Onna Bugeisha

Guerreras de la Ciencia

Hackers & Developers™ Press
2020

Onna Bugeisha – Guerreras de la Ciencia

ISBN 978-987-86-3950-5

Editado por Eugenia Bahit
1ª edición, marzo de 2020. Avellaneda, Buenos Aires. Rep. Argentina.
Hecho el depósito que marca la Ley 11.723.

Publicado por Hackers & Developers™ Press.
483 Green Lanes. N13 4BS, London. United Kingdom.
https://www.hdmag.press/

Compilación de glosario: Maiara Beatriz Cercasi
Revisión de artículos: A. Navarro, E. Bahit, F. Hernández, J. Monsegur y M. Mora.

Diseño de portada: © 2019 Eugenia Bahit. CC BY-NC-ND 4.0
Diseño y maquetación interior: © 2019 Eugenia Bahit. CC BY-NC-ND 4.0

A todas las mujeres que dedican su vida a la ciencia, luchando por la igualdad y la equidad en el mundo.

Andrea Navarro Moreno

Autora

(Argentina, 1989)

Ingeniera en Informática por la Universidad de Mendoza, Argentina. Especializada en Inteligencia Artificial, se desempeña como codirectora del Laboratorio de Inteligencia Artificial de la sede San Rafael de la Facultad de Ingeniería de la UM, donde también ejerce como docente de las cátedras de Programación, e Inteligencia Artificial. En 2016, cofundó su propia empresa de formación, Junco TIC, donde además de impartir cursos, se encuentra a cargo de la redacción de artículos técnicos y educativos.

INFORMÁTICA APLICADA

Introducción a las Redes Neuronales aplicadas al Aprendizaje Supervisado

Andrea Navarro Moreno

Definiciones previas

Red neuronal artificial

Las *redes neuronales artificiales* son un conjunto de unidades de procesamiento llamadas *neuronas artificiales* conectadas entre sí a través de conexiones ponderadas que permiten la transmisión de información.

Aprendizaje supervisado

Se refiere al proceso mediante el cuál un programa informático puede realizar predicciones, a partir de patrones que aprende continuamente con base en datos. Dicho en otras palabras, es la automatización del proceso de creación de modelos analíticos que permiten que un programa informático se adapte a situaciones nuevas (Ana Loyo, 2018).

Base biológica: la neurona

La neurona es una célula del sistema nervioso cuyo funcionamiento puede compararse con el de un interruptor que será activado si recibe un estímulo de entrada suficiente y de ser así, enviará un pulso de salida. Anatómicamente, se compone de las siguientes partes:

1. **Dendritas:** ramificaciones protoplasmáticas que sirven de canal de entrada al soma neuronal, para los estímulos provenientes de los axones de las neuronas aledañas.

2. **Soma neuronal:** es el cuerpo celular de la neurona en el que se encuentra el núcleo —sección central de la neurona donde se generan los estímulos—, rodeado por el citoplasma.

3. **Axón:** prolongación fibrosa cubierta por una vaina de mielina que sirve como canal de salida de los estímulos generados en el soma, realizando conexiones sinápticas con otras neuronas.

La transmisión de señales entre neuronas es un proceso que se inicia cuando las dendritas reciben un estímulo sináptico. Esta sinapsis puede ser química o eléctrica. La *sinapsis química* se produce por la liberación de neurotransmisores en mayor o menor medida, aumentando o disminuyendo la intensidad del estímulo en el núcleo, mientras que la *sinapsis eléctrica* se produce por un impulso nervioso. Cuando las señales recibidas se acumulan en el soma superando un cierto umbral, se produce un nuevo estímulo que viaja a través del axón y se transporta a las neuronas vecinas.

Neurona artificial

Las redes neuronales artificiales son una simplificación matemática del modelo neuronal biológico que consiste en un conjunto de unidades de procesamiento (neuronas artificiales) interconectadas entre sí (Kriesel, 2007).

