

Integrantes

Daniel Alarcón 0905-17-3496 (100%)

Juan Sandoval 0905-17-5470 (100%)

Byron Vivas 0905-17-1874 (100%)

Mariela Micheo 0905-17-7260 (100%)

Aury Linares 0905-17-5457 (100%)

Pablo Josué Corado Elias 0905-17-9677 (100%)

Eduardo Emanuel Cermeño Pineda 0905-17-19312 (100%)

Gustavo Enrique Godoy Cabrera 0905-17-22303 (100%)

Carlos Eduardo Godoy Cabrera 0905-17-22304 (100%)

Kevin Javier Martinez Najarro 0905-17-2156 (100%)

William Antonio Garcia Lucero 0905-17-3771 (100%)

Bonnie Elizabeth Mora Barquero 0905-17-11142 (100%)

Jose Benedicto Ortega Valladares 0905-13-11345 (100%)

Kevin Eduardo Villeda Trujillo 0905-15-8941 (100%)

Gilberth Ozziel Zeceña Garcia 0905-15-7542 (100%)

Claudia Elizabeth Cardona Montoya 0905-17-10611 (100%)

Lesli Mayarith Flores Palma 0905-17-2218 (100%)

Karen Rocío Vega Gudiel 0905-17-12697 (100%)

Erickzon Alexis Villanueva Virula 0905-17-12538 (100%)

Héctor Luis Washington Arana 0905-17-6678 (100%)

Carlos Fernando Cantoral Vega 0905-16-19435 (100%)

Oscar Daniel Jacinto Cifuentes 0905-16-4554 (100%)

Mynor Orlando Ruano Peña 0905-17-14549 (100%)

Bryan Gerardo Carrillo Salguero 0905-16-6879 (100%)

Fersin Adeli García Corado 0905-17-18262 (100%)

Cristhofer José Rodríguez Rueda 0905-17-4844 (100%)

Christopher Giancarlo Rivera Cerón 0905-17-7024 (100%)

Kevin Gersson Itzep Ixcoy 0905-17-4851 (100%)

Douglas Alexander Samayoa Carrillo 0905-17-2743 (100%)

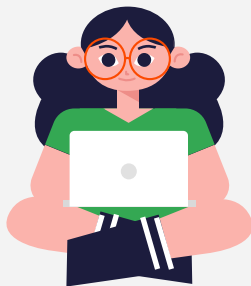
Jezer Adonias Carrillo Boteo 0905-17-3703 (0%)

Artificial Intelligence



Redes Neuronales Artificiales

Start!





Redes Neuronales Artificiales

Las redes neuronales son modelos simples del funcionamiento del sistema nervioso. Las unidades básicas son las neuronas, que generalmente se organizan en capas. Un modelo simplificado que emula el modo en que el cerebro humano procesa la información: Funciona simultaneando un número elevado de unidades de procesamiento interconectadas que parecen versiones abstractas de neuronas. Como modelo computacional, las RNA utilizan grafos y funciones, conformadas por elementos de proceso (EP o nodos) y conexiones (enlaces). Procesan entradas y generan salidas que ayudan a resolver problemas. En algunos modelos se utiliza memoria local en los nodos o elementos de proceso. Los nodos y conexiones de la red neuronal se organizan en capas.





Las características básicas de las redes neuronales artificiales son:



1

Aprenden de la experiencia

2

Generalizan de ejemplos anteriores a los ejemplos nuevos

3

Abstracción de la esencia de las entradas

M

T

X

T

F



Funcionamiento de las redes neuronales artificiales

Las redes neuronales artificiales se inspiran en el sistema nervioso y el comportamiento biológico para crear un sistema interconectado jerárquico de neuronas artificiales que cooperan entre sí para procesar datos de entrada y generar resultados.



