.

CAPÍTULO 2. PREDICCIÓN UTILIZANDO REDES NEURONALES

En este capítulo intentaremos realizar una introducción a la predicción, primero de forma general, estudiando algunas de las técnicas, conocidas, más utilizadas actualmente, para luego introducirnos en el mundo de la predicción con redes neuronales artificiales. Veremos las redes neuronales de forma genérica y nos introduciremos en el funcionamiento de la red neuronal que hemos usado en este trabajo, la feedforward-backpropagation, especificando tanto su estructura como sus usos más comunes, acabaremos explicando cómo hemos configurado las redes neuronales usadas dentro de este trabajo.

Introducción a la predicción

La predicción tal y como la conocemos es un intento de descubrir o intentar reproducir un evento que puede llegar a ocurrir en el futuro. Hoy en día el término más usado es forecast, ya que es en los mercados financieros donde más furor está haciendo, y quien más está invirtiendo en modelos de predicción.

Predicción viene del latín y significa, *"antes de que se diga"*, esto significa que una predicción es intentar descubrir, basándose en los indicios y en la información pre-existente, algo que es probable que suceda en el futuro [Forecast].

La idea es crear modelos matemáticos que nos permitan simular y predecir el comportamiento de un evento futuro. Existen muchos modelos para realizar predicciones hoy en día, desde modelos dinámicos que permiten modelar un comportamiento, modelos estadísticos basados en el análisis de datos, modelos estocásticos como las series temporales o de Markov [NNWPREDICTION], y hoy en día cada vez más se han empezado a usar modelos de inteligencia artificial y redes neuronales artificiales [MarkPred].

Podemos calcular el error de una predicción como la diferencia entre el valor actual de la predicción y el valor real que se produce

$$E_t = Y_t - F_t$$

Donde E es el error en la predicción, en el periodo t, Y es el valor actual en el periodo t, y F es la predicción en el periodo t.

Medición del error:

Mean Absolute Error (MAE) $$MAE = \frac{\sum_{t=1}^{N} |E_t|}{N}$$

Mean Absolute Percentage Error (MAPE) $$MAPE = \frac{\sum_{t=1}^{N} \left| \frac{E_t}{N} \right|}{N}$$

Percent Mean Absolute Deviation (PMAD)	$PMAD = \frac{\sum_{t=1}^{N} \lvert E_t \rvert}{\sum_{t=1}^{N} \lvert Y_t \rvert}$
Mean squared error (MSE)	$MSE = \frac{\sum_{t=1}^{N} E_t^2}{N}$
Root Mean squared error (RMSE)	$RMSE = \sqrt{\frac{\sum_{t=1}^{N} E_t^2}{N}}$
Forecast skill (SS)	$SS = 1 - \frac{MSE_{forecast}}{MSE_{ref}}$

Podemos comentar más modelos matemáticos, por ejemplo Qlearning que es un modelo de aprendizaje para predecir modelos basados en inteligencia artificial, Modelos estocásticos, probabilísticos, como son las ecuaciones probabilísticas de Cauchy y Gauss, Modelo de Montecarlo. También teniendo en cuenta que lo que usamos son redes neuronales, existen modelos neuronales que dependen del tiempo. Las redes recurrentes y Adaline pueden simular series temporales y por lo tanto servirnos como modelos de predicción [JuCamSan06].

Otros modelos matemáticos

Antes de comenzar con el modelo seleccionado para desarrollar nuestro proyecto de predicción, es bueno conocer que otros modelos diferentes a las redes neuronales feedforward, que nosotros usamos, se han desarrollado y pueden servir para dar una solución más o menos viable del problema. Es evidente, y de hecho queda para una posible ampliación del trabajo, la posibilidad de enfocar el problema con algunos de estos otros tipos de soluciones y comparar sus resultados con los resultados obtenidos por nuestra solución. Aquí solamente presentaremos de forma resumida las que nos han parecido más interesantes, de la bibliografía revisada, que hoy en día parece que son las más utilizadas. Comentaremos las posibles ventajas e inconvenientes de aplicar dichos métodos a nuestro proyecto de investigación.

Predicción con Q-Learning

Es una técnica de aprendizaje reforzado [QLEARNING-2] que trabaja mediante el aprendizaje de una función de acción-valor que nos da la utilidad esperada de tomar o seleccionar una acción dada en un determinado estado y seguido de una política de errores. El Q-Learning es un modelo bastante consistente, debido a que es capaz de comparar la utilidad esperada de las acciones disponibles sin ningún modelo de requerimientos del entorno. Existe una variación llamada delayed-Q-Learning que permite mejoras mediante PAC (Aproximación probable para corregir el aprendizaje) [QLEARNING-3], que está delimitado por procesos de decisión de Markov [QLEARNING-4].
Disponemos de una gran cantidad de ejemplos de Q-Learning en la página Web, [QLEARNING-1]. También en la página de IBM [QLEARNING-6]. Y una plataforma ya construida en [QLEARNING-7].

El algoritmo que resuelve el problema consiste en un agente, con estado *S* y un número de acciones por estado *A*. Cuando realizamos la acción *a* con $a \in A$, el agente puede moverse de un estado a otro. Cada estado provee al agente de una recompensa (puede ser real o natural) o un castigo (igual que el anterior). El objetivo del agente es maximizar las recompensas. Esto hace que el aprendizaje de cada acción sea óptimo para cada estado.

Este algoritmo tiene una función del tipo

$$Q : S \times A \rightarrow \mathbb{R}$$

Antes del comienzo *Q* debe retornar un valor fijado, elegido por designación. Entonces en cada intervalo de tiempo el agente consigue una recompensa, el estado cambia, y los nuevos valores son calculados para cada combinación de estados *s* de *S*, y para cada acción *a* de *A*. El núcleo del algoritmo es simplemente una actualización de los valores mediante iteraciones. Modificamos los antiguos valores realizando correcciones con la nueva información.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a_t)}_{\text{max future value}}}^{\text{expected discounted reward}} - \overbrace{Q(s_t, a_t)}^{\text{old value}} \right]$$

Donde $r_t + 1$ es la recompensa dada en el intervalo de tiempo $t + 1$, $\alpha_t(s_t,a_t)$ *con* $(0 < \alpha \leq 1)$ la tasa de aprendizaje. El valor del factor de descuento γ toma valor $0 \leq \gamma \leq 1$.

La fórmula es equivalente a:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)\big(1 - \alpha_t(s_t, a_t)\big) + \alpha_t(s_t, a_t)\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) \right]$$

El algoritmo finaliza cuando el estado $s_t + 1$ es un estado final (o, *"estado de absorción"*). Tenemos que darnos cuenta de que para todos los estados finales s_f, $Q(s_f,a)$ nunca es actualizada y conserva su valor inicial.

Este modelo donde la predicción estaría basada en un agente que contiene un conjunto de estados y acciones por estado, primero habría que definir correctamente ese número de estados y acciones por estado. Esta máquina de estados adaptativa parece adaptarse al modelo que hemos creado para resolver el problema, si el modelo cambiara en un futuro habría que cambiar el rango de estados y de acciones por estado para adaptarse al nuevo modelo. Es difícil de compararlo con nuestro modelo de redes neuronales así como valorar el resultado final pero lo que sí es fácil de valorar es el coste de modificarlo en caso de que en el futuro nos encontráramos con variaciones de nuestro modelo, evidentemente en ese apartado parece no ser el más indicado para la predicción y decisión de nuestro modelo.

Predicción con modelos de Montecarlo

Son una clase de algoritmos computacionales que confían en repetir ejemplos aleatorios para computar sus resultados. Los métodos de Montecarlo [MONTECARLO-1] son usados en

simulaciones físicas y sistemas matemáticos. Debido a la dependencia de ejecución de números aleatorios y pseudo-aleatorios, estos métodos son usados en cálculos de ordenadores donde es casi imposible conseguir un resultado exacto con algoritmos deterministas [MONTECARLO-2].

Los métodos de simulación de Montecarlo son muy usados en estudios de sucesiones de números como, fluidos, materiales, estructuras celulares. Más allá, los métodos de Montecarlo son usados para modelar fenómenos donde existe incertidumbre sobre la entrada por ejemplo en el cálculo de riesgos dentro del entorno de negocios. Estos métodos comúnmente son usados en matemáticas para evaluar integrales multidimensionales. Y donde probablemente más nos puede interesar, su uso para realizar análisis de riesgos y predicciones donde puede ser comparado con la intuición humana. Los métodos de Montecarlo son muy usados para realizar simulaciones aplicadas a un espacio de exploración (por ejemplo explotaciones petrolíferas [SebasGue06]) donde no existe otra forma mejor que la propia intuición humana, es en este punto donde mejor se pueden comparar con modelos de redes neuronales.

No existe un solo método de Montecarlo, se puede decir que existen un conjunto más o menos amplio de aproximaciones que siguen un patrón particular.

1. Definir el modelo de posibles entradas.
2. Generar las entradas de forma aleatoria desde un dominio usando una distribución probabilística.
3. Realizar la computación determinista usando las entradas.
4. Finalmente, agregar los resultados de esta computación al final.

Por ejemplo esto es muy usado en finanzas [MONTECARLO-3], sobre todo en matemáticas financieras para evaluar y analizar las carteras de inversiones, simulando los orígenes que de una forma incierta afectan a un valor, y determinando el valor promedio sobre el rango de resultados que nos está dando. La gran ventaja de estos métodos sobre otras técnicas es que pueden incrementarse las dimensiones si el problema aumenta, algo parecido a las redes neuronales que permiten añadir nuevos valores de entrada si vemos que queremos que el rango de aplicación de estas al problema sea mayor.

Disponemos de una variedad de ejemplos en algunas Webs [MONTECARLO-3], donde disponemos de modelos y documentación sobre predicción con modelos de Montecarlo. Aplicaciones físicas y ejemplos [MONTECARLO-3]. Predicción de modelos financieros [MONTECARLO-3].

Los modelos de Montecarlo son bastante aleatorios, computacionalmente pesados pero dan resultados tanto o más precisos que otros modelos incluyendo las redes neuronales. La gran ventaja sobre todo para nuestro problema es la posibilidad de incrementar las dimensiones si más adelante encontramos nuevas variables que incorporar a nuestro modelo de forma sencilla.

Predicción de n-Periodos en Adelante (n-Period-Ahead Forecasting)

Es un modelo que usa valores generados dinámicamente en base a la información de los meses anteriores, generamos una nueva predicción para el mes actual y realizamos la comparación con los resultados reales, testearemos a lo largo del tiempo lo preciso que es nuestro modelo dinámico. Estos valores dinámicos sirven para compensar el retraso que se produce entre la

predicción y los valores reales en el periodo de 1 a *n-1* [NPerAheadfore]. Una vez tenemos los valores dinámicos que representan el retraso intentamos usarlos para predecir el valor que debemos obtener en periodo n.

Este modelo es parecido al de las series temporales, de hecho es una variación, sirve para intentar predecir eventos que son repetitivos a lo largo del tiempo y que tienen pequeñas modificaciones. Sería posible usarlo para predecir el movimiento de las mareas, siempre que entre un periodo y otro no hubiera grandes modificaciones en los resultados que deben obtenerse. Para la toma de decisiones no es fácil usarlo. Por otro lado en zonas donde entre un mes y otro las diferencias sean muy grandes en dirección y velocidad, sería complicado usarlo ya que los parámetros dinámicos generados para describir los retrasos resultarían bastante aleatorios.

Predicción mediante modelos estocásticos

Se denomina estocástico a aquel sistema que funciona, sobre todo, por el azar. Suelen ser un conjunto de variables que evolucionan en función de otra variable (sistemas deterministas), generalmente el tiempo. Cada una de las variables aleatorias del proceso tiene su propia función de probabilidad y entre ellas, pueden estar correlacionadas o no. Cada variable o conjunto de variables sometidas a influencias o impactos aleatorios constituye un proceso estocástico. Se puede decir que todo proceso aleatorio tiene cierta indeterminación en su futura evolución descrita por distribuciones de probabilidad [ProcEsto].

Los modelos estocásticos sirven tienen diversas utilidades, por ejemplo.
Señales de telecomunicación
Señales biomédicas (electrocardiograma, encefalograma, etc.)
Señales sísmicas
El número de manchas solares año tras año
El índice de la bolsa segundo a segundo
La evolución de la población de un municipio año tras año
El tiempo de espera en cola de cada uno de los usuarios que van llegando a una ventanilla

Dado un espacio probabilístico *(Ω, F, P)*, un proceso (proceso aleatorio) con su estado en el espacio *X* es una colección *X-variables* aleatorias indexadas por el conjunto *T*, tiempo. Siendo *F* una colección.

$$\{F_t : t \in T\}$$

Donde cada F_t es una *X-valorada* variable aleatoria. Una modificación *G* del proceso *F*, es un proceso estocástico en el mismo espacio, con el mismo conjunto de parámetros *T* como esto:

$$P(F_t = G_t) = 1 \qquad \forall t \in T$$

La modificación es indistinguible si:

$$P(\forall t \in T, F_t = G_t) = 1$$

Los métodos estocásticos son muy usados en modelos de predicción que tengan que ver con variables temporales. Una de sus principales ventajas es que podríamos definir cada uno de

nuestros náufragos como una variable aleatoria, cada una independiente, y con su propia función de probabilidad, modelizar el comportamiento de la velocidad y dirección con dicha función, y usarlo para la predicción del movimiento de nuestros náufragos. Básicamente es lo mismo que hacemos con redes neuronales, la cuestión del coste computacional sería más alto pero el resultado debe de ser equivalente. Respecto a la toma de decisiones, los procesos estocásticos son n-dimensionales y permiten la correlación de las diferentes variables aleatorias, por lo tanto, se pueden modelizar la velocidad y la dirección del UAV dependiendo de las mismas variables aleatorias de entrada que usamos para nuestra red neuronal. Es conocido el uso de redes neuronales para modelizar la predicción en procesos estocásticos, por lo tanto, los resultados deberían ser equivalentes, tanto si usamos redes neuronales, como si usamos estos para nuestro sistema.

Predicción mediante ecuaciones probabilísticas, distribución de Gauss y de Cauchy

En teoría de la probabilidad y estadística, la distribución de probabilidad de una variable aleatoria es una función que asigna a cada suceso definido sobre la variable aleatoria la probabilidad de que dicho suceso ocurra. La distribución de probabilidad está definida sobre el conjunto de todos los eventos rango de valores de la variable aleatoria. Cuando la variable aleatoria toma valores en el conjunto de los números reales, la distribución de probabilidad está completamente especificada por la función de distribución, cuyo valor en cada real x es la probabilidad de que el valor de la variable aleatoria sea menor o igual que el de x.

Dada una variable aleatoria todos son puntos X, su función de distribución, *Fx(x)*, es

$$F_x(x) = P(X \leq x)$$

Por simplicidad, cuando no hay lugar a confusión, suele omitirse el subíndice X y se escribe, simplemente, *F(x)*.

Distribución normal o de Gauss.

En estadística y probabilidad se llama distribución normal, distribución de Gauss [DisProb-2], a una de las distribuciones de probabilidad de variable continua que con más frecuencia aparece en fenómenos reales. La gráfica de su función de densidad tiene una forma acampanada y es simétrica respecto de un determinado parámetro. Esta curva se conoce como campana de Gauss. La importancia de esta distribución radica en que permite modelar numerosos fenómenos naturales, sociales y psicológicos. Mientras que los mecanismos que subyacen a gran parte de este tipo de fenómenos son desconocidos, por la enorme cantidad de variables incontrolables que en ellos intervienen, el uso del modelo normal puede justificarse asumiendo que cada observación se obtiene como la suma de unas pocas causas independientes.
Función de densidad:

Se dice que una variable aleatoria continua X sigue una distribución normal de parámetros μ y σ y se denota $X \sim N(\mu, \sigma)$ si su función de densidad está dada por:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, \qquad x \in \mathbb{R}$$

Donde μ (mu) es la media y σ (sigma) es la desviación típica (σ^2 es la varianza).

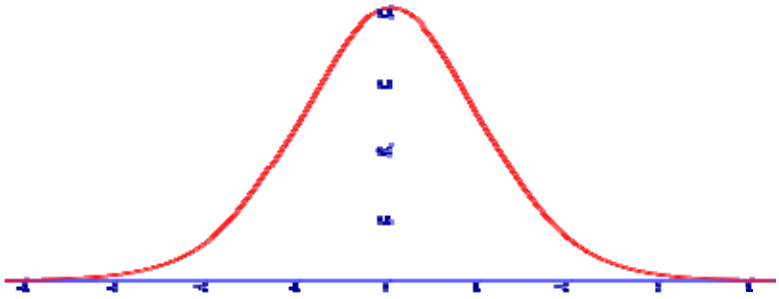


Ilustración 3: Distribución de Cauchy.

Distribución de Cauchy.

La distribución Cauchy-Lorentz es una distribución de probabilidad continua. Es conocida como la distribución de Cauchy [DisProb-1]. En general la distribución de Cauchy no tiene valor esperado ni varianza.

En estadística la distribución de Cauchy es una distribución de probabilidad continua cuya función de densidad es

$$f(x;x_0,\gamma) = \frac{1}{\pi\gamma\left[1+\left(\frac{x-x_0}{\gamma}\right)^2\right]} = \frac{1}{\pi}\left[\frac{\gamma}{(x-x_0)^2+\gamma^2}\right]$$

Donde *x0* es el parámetro de corrimiento que especifica la ubicación del pico de la distribución, y γ es el parámetro de escala que especifica el ancho medio al máximo medio.
En el caso especial donde $x0 = 0$ y $\gamma = 1$ es denominado la distribución estándar de Cauchy con la función de densidad de probabilidad.

$$f(x;0,1) = \frac{1}{\pi(1+x^2)}$$

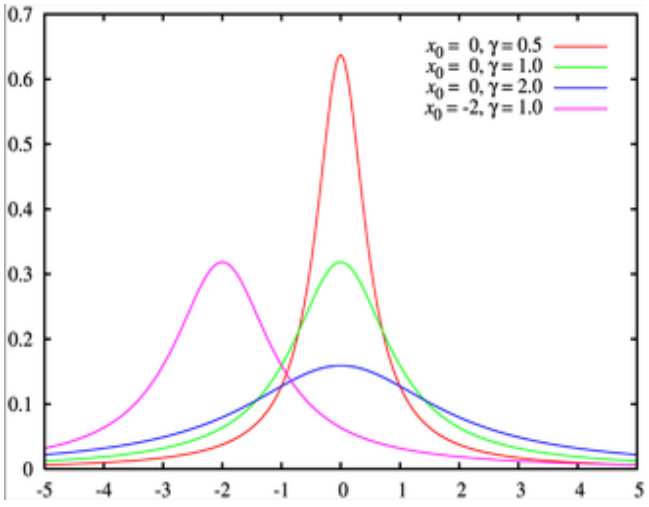


Ilustración 4: Función de densidad de probabilidad

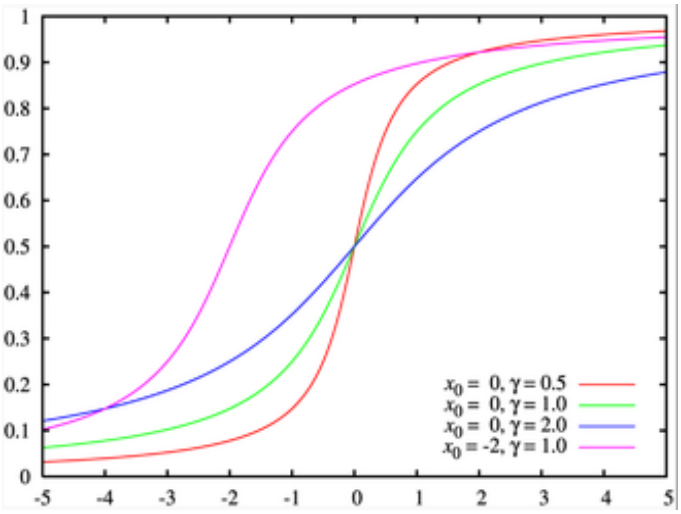


Ilustración 5: Función de distribución de probabilidad

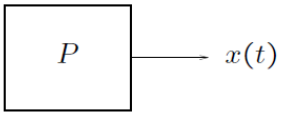
Este método basado en el cálculo de probabilidades, tiene como principales propiedades la estabilidad y la asunción de que los resultados están normalmente distribuidos, por un lado podemos suponer que para predecir el modelo de corrientes puede ser un buen método ya que si conocemos en un punto dado la velocidad y la dirección de la corriente, es muy probable que a una distancia suficientemente cercana dicha velocidad y dirección no varíen mucho. Por lo tanto a la hora de predecir el movimiento de las corrientes, puede ser un método preciso siempre que las condiciones no sean muy cambiantes, por otro lado usarlo para una toma de decisiones en un modelo tan dinámico como el que tenemos donde el UAV está en movimiento y las condiciones (variables de entrada con un amplio rango de posibilidades) son tan cambiantes no parece que pueda ser un modelo indicado para este tipo de problemas.

Predicción con series temporales

Es una secuencia de vectores (escalares) que dependen del tiempo [HorPag].

$$\{x(t_0), x(t_1), \ldots, x(t_{i-1}), x(t_i), x(t_{i+1}), \ldots\}$$

Esta es la salida de un proceso *P*, en el que estamos interesados.



Las series temporales tienen gran cantidad de usos por ejemplo: Promedios industriales del Dow-Jones [a], demanda de electricidad de una ciudad, temperatura en un edificio, estimación solar, predicción de vientos y corrientes marinas etc…

Estos fenómenos pueden ser continuos o discretos:

Fenómenos discretos: El promedio industrial del Dow-Jones [Hausdorff] diario.
Fenómenos continuos: *t* es un valor real y *x(t)* es una señal continua, para conseguir una serie {x[t]}.

$$\{x[t]\} = \{x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), \ldots\}$$

Las series temporales pueden predecir valores futuros de *x[t]*, clasificar series dentro de un conjunto de clases, descripción de las series según un conjunto de parámetros, transformarlas en otras.

En definitiva permiten describir y predecir fenómenos continuos a lo largo del tiempo. El problema que nos atañe a nosotros es la predicción, en este aspecto supongamos que la posición de los náufragos a lo largo del tiempo es un problema de series temporales. Nosotros deberemos estimar el valor de *x* (siendo *x* la posición de nuestros náufragos) en un tiempo futuro.

$$x[t+s] = f(x[t], x[t-1], \ldots)$$

A *s* se le llama el horizonte de predicción. *S=1* sería la siguiente secuencia de la serie y así podríamos ir prediciendo todos los valores. Para resolverlo, tendremos que asumir un modelo generalizado, para cada punto *x[ti]* en el pasado hay que entrenar nuestro modelo, de esta manera con cada ti de entrada conseguiremos la salida deseada. Ahora ejecutamos el modelo para predecir *x[t+s]* desde *{x[t] ....}*.

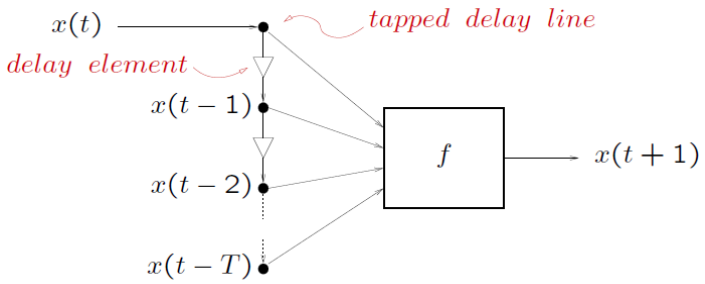


Ilustración 6: Uso del pasado para predecir el futuro.

Los modelos de series temporales son bastante interesantes ya que nos pueden servir para intentar predecir un evento futuro basándose en eventos pasados, pueden ser buenos tanto para la predicción del movimiento de los náufragos como para la toma de decisiones. El único problema que se puede ver es la cantidad de información que es necesario mantener y procesar a la hora de tomar una decisión, ya que cuánto más información tengamos, más probabilidades tendremos de que tanto nuestras predicciones como las decisiones sean precisas. Los modelos de redes neuronales son usados para modelizar series temporales, de hecho es una forma de optimizarla [HorPag] y [NNWTechPred], por lo tanto para este caso el uso de redes neuronales y series temporales sería equivalente.

Predicción mediante modelos dinámicos

Modelo de viento y oleaje:

El viento induce un oleaje que desplazará un objeto situado sobre la superficie mediante la ecuación:

$$u_w = V_w \cos\psi_w$$
$$v_w = V_w \sin\psi_w$$

Donde *Vw* representa la celeridad con la que se mueve el objeto, y que será una cierta fracción de la velocidad del viento a, por ejemplo 10 metros de altura, y ψw es el rumbo que lleva el viento y las olas. No tenemos información detallada de cómo son estos valores, por lo que debemos hacer suposiciones. La más sencilla es que ambos valores son constantes, lo que dará un comportamiento determinista. Para introducir un comportamiento estocástico podemos proceder de distintas formas. En [Fossen94 (2, pp. 137)], se utiliza un modelo para las fuerzas como un proceso de Markov de 2º orden junto con una deriva aleatoria. Introduce un cero en el origen. Este mismo modelo lo utiliza en la librería *gnc* desarrollada en Trodheim para modelar las perturbaciones de rumbo en el barco Mariner. Sin embargo, nosotros eliminamos el cero en el origen para generar el rumbo. El modelo usado es:

$$\psi_w(s) = \frac{2\lambda w_o \sigma K}{s^2 + 2\lambda w_o s + w_o^2} w_w + d_w$$

$$\dot{d}_w = n_w$$

Donde *ww* y *nw* son señales de ruido blanco gausiano, *dw* representa la deriva, y el valor inicial de su integrador nos daría el rumbo dominante, σ es la ganancia del espectro de las olas, *wo* es la frecuencia de pico del espectro, λ es el amortiguamiento relativo y *K* es la amplitud de la ola. Podemos usar los valores

$$\sigma = 0.5 \text{ m}, \text{ wo} = 1.2 \text{ rad/seg}, \lambda = 0.1, \text{ K} = 3.$$

El rumbo está dado en grados. El rumbo dominante del viento es el que fijemos: [0º, 360º]. El ruido blanco gaussiano es en ambos cados de media nula y varianza 10 (podemos modificarlo y ver distintos supuestos).

La velocidad del viento se puede dar como una señal aleatoria de ruido blanco con una media y unas varianzas adecuadas, o bien como un proceso de Markov de primer orden. Podemos considerar lo primero que es más fácil. Para el proceso de Markov procederíamos como veremos en las corrientes.

Modelo de corrientes:

Las corrientes tienen un modelo similar al del viento:

$$u_c = V_c \cos\psi_c$$
$$v_c = V_c \sin\psi_c$$

Donde *Vc* es la velocidad de la corriente y ψc es el rumbo. Al igual que con el viento podemos utilizar modelos muy sencillos de velocidad y rumbo constante, o bien con características aleatorias (función de distribución) dadas.

En [Fossen02 (2, pp.138)] se utiliza el siguiente modelo: la velocidad se considera un proceso de Markov de primer orden:

$$\dot{V}_c + \mu V_c = w_c$$

Donde *wc* es un ruido blanco gaussiano y $\mu \geq 0$ es una constante. El integrador debe tener un saturador para limitar la velocidad entre unos valores máximo y mínimo, por ejemplo:

$$V_{\min} \leq V_c \leq V_{\max}$$

Con Vmin=0.5 m/s, Vmax= 3 m/s.

Al rumbo ψc le podemos asociar también una dinámica. Podemos usar una función determinista de la posición, una función de distribución de probabilidad determinada, o asignarle un proceso de Markov de 1º o 2º orden. Quizás lo mejor sea usar una función determinista de la posición.

Los modelos dinámicos nos permiten definir mediante ecuaciones el movimiento de un fluido, son relativamente precisos y probablemente los mejores para la predicción del movimiento de los náufragos, el único problema que contienen es la imposibilidad de añadirles indecidibilidad al movimiento, modelizas un movimiento y siempre se comporta como dicen las ecuaciones no existe la posibilidad de que debido a las condiciones que existen en un momento dado el movimiento cambie y quizá no sea tan previsible, por esto último no tienen ninguna utilidad para la toma de decisiones respecto a la dirección de un camino dependiendo de unas condiciones no previstas.

Redes neuronales

Desde que se descubrieron, los usos y aplicaciones que se han dado a las redes neuronales han ido creciendo día a día, desde un simple reconocimiento de patrones sencillos (Perceptron 1958) pasando por el reconocimiento de caracteres (Hopfield), hasta usos más complicados en toma de decisiones y modelos de predicción (FeedForward).

Una de las razones principales por la que el uso de redes neuronales se ha ido extendiendo es por sus ventajas, aquí añadimos algunas de ellas:

Aprendizaje adaptativo: Capacidad de aprender a realizar tareas basadas en entrenamiento o en experiencia.
Auto-organización: La red puede crear su propia organización o su propia representación de la información después de una etapa de aprendizaje.
Tolerancia a fallos: Una destrucción parcial de la red conduce a una degradación de los resultados pero puede seguir funcionando.
Operación en tiempo real: Las redes (los computadores neuronales) pueden ser realizadas en paralelo.
Fácil inserción dentro de la tecnología existente: Una red puede ser rápidamente entrenada, comprobada, verificada y trasladada a hardware con bajo coste, lo que permite su inserción para tareas específicas.

Lo cual les permite ser de una gran utilidad en el desarrollo de productos comerciales de I+D+I. Y también en el desarrollo de aplicaciones para un número muy heterogéneo de mercados por ejemplo:

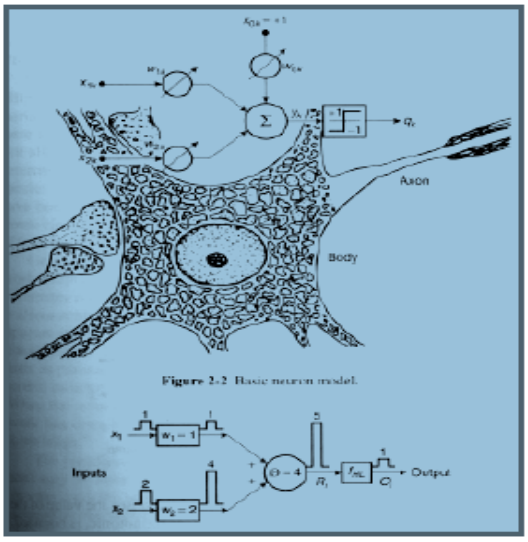
Aeroespacial. Mejorando el desarrollo de los pilotos automáticos, sistemas de control en tiempo real, UAV, control de fallos en componentes y pilotaje del avión.
Automoción. Sistemas de guía automáticos y análisis de fallos del vehículo.
Medio Ambiente. Desarrollando el análisis de patrones, previsión del tiempo, previsión solar.
Biología. Analizando el funcionamiento del cerebro humano.

Medicina. Analizadores de habla, monitorización en cirugía, diagnostico.
Finanzas, Business. Identificación de firmas, valoración de riesgos en créditos, bolsa, evolución de precios tanto de monedas como de manufacturados.
Militares. Clasificación de señales de radar, optimización de recursos

La base de las redes neuronales es la neurona, esta representa la unidad mínima implementada. Más tarde dependiendo del tipo de organización que implementemos para la agrupación de neuronas que tengamos se definirán uno u otros tipos de arquitecturas de neuronas. Cada neurona va unida a las neuronas que le preceden y a las que le suceden mediante una conexión de esta forma un conjunto de neuronas conforman una red y un modelo u arquitectura neuronal.

La neurona como hemos dicho es la unidad mínima dentro de nuestra red, y es importante conocer, como son sus componentes principales y como se construyen.

La neurona artificial es la encargada de recibir la entrada de sus compañeras y calcular una salida, siempre tanto las entradas como las salidas son numéricas.



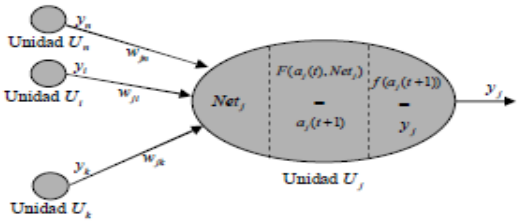


Ilustración 7: Neurona, definición y clasificación.

El primer componente es el ***estado de activación***, es el estado en el que se encuentra toda neurona, esto es en reposo o en excitación, si está en reposo no emitirá ninguna respuesta y si esta excitada emitirá como respuesta un valor hacia las neuronas que le suceden. Normalmente a cada uno de los estados se le asigna un valor que puede ser discreto, *0* para reposo o *1* para excitado o puede ser un valor continuo asociado a una función lineal, Sigmoidal, Tangencial o Gaussiana, depende de cómo queramos tratar la salida de nuestra neurona. Toda neurona recibe siempre el resultado de la suma de los estados por los pesos de las conexiones que hay entre ellas y realizan el tratamiento a partir de la *función de salida o transferencia* que tienen implementada. Esta ***función de transferencia o salida*** obtiene la salida de la neurona a partir de su estado de activación usando una función típica como puede ser una función discreta, sigmoidal, a veces lineal, gaussiana etc. De esta manera hemos generado la salida de nuestra neurona. Hay que decir de lo más común es que todas las neuronas pertenecientes a la misma red tengan la misma función de transferencia, al menos las que pertenecen a la misma capa.

Otra parte muy importante es la definición de la conexión entre neuronas. La ***conexión entre neuronas*** se denomina, ***sinapsis***. Cada conexión (sinapsis) entre la neurona *i* y la neurona *j* viene ponderada por un peso *wji*. Si el peso *wji* es positivo se considera una unión excitadora, si es

negativo inhibidora. Si es *0*, no existe conexión. Se suele utilizar una *matrizW* para representar todos los pesos o conexiones entre neuronas. La entrada de una neurona, *neti* (potencial sináptico) recibe la suma de todas las salidas ponderadas por el peso de cada sinapsis.

$$Net_j = \sum_{i=1}^{n} w_{ij} \cdot y_i \quad , \quad \text{Regla de propagación}$$

El siguiente componente de nuestra neurona es la ***Función o regla de activación*** que se encarga de combinar la entrada de la neurona (net_i) con el estado en el que se encuentra para obtener un nuevo estado de activación o excitación. En la mayoría de los casos suele ser la función identidad. Aunque hemos dicho que solemos usar el estado de activación anterior en la mayoría de las redes este es ignorado. Otro componente importante es el ***umbral de activación*** que es el desplazamiento que sufren nuestros valores para dicha neurona.

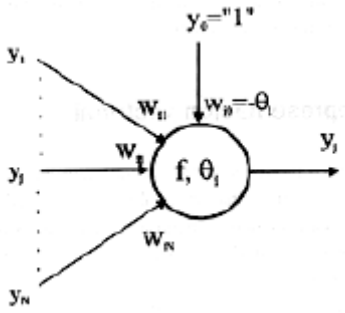
Función o regla de activación:

$$a_j(k+1) = F(Net_j, a_j(k))$$

Umbral de activación:

$$y_i(t+1) = f(net_i - \theta_i) = f(\sum_{j=1}^{N} w_{ji} \cdot y_j(t) - \theta_i)$$

Dibujo de una neurona y sus variables:



Las neuronas dentro de toda red están organizadas dentro de capas de esta manera podemos agruparlas y establecer conexiones con las neuronas de las capas anexas.

Existen dos grandes grupos de neuronas o redes neuronales, clasificadas en redes mono-capa y multi-capa. Las redes mono-capa establecen conexiones laterales dentro de las neuronas de la misma capa fundamentalmente son redes auto-asociativas que permiten reconstruir entradas incompletas a partir de unos patrones almacenados por ejemplo una red Hopfield. Las redes multi-capa agrupan las neuronas en varios niveles. Entre unos niveles y otros existen conexiones y esto nos permite establecer una clasificación dependiendo de dichas conexiones. Existen las redes con conexiones solo hacia adelante, ***feedforward*** como las que estamos usando en predicción de forma que la capa *i+1* no tiene relación con la capa anterior y las redes con conexiones hacia atrás ***feedback*** normalmente bi-capas y útiles en la asociación de información en las cuales si existen conexiones desde la capa *i+1* hacia la *i*.

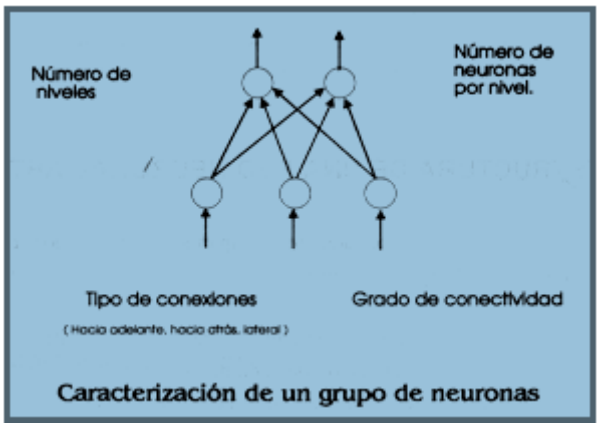


Ilustración 8: Caracterización de un grupo de neuronas.