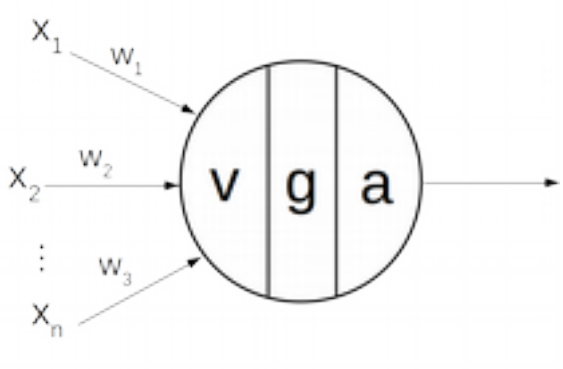
Esta versión simplificada cuenta con los siguientes componentes:

Sinapsis o señales de entrada: cada sinapsis cuenta con un dato de entrada que representa la señal enviada, y además, con un peso asociado. Este peso representa el nivel de estímulo que acompañará a cada señal.

Función de agregación o sumatorio: determina cómo la neurona acumulará las señales de entrada recibidas. Una aplicación común es realizar la suma de las señales de entrada ponderadas utilizando el peso asociado a cada sinapsis.

Función de activación: calcula la señal de salida de la neurona a partir del resultado de la función de propagación. La elección de la función de activación dependerá de los tipos de datos utilizados, la arquitectura de la red y el algoritmo de aprendizaje.

Señal de salida: el resultado de la función de activación que saldrá de la neurona.



2. Ilustración: Neurona artificial

Donde:

$x = Señal\ de\ entrada$

$w = Peso$

$v = Función\ de\ agregación$

$g = Función\ de\ activación$

$a = Señal\ de\ salida$

Primeras redes neuronales artificiales

McCulloch-Pitts MPN

Warren S. McCulloch y Walter Pitts introdujeron en 1943 el modelo de neurona simplificado MPN con el objetivo de entender la capacidad del cerebro para producir y entender patrones complejos (Kriesel, 2007) (Russell & Norvig, 2009). La función de agregación de esta neurona consiste en una suma de las señales de entrada ponderadas:

$$v = \sum_{i=1}^{n} x_i w_i$$

Los pesos de las conexiones son constantes por lo que no es posible el aprendizaje para este tipo de red. La función de activación es del tipo binaria, donde la función es activada si se supera un cierto umbral θ.

$$f(v) = \begin{cases} 1, v \geq \theta \\ 0, v < \theta \end{cases}$$

Una práctica común consiste en integrar el umbral dentro de la neurona como si se tratara de una entrada más x_0 cuya señal siempre es enviada $x_0 = 1$ y su peso es equivalente al negativo del valor del umbral $w_0 = -\theta$. De esta manera la nueva función de agregación se modificaría para ser:

$$v = \sum_{i=0}^{n} x_i w_i$$

y la función de activación podría expresarse como:

$$f(v) = \begin{cases} 1, v \geq 0 \\ 0, v < 0 \end{cases}$$

Perceptrón

En 1949 fue introducida la regla de *Hebb*. Esta técnica teorizada por Donald Hebb permitía que las redes neuronales realizaran un aprendizaje a través del debilitamiento y refuerzo de las sinapsis (Russell & Norvig, 2009).

En en 1962, Frank Rosenblatt la aplicó a un modelo MPN generando así lo que se conocería como perceptrón. A diferencia de la neurona MPC, el perceptrón permite un aprendizaje supervisado comparando la salida esperada con la salida real de la neurona y ajustando los pesos de las entradas hasta que el error disminuya o desaparezca. Sus funciones de activación más comunes son la función binaria ya utilizada por MPN, la función de signo y la función sigmoide.

Función de signo

$$f(v)=\begin{cases} 1, v \geq 0 \\ -1, v < 0 \end{cases}$$

Función sigmoide

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

Solo debe utilizarse en el caso que se espere siempre una salida negativa ya que devuelve valores entre 0 y 1.

```
para cada conexión:
       establecer valor de peso aleatorio
mientras que no se identifiquen correctamente todos los ejemplos:
       para cada ejemplo del conjunto de entrenamiento:
              para cada neurona:
                     calcular señal de salida
            calcular error
              por cada conexión:
                     actualizar pesos
```

El **error** de la red para cada ejemplo presentado se calcula restando la salida esperada b y la señal de salida a: $\delta = b - a$

