Redes Neuronales

**Franco Bernaldo de Quirós**

**Resumen del proyecto**

El proyecto consta de la explicación e implementación de una red neuronal básica.

Hablaré de las redes neuronales y de su implementación en modelos de algoritmos genéticos.

**Objetivo**

En este trabajo se expondrá y explicará el funcionamiento de una red neuronal y su orientación a algoritmos genéticos.

Todo esto con el objetivo final de lograr comprender y aplicar una red neuronal con el modelo Neuro-Evolution.

**Introducción**

El uso de las redes neuronales es muy variado y es prácticamente un fenómeno de lo más común a día de hoy. Desde finanzas hasta lingüística, reconocimiento de imágenes y muchos, muchos más.

Las redes neuronales son muy variadas en cuanto a los tipos de tareas que pueden aprender, y, exactamente por esto último es que son tan utilizadas, por su capacidad de **aprender**.

Es importante mencionar que no hay una única forma de aprender ni de realizar una tarea y esto es igual en las redes neuronales.

El único propósito de la red neuronal es el de separar grupos de datos y luego afirmar una respuesta en base a esta separación.

**La neurona**

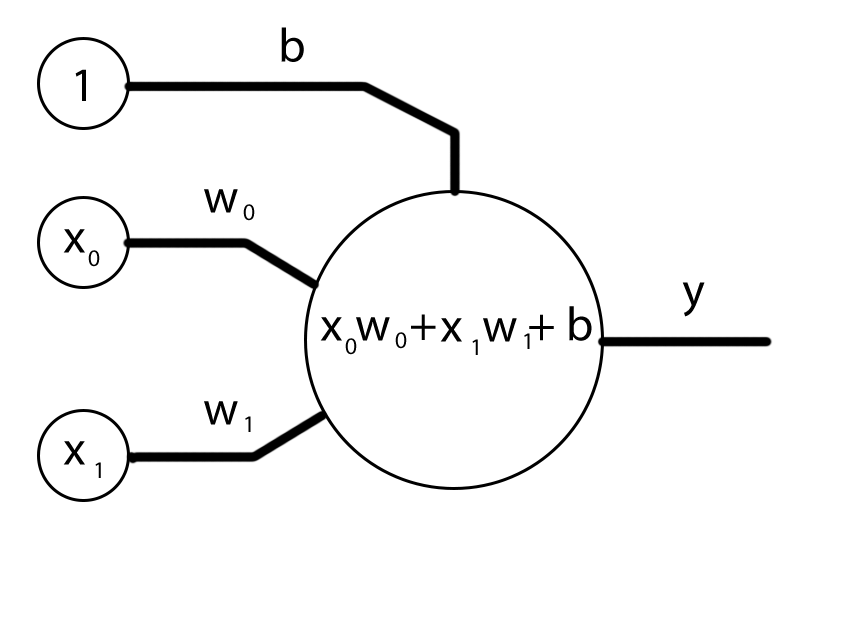
La neurona es el concepto más básico de la “red **neuronal**” pero ¿A qué nos referimos con neurona?

Se puede ver a una neurona como una caja negra, en la que podemos introducir distintos tipos de datos, “la caja” producirá una única respuesta a estos datos, es decir, una salida.

Si analizamos el concepto recuerda mucho a una **función** **matemática** y es que justamente eso es.

La neurona es una función.

Bueno, ahora que tenemos el concepto de neurona puedo pasar a explicar lo siguiente, el funcionamiento ¿Como esta “caja negra” logra esa salida? La neurona se encarga multiplicar a cada **entrada**(x) un determinado **peso**(w) y sumar todos los resultados. También se le suma un **bias**(b) extra, que se puede imaginar como el peso de una entrada siempre en uno. El bias se puede ver como una forma de darle más “fuerza” o menos a la neurona, es decir, como una forma de desactivarla o aumentar su poder frente a el resto en la red.



La neurona puede ser tratada como matriz la fórmula en ese caso la formulación sería:

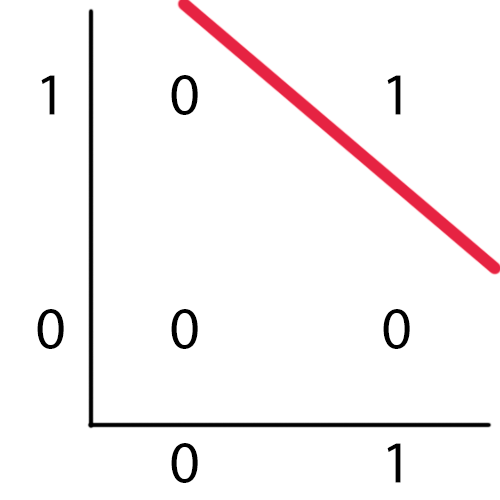
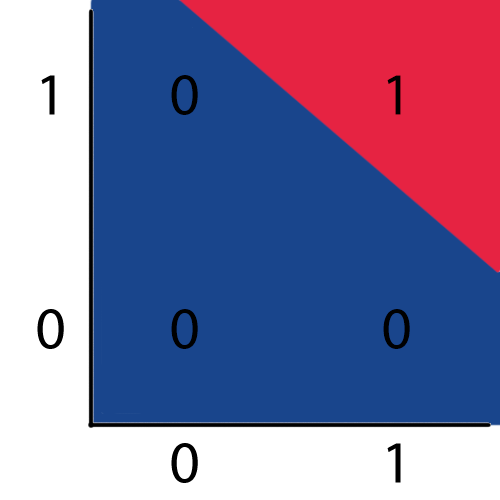
**Y = XW+B** .

Cabe agregar un pequeño comentario y es que la neurona por sí misma es aceptada como una **red neuronal**. Esto es así por la capacidad de **extenderse** que posee una red neuronal.

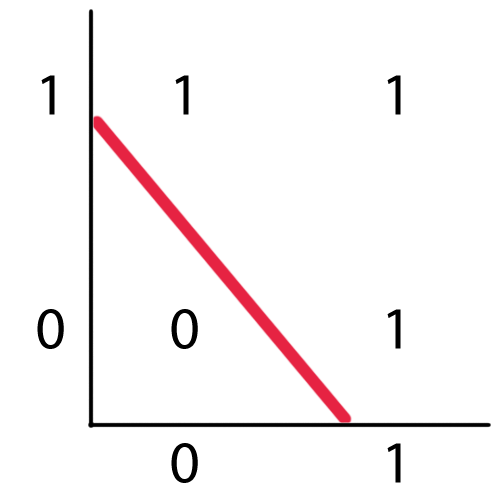
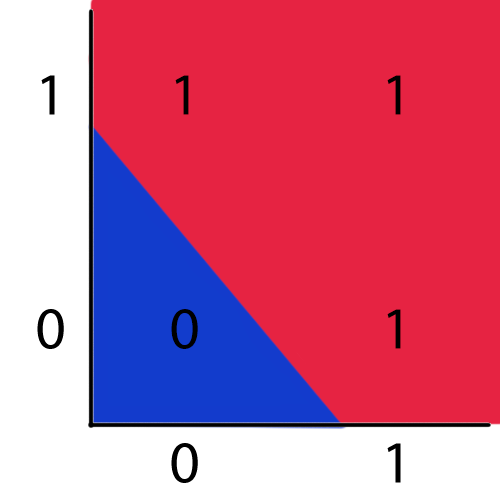
**Necesidad de más neuronas**

Una vez que tenemos la neurona ¿Por qué necesitamos más? es decir, ya tenemos nuestra función encargada de producir resultados ¿ Por qué más?

El objetivo de la red neuronal es separar en grupos los datos, y una neurona actúa como una línea entre la separación de estos grupos. Veamos el caso de la puerta lógica AND y la OR.

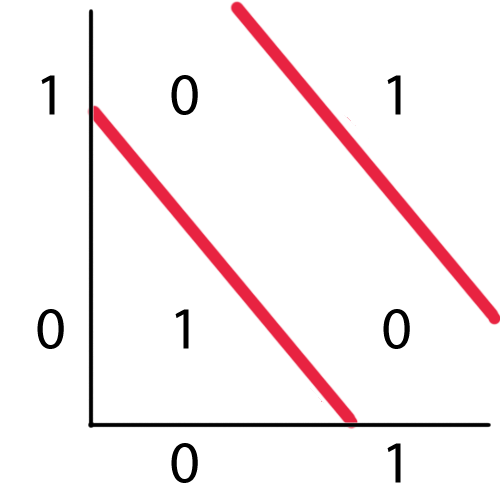
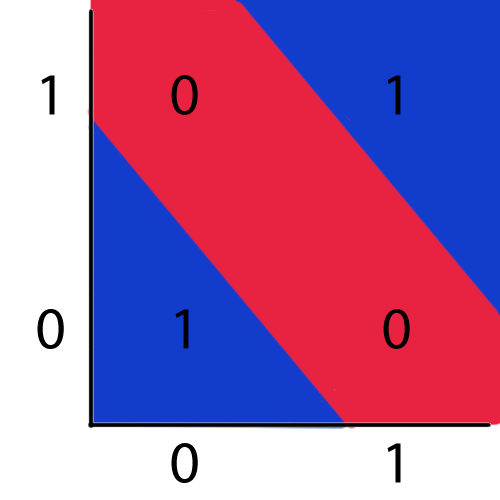
 

AND separación de datos AND grupos

OR separación de datos OR grupos

Podemos ver que la separación de ambos grupos está dada por una línea, a partir de esa línea la red considera si una opción pertenece a un grupo (verdadero) u a otro (falso) ahora bien, veamos el caso de la puerta XNOR

XNOR separación de datos XNOR grupos

Se puede ver en este caso que los grupos en la puerta XNOR deben ser divididos por 2 líneas (ya que con una es imposible), es decir, se necesitan **dos neuronas**.