La red neuronal tiene que aprender ya que es la única manera que le permite más adelante tomar decisiones dentro del proceso al cual la vamos a añadir. Cuando programamos una red neuronal tanto la función de transferencia como la regla de activación vienen dadas por defecto, entonces ¿Cómo aprende una red neuronal?, pues fundamentalmente modificando el valor de los pesos o conexiones entre las diferentes neuronas en respuesta a cada una de las informaciones de entrada. Se considera que la red ha aprendido cuando los pesos empiezan a ser prácticamente invariables.

$$\frac{dw_{ij}}{dt} = 0$$

El criterio que se usa para decidir cómo se modifican los valores de los pesos de las conexiones se denomina regla de aprendizaje. Existen dos tipos diferentes de aprendizaje el ***supervisado*** y el no ***supervisado***. También existe el aprendizaje online y offline que significa que la red puede mantener o no el desarrollo de su aprendizaje mientras está funcionando dentro de nuestro sistema real. Si no pudiera deberíamos entrenarla antes de implantarla en el sistema y definir un modelo que nos permita substituirla por la nueva red entrenada de la forma más rápida posible.

Las redes con ***aprendizaje supervisado*** realizan su entrenamiento mediante un agente externo que se encarga de supervisarlo, este agente se llama supervisor o maestro. El supervisor le provee a la red de las posibles respuestas para una entrada dada y si la respuesta de la red no es correcta entonces se modifican los pesos para conseguir la salida deseada, este entrenamiento se va continuando hasta que todas las salidas de la red son correctas. Existen varios tipos de diferentes de aprendizajes asociados a este tipo, el aprendizaje por corrección de error, el aprendizaje por refuerzo y el aprendizaje estocástico.

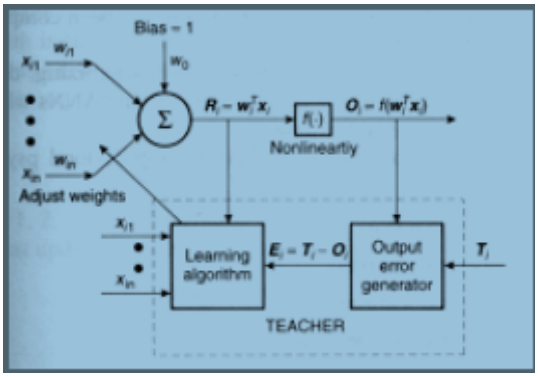


Ilustración 9: Modelo de entrenamiento de una red neuronal.

Por otro lado las redes con aprendizaje no supervisado tienen la ventaja que no requieren influencia externa para ajustar los pesos de las conexiones entre neuronas. La red no recibe de forma externa información acerca de si su respuesta es correcta o no a una entrada. Normalmente se usan para analizar el grado de familiaridad entre la información presentada y la mostrada en el pasado, establecer categorías de forma que podemos clasificar la información que nos llega, prototipado, creando un conjunto de clases que representan la información de entrada, también pueden mantener la información relevante de entrada, mapeo de características donde las neuronas de salida son un mapa de los datos de entrada ya que informaciones similares afectan a neuronas próximas entre sí. A esta categoría pertenecen las redes con aprendizaje hebbiano y aprendizaje competitivo y cooperativo.

Toda red neuronal debe primero construirse, luego entrenarse y finalmente podemos simular sus nuestras entradas de forma que la red neuronal intente predecir los posibles resultados. Realmente la red lo que hace es una interpolación por lo tanto este dato es importante tenerlo en cuenta a la hora de ver cuál es la parte de nuestro sistema inteligente que queremos que esté controlado por redes neuronales.

Backpropagation, Red Neuronal FeedForward

Es una red multicapa con retroalimentación. Su diseño básico es bastante simple aunque de gran potencia. El uso de redes neuronales para la predicción se está haciendo bastante popular debido a que da un enfoque y aproximación fácil para resolver problemas complejos. El algoritmo de retro-propagación o propagación hacia atrás permite una transmisión de los errores de la salida hacia atrás, hacia todas las neuronas de forma que se van ajustando los pesos de las conexiones con la idea de que el error disminuya. El algoritmo de aprendizaje está basado en la regla delta, ya que tenemos un conjunto de capas intermedias con funciones de activación continuas (lineales, sigmoidales o logarítmica) crecientes y derivables. De esta forma la función o superficie de error que tenemos asociada a la red se va modificando según vamos cambiando el tamaño de los pesos entre las conexiones neuronales, hasta que se consigue minimizar el error de la superficie.

A continuación veremos las posibilidades de construcción de una red neuronal de estas características que tenemos.

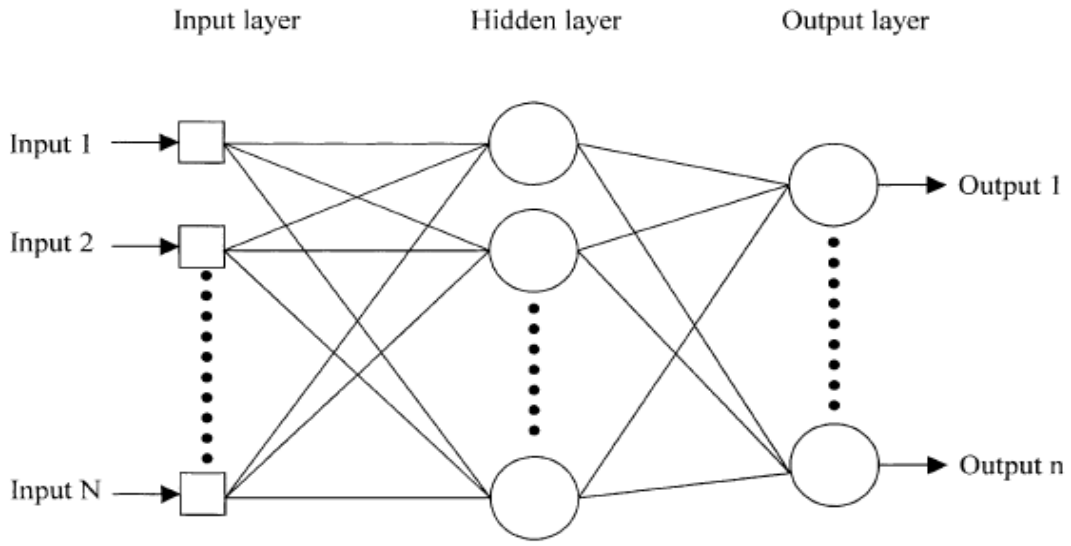


Ilustración 10: Típica red neuronal feedforward multi-capa.

El algoritmo base y más importante en una red neuronal del tipo Backpropagation es la regla delta generalizada. A continuación damos un esquema de dicho algoritmo.
Ajustamos los pesos *Ui*:

$$w_{ji}(t+1) = w_{ji}(t) + \lfloor \Delta w_{ji}(t+1) \rfloor$$
$$\Delta w_{ji}(t+1) = \alpha \cdot \delta_{pj} \cdot y_{pi} + \beta \cdot \Delta w_{ji}(t)$$

Calculamos la delta de error:

Si es neurona de salida:

$$\delta_{pj} = (d_{pj} - y_{pj}) \cdot f'(net_j)$$

Si no es neurona de salida:

$$\delta_{pj} = (\sum_k \delta_{pk} w_{kj}) \cdot f'(net_j)$$

El proceso debe de repetirse hasta minimizar el error.

$$E_p = \frac{1}{2} \sum_{k=1}^{M} \delta_{pk}^2$$

Podemos añadir un término momento que nos permitirá evitar las grandes fluctuaciones en la convergencia acelerando el proceso de aprendizaje. También se da la posibilidad de caer en un mínimo local ya que al minimizar una función de error con forma de superficie, que contiene muchos máximos y mínimos locales la probabilidad de caer en uno de estos es muy alta. Si ese mínimo local cumple el error puede ser suficiente, pero podemos modificar los métodos de búsqueda de forma que el salto sea mayor, de esa manera controlamos la convergencia para que se pueda saltar cada uno de los mínimos locales, el término momento nos puede ayudar a la hora de ajustar la fluctuación.

Debemos de tener cuidado a la hora de inicializar los pesos, ya que unos pesos cercanos a la solución final nos ayudaría en la fase de entrenamiento minimizando esta.

El número de capas y de neuronas ocultas para cada capa es muy importante, ya que aunque generalmente debería de ser suficiente con tres capas (entrada-oculta-salida), hay veces que dependiendo el problema la red va a aprender más si existen más capas ocultas. En cualquier caso es difícil dar una regla para decidir la cantidad de capas ocultas y de neuronas, aunque gran cantidad de neuronas y capas puede influir en la eficacia de aprendizaje y generalización de la red haciendo que esta se adapte rápidamente a los patrones que le hemos enseñado, pero pierda eficacia a la hora de reconocer otros que queríamos que reconociera. Hay que ensayar varias organizaciones hasta elegir la mejor, y procurar tomar el menor número de neuronas en la capa oculta ya que se genera menor carga computacional. En nuestro modelo de predicción y en el sistema inteligente fueron suficientes *10* y *8* neuronas en la capa oculta.

Diferentes tipos de funciones asociadas a una red neuronal de tipo feedforward.

Empezaremos estudiando las posibles **funciones de activación** que podemos asociar a nuestras neuronas. Las funciones de activación normalmente son continuas y derivables. Estas son:

Función logarítmica: **logsig**. *a = logsig(n)*

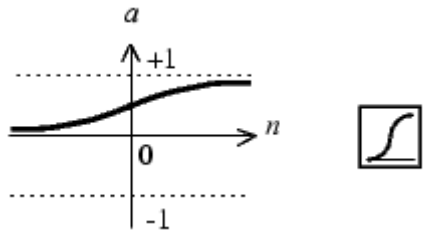


Ilustración 11: a = logsig(n)

Función sigmoidal: **tansig.** *a = tansig(n)*

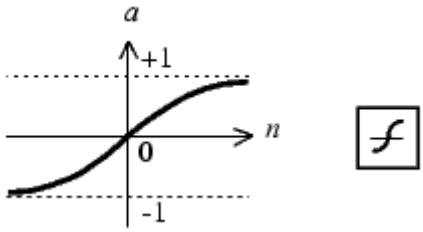


Ilustración 12: a = tansig(n)

Función lineal: **purelin.** *a = purelin(n)*

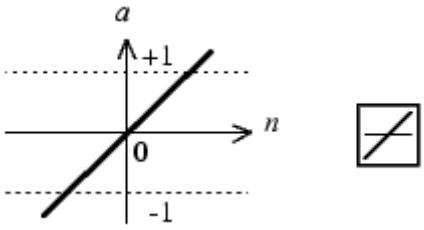


Ilustración 13: a = purelin(n)

Lo siguiente son las **funciones de aprendizaje** que podemos asociar a nuestra red. Estas son:

learngd: Función de aprendizaje de gradiente descendente en pesos y *bias*. Calcula los cambios sobre los pesos *dW* para una neurona desde la entrada *P* y error *E* y la tasa de aprendizaje l_r acorde con el gradiente descendente: *dw = lr*gW*.

learngdm: Función de aprendizaje de gradiente descendente con momento en pesos y *bias*. Calcula los cambios sobre los pesos *dW* para una neurona desde la entrada *P* y error *E* y la tasa de aprendizaje *LR* y momento constante *MC* acorde con el gradiente descendente con momento:

$$dW = m_c \cdot d \cdot dW_{prev} + (1 - m_c) \cdot l_r \cdot gW$$

Los cambios en los pesos previos *dWprev* son guardados y leídos desde el estado de aprendizaje *LS*.

Las siguientes son las funciones de cálculo del error:

- **msereg**: función de rendimiento sobre el cuadrado significativo del error con regularización.
- **mse**: función de rendimiento sobre el cuadrado significativo del error.
- **mae**: función de rendimiento sobre valor absoluto significativo del error.

Las siguientes, son las **rutinas de búsqueda** lineal

- **srchcha**: Unidimensional minimización usando el método de Charalambous.
- **srchbac**: Unidimensional minimización usando backtracking.
- **srchbre**: Unidimensional localización del interval usando el método de Brent.
- **srchgol**: Unidimensional minimización usando la búsqueda de la sección golden.
- **srchhyb**: Unidimensional minimización usando la búsqueda hibrida de la bisección cubica.

Continuamos con las diversas **funciones de entrenamiento**, cada una asociada a un algoritmo diferente.

traingd: Gradiente descendiente de la Backpropagation. Puede entrenar cualquier red neuronal que tenga pesos, entradas y funciones de transferencia derivables. Las variables son ajustadas acorde con el gradiente descendiente actualizando los pesos y ganancias variándolos según la dirección negativa del gradiente de la función de error.

$$dX = l_r \cdot \frac{dperf}{dX}$$

El entrenamiento acaba cuando alguna de las siguientes condiciones ha ocurrido:

- El máximo número de *epochs* es alcanzado.
- El tiempo es excedido.
- El rendimiento se ha minimizado hasta el objetivo.
- El gradiente cae por debajo del *mingrad*.
- Hay más fallos que los indicados en *max_fail*.

traingdm: Gradiente descendiente con momento de la Backpropagation. Cada variable es ajustada conforme al gradiente descendiente con momento.

$$dX = m_c \cdot dX_{prev} + l_r \cdot (1 - m_c) \cdot \frac{dperf}{dX}$$

La variable dX_{prev} representa los cambios previos de pesos y *bias*.
Con respecto a la finalización del entrenamiento este ocurre cuando ocurren las mismas causas que la función anterior.

traingda: Gradiente descendiente con tasa adaptativa de aprendizaje de la Backpropagation. Las variables son ajustadas conforme al gradiente descendiente.

$$dX = l_r \cdot \frac{dperf}{dX}$$

Para cada *epoch* si el rendimiento decrece hacia el objetivo, entonces el factor de aprendizaje es incrementado con el factor *lr_inc*. Si al contrario se incrementa más que el *max_perf_inc*, el factor de aprendizaje es ajustada por el factor *lr_dec* y se cambia. Con respecto a la finalización del entrenamiento este ocurre cuando ocurren las mismas causas que la función anterior.

trainrp: Resistencia Backpropagation. Cada variable es ajustada con lo siguiente.

$$dX = deltaX \cdot sign(gX)$$

Donde cada elemento de *deltaX* es inicializado a *delta0* y *gX* es el gradiente. En cada iteración los elementos son modificados de la forma: Si un elemento de *gX* cambia de signo en alguna iteración, el elemento correspondiente de *deltaX* es decrementado por *delta_dec*. Si alguno mantiene el mismo signo de una iteración a otra el elemento correspondiente de *deltaX* es incrementado por *delta_inc*. Con respecto a la finalización del entrenamiento este ocurre cuando ocurren las mismas causas que las funciones anteriores.

trainbfg: BFGS quasi-Newton Backpropagation. Backpropagation es usada para calcular las derivadas con respecto al peso y *bias* de *X*, ajustaremos las variables con la siguiente fórmula.

$$X = X + a \cdot dX$$

Donde *dX* es la dirección de búsqueda. El parámetro es seleccionado para minimizar el rendimiento a lo largo de la dirección de búsqueda. La función de búsqueda lineal *searchFcn* es usada para localizar el mínimo. La primera búsqueda es el la dirección negativa del gradiente. En sucesivas iteraciones la dirección de búsqueda será según la siguiente fórmula.

$$dX = -\frac{H}{gX}$$

Donde *gX* es el gradiente y *H* es la matriz Hessiana. Con respecto a la finalización del entrenamiento este ocurre cuando ocurren las mismas causas que las funciones anteriores.

trainlm: Levenberg-Marquardt Backpropagation. Cada variable es ajustada según el algoritmo de Levenberg-Marquardt.

$$jj = jX \cdot jX$$
$$je = jX \cdot E$$
$$dX = -(jj + I \cdot mu) / je$$

Donde *E* son los errores y *I* es la matriz identidad. El valor adaptativo de *mu* es incrementado por *mu_inc* hasta que se reduce el cambio de los resultados en función del rendimiento. Los cambios son introducidos a la red y *mu* es decrementado por *mu_dec*. El parámetro *mem_reduc* indica cómo se usa y la velocidad para calcular el Jacobiano *jX*, si *mem_reduc* es *1* la función de entrenamiento irá más rápido pero requerirá mas memoria incrementando *mem_reduc* a *2* cortamos la necesidad de memoria por dicho factor pero el entrenamiento irá más lento y cuanto más incrementemos el valor más lento irá. Con respecto a la finalización del entrenamiento este ocurre cuando ocurren las mismas causas que las funciones anteriores.

trainbr: Regularización Bayesiana de la Backpropagation. Puede entrenar cualquier red neuronal que tenga pesos, entradas y funciones de transferencia derivables. La regularización Bayesiana minimiza la combinación lineal de los cuadrados de los errores y pesos. Esto también modifica la combinación lineal, el resultado final del entrenamiento de la red tiene buenas cualidades de generalización. Se puede ver [Mackay] y [ForHag] para más detalles sobre regularización Bayesiana. Esta regularización Bayesiana toma lugar dentro del algoritmo de Levenberg-Marquardt. Backpropagation es usada para calcular el Jacobiano *jx* con respecto al peso al *bias* y las variables *x*. Cada variable es ajustada acorde con Levenberg-Marquardt

$$jj = jX \cdot jX$$
$$je = jX \cdot E$$
$$dX = -(jj + I \cdot mu) / je$$

Donde *E* son los errores y *I* es la identidad. El valor de mu adaptativo es incrementado por *mu_inc* hasta que los cambios mostrados reducen el valor del rendimiento. El cambio es realizado sobre la red y *mu* es decrementado por *mu_dec*. El parámetro *mem_reduc* indica como usamos a memoria y la velocidad para calcular el Jacobian *jx*. Si *mem_reduc* es *1* entonces el entrenamiento corre más rápido, pero requiere mucha memoria. Incrementando *mem_reduc* a *2* cortamos algo de memoria reduciéndola por el factor *2*, esto reduce el entrenamiento. Altos valores decrementarán las necesidades de memoria.
El entrenamiento para cuando alguna de estas condiciones ocurran:

- Se alcanza el máximo de *epoch*.
- La cantidad máxima de tiempo se excede.
- El rendimiento se ha minimizado hasta el objetivo.
- El gradiente cae debajo del *mingrad*.
- *mu* excede de *mu_max* o hay más fallos de los indicados por *max_fail*.

traincgf: Gradiente conjugado Backpropagation con actualizaciones Fletcher-Reeves.

traincgb: Gradiente conjugado Backpropagation con reinicio Powell-Beale.

traincgp: Gradiente conjugado Backpropagation con actualizaciones Polak-Ribiere.
trainoss: Secante de un paso de la Backpropagation.

Cuando construimos una red de tipo backpropagation (feedforward) debemos tener en cuenta los

siguientes elementos, el intervalo de los datos de entrada de la red, de esta manera la red puede ajustarse a los posibles parámetros de entrada. El número de neuronas y capas que queremos que tenga dicha red, lo normal es ajustarse a tres capas una de entrada, la cual tiene tantas neuronas como parámetros de entrada tiene nuestra red, la capa de salida que debe de tener tantas neuronas como parámetros de salida tiene nuestra red, y por último las capas ocult depende de la estructura de lo que queremos ajustar. Lo siguiente es la función de entrenamiento con la que deberemos de tener cuidado ya que algunas pueden tardar mucho tiempo de entrenamiento y facilidad para caer en mínimos locales. Lo siguiente es la función de aprendizaje y por último la función de cálculo del error. Como ejemplo podríamos poner el siguiente [MatlabNNT], con tres capas y las tres con funciones lineales.

```
Net = newff (I,[10,10,2],{'purelin','purelin','purelin'},'trainbr','learngd', 'mse')
```

Lo siguiente que tendremos que hacer es entrenar la red para eso usaremos funciones de entrenamiento estándar, que nos permiten entrenar, dados unos datos de entrada y sus posibles salidas asociadas, nuestra red. Un ejemplo podría ser el siguiente [MatlabNNT]. El entrenamiento como vemos es dirigido.

```
Net = train(net,x,z)
```

Una vez hemos visto las posibles parametrizaciones que existen para una red neuronal de tipo Feedforward-Backpropagation es interesante ver con cuales de estas parametrizaciones hemos desarrollado y entrenado las redes neuronales de nuestro sistema inteligente de ayuda al rescate. En nuestro sistema tenemos dos redes neuronales, la primera es la encargada de simular el componente final de vientos y corrientes o Red de decisión (netMalla), que nos permite simular el desplazamiento de uno o varios náufragos a lo largo del tiempo que se encuentren en el mar. Por otro lado está la red de toma de decisiones (netDirecciones), encargada de mostrarnos la dirección y velocidad que debemos de tomar en una posición dada con la información que tengamos sobre los náufragos hasta este momento.

Red de predicción (netMalla): red que simula el movimiento (velocidad y dirección) de los náufragos.

Parametrización de la red:

- Tipo de red: **feedforward-backpropagation**.
- Rangos de entrada: recogidos del input.
- Función de entrenamiento: **trainbr**. Regularización Bayesiana de la Backpropagation
- Función de aprendizaje: **learngdm**. Función de aprendizaje de gradiente descendente con momento en pesos y bias
- Función de rendimiento: **mse**.
- Número de capas: 2.
  - Capa 1. 4 neuronas con una función de transferencia **tansig**.
  - Capa 2. 2 neuronas con una función de transferencia **purelin**.

Red de decisión (netDirecciones): red que simula el movimiento (velocidad y dirección) que debe de seguir el UAV.

Parametrización de la red:

- Tipo de red: **feedforward-backpropagation**.
- Rangos de entrada: recogidos del input.

- Función de entrenamiento: **trainbr**. Regularización Bayesiana de la Backpropagation
- Función de aprendizaje: **learngdm**. Función de aprendizaje de gradiente descendente con momento en pesos y bias
- Función de rendimiento: **msereg**.
- Número de capas: 2.
  Capa 1. 8 neuronas con una función de transferencia **tansig**.
  Capa 2. 2 neuronas con una función de transferencia **purelin**.

Nuestra red de predicción

En nuestro proyecto, tanto la red de predicción como la de decisión son redes feedforward con algoritmo de aprendizaje *trainbr*, que es el algoritmo basado en el Bayesian Regulation. Este tipo de entrenamiento actualiza los pesos y el *bias* acorde con la optimización de Levenberg-Marquardt, minimiza la combinación de los cuadrados de los errores y lo pesos determinando la combinación más correcta de cara a producir una red que generalice lo más correctamente posible.

A continuación veremos cómo hemos obtenido la red de predicción para indicar la zona a la que se debe dirigir el avión. La red de decisión, encargada de guiar al avión una vez hemos llegado a la zona predicha de encuentro con los náufragos, se verá en el siguiente capítulo.

Aproximación inicial

Suponemos que el viento tiene velocidades y rumbos normales, con medias y varianzas dadas a priori. Primero lanzaremos un conjunto de náufragos que representan nuestra masa, iremos simulando con la red neuronal el movimiento de los náufragos, debemos decidir si usamos una red neuronal diferente para cada naufrago o la misma para todos. Es evidente que este modelo puede parecer bastante impreciso a priori, ya que estamos intentando que nuestra red, dada la posición de un náufrago, nos diga la siguiente posición que ocupará.

En una solución a este problema, aplicando redes neuronales para simular el movimiento de los náufragos, al principio pensamos usar una red neuronal para simular el movimiento de cada uno de los náufragos, pero si usamos una red por cada uno de los náufragos, entonces estaríamos obligados a crear una matriz de entrenamiento para cada una de las redes neuronales, perjudicando seriamente el rendimiento del sistema. Por otro lado, si realizamos el entrenamiento de las redes en tiempo de ejecución del sistema, debido a la gran cantidad de tiempo necesaria para entrenar una red neuronal, ralentizaría mucho la ejecución del programa. Por lo tanto, se decidió usar una sola red neuronal de las entrenadas, para todos los náufragos, descartando aquellas cuyo gradiente fuera muy grande y por lo tanto no se pudieran aproximar a la solución final de forma eficaz.

Intentamos crear un modelo para nuestra red de forma que esta sea capaz de predecir, dados los primeros movimientos de los náufragos, el resto de posiciones a lo largo del tiempo. Después de usar dicha red neuronal para generar a lo largo del tiempo nuestra mancha de náufragos e intentar simular su desplazamiento por la corriente marina, obtuvimos el resultado de la ilustración inferior.

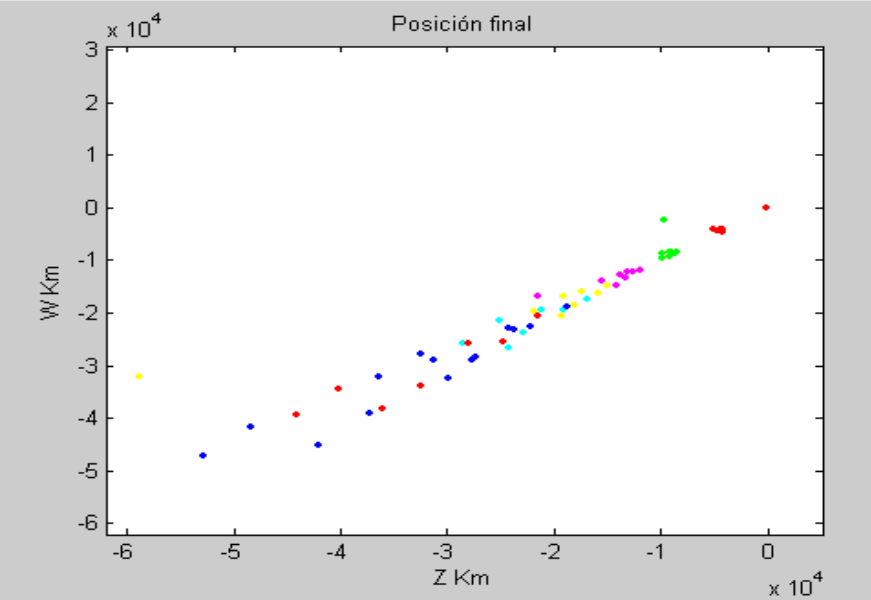


Ilustración 14: Desplazamiento de los náufragos dado por la red neuronal.

Esta aproximación resulto tener un gran número de inconvenientes. Primero era que la red alcanzaba en un gran número de casos un mínimo local, por lo tanto veíamos que aunque mantenía la dirección le era casi imposible mantener la velocidad de los náufragos y a veces estos avanzaban demasiado deprisa con respecto a la media de las mareas calculadas. Algunas veces la red neuronal no caía en un mínimo local y conseguía una precisión bastante aceptable generándonos de forma bastante precisa la dirección de los náufragos. Como vemos en la Ilustración 2, no se puede precisar la situación exacta de la mancha generada por los náufragos, todos los náufragos del mismo color, en un instante determinado, que deben de generar una sola mancha. Además con esta aproximación cualquier error en la estima era acumulativo con lo que la red neuronal tiende a dispersarlos cada vez más y a cometer un error muy grande. Se supone que la ilustración 2, generada por nuestra red neuronal, debe de parecerse lo más posible a la ilustración 1 mostrada en el capítulo 1, como se ve, esto no sucede.

Pronto nos dimos cuenta que por mucho que buscáramos que la red neuronal nos diera una aproximación de la posición final de los náufragos a partir de unos parámetros iniciales, esto no iba a ocurrir ya que esta no era capaz de interpolar mas allá de la situación para la que la habíamos entrenado. Esto nos hizo replantearnos la solución, la cual aportaremos en el próximo apartado, a partir de unas preguntas. ¿Qué dependencia tiene la red de los datos de entrenamiento utilizados?, ¿qué necesita la red neuronal para realizar una buena predicción? y quizá la más importante, ¿qué cambios hay que realizar en el modelo inicial para que la red haga la predicción que nosotros deseamos?

La red neuronal crea un modelo continuo para el viento y marea en cualquier instante, tiempo y posición a partir del modelo discreto, lo cual no nos asegura que para datos discretos para los cuales no ha sido entrenada la red y fuera del espacio creado por esta, los resultados puedan ser óptimos. Por esa razón, es probable que no consiguiéramos un buen resultado con el modelo que hemos diseñado, la red no puede predecir nada que esté fuera del modelo que ha creado asociado a los elementos discretos con los que la hemos entrenado. Esto nos lleva a plantear otro modelo diferente, un modelo que nos permita realizar la simulación del movimiento y dispersión de la mancha generada por los náufragos a lo largo del tiempo de una forma realizable mediante modelos de redes neuronales. En el siguiente apartado veremos la solución a nuestro problema.

Solución adoptada

Después de analizar los datos y la bibliografía sobre predicción con redes neuronales, vemos que el modelo que estábamos siguiendo no era el ideal para este tipo de predicción, quizá estábamos planteando el problema desde un punto de vista, del cual no íbamos a obtener un resultado positivo. Como elemento de investigación resulta positivo ver las dificultades a la hora de crear un modelo y ver como no se consigue que funcione de forma correcta y estable, pero desde el punto de vista práctico no lo es tanto.

En ese momento vemos que con el modelo que estamos construyendo, no es posible continuar trabajando. Aquí cambiamos nuestra posición y definimos un nuevo modelo que nos permita predecir el movimiento de los náufragos.

Lo primero es definir un área de trabajo, un espacio o región de donde dispongamos una gran cantidad de datos acerca de las corrientes y de la velocidad del viento, a ser posible en diferentes épocas del año. Esto nos permite establecer nuestra componente final de vientos y corrientes asociada al mapa de vientos y corrientes en dicha zona a lo largo de un tiempo determinado. Una vez tenemos construido nuestro mapa pasamos a ver cómo podemos integrar esto dentro de nuestra red neuronal y si el planteamiento que estamos introduciendo aquí tiene alguna posibilidad de éxito.

Basándonos en diferentes modelos predictivos [KaloNeoPas], estos nos dan una aproximación a la solución que debemos de realizar. Buceando en la bibliografía descubrimos que ya se han usado las redes neuronales para la predicción de mareas y de viento además de para predicciones solares [HontAgui], [AtsuDorv] y [OzanSenkal]. Esto es posible debido a que las redes neuronales son buenas para interpolar y completar conjuntos de datos, lo que quiere decir que si le damos nuestro mapa a la red neuronal esta debería de aprendérselo y ser capaz de que cuando le demos otras fechas y otros puntos, intentar dar una estimación de lo que podría pasar en dicho punto para esa fecha. Lo primero es delimitar los parámetros sobre los que deseamos crear nuestro modelo. Debemos de obtener unos datos simulados o experimentales que nos permita definir un área suficientemente grande sobre la cual medir en diferentes épocas del año la fuerza del viento y de las mareas, esto es, la dirección del viento y de la marea, así como la velocidad de estas. Con el componente final de vientos y corrientes calculado con estos datos, entrenaremos nuestra red neuronal. Realmente lo que estamos haciendo es para n días del año medir la fuerza del viento y de la marea así como su velocidad creando una malla equivalente de x por y kilómetros cuadrados [GomCarr07]. Con este modelo podemos entrenar nuestra red teniendo en cuenta la posición del náufrago y el instante de tiempo en el que se encuentra. Usaremos una red del tipo feedforward (backpropagation), que generará internamente un modelo desconocido, pero que será capaz de predecir en diferentes instantes de tiempo y posiciones de entrada, la velocidad y dirección asociadas a dicha posición. El resultado será parecido a una interpolación entre los diferentes puntos y tiempos que se le dio en el momento del entrenamiento y a la vez distinto ya que no conocemos como la red ha construido la distribución de los pesos [HilMart95]. Podemos decir que la red ha creado un modelo continuo para el viento y marea en cualquier instante, tiempo y posición a partir de un modelo discreto apropiado a la zona sobre la que deseamos realizar la predicción.

Cuando disponemos de nuestra red neuronal entrenada con el mapa del componente de mareas y vientos disponibles, la red está entrenada para predecir el movimiento de los náufragos por el espacio que ha reconocido, que es el que le hemos dado como entrenamiento. Ahora esta red forma parte de nuestro sistema inteligente como un componente muy importante, ya que es el

que va a ser capaz de predecir el movimiento de los náufragos y nos permitirá dirigir nuestro UAV. Vemos que el modelo que hemos elegido, ya sea por la bibliografía existente o bien por nuestros razonamientos acerca del comportamiento de las redes neuronales, parece que va por buen camino. De hecho las pruebas realizadas así lo demuestran.

Esta solución para predecir el movimiento de los náufragos es extensible, podemos añadir parámetros de comportamiento nuevos, entrenando la red con los nuevos parámetros, incluso cambiar la interpretación de los parámetros de entrada de la red que modeliza el componente final de vientos y corrientes que usamos, por ejemplo, cambiar las coordenadas (x,y) donde está situado el náufrago, y por lo tanto el lugar en el que se está intentando predecir su velocidad y dirección, por la latitud y la longitud. Después del entrenamiento el comportamiento de la red debería ser el mismo. Otra de sus características es la portabilidad ya que después de tener la red entrenada podemos integrarla dentro de cualquier sistema inteligente que se adapte a los parámetros de entrada y espere la misma salida que la red neuronal ofrece.

Conocidas la posición actual (X', Y'), la velocidad (v) y la dirección (a, en radianes) del náufrago, calcular la siguiente posición de nuestro náufrago es bastante sencillo, bastará con aplicar la siguiente fórmula.

$$X = X' + dt \cdot v \cdot \cos(a)$$
$$Y = Y' + dt \cdot v \cdot \sin(a)$$

Donde dt es el tiempo entre dos cálculos de posición independientes.

Con esto tenemos indicada la zona en la que estarán los náufragos cuando llegue el UAV, es decir, el lugar donde debe dirigirse el avión para encontrar los náufragos.

CAPÍTULO 3. MODELO DE DISPERSIÓN DE NÁUFRAGOS

Suponemos un conjunto de objetos distribuidos aleatoriamente en una superficie perfectamente delimitada del océano, por ejemplo, treinta kilómetros cuadrados. Allí cada elemento se ve sometido a las fuerzas del viento (oleaje) y de las corrientes [Fossen02], lo que hace que se vayan dispersando los objetos. Transcurrido un cierto tiempo se inicia la búsqueda de los objetos por un UAV [UAV]. Este posee un elemento detector que permite la localización de objetos que se encuentren dentro de una circunferencia centrada en el UAV de, por ejemplo, un kilómetro de radio. También permite determinar cómo se va desplazando el objeto. ¿Qué trayectoria debe seguir el UAV para encontrar el máximo número de objetos en el menor tiempo posible? ¿Cómo debe modificar su trayectoria para mejorar la búsqueda si encuentra en su campo de visión un nuevo objeto y mide su posición y trayectoria? Para responder a estas preguntas es necesario que el UAV disponga de un predictor que le indique donde debería encontrar a los náufragos en un instante determinado. Se debe tener un tiempo estimado de llegada al punto sugerido y en ese instante los náufragos se habrán dispersado, por lo que no es allí donde se debe ir, sino donde indique el predictor. Para entrenar la red de momento, y como fase inicial, se utilizará un modelo de oleaje y vientos sencillo que permita realizar simulaciones y predicciones de la Red Neuronal. De este modo se puede analizar la eficacia del predictor. Posteriormente se podrán incluir otras variables como tipo de naufragio, etc.

Viento y oleaje. Corrientes marinas

El **viento** es el movimiento del aire que está presente en la atmósfera, especialmente, en la troposfera, producido por causas naturales. Se trata de un fenómeno meteorológico. La causa de los vientos está en los movimientos de rotación y de traslación de la tierra.

Una **corriente oceánica o marina** es un movimiento de traslación, continuado y permanente de una masa de agua determinada de los océanos y en menor grado, de los mares más extensos. Estas corrientes tienen multitud de causas, principalmente, el movimiento de rotación terrestre (que actúa de manera distinta y hasta opuesta en el fondo del océano y en la superficie) y por los vientos constantes o planetarios, así como la configuración de las costas y la ubicación de los continentes.

El **oleaje** del mar está compuesto por las olas, que son ondas mecánicas, es decir perturbaciones de un medio material, superficiales, que son aquellas que se propagan por la frontera entre dos medios materiales. En este caso se trata del límite entre la atmósfera y el océano.

El **viento y las corrientes** inducen un **oleaje** que desplazará un objeto situado sobre la superficie y que podemos modelar mediante la ecuación:

$$u_w = V_w \cos\psi_w$$
$$v_w = V_w \sin\psi_w$$

Donde *Vw* representa la celeridad con la que se mueve el objeto, y que será una cierta fracción de la velocidad del viento, y ψw es el rumbo que lleva el viento y las olas. Todos estos resultados se pueden simular mediante una red neuronal.

Componente final de vientos y corrientes es el componente generado con la velocidad y la dirección del viento y la corriente marina, esto se ha hecho para conseguir simplificar el modelo que le damos a conocer a la red neuronal que usamos para la predicción.

Conocimiento a priori

El conocimiento a priori de que partimos se puede modificar según el grado de facilidad o dificultad que veamos en el problema. En cualquier caso, podemos tener una idea de las velocidades de viento y corrientes y también del rumbo de éstos. También podemos tener una idea aproximada de varianzas. Esto lo estiman los organismos meteorológicos y marítimos correspondientes.