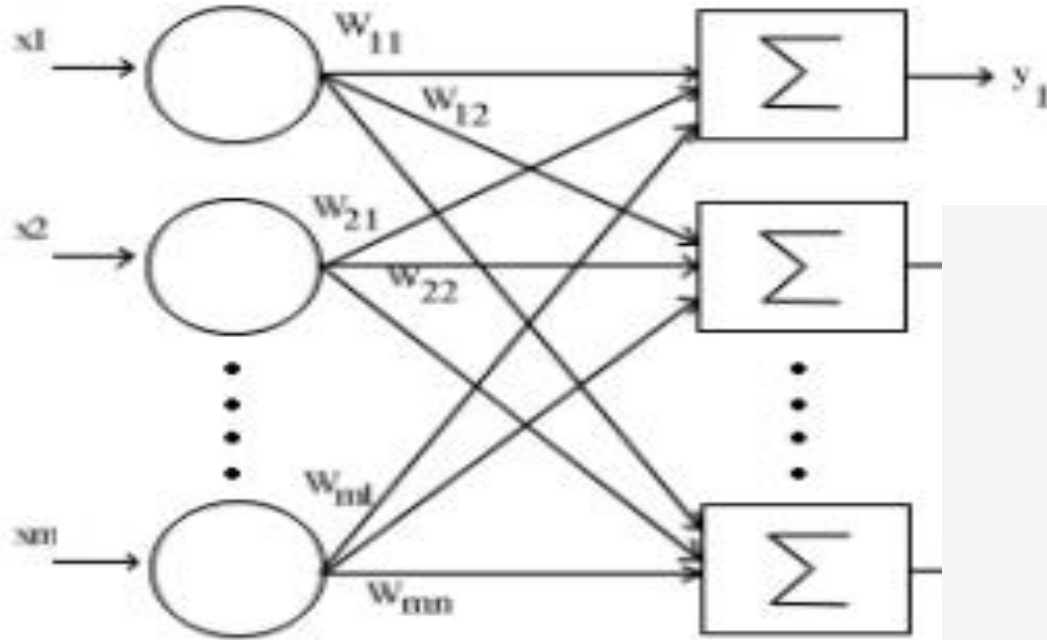


Redes de capa simple

Aunque las neuronas individuales pueden ejecutar modelos funcionales simples, están organizadas de manera más eficiente en redes. La red más simple consiste en un conjunto de perceptrones, unos conjuntos de patrones de entrada ingresan al perceptrón y proporcionan la salida correspondiente. Para cada perceptrón presente en la red, tendremos una salida que se verá como un solo perceptrón, multiplicando la suma de todas las entradas por el peso. Cuando se dibujan gráficos de red, se agrega una "capa" inicial. Esta capa no se usa con fines informáticos, solo se usa para distribuir la entrada entre los perceptrones. Lo llamamos la capa 0.



De esta manera, la representación gráfica de una red de capas simple se verá así:

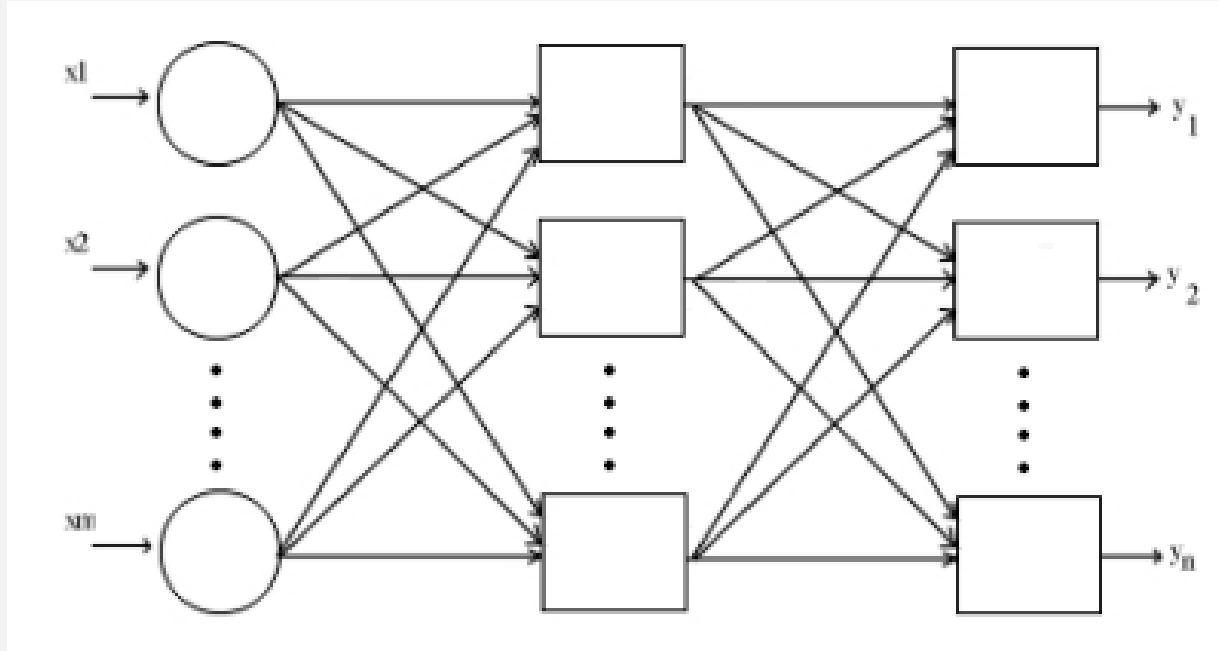




Redes multicapa

Una red multicapa consiste en un conjunto de redes en cascada de una sola capa unidas por pesos, donde la salida de una capa es la entrada de la siguiente. Por lo general, pueden aprender funciones que las redes de capas simples no pueden aprender, lo que proporciona una mejor potencia de cálculo. Para aumentar esta potencia de cálculo, debe haber una función de activación no lineal entre las capas, por lo que generalmente se usa una función de activación sigmoidea para afectar la linealidad o el umbral.