**La red**

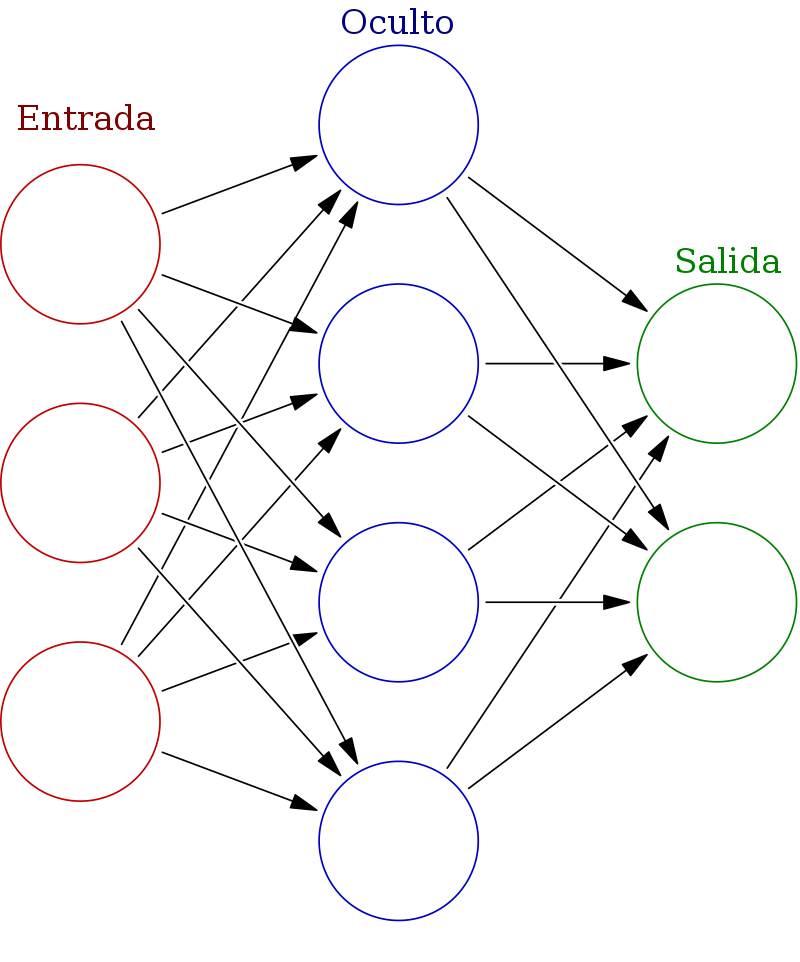
Una vez tenemos más de una neurona surge una duda ¿ Qué tipo de organización debe tener? ¿ Como se colocan las neuronas dentro de la red?. El tipo de organización que se le da a una red neuronal se conoce con el término **topología**.

Las **capas** de una red neuronal son neuronas en paralelo, es decir, todas reciben las mismas entradas pero no están conectadas entre sí. Los resultados de una capa son enviados a cada neurona de la capa siguiente hasta la última capa que es la responsable de producir la salida de la red.

La primer capa es conocida como la capa **de entrada**, las interiores como las capas **ocultas** y la última como la capa de **salida**.

Cada capa puede contener distintos números de neuronas.

Las entradas a la red son valores numéricos normalizados.



¿Qué ventajas posee un **perceptrón multicapa** (una red neuronal de más de una capa)?

La ventaja que posee es la **jerarquía de conocimiento**, la primer capa se encarga de procesar información en crudo de las entradas, la siguiente procesa información ya procesada un poco más y así capa tras capa.

Por ejemplo, una red neuronal cuyo objetivo es detectar si hay un auto o no en la imagen. La primer capa recibe la imagen, la procesan y este procesamiento se encarga de buscar algunos patrones básicos, líneas, círculos , etc. Las siguientes reciben esta información e identifican los conjuntos de patrones buscando elementos que posee un auto, espejos, puertas y con toda esta información la última capa se encarga de procesar y decir “Si, hay un auto” o “No, no hay un auto”.