Lo más interesante puede ser el aprendizaje que se va realizando según se va encontrando a los náufragos, y la estrategia para intentar encontrar a los náufragos que se van dispersando a partir de los primeros náufragos que se han encontrado. Durante todo el proceso mantenemos la simulación de los náufragos, de esta forma podemos comparar la posición actual de estos con la posición real de los náufragos que vamos encontrando.

Evidentemente las redes neuronales son entrenadas por nosotros, y éstas intentan adaptarse al conocimiento que le hemos dado, si nosotros definimos un modelo de búsqueda, la red neuronal intentará aproximarse a dicho modelo interpolando los datos que no conoce. Con respecto a esto, tenemos que tener cuidado ya que estamos entrenando a la red con un algoritmo que representa, de una forma más o menos aproximada, la interpretación de la solución del problema que nosotros creemos que debe de tener. Una forma que tenemos de no caer en esta trampa, es no construir un algoritmo que de forma implícita tenga una solución del problema, si no que debemos darle un mínimo de datos a nuestra red neuronal, y dejar que nuestra red cree el resto de soluciones probables, de esta manera la red construirá un modelo completamente independiente.

UAV

El modelo de UAV (Avión no tripulado) es un modelo ideal en el que conocemos la velocidad y se mueve siguiendo las trayectorias que se le fijen, sin importarnos su dinámica. De modo que conocemos el tiempo que le lleva ir de un punto a otro, y el área que ha recorrido en su trayectoria. Lo único que tenemos que hacer es construir un modelo que permita compenetrarse de la forma más homogénea posible al cerebro del UAV.

Aunque el modelo se han realizado suponiendo un UAV eso no significa que sea la única posibilidad, también se podría aplicar el sistema a un barco no tripulado, este podría ser ampliado con el modelo que presentamos en este trabajo.

Pero, qué es exactamente un UAV. Es un avión que vuela sin una tripulación humana dentro del avión [UAV]. Su mayor uso está en aplicaciones militares, los UAV son ante todo reusables, el vehículo tiene la capacidad de ser controlado y sostenido por una maquina o de forma remota. Existe una gran variedad de UAV, según su tamaño, forma, configuración y características. Normalmente los UAV son teledirigidos pero poco a poco han ido incrementando su autonomía. Esto hace que podamos dividir los UAV en dos variedades, unos controlados remotamente y

otros autónomos basados en una pre-programación del vuelo o usando complejos sistemas dinámicos y expertos dentro de los cuales añadiremos nuestro trozo del sistema.

Objetivo y señuelo – Usados para parecer un objetivo a la artillería y aviación de forma que simula un avión enemigo o un misil.

- Reconocimiento – proporcionar información del campo de batalla al servicio de inteligencia.
- Combate - proporcionando una capacidad de ataque de misiones de alto riesgo.
- Logística - UAV diseñado específicamente para la carga y operaciones de logística.
- Investigación y desarrollo - se utiliza para seguir desarrollando las tecnologías UAV que deben integrarse en el campo del desarrollo de nuevos aviones.
- Civil y Comercial - UAV diseñado específicamente para aplicaciones civiles y comerciales.

UAVs, poco a poco están incrementando su rol en la búsqueda y rescate dentro de los USA. Quedó demostrado su uso de forma exitosa durante los huracanes del 2008 en Louisiana y Texas. Por ejemplo los Predators [PlanRescate-2], operaron sobre los 18000-29000 pies por encima del mar realizando tareas de búsqueda y rescate además de evaluaciones de los daños producidos. Usaron sensores ópticos, cámaras infra-rojas y radar. El Predator dispone de sensores que permiten analizar las imágenes a pesar de las condiciones meteorológicas, ya sea lluvia o niebla, día o noche. La calidad de las imágenes, permiten ampliar la capacidad de búsqueda y rescate del UAV.

Nosotros construiremos un pequeño trozo del cerebro del UAV, el sistema de ayuda al rescate, donde una red neuronal va a ser la encargada de tomar las decisiones sobre la dirección a tomar en cada momento y probablemente la velocidad a la que tenemos que situar nuestro UAV. Con la simulación del movimiento de los náufragos podremos ver el movimiento que realizaría el UAV siguiendo los consejos de nuestro sistema. Prevemos que dicho movimiento será oscilante entorno a la posición que el sistema prevé que deberían estar los náufragos.

Aquí tenemos que destacar que los sistemas UAV son bastante complejos, por lo tanto nuestro sistema necesitaría de pequeños cambios para crear un interface que le permitiera hablar con el resto de los sistemas que controlan el UAV. Un estudio de los estándares actuales de integración entre estos sistemas, nos permitiría realizar las mejoras necesarias en nuestro sistema inteligente de forma que, más adelante, fuera sencillo de instalar en el UAV.

Rescate de los náufragos

Como se ha mencionado antes, además de predecir la situación de los náufragos, debemos ser capaces de aconsejar cuales serían las decisiones, respecto a la velocidad y dirección, que el sistema considera que son las mejores para conseguir nuestro objetivo. Lo que tenemos es una mancha, conjunto disperso de náufragos, que a lo largo del tiempo se irá haciendo más grande, según va aumentando la dispersión de los náufragos, a la que hay que seguir, prediciendo su dirección y velocidad, para más adelante intentar recorrerla maximizando las probabilidades de localización de cada uno de los náufragos. Nuestro objetivo final es obtener una curva más o menos inteligente, que debe de seguir el UAV y mediante la cual todos, o al menos la mayoría de los náufragos, serán localizados.

Ya hemos visto en el apartado anterior cómo usando una red neuronal hemos construido un modelo que nos permite predecir el movimiento de los náufragos, por lo tanto podemos más o menos estimar el lugar donde la trayectoria de nuestro UAV enlazará con la trayectoria de los náufragos. En dicho lugar es donde empezará a funcionar la segunda parte del sistema que describimos a continuación.

Una vez se ha conocido la dirección hacia la que se dirige la masa de náufragos, la cantidad de estos puede ser definida al azar, pongamos un número por ejemplo *N*, nos permite calcular en todo momento la media de todos los puntos conociendo la dirección y la velocidad de esta. De hecho esta media es la que deseamos que nuestro UAV persiga ya que alrededor de ella se van a encontrar los náufragos que tenemos que recoger.

En todo momento podemos, gracias a los datos que va dando la red neuronal, conocer la media posicional de los náufragos. También podemos calcular los auto-valores (tamaño de los ejes) y los auto-vectores (dirección de los ejes) definidos por la posición actual de los náufragos, esto nos da una elipse que debe de contener el espacio por el que se mueven en cada instante nuestros náufragos.

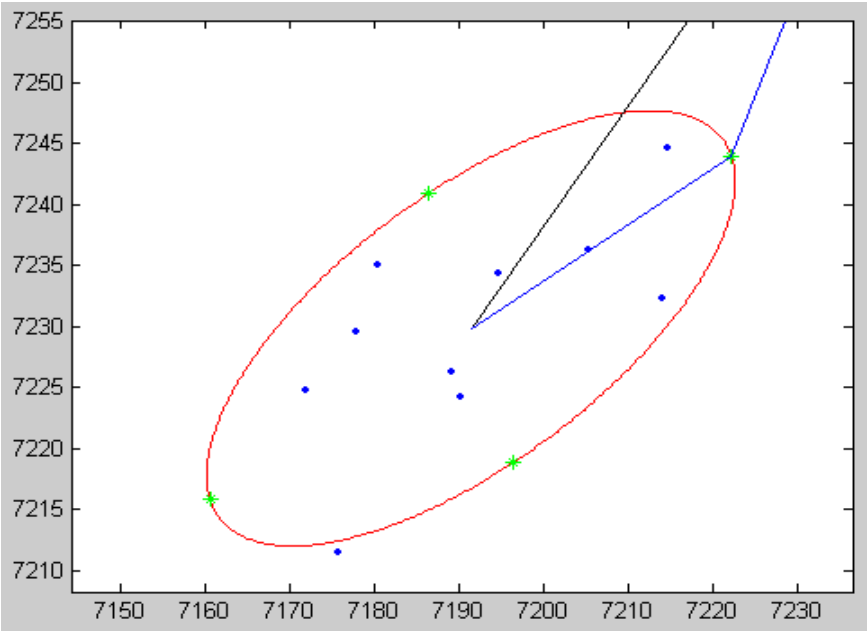


Ilustración 15: Elipse y posición de los náufragos.

Rojo: la elipse, el avión (+) y los puntos que no han sido descubiertos.

Azul: los puntos descubiertos y la curva que sigue el avión.

Negro: la dirección que ha seguido la media de los puntos desde el naufragio hasta la intercepción del avión.

La mejor forma de comprender el gráfico es ver exactamente qué es lo que estamos haciendo dentro de nuestra búsqueda de la solución, para ello debemos de pensar como si viéramos el problema desde una perspectiva más amplia. Lo primero que tenemos que tener en cuenta es que, una vez se ha producido el naufragio, pasará un tiempo *t* hasta que nos den el aviso y nuestro UAV pueda despegar y ponerse en camino. La dirección que debe de tomar el UAV es la dirección que nuestra simulación obtenga, para ello la solución optima es calcular donde se cruzan, la trayectoria de los náufragos, calculada por la red neuronal que realiza la predicción, y la trayectoria que va a seguir el UAV. La forma de calcular dicha trayectoria es, dada la velocidad máxima a la que el UAV puede desplazarse, calculamos la distancia a las posiciones,

en función del tiempo, que ocuparían los náufragos y ver si el UAV puede alcanzar dicha posición a una velocidad inferior (pero suficientemente cercana) a la máxima que pueda alcanzar. De esta forma conoceremos el punto aproximado de intersección entre ambas trayectorias, la de los náufragos y la del UAV, posteriormente el UAV iría en línea recta hacia dicha posición. Una vez conocida esa posición, calculamos mediante los autovectores, la elipse que forman estos, donde la probabilidad de encontrar dispersos a los náufragos simulados se maximiza. Una vez calculada dicha elipse lo que tenemos que hacer es recorrerla en la dirección del eje mayor ya que este nos da mayor probabilidad de localizar náufragos. La simulación se acaba una vez se hayan localizado todos, o en su defecto el tiempo transcurrido, que es un parámetro del sistema, hace imposible localizarlos a todos.

En una segunda fase, y una vez hemos encontrado el centro de la elipse donde con mucha probabilidad se encuentren nuestros náufragos, si no los hemos encontrado a todos, debemos de establecer un sistema que nos permita encontrarlos lo antes posible dentro de un margen de tiempo adecuado. Más allá de ese margen, las probabilidades serían cercanas a cero. También debemos de establer una distancia de visualización, otro parámetro del sistema, ya que si en un momento dado pasamos lo suficientemente cerca de un náufrago, este habrá que marcarlo como localizado.

Por otro lado queda el modelo de desplazamiento del avión. Utilizaremos un modelo sencillo para desplazar el UAV en la dirección hacia la que van los náufragos predicho por la red neuronal anteriormente descrita, para eso no es necesario una red neuronal.

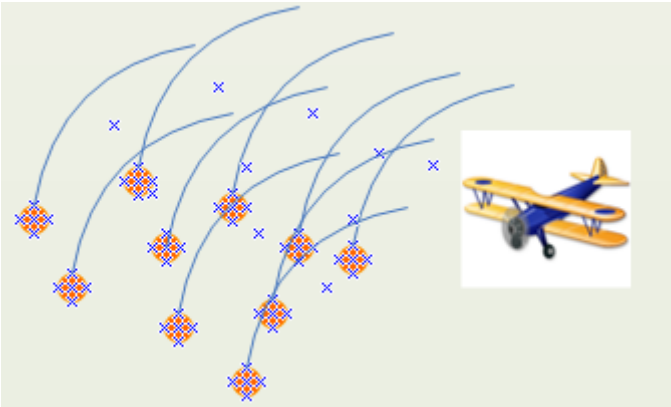


Ilustración 16: Gráfico de persecución del avión a los náufragos.

El siguiente paso a aclarar es como se puede tomar la decisión de las acciones que hay que realizar, [ExpertSystem] y [DecisionSupport], en otras palabras hay que dotar de un cerebro al UAV que le permita definir una trayectoria que persiga o al menos trate de perseguir a los náufragos hasta recogerlos a todos. Después de evaluar varias posibilidades vimos que otra de las aplicaciones de las redes neuronales [ManGut] y en concreto la red neuronal feedforward, es la toma de decisiones por lo tanto lo aplicaremos al cerebro de nuestro UAV.

Una vez estamos en la posición final generada por la media de los náufragos debemos de realizar un análisis de los datos de los que disponemos en dicho punto. Estos son la posición del naufragio, los náufragos descubiertos, nuestra posición respecto a los náufragos descubiertos, la media de los náufragos descubiertos y su dirección. Con estos datos debemos de construir una red neuronal que nos permita ser entrenada de forma independiente y que haga que el UAV con los datos disponibles en cada momento tome la decisión más adecuada a la hora de perseguir, localizar y recoger a los náufragos restantes que faltan. Añadiremos, a los datos anteriores, el

porcentaje de náufragos descubiertos por nuestro sistema, este porcentaje será un número entre 0 y 1. El porcentaje es un valor que deberemos estimar, ya que a priori es posible que no conozcamos exactamente el número total de náufragos que hay, desde el principio, en el mar.

A continuación, para nuestra red neuronal de decisión tomaremos cuatro parámetros de entrada y uno o dos de salida, los parámetros de salida serían la dirección en la que se tiene que mover el UAV y la velocidad normalizada. La velocidad plantea un problema de complejidad a la hora de entrenar nuestra red pero nos permite una mayor precisión en la trayectoria calculada. La entrada de la red neuronal es menos discutible, estos serán, la dirección de los náufragos que hemos descubierto y sobre los que continuaremos la simulación ya que son una fuente importante de datos, esta dirección se calcula siempre como el vector creado entre la media anteriormente descubierta y la actual, como dato inicial podemos usar el punto origen del naufragio. Como segundo parámetro podemos usar la dirección de la media de los náufragos, calculada como la dirección entre nuestra posición y la posición actual de la media de los náufragos. Esta dirección es importante ya que en el fondo es hacia donde tendríamos tendencia a dirigirnos de forma natural ya que es la zona donde creeríamos que tenemos más probabilidad de encontrar a nuestros náufragos. El siguiente parámetro de entrada sería la **distancia a la media** que nos encontramos normalizada por la distancia que podemos visualizar, esto es porque si estamos muy cerca de la media nuestra tendencia cambiaría e intentaríamos alejarnos de ella para encontrar los náufragos restantes, por esa razón usamos la distancia de visión del UAV como elemento de normalización, de esa forma sabemos si estamos muy cerca de la media $n<1$, lejos $n<2$ o muy lejos $n>2$. Por último el parámetro elegido es la **cantidad de náufragos** que llevamos encontrados expresados como porcentaje, tanto por 1, dato entre 0 y 1, la razón de usar dicho parámetro se debe a que la veracidad, credibilidad, de la media viene expresada por dicho parámetro. Si llevamos encontrados muchos náufragos la credibilidad de esta aumentará y nos hará ser más propensos a seguirla que si no somos capaces de encontrar muchos náufragos, en cuyo caso, deberemos de suponer que nuestro sistema ha cometido bastantes errores y deberemos de acabar la búsqueda y revisar el sistema.

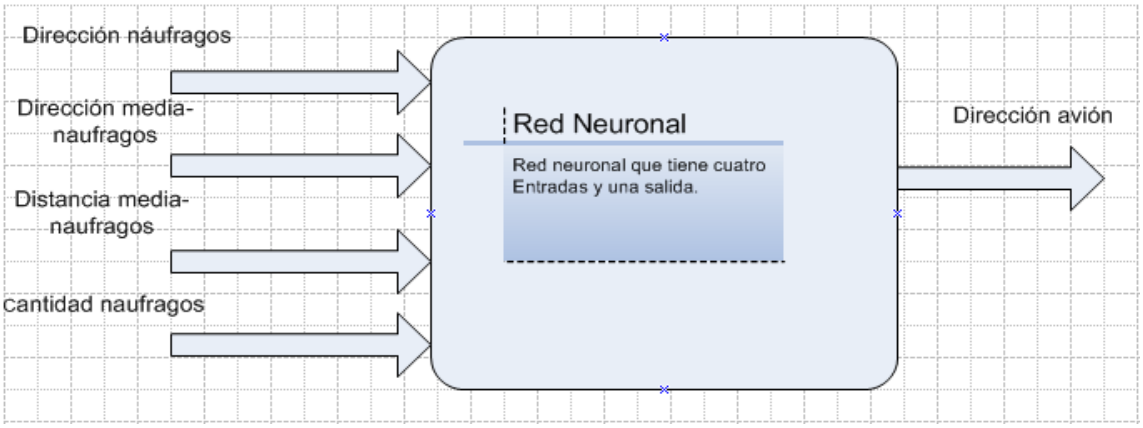


Ilustración 17: Esquema de la red neuronal de decisión.

La construcción de dicho sistema inteligente no parece ser excesivamente complicada tomando como base que el cerebro de este es nuestra red neuronal, esta debe recibir un conjunto de parámetros conocidos como son, la cantidad de náufragos encontrados hasta ese momento, la dirección hacia la que se encuentran dichos náufragos, la dirección seguida por los náufragos hasta ese momento y la distancia hasta dichos náufragos. Con estos datos, deberíamos ser capaces de obtener la velocidad y la dirección a la que tiene que dirigirse nuestro UAV. Una vez conocida la dirección y la velocidad, lo que haremos es que nuestro sistema guiará al UAV y

analizará el recorrido para comprobar si en dicho recorrido se encuentran más náufragos que recoger, añadiendo dicha información al sistema.

Debemos tener claro que aunque se llamen redes neuronales artificiales y sean útiles en la toma de decisiones en ningún caso son capaces de realizar acciones para las que no hayan sido entrenadas, ellas se auto-organizan solas, conforme a unos parámetros de entrada y de salida creando dependencias entre ellos y calculando una serie de pesos que les sirven para predecir una salida para la que en teoría no han sido entrenadas. En el fondo internamente crean una organización, que les permite asociar un valor matemáticamente calculable, a cualquier entrada que le demos, pero esta salida será más o menos precisa según lo cerca que esté del entrenamiento a la que la hayamos sometido.

Una vez tenemos dentro del cerebro de nuestro UAV la red neuronal que va a ser la encargada de tomar las decisiones sobre la dirección a tomar en cada momento y la velocidad a la que tenemos que situar nuestro UAV, el siguiente paso será continuar la simulación de nuestros náufragos y ver el movimiento que va realizando el UAV alrededor estos, vemos que será un movimiento oscilatorio y de persecución alrededor de la media de los náufragos descubiertos. Si el UAV no encontrara ningún náufrago debe tomar una decisión, esta es seguir la línea, que en teoría, según la red neuronal de predicción, deben de seguir nuestros náufragos.

Iremos por cada periodo de muestreo, dt, que seleccionemos, calculando la distancia de los náufragos y viendo si se pueden añadir más náufragos a los ya rescatados, así hasta que todos los náufragos sean rescatados o consideremos que el tiempo transcurrido es ya demasiado grande para que nuestras predicciones tengan la precisión mínima deseada. En este punto se nos plantea un dilema ya que tenemos que decidir el momento en el cual nuestras predicciones dejan de ser válidas y no merece la pena que el UAV continúe la búsqueda ya que con los datos que tiene solo puede deducir resultados erróneos. Nosotros lo hemos limitado para esta simulación en ocho horas ya que a partir de ese momento lo más probable es que el error se haya hecho demasiado grande.

Finalmente dibujamos los náufragos y la trayectoria seguida tanto por estos como por el UAV en su seguimiento y recogida de estos, en la simulación del sistema veremos el movimiento que realiza el UAV intentando localizar a los náufragos.

CAPÍTULO 4. SISTEMA INTELIGENTE DE AYUDA AL RESCATE

En los capítulos anteriores hemos ido introduciendo la estructura de los componentes que van a ser usados en nuestro sistema, aquí desarrollaremos, utilizando la información proporcionada sobre la estructura de los componentes usados, el sistema inteligente de ayuda al rescate completo. Veremos cómo interactúan todos los componentes en conjunto, para predecir el movimiento de los náufragos, e ir aconsejando a nuestro UAV acerca de las direcciones y velocidades que considera las más adecuadas en cada momento, para la localización de los náufragos.

Sistema inteligente de ayuda

Un sistema inteligente es un software que intenta ofrecer una respuesta o una aproximación a un problema que normalmente sería resuelto con la aportación de uno o más expertos en la materia. Los sistemas inteligentes se desarrollan para dar respuesta a un problema específico dentro de un dominio determinado. Existen una gran cantidad de métodos para simular la respuesta deseada. Normalmente lo primero es recoger la experiencia del experto y creamos una base de conocimiento con dicha experiencia, por otro lado el sistema debe de recoger dicha experiencia y aprender a tratarla para dar las respuestas adecuadas. A veces en un sistema inteligente podemos añadir un sistema de aprendizaje, ya que es la única forma de que el sistema consiga resolver o simular problemas del mundo real y dar con la respuesta asociada a dicho problema.

Existen algunos puntos que nos permiten detectar que un programa representa a un sistema inteligente.

1. La secuencia de pasos para alcanzar la conclusión es dinámicamente sintetizada con cada nuevo caso, no debe de ser explícitamente programada.
2. Los sistemas inteligentes pueden procesar múltiples soluciones para un problema, lo cual permite más de una línea de razonamiento.
3. La solución aportada va acompañada de un conocimiento específico más que una técnica específica, como si fuera hecho por un ser humano.

Nuestro sistema inteligente ha sido diseñado para predecir la posición de los náufragos y tomar decisiones acerca del camino que nuestro UAV debe de seguir para localizar dichos náufragos. Intentaremos describir nuestro sistema, veremos cómo puede ayudarnos en el rescate de los náufragos y también veremos que nuestro sistema puede ser considerado un sistema inteligente.

Una visión general sobre nuestro sistema inteligente nos permite distinguir dos componentes principales, el primero es el sistema de predicción del movimiento de los náufragos, que nos permite de una forma genérica simular el desplazamiento que realizarían las corrientes marinas y el viento sobre nuestros posibles náufragos. Este módulo del sistema debe ser entrenado con una base de datos de corrientes marinas de la zona sobre la que queremos que nuestro sistema inteligente realice las predicciones. Con dichas predicciones nuestro UAV se puede dirigir directamente al encuentro de los náufragos, es decir a la zona donde estarán los náufragos en un instante dado. Por otro lado está la segunda parte del sistema, en este caso nuestro UAV ya ha

alcanzado a nuestros náufragos y en base a los datos recogidos, la dirección observada de los náufragos y la distancia a éstos, podemos ir calculando en cada instante de tiempo la dirección que nuestro UAV debe de tomar para ir detectando a los náufragos que le falta por descubrir. Cada vez que descubrimos un nuevo náufrago este nos provee de una información importante con la que podemos ir actualizando nuestra base de conocimiento acerca de la dirección hacia la que se van dirigiendo los náufragos.

Desde el punto de vista de un sistema inteligente intentaremos describir el funcionamiento de los dos componentes que modelan el sistema.

El primero de los componentes es el encargado de predecir la dirección de los náufragos, el componente se ha construido de forma independiente de la zona de actuación en la que estemos. Antes de empezar a usar nuestro sistema inteligente, habrá que entrenar la red neuronal que nos va a servir para predecir el movimiento de los náufragos. La red debe de ser entrenada con un plano de corrientes marinas y vientos de la zona a tratar. A la red le asociaremos diversas posiciones del mapa y épocas del año, como retorno le asociaremos la velocidad y la dirección media del componente generado del viento y corrientes marinas en dicho punto para la época del año que estemos tratando. Una vez la red está entrenada, esta será capaz de interpolar y darnos, dada una posición del mapa y un momento del año, cuál será la velocidad y dirección que debería tomar nuestro náufrago en dicho punto, con eso nos será fácil calcular la siguiente posición que este tomará en el mapa con el que estamos trabajando. Esto nos permite configurar una trayectoria a lo largo del tiempo. Ya hemos visto que el primer parámetro de nuestro sistema inteligente es la posición aproximada del naufragio y el instante de tiempo en el que éste se ha producido, por otro lado el segundo parámetro importante es la velocidad máxima y posición actual de nuestro UAV, por razones de cálculo nuestro sistema funciona con una velocidad constante inferior a la velocidad máxima que le hayamos introducido.

Es importante recalcar que nuestro UAV no conoce la posición actual de los náufragos, la red neuronal nos dará la trayectoria aproximada de los náufragos, Con dicha trayectoria aproximada y la posición de nuestro UAV es fácil construir una trayectoria de encuentro entre ambos. Eso es exactamente lo que hace nuestro sistema definiendo una elipse construida con los auto-valores, que son los ejes, y los auto-vectores que son las direcciones de estos, con la cual intentamos definir la trayectoria de entrada del UAV sobre los náufragos para localizar a estos. Debemos decir que dicha posición donde se cruzan las trayectorias de ambos son simuladas y puede ocurrir que en dicho punto no exista ningún náufrago.

Una vez hemos construido el sistema que permite llevar nuestro UAV de la forma más óptima y rápida a una trayectoria de intercepción con los náufragos, dibujamos esto con el sistema y preparamos los datos para la entrada del siguiente sistema, el sistema de toma de decisiones del UAV.

Lo primero que tenemos que hacer antes de nada es entrenar la red neuronal que es el núcleo de nuestro sistema de toma de decisiones. Esta red neuronal ha sido entrenada con un algoritmo que usa la información disponible en ese momento, esto es, la posición de la media de los náufragos descubiertos, la posición del UAV, la distancia a dicha media normalizada por la distancia de visión del UAV (distancia a la que dado un náufrago el UAV es capaz de visualizarlo), y la cantidad de náufragos descubiertos en tanto por 1. La red neuronal devuelve la dirección hacia donde se tiene que dirigir el UAV, y a qué velocidad debe de ir, para maximizar las probabilidades de localizar más náufragos.

El algoritmo usado para entrenar la red es independiente de esta, aquí hemos usado uno bastante sencillo que se basa en tomar siempre una dirección media entre la que marca el UAV y la media calculada de los náufragos y la dirección marcada entre dos medias temporalmente consecutivas de los náufragos descubiertos hasta ese momento.

Una vez entrenada nuestra red disponemos de un sistema que tomara la decisión de la dirección a toma del UAV e irá incorporando la información nueva que se vaya acumulando a lo largo del tiempo. Podremos ver según avanza el tiempo, como el sistema va descubriendo, incorporando y tratando dicha información al sistema.

Ya hemos visto que el sistema incorpora dos redes neuronales para predecir el recorrido de los náufragos y decidir tanto la velocidad como la dirección que debe de seguir nuestro UAV con la idea de maximizar la trayectoria que recorre, pero, hemos dicho que es un sistema inteligente, como integra el sistema estas dos redes neuronales, y que parámetros usa para conseguir calcular los datos necesarios para que el sistema funcione.

Lo primero que podremos parametrizar del sistema es el número de náufragos, este dato debe de ser introducido para que la red neuronal de predicción nos calcule la trayectoria de cada uno de nuestros náufragos de forma independiente. Por otro lado necesitamos un sistema que de forma iterativa sea capaz de calcular la posición exacta de encuentro entre nuestros náufragos virtuales, llamados así ya que son los calculados por el sistema, y nuestro UAV. Una vez calculada esta posición, debemos de llevar el UAV a la máxima velocidad que pueda, a dicha posición, ya que no debe de tardar más tiempo en llegar que el calculado por el sistema inteligente.

Llegados a este punto, entra en funcionamiento la parte del sistema inteligente que gestiona la red neuronal, aquí serán importante un conjunto de parámetros, aparte de los específicos de entrada de la red neuronal, como son, la velocidad media de los náufragos (velocidad media real) que usaremos como parámetro para calcular la velocidad a la que debe de ir el UAV en cada tramo, esto es debido a que el UAV usará esta velocidad como velocidad base, la red neuronal no nos da una velocidad, si no que nos da un valor normalizado que debemos de multiplicar por la velocidad media de los náufragos para conocer la velocidad que debe de llevar nuestro UAV.

Con este parámetro de la velocidad y usando las ecuaciones del movimiento podemos calcular en base al tiempo el desplazamiento que debe de realizar en línea recta nuestro UAV. En este momento acaba de salir otro parámetro importante, el tiempo, el tiempo de cálculo es introducido manualmente en el sistema, es importante no usar un intervalo temporal muy corto, ya que el sistema tomaría muchas decisiones innecesarias, ni tampoco excesivamente grande ya que es posible que esto impidiera que el sistema tomara las decisiones adecuadas.

Los dos últimos factores importantes que tenemos son la distancia de visión y el radio de maniobrabilidad. La distancia de visión es la base con la cual conocemos si un náufrago ha pasado lo suficientemente cerca del UAV para ser descubierto, por lo tanto este será añadido a la lista de náufragos descubiertos y se usará para los cálculos posteriores de la media y porcentaje, esto permite introducir la información descubierta sobre dicho náufrago dentro de nuestra red neuronal de decisión. El radio de maniobrabilidad es un parámetro parecido a la distancia de visión que nos permite normalizar la distancia a la media, de forma la red neuronal puede tomar decisiones acerca de las trayectorias que debe realizar alrededor de los náufragos para localizarlos.

Como hemos visto una vez entrenado el sistema para una zona determinada, avisado de un posible naufragio, y definida la zona inicial de actuación, el sistema se pone en marcha calculando primero como llevar al UAV a la zona más lógica de actuación, y luego realizando una progresiva persecución de los náufragos hasta localizarlos a todos. Como todos los sistemas que tratan de predecir algo, puede fallar, para ese caso o para el caso de que nuestro UAV pueda quedarse sin combustible, existe la posibilidad de darle un número de horas máximo de búsqueda a partir del cual el UAV volverá, aunque la información acumulada hasta el momento puede ser pasada a otro UAV que continúe la búsqueda.

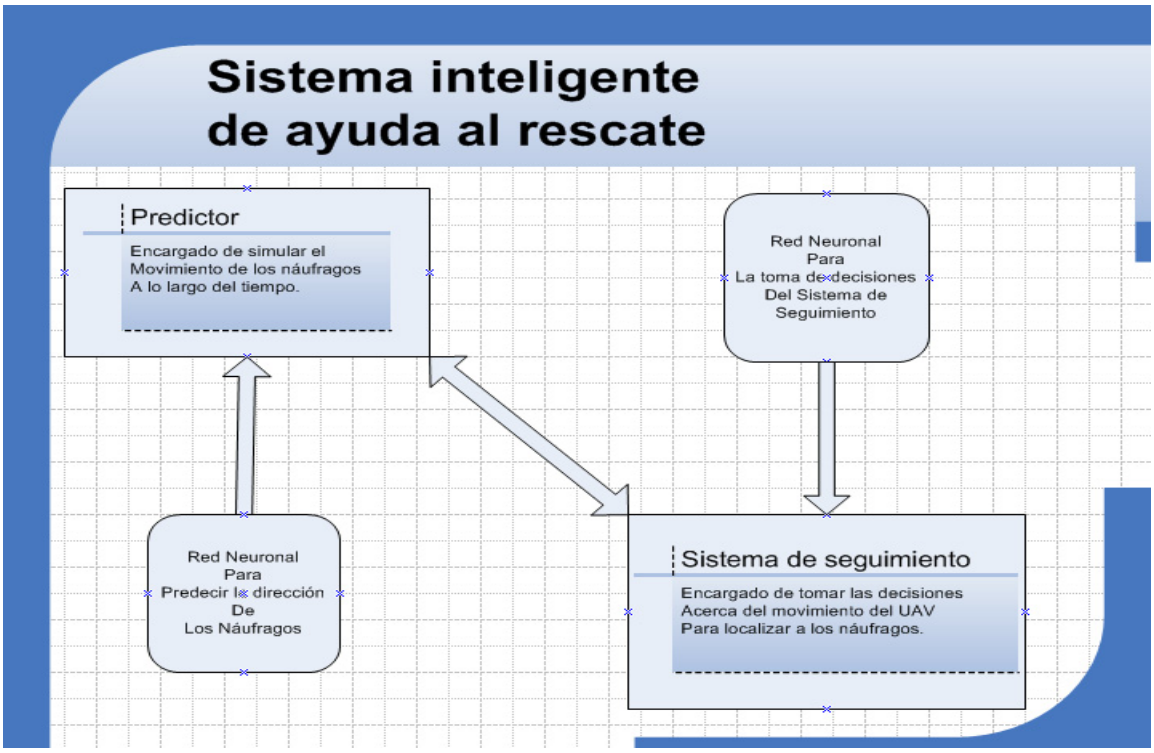


Ilustración 18: Sistema inteligente de ayuda al rescate.

Capacidad de predicción y margen de error

Es complicado estimar las probabilidades, a priori de un sistema inteligente, ya que nunca existe una seguridad de que este vaya a funcionar correctamente e incluso de que fuera de la simulación, en el mundo real, el sistema se vaya a comportar exactamente igual que en el entorno en el que lo simulamos. En la realidad se pueden llegar a dar casos que no teníamos controlados dentro de nuestra simulación.

Lo que si podemos es ver que situaciones pueden perjudicar la posibilidad de encontrar una buena solución a nuestro sistema. La primera son los errores de entrenamiento de nuestra red neuronal, lo ponemos aquí ya que estos existen aunque normalmente lo único que pueden hacer es desviar un poco el rumbo de los náufragos o del UAV cuando este está persiguiendo a los náufragos, si la red está bien entrenada y el error es mínimo, a lo largo de un tiempo corto en horas no debería de perjudicar el desarrollo del rescate. Se supone que la red de toma de decisiones debe de tener en cuenta esta situación. Por otro lado están los errores detectados por el usuario, estos son, los errores producidos por ser un sistema real, por ejemplo errores en la detección del naufragio y errores debido a cambio meteorológicos no normales en dicha época

del año. Estos últimos muy difíciles de predecir y que a lo mejor necesiten de un factor corrector o de una red neuronal entrenada específicamente para este caso. Por otro lado quedan casuísticas que probablemente no han sido previstas dentro del sistema, para estos casos existe una incertidumbre de lo que el sistema puede hacer y solo lo sabremos cuando se den, se supone que es sistema estará algún tiempo de prueba para poder detectar y depurar dichos casos.

La red neuronal cuando está entrenándose genera dos tipos de errores, el primero es debido a los propios datos. A la red neuronal solo se le presentan un conjunto pequeño de la solución y se le pide que a partir de ahí tome decisiones o más o menos interpole el resto de los datos, esto puede producir errores ya que en el fondo se le está pidiendo que adivine una solución conociendo una parte de esta. Por otro lado cuando entrenamos la red intentamos minimizar la superficie de error pero es casi imposible hacerla cero, de hecho la red neuronal te devuelve junto con la red entrenada los errores cometidos dentro de esta, según la función de entrenamiento usada.

[net,tr,Y,**E**,Pf,Af] = train(net,P,T,Pi,Ai,VV,TV)

Donde ***E*** es el **error de entrenamiento** de la red.

Errores detectados por el usuario externo. El sistema no puede ser retroalimentado en la primera fase, el sistema ha sido entrenado con un conjunto de datos más o menos estadísticamente recogidos y con esos va a funcionar, si por ejemplo hace un día con mas viento de lo estadísticamente normal y al comparar las cartas con las que se entreno la red estas no se parecen, es muy probable que sea imposible que el sistema sea capaz de aproximar de forma correcta la fuerza y dirección de los náufragos. En este momento es necesario añadir un control que le permita al usuario desactivar el sistema, ya que los consejos que se puedan llegar a dar, serán incorrectos y pueden llevar a tomar decisiones erróneas. El sistema presenta la evolución de los náufragos que se van descubriendo, esto nos permite comparar las medias con las dadas por nuestra simulación y nos permite ver en tiempo real si el error que se está produciendo es más o menos conflictivo.

Errores instrumentales. Estos siempre se pueden producir ya que es posible que el naufragio no se produjera en el sitio indicado, o que cuando pasemos por encima de un náufrago el sistema no lo vea y por lo tanto no lo rescatemos. Esto haría perder una información muy valiosa al sistema, que podría haber sido usada para reafirmar el sistema y localizar al resto de náufragos. Estos errores son bastante complicados de detectar, ya que, el no haber localizado ningún náufrago en el los primeros momentos después de la intersección de las trayectorias, no implica que el sistema esté basándose en un conjunto de datos erróneos de inicialización. Es posible que los náufragos se hayan, como ocurre en los ejemplos presentados, dispersado más de lo inicialmente calculado, y por lo tanto las correcciones que la red neuronal de decisión realice, es posible, que permitan localizar náufragos en un espacio de tiempo más o menos cercano.

Al final cuando estamos usando un sistema inteligente, nunca estamos seguros de si este está respondiendo exactamente a las preguntas que se le hacen de la forma como se le ha entrenado, incluso es posible que tengamos una respuesta, de alguna manera conseguida, para algo en lo que el sistema no ha sido entrenado. En el mundo real siempre hay que tratar con un cierto margen de **incertidumbre** ya que no todo el mundo para un problema daría la misma respuesta y nuestro sistema inteligente aunque ha sido entrenado con una lógica, es posible que no de las respuestas o soluciones que nosotros esperaríamos o para las que creíamos haberle entrenado. La incertidumbre es un parámetro a tener en cuenta en el desarrollo de todo sistema inteligente.