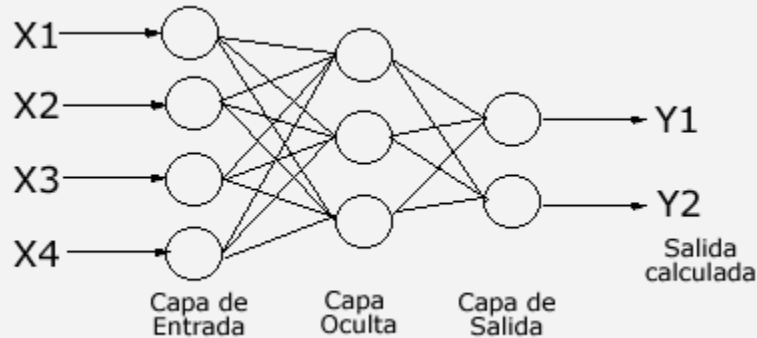
- Para calcular la salida de una red multicapa, se debe operar de la misma manera que una red de una sola capa, teniendo en cuenta que la salida de una capa es la entrada de la siguiente capa:





Redes recurrentes

Hasta ahora, la red considerada no tiene conexión entre los pesos de la salida de una capa a la entrada de la misma capa o antes. Una red con esta característica se denomina red cíclica. La red recurrente no tiene memoria, es decir, la salida solo está determinada por la entrada y los pesos. La capa de bucle redirige la salida anterior a la entrada. Su salida depende de su entrada y salida previa, por lo que puede ser similar a la memoria a corto plazo humana.





Utilidad de las redes neuronales artificiales

Las redes neuronales se diferencian de otros modelos de IA en tener la capacidad de aprender en forma automática. Este proceso también es conocido como machine learning o aprendizaje de máquina.

Algunas de las aplicaciones generales de las redes neuronales artificiales son:

- Sistemas inteligentes para la toma de decisiones en la gestión empresarial.
- Predicción.
- Reconocimiento de tendencias.
- Reconocimiento de patrones y gestión de riesgo, aplicados por ejemplo en la detección de fraude.
- Artefactos inteligentes con capacidad de aprendizaje, por ejemplo, los homepods o altavoces inteligentes.



¿Cómo se entrena una red neuronal?

Entrenar una red neuronal consiste en ajustar cada uno de los pesos de las entradas de todas las neuronas que forman parte de la red neuronal, para que las respuestas de la capa de salida se ajusten lo más posible a los datos que conocemos.



¿Cuál es el objetivo de las redes neuronales?



1

El objetivo principal de este modelo es aprender modificándose automáticamente a si mismo de forma que puede llegar a realizar tareas complejas que no podrían ser realizadas mediante la clásica programación basada en reglas. De esta forma se pueden automatizar funciones que en un principio solo podrían ser realizadas por personas.