**Función de activación**

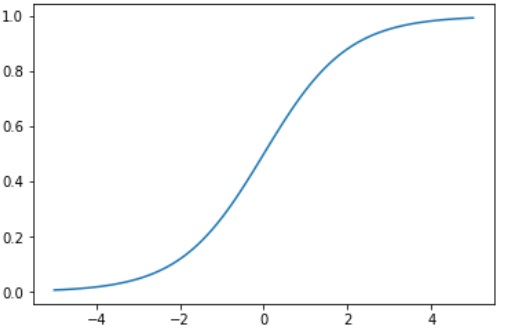
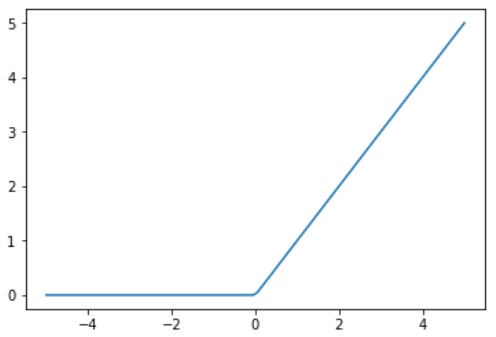
Hasta ahora estuve obviando un problema enorme en nuestra red neuronal por un principio básico en las matemáticas.

He dicho que las neuronas son una función, en concreto una **función lineal**. En nuestra red neuronal estamos **concatenando** funciones lineales y la concatenación de muchas funciones lineales es una función lineal.

Nuestra red neuronal colapsa en una única neurona.

Para resolver este problema surgen las **funciones de activación** que es una función no lineal aplicada al resultado de nuestras neuronas en cada capa.

Hay distintos tipos de funciones de activación, dos ejemplos de las más utilizadas pueden ser la función **sigmoide** y la función **RELU**.

Sigmoide RELU

La función sigmoide se la suele usar en casos de necesidad de una probabilidad o bien “que tan probable es algo”, devuelve números entre 0 y 1 (por ejemplo, si la red devolviera 0.87, está un 87% segura de esa decisión . Es la función utilizada en la topología de mi trabajo.

La función RELU se suele utilizar en casos que se necesiten cantidades, por ejemplo, la cantidad de personas en una imagen.

Cabe destacar que la función de activación es compartida en cada capa y una red neuronal puede tener distintas funciones de activación por cada una de sus capas.



**Función de coste**

Una vez que nuestra red neuronal es capaz de lanzar una predicción, es decir, somos capaces de obtener una salida, nos viene a la cabeza como la red puede lograr aprender y para aprender la red necesita saber si **está equivocada** y no solo si está o no lo está sino **que tan equivocada está**. y esto es justamente lo que la función de coste nos indica.

La función de coste es una función aplicada a la salida de nuestra red para colocar un único valor numérico a nuestro error.

**Tipos de aprendizaje**

Existen distintas formas de enseñarle a nuestra red, los 2 tipos que reinan en el campo son el aprendizaje **supervisado** y el aprendizaje **no supervisado**.

El aprendizaje supervisado consta de 3 pasos:

1. Se le suministra un dato de entrada a la red del que ya conocemos el valor deseado de salida
2. Se obtiene la salida de la red pasada por la función de coste
3. Se corrige los pesos de la red por distintos tipos de técnicas

Estos 3 pasos se calculan con muchos datos de entrada en bucle. Y una vez que se ajusta a la red (con cierto margen de error aceptable) se la prueba con datos que la red jamás conoció pero que nosotros conocemos su valor si acierta, la red superó la prueba y se considera con la capacidad de **generalizar**. Sinó, se necesitarán realizar los ajustes necesarios.

El aprendizaje no supervisado(qué es en el que consta este trabajo) trabaja de manera distinta. No necesitamos conocer salidas deseadas ni tampoco conseguir datos fuera del sistema en el que la red trabajará.

La idea del aprendizaje supervisado es que la red neuronal pueda aprender a partir de los datos que dispone en todo momento y encontrar la relación lógica entre estos.

Hay muchos tipos de aprendizajes no supervisados pero del que voy a hablar es de los **algoritmos evolutivos**. Y hablaré, más concretamente, del modelo **Neuro-Evolution**, que es el modelo implementado en este trabajo.

**Introduccion a Algoritmos Evolutivos**

Como dije previamente los algoritmos evolutivos son un tipo de aprendizaje no supervisado.

Al principio suena muy dificil de creer y entender ¿Como una red neuronal puede aprender sin necesidad de un proceso de aprendizaje supervisado?

La respuesta que dan los algoritmos evolutivos es : simulando la evolución biológica.

En el método evolutivo se utilizan cierto número de individuos (redes neuronales) y se los pone a prueba en su entorno, en la resolución de la tarea a resolver, se los puntúa (con la función de coste), luego, a los mejores individuos se los “reproduce”, esta reproducción es conocida como **crossover**. La siguiente **generación** consta del mismo número de individuos, pero estos son los hijos de los mejores de la generación anterior. Se conoce como **población** al número de individuos de una generación.

