1. Redes Neuronales

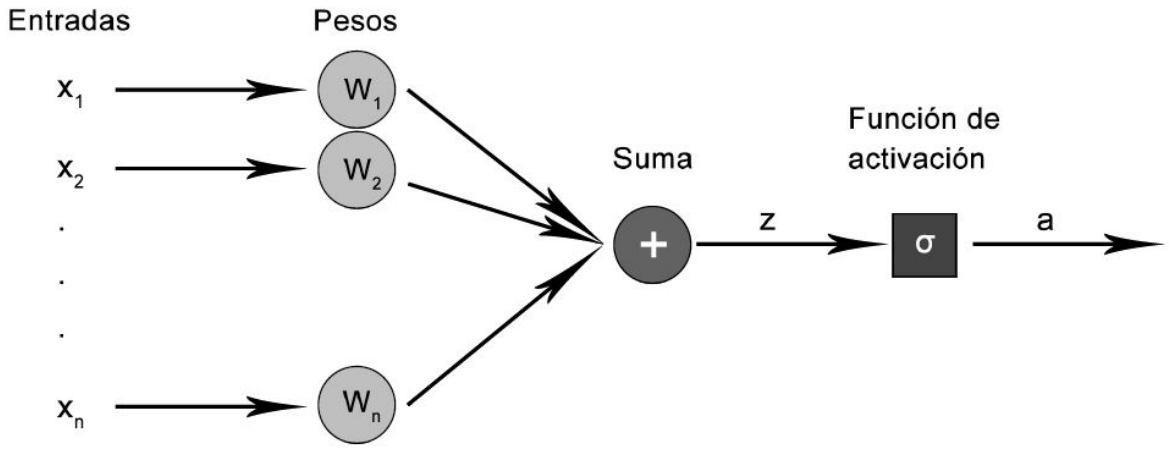
En este apartado se incluyen los conocimientos necesarios para entender el funcionamiento de las redes neuronales y su uso para problemas de segmentación.

1.1 Introducción a las redes neuronales

Una red neuronal es un sistema de procesamiento de la información compuesto por nodos o neuronas interconectadas. Estas neuronas asignan un valor numérico o peso a sus entradas y producen una salida que permite resolver problemas de regresión (predecir una cantidad) y clasificación (predecir una clase).

1.1.1 Funcionamiento de una neurona

Cuando las entradas a una neurona superan un umbral, esta se activa propagando su salida. Las conecciones de entrada que son fuertes aportan más que las débiles a esta activación. A continuación se describe el modelo matemático más simple que implementa el funcionamiento de una neurona biológica.



Modelo de una neurona.

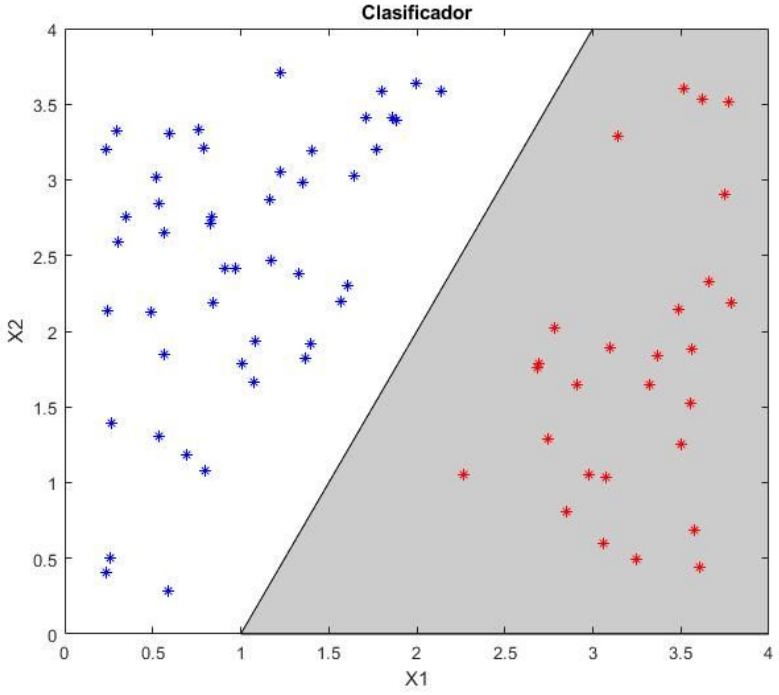
Siendo x_i la entrada y los parámetros ajustables w_i y b, tenemos que la función para la regresión lineal es:

$$z = \sum_i (w_i \times x_i) + b$$

Si utilizamos una función para la activación de la neurona que aplique un umbral sobre z, $a = \sigma(z)$, obtenemos la ecuación de un clasificador lineal:

$$a = \sigma(\sum_i (w_i \times x_i) + b)$$

En el caso de que la función de activación sea $\sigma(z) = 1$ para $z > 0$ y $\sigma(z) = 0$ para $z \leq 0$ la neurona puede resolver problemas de clasificación binaria. Utilizando dos entradas (x1,x2), dos tipos de datos (Rojo, Azul) y el siguiente hiperplano:

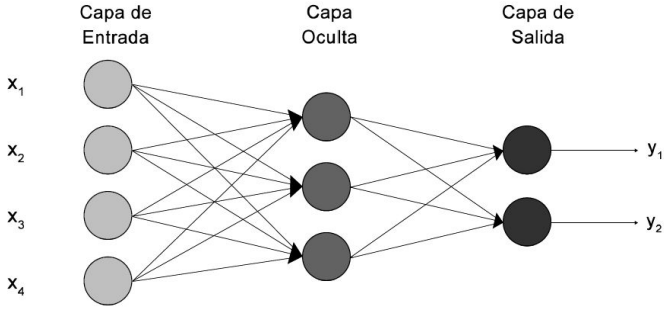


Siendo m la pendiente y c el punto de corte con el eje x2, tenemos que el espacio de activación (sombreado) viene viene descrito por:

$$x_2 < m \cdot x_1 + c \Rightarrow m \cdot x_1 - x_2 + c > 0$$

Al compararlo con la ecuación de la neurona para la clasificación binaria
$w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0$ obtenemos los valores $w_1 = m$, $w_2 = -1$ y $b = c$.
De esta forma la neurona se activará siempre que la entrada se encuentre en la región descrita por sus parámetros internos.

La colocación de neuronas en forma de capas se denomina arquitectura multicapa y permite producir regiones de decisión de diferente complejidad y forma, siempre y cuando no se utilicen funciones de activación lineales. Al número de capas se le conoce por profundidad.



Arquitectura multicapa con dos salidas.

Siendo la salida de la neurona j en la capa L igual a:

$$a_j^L = \sigma_j^L(z_j^L)\ ,\ z_j^L = \sum_i (w_{i\to j}^L \times a_i^{L-1}) + b_j^L\ ,\ y_i^0 = x_i$$

Se puede convertir en operaciones con matrices:

$$A^L = \sigma^L(Z^L)\ ,\ Z^L = W^L \times A^{L-1} + B^L\ ,\ Y^0 = X$$

Cuando usamos la función de activación identidad ($\sigma(z) = z$) en arquitecturas multicapa, al igual que el resto de funciones de activación lineales, se produce una salida lineal. Esto quiere decir que equivale a implementar una monocapa como se muestra en la siguiente transformación:

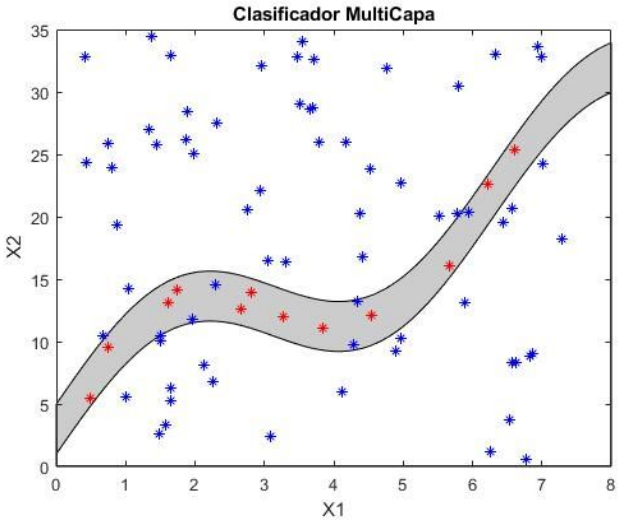
Utilizando una red de dos capas con una neurona en cada capa y una sola entrada:

$$y = w_2 \cdot (w_1 \cdot x_1 + b_1) + b_2$$

$$w_{monocapa} = w_1 \cdot w_2 \text{ y } b_{monocapa} = w_2 \cdot b_1 + b_2$$

Se comprueba que el resultado sigue siendo una función lineal, pudiendo reducirse a una sola neurona y perdiendo la capacidad de solucionar problemas complejos.

La introducción de funciones no lineales permiten a la redes neuronales, con los suficientes parámetros y tiempo, modelar cualquier tipo de función en arquitecturas multicapa. Es común utilizar un tipo de activación para construir la red y otro diferente justo antes de la salida para cambiar el tipo de resultado buscado (clasificación binaria, regresión…).