2

3

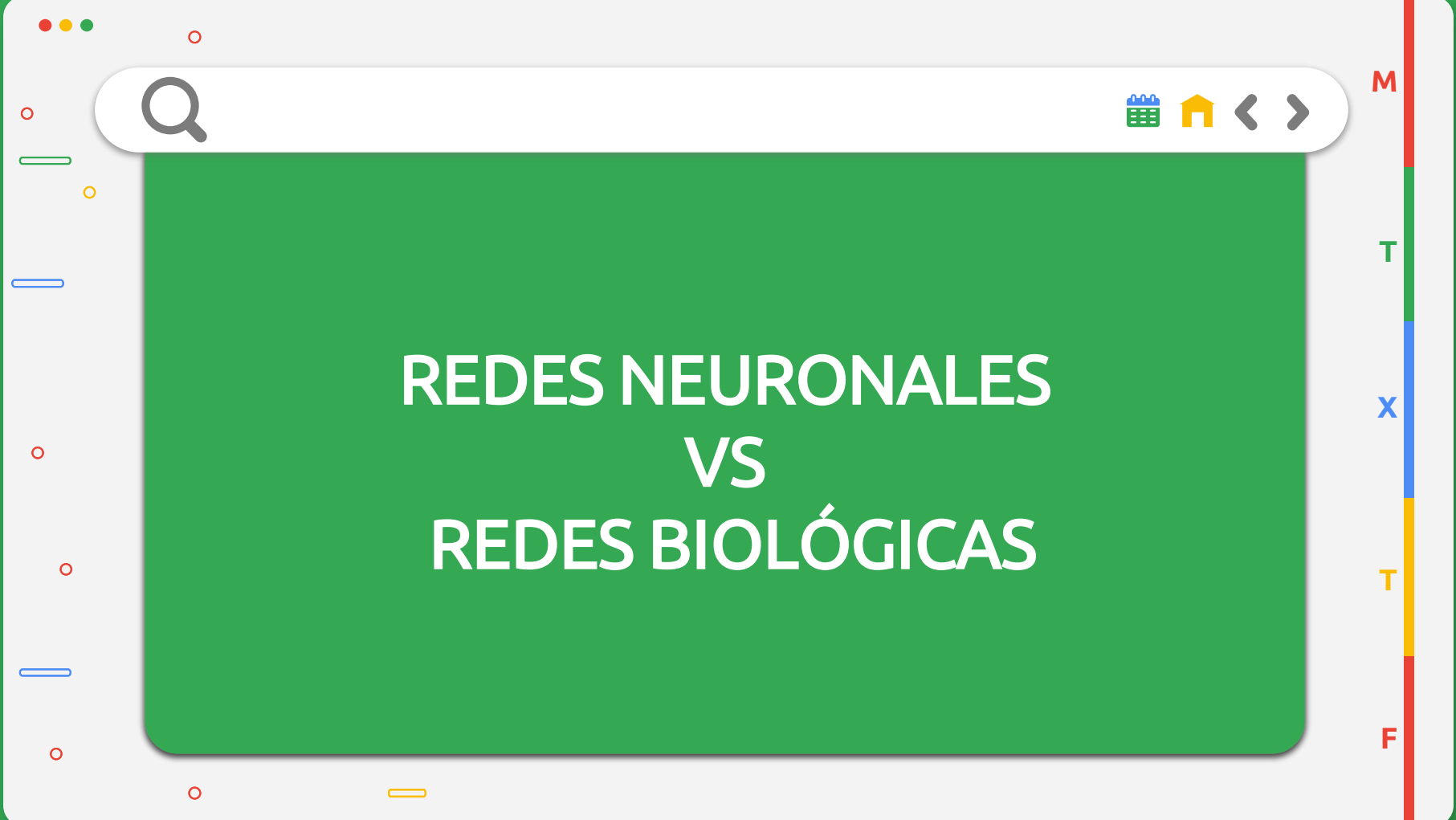
4



Usos exitosos de redes neuronales artificiales

Captar clientes mediante el envío manual de mensajes de correo electrónico es un proceso que puede ser poco eficiente. Microsoft utilizó el software de red neuronal BrainMaker para determinar cuáles son los clientes más probables en convertirse en compradores, tras recibir correos de campañas de e-marketing.

Se utilizaron como datos la fecha de la compra, la cantidad y el tipo de producto adquirido. Microsoft aumentó del 4,9 % al 8,2 % la tasa efectiva de respuesta al correo directo. Esto implica para la empresa una campaña más efectiva de mercadeo, obteniendo los mismos ingresos con 35 % menos de costos.



REDES NEURONALES VS REDES BIOLÓGICAS



REDES NEURONALES

Las redes neuronales artificiales fueron inspiradas a partir del funcionamiento del cerebro y sus neuronas, por lo que poseen semejanzas relevantes pero con diferencias notorias, a continuación se explicaran brevemente ambos aspectos.

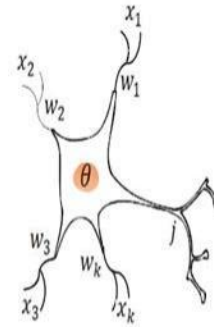




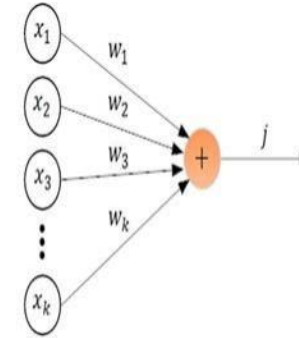
Semejanza



La estructura básica de la Neurona posee dendritas por las cuales recibe los estímulos, un núcleo donde son procesados estos estímulos y axones a través de los cuales transmite la respuesta. Las neuronas artificiales se diseñaron siguiendo esta estructura, poseen conexiones de entrada (dendritas) donde reciben la información, un elemento procesador (núcleo) y conexiones de salida (axones)



a) Neurona biológica



a) Neurona artificial

1

2

3

4

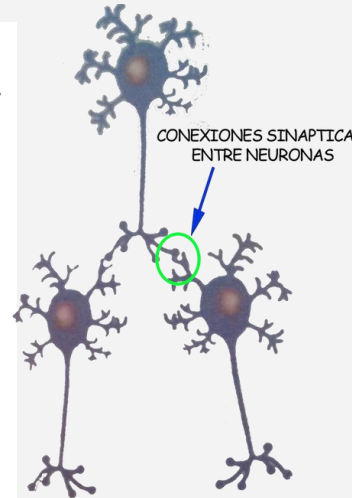
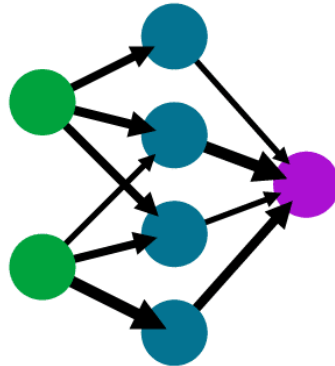


Semejanza

Las neuronas en el cerebro se pueden comunicar entre si mediante las dendritas y axones, llegando a tener cientos de millones de conexiones, formando una Red Neuronal Biológica, lo que permite el aprendizaje, reconocer patrones, entre otros. Ya que para un estímulo pueden trabajar cientos de miles de neuronas. Este principio es el que se utilizó para crear las Redes Neuronales Artificiales.