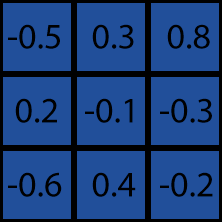
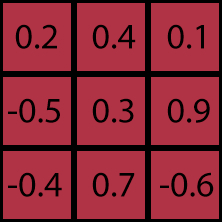
La cantidad de población en una generación es importante por el crossover (que explicaré en breves), se necesitan bastantes individuos en una población para generar la **variedad necesaria** y poder elegir al mejor. Pero muchos no harán más que ralentizar el proceso de aprendizaje por lo que su valor debe ser nivelado.

**Crossover**

Una vez terminada la generación, se debe “reproducir” a los padres, esta reproducción se implementa en las redes neuronales por medio del crossover. El crossover se realiza de 2 a “n” redes neuronales. Su topología debe ser exactamente igual.

La esencia es mezclar los pesos(w) de las redes de forma aleatoria en el nuevo hijo, pero no en cualquier orden.

Los pesos se pueden agrupar o pensar como matrices (como ya dije en la explicación de las neuronas). Si los pensamos de esa forma es más fácil entender esta mezcla de matrices.

MATRIZ W del padre 1 MATRIZ W del padre 2



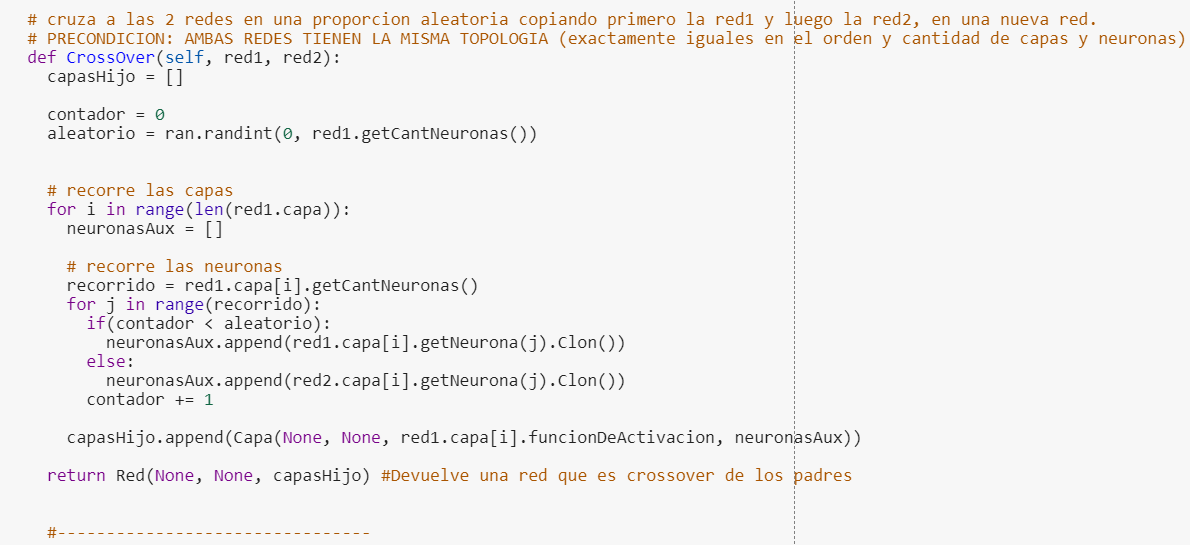
Resultado del crossover (hijo)

Para ejemplificar la matriz w tiene solamente 9 pesos.

La cantidad de pesos de cada padre que recibirá el hijo, es el valor aleatorio, puede ser que incluso el hijo sea igual a alguno de sus padres (en el ejemplo anterior recibe 7 del primer padre y 2 del segundo).

Con esto se busca alterar levemente las estructuras neuronales de ambos padres es decir, dejar neuronas intactas y cambiar levemente algunas. Que no sea una mezcla totalmente aleatoria.

Todas estas operaciones son igual para los bias de la red (ya que también son pesos).



**Mutaciones**

Las mutaciones, tanto en la biología como en las redes neuronales, son errores que bien pueden beneficiar como entorpecer a un individuo y favorecen a generar variedad de especies.