Luego el peso de cada conexión sináptica es modificado realizando un incremento o decremento resultante de la multiplicación del error de la red δ, la entrada correspondiente a esa conexión y un parámetro llamado **tasa de aprendizaje** η. La tasa de aprendizaje determina qué tanto se ajustan los pesos durante el entrenamiento. Una tasa muy alta generará un aprendizaje más rápido pero aumentará el riesgo de que la red nunca logre clasificar correctamente todos los ejemplos, mientras que una tasa muy pequeña generará un aprendizaje muy lento. La modificación de pesos se calcula entonces mediante la ecuación:

$$w_i = w_i + \delta \eta x_i$$

Una vez ajustados los pesos se presenta a la red un nuevo ejemplo del grupo de entrenamiento y se repite el proceso. Cuando un perceptrón ha sido capaz de clasificar correctamente todos los ejemplos brindados se dice que esta ha convergido.

No todo problema, sin embargo, puede ser representado por un perceptrón. El teorema de convergencia del perceptrón asegura el aprendizaje en un número infinito de pasos para problemas linealmente separables. Se dice que una función es linealmente separable si su salida puede ser separada o discriminada por una función que se componga de una combinación lineal de características.

Perceptrones multicapa

Los perceptrones multicapa (en inglés, *Feed Forward* —FF—), se caracterizan por tener sus neuronas divididas en capas, comenzando por la capa de entrada

por la que se recibirá la señal entrante de la red, y terminando con la capa de salida por donde la red enviará su señal de salida, pudiendo existir entre ellas, capas intermedias denominadas *capas ocultas*. Cuando una red de estas características tiene más de una capa oculta suele denominarse *Deep Feed forward (DFF)*. Las señales solo se mueven en una sola dirección desde la entrada hacia la salida, no permitiendo bucles o conexiones a neuronas de la misma capa o capas anteriores (Gurney) (Kriesel, 2007).

Dependiendo de las características del problema y el algoritmo de aprendizaje, las neuronas de estas redes pueden tener diferente funciones de activación. Algunas de las más comunes son la función sigmoide, la tangente hiperbólica, ReLU y Leaky ReLU.

Retropropagación

Retropropagación (en inglés, *backpropagation*) es un algoritmo que permite el entrenamiento de redes *FF* que contengan una o más capas ocultas. A diferencia de los perceptrones simples es necesario determinar el nivel de injerencia o "responsabilidad" de cada neurona en el error total de la red, al calcular la modificación de pesos que sufrirá cada conexión de manera que la modificación de dicha sinapsis disminuya el error total de la red.

Al igual que en el algoritmo de aprendizaje anterior, se presentarán uno a uno los ejemplos de entrenamiento de la red, se calculará el error y se ajustarán los pesos. Este proceso se repetirá hasta que el error esté dentro del rango aceptado. Para cada ejemplo presentado, el algoritmo de retropropagación

ejecuta dos etapas: la **propagación hacia adelante** y la **propagación hacia atrás.**

La **propagación hacia adelante** consiste en presentar el ejemplo a la capa de entrada y calcular la salida de cada una de las neuronas; luego se calcula la salida de cada una de las neuronas de la capa oculta, y así sucesivamente, hasta obtener la salida de las neuronas de la capa de salida. La función de activación de las neuronas en esta red debe ser no lineal y tener una derivada calculable por lo que una función muy utilizada es la función sigmoide.

En la **propagación hacia atrás** se calcula el error comenzando desde la capa de salida. Se convierte este error en una derivada del mismo y se propaga este dato a la capa anterior, para calcular la derivada de error para cada una de las neuronas. Este proceso se continúa retrocediendo a través de las capas ocultas hasta llegar a la capa de entrada.