Después de visualizar los posibles errores que el sistema podría cometer, es posible aunque se no está implementado en esta versión, construir un módulo que permita modificar el entrenamiento de las redes neuronales o al menos corregir los posibles errores producidos por un cambio brusco en las condiciones meteorológicas, ya que si estas no son normales el sistema tiene una alta probabilidad de fallo, también la posibilidad de cambiar las condiciones de búsqueda, teniendo más de una red entrenada con diferentes algoritmos y permitir que, si vemos que a lo largo del tiempo la red entrenada no encuentra más náufragos, continuar la búsqueda con otro modelo que nos permita un acercamiento mayor a la solución del problema.

CAPÍTULO 5. RESULTADOS EXPERIMENTALES

Una vez tenemos nuestro sistema desarrollado, debemos de chequearlo, en este capítulo vamos a intentar poner a prueba nuestro sistema, con un conjunto de test para ver como reacciona. Probaremos el sistema componente a componente con un conjunto diverso de valores, y también realizaremos una prueba conjunta de todo el sistema para ver si los consejos que nos da, son suficientemente fiables. Trataremos de sacar una idea general del comportamiento del sistema y un conjunto de consejos para una parametrización correcta de este, a fin de sacar el máximo provecho en el uso del sistema inteligente de ayuda al rescate.

Parametrización

A la hora de analizar la viabilidad de nuestro sistema, lo primero que debemos de hacer es analizar los parámetros en los que está basado el sistema inteligente, es decir cuáles son los diferentes parámetros usados en estas pruebas.

Los primeros parámetros son precisamente los parámetros de entrada de las diferentes redes neuronales. En la primera, estos parámetros son la posición (x,y) y el tiempo, para obtener la velocidad y la dirección en dicho punto. Sería ideal probar la red neuronal de forma independiente, con un conjunto de valores elegidos al azar para ver que las predicciones que realiza, son más o menos coherentes con los datos que le hemos entregado a dicha red.

Una vez hemos visto los parámetros de la red que realiza las predicciones, analizaremos los parámetros, de las red de decisiones, los parámetros de salida de esta red son la dirección y la velocidad que debe de tomar el avión. Esta red recibe como parámetros de entrada, la dirección de la media (dirMedia), dirección de los náufragos (dirNaufragos), porcentaje de náufragos descubiertos (porNaufragos), y distancia del avión a la media calculada en cada instante (distancia). Estos parámetros deben de ser calculados en todo momento con los datos disponibles por nuestro sistema. Nuestra red neuronal retornará la dirección hacia la que el avión debe dirigirse y la velocidad que debe de llevar en dicho momento.

Una vez conocemos la construcción de nuestras redes neuronales, y los parámetros que estas usan a la hora de construir los parámetros de salida, podemos ver cuáles son los parámetros a modificar para las diferentes pruebas que queremos realizar. Para la primera red neuronal, la que realiza la predicción de la dirección y velocidad que toma un náufrago, deberemos, una vez entrenada la red, darle una posición inicial, para poder ver que curva es la que se está construyendo, a lo largo del tiempo, con las diferentes posiciones que dicho náufrago va ocupando, es decir, el movimiento del náufrago.

Por otro lado para el caso de la red neuronal que usamos para tomar las decisiones necesarias para seleccionar la dirección del avión, deberemos darle una posición inicial tanto de los náufragos (x,y) como del avión respecto de los náufragos, también un número de horas de simulación, es también bastante importante el radio de visión del avión ya que este nos permitirá decidir si el avión ha pasado lo bastante cerca del náufrago como para descubrirle o no, no hemos incluido aquí la posibilidad de que aunque el avión pase lo suficientemente cerca, los

náufragos no sean descubiertos. También tendremos que tener en cuenta tanto la velocidad máxima del avión, como la velocidad de búsqueda que este deba llevar, el avión debe adaptarse lo más posible a la velocidad de los náufragos para poder perseguirlos de forma óptima.

Aunque no son parámetros de entrada, es importante modificar tanto la curva que realizan los náufragos como el algoritmo de entrenamiento de la red neuronal, esto requiere un mayor trabajo ya que representa el núcleo de nuestro sistema, pero es una prueba interesante de realizar para ver diferentes curvas de persecución. Por otro lado el modificar la curva que los náufragos realizan nos permite ver el comportamiento de nuestra red neuronal y como se va adaptando de forma dinámica a las trayectorias que nosotros vayamos definiendo, de esta manera probamos la adaptabilidad de nuestra red neuronal y del algoritmo que estamos usando para entrenarla. Con esta idea, la de probar la adaptabilidad y precisión de nuestra red neuronal de toma de decisiones es por lo que hemos seleccionado un conjunto de pruebas a realizar, basándonos en la modificación de diferentes parámetros.

Existen varias pruebas a realizar, la primera es sobre la red neuronal de toma de decisiones, midiendo mediante diferentes casos de uso algunas de las propiedades más relevantes que tiene dicha red neuronal, otras pruebas serán realizadas para medir la red neuronal de decisión y por último el sistema inteligente completo.

Modificaremos la posición inicial del avión respecto a los náufragos, haciendo que el avión comience en posiciones diferentes según la posición actual de los náufragos. Por ejemplo encima de estos, debajo, a un lado, más lejos, más cerca. Con esto veremos el comportamiento de la red neuronal y la curva que traza el avión antes de iniciar la persecución de los náufragos. Suponemos que la red intentará acercarse lo más posible a la media de los náufragos antes de iniciar un recorrido circular alrededor de esta.

- Cambios en los tiempos de cálculo, con la idea de ver de una forma más exacta la curva que describe el avión por encima de la posición sobre la que suponemos que están los náufragos. Es interesante probar con tiempos pequeños, dt=10 y posiblemente, aunque nos dará menos información, con tiempos grandes de cálculo, dt=120 (dt=60, dt=120, dt=10, dt=30).

- Cambios en la velocidad del avión. Esto nos permitirá ver de una forma más precisa el dato que estamos buscando, esto es, la curva que el avión describe sobre los náufragos y el comportamiento de la red neuronal con respecto a estos.

- Cambios en la curva que realizan los náufragos. Usando curvas trucadas como la sigmoide, seno etc. podemos ver el comportamiento de la red neuronal a la hora de perseguir diferentes casuísticas.

- Cambios en el algoritmo de entrenamiento de la red neuronal. Antes de llegar al algoritmo actual, se realizaron diferentes pruebas con algoritmos que a lo largo del tiempo empezaban a cometer fallos que hacían que se alejaran del resultado final deseado, es bueno ver por que fallaron.

Por otro lado, también queremos probar la red neuronal que realiza las predicciones del movimiento de nuestros náufragos, la idea es realizar varios tipos de entrenamiento, fundamentalmente con entradas diferentes que afectan a la dirección y velocidad del movimiento

de los náufragos, así como, la posibilidad de ver que curvas y recorridos son los que se han creado para nuestro náufrago de prueba.

Diferentes pruebas, casos de uso

Nuestro objetivo con estás pruebas es ver las curvas de persecución que realiza el avión según el recorrido de los náufragos y como los va encontrando, además de realizar la predicción del recorrido de los náufragos. Para ello definiremos los siguientes casos de uso que nos permitirán visualizar los resultados que deseamos obtener. La parametrización inicial del sistema es siempre, con el algoritmo actual de entrenamiento, curva normal, velocidad de 5 m/s y tiempo de cálculo en 60 s.

Veamos los **diferentes casos de uso** que hemos realizado, con ello intentamos probar la capacidad de nuestro sistema inteligente. Distribuiremos los ejemplos en cinco casos de uso:
Caso de uso A. Independencia de la posición náufragos/avión. Ejemplos 1, 2 y 3.
Caso de uso B. Influencia de la velocidad y el tiempo. Ejemplos 4, 5, 6, 7 y 8.
Caso de uso C. Seguimiento de la trayectoria de los náufragos. Ejemplos 9 y 10.
Caso de uso D. Predicción. Ejemplos 11 y 12.
Caso de uso E. Problema integrado. Ejemplos 13, 14 y 15.
Caso de uso F. Integración de la velocidad en el sistema. Ejemplos 16, 17 y 18.

Ejemplos desarrollados:
Ejemplo 1: Seleccionamos una posición inicial del avión por encima de los náufragos, vernos el acercamiento del UAV a estos.
Ejemplo 2: Seleccionamos una posición inicial del avión por debajo de los náufragos.
Ejemplo 3: Seleccionamos una posición inicial del avión lejos de los náufragos.
Ejemplo 4: Seleccionamos un espacio de tiempo de cálculo de 10 s, dt=10.
Ejemplo 5: Seleccionamos un espacio de tiempo de cálculo de 120 s, dt=120.
Ejemplo 6: Seleccionamos un espacio de tiempo de cálculo de 30 s, dt=30.
Ejemplo 7: Seleccionamos una velocidad de búsqueda del avión de 2 m/s, v=2.
Ejemplo 8: Seleccionamos una velocidad de búsqueda del avión de 20 m/s, v=20.
Ejemplo 9: Seleccionamos una curva de desplazamiento para los náufragos, *x=sin(y)* metros.
Ejemplo 10: Seleccionamos una curva de desplazamiento para los náufragos, *x=logsig(y)* metros.
Ejemplo 11: Seleccionamos para la red neuronal de predicción de velocidad y desplazamiento unos datos de entrenamiento de, dirección -3*pi/4 radianes, y velocidad 2 m/s.
Ejemplo 12: Seleccionamos para la red neuronal de predicción de velocidad y desplazamiento unos datos de entrenamiento de, dirección pi/2 radianes, y velocidad 1 m/s.
Ejemplo Integrado: Vemos el resultado de la integración en nuestro programa principal de ambas redes neuronales.
Ejemplo 14: Dejamos los náufragos inmovilizados para comprobar que es lo que pasa con el sistema en caso de que tenga que se encuentre muy cerca de la media y no haya descubierto a ninguno de los náufragos.
Ejemplo 15: Intentamos ver que es lo que sucede si los náufragos van en la dirección contraria al UAV.
Ejemplo 16: Intentamos ver que es lo que sucede si integramos la velocidad en el ejemplo 4.
Ejemplo 17: Intentamos ver que es lo que sucede si integramos la velocidad en el ejemplo 9.
Ejemplo 18: Intentamos ver que es lo que sucede si integramos la velocidad en el ejemplo 15.

Importante, **Nomenclatura a seguir**, en los gráficos del movimiento tanto de los náufragos como del UAV hay que tener en cuenta una nomenclatura básica. Los náufragos están coloreados en azul (.), y cuando son descubiertos son marcados con un círculo verde. El UAV en rojo (punto o un +), la trayectoria del UAV está marcada en rojo, determinándose mediante puntos la trayectoria a seguir por los náufragos. Solo en el caso integrado existen unos puntos de color amarillo que describen a los náufragos reales, cuya trayectoria está simulada por un modelo dinámico y aleatorio. Todas las distancias de las gráficas obtenidas y los ejes de estas están medidos en metros.

Caso de uso A (Independencia de la posición náufragos/avión)

En este caso de uso intentaremos demostrar como la curva de persecución generada por el UAV es totalmente independiente de la posición inicial de los náufragos respecto a la posición donde se encuentre el UAV en ese momento. Los ejemplos están diseñados para mostrar el comportamiento del UAV y la trayectoria seguida por este dependiendo del lugar de origen desde el que comience la persecución.

Ejemplo 1

Especificación/descripción del ejemplo**.** Seleccionamos una posición inicial del avión por encima de los náufragos, los náufragos están especificados en color azul y el UAV en color rojo. Esto nos permitirá realizar un análisis del acercamiento del avión sobre los náufragos, este ejemplo representa tanto la posición como la trayectoria que creemos sea la más frecuente o por lo menos la que en más casuísticas se debería de dar.

Breve descripción del resultado obtenido. En azul está el náufrago, en la primera ilustración vemos la trayectoria seguida por el náufrago, de izquierda a derecha y las posiciones que va tomando el UAV con respecto al náufrago. En la segunda ilustración vemos la trayectoria que empieza a tomar el UAV siguiendo la curva elíptica que genera el náufrago. Este ejemplo pretende demostrar como nuestra red neuronal toma las decisiones y con esas decisiones se construye la trayectoria alrededor de los náufragos. En la última ilustración vemos la curva un poco más cerca para ver desde otro punto de vista la aproximación que nos está creando nuestra red neuronal.

Resultado del ejemplo. Gráficos obtenidos:

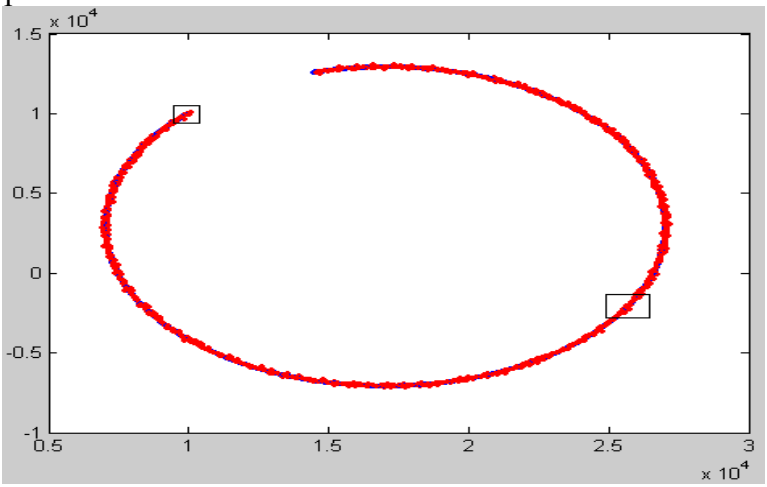


Ilustración 19: Ejemplo 1, curva de persecución

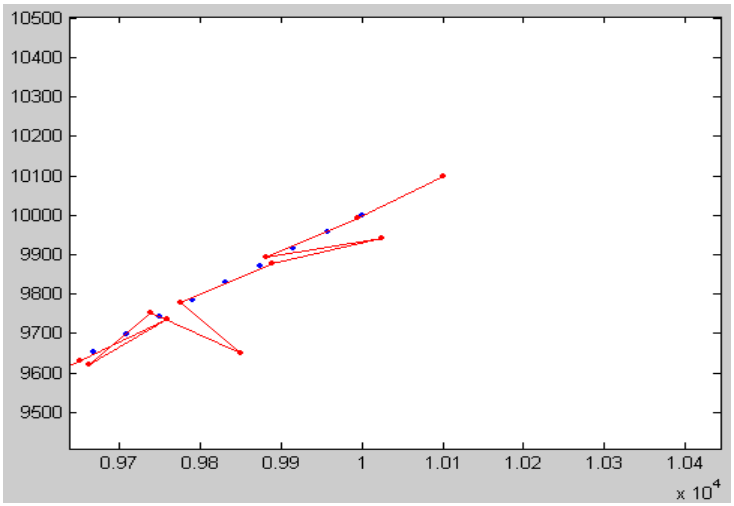


Ilustración 20: Ejemplo 1, inicio de la persecución

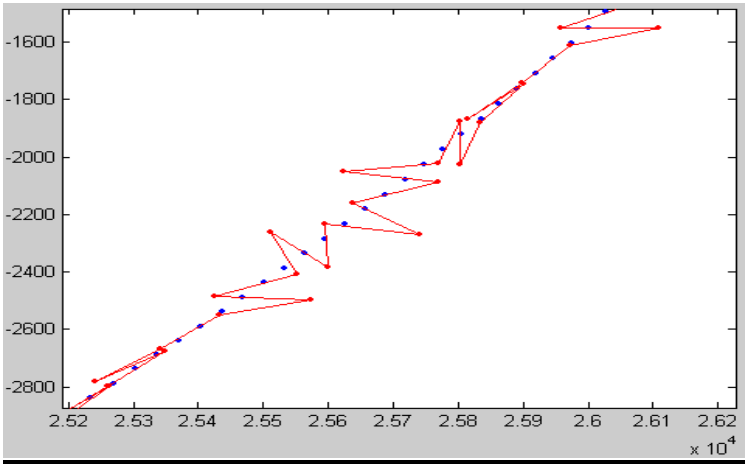


Ilustración 21: Ejemplo 1, forma de la persecución

Ejemplo 2

Especificación/descripción del ejemplo. Seleccionamos una posición inicial del avión por debajo de los náufragos. Este caso es una variación del anterior, ya que no podemos predecir la posición exacta en la que estará el avión, pero si analizar que sin importar dicha posición el comportamiento será siempre el mismo. Vemos claramente que la posición de salida del UAV no es importante a la hora de tomar una decisión por parte de la red neuronal, esta es totalmente independiente de la posición inicial desde la que parte el UAV.

Breve descripción del resultado obtenido. En este caso pretendemos visualizar como la red neuronal encargada de tomar la decisión, en la primera ilustración el UAV se dirige hacia la posición donde se cree que se encuentran los náufragos, estos comienzan la simulación en el centro de la imagen y el UAV a la derecha y debajo de los náufragos, no importa la posición inicial del avión la red neuronal intentará acercarse lo más posible a la posición de los náufragos. La segunda ilustración representa un trozo del recorrido realizado por nuestra red neuronal a lo largo del tiempo, como vemos se ajusta de forma bastante precisa a la curva realizada por los náufragos, quizá tanto que si algún náufrago no ha sido descubierto hasta este momento, nunca lo será.

Resultado del ejemplo. Gráficos obtenidos:

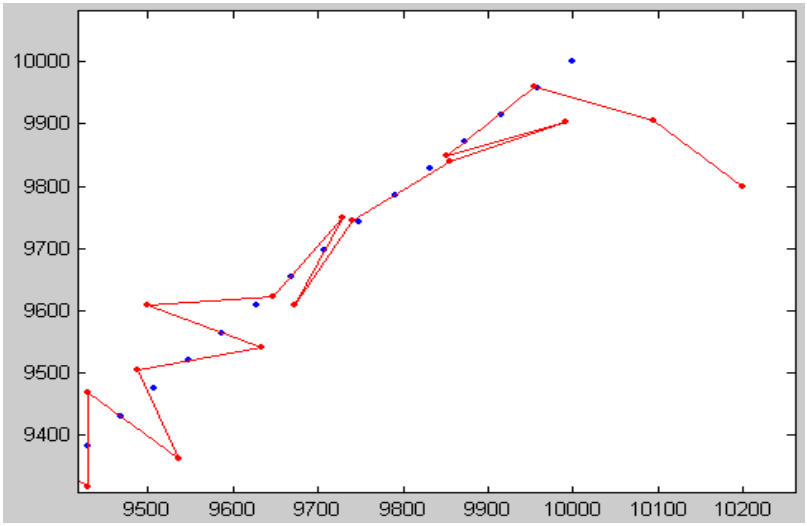


Ilustración 22: Ejemplo 2, inicio de la persecución

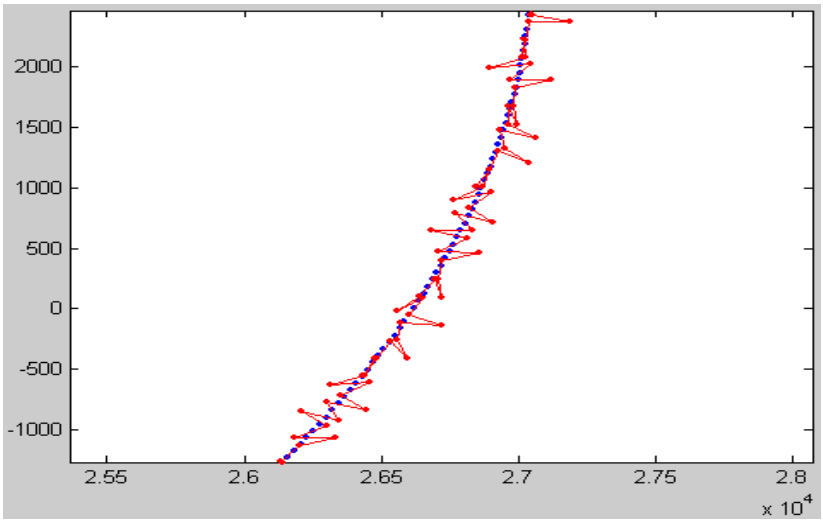


Ilustración 23: Ejemplo 2, forma de la persecución

Ejemplo 3

Especificación/descripción del ejemplo. Seleccionamos una posición inicial del avión lejos de los náufragos. La idea aquí es comprobar el comportamiento que tendría el avión si la posición en la que este se queda, difiere mucho de la posición donde, en teoría, se localizan los náufragos. Breve descripción del resultado obtenido. Este ejemplo es muy parecido a los anteriores, de hecho las condiciones son prácticamente las mismas, la única diferencia es que el avión empieza a volar a una distancia bastante más lejana que los casos anteriores y nos permite observar la curva de aproximación que realiza, así como el giro que hace para iniciar la persecución. En la segunda ilustración podemos ver como el UAV se acerca a la curva generada por el náufrago, la intercepta y luego continúa persiguiéndolo. Este ejemplo nos ha servido para completar lo visto en los ejemplos anteriores.

Resultado del ejemplo. Gráficos obtenidos:

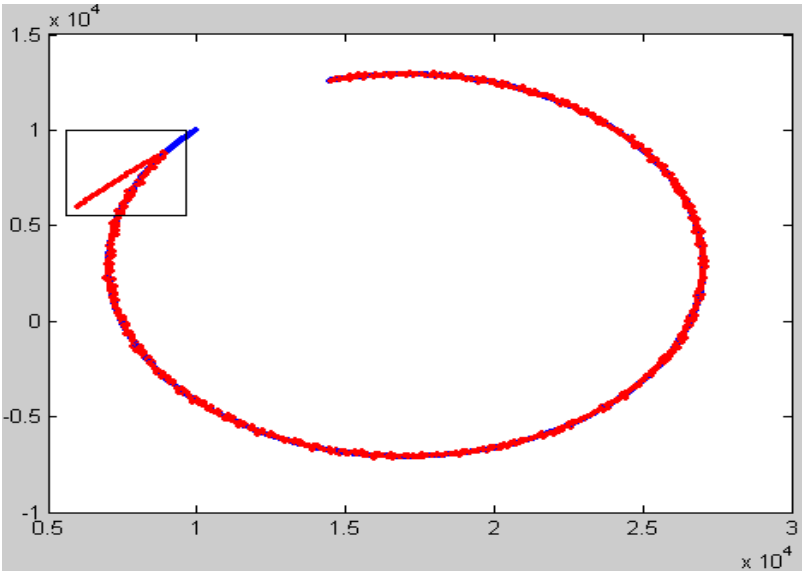


Ilustración 24: Ejemplo 3, curva de persecución.

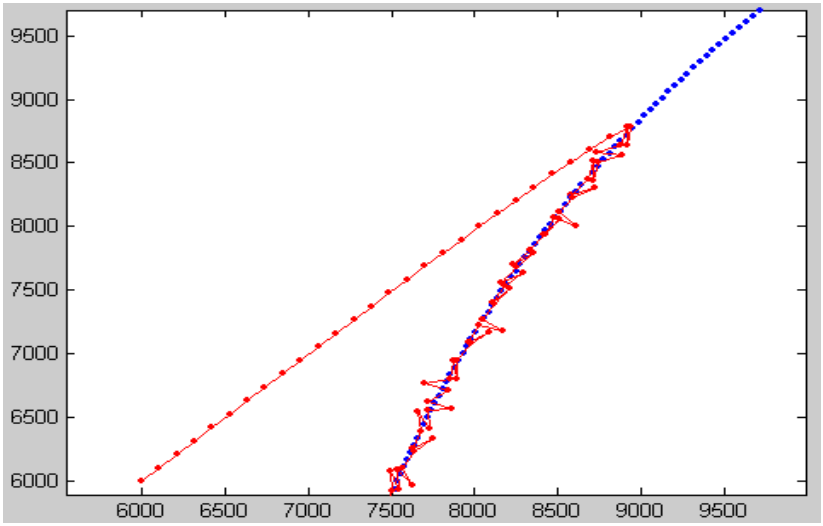


Ilustración 25: Ejemplo 3, inicio de la persecución

Caso de uso B (Influencia de la velocidad y el tiempo)

En este caso de uso queremos estudiar la influencia que tiene la velocidad, y el intervalo de tiempo que seleccionemos, a la hora de que el sistema tome decisiones, acerca, de la dirección que el UAV debe de adoptar. El intervalo de tiempo es bastante ya que es junto a la velocidad el parámetro que nos marca el espacio que el UAV va a recorrer. Si el tiempo o la velocidad son excesivamente grandes, puede ocurrir que la decisión sobre la dirección a tomar se realice demasiado tarde, por lo tanto la red neuronal deberá de tomar una decisión diferente que probablemente no será tan buena como la que hubiera tomado con las condiciones anteriores. Probablemente la posibilidad de tomar decisiones en intervalos de espacio suficientemente cortos, nos permita aproximar la curva de persecución a su resultado más óptimo.

Ejemplo 4

Especificación/descripción del ejemplo. Seleccionamos un espacio de tiempo de cálculo de 10 s, dt=10. La idea aquí es comprobar el comportamiento del avión alrededor de los náufragos y el tipo de curvas que describe.

Breve descripción del resultado obtenido. En este caso se puede apreciar con más detalle, mayor resolución, la curva que realmente está realizando el UAV alrededor de los puntos que está intentando descubrir, esta curva es como una catenaria y se va cruzando con la línea seguida por los náufragos. En la primera ilustración tenemos una visión general de la curva de persecución, en la segunda ilustración vemos la curva generada. Tanto en este como en ejemplos posteriores pretendemos descubrir cual es el resultado de las decisiones de la red neuronal y ver si es que se puede que aunque cambien los factores de velocidad y tiempo de cálculo, tiempo entre una toma de decisión y la siguiente, las decisiones se toman teniendo en cuenta las circunstancias actuales y no modifican el resultado final, aunque si el comportamiento que está definido por la curva de persecución de los náufragos.

Resultado del ejemplo. Gráficos obtenidos:

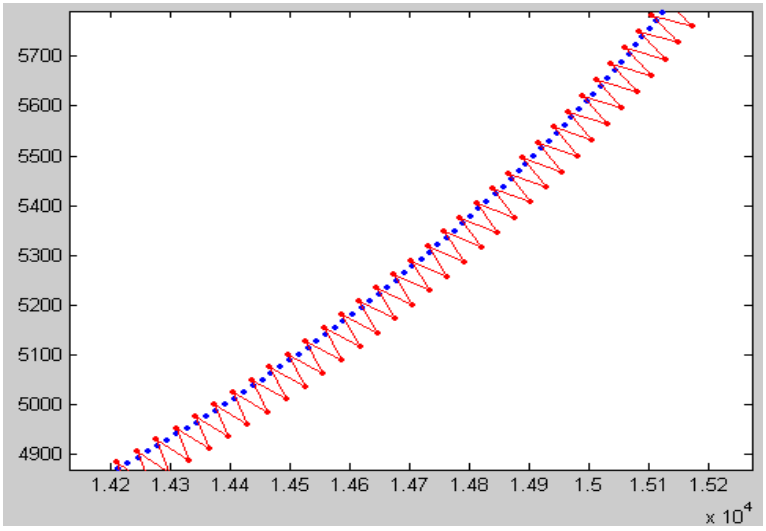


Ilustración 26: Ejemplo 4, curva de persecución

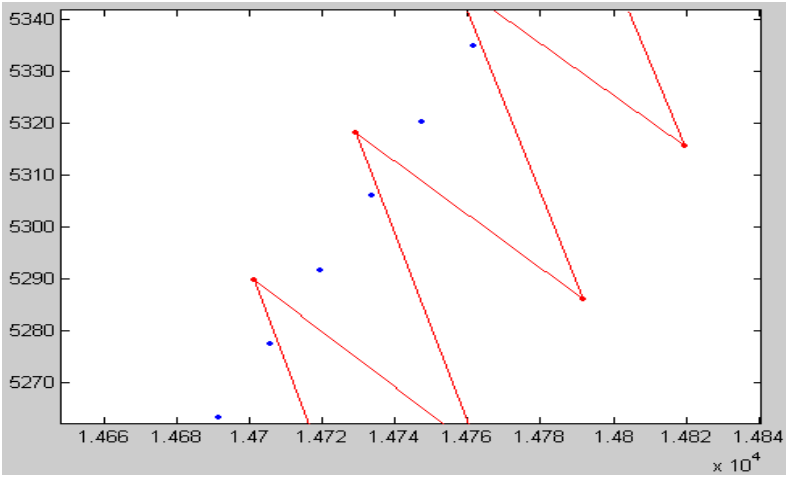


Ilustración 27: Ejemplo 4, detalle de la persecución

Ejemplo 5

Especificación/descripción del ejemplo. Seleccionamos un espacio de tiempo de cálculo de 120 s, dt=120. La idea aquí es visualizar de una forma más amplia la posición del avión alrededor de los náufragos, de esta forma veremos el tipo de persecución que realiza y lo mucho o poco que la trayectoria se va acercando a los náufragos a lo largo del tiempo, debido a que serán menos puntos de cálculo.

Breve descripción del resultado obtenido. Al elevar el número de segundos perdemos detalle de la curva real pero obtenemos una visión más amplia, que nos permite ver como nuestra red neuronal se adapta al movimiento del UAV. Con este ejemplo, tratamos de ver, comparativamente con el caso anterior, como afecta a la persecución el cambio en el intervalo de tiempo, como podemos ver por las gráficas obtenidas, el comportamiento del UAV es el mismo solo modificamos la precisión de nuestros cálculos que hacen que el UAV oscile mucho más y de forma más imprecisa alrededor de la curva desarrollada por los náufragos.

Resultado ejemplo. Gráficos obtenidos:

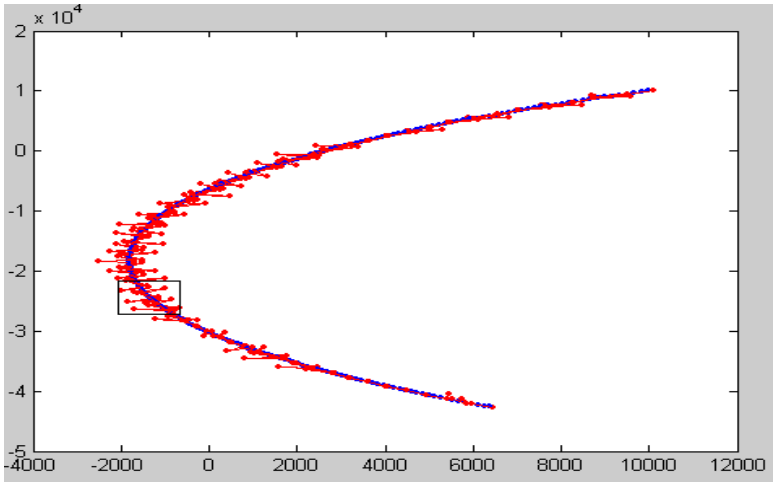


Ilustración 28: Ejemplo 5, curva de persecución

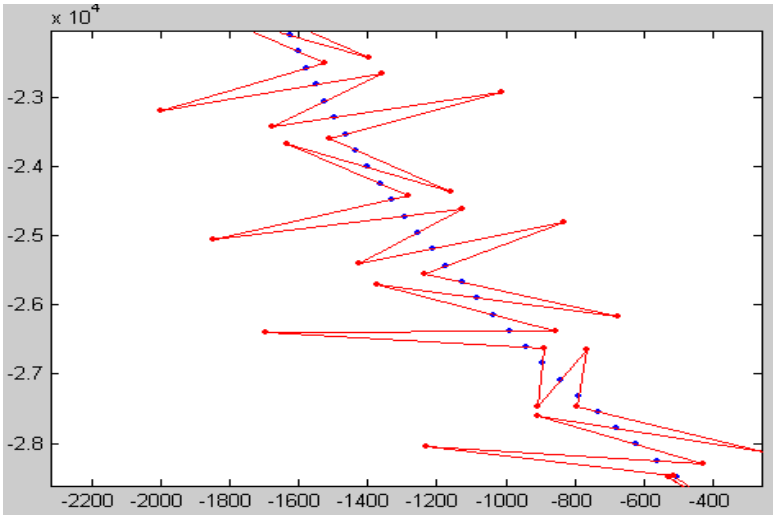


Ilustración 29: Ejemplo 5, detalle de la persecución

Ejemplo 6

Especificación/descripción del ejemplo. Seleccionamos un espacio de tiempo de cálculo de 30 s, dt=30. Idéntico al caso 4, solo que con un número menor de puntos, este caso es solo con la idea de comparar al con el caso 4.

Breve descripción del resultado obtenido. Este es un caso intermedio entre los dos anteriores, donde podemos apreciar una mezcla de ambos casos, por un lado la aproximación general que hace la red neuronal a la curva descrita por los náufragos, y por otro lado como esa curva, en realidad, oscila alrededor de los náufragos, con la idea de ir descubriéndolos. En la ilustración 29 se ve bastante bien la oscilación del UAV.

Resultado del ejemplo. Gráficos obtenidos:

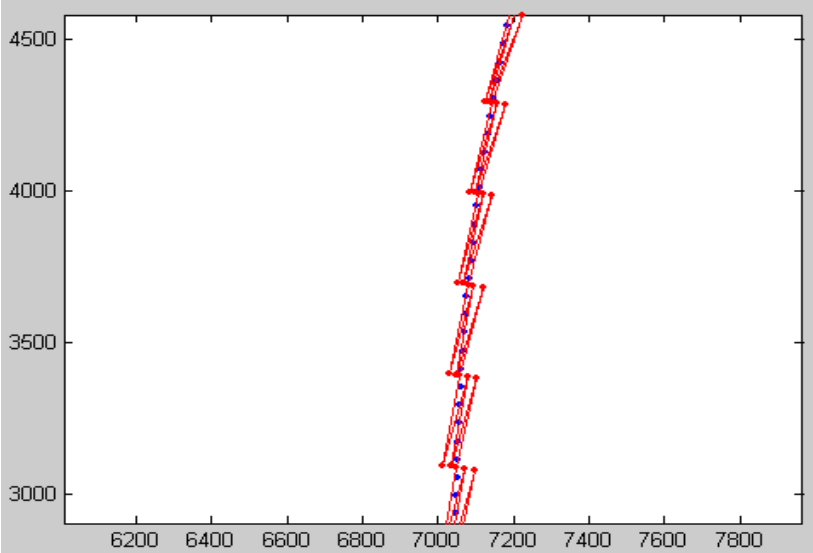


Ilustración 30: Ejemplo 6, persecución

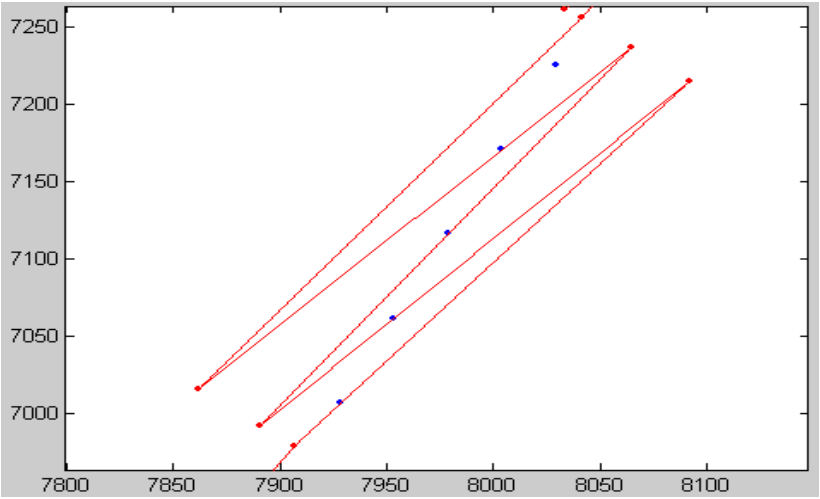


Ilustración 31: Ejemplo 6, detalle de la persecución

Ejemplo 7

Especificación/descripción del ejemplo. Seleccionamos una velocidad de búsqueda del avión de 2 m/s, v=2. La velocidad estándar que estamos usando es 5 m/s, este caso nos permitirá ver cuáles son las decisiones de la red neuronal cuando la velocidad del avión es parecida a la de los náufragos. Analizaremos las curvas y las compararemos con el resto de los casos.

Breve descripción del resultado obtenido. Vemos como la velocidad influye en la persecución, el UAV, con una velocidad igual a la de los náufragos se adapta a la curva de los náufragos pero nunca los alcanza. Pretendemos demostrar la necesidad de una velocidad adaptativa y por lo tanto vemos de forma clara con este ejemplo que la velocidad debe de ser un parámetro a tener en cuenta por su importancia a la hora de obtener éxito en nuestra persecución. Probablemente este ejemplo visualice la necesidad de añadir la velocidad como parámetro de salida de la red neuronal de decisiones, o en otro caso de añadirlo de alguna otra forma en el sistema inteligente. En ambas ilustraciones vemos que el UAV nunca alcanza al náufrago.