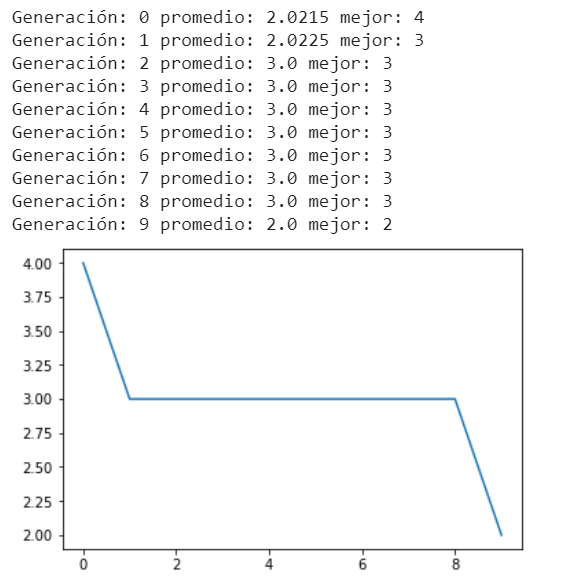
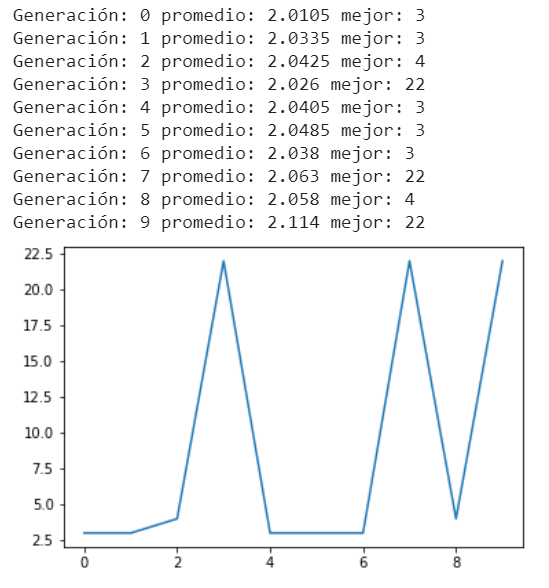
Si hablamos de mutaciones en las redes neuronales, no es más que la probabilidad de alterar o no un peso de la red luego del crossover. Esto ayuda a generar variedad y a salir de “pozos” donde la red podrá ser aún mejor.

Las mutaciones en los algoritmos evolutivos **generan la variedad necesaria para el aprendizaje**. Sin ellas, los siguientes individuos perderán la capacidad de cambio y con ella, la capacidad de aprender.

Es importante estar conforme con la probabilidad de mutaciones que queremos en nuestra red. Mucha probabilidad de mutar y el crossover no servirá de nada, muy poca y nos estancamos en la igualdad previamente mencionada.



Hijo con una mutación (puede tener más por probabilidad, o bien ninguna).

**CrossOver + mutación del 0% CrossOver + mutación del 100%**

Por un lado con la ausencia de mutación (del 0%) podemos ver una estabilidad en el promedio, es decir, mismo problema igual a mismos resultados. **Todas las redes finalizan siendo iguales.**

Hay que recordar que la diferencia de los datos entre generaciones resulta de que el problema no es exactamente igual, es decir, la comida aparece en diferentes posiciones.

Por el otro lado, podemos apreciar una alta mutación(100%), es decir, las redes neuronales totalmente aleatorias en cada generación. **Perdiendo la capacidad de mejorar y aprender.**

**“auto-Snake”**

El desafío de este trabajo consta en, por medio del modelo “Neuro-Evolution”, que una red neuronal mejore su eficacia jugando a “snake”. a partir de ahora con toda la teoría explicada empezaré a contar los pasos que fuí dando y los desafíos de este trabajo.

El juego snake, si bien tuve que programarlo para poder controlar todos los datos que la red va a recibir, cabe mencionar la comunicación de la red con el mismo.

El juego está creado en una matriz, un array bidimensional, donde la red recibe los datos y responde luego de cada movimiento (cada frame), es decir, los pasos son: **la red recibe los datos** (más adelante mencionaré cuales), **responde** a los mismos con el movimiento generado, **se actualiza el tablero**.

En caso que la víbora se esté moviendo para la derecha y la red quiera moverse hacia la izquierda ( movimiento no permitido en el juego snake) es como si no quisiera moverse en ninguna dirección y su siguiente movimiento será, en este caso, derecha.

También agregue un valor numérico a los pasos que da la serpiente, en cada paso este valor se resta en uno y cuando come se le agregan pasos. Si los pasos llegan a 0, la víbora muere. Implementé esto para que la serpiente no quede en un bucle infinito girando sobre sí misma y su objetivo sea la comida.

**“Visión” de la red**

¿Que visualizará la red? Es decir, nosotros al jugar vemos todo el tablero de juego. Podría simplemente “aplanar” ese array bidimensional en uno de una dimensión y pasarle todo el tablero a la red.