El error de una neurona de la capa de salida se calcula como la diferencia de la salida esperada y la salida de dicha neurona multiplicada por la derivada de la función de activación:

$$\delta = f'(v)(b-a)$$

Para las neuronas de capas ocultas, el error se calcula como la suma de la diferencia entre los errores de las neuronas de la capa siguiente y el peso de las conexiones con estas neuronas:

$$\delta = f'(v)\sum_{k}(\delta_k - w_k)$$

Luego de calcular los errores, se ajustan los pesos de las sinapsis utilizando la tasa de aprendizaje y la derivada de error de cada una de las neuronas:

$$W_{ji} = W_{ji} + \eta \delta_{ji} x_{ji}$$

Para evitar que el gradiente de error resulte en una convergencia de la red a un mínimo local es posible ingresar un nuevo parámetro a la red llamado inercia o *momentum* λ. Este valor tiene en cuenta la dirección de la modificación del peso, en una iteración de entrenamiento anterior t, lo que acelerará el aprendizaje:

$$W_{ji}(t+1) = W_{ji}(t) + \eta \delta_{ji} x_{ji}(t+1) + \eta \delta_{ji} x_{ji}(t) \lambda$$

```
para cada conexión:
      establecer valor de peso aleatorio
mientras que el valor del error no sea aceptable:
      para cada ejemplo del conjunto de entrenamiento:
            por cada capa partiendo de capa de entrada:
                  para cada neurona:
                        propagar señal de salida a siguiente capa
            por cada capa partiendo de capa de salida:
                  para cada neurona:
                        propagar error capa anterior
            por cada conexión:
                  actualizar pesos
```

Elección de arquitectura

Aunque las capas de entrada y salida tienen la cantidad de neuronas determinada por la naturaleza del problema, se necesita establecer la cantidad óptima de capas ocultas y neuronas en cada una de estas, para asegurar el correcto funcionamiento y rapidez de la red.

Si se está trabajando con problemas linealmente separables, entonces no será necesaria la utilización de capas ocultas. Para representar problemas que trabajen con conjuntos finitos de entrada a salida, una capa oculta será suficiente. Con dos o más capas ocultas es posible representar un problema con límites arbitrarios y funciones de activación racionales.

Para definir el número correcto de neuronas en capas ocultas pueden utilizarse reglas básicas que luego se irán ajustando en un proceso de prueba y error, verificando la tasa de error resultante en cada caso.

Siendo h la cantidad de neuronas en la capa oculta; i, la cantidad de neuronas en la capa de entrada; y o, la cantidad de neuronas en la capa de salida, algunas reglas de elección de neuronas ocultas comunes son las siguientes:

- La cantidad de neuronas en la capa oculta debe ser un número entre el número de neuronas en la capa de entrada y el número de neuronas en la capa de salida: $i \leq h \geq o \vee o \leq h \geq i$

- La cantidad de neuronas en la capa oculta debe ser equivalente a las dos terceras partes de la cantidad de neuronas en la capa de entrada más la cantidad de neuronas en la capa de salida:

$$h = \frac{2}{3} i + o$$

- La cantidad de neuronas en la capa oculta debe ser menor que el doble de neuronas en la capa de entrada: $h < 2 * i$

Una técnica más costosa pero más efectiva es la del **podado.** Esta técnica permite obtener la arquitectura de capa oculta más eficiente que resuelva el problema planteado, eliminando o no incluyendo neuronas innecesarias en la capa oculta. Existen dos técnicas de podado: **selectivo** e **incremental.**

En el **podado selectivo** se trabaja con redes ya entrenadas para resolver el planteo. Se evalúan los pesos sinápticos de las neuronas de capas ocultas y se procede a eliminar la que tenga el peso más pequeño, entendiendo que esta neurona es la que tiene un efecto menor en el resultado total de la red.

La red neuronal es entonces evaluada con esta nueva configuración y si la tasa de error sigue encontrándose dentro de un valor aceptable, el proceso se repite dando como resultado la red más pequeña, y por lo tanto más eficiente, capaz de resolver el problema dentro de la tolerancia de error aceptada.

```
hacer
       n=1
       si error <= error_aceptado:
              eliminar neurona n
       sino:
              n++
              si n > numero_de_neuronas:
                     evaluar red sin neurona n
mientras que se hayan eliminado neuronas en este ciclo
```