Resultado del ejemplo. Gráficos obtenidos:

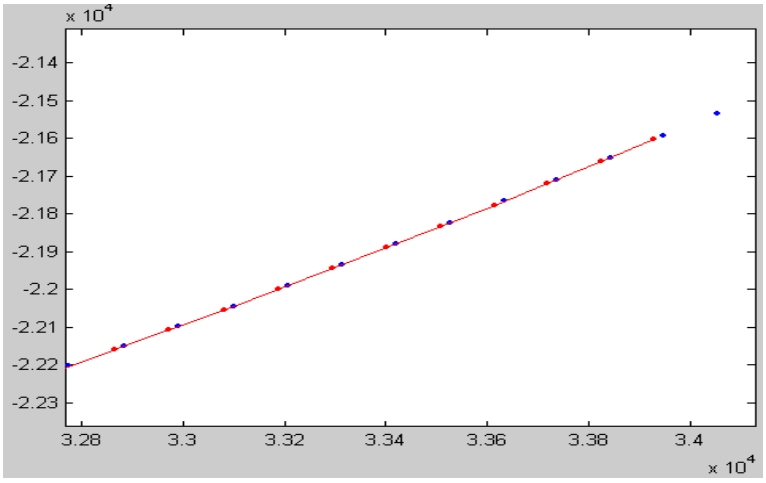


Ilustración 32: Ejemplo 7, final de la persecución

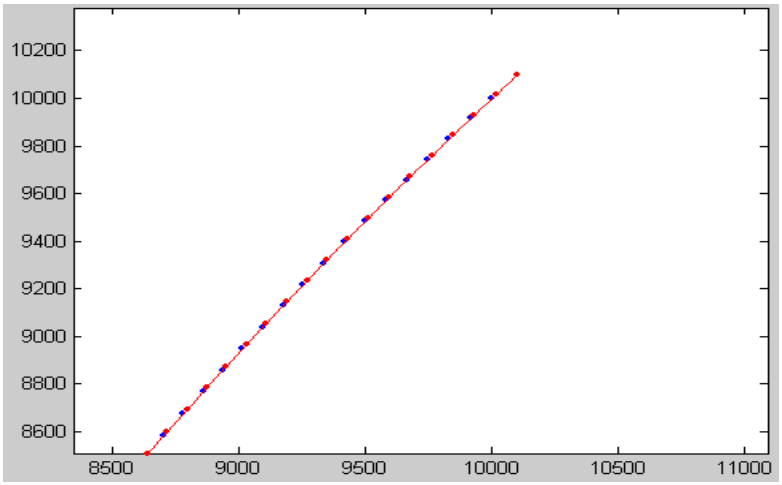


Ilustración 33: Ejemplo 7, detalle de la persecución

Ejemplo 8

Especificación/descripción del ejemplo. Seleccionamos una velocidad de búsqueda del avión de 20 m/s, v=20. Velocidad mucho mayor que la de los náufragos, será interesante ver la capacidad de recuperación de la red neuronal al ver que se está alejando demasiado de los náufragos, debido a la velocidad del avión.

Breve descripción del resultado obtenido. Vemos como cambia la curva del UAV al aumentar la velocidad, el UAV aumenta la distancia entre los náufragos y la posición de retorno, de esta forma construye una curva alrededor de los náufragos en forma de oscilación más amplia de lo que hemos visto en los otros casos. En este caso una velocidad demasiado grande perjudica de forma clara a la persecución haciendo que el UAV amplie demasiado las oscilaciones y se aleje de su objetivo, en la ilustración 33 se ve claramente como el UAV genera un movimiento oscilatorio demasiado amplio sobre el náufrago.

Resultado del ejemplo. Gráficos obtenidos:

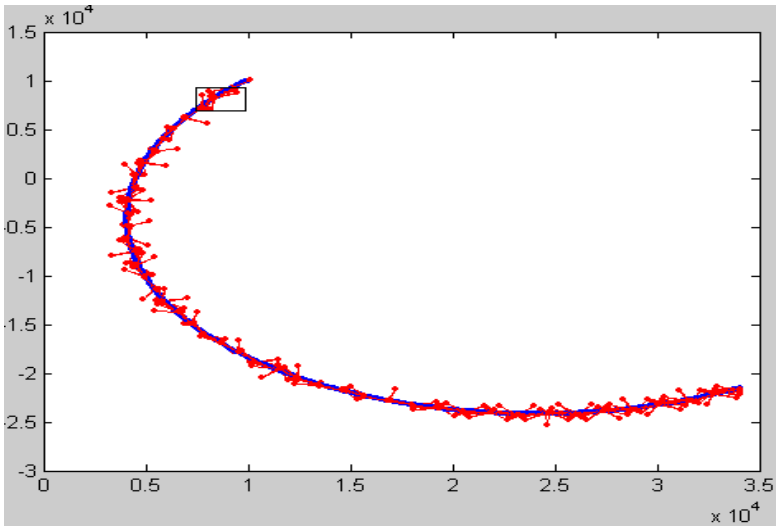


Ilustración 34: Ejemplo 8, curva de persecución

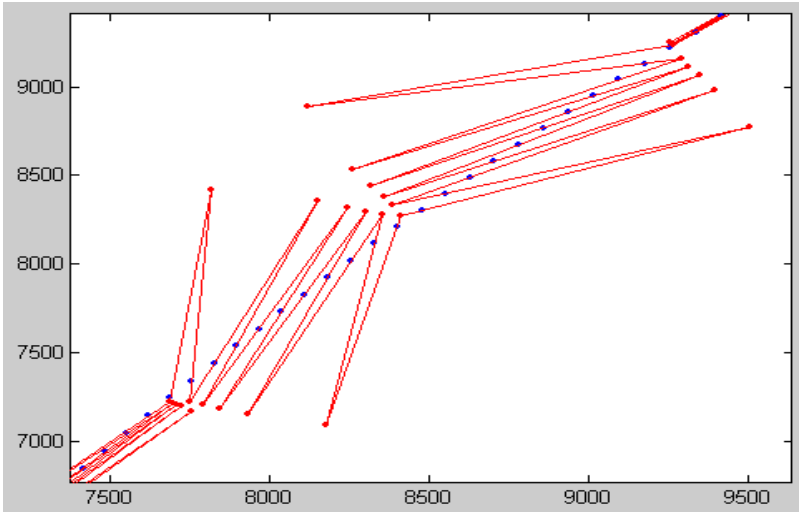


Ilustración 35: Ejemplo 8, detalle de la persecución

Caso de uso C (Seguimiento de la trayectoria de los náufragos)

Este caso de uso se desarrolló con la idea de mostrar el comportamiento, de la red neuronal y del sistema completo, de forma totalmente independiente del modelo de predicción. Aquí podemos ver a nuestro náufrago moverse de forma controlada, siguiendo unas curvas complejas y definidas de antemano, para poner a prueba el sistema de seguimiento. Como vemos existe un desfase del sistema respecto a la curva realizada por el náufrago, esto es debido a la influencia del parámetro velocidad dentro del sistema. Por otro lado nuestro modelo de decisión intenta adaptarse, lo más posible a la curva generada por el náufrago, no importa cuál sea la curva que este describa. Estos ejemplos y este caso de uso, es una demostración del éxito que puede llegar a tener el sistema.

Ejemplo 9

Especificación/descripción del ejemplo. Seleccionamos una curva de desplazamiento para los náufragos, ***x=sin(y)*** metros. Hemos cambiado la curva de desplazamiento de los náufragos, ahora el desplazamiento de los náufragos no está dirigido por los parámetros de velocidad y dirección, si no que están regidos por una ecuación y unos puntos elegidos al azar de esta. ¿Qué hará nuestra red neuronal?

Breve descripción del resultado obtenido. En este caso vemos como la red neuronal de decisión realiza la persecución de la curva de náufragos que le hemos dado, realiza el mismo movimiento, aunque con un desfase de unos 10 m de media por delante, debido a la diferencia de velocidad entre ambos, ya que como la red neuronal no es capaz de decidir los cambios de velocidad, no consigue alejarse nunca lo suficiente para luego intentar retroceder y localizar a nuestro náufrago. Hasta ahora los náufragos han tomado caminos más o menos homogéneos, digamos que la curva que construían iba dirigida por el flujo de las mareas y corrientes que normamente es unidireccional, aquí hemos pretendido probar nuestro sistema inteligente y en particular nuestra red neuronal de decisión con una corriente marina que oscila, de esta manera intentamos ver que es lo que hace nuestro sistema, si realmente es capaz de seguir a un náufrago cuya dirección es "impredecible" en el tiempo. Como vemos en ambos gráficos el UAV toma la dirección que han tomado los náufragos e intenta perseguirlos, como en otros casos la velocidad y el tiempo de cálculo, hace que el UAV se desfase respecto a los náufragos unos 10 metros.

Resultado del ejemplo. Gráficos obtenidos:

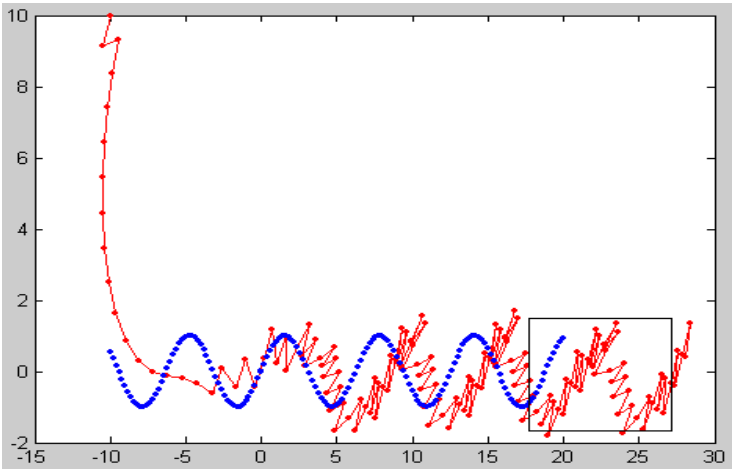


Ilustración 36: Ejemplo 9, curva de persecución

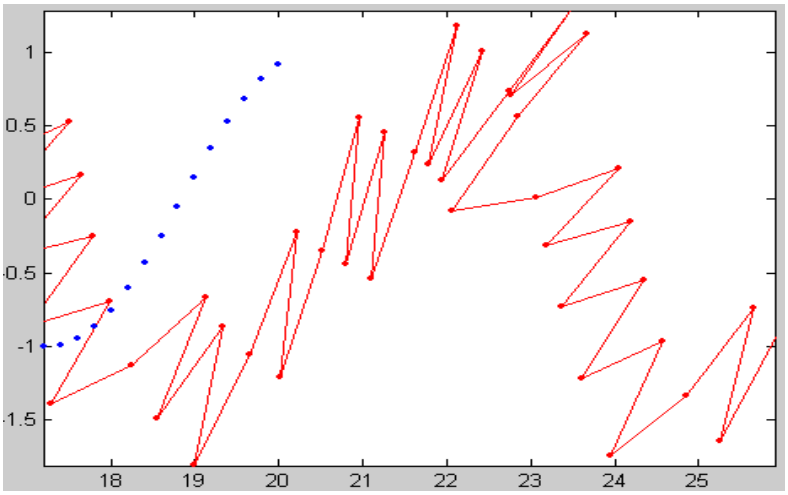


Ilustración 37: Ejemplo 9, detalle de la persecución

Ejemplo 10

Especificación/descripción del ejemplo. Seleccionamos una curva de desplazamiento para los náufragos, ***x=logsig(y)*** metros. Hemos cambiado la curva de desplazamiento de los náufragos, ahora el desplazamiento de los náufragos no está dirigido por los parámetros de velocidad y dirección, si no que están regidos por una ecuación y unos puntos elegidos al azar de esta. ¿Qué hará nuestra red neuronal?
Breve descripción del resultado obtenido. Este caso es parecido al anterior con la diferencia de la curva que realizan los náufragos, vemos como la red neuronal intenta ajustar a la curva de náufragos, y debido a la diferencia de velocidad, realiza un movimiento de oscilación alrededor de esta.

Resultado del ejemplo. Gráficos obtenidos:

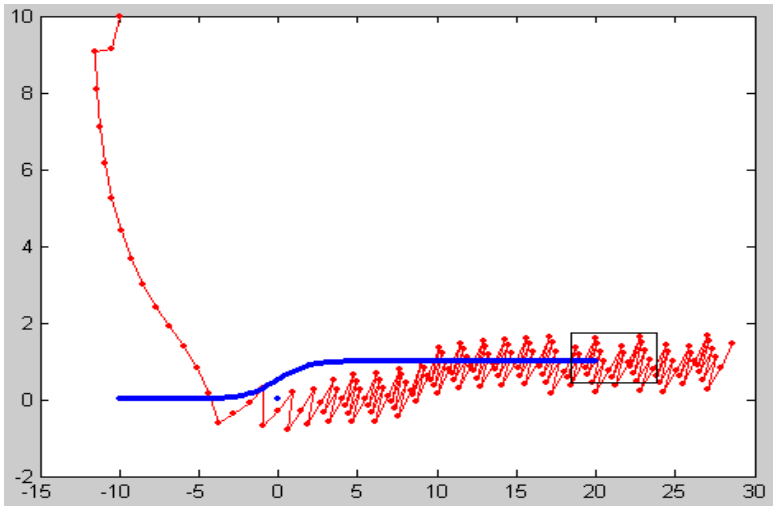


Ilustración 38: Ejemplo 10, curva de persecución

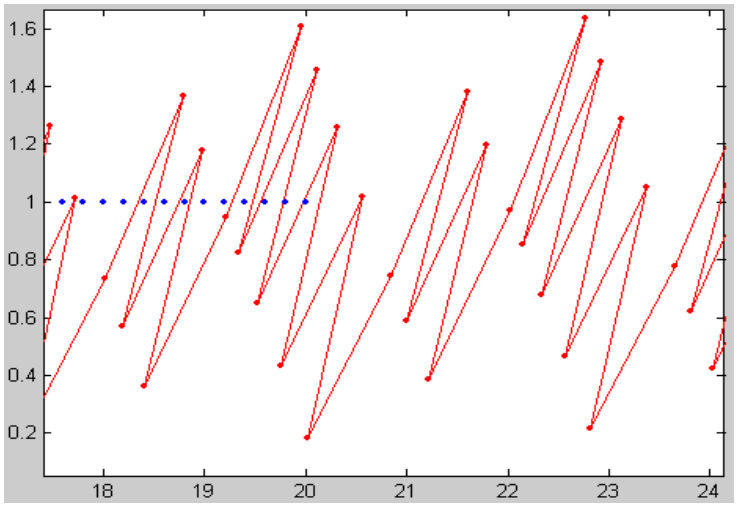


Ilustración 39: Ejemplo 10, detalle de persecución

Caso de uso D (Predicción)

Hemos construido una malla de 30 * 30 km, para poder predecir lo que ocurre con las corrientes marinas y el viento, para ello entrenaremos nuestra red neuronal para que pueda predecir, dados dos puntos de la malla, que dirección y velocidad tendrá un náufrago que esté en dicho punto. Para realizar esta prueba necesitamos un mapa marino de las corrientes y vientos de la zona, como no disponemos de ninguno, tendremos que definir unos valores de prueba con los que entrenar nuestra red. En el primero de los casos el valor de prueba será velocidad constante 2 m/s y dirección -3*pi/4, para el segundo caso lo complicaremos algo más definiendo diferentes direcciones del viento y corrientes según el cuadrante donde se sitúe el náufrago. La idea es probar el comportamiento de la red neuronal de predicción del movimiento de los náufragos y comparar su estabilidad con los patrones que le hemos enseñado.

Ejemplo 11

Especificación/descripción del ejemplo. Seleccionamos para la red neuronal de predicción de velocidad y desplazamiento unos datos de entrenamiento de, dirección -3*pi/4 radianes, y velocidad 2 m/s. Este ejemplo no pretende probar la red neuronal de decisión si no que pretende probar la red neuronal que permite predecir la dirección y la velocidad de desplazamiento de los náufragos. La idea es ver la curva que se genera con las predicciones de nuestra red neuronal

Breve descripción del resultado obtenido. La curva es una recta descendente debido a que la malla ha sido entrenada con la dirección -3*pi/4, en un caso real sería entrenada con un conjunto de datos que generarían diferentes direcciones para cada punto.

Resultado del ejemplo. Gráficos obtenidos:

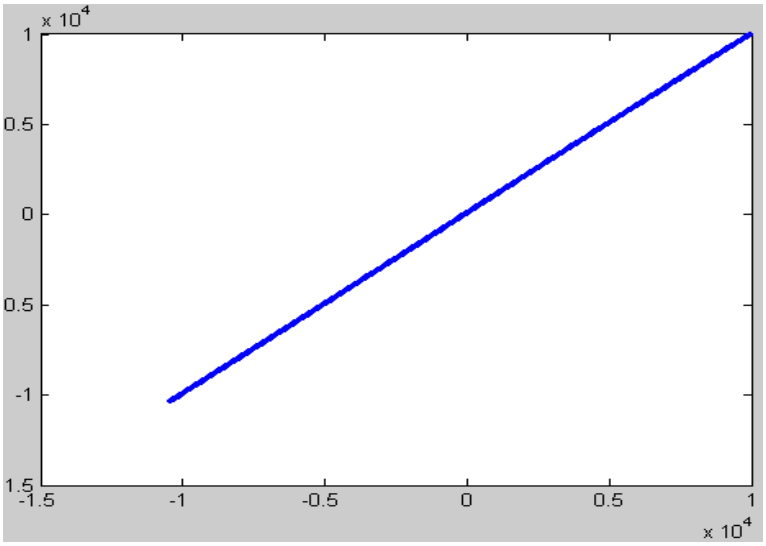


Ilustración 40: Ejemplo 11, curva generada por los náufragos con esta dirección

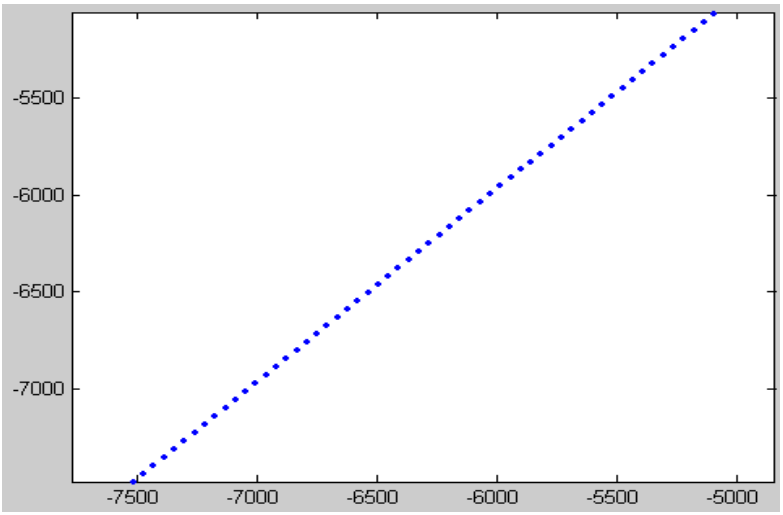


Ilustración 41: Ejemplo 11, detalle de la curva generada por los náufragos

Ejemplo 12

Especificación/descripción del ejemplo. Seleccionamos para la red neuronal de predicción de velocidad y desplazamiento unos datos de entrenamiento de, dirección dada por el cuadro anexo, y velocidad 1 m/s. Este ejemplo pretende probar el grado de adaptabilidad de la red neuronal al mapa de corrientes y vientos que le estamos enseñando, y ver como lo aprende y es capaz de generar los valores que no ha aprendido. Veremos la curva que se genera con las predicciones de nuestra red neuronal.

Breve descripción del resultado obtenido. Hemos cambiado la dirección y la velocidad de entrenamiento, el resultado es parecido al anterior, pero esto demuestra que la red cambia su comportamiento según el entrenamiento que realicemos.

Mapa de direcciones que le hemos enseñado a nuestra red neuronal:

Dirección	0-10 km	10-20 km	20-30 km
22.5-30 km	7*pi/4	7*pi/4	7*pi/4
15-22.5 km	3*pi/4	3*pi/4	3*pi/4
7.5-15 km	7*pi/4	7*pi/4	7*pi/4
0-7.5 km	2*pi	2*pi	2*pi

Resultado del ejemplo. Gráficos obtenidos:

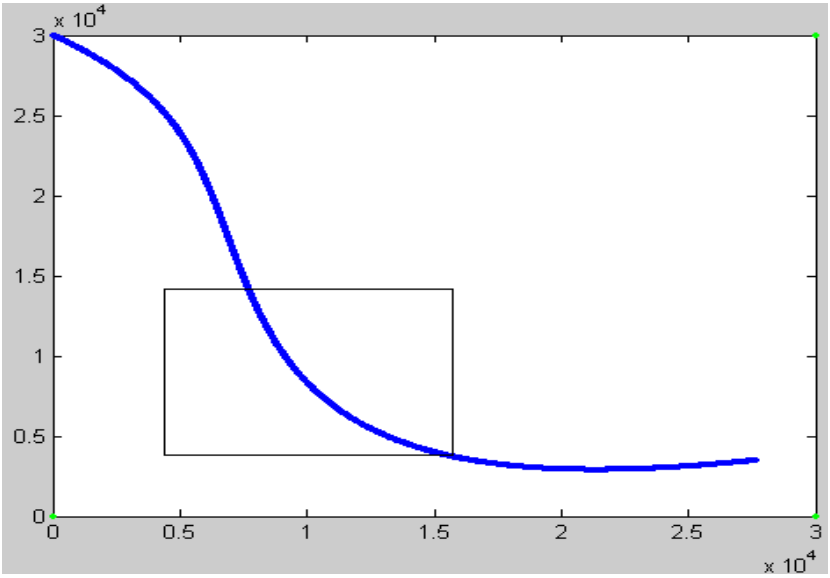


Ilustración 42: Ejemplo 12, curva generada por los náufragos para el mapa de direcciones

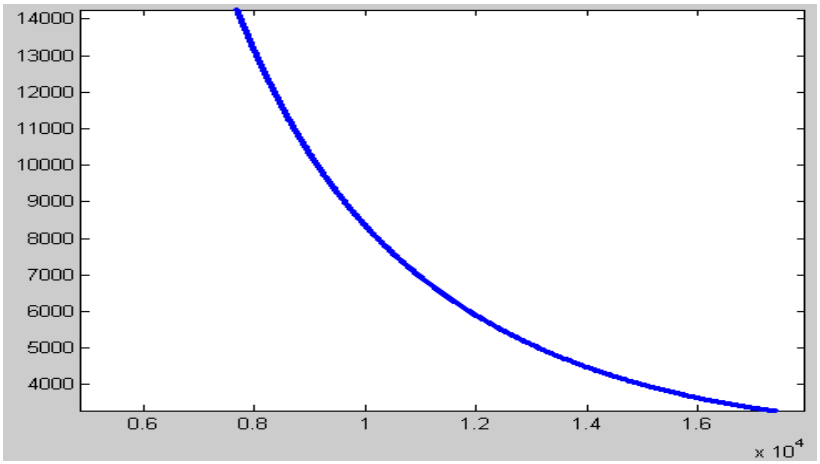


Ilustración 43: Ejemplo 12, detalle del cambio de dirección en la curva generada por nuestro náufrago

Caso de uso E (Problema integrado)

En este caso de uso pretendemos demostrar el comportamiento local del sistema que decide la dirección a seguir por el UAV, no solo pretendemos ver de forma general, como hasta ahora hemos visto, que la red neuronal y el sistema intentan perseguir a los náufragos adoptando trayectorias de aproximación hacia estos. Veremos como también es capaz de ir localizando estos náufragos y usar la información descubierta para mejorar dichas trayectorias, hasta la completa localización de los náufragos. El sistema no solo persigue a los náufragos sino que además se mueve de una forma más o menos inteligente entre ellos intentando descubrir los náufragos que le faltan hasta completar el 100% de estos.

Evidentemente, como todo sistema, este también tiene un margen de error, por lo que es posible, y lo vamos a ver en los ejemplos, que no podamos descubrir a todos los náufragos, por lo que se hace necesario, añadir limitaciones temporales al sistema a fin de limitar el error cometido, y no entrar en un bucle de búsqueda sin ningún final.

Ejemplo Integrado

Especificación/descripción del ejemplo. En este caso vamos a integrar y estudiar ambas redes neuronales trabajando conjuntamente, veremos como una define la curva que siguen los náufragos, y la otra va decidiendo el movimiento del avión, mientras se van descubriendo los náufragos que faltan, esto nos permitirá comprobar las características de nuestro sistema inteligente en acción.

Breve descripción del resultado obtenido. Vemos como el UAV localiza la posición de los náufragos, estos se encuentran dentro de la elipse, y como mas adelante va descubriendo estos y añadiéndolos a la lista de náufragos descubiertos. Los náufragos rojos son los que faltan por descubrir y los azules los ya descubiertos. Por otro lado vemos que el resultado es el esperado ya que el movimiento de los náufragos está generado por el propio sistema, en realidad en este ejemplo, nuestro UAV persigue e intenta localizar los náufragos que el propio mapa de direcciones del sistema genera.

Este es el caso en el que se ve el funcionamiento completo del sistema. Los náufragos comienzan su movimiento en la dirección -3*pi/4, cuando el UAV comienza la persecución al cabo de 20 minutos. Finalmente se encuentran y dicho encuentro está marcado por la elipse que vemos en la primera ilustración, una vez se han encontrado, el UAV localiza 7 de los 10 náufragos, por lo que tiene que iniciar la persecución de los náufragos, con los datos de que dispone hasta este momento. El símbolo + representa el UAV y los náufragos están marcados, en azul los que se han encontrado hasta este momento y en rojo los que faltan por ser encontrados. La segunda ilustración marca el momento en el que el UAV ha sido capaz de encontrar a dos de los náufragos, siendo marcados en dicho momento con color azul, también podemos ver como el UAV marcado con + ha cambiado la posición de persecución con respecto a la ilustración anterior, mostrándonos ese movimiento oscilante que hemos visto en ejemplos anteriores. Durante los siguientes intervalos de tiempo hasta completar las 8 horas de persecución el UAV no es capaz de encontrar al náufrago que falta.

A continuación de esta simulación con náufragos simulados por el sistema, podremos ver la simulación con náufragos reales. En esta simulación, el UAV, intentará localizar todos los náufragos que pueda de los que hemos generado mediante una simulación dinámica e

independiente de la red neuronal de predicción, que es la que ha simulado los náufragos con los que hemos trabajado hasta ahora.

Resultado del ejemplo. Gráficos obtenidos:

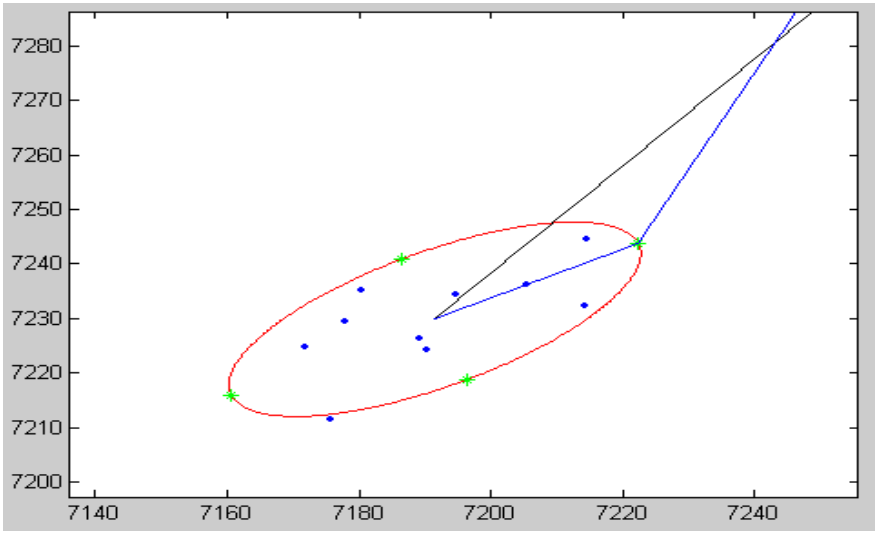


Ilustración 44: Ejemplo Integrado, localización de los náufragos

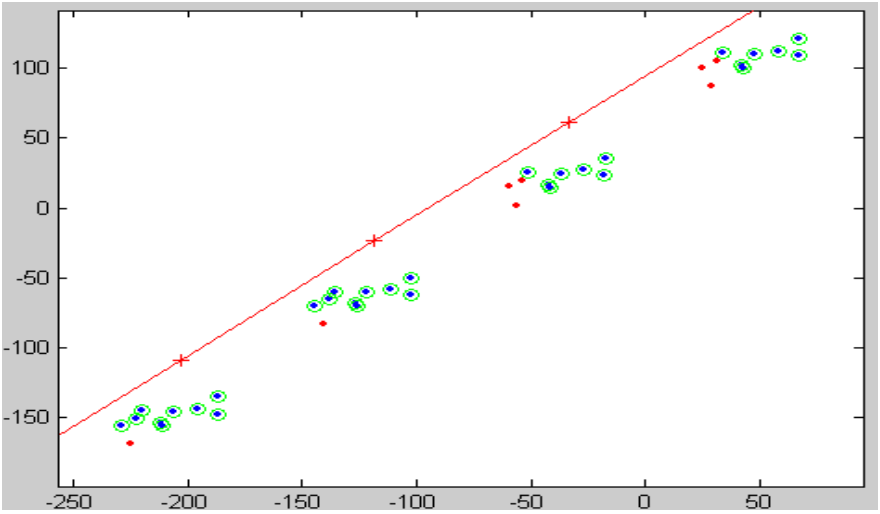


Ilustración 45: Ejemplo Integrado, descubrimiento de náufragos

Una vez hemos visto como el UAV se integra con el sistema de localización y decisión, nos interesaría ver cómo se comporta todo el sistema, con un ejemplo, donde los náufragos son guiados de forma independiente respecto del cerebro del UAV. Para determinar la fiabilidad del sistema, una vez visto el funcionamiento en este mismo ejemplo, lo que haremos es añadir un conjunto de náufragos de forma independiente al sistema, e intentaremos responder a dos preguntas, la primera se refiere a la red neuronal de predicción del movimiento de los náufragos, ¿Qué capaz es dicha red de predecir el movimiento de los náufragos?, y la segunda respecto al sistema de seguimiento, ¿Es capaz de perseguir a los náufragos reales e ir descubriéndolos?
Vamos a intentar responder a estas preguntas añadiendo nuestros náufragos independientes, comparando su movimiento con el definido por la red neuronal y revisando el movimiento que realiza el UAV para intentar descubrirlos a todos.

Antes de ver los resultados, intentaremos explicar la terminología usada para describir cada uno de los componentes de la simulación. El símbolo + representa el UAV, las líneas rojas entre dos + marcan la trayectoria del UAV, los náufragos reales que hemos encontrado están marcados en azul con un circulo verde alrededor, los náufragos reales que no han sido encontrados están marcados en amarillo, en azul la dirección de los náufragos que predice el sistema.

Resultado obtenido. Lo primero que vemos es que el modelo dinámico se expande bastante más que nuestro modelo de predicción, aunque sigue la misma dirección, esta expansión se realiza a ambos lados de nuestra predicción y ocupa unos 500 metros, lo que nos da la idea de que nuestra red neuronal de predicción no va mal desencaminada, prácticamente es capaz de clavar el movimiento de la media de los puntos. Por otro lado observamos que el UAV una vez ha llegado al centro de la elipse generada por los autovectores, y ha descubierto algún náufrago, inmediatamente inicia la localización de los náufragos restantes hasta, en este caso lograr descubrir un 60% de los náufragos, lógicamente esperábamos obtener el 100%. En esta simulación hemos tenido la oportunidad de ver como el sistema trabaja de forma conjunta e independiente de los náufragos reales, cuando lo integramos con los náufragos reales, una vez llega al punto de encuentro que hemos calculado, el sistema responde tal y como habíamos previsto que lo haría, oscilando hasta encontrar algún náufrago, persiguiendo a los náufragos y encontrándolos.

Esta simulación es bastante precisa con respecto a lo que ocurriría si el sistema se encontrara en una situación real, podemos comparar, tanto la aproximación generada por el sistema con respecto al modelo que predice el movimiento de los náufragos, como el movimiento que el UAV realiza alrededor de los náufragos reales, desvinculándose de los náufragos de la simulación, para llegar a encontrarlos y usar dicha información para encontrar al resto de los náufragos.

Resultado del ejemplo. Gráficos obtenidos:

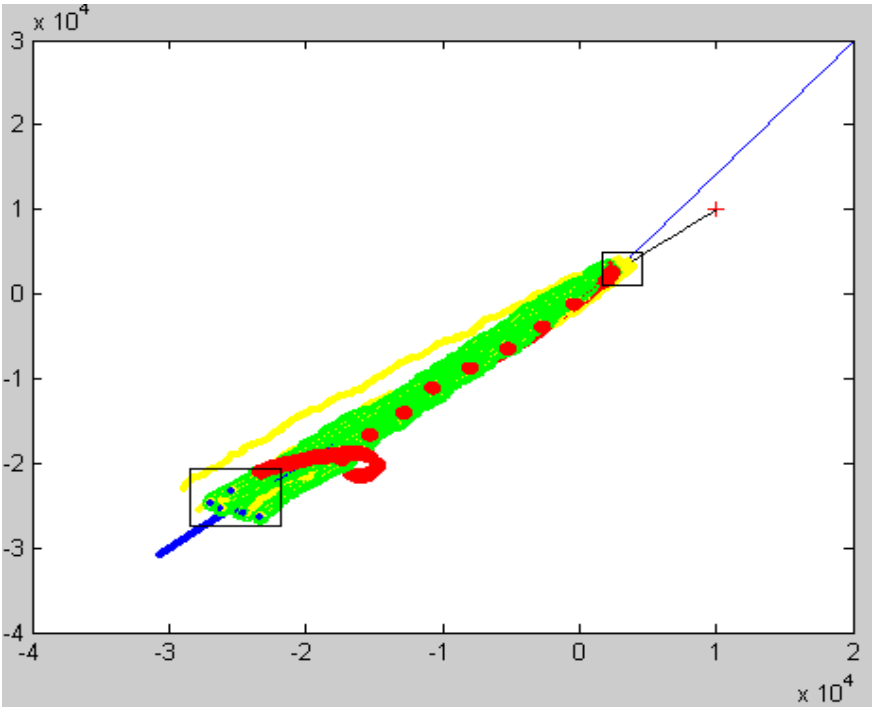


Ilustración 46: Ejemplo integrado, comparativa de la simulación del sistema

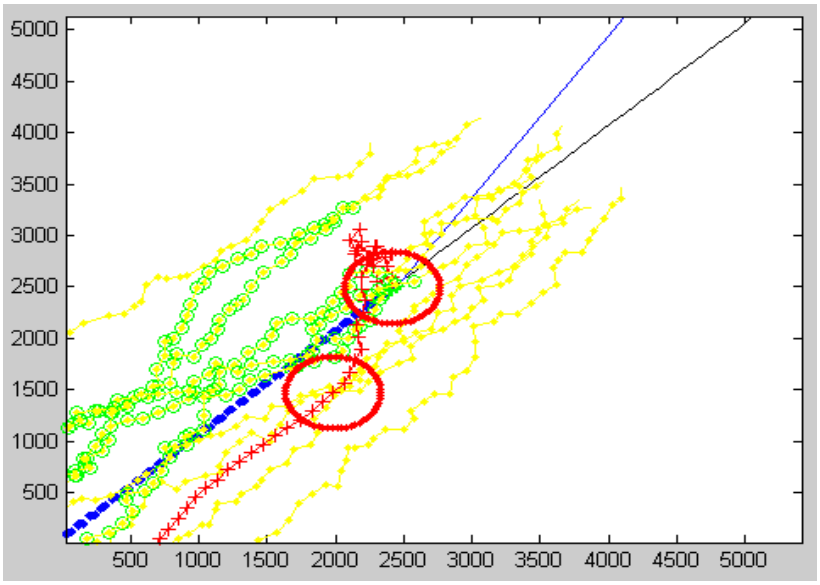


Ilustración 47: Ejemplo integrado, inicio de la persecución

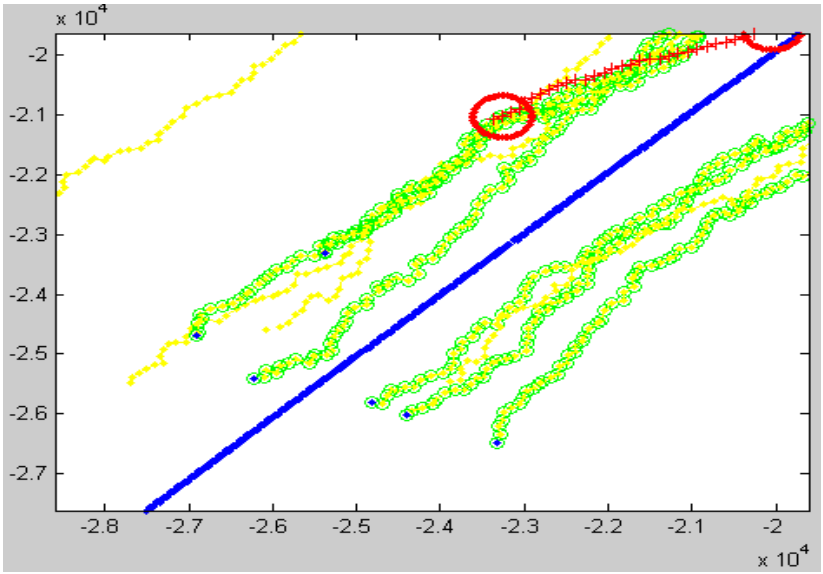


Ilustración 48: Ejemplo integrado, final de la simulación

Finalmente añadimos una aproximación del ejemplo integrado a la que se le ha añadido la velocidad como parámetro de salida, en la red neuronal de toma de decisiones. En este caso el porcentaje de acierto ha sido menor entorno al 50%, aunque la curva que ha descrito el UAV, para intentar encontrar a los náufragos está dotada de un modelo más inteligente donde se ve, como intenta realizar un recorrido circular por las posiciones que ocuparon los náufragos, aunque un poco alejado de la posición real de estos en este momento. Parametrizando el sistema inteligente hemos visto que podemos ir dotándolo cada vez de más inteligencia.