Ejemplo de espacio de decisión complejo

Un problema complejo es capaz de necesitar miles de neuronas y el ajuste de sus parámetros se vuelve una práctica difícil de hacer manualmente. Según el tipo de problema y la información conocida previamente, existen diferentes técnicas de aprendizaje que se pueden agrupar en 3 tipos.

Aprendizaje por refuerzo
Los datos de entrada no son conocidos directamente. Existe un agente (la red neuronal) que interacciona con un entorno o medio realizando acciones y cambiando de estado. Las acciones en cada paso que ayuden a llegar al estado deseado son reforzadas de manera que, cuando el agente se encuentre en la misma situación, seleccione la acción con mayor recompensa.

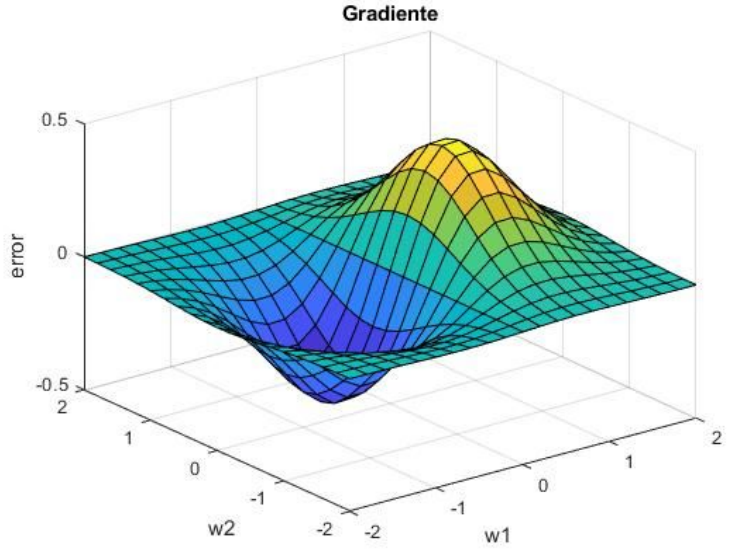
Aprendizaje no supervisado
Los datos de entrada son conocidos y utilizamos una función de coste dependiente de los valores de entrada y la salida de la red que tratamos de minimizar. La selección de esta función depende del problema a solucionar.

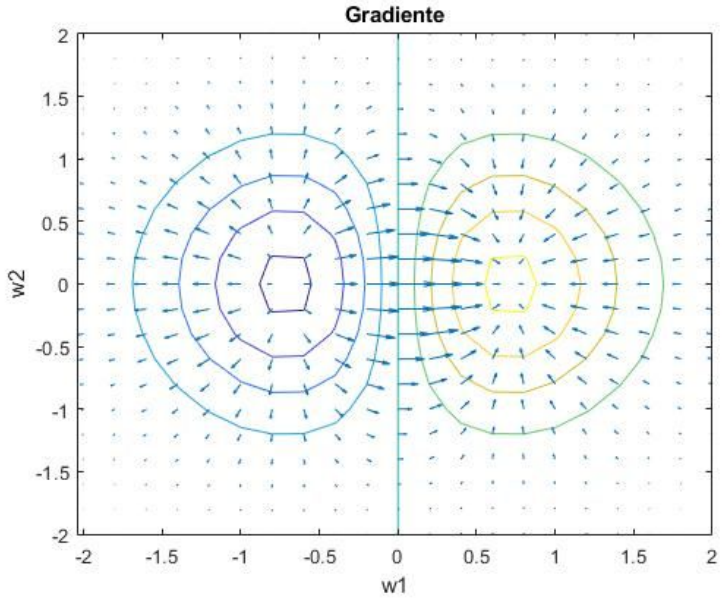
Aprendizaje supervisado
En el aprendizaje supervisado se conocen los pares de datos de entrada y la salida. Se utiliza una función de error, como puede ser el error cuadrático medio, entre la salida de la red y la salida deseada para ajustar los parámetros de la red.