A simple neural network

input layer hidden layer output layer

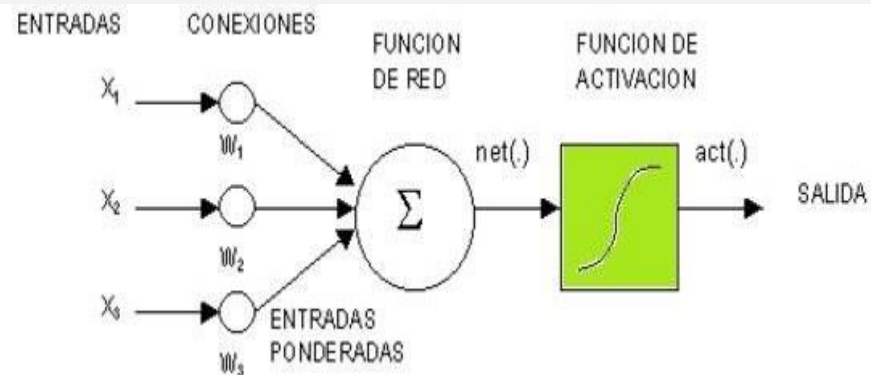
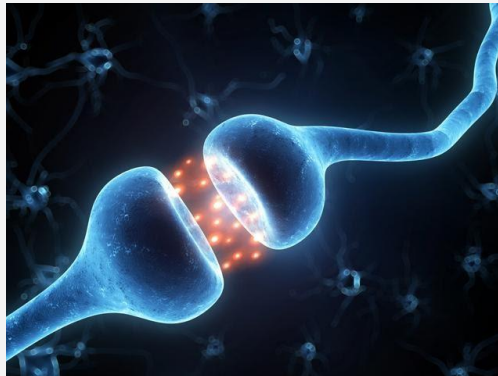




Semejanza

Al procesar un estímulo, las neuronas comienzan a producir neurotransmisores mediante los cuales envían una respuesta a otras neuronas, de no producir la cantidad necesaria de transmisores no envían una respuesta, es decir no hay comunicación.

En las redes artificiales, “La cantidad” para activar una neurona viene dada por los pesos sinápticos y función de activación, si esta no se cumple, no ocurre el envío de información.





1



2

Diferencias Redes Neuronales

Contact

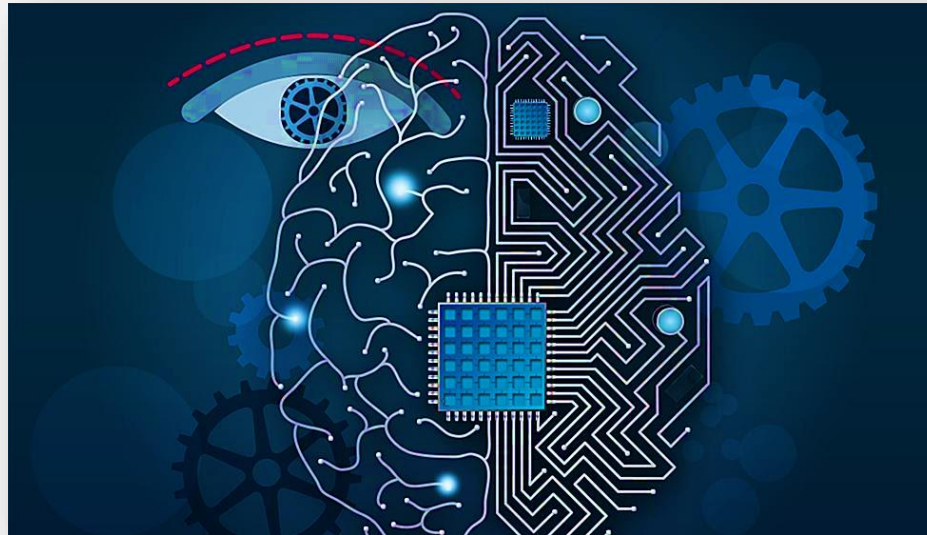
3

4



Diferencias

La principal es que las Redes Neuronales Biológicas como bien su nombre se indica, están compuestas por células vivas las neuronas, mientras que las Redes Neuronales Artificiales son unidades procesadoras, es decir elementos informáticos.





Diferencias

Las neuronas debido a enfermedades, consumo de drogas, traumas, entre otros, pueden morir o sufrir daños severos, sin embargo en algunos casos, esto no indica un cese del funcionamiento del cerebro, ya que otras neuronas pueden ser entrenadas para cumplir con las funciones de las faltantes. En cambio, si una neurona artificial sufre daños, esto afecta el desempeño de toda la red.





Diferencias

A pesar de que el proceso de aprendizaje es similar, las RNA parten de reglas y funciones establecidas previamente, por lo que aquello que no se encuentre dentro de esta base de conocimiento o se derive directamente de ella, no será reconocido, mientras que RNB son capaces de procesar distintos aspectos y generalizarlos, por ejemplo una RNA diseñada para reconocer rostros humanos, no detectaría la copa.





Diferencias

Las Redes Neuronales Biológicas son sistemas complejos, que controlan no solo procesos mentales como el razonamiento, sino también motores; en cambio las Redes Neuronales Artificiales se enfocan principalmente en el aspecto analítico, llegando a ser mucho mas rápidos en el procesamiento de la información que el cerebro.





Ejemplo práctico de una RNA

Red Neuronal en Python con Keras y Tensorflow

Crearemos una red neuronal artificial muy sencilla en Python con Keras y Tensorflow

- para comprender su uso. Implementaremos la compuerta XOR e intentaré comparar las ventajas del aprendizaje automático frente a la programación tradicional. Utilizaremos las compuertas XOR.

Funcionamiento

Tenemos dos entradas binarias (1 ó 0) y la salida será 1 sólo si una de las entradas es verdadera (1) y la otra falsa (0).

Es decir que de cuatro combinaciones posibles, sólo dos tienen salida 1 y las otras dos serán 0, como vemos aquí:

- $XOR(0,0) = 0$

$$XOR(0,1) = 1$$

$$XOR(1,0) = 1$$

$$XOR(1,1) = 0$$



Q ¿Keras y Tensonflow?

Utilizaremos Keras, que es una librería de alto nivel, para que nos sea

más fácil describir las capas de la red que creamos y en background es

- decir, el motor que ejecutará la red neuronal y la entrenará estará la implementación de Google llamada Tensorflow.

Una Red Neuronal Artificial sencilla con Python y Keras

Veamos el código completo en donde creamos una red neuronal con

datos de entrada las 4 combinaciones de XOR y sus 4 salidas ordenadas.

Luego analizamos el código línea a línea.





```
1 import numpy as np
2 from keras.models import Sequential
3 from keras.layers.core import Dense
4
5 # cargamos las 4 combinaciones de las compuertas XOR
6 training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
7
8 # y estos son los resultados que se obtienen, en el mismo orden
9 target_data = np.array([[0],[1],[1],[0]], "float32")
10
11 model = Sequential()
12 model.add(Dense(16, input_dim=2, activation='relu'))
13 model.add(Dense(1, activation='sigmoid'))
14
15 model.compile(loss='mean_squared_error',
16               optimizer='adam',
17               metrics=['binary_accuracy'])
18
19 model.fit(training_data, target_data, epochs=1000)
20
21 # evaluamos el modelo
22 scores = model.evaluate(training_data, target_data)
23
24 print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
25 print (model.predict(training_data).round())
```



Análisis del Código

Importación de las clases utilizadas

```
1 import numpy as np
2 from keras.models import Sequential
3 from keras.layers.core import Dense
```

Utilizaremos numpy para el manejo de arrays. De Keras importamos el tipo de modelo Sequential y el tipo de capa Dense que es la normal.



Creamos los arrays de entrada y

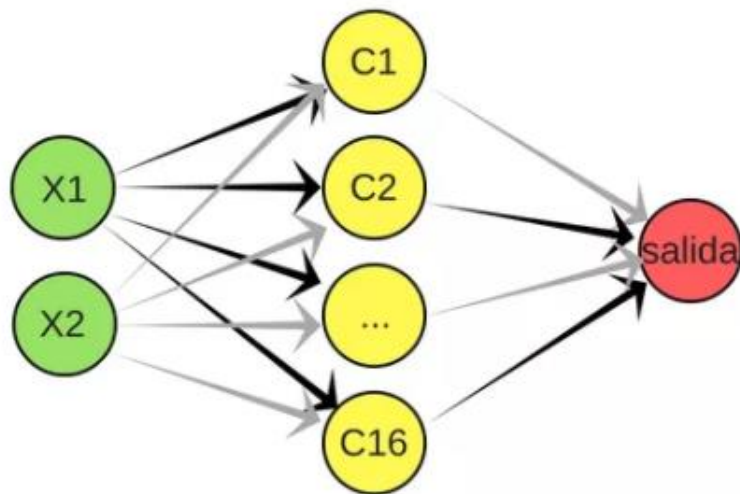
```
1 # cargamos las 4 combinaciones de las compuertas XOR
2 training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
3
4 # y estos son los resultados que se obtienen, en el mismo orden
5 target_data = np.array([[0],[1],[1],[0]], "float32")
```

Arquitectura de Red Neuronal

```
1 model = Sequential()
2 model.add(Dense(16, input_dim=2, activation='relu'))
3 model.add(Dense(1, activation='sigmoid'))
```



Primero creamos un modelo vacío de tipo Sequential. Este modelo se refiere a que crearemos una serie de capas de neuronas secuenciales, una delante de otra. Agregamos dos capas Dense con “model.add()”. Realmente serán 3 capas, pues al poner **input_dim=2** estamos definiendo la capa de entrada con 2 neuronas (para nuestras entradas de la función XOR) y la primera capa oculta (hidden) de 16 neuronas. Como función de activación utilizaremos “relu” que sabemos que da buenos resultados. Podría ser otra función, esto es un mero ejemplo, y según la implementación de la red que haremos, deberemos variar la cantidad de neuronas, capas y sus funciones de activación. Y agregamos una capa con 1 neurona de salida y función de activación sigmoid.



La arquitectura de 3 capas que creamos para esta Red Neuronal Artificial



Entrenamiento de la Red

```
1 model.compile(loss='mean_squared_error',  
2               optimizer='adam',  
3               metrics=['binary_accuracy'])
```

- Con esto indicamos el tipo de pérdida (loss) que utilizaremos, el “optimizador” de los pesos de las conexiones de las neuronas y las métricas que queremos obtener.

```
1 model.fit(training_data, target_data, epochs=1000)
```

- Indicamos con model.fit() las entradas y sus salidas y la cantidad de iteraciones de aprendizaje (epochs) de entrenamiento. En un modelo más grande y complejo, se necesitarán más iteraciones y a la vez será más lento el entrenamiento.

Resultado del entrenamiento

```
1 Epoch 1/1000
2 4/4 [=====] - 0s 43ms/step - loss: 0.2634 - binary_accuracy: 0.5000
3 Epoch 2/1000
4 4/4 [=====] - 0s 457us/step - loss: 0.2630 - binary_accuracy: 0.2500
```

Si vemos las salidas del entrenamiento, con esto vemos que la primera

- iteración tuvo acierto la mitad de las salidas (0.5) pero a partir de la segunda, sólo acierta 1 de cada 4 (0.25).

Luego en la “epoch” 24 recupera el 0.5 de aciertos, ajusta correctamente los pesos de la red.

```
1 Epoch 24/1000
2 4/4 [=====] - 0s 482us/step - loss: 0.2549 - binary_accuracy: 0.5000
3
4 Epoch 107/1000
5 4/4 [=====] - 0s 621us/step - loss: 0.2319 - binary_accuracy: 0.7500
6
7 Epoch 169/1000
8 4/4 [=====] - 0s 1ms/step - loss: 0.2142 - binary_accuracy: 1.0000
```



- Y en la iteración 107 aumenta los aciertos al 0,75 (son 3 de 4) y en la iteración 169 logra el 100% de aciertos y se mantiene así hasta finalizar. Como los pesos iniciales de la red son aleatorios, puede que las salidas que tengas en tu ordenador sean levemente distintas en cuanto a las iteraciones, pero llegarás a la precisión binaria (binara_accuracy) de 1.0

Q Evaluación y Predicción

— Primero se evalúa el modelo

```
1 scores = model.evaluate(training_data, target_data)
2 print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

○ Y vemos que tuvimos un 100% de precisión (recordemos lo trivial de este ejemplo).

Y hacemos las 4 predicciones posibles de XOR, pasando nuestras entradas:

```
1 print (model.predict(training_data).round())
```

○ y vemos las salidas 0,1,1,0 que son las correctas. ○



Afinando parámetros de la RNA

- Recordemos que este es un ejemplo muy sencillo y con sólo 4 entradas posibles. Pero si en la realidad tuviéramos una red compleja, deberemos poder ajustar muchos parámetros, repasemos:
 - Cantidad de capas de la red (en nuestro caso son 3)
 - Cantidad de neuronas en cada red (nosotros tenemos 2 de entrada, 16 en capa oculta y 1 de salida)
 - Funciones de activación de cada capa. Nosotros utilizamos relu y sigmoid
 - Al compilar el modelo definir las funciones de pérdida, optimizer y métricas.
 - Cantidad de iteraciones de entrenamiento.
 - En este ejemplo que es muy sencillo, puedes intentar variar por ejemplo la cantidad de neuronas de entrada, probar con 8 o con 32 y ver qué resultados obtienes.



Para guardar y cargar la red, se utiliza el siguiente código. Lo que hacemos es guardar esa red y en otro código cargaríamos la red y la utilizamos como si fuera una librería o una función que creamos.

Pasándole entradas y obteniendo las predicciones.