A continuación vemos un conjunto de gráficas donde se puede apreciar como la red neuronal cambia la velocidad y la dirección para de una forma inteligente ir descubriendo los náufragos, todas las líneas debería tener la misma longitud, si la velocidad fuera la misma ya que el intervalo de tiempo que usamos para la toma de datos es constante, pero en este caso vemos que no es así, ya que la red neuronal intenta adaptar la velocidad para modificar la trayectoria descrita por el UAV.

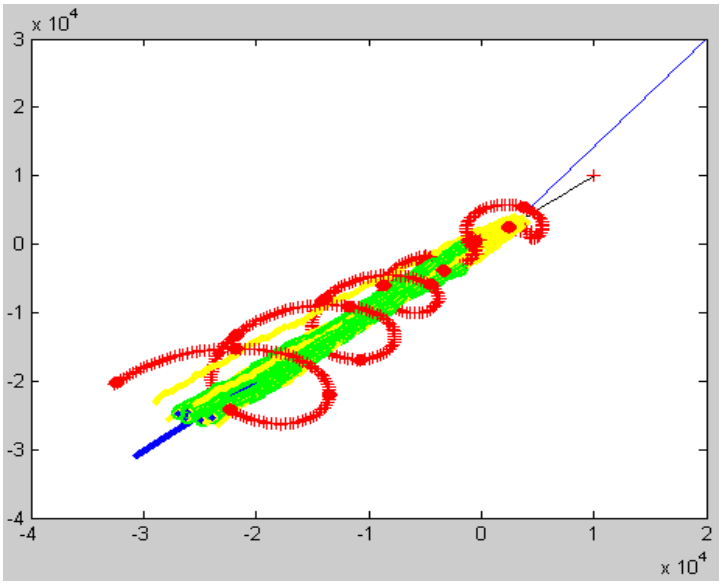


Ilustración 49: Ejemplo integrado, simulación con la velocidad como parámetro

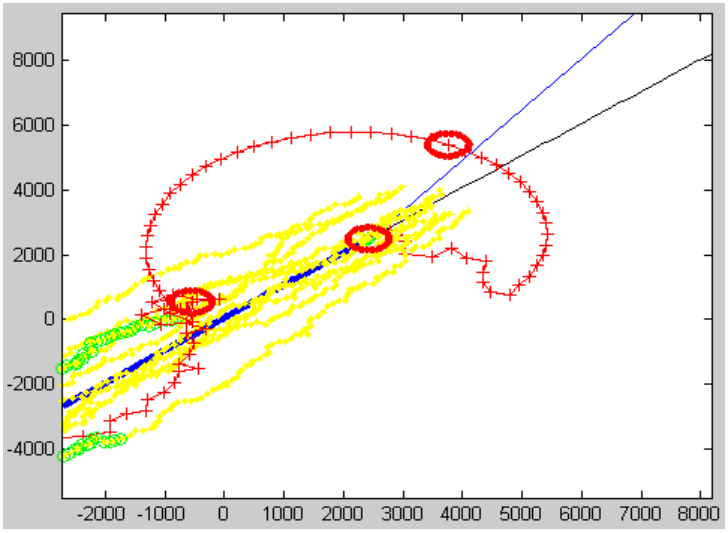


Ilustración 50: Ejemplo integrado, inicio de la simulación

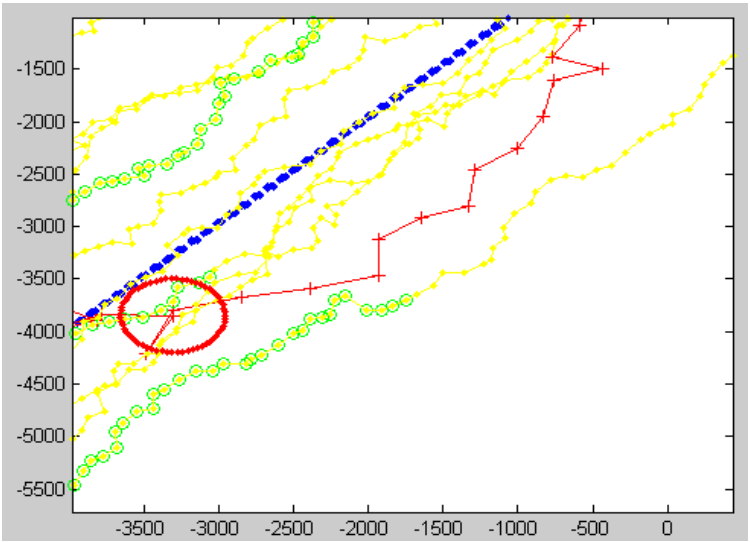


Ilustración 51: Ejemplo integrado, localización de los náufragos.

Ejemplo 14

Especificación/descripción del ejemplo. En este caso de uso intentamos probar que es lo que pasa si los náufragos no se mueven, hay que ver que nuestro sistema inteligente va descubriendo todos los náufragos y acaba recorriendo la elipse completa.

Breve descripción del resultado obtenido. Vemos como el UAV se está moviendo dentro de la elipse generada por los auto-valores y auto-vectores hasta descubrirlos todos. En este caso no se han movido los náufragos y vemos como el UAV se ha movido hasta localizarlos todos. El UAV inicia un movimiento en el centro de la elipse, marcado en verde en la ilustración, y empieza a moverse buscando a todos los náufragos, como estos no se mueven, el UAV no dispone de referencias sobre el movimiento de los náufragos. Va realizando la búsqueda, en rojo, quedando marcadas todas las posiciones por las que el UAV se ha ido moviendo. En este caso intentábamos ver que es lo que sucede en un caso extremo cuando el UAV debe realizar la búsqueda de los náufragos sin referencias al movimiento de estos.

Resultado del ejemplo. Gráficos obtenidos:

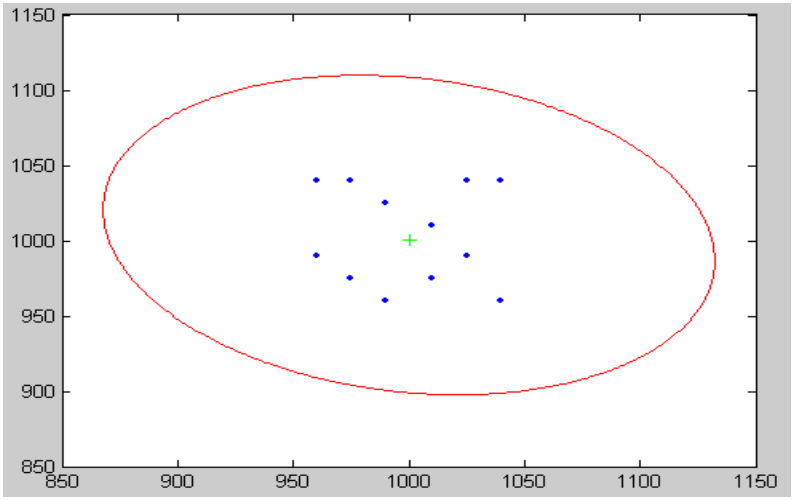


Ilustración 52: Ejemplo 14, posición inicial de los náufragos

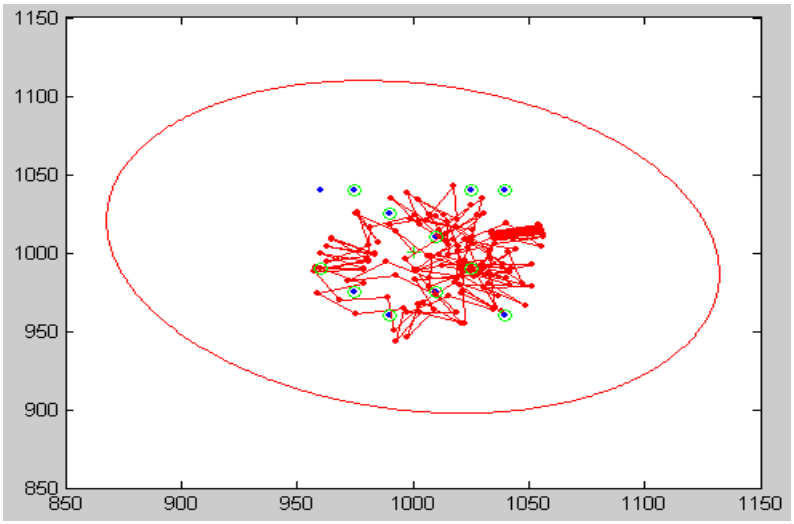


Ilustración 53: Ejemplo 14, detalle del movimiento del UAV

Ejemplo 15

Especificación/descripción del ejemplo. Imaginemos que nuestro conjunto de náufragos va en dirección contraria a la dirección del UAV, tenemos que ver que la red hace cambiar la dirección e intenta ir descubriendo a nuestros náufragos. Veremos un caso en el que la dirección de los náufragos es contraria, y el segundo caso donde es contraria y además independiente.

Breve descripción del resultado obtenido. En el primero de los casos hemos visto que primero el UAV se dirige hacia los náufragos y luego los persigue manteniendo el rumbo y descubriéndolos a todos. En el segundo caso como el comportamiento de UAV es el mismo pero no consigue localizar a todos los náufragos. En este caso el UAV comienza su movimiento en un punto alejado de la elipse y se mueve hacia los náufragos, estos comienzan el movimiento desde dentro de la elipse, dirigiéndose hacia el UAV. El UAV en un momento dado tiene que cambiar la dirección para poder perseguir a los náufragos y seguir su movimiento. La ilustración siguiente es la misma prueba pero haciendo que los náufragos sigan direcciones diferentes, vemos que en este caso el UAV sigue una dirección más o menos cercana a la media de las direcciones de los náufragos, también que existen algunos náufragos que no hemos sido capaces de localizar ya que no están señalados con el círculo verde.

Resultado del ejemplo. Gráficos obtenidos:

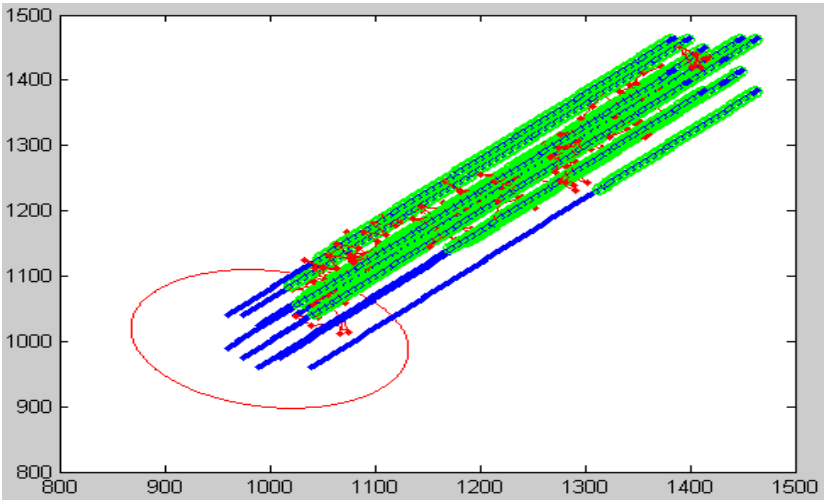


Ilustración 54: Ejemplo 15, movimiento del UAV con los náufragos en contradirección

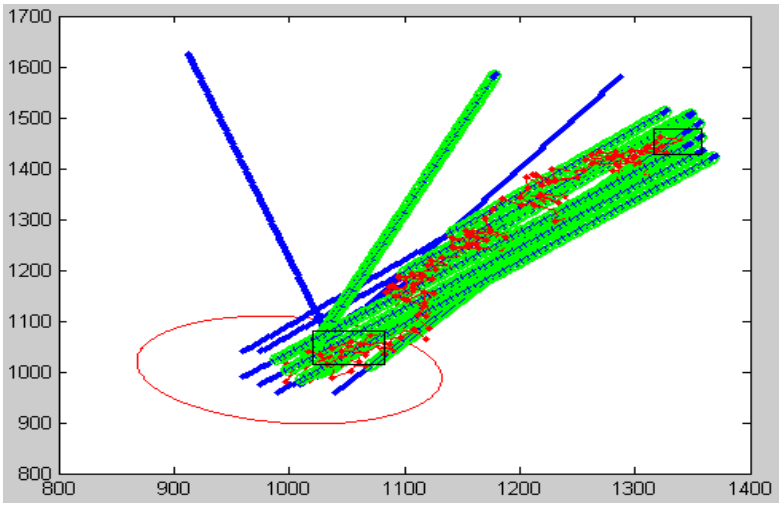


Ilustración 55: Ejemplo 15, movimiento del UAV con los náufragos en diferentes direcciones

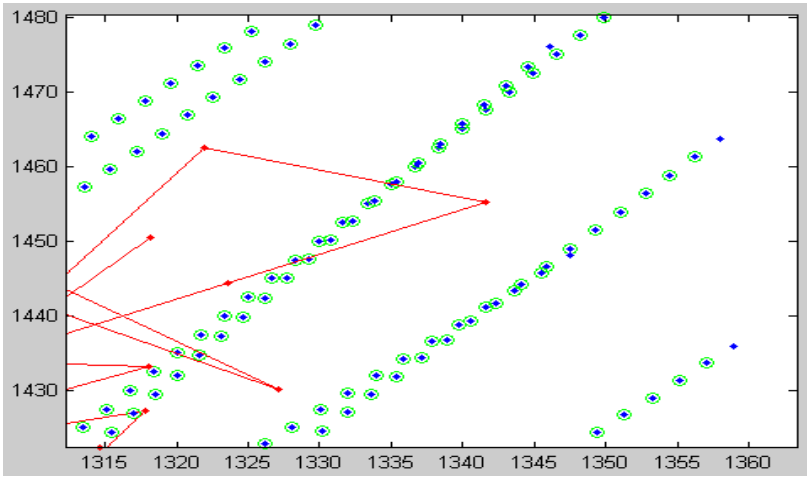


Ilustración 56: Ejemplo 15, movimiento del UAV posición final

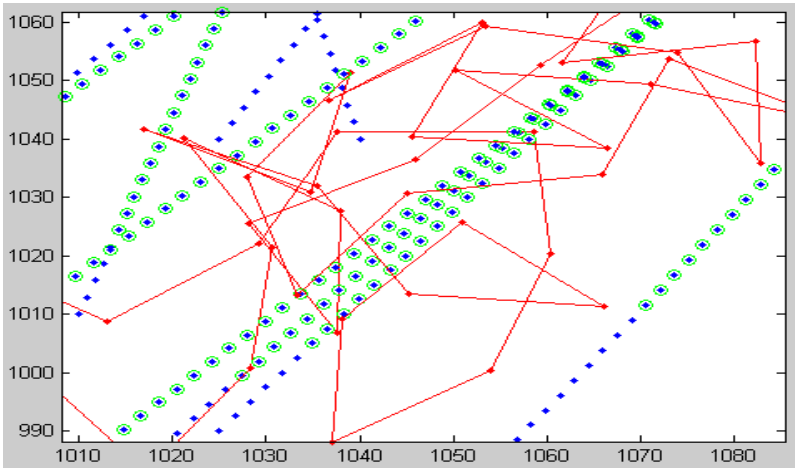


Ilustración 57: Ejemplo 15, movimiento del UAV, descubrimiento de los náufragos

Caso de uso F (Integración de la velocidad en el sistema)

En este caso de uso intentaremos ver la influencia de la velocidad en el sistema, para los ejemplos 16, 17 y 18, se ha añadido a la red neuronal de decisión un nuevo parámetro de salida, este parámetro es una relación numérica estimada a partir de la velocidad que en esos momentos llevan los náufragos, de esta manera el UAV adaptará la velocidad que lleva en estos momentos a un porcentaje de la velocidad que llevan los náufragos. La idea es doble, por un lado intentar añadir una estrategia que nos permita modificar la velocidad del UAV adaptándose a las circunstancias del entorno, y por otro lado ver como influye la velocidad en la localización de los náufragos, en este último aspecto veremos la importancia de la distancia de visión entre el UAV y los náufragos así como la relación directa que tiene esté parámetro con la velocidad a la que debe ir el UAV.

Necesitamos un nuevo modelo de red neuronal dentro del cerebro del UAV, en el siguiente gráfico podemos ver el nuevo modelo de red neuronal aplicado.

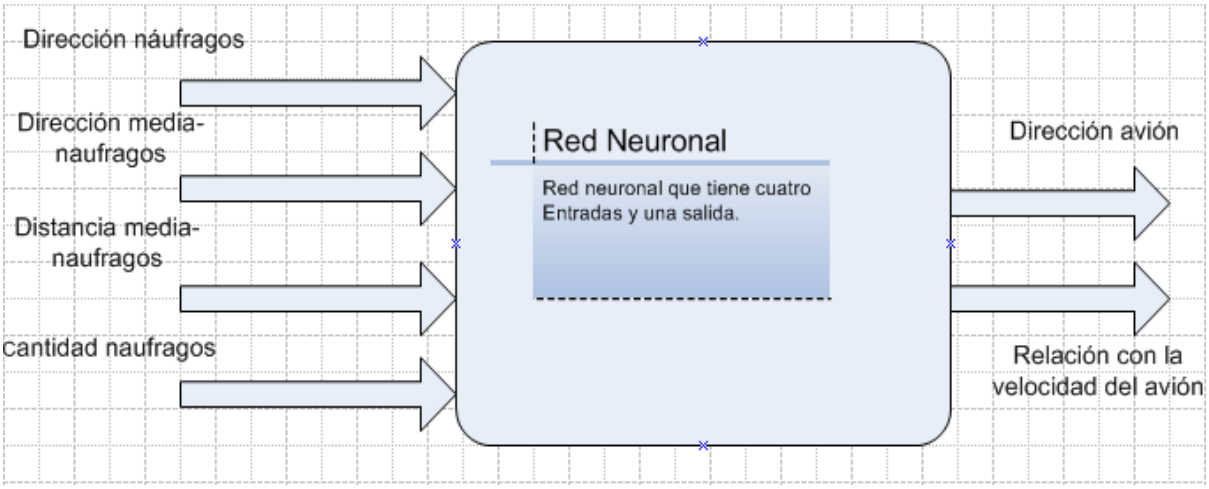


Ilustración 58: Esquema de la red neuronal de decisión con la relación de velocidad como parámetro de salida

Añadir la velocidad a nuestros ejemplos ha hecho que salte a primera línea un concepto, que hasta ahora no parecía tener mucha importancia, la distancia entre el UAV y los náufragos. Esto ha sido debido al algoritmo usado para entrenar la red neuronal con respecto a la velocidad, la distancia era uno de los parámetros principales. Cuanto mayor sea la distancia, mayor debe de ser nuestra velocidad, con la idea de acercarnos lo más posible, y cuanto menos sea la distancia igualmente deberá ser mayor la velocidad, con la idea de poder alejarnos y mantener una distancia relativa al entorno donde se encontrarán los náufragos. La distribución de la velocidad quedará de la forma siguiente, si estamos en un entorno cercano a donde deben de estar distribuidos los náufragos, la velocidad debe de aproximarse a la de los náufragos, y si estamos lejos de estos debemos de aproximarnos lo más rápidamente posible. En todo momento la salida de la red neuronal debe de ser un parámetro que multiplicado por la velocidad aproximada de los náufragos nos de la velocidad real del UAV.

Ejemplo 16

Especificación/descripción del ejemplo Seleccionamos un espacio de tiempo de cálculo de 10 s, dt=10, le añadimos el parámetro de la velocidad. La idea aquí es comprobar el comportamiento del avión alrededor de los náufragos y el tipo de curvas que describe comparándolo con el ejemplo 4 donde la velocidad era constante.

Breve descripción del resultado obtenido. En este caso se puede apreciar con más resolución, el detalle de la curva que realiza, en líneas generales es muy parecido al ejemplo 4, que es en el que nos basamos, pero debido a los cambios en la velocidad del UAV, la curva parece inestable a la hora de aproximarse al náufrago. En la segunda y tercera ilustración vemos claramente la trayectoria seguida por el UAV y como las distancias entre los puntos de cálculo que antes eran más homogéneas, ahora varían mucho de longitud, eso quiere decir que, sin perder el movimiento oscilatorio sobre los náufragos que teníamos en el ejemplo 4, si que el tener una velocidad variable afecta al detalle del comportamiento de la curva cuando estamos muy cerca del náufrago.

Resultado del ejemplo. Gráficos obtenidos:

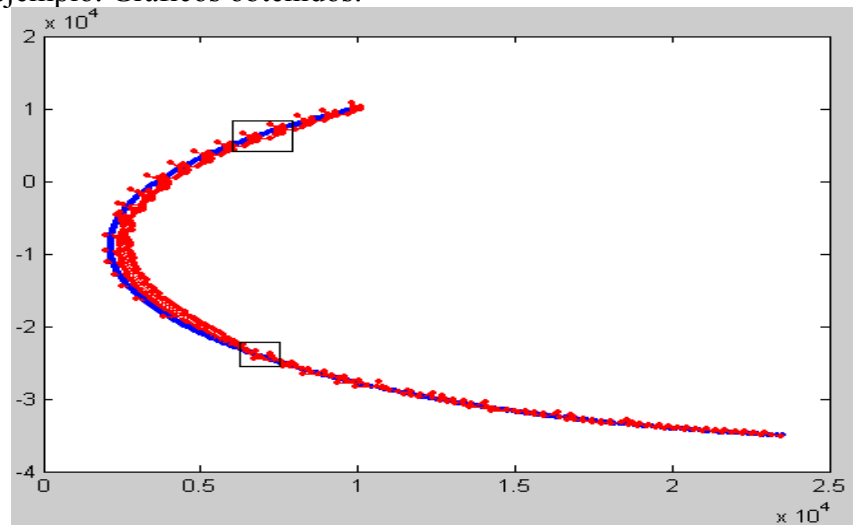


Ilustración 59: Ejemplo 16, curva de persecución

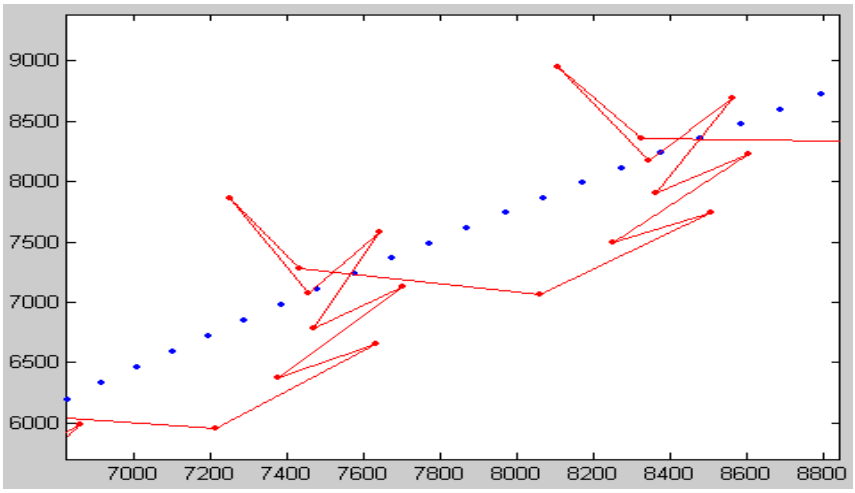


Ilustración 60: Ejemplo 16, detalle de la persecución I

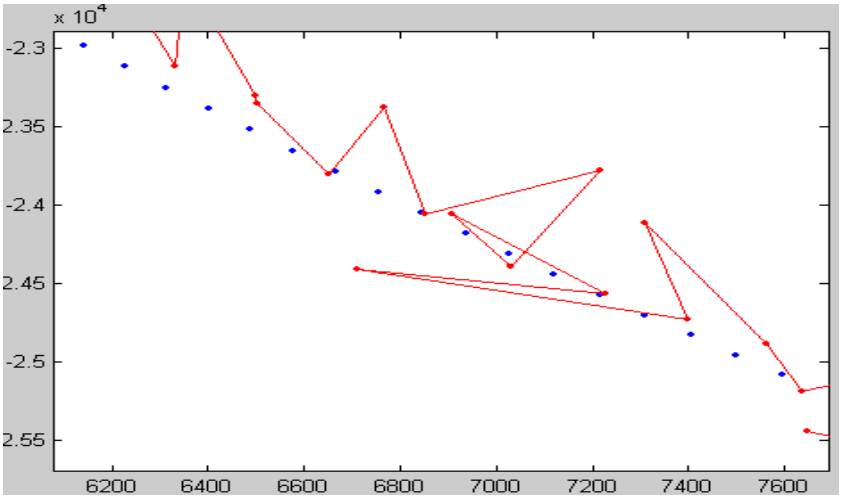


Ilustración 61: Ejemplo 16, detalle de la persecución II

Ejemplo 17

Especificación/descripción del ejemplo**.** Seleccionamos una curva de desplazamiento para los náufragos, ***x=sin(y)*** metros. Hemos cambiado la curva de desplazamiento de los náufragos, ahora el desplazamiento de los náufragos no está dirigido por los parámetros de velocidad y dirección, si no que están regidos por una ecuación y unos puntos elegidos al azar de esta. Este caso es idéntico al caso 9, solo que aquí podremos estudiar el verdadero valor de tener un parámetro velocidad que es dinámico.

Breve descripción del resultado obtenido. En este caso vemos como la red neuronal de decisión realiza la persecución de la curva de náufragos que le hemos creado, realiza el mismo movimiento que en el ejemplo 9 intentando seguir el movimiento oscilatorio de los náufragos, pero a diferencia del ejemplo 9 ya no tenemos un desfase de unos 10 m de media como teníamos antes. Como la velocidad es adaptativa el UAV sigue a la curva descrita por los náufragos, incluso llega a cortarla y se aleja si está muy cerca para intentar localizar a otros náufragos, Creo que en este ejemplo se ve claramente la importancia de los dos parámetros descritos en el ejemplo anterior, que son la distancia del UAV a la media y la velocidad adaptativa de la que le hemos provisto. En los gráficos siguientes es donde se ve como el UAV se acerca a la curva de los náufragos y como modifica la velocidad generando tramos muy largos, velocidad más alta y tramos muy cortos, velocidad más baja.

Resultado del ejemplo. Gráficos obtenidos:

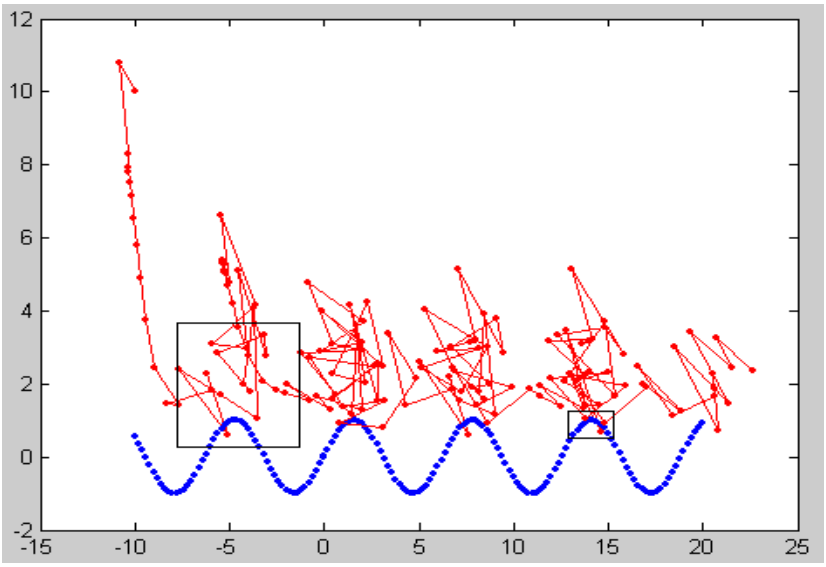


Ilustración 62: Ejemplo 17, curva de persecución

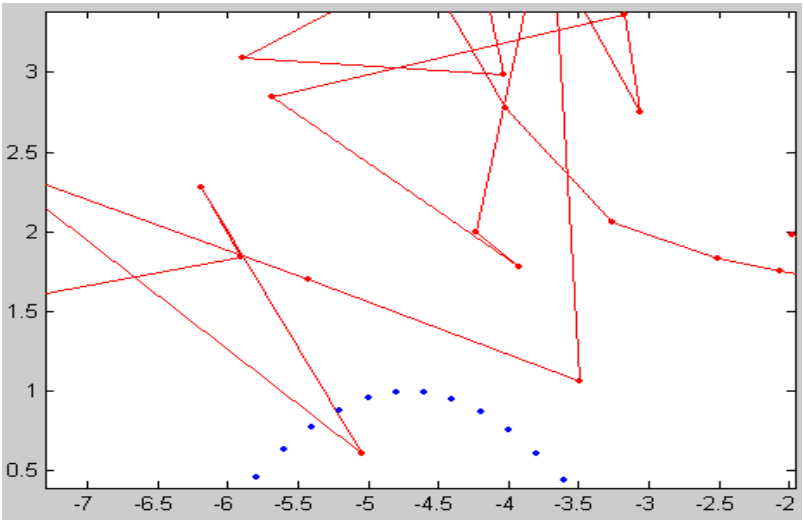


Ilustración 63: Ejemplo 17, detalle de la intersección entre los náufragos y el UAV

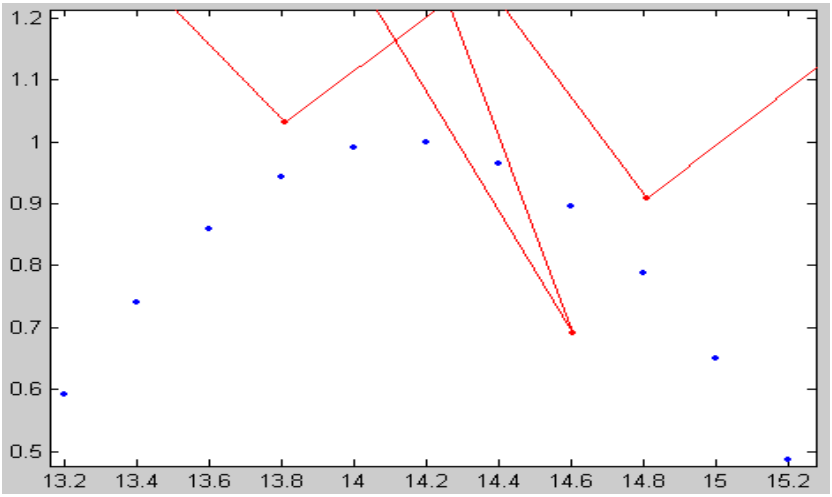


Ilustración 64: Ejemplo 17, detalle de la persecución

Ejemplo 18

Especificación/descripción del ejemplo**.** Este ejemplo es una escenificación del ejemplo 15 donde nuestro conjunto de náufragos va en dirección contraria a la dirección del UAV, tenemos que ver que la red hace cambiar la dirección e intenta ir descubriendo a nuestros náufragos. Observaremos y podremos comparar que es lo que ocurre en este ejemplo cuando la velocidad es dinámica.

Breve descripción del resultado obtenido. En este ejemplo hemos visto que primero el UAV se dirige hacia los náufragos, los náufragos comienzan su movimiento desde dentro de la elipse, y luego los persigue manteniendo el rumbo e intentando descubrirlos a todos, prácticamente mantiene el comportamiento que vimos en el ejemplo 15, aunque nos llevamos la decepción de que en este caso no llega a descubrir a todos los náufragos, solo al 80% de ellos, como vemos el comportamiento se parece mucho al anterior. Los náufragos que hemos conseguido localizar están señalados con un círculo verde. En la última ilustración, podemos ver el detalle de la curva de persecución y como afecta, la velocidad, al comportamiento de esta, abajo a la izquierda vemos como los intervalos son muy pequeños, y poco a poco según se van alejando los náufragos, estos intervalos se van haciendo más grandes.

Resultado del ejemplo. Gráficos obtenidos:

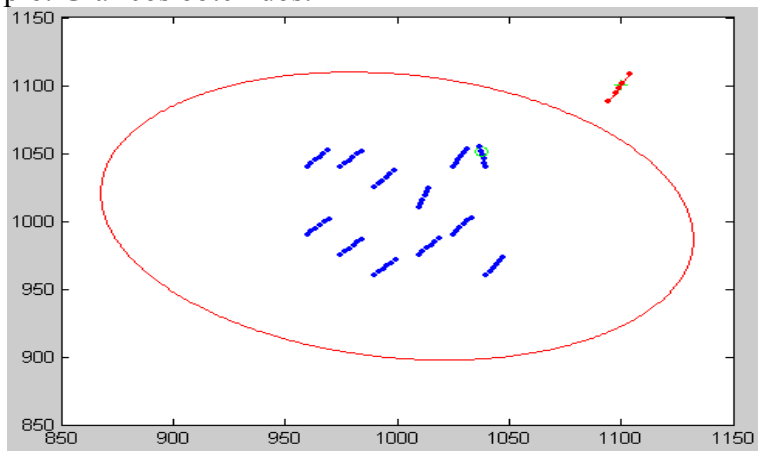


Ilustración 65: Ejemplo 18, el UAV se dirige hacia los náufragos

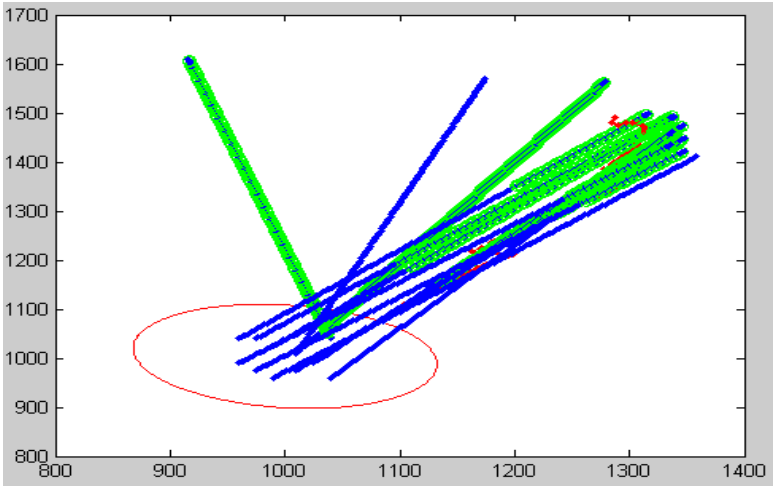


Ilustración 66: Ejemplo 18, comienzo de la curva de persecución

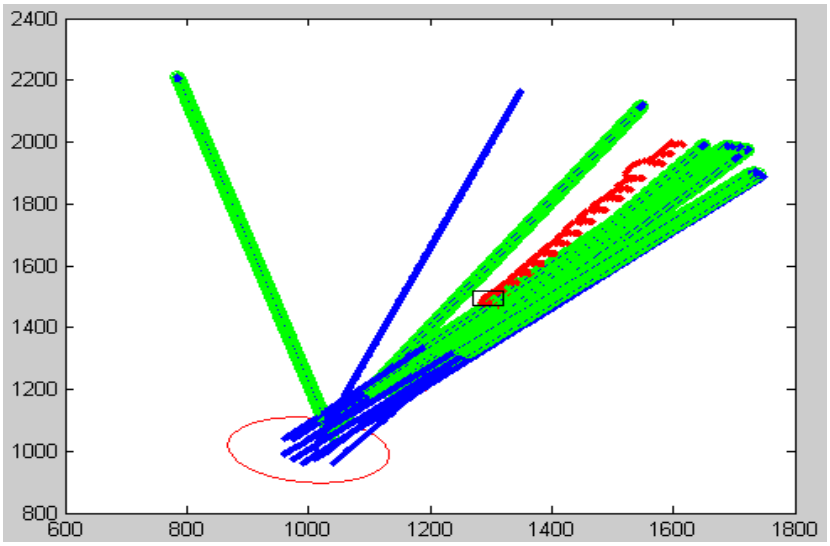


Ilustración 67: Ejemplo 18, curva de persecución

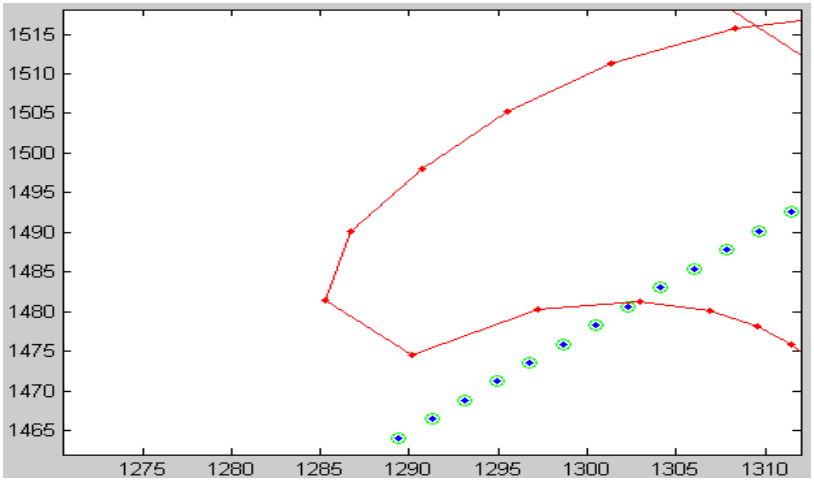


Ilustración 68: Ejemplo 18, detalle de la curva descrita por el UAV

Comentarios sobre los resultados y el uso del sistema inteligente

Tenemos que ver para cada caso de uso definido, si los resultados obtenidos son suficientemente fiables o por el contrario el sistema no tiene ningún valor experimental. Para ello hemos desarrollado 18 ejemplos en 6 casos de uso para probar cada uno de los componentes de nuestro sistema inteligente, también hemos intentado descubrir y analizar los parámetros que más pueden afectar a nuestro sistema, ya que con ellos es con los que tendremos que jugar para ajustar el sistema.