El **podado incremental** sin embargo, se realiza sobre una red sin entrenar y que por lo tanto no tiene todavía pesos asociados. En esta técnica se inicia con la red más pequeña y de manera iterativa se van agregando neuronas a la capa oculta y evaluando la tasa de error de la red hasta que este sea aceptable o se llegue a un límite de neuronas predefinido. Al no contar con un red ya entrenada es necesario entrenar cada una de las nuevas configuraciones para

poder obtener la tasa de error. Esto aumenta el tiempo de procesamiento con respecto al podado selectivo.

```
n=1
hacer
       si cantidad_neuronas >= error_aceptado:
              salir con error
       crear red con n neuronas
       entrenar red
       n++
mientras error > error_aceptado
```

Redes Neuronales convolucionales (CNN)

Una CNN es una red FF cuyas capas ocultas se componen de una combinación de tres tipos diferentes: capas convolucionales, capas de agrupamiento y capas completamente conectadas. Estas redes fueron presentadas por LeCun en 1988, y han permitido la identificación automática de información espacial contenida dentro de imágenes, sin requerir una gran cantidad de experiencia por parte de la red (Nielsen, n.d.).

Capas convolucionales: son las primeras en colocarse. Su propósito es el de convolucionar[39] datos de entrada a través del uso de unidades de datos llamadas *kernel*[40] o filtros. Cada filtro se utiliza para que la red aprenda una característica diferente de los datos de entrada. Los filtros de las primeras capas convolucionales pueden utilizarse para detectar bordes, formas, etc. A medida que la arquitectura de la red se hace más compleja, las capas

39 Convolución de Funciones (Universidad de Almería): https://w3.ual.es/~vruiz/Docencia/Apuntes/Signals/Theory/index.html#x1-2100011

40 Convolución de matrices (Universitat Politècnica de València): https://riunet.upv.es/bitstream/handle/10251/69639/4524-15767-1-PB.pdf

convolucionales más profundas pueden detectar elementos más específicos como texturas y objetos. La operación de convolución consiste en utilizar una ventana para tomar una sección del valor que sea del mismo tamaño que el filtro a aplicar. Se realiza el producto entre cada valor de los datos dentro de la ventana con el del filtro, y finalmente se suman estos valores que darán como resultado los de la señal convolucionada. Luego la ventana se desplaza una cierta distancia, según un parámetro llamado zancada (*stride*), y se repite el proceso.

Es posible aplicar varios filtros a una misma entrada obteniendo de esta manera diferente información sobre la misma imagen o dato. Esto generará una matriz n-dimensional llamada *mapa de características*. Esta operación es combinada con la función de activación que generalmente es ReLU o similar.

Función de activación

Cuando dentro de la arquitectura de la red se utilizan dos capas convolucionales secuenciales es necesario utilizar una función de activación no lineal. Comúnmente se utiliza ReLU o alguna de sus variantes.

ReLU (Unidad lineal rectificada)

$$f(v)=\frac{1}{1+e^{-x}} \text{ o } f(v)=\begin{cases} v, v \geq 0 \\ 0, v < 0 \end{cases}$$

ReLU es lineal para todos los valores positivos y/o para los valores negativos. Esta función tiene una característica desfavorable para la red, llamada *ReLU muerto*, que surge cuando una neurona mantiene valores negativos generando

que nunca pueda producirse una salida con un valor diferente de 0. Cuando esto sucede, las neuronas afectadas no cumplen ninguna función en la red.

Leaky ReLU

Es una alternativa a ReLU que soluciona el problema de *ReLU muerto* modificando el valor fijo de 0 para valores negativos por una pequeña pendiente, determinado por la siguiente función:

$$f(v)=\begin{cases} v, v \geq 0 \\ px, v<0 \end{cases}$$

donde p es una pendiente pequeña del orden de $0{,}01$.

Capas de agrupación

Estas capas permiten simplificar la dimensión de los datos eliminando parámetros y permitiendo un aprendizaje más rápido de la red. Para calcular la salida de cada una de estas neuronas es necesario recorrer nuevamente los valores de la señal de entrada con una ventana móvil y calcular los datos resultantes. Dos de los métodos más comunes, son el uso funciones de cálculo del valor mayor dentro de la ventana, y la función de cálculo del promedio.