```
1 # serializar el modelo a JSON
2 model_json = model.to_json()
3 with open("model.json", "w") as json_file:
4     json_file.write(model_json)
5 # serializar los pesos a HDF5
6 model.save_weights("model.h5")
7 print("Modelo Guardado!")
8
9 # mas tarde...
10
11 # cargar json y crear el modelo
12 json_file = open('model.json', 'r')
13 loaded_model_json = json_file.read()
14 json_file.close()
15 loaded_model = model_from_json(loaded_model_json)
16 # cargar pesos al nuevo modelo
17 loaded_model.load_weights("model.h5")
18 print("Cargado modelo desde disco.")
19
20 # Compilar modelo cargado y listo para usar.
21 loaded_model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])
```

¿Qué es Underfitting?



se refiere al escenario en el que un modelo de aprendizaje automático no puede generalizarse o encajar bien en un conjunto de datos invisible. Una clara señal de sobreajuste del aprendizaje automático es si tu error en el conjunto de datos de prueba o validación es mucho mayor que el error en el conjunto de datos de entrenamiento.

El subajuste es un término utilizado en las estadísticas que se refiere a un error de modelado que se produce cuando una función se corresponde demasiado con un conjunto de datos. Como resultado, el sobreajuste puede no ajustarse a datos adicionales y esto puede afectar la precisión de la predicción de observaciones futuras. Es lo contrario al overfitting.



¿Cuáles son sus causas?



1

Las principales causas del underfitting o sobreajuste son:

- No hay suficientes parámetros o complejidad para modelar adecuadamente los datos
- Los priores bayesianos son demasiado restrictivos o ciertos (baja entropía)
- No se le dio suficiente tiempo al algoritmo de aprendizaje automático para entrenar.

2

3

4



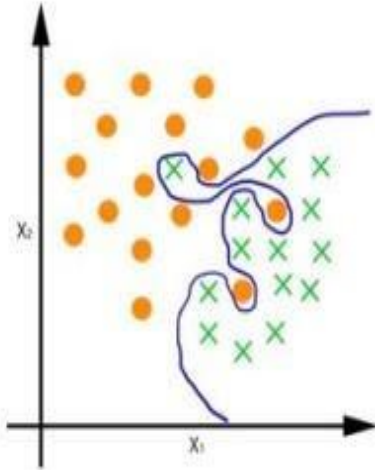
¿Y sus consecuencias?

Underfitting es lo que ocurre cuando nuestro modelo es muy simplista, insuficiente para capturar los matices, particularidades y complejidades en los datos.

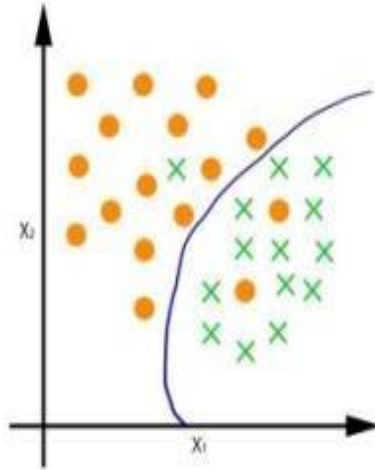
Cuando estudiamos la noche anterior a un examen, sólo ojeando breve y superficialmente los conceptos, estamos incurriendo en “underfitting”, ya que no hicimos el trabajo necesario para aprender, sino que nuestro exceso de confianza nos llevó a pensar que entendíamos algo que, en esencia, es más complejo. El resultado natural de esta subestimación de los datos es que no sólo nos desempeñaremos mal en el examen, sino muy posiblemente en los ejercicios que se encuentran en el libro.



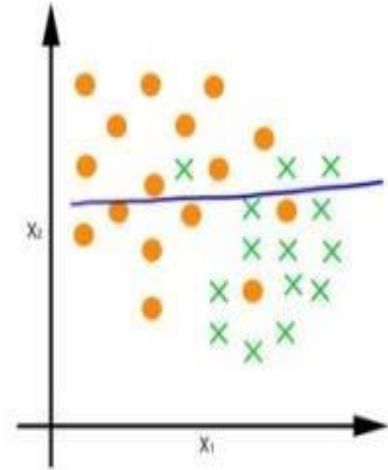
Sobreajuste (Overfitting) y sobregeneralización (underfitting)



Overfitting



OK



Underfitting

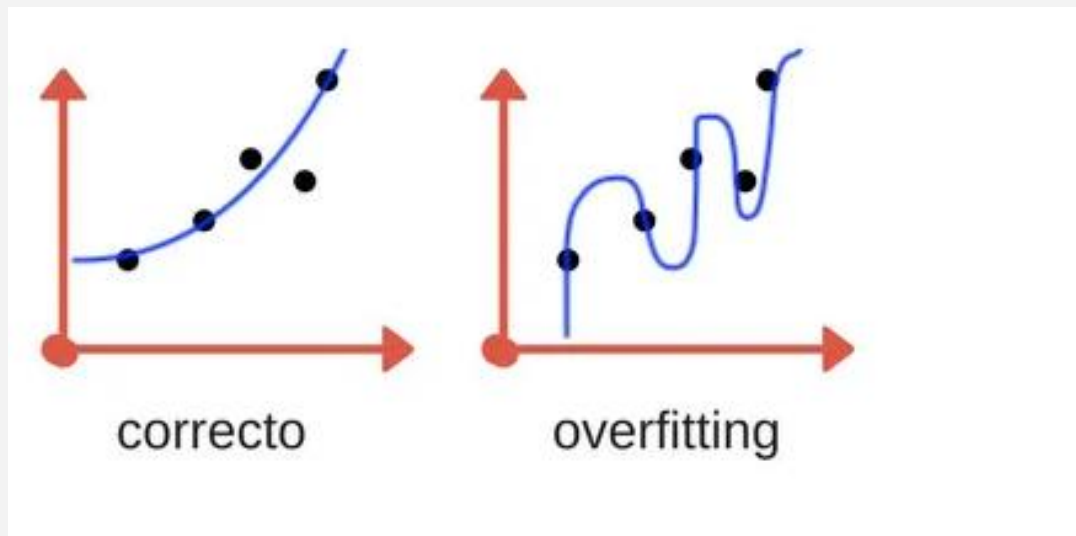


Overfitting

Es muy común que al comenzar a aprender machine learning caigamos en el problema del Overfitting. Lo que ocurrirá es que nuestra máquina sólo se ajustará a aprender los casos particulares que le enseñamos y será incapaz de reconocer nuevos datos de entrada. En nuestro conjunto de datos de entrada muchas veces introducimos muestras atípicas en alguna de sus dimensiones, o muestras que pueden no ser del todo representativas.

Cuando “sobre-entrenamos” nuestro modelo y caemos en el overfitting, nuestro algoritmo estará considerando como válidos sólo los datos idénticos a los de nuestro conjunto de entrenamiento –incluidos sus defectos– y siendo incapaz de distinguir entradas buenas como fiables si se salen un poco de los rangos ya preestablecidos.

Q Overfitting



Q ¿Cómo evitarlo?

- Dividir nuestros datos en training, validación y testing.
- Obtener un mayor número de datos.
- Ajustar los parámetros de nuestros modelos.
- Utilizar modelos más simples.
- Los datos vienen de distintas distribuciones.
- Bajar el número de iteraciones en los algoritmos iterativos.

Ejemplo

Si entrenamos a nuestra máquina con 10 razas de perros sólo de color marrón de manera rigurosa y luego enseñamos una foto de un perro blanco, nuestro modelo no podrá reconocerlo cómo perro por no cumplir exactamente con las características que aprendió (el color forzosamente debía ser marrón). Aquí se trata de un problema de overfitting.