Una vez entrenadas las redes neuronales, tanto la encargada de predecir el movimiento de los náufragos, ***netMalla***, como la encargada de tomar las decisiones acerca de la dirección que el avión debe de tomar, ***direccionesNet***.
En los ejemplos 9 y 10 (correspondientes al caso de uso C, seguimiento de la trayectoria de los náufragos), intentamos comprobar si el comportamiento del sistema inteligente es modificado dependiendo del tipo de desplazamiento que realicen los náufragos, en dichas pruebas se puede

ver que el comportamiento no varía, por lo tanto debemos de suponer que nuestro sistema inteligente no tiene ninguna dependencia del movimiento de los náufragos, esto implica que puede ser usado independientemente del lugar y posición donde estemos. Creo que estos casos de uso demuestran la independencia del sistema respecto a las condiciones que puedan afectar sobre los náufragos.

Los ejemplos 11 y 12 (correspondientes al caso de uso D, predicción), tienen el valor de demostrar cómo funciona la red neuronal encargada de predecir, dada la posición de un náufrago, la dirección que tomará el náufrago y cuál será su velocidad en dicho punto. Hemos entrenado nuestra red neuronal con dos direcciones y velocidades diferentes y hemos visto como los resultados a la hora de predecir el movimiento son tanto en velocidad como en dirección bastante parecidos a lo entrenado.

Los ejemplos 1, 2 y 3 (correspondientes al caso de uso A, independencia de la posición náufragos/avión), se han definido para ver la independencia que tiene el sistema de la posición que tenga el UAV respecto de los náufragos. Hemos visto que no importa lo lejos, cerca, ni la posición de ninguno de nuestros elementos, el comportamiento no se modifica.

Los ejemplos 4, 5, 6, 7 y 8 (correspondientes al caso de uso B, influencia de la velocidad y el tiempo), se desarrollaron para ver qué influencia podían tener sobre el sistema tanto los tiempos de cálculo como la velocidad del UAV a la hora de realizar el movimiento de persecución. Vemos que lo que más afecta a la curva de persecución del UAV es el producto de la velocidad por el tiempo, cuanto más amplio es el tiempo o la velocidad más tendencia tiene el sistema a realizar un movimiento oscilatorio alrededor del punto medio de los náufragos. Estas pruebas demuestran que una buena elección de estos parámetros maximiza la posibilidad de éxito en las decisiones del sistema.

El ejemplo integrado (correspondientes al caso de uso E, problema integrado). Es el programa principal y demuestran el resultado de la interacción de ambas redes neuronales conjuntamente.

Los ejemplos 14 y 15 (correspondientes al caso de uso E, problema integrado), se han desarrollado para ver que comportamiento tiene la red neuronal cuando tiene que enfrentarse a casos extremos, como son, cuando no tiene ningún punto calculado, ver el movimiento que crea si se encuentra en el centro del elipsoide de los puntos o si los náufragos se dirigen hacia el UAV, viendo que decisión toma nuestra red neuronal. En todos estos casos el comportamiento fue correcto, cuando estaba en el centro del elipsoide, intentó recorrerlo para encontrar los náufragos y uso la información para ir descubriéndolos a todos, cuando los náufragos se dirigieron hacia el UAV este se dirigió hacia los puntos y en un momento dado cambió la dirección para perseguirlos.

Los ejemplos 16, 17 y 18 (correspondientes al caso de uso F, integración de la velocidad en el sistema), introducen una modificación en las gráficas que representan las trayectorias del UAV, haciendo que este acelere y baje su velocidad hasta adaptarla al entorno de una forma claramente oscilante alrededor del punto que representa la media de los náufragos.

CAPÍTULO 6. CONCLUSIONES Y TRABAJO FUTURO

Conclusiones

Este proyecto comenzó con la idea de evaluar la posible utilidad de las redes neuronales para la predicción de sucesos, en particular, aplicado al campo del salvamento marítimo, intentando predecir el movimiento de un conjunto de náufragos azotados por el movimiento del viento y de las mareas. Estas corrientes marinas que se generan en la superficie del mar, que dan lugar al oleaje, son las que debemos de predecir para poder simular, la localización de los náufragos después de pasado un tiempo. Pronto gracias a artículos como *[HontAgui], [AtsuDorv], [OzanSenkal], [ManGut] y [GomCarr],* donde se analizan diversos usos de las redes neuronales para la estimación de sismos, mareas y radiación solar, pudimos darnos cuenta del camino que deberíamos seguir, si queríamos tener éxito en nuestra empresa.

Hemos conseguido realizar un análisis de la forma como debe de plantearse un modelo a partir de redes neuronales, que nos permita realizar una predicción de la dirección movimiento de un náufrago, o de un conjunto de náufragos, a lo largo del tiempo, de forma que podamos, de forma sencilla, deducir su trayectoria y así componer el movimiento de este conjunto de náufragos. Esto nos permitirá construir una mancha, que a lo largo del tiempo, tendrá el comportamiento tanto de dispersión como de dirección del conjunto de náufragos estudiado.

En los modelos analizados (ver casos de uso definidos), vemos como la masa de náufragos se amplia y dirige en las direcciones que nuestra red neuronal nos entrega. Esta primera parte del proyecto deja una conclusión clara, las posibilidades de las redes neuronales en el campo de la predicción de elementos naturales tiene todavía un recorrido muy amplio y probablemente exitoso.

Otro de los elementos que hemos querido desarrollar en este trabajo es el uso de las redes neuronales para la toma de decisiones. Para ello hemos creado un sistema inteligente de ayuda, y le hemos dotado de su inteligencia, mediante el uso de redes neuronales, creando un modelo que permita tomar decisiones acerca de la trayectoria, que nuestro UAV debe de tomar para intentar moverse por la zona donde deberían estar situados nuestros náufragos, localizarlos, y usar la información de localización de cada uno de los náufragos para continuar la búsqueda.

Como se ha mostrado, en los diferentes casos de uso, hemos conseguido un éxito relativo, por un lado hemos conseguido nuestro objetivo, que es que nuestras redes neuronales realicen la predicción del movimiento más o menos exacto de los náufragos, y por otro lado que realicen movimientos alrededor de las posibles posiciones de estos, con la idea de maximizar las probabilidades de encontrarlos. El sistema parece capaz de una vez definida la zona de rastreo, y la posición de inicio, intentar localizar la zona donde podremos encontrar los náufragos, y realizar una serie de movimientos coordinados, para localizar todos los náufragos que se encuentran en ese momento en el mar. Indicamos un éxito relativo porque la velocidad indicada no es lo suficientemente fina, de hecho este ha sido el parámetro que más dolores de cabeza nos ha creado en todas nuestras pruebas, debido a que el sistema a veces tiende a adelantarse al

recorrido de los náufragos y luego tiene que realizar retornos demasiado forzados. Ajustes más finos al entrenar la red neuronal para que según se acerque a los náufragos, la velocidad sea cada vez más lenta acercándose lo más posible a la velocidad que estimamos que tienen los náufragos parece una mejor solución, pero no en todos los casos.

Por otro lado y para finalizar, hemos podido comprobar algunas de las cualidades más importantes de las redes neuronales. Por un lado el *aprendizaje adaptativo*, que queda demostrado en la capacidad de nuestra red neuronal de decisión a la hora de usar la experiencia para decidir cuál es la dirección que debe de tomar el avión. Por otro la realización de *operaciones en tiempo real*, ya que el movimiento de nuestro avión, y tanto las decisiones, como el movimiento de los náufragos, está haciéndose en tiempo real y en paralelo al movimiento del UAV. Por último la facilidad de inserción dentro de una tecnología existente, ya que las redes han sido entrenadas de forma independiente y luego insertadas dentro del sistema que controla el UAV.

Trabajo futuro

Cuando comenzamos cualquier proyecto, se suele, también, definir tanto un ámbito, como un alcance del proyecto, esto lo hacemos para evitar bloqueos con el exceso de información que se produce evitando pérdidas de tiempo innecesarias. Esta forma de trabajar hace que los proyectos tengan una fecha más o menos razonable de finalización, obligando a la gente a centrarse y preocuparse de los asuntos que están dentro de dicho ámbito. En este apartado haremos precisamente lo contrario, esto es, preocuparnos tanto de los elementos que no hemos podido acabar, como de elementos que no estaban en el ámbito principal del proyecto, pero que quizá sea importante tratar en proyectos futuros.

Sólo nos hemos centrado en el uso de las redes neuronales feed-forward, pero entrenar otros modelos y comprobar su comportamiento es algo que sería interesante de realizar. Hacer esto nos llevaría mucho tiempo pero nos permitiría profundizar y alcanzar un mayor conocimiento de las redes neuronales y que podemos hacer con ellas. En trabajos posteriores sería ideal usar una parametrización mayor asociada a los modelos neuronales usados y la posibilidad de usar otros modelos diferentes.

Por otro lado y centrándonos más en el ámbito del trabajo, sería interesante ver un mapa real de mareas de forma que pudiéramos realizar y optimizar la aplicación con un conjunto de modelos más realistas que los aquí tratados. Todos los ejemplos realizados han sido sobre modelos simulados donde las posibilidades de creación de situaciones extrañas o impredecibles no están contempladas, por ejemplo, que pasaría si hubiera una tormenta y al modelo de mareas real le tuviéramos que añadir ese tipo de desviación. La posibilidad de realizar pruebas sobre el terreno, y con mapas más reales, le daría a nuestro modelo un empujón para un desarrollo más amplio del uso de sistemas inteligentes con redes neuronales y demostraría las grandes habilidades de estas.

Según hemos ido realizando los diferentes casos de uso hemos podido comprobar la importancia de la velocidad y del tiempo en la veracidad de los resultados obtenidos, de hecho lo que más ha llegado a afectar es el producto de ambas variables. Por ello hemos llegado a la conclusión, ya que el tiempo de muestreo no es fácilmente modificable, que variar la velocidad del UAV permite que el funcionamiento sea más preciso, deberíamos estudiar más modelos de entrenamiento para optimizar el sistema de control de la velocidad que la red neuronal de decisión proporciona. Aun así queda demostrado que un buen control de la velocidad permitirá localizar de forma más óptima a los náufragos, optimizando la curva de persecución obtenida.

Como núcleo de una investigación futura, sería interesante realizar un estudio de los algoritmos de entrenamiento de la red de decisión, no solo fijándonos en los parámetros que afectan a ésta, sino en todas las posibles casuísticas con las que podríamos entrenar nuestra red, a fin de hacerla mucho más inteligente a la hora de tomar ciertas decisiones, mejoras no solo en el entrenamiento de la velocidad, sino también en la dirección que se debe de tomar.

Por otro lado, en el entrenamiento de la red neuronal de predicción se planea la posibilidad de separar ambos componentes, el de corrientes marinas y el del viento, añadiendo nuevos parámetros de salida a dicha red neuronal, e incluso añadiendo otros factores. Esto requerirá un entrenamiento mucho más complejo pero añadirá mayor precisión a la predicción del movimiento de los náufragos.

En los artículos *[HontAgui], [AtsuDorv], [OzanSenkal], [ManGut] y [GomCarr]* se demuestra que el futuro de los modelos predictivos pasa por modelos de redes neuronales. No sería extraño que veamos modelos neuronales en futuros artículos sobre predicciones meteorológicas, terremotos, erupciones volcánicas y otros tipos de fenómenos naturales, ya que cada día se demuestra más su capacidad para acertar en las predicciones. Es posible que el futuro de la inteligencia artificial, automatización de procesos industriales, etc.…, esté en estos modelos tecnológicos.

El tiempo puede ser un nuevo parámetro a introducir en la red neuronal de toma de decisiones ya que nos hemos encontrado con algunas casuísticas que pueden llegar a bloquear la red neuronal y modificando algún parámetro, como el tiempo, del sistema vuelve a responder adecuadamente. Un estudio profundo de cómo puede afectar el tiempo que ha pasado en la toma de decisiones puede llegar a solucionar problemas de bloqueos temporales del sistema.

También debemos incluir en futuras versiones, un sistema de control de errores más preciso, que vaya más allá de controlar el tiempo de búsqueda y ofrecerle visualmente al usuario el movimiento de los náufragos. Quizá un sistema que compare las predicciones con los resultados reales de la posición de los náufragos recogidos por el UAV y el tiempo que llevamos sin descubrir nuevos náufragos, permitiría un control más exacto que recomendara al UAV cuando se debe de dejar de buscar más náufragos ya que las probabilidades de encontrarlos son bastante pequeñas.

BIBLIOGRAFÍA

[AngKanas] Neural network linear forecasts for stock returns. Angelos Kanas.

[AnuDosh] Aircraft Position Prediction Using Neural Networks (2005). Anuja Doshi.

[AguilarQuis] Algoritmos genéticos en el aprendizaje de redes neuronales artificiales. Ramiro Aguilar Quispe. Universidad de Salamanca.

[AtsuDorv] Solar radiation estimation using artificial neural networks (2002). Atsu S.S. Dorvlo, Joseph A. Jervase, Ali Al-Lawati.

[BusRes] Links de búsqueda y rescate y protocolos de naufragios.
[1] http://www.worldlingo.com/ma/enwiki/es/Search_and_rescue
[2] http://en.wikipedia.org/wiki/Search_and_rescue
[3] http://en.wikipedia.org/wiki/International_Convention_for_the_Safety_of_Life_at_Sea
[4] http://en.wikipedia.org/wiki/International_Search_and_Rescue_Advisory_Group
[5] http://en.wikipedia.org/wiki/Search_and_Rescue_Optimal_Planning_System_(SAROPS)
[6] http://www.imo.org/conventions/contents.asp?doc_id=653&topic_id=257
[7] http://ieeexplore.ieee.org/Xplore/login.jsp?url=http://ieeexplore.ieee.org/iel5/4736903/4736904/04737118.pdf%3Farnumber%3D4737118&authDecision=-203
[7] http://sciencestage.com/d/5549217/communication-mechanism-of-search-and-rescue-at-sea-intelligent-decision-support-system-based-on-rmi.html

[ChengHengSiew] Detecting macroeconomic phases in the Dow Jones Industrial Average time series (2009). Jian Cheng Wonga, Heng Liana and Siew Ann Cheong.

[DecisionSupport] http://en.wikipedia.org/wiki/Decision_support_system

[DemoMPIDR] Demografy MPIDR WORKING PAPER WP 2006-026 (2006). Anna Matysiak Beata Nowok. http://www.demogr.mpg.de/papers/working/wp-2006-026.pdf

[DisProb] Links de distribuciones de probabilidad
[1] http://es.wikipedia.org/wiki/Distribuci%C3%B3n_de_Cauchy
[2] http://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal

[ExpertSystem] http://en.wikipedia.org/wiki/Expert_system

[Forecast] http://en.wikipedia.org/wiki/Forecasting

[ForHag] Proceedings of the International Joint Conference on Neural Networks. Foresee and Hagan,

[Fossen02]Marine control systems, Marine cybernetics (2002), Fossen, T.I.

[Fossen94] Guidance and control of ocean vehicles (1994), Fossen, T.I, Pp. 173-175, John-Wiley &Sons.

[GioPat06] Time series analisys of atmosphere pollution data using artificial neural networks techniques (2006). Giovanni Salini Calderón, Patricio Pérez Jaraz.

[GomCarr] Wave forecasting at the Spanish coasts, M. Gomez Lahoz and J. C. Carretero Albiach,

[Hausdorff] Hausdorff clustering of financial time series (2006). Nicolas Basaltoa, Roberto Bellottib, Francesco De Carlob, Paolo Facchic, Ester Pantaleob, and Saverio Pascaziob.

[HilMart95] Hilera J.R. y Martínez V.J. (1995). Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones. Ed. RA-MA.

[HontAgui] Mapas de radiación solar en la provincia de Jaén (2004). L. Hontoria, J. Aguilera, G. Almonacid. P. Zufiria.

[HorPag] Predicción De Series Temporales Usando Redes Neuronales: Un Caso De Estudio, Horacio Paggi.

[HushHorne92] An overview of neural networks (1992). Part I: static networks. Informática y automática, vol. 25. Hush D.R. y Horne B.

[JuCamSan06] Forecasting Time Series with Neural Networks, an Application to the Colombian Inflation (2006). Revista Colombiana de Estadística. Juan Camilo Santana.

[KaloNeoPas] Wind Speed Prediction Using Artificial Neural Networks (1999), Soteris Kalogirou, Costas Neocleous, Stelios Pashiardis and Christos Schizas. Higher Technical Institute. Department of Mechanical Engineering.

[KimotoAsaYoda] Stock Market Prediction System with Modular Neural Networks (1990). T Kimoto, K Asakawa, M Yoda. IEEE Neural Networks Council

[Kornel04] Neural Networks for Time Series Prediction, Artificial Neural Network (2004). Kornel Laskowski.

[KroseVaS96] An introduction to Neural Networks (1996). Kröse B. and Van der Smart P.

[Mackay] Neural Computation (1992). Mackay.
http://www.cs.toronto.edu/~mackay/BayesNets.html

[ManGut] Aplicación de redes neuronales para el pronóstico de sismo. Luis Carlos Manrique Ruiz, Edgar Gutiérrez Franco.

[MarkPred] http://www.learnartificialneuralnetworks.com/stockmarketprediction.html

[Matlab7] Aprenda Matlab 7.0 como si estuviera en primero (2005). Javier García de Jalón, José Ignacio Rodríguez, Jesús Vidal.

[MatlabNNT] Neural Network Toolbox User's Guide (2008). Howard Demuth, Mark Beale, Martin Hagan.

[MillOst06] Predicción mediante redes neuronales artificiales de la transferencia de masa en frutas osmóticamente deshidratadas (2006). Millan Trujillo, Félix Rafael y Ostojich Cuevas, Zoitza.

[MONTECARLO]
[1] http://en.wikipedia.org/wiki/Monte_Carlo_method
[2] http://en.wikipedia.org/wiki/Deterministic_algorithm
[3] http://en.wikipedia.org/wiki/Monte_Carlo_methods_in_finance
[4] http://ideas.repec.org/p/bdi/wptemi/td_723_09.html
[5] http://teorica.fis.ucm.es/programas/MonteCarlo.pdf
[6] http://www.riskamp.com/files/RiskAMP%20-%20Monte%20Carlo%20Simulation.pdf

[NNWPREDICTION] Predicción mediante redes neuronales que simulan series temporales. http://www.emis.de/journals/RCE/V29/V29_1_77Santana.pdf

[NNWTechPred] Neural network techniques for financial performance prediction: integrating fundamental and technical analysis, Management Information Science (2003). College of Business Administration, California State University.

[NPerAheadfore] http://www.uc.edu/sashtml/ets/chap14/sect55.htm

[OzanSenkal] Estimation of solar radiation over Turkey using artificial neural network and satellite data (2008). Ozan Senkal,. Tunkal Kuleli.

[Patterson96] Artificial Neural Networks. Theory and Applications, Prentice Hall (1996). Patterson D.W.

[PlanRescate]
[1] http://sinabaco.proxecto.net/
[2] http://www.frf.usace.army.mil/cgi-bin/wis/atl/atl_main.html
[3] http://www.stormingmedia.us/75/7519/A751934.html
[4] http://www.informaworld.com/smpp/content~content=a919112468&db=all

[PredWeather] Numerical weather prediction
http://en.wikipedia.org/wiki/Numerical_weather_prediction

[ProcEsto] Links sobre procesos estocásticos
http://en.wikipedia.org/wiki/Stochastic_process

[QLEARNING] Qlearning,
[1] http://people.revoledu.com/kardi/tutorial/index.html.
[2] http://en.wikipedia.org/wiki/Reinforcement_learning
[3] http://en.wikipedia.org/wiki/Probably_approximately_correct_learning
[4] http://en.wikipedia.org/wiki/Markov_decision_process
[5] http://people.revoledu.com/kardi/tutorial/index.html
[6] http://www.research.ibm.com/infoecon/paps/html/ijcai99_qnn/node4.html
[7] http://code.google.com/p/opennero/wiki/QLearning

[RanRobJohn] Optimization of Neural Networks: A Comparative Analysis of the Genetic Algorithm and Simulated Annealing, Randall S. Sexton, Robert E. Dorsey, and John D. Johnson.

[RanRobJohn99] Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing (1997). Randall S. Sextona, Robert E. Dorseyb and John D. Johnsonc.

[RichProv] Identifying Temporal Patterns for Characterization, Richard J. Provinelli.

[SebasGue06] Uncertainty in predictions of oil spill trajectories in a coastal zone (2006). P. Sebastiao and C. Guedes Soares. ***Journal of Marine Systems*** *Volume 63, Pages 257-269.*

[UAV] http://en.wikipedia.org/wiki/Unmanned_aerial_vehicle

APÉNDICES

Instalación y uso

Vamos a describir cómo podemos instalar y usar los programas que se han creado para la predicción del movimiento de los náufragos. Necesitamos una versión de Matlab, superior a la 7 si queremos hacer correr los programas. Estos programas todavía no están preparados para ejecutarse fuera del entorno de simulación de Matlab aunque es una de las cosas que haremos en el futuro.

Lo primero que hay que construir es la superficie sobre la que queremos realizar nuestra predicción, una vez que tenemos definida dicha superficie deberemos encontrar un mapa, o varios mapas en diversas épocas, de dicha superficie donde tengamos los datos de velocidad y dirección del viento y de las corrientes marinas en dicho área. Con estos datos podemos simular y entrenar la red que nos realizará la predicción de la dirección y la velocidad del náufrago.

Por otro lado deberemos crear también la red que realiza la predicción de la dirección que debe de tomar nuestro UAV, el algoritmo de entrenamiento se puede cambiar y mejorar retocando el programa direccionesNNTool.m.

Finalmente ejecutaremos el programa principal (main). En el programa principal los valores principales sobre los que deberemos de actuar son, la velocidad del UAV, el punto de inicio de los náufragos, punto de inicio del UAV, radio de visión del UAV. Con estos datos iniciaríamos la simulación del movimiento a lo largo del tiempo, en estos momentos está definido a 60 segundos, aunque se puede cambiar. El programa te dará la posición de contacto entre los náufragos y el UAV y luego irá simulando la persecución que hace este sobre los náufragos hasta que los vaya recogiendo a todos o no.

Implementación de la herramienta

En este apartado vamos a desarrollar todos los programas que se han creado para las pruebas y configuración de los componentes de nuestro sistema inteligente. Intentaremos dar el código del programa y una breve explicación del funcionamiento de este.

Programa principal (main), donde cargamos las redes neuronales que anteriormente hemos entrenado y las introducimos dentro del cerebro de nuestro sistema inteligente. La primera red sirve para predecir la posición de los náufragos y la segunda red es el núcleo del cerebro que toma decisiones a la hora de perseguir a los náufragos. El programa AllNaufragos0.m es el programa principal que tenemos que ejecutar para iniciar la simulación

Programas de entrenamiento de la red. Estos programas generan dos redes neuronales direccionesNet.mat y netMalla.mat. Para el entrenamiento de la red que genera el movimiento de los náufragos, hemos construido una interface gráfica con los datos necesarios para el entrenamiento. Es programa es interfaceNN.m.

Por otro lado está el algoritmo de entrenamiento de la red de toma de decisiones, *direccionesNNTool.m*. El código del algoritmo de entrenamiento puede ser cambiado para modificar la forma de tomar decisiones de la red neuronal, hay que tener cuidado con el código de entrenamiento ya que puede cambiar el comportamiento de todos los programas. Existe un programa de entrenamiento específico, *direccionesNNTool.m,* para añadir la velocidad a la red neuronal.

Programa para simular la red neuronal creada, *SimularRedDistancias.m*, nos permite realizar una simulación de cómo la red neuronal va tomando decisiones y hace que se mueva el UAV. Este programa se ejecuta de forma independiente al programa principal y necesita del entorno de desarrollo de Matlab.

Programa para simular la red neuronal creada, *SimularRedMalla.m*, nos permite realizar una simulación de cómo la red neuronal va prediciendo el movimiento que deben realizar los náufragos, su velocidad y dirección en un punto dado. Este programa se ejecuta de forma independiente al programa principal y necesita del entorno de desarrollo de Matlab.

Programa para simular el comportamiento del sistema inteligente, *SimularSistemaExperto.m*, nos permite realizar una simulación de cómo la red neuronal se integra y va tomando decisiones que hace que se mueva el UAV. Este programa se ejecuta de forma independiente al programa principal y necesita del entorno de desarrollo de Matlab.

Uso de los programas de prueba, hemos desarrollado tres programas que nos permiten probar por separado las características de nuestro sistema. En los siguientes apartados desarrollaremos el uso de dichos programas.

SimularRedMalla, encargado de probar la malla que realiza la predicción del movimiento de los náufragos. Modificaremos las variables para la realización de las pruebas. Las variables son las siguientes: Las variables N y c son la cantidad de náufragos y la cantidad de elementos inicial, en principio son iguales (1 náufrago). La variable dt es el tiempo que usamos para nuestro cálculo (30). La variable nhr es el número de horas de simulación (4). Las variables (x,y) contienen el punto de inicio del recorrido de los náufragos (10000,10000). La variable Simulacion contiene el recorrido de los náufragos que va generando la red neuronal.

SimularRedDistancias, encargado de probar la red que toma la decisión acerca de la dirección a seguir. Las variables son las siguientes: La variable dt es el tiempo que usamos para nuestro cálculo (30). La variable nhr es el número de horas de simulación (4). Velocidad de los náufragos, vm, dirección del naufragio Rm, dirNaufragio. La variable mediaxy contiene la posición actual de la media, avionxy la posición actual del avión. Las variables (x,y) inicializan la posición de los náufragos y xav y yav la posición del avión. distVision inicializa la distancia a la que el avión puede ver a los náufragos. mpslocal es la velocidad del avión. Las variables de entrada de la red neuronal, dirMedia dirección donde se encuentra la media respecto al avión, dirNaufragio dirección hacia la que se dirigen los náufragos, pPuntosAcertados porcentaje de náufragos encontrados, dPorDistancia distancia a la que se encuentra el avión de los náufragos. Existe una modificación de este programa llamado *SimularRedDistanciasFunciones*, que es el encargado de realizar una simulación de nuestra red de distancias, aplicada a una función específica que se encarga de simular el movimiento de nuestros náufragos.

Como consejos para las pruebas con SimularRedDistancias, debemos tener especial cuidado con ciertos parámetros, los cuales son muy importantes, como dt, vm, pPuntosAcertados, mpslocal,

distVision. Representan el tiempo, la velocidad de los náufragos, el porcentaje de puntos acertados, velocidad del avión y distancia de visión del avión.

SimularSistemaExperto, encargado de probar el algoritmo que usa el sistema inteligente y la red neuronal de decisión en un entorno controlado. Las variables son las siguientes: Las variables x,y contienen el punto de inicio del recorrido de los náufragos (10000,10000). La variable dt es el tiempo que usamos para nuestro cálculo (30). La variable mediaxy contiene la posición actual de la media, avionxy la posición actual del avión. Las variables de entrada de la red neuronal, dirMedia dirección donde se encuentra la media respecto al avión, dirNaufragio dirección hacia la que se dirigen los náufragos, pPuntosAcertados porcentaje de náufragos encontrados, dPorDistancia distancia a la que se encuentra el avión de los náufragos. La variable distVision inicializa la distancia a la que el avión puede ver a los náufragos. La variable mpslocal es la velocidad del avión. La variable posicionNoDescubiertos es la lista de los náufragos que todavía no se han descubierto, posicionDescubiertos es el conjunto de los náufragos que se han descubierto.

Como consejos para las pruebas con SimularSistemaExperto, debemos tener especial cuidado con ciertos parámetros, los cuales son muy importantes, dt, vm, pPuntosAcertados, mpslocal, distVision. Representan el tiempo, la velocidad de los náufragos, el porcentaje de puntos acertados, velocidad del avión y distancia de visión del avión.

Por otro lado existen en las carpetas **decisiones-velocidad** y **predicciones-rna-complejo** un conjunto de programas de prueba que nos permiten entrenar ambas redes neuronales con un formato distinto al que se ha desarrollado hasta ahora. La red malla se va a poder simular, con un algoritmo, que permite modificar el comportamiento de la velocidad y la dirección por sectores dentro de la malla, es decir que no vamos a tener la misma dirección y velocidad en todos los puntos de la red, por otro lado, también podremos probar y entrenar la red de distancias para que esta nos de un parámetro que multiplicado por la velocidad media de los náufragos nos diga la velocidad que debe de llevar nuestro UAV. Esto quiere decir que la velocidad del UAV se irá modificando dependiendo de las condiciones existentes entre el UAV y los náufragos, condiciones como la distancia, la cantidad de náufragos descubiertos, velocidad de estos, etc.

Programas para el cálculo de distancias y dibujos de los datos. Estos programas son ayudas que nos permite realizar los cálculos más comunes, necesarios para preparar los datos de entrada para las redes neuronales.

Dibujamos la elipse generada por la dispersión de los puntos, dibujoElipse.m. Estas funciones permiten calcular los puntos de la elipse y dibujarla a partir de los auto-valores y auto-vectores de la matriz de puntos actuales. Dibujo de los puntos de los náufragos que se han descubierto y de los que todavía no se han descubierto, dibujaNaufragos.m, dibujaNaufragosDescubiertos.m.

Funciones para el cálculo de las distancias y posiciones, elipseNaufragos.m, pendienteRecta.m, distBetw2Puntos.m, distBetwRectaPunto.m, distPuntoTrayectoria.m. Estas funciones nos permiten de una forma básica calcular tanto las pendientes de las direcciones que toman nuestros náufragos y el UAV como las distancias que hay entre ellos en todo momento, haciendo posibles los cálculos como por ejemplo si un náufrago está en el ángulo de visión del UAV en un momento determinado.

Matlab Neural Network

Intentaremos describir cómo funciona la Toolbox de Neural Network en Matlab. Queda fuera de este documento la Toolbox de Matlab de desarrollo gráfico, de Cálculo y uso de Matrices. Nos centraremos en el diseño y modelado de redes neuronales de Matlab y en especial de la red neuronal Backpropagation (feedforward) que es la que hemos estado usando durante este documento.

Primero hay que comprender el diagrama de flujo de la Neural Network Toolbox de Matlab. Es bastante sencillo y a la vez potente.
Creación de una estructura de red, **newff**.

```
net = newff(interv,[10, 2],{'purelin', 'purelin'},'trainbr','learngd', 'mse');
```

Entrenamiento de la red, **adapt** o **train**.

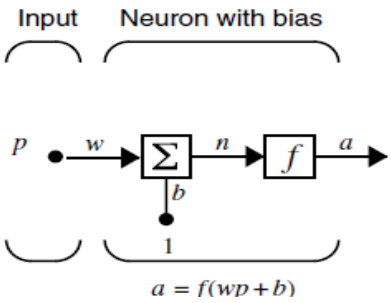
```
[red,tr] =train(net,x,z);
```

Simulación de la red, **sim**.

```
[y,Pf,Af,E,perf] = sim(netT,x');
```

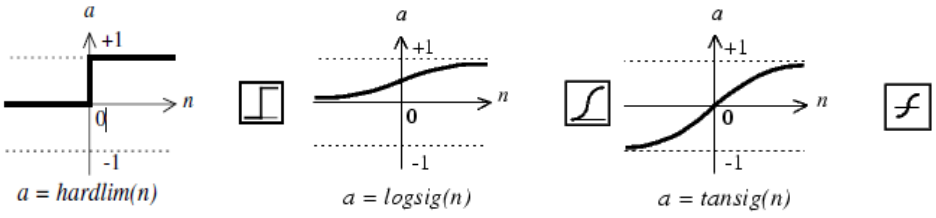
Primero empezaremos con una introducción a las redes neuronales con Matlab y más adelante pasaremos a definir el uso de la *Toolbox* para la backpropagation.

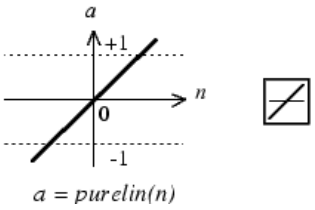
Modelo neuronal:
La neurona simple viene representada en Matlab con el siguiente gráfico.



Donde el escalar p es lanzado a través de la conexión, multiplicado por el escalar *w* formando el escalar *wp*. Se le suma el *bias* para producir *wp+p* y se suman todas las entradas de la neurona para formar el escalar n después, aplicamos la función de transferencia *f* para obtener como resultado el escalar *a*, *a=f(n)*.

Existen principalmente cuatro funciones de transferencia, **hardlim**, **purelin**, **logsig** y **tansig**. Como podemos ver a continuación.





Si queremos ver en el tratamiento que hace de ellas Matlab bastará con poner en la consola:

```
n = -5:0.1:5;
plot(n,hardlim(n),'c+:');
plot(n,logsig(n),'c+:');
plot(n,tansig(n),'c+:');
plot(n,purelin(n),'c+:');
```

Y podremos ver representada las funciones de transferencia. Toda función de Matlab puede ser representada de la misma forma que hemos representado a esta, con la función *plot* dibujamos la función. Normalmente usaremos solo funciones de transferencia derivables.