1.1.2 El Gradiente y la propagación hacia atrás

El gradiente es un vector de n-dimensiones que proporciona la dirección de crecimiento y la variación de una función con respecto a los cambios de sus variables independientes. En nuestro caso el gradiente del error tendrá tantas dimensiones como variables entrenables y, al conocer hacia dónde decrece la función de error, podremos encontrar sus valores óptimos.

$$\nabla E = \left(\frac{\partial E}{\partial w^1}, \frac{\partial E}{\partial b^1}, \ldots, \frac{\partial E}{\partial w^L}, \frac{\partial E}{\partial b^L}\right)$$





Recordar que cuando tenemos una función dentro de otra, $f(g(x))$, su derivada es la variación de f con respecto a g por la variación de g respecto a la variable independiente x. Esto se puede aplicar tantas veces como funciones intermedias existan por lo que es conocida como la regla de la cadena.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\ \frac{\partial g}{\partial c}\ \frac{\partial c}{\partial y} \cdots \frac{\partial b}{\partial x}$$

En el caso de una arquitectura con una neurona por capa definimos su función de error como:

$E(w^1, b^1, \ldots, w^L, b^L) = (y_{predicción} - y_{esperada})^2 = (a^L - y_{esperada})^2$

$a^L = \sigma(z^L)$ y $z^L = w^L\, a^{L-1} + b^L$

El cálculo de las componentes del gradiente en cada capa viene dado por:

Capa de Salida L:

$$\frac{\partial E}{\partial w^L} = \frac{\partial E}{\partial a^L}\frac{\partial a^L}{\partial w^L} \Rightarrow \frac{\partial a^L}{\partial w^L} = \frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial E}{\partial w^L} = \frac{\partial E}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial w^L} \Rightarrow \frac{\partial z^L}{\partial w^L} = a^{L-1}\,,\ \frac{\partial E}{\partial a^L} = 2(a^L - y)\,,\ \frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

En el caso del bias hay que sustituir $\frac{\partial z^L}{\partial w^L} \Rightarrow \frac{\partial z^L}{\partial b^L} = 1$

Capa L-1:

$$\frac{\partial E}{\partial w^{L-1}} = \frac{\partial E}{\partial a^L}\frac{\partial a^L}{\partial w^{L-1}} \Rightarrow \frac{\partial a^L}{\partial w^{L-1}} = \frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial w^{L-1}} \Rightarrow \frac{\partial z^L}{\partial w^{L-1}} = \frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial w^{L-1}} \Rightarrow \frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial E}{\partial w^{L-1}} = \frac{\partial E}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial w^{L-1}} \Rightarrow \frac{\partial z^{L-1}}{\partial w^{L-1}} = a^{L-2}\,,\ \frac{\partial z^L}{\partial a^{L-1}} = w^L\,,\ \frac{\partial a^{L-1}}{\partial z^{L-1}} = \sigma'(z^{L-1})$$

Capa L-2:

$$\frac{\partial E}{\partial w^{L-2}} = \frac{\partial E}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}\frac{\partial a^{L-1}}{\partial z^{L-1}}\frac{\partial z^{L-1}}{\partial a^{L-2}}\frac{\partial a^{L-2}}{\partial z^{L-2}}\frac{\partial z^{L-2}}{\partial w^{L-2}} \Rightarrow$$

$$\Rightarrow \frac{\partial z^{L-2}}{\partial w^{L-2}} = a^{L-3}\,,\ \frac{\partial z^{L-1}}{\partial a^{L-2}} = w^{L-1}\,,\ \frac{\partial a^{L-2}}{\partial z^{L-2}} = \sigma'(z^{L-2})$$

Capa X:

$$\frac{\partial E}{\partial w^X} = 2(a^L - y) \times a^{X-1} \times \prod_{i=x}^{i=L}(\sigma'(z^i)) \times \prod_{i=x+1}^{i=L} w^i$$

Como se observa en las ecuaciones, todas las operaciones deben ser diferenciables y, para saber cuánto aportan los parámetros de una capa al error total, es necesario conocer el resultado de su variación por las capas siguientes (elementos coloreados). Para ello comenzamos el cálculo desde la salida dándosele el nombre de propagación hacia atrás del gradiente.

Existen dos problemas con el cálculo del gradiente a medida que aumenta la profundidad de la red. Supongamos el siguiente caso.

Uno de los términos de los que depende $\frac{\partial E}{\partial w^X}$ son los pesos de las capas consecuentes $\prod_{i=x+1}^{i=L} w^i$.

Utilizando el mismo peso en cada capa, cuando |w|>1 obtenemos un aumento o explosión del gradiente pero si 0<|w|<1 tendemos a 0, lo que se conoce como desvanecimiento del gradiente. En caso de que sature, las capas más cercanas a la entrada de la red tendrán mayores modificaciones que las últimas. Si por el contrario ocurre el desvanecimiento, las modificaciones de los pesos son insignificantes haciendo que la red deje de aprender en las primeras capas y repercutiendo en el resto.

La modificación de los pesos tambien depende de la función de activación elegida $\prod_{i=x}^{i=L}(\sigma'(z))$ permitiendo mitigar o agravar los problemas de gradiente como veremos en el siguiente