Capa completamente conectada

Una vez colocadas las capas de convolución y agrupación, se colocan capas que convierten los valores de salida de estas en un vector. Estas capas se utilizan para permitir a la red aprender una función no lineal que represente una combinación de las características obtenidas de las capas anteriores.

Para esto es necesario convertir el mapa 3D creado por las capas convolucionales y de agrupación, en un vector unidimensional.

Arquitectura

La elección de combinaciones de estas capas para formar la arquitectura final de la red depende de la naturaleza del problema y de los resultados obtenidos en otras arquitecturas ya probadas para problemas similares, puesto que no existe un método para determinar la arquitectura exacta para la resolución correcta de un problema. Una aplicación común es encadenar repetidamente capas convolucionales seguidas por capas de agrupamiento y finalmente colocar una capa completamente conectada.

El diseño original de LeCun utilizado para el reconocimiento de dígitos, llamado LeNet, está compuesto por un encadenamiento de dos capas convolucionales y de agrupamiento con dos capas completamente conectadas. Debido a la aparición del GPU, arquitecturas más complejas se han vuelto computacionalmente practicables.

Redes Neuronales recurrentes (RNN)

Las redes neuronales recurrentes, a diferencia de las redes FF, permiten a las neuronas enviar señales hacia capas anteriores, neuronas de la misma capa o incluso a ellas mismas. Esto genera en la red un estado interno que causa un comportamiento temporal dinámico que las convierten en herramientas para inferir datos secuenciales como es el caso de las series temporales.

Hopfield Network

En 1982 Hopfield propuso un modelo de red neuronal completamente conectada donde las neuronas cumplen simultáneamente la función de capa de entrada, capa oculta y capa de salida. Las conexiones entre neuronas son bidireccionales, lo que significa que los pesos son simétricos.

Para lograr el aprendizaje se presentan a la red los patrones de tipo binario. Cada neurona, entonces, ejecuta su función de activación y envía la señal al resto de las neuronas, y a su vez recibe señales de todas ellas. El resultado obtenido es la salida de la red que puede compararse con el patrón de entrada para detectar el error y hacer los ajustes de pesos necesarios. Este proceso se repite sucesivamente provocando que la salida obtenida reduzca su variación con respecto a los patrones de entrada hasta llegar a un punto de estabilidad.

Una de las ventajas de este tipo de redes es que una vez entrenadas son muy tolerantes al ruido, lo que significa que son capaces de reconocer un patrón, incluso si se le presenta de manera parcial (Szandała, 2015).

Regla de aprendizaje de Hebb

En este algoritmo de aprendizaje, los pesos entre las conexiones de las neuronas son incrementados cuanto más señales positivas sean enviadas entre ellas. Utilizando la función de activación de signo, se calculan los valores de salida de las neuronas y se suma su multiplicación para obtener el nuevo valor de peso (Szandała, 2015), mediante la siguiente ecuación:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^{m} x_i^k x_j^k$$

Donde:

$n = número\ de\ neuronas$

$m = número\ de\ patrones$

Regla de aprendizaje de Oja

Este método tiene como objetivo mantener normalizados los pesos de la red a fin de evitar problemas de estabilidad. Para hacerlo utiliza un valor llamado *factor V* (Szandała, 2015):

$$V_{ij} = \sum_{k=1}^{m} w_{ij} x_j^k$$

Este factor es utilizado junto con la tasa de entrenamiento para calcular la modificación de peso de cada sinapsis:

$$w_{ij}^{k+1} = w_{ij}^k + \mu V \left(x_j^{k+1} - V w_{ij}^k \right)$$

Referencias

[0] Gurney, K. Introduction to Neural Networks. Taylor & Francis.

[1] Kriesel, D. (2007). A Brief Introduction to Neural Networks. Recuperado de http://www.dkriesel.com

[2] Nielsen, M. (n.d.). Neural Networks and Deep Learning. 224.

[3] Russell, S., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Prentice Hall.

[4] Szandała, T. (2015). Comparison of Different Learning Algorithms for Pattern Recognition with Hopfield's Neural Network. Procedia Computer Science, 71, 68–75. https://doi.org/10.1016/j.procs.2015.12.205