Arquitectura de una neurona con n-entradas:
Una neurona con n-entradas se representa de la siguiente forma:

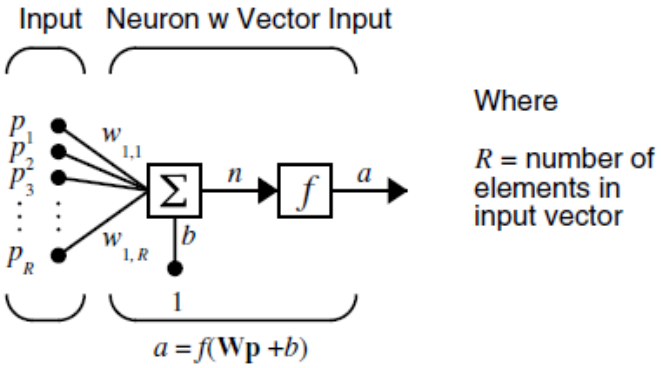
p1, p2, p3…. pR con R pesos w1, w2, w3…. wR

Donde los *R* pesos son multiplicados por las *R* entradas, añadiéndole el **bias** *b*. Esto generará la entrada *n* al que luego le pasaremos la función de transferencia.

n= w1,1*p1 + w1,2*p2 + ……+ w1,R*pR + b

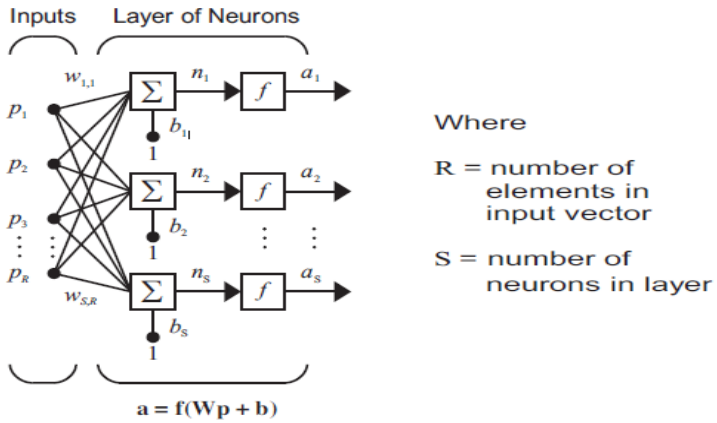
Representado en modo matricial. $n = W*p + b$

La estructura quedará como sigue. Siendo *a=f(n)*

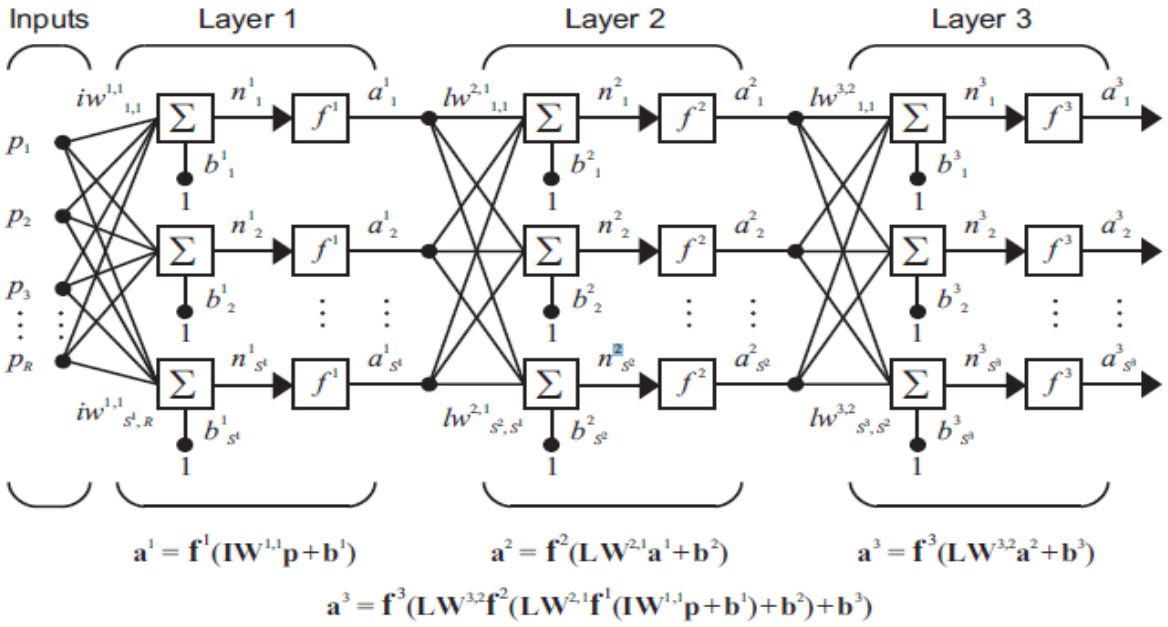


Con esto ya tenemos una idea de cómo es la estructura de una neurona en Matlab.

Arquitectura de una red neuronal:
Una red neuronal debe de contener más de una neurona, estas deben de agruparse en capas. Matlab agrupa y tiene una forma de representar los grupos de neuronas asociadas a sus correspondientes capas.

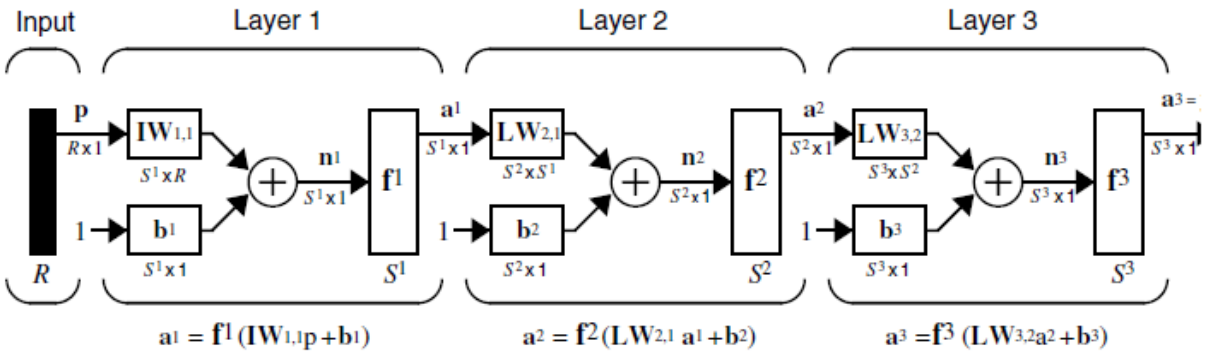


Pero las redes neuronales no tienes solo una capa si no que pueden llegar a tener n-capas. En Matlab esto queda representado de esta forma.



Donde *(p1 .. pR)* es la entrada de la red neuronal, *(a1 .. aS)* los vectores que representan la salida de las diferentes capas. Vemos que cada salida de cada neurona de la capa anterior es entrada de todas las neuronas de la capa siguiente, es decir si una capa tiene un vector de 10 elementos de salida, la capa siguiente tiene para cada neurona como entrada esos 10 elementos.

La topología con la que lo describe Matlab, esquema asociado es.



$$a^3 = f^3\left(LW^{3,2}\, f^2\left(LW^{2,1} f^1\left(IW^{1,1}p + b^1\right) + b^2\right) + b^3\right) = y$$

Representando de forma esquemática las diferentes matrices de pesos y vectores de entrada y salida de las diferentes capas.

Simulación en una red neuronal:
Existen varios modelos de simulación para redes neuronales.

Simulación con entradas concurrentes en una red estática. Es una red que no tiene feedback ni retardos con lo que no tenemos que estar preocupados acerca de si los vectores de entrada ocurren en una determinada secuencia.

```
P = [1 2 2 3; 2 1 3 1];
A = sim(net,P);
```

Simulación con entrada secuencial en una red dinámica. Cuando una red contiene retardos, suponemos que la entrada es una secuencia de vectores que ocurren en un cierto orden temporal.

```
P = {1 2 3 4};
A = sim(net,P);
```

Simulación con entradas concurrentes en una red dinámica. Tenemos que tener en cuenta el orden de aplicación ya que una entrada concurrente implica que cada vector de entrada debe de ser tratado como si estuviera en una red paralela y separada.

```
P = {[1 4] [2 3] [3 2] [4 1]};
A = sim(net,P);
A = {[ 1 4] [4 11] [7 8] [10 5]};
```

Entrenamiento de redes neuronales:
Existe un conjunto de estrategias diferentes de entrenamiento también llamados estilos de entrenamiento, **entrenamiento incremental** y **entrenamiento en batch**.

Entrenamiento Incremental: Es más usado con redes dinámicas con filtros adaptativos.
Entrenamiento incremental con redes estáticas. Si queremos entrenar una red de forma incremental de forma que los pesos y *bias* serán actualizados después de cada entrada, usaremos la función *adapt* y presentaremos las entradas como secuencias.

```
P = {[1;2] [2;1] [2;3] [3;1]};
T = {4 5 7 7};
[net,a,e,pf] = adapt(net,P,T);
```

Entrenamiento incremental con redes dinámicas. Tomamos una red lineal con uno o más retrasos y una entrada, Deberemos inicializar los pesos e inicializar la tasa de aprendizaje.

```
Pi = {1}; delay.
P = {2 3 4};
T = {3 5 7};
[net,a,e,pf] = adapt(net,P,T,Pi);
```

Entrenamiento en *batch*: En este caso los pesos y *bias* son actualizado después de que todas las entradas y objetivos son presentados. El entrenamiento *batch* puede ser usado con *adapt* y con *train* aunque normalmente *train* es la mejor opción.
Entrenamiento batch con redes estáticas.

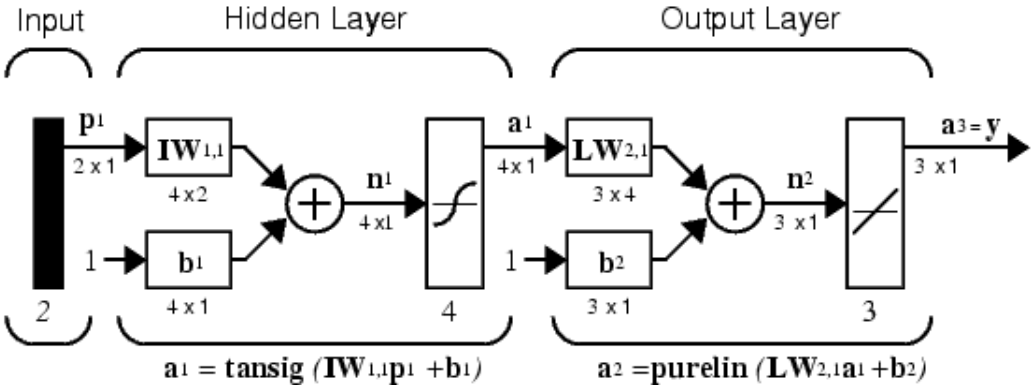
```
P = [1 2 2 3; 2 1 3 1];
T = [4 5 7 7];
net = train(net,P,T);
```

Entrenamiento *batch* con redes dinámicas.

```
Pi = {1};
P = {2 3 4};
T = {3 5 6};
net=train(net,P,T,Pi);
```

Backpropagation: Fue creado mediante la generalización de la regla de aprendizaje de **Widrow-Hoff** para múltiples capas y funciones de transferencia diferenciables. Los vectores de entrada y sus objetivos son usados para entrenar la red hasta que aproxima una función. Es capaz de aproximar cualquier función con un número finito de discontinuidades.

Normalmente solo usaremos una capa oculta con el suficiente número de neuronas. La descripción de esta red en Matlab es la siguiente:



El primer paso es crear una red feedforward, la primera entrada es una matriz con el máximo y el mínimo valor de cada uno de los elementos del vector de entrada, el segundo es una matriz que contiene la cantidad de neuronas que tendremos en cada capa, el tercero es un vector con las funciones de transferencia aplicadas a cada una de las capas y por último la función de entrenamiento de la red.

```
net=newff([-1 2; 0 5],[3,1],{'tansig','purelin'},'traingd');
```

El Segundo paso es inicializar los pesos y *bias* de la red.

```
net = init(net);
```

Para simular la red usaremos la función *sim*. Esta recoge el vector de entrada *p* y la red *net* y realiza una ejecución de la red retornando el vector de salida *a*.

```
p = [1;2];
a = sim(net,p);
```

Anterior a la fase de simulación deberemos entrenar la red, para ello existen una gran cantidad de algoritmos que podemos usar, cada uno de ellos con diferentes propiedades.

Entrenamiento: Quizá sea la fase más importante y complicada ya que hasta ahora la red no tiene conocimiento de los datos que está manejando, a partir de este momento la red empezará a trabajar con el modelo que le hemos enseñado. Durante el periodo de entrenamiento la red intentará minimizar su rendimiento modificando su peso y bias. El algoritmo básico de entrenamiento del backpropagation es en el que los pesos son movidos en la dirección negativa del gradiente, a lo largo del tiempo se han desarrollado algoritmos más complejos para incrementar la velocidad de la convergencia, como Matlab los tiene integrados haremos una breve descripción de algunos de ellos.

Algoritmo de backpropagation: Existen variaciones del algoritmo de backpropagation, el más simple es actualizar los pesos y bias de la red en la dirección en la que el rendimiento de la función decrece más rápidamente, el gradiente negativo. Cada iteración puede ser descrita de la siguiente forma.

$$x(k+1) = x(k) - a(k)g(k)$$

Donde *x(k)* es el vector con los pesos actuales, *g(k)* es el gradiente actual y *a(k)* es la rata de aprendizaje. Existen dos formas de aplicar el entrenamiento, **incremental** y **batch**, en modo incremental el gradiente y los pesos son modificados después de cada entrada, en cambio en modo *batch* todas las entradas son aplicadas antes de que los pesos sean incrementados.

El entrenamiento en batch (train). Los pesos y bias de la red son actualizados solo después de que se ha aplicado todo el entrenamiento a la red, los gradientes calculados en cada entrenamiento son añadidos para determinar el cambio de los pesos y bias. Existen dos algoritmos principales en la Toolbox de Matlab estos son, gradiente descendente batch (traingd) y gradiente descendente batch con momento (traingdm)

En el primero, **traingd**, los pesos y bias son actualizados en la dirección negativa del gradiente. La forma de especificar esta función en Matlab es:

Definimos los parámetros a entrenar.

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
```

Creamos la red con la función de entrenamiento especificada y modificamos los parámetros iniciales de entrenamiento.

```
net = newff(minmax(p),[3,1],{'tansig','purelin'},'traingd');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

Entrenamos y simulamos la red, estos últimos pasos siempre son los mismos para todas los tipos de entrenamiento.

```
[net,tr]=train(net,p,t);
a = sim(net,p)
a =
  -1.0010   -0.9989    1.0018    0.9985
```

El Segundo es, **traingdm**, el gradiente descendente con momento. Provee de una convergencia más rápida, el momento permite a la red responder no solo al gradiente local si no también a las tendencias que tenga la superficie de error, de esta manera pequeñas características en la superficie de error son ignoradas. Esto en Matlab se realiza con el parámetro de entrenamiento *ms*.

```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traingdm');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

Existen un conjunto de entrenamientos que son más rápidos que los anteriores. En la siguiente sección desarrollaremos un conjunto de algoritmos que trabajan en batch y que son cientos de veces más rápidos que los algoritmos básicos anteriormente estudiados. Existen dos categorías, la primera usa técnicas heurísticas y los dos algoritmos asociados son, **traingda** y **trainrp**. La segunda categoría usan técnicas de optimización numérica las cuales son, el gradiente conjugado (**traincgf**, **traincgp**, **traincgb**, **trainscg**), quasi-Newton (**trainbfg**, **trainoss**), and Levenberg-Marquardt (**trainlm**).

Rata de aprendizaje variable, **traingda**, **traingdx**. Normalmente en los algoritmos estándares, la tasa de aprendizaje se mantiene constante a lo largo del entrenamiento. Si la tasa es muy grande normalmente el algoritmo se puede convertir en inestable y si es pequeña tarda mucho en converger. Lo lógico es modificar dicha rata de aprendizaje durante el desarrollo del entrenamiento según la posición en la que nos encontremos dentro de la superficie de error.

```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traingda');
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.lr_inc = 1.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

Resilient backpropagation, **trainrp**. Los parámetros de entrenamiento son *epochs, show, goal, time, min_grad, max_fail, delt_inc, delt_dec, delta0, deltamax*. En este algoritmo solo es usada la dirección de la derivada para determinar la actualización de los pesos, la magnitud de la derivada no tiene efector en la actualización de los pesos, el incremento de los pesos y del *bias* viene dado por el factor *delt_inc* cuando la derivada de la función de rendimiento respecto a los pesos tiene el mismo signo en dos iteraciones sucesivas. El valor se desciende por el valor *delt_dec* cuando la derivada cambia de signo en dos iteraciones, si esta es cero el valor se mantiene.

```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainrp');
```

Existen un conjunto de retinas lineales de búsqueda que vamos a presentar aquí ya que la mayoría de los algoritmos de gradiente conjugado y quasi-Newton las requieren. Para usarlas debes de especificarla entre los parámetros de la red con el nombre **srchFcn**. Siempre es complicado saber cuál de estas rutinas es la mejor en el caso que nos encontremos.

Golden Section Search (**srchgo**).
Brent's Search (**srchbre**).
Hybrid Bisection-Cubic Search (**srchhyb**).
Charalambous' Search (**srchcha**).
Backtracking (**srchbac**).

Algoritmos del gradiente conjugado:

El algoritmo básico ajusta los pesos en la dirección negativa del gradiente, esto hace que el rendimiento decrezca rápidamente, esto no siempre produce la convergencia más rápida. En los algoritmos del gradiente conjugado la búsqueda es realizada a lo largo de las direcciones conjugadas, lo cual produce generalmente una convergencia más rápida en cada paso del descenso.

Actualización Fletcher-Reeves, **traincgf**. Los algoritmos del gradiente conjugado empiezan realizando la búsqueda en la dirección negativa del gradiente.

$$\mathbf{p}_0 = -\mathbf{g}_0$$

La búsqueda lineal se realiza para determinar la distancia óptima para moverse a lo largo de la dirección de búsqueda actual.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

El procedimiento general para determinar la nueva dirección de búsqueda es combinar el nuevo paso de la dirección descendente con la dirección de búsqueda previa.

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

Las diferentes versiones del gradiente se distinguen por la manera en la que la constante β_k es computada. Para la actualización de Fletcher-Reeves el procedimiento es

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Los parámetros de entrenamiento para la función **traincgf** son:
epochs, show, goal, time, min_grad, max_fail, srchFcn, scal_tol, alpha, beta, delta, gama, low_lim, up_lim, maxstep, minstep, bmax.

```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgf');
```

Actualización Polak-Ribiére, **traincgp**. Como en el Fletcher-Reeves, la dirección de búsqueda en cada iteración es determinada por.

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

La constante β_k es computada por.

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Es el producto de cambios previos en el gradiente con el gradiente actual dividido por el cuadrado normalizado el gradiente previo. Los parámetros de entrenamiento son los mismos que los de traincgf.

net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgp');

Reinicio de Powell-Beale, **traincgb**. Para todos los algoritmos del gradiente conjugado, la dirección es periódicamente reiniciada al negativo del gradiente. El reinicio ocurre cuando el número de iteraciones es igual al número de parámetros de la red, pero existen otros métodos de reinicio que pueden mejorar el entrenamiento. En este método, reiniciaremos si hay una pequeña diferencia ortogonal entre el actual gradiente y el anterior, dato por la desigualdad.

$$\left| \mathbf{g}_{k-1}^T \mathbf{g}_k \right| \geq 0.2 \left\| \mathbf{g}_k \right\|^2$$

Si la condición se satisface, la dirección de búsqueda es reiniciada al gradiente negativo. Los parámetros de entrenamiento son los mismos que los de traincgf.

net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgb');

Gradiente conjugado escalado, **trainscg**. Cada uno de los algoritmos del gradiente conjugado que hemos discutido necesita una línea de búsqueda en cada iteración. Esto es computacionalmente caro ya que requiere una respuesta para cada búsqueda. Este algoritmo (SCG) fue diseñado para evitar el tiempo consumido en cada línea de búsqueda. La idea básica es combinar una aproximación a la región de confianza con el gradiente conjugado.

Los parámetros de entrenamiento son, epochs, show, goal, time, min_grad, max_fail, sigma, lambda.

net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainscg');

Algoritmos Quasi-Newton:
Algoritmo BFGS, **trainbgf**. El paso básico para este algoritmo es:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$$

Donde $\mathbf{A}_k$ es el Hessiano de la matriz (segundas derivadas), es el índice del rendimiento de los valores actuales de los pasos y bias. Este método a menudo converge más rápidamente que los del gradiente conjugado, desgraciadamente suele ser costosa de computar la matriz Hessiana de la red. Este método actualiza y aproxima la matriz Hessiana en cada iteración del algoritmo.

Los parámetros de nuestra función son los mismos que los de la función traincgf.

net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainbfg');

Algoritmo de la secante de un paso, **trainoss**. El algoritmo BFGS requiere mayor memoria y computación en cada iteración que el algoritmo del gradiente conjugado, si usamos una

aproximación por secante podemos reducir la memoria y el tiempo de computación. El método de la secante de un paso (OSS) intenta ser un puente entre el gradiente conjugado y el algoritmo de quasi-Newton, el primer Hessiano es la matriz identidad, esto nos da una ventaja adicional ya que la búsqueda de la dirección puede ser calculada sin usar la matriz inversa.

Los parámetros de nuestra función son los mismos que los de la función traincgf.

```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainoss');
```

Levenberg-Marquardt, **trainlm**. Como los métodos quasi-Newton, el algoritmo de Levenberg-Marquardt fue diseñado como una aproximación de segundo orden sin tener que computar la matriz del Hessiano. La matriz del Hessiano puede ser aproximado es.

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

Y el gradiente puede ser computado como.

$$\mathbf{g} = \mathbf{J}^T\mathbf{e}$$

Donde ***J*** es la matriz del Jacobiano que contiene las primeras derivadas de los errores de la red respecto a los pesos y bias, y ***e*** es un vector con los errores de la red. La matriz del Jacobiano puede ser computada con técnicas estándares del algoritmo de backpropagation.
El algoritmo de Levenberg-Marquardt usa esta aproximación a la matriz del Hessiano en la siguiente actualización de Newton.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e}$$

Cuando el escalar μ es cero, el método de Newton usa la aproximación a la matriz del Hessiano. Los parámetros de entrenamiento son, *epochs, show, goal, time, min_grad, max_fail, mu, mu_dec, mu_inc, mu_max, mem_reduc*.

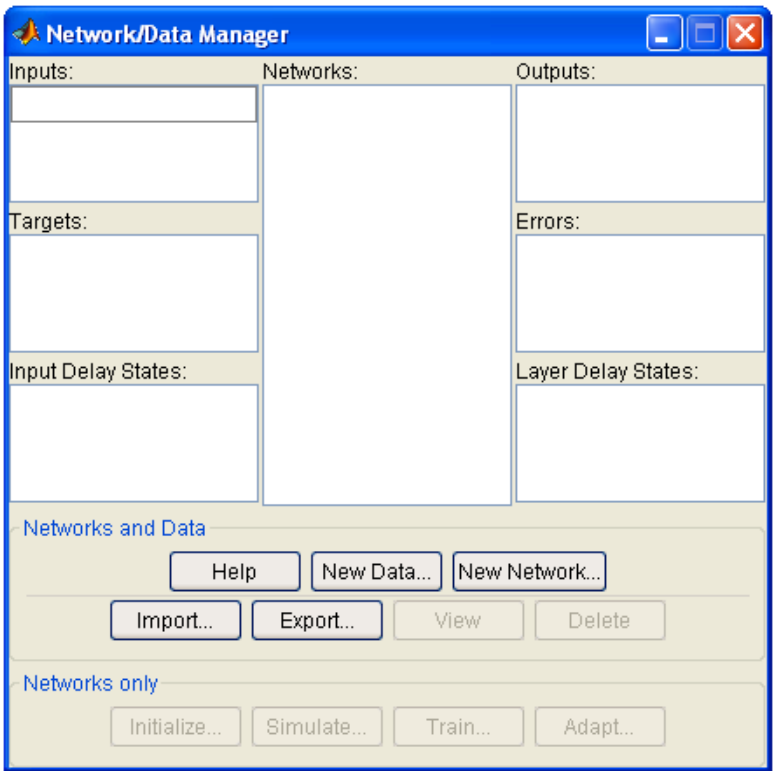
```
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainlm');
```

Reducción de memoria de Levenberg-Marquardt, **trainlm**. Lo principal del algoritmo de Levenberg-Marquardt es que requiere una cantidad de almacenamiento muy grande para ciertos problemas. El tamaño de la matriz del Jacobiano es *Q x n*, donde *Q* es el numero del conjunto de entrenamiento, *n* es el numero de pesos y bias de la red.

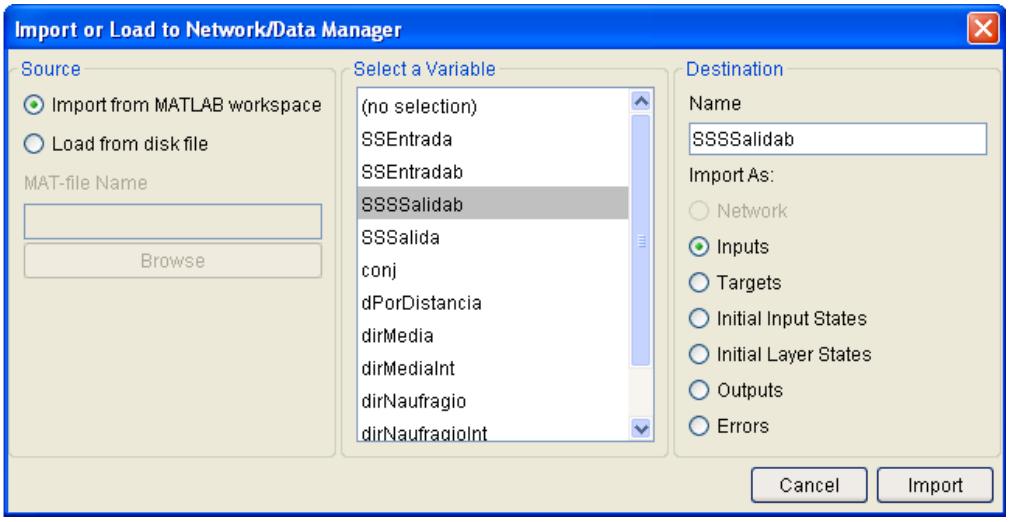
nntool

Por otro lado existe una herramienta que nos permite crear cualquier tipo de red, entrenarla y simularla, esta es la ***nntool*** (Neural Network Tool-Graphical User Interface), de hecho la hemos usado para crear algunas de las redes de este proyecto. El parámetro ***nntool*** abre la ventana del gestor de datos y redes, el cual permite importar, crear, usar y exportar redes neuronales y datos.

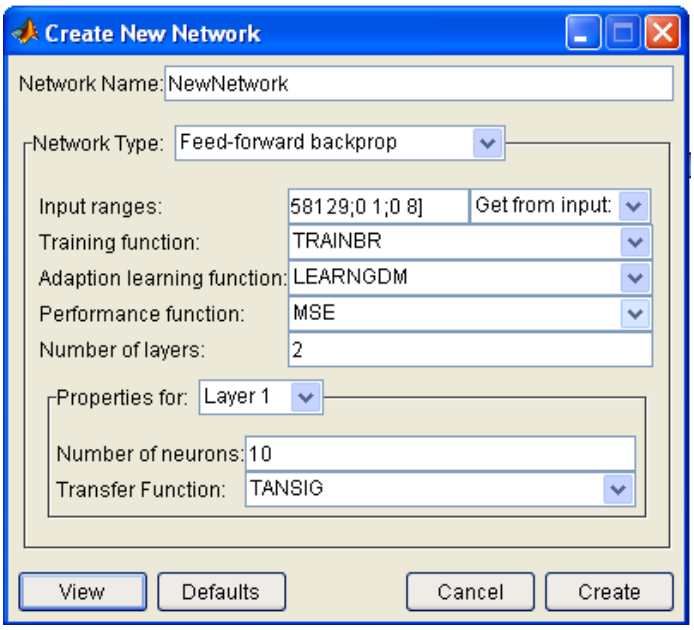
Antes de usar las ***nntool***, para lo cual solo hay que ejecutar dicho comando deberemos de preparar los datos de entrenamiento y dejarlos en variables de entorno, una para la entrada y otra para la correspondiente salida.



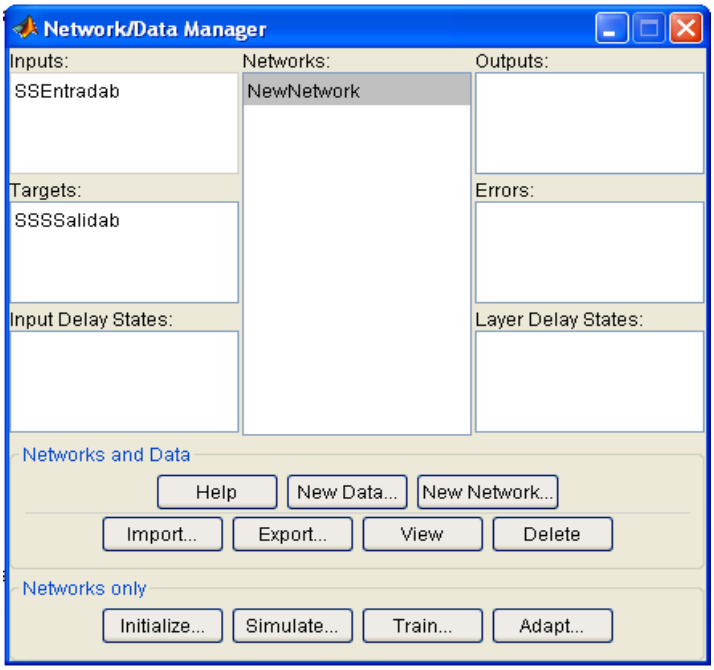
Primero ejecutaremos la ventana, **nntool**, después traeremos los inputs y targets con el botón de **import**.



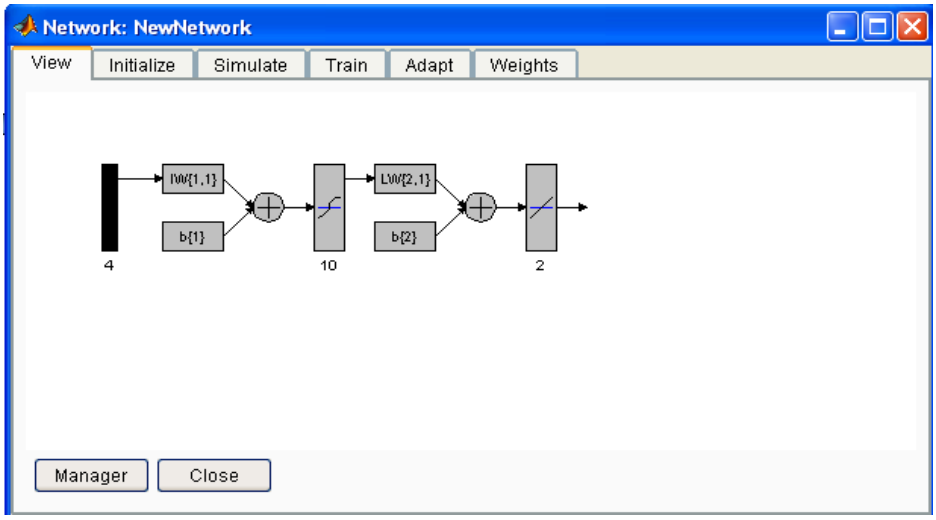
Una vez tenemos los datos de entrada y objetivos de salida importados en la herramienta procederemos a crear la red que queremos entrenar. Hacemos "**new network**" y nos encontraremos con la ventana de creación de cualquier red neuronal. Aquí podremos ponerle el nombre a nuestra red, elegir las funciones de entrenamiento, el tipo de red, número de capas, y función de transferencia para cada una de las capas.

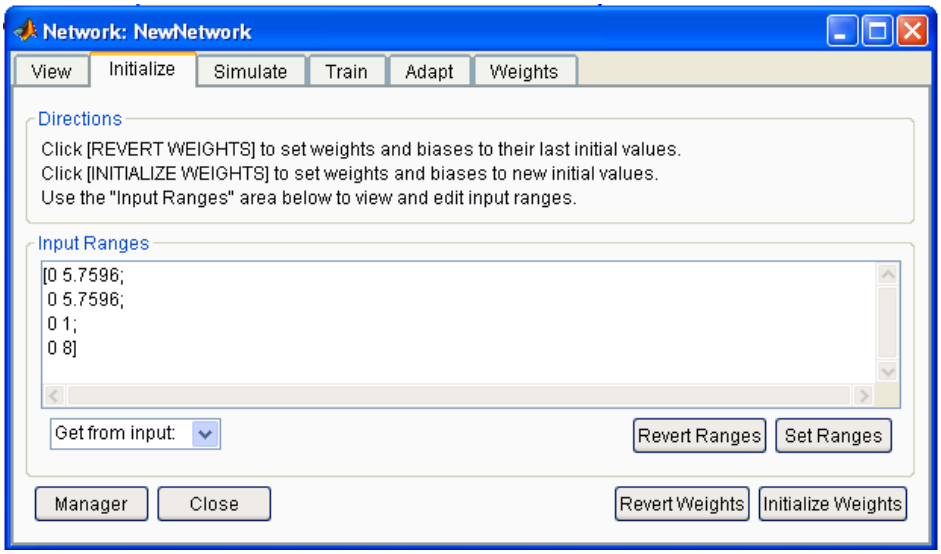


Una vez tenemos creada nuestra red, la seleccionamos. Pulsamos **initialize**.

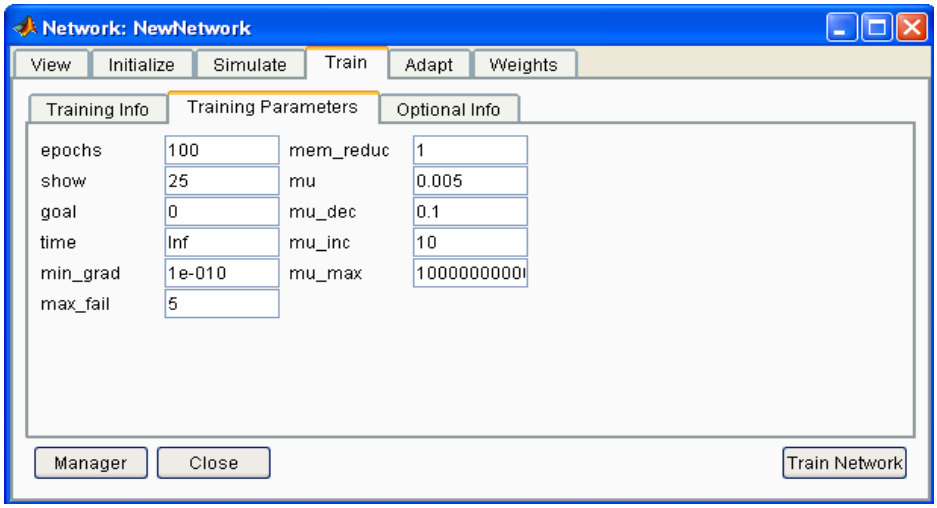


Nos encontramos con una ventana que nos permite realizar diferentes acciones sobre nuestra red neuronal. En nuestro caso seleccionamos **train** que es la ventana de entrenamiento, y según la red que hayamos elegido esta ventana nos presentará sus parámetros de entrenamiento.

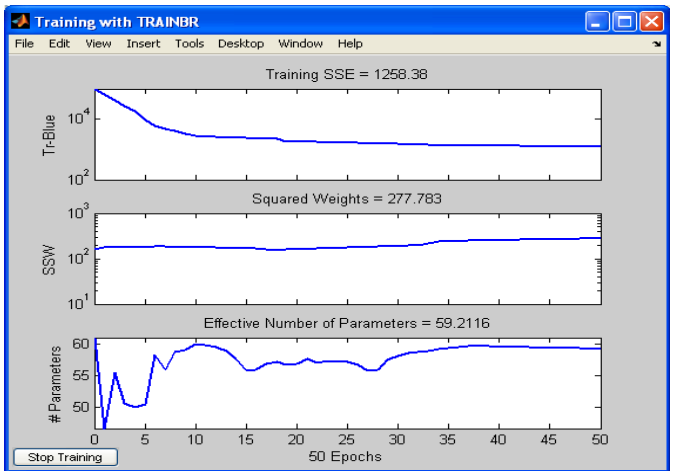




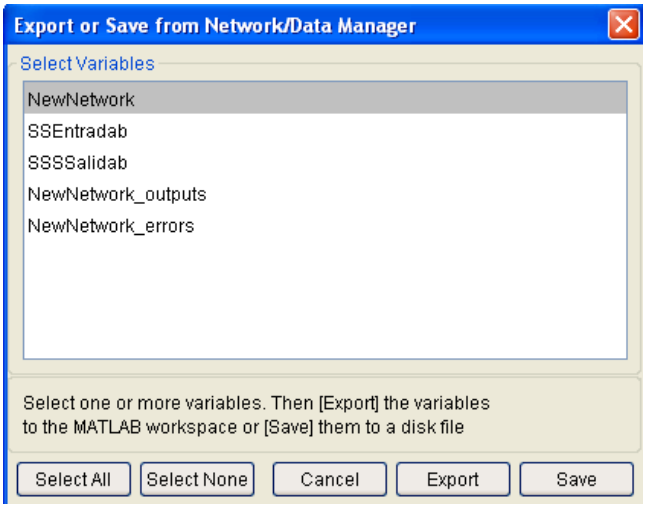
Seleccionamos las entradas de la red y posteriormente, los parámetros del entrenamiento, para finalizar ejecutaremos el entrenamiento.



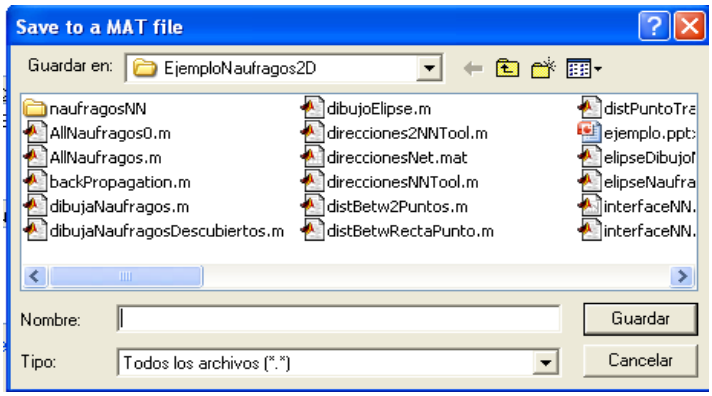
La pantalla con el entrenamiento será.



Una vez acabada la simulación podemos guardar la red neuronal para ser usada en cualquier otro programa. Volvemos a la pantalla principal, seleccionamos la red y pulsamos **Export**.



Después de seleccionar la red, pulsamos **save**, esto nos permitirá salvar la red con el nombre que le pongamos en el sistema de ficheros con la extensión **.m**



FIN DEL DOCUMENTO