Eso fue lo que pensé pero terminó siendo el primer gran problema o mejor dicho, los primeros grandes problemas. En primer lugar la cantidad de datos es inmensa, incluso para una matriz pequeña de 10x10 sería una entrada de **100 datos**, 100 neuronas individuales para aceptar la entrada de cada uno de esos datos. En segundo lugar a raíz de ser muchos datos se **ralentiza enormemente** el proceso de aprendizaje, 100 neuronas (iniciales, más las de la red) cada una con sus pesos, sus bias y sus sumas multiplicado por la cantidad de integrantes en la población.

Sin mencionar que el problema de buscar la relación lógica entre los datos individuales de cada posición (de la entrada) es increiblemente dificil.

No, se necesita una respuesta más sencilla, más humana, o mejor dicho, **más vibora**. Y es que justamente lo que hice fue tratar a la víbora como un individuo y no como un jugador que conoce el tablero, y es lógico pensarlo así es decir, estoy tratando con poblaciones y con generaciones. Las redes pueden ser tratadas como seres vivos… al menos en concepto.

Los siguiente que se me ocurrió es hacer que vea en las 4 direcciones y sus diagonales (como se mueve una reina en el ajedrez) es decir, 8 direcciones. Y en cada dirección visualizar la casilla con la que colision más cercana, es decir, si a la izquierda lo más cercano a la cabeza de la serpiente es la cola se enviará por esa entrada “cola” a la red.

Estos serian 8 datos y habría 8 más, por cada uno se envía la distancia entre la cabeza y el objeto.

Esto parecía tener mucho sentido, de 100 entradas se reducen a 16, pero resulta que a la red le costaba mucho encontrar la comida y aprender. Tardando demasiadas generaciones en mejorar, e incluso esta mejora era relativamente más pequeña.

Estaba claro que tenía algún problema con la entrada de los datos pero no entendí, en su momento, cúal era el error.

Investigando encontré que un usuario utilizó una técnica muy similar para el aprendizaje, solo que por cada **dirección** utilizó 3 entradas, una para la **comida**, una para la **cola** y otra para el **muro**.

Si existía en esa dirección alguno de esos 3 datos, enviaba por la entrada correspondiente la distancia entre la cabeza del objeto y la víbora.

La red posee 24 entradas (3 por cada dirección).

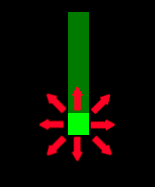
Entendí que la red podía asociar mejor los datos si el objeto entraba por una entrada correspondiente (en vez de 3 posibles objetos por una entrada) y con su distancia ya implementada allí mismo.

Pero a la red le pasaba algo, los objetos a partir de la 3 casilla estaban a una distancia muy poco distinguible, siendo la fórmula de la distancia : **1 / distancia (en casillas)**  es fácil notar que los datos a partir de 1/3 comienzan a ser cada vez más cercanos entre sí mismos y a 0.

Con la cola y el muro no es un problema ya que es necesario que la serpiente esquive estos objetos cuando los tiene cerca, una distancia de 3 casillas es más que suficiente, **el problema es la comida**, ya que incluso siendo difícil de ver (por la limitada visualización al tener simplemente 8 direcciones,y por la ausencia de memoria) a partir de 4 casillas para la red es prácticamente indistinguible, o mejor dicho, muy difícil de equilibrar los pesos necesarios para detectar un incremento tan bajo.

Todo esto causando una ralentización innecesaria.

La solución a este problema fue una solución simple, en lugar de enviar la distancia a la comida, por la entrada de la comida, solo envía un valor binario, 1 si la comida se encuentra en esa dirección y 0 si no.



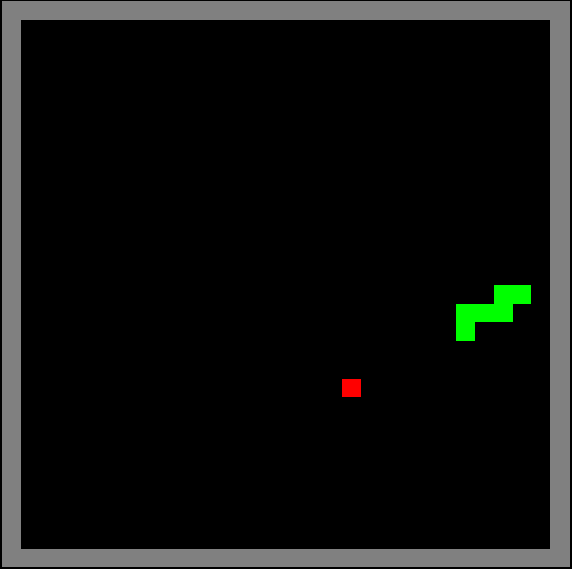
Direcciones en la que la serpiente observa.

**Especies**

Durante el entrenamiento de la red neuronal se pueden apreciar diferencias notables en su comportamiento. Diferentes tipos de **especies**.

Este tipo de serpiente busca la comida de esa forma y una vez encontrada se mueve en forma recta hacia ella busca.

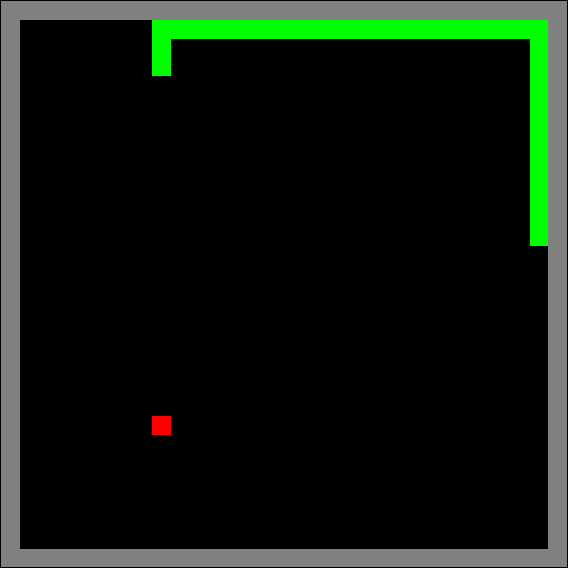
Esta técnica puede ser entendida por la visión de la que dispone la red, moverse en diagonal reduce los tiempos en los que la red tarda en visualizar la comida.



Otro tipo es la serpiente que se mueve realizando cuadrados, una vez capturada la comida se mueve hacia el borde derecho, luego hacia arriba y busca su comida observando hacia abajo desde el borde superior. Una vez encontrada se mueve en línea recta hacia ella y vuelve al borde derecho.

Es fácil entender las ventajas de esta técnica al recordar que la serpiente actual en cada momento y que no tiene la capacidad ni de memorizar ni de planear. Moverse hacia el borde el evitaría encerrarse a sí misma, logrando un tamaño considerable. Su muerte es inevitable al llegar a cierto tamaño, o que la comida se encuentre muy cerca del borde superior derecho.

Por alguna razón, noté que este comportamiento es el más elegido por las redes neuronales cerca de la generación 60.



Luego se aprecian pequeñas mejoras en este comportamiento, como cortar camino en diagonal y también cambios que no son mejoras sino cambios pequeños en comportamiento como buscar la comida desde debajo.

**Conclusiones**

El aprendizaje no supervisado si bien tiene un poco más de complejidad a la hora de programar ya que se necesita programar el entorno donde se utilizará y, actualmente, es más lento a la hora de aprender que el aprendizaje supervisado. Compensa todo esto con la no necesidad de recolección de datos.

Las redes neuronales pueden sorprender a la hora de aprender, porque incluso un juego **tan simple** como el snake resultó tener ciertas **complicaciones**.

Personalmente gracias a esto reforcé muchos conocimientos matemáticos. Operaciones con matrices, con funciones, espacios vectoriales y transformaciones del espacio (el espacio bidimensional al unidimensional para los datos de la red), y, mientras aprendía sobre el aprendizaje supervisado, también derivadas.

Aprendí ciertas cosas sobre biología. Los conceptos como la evolución desde diferentes puntos de vista y el gen egoísta, de richard dawkins. También sobre funcionamiento básico de células (la respuesta a cambios en el entorno).

Muchas veces tuve que mirar atrás en el tiempo y pensar sobre los cambios evolutivos. Por ejemplo, donde más evolucionó la vida fue luego de una extinción masiva, es decir, **mediante desafíos**. Y es que justamente por eso aprende una red neuronal en este tipo de aprendizajes, mediante desafíos.

Y de esta forma aprendí yo también, mediante el desafío de cómo crear una red neuronal con aprendizaje no supervisado.

**Referencias**

Correspondiente a las redes neuronales, el tema es conocido por mi desde hace mucho tiempo y visite muchos foros, canales de youtube, y sitios web.

Los de mayor ayuda fueron **dot CSV** un canal de youtube orientado exclusivamente a machine learning y una lista de reproducción de redes neuronales de **Javier García** (ese es el nombre del canal).

Y con respecto a la parte biológica, es decir, la **parte orgánica** dado por los algoritmos evolutivos no hay una fuente específica, pero digamos que todo se basa en la “teoría de la evolución” de darwin y “el gen egoísta” de richard dawkins. En este tema **Primer** (un canal de youtube) aportó mucho conocimiento.

También aprendí bastante de ver pruebas y errores de algunos videos de youtube, recomendados hace ya demasiado tiempo como para recordar sus nombres.

De cualquier forma, todos estos conceptos son teóricos y la forma en la que aprendí fue mediante experiencia y búsqueda ante errores, prueba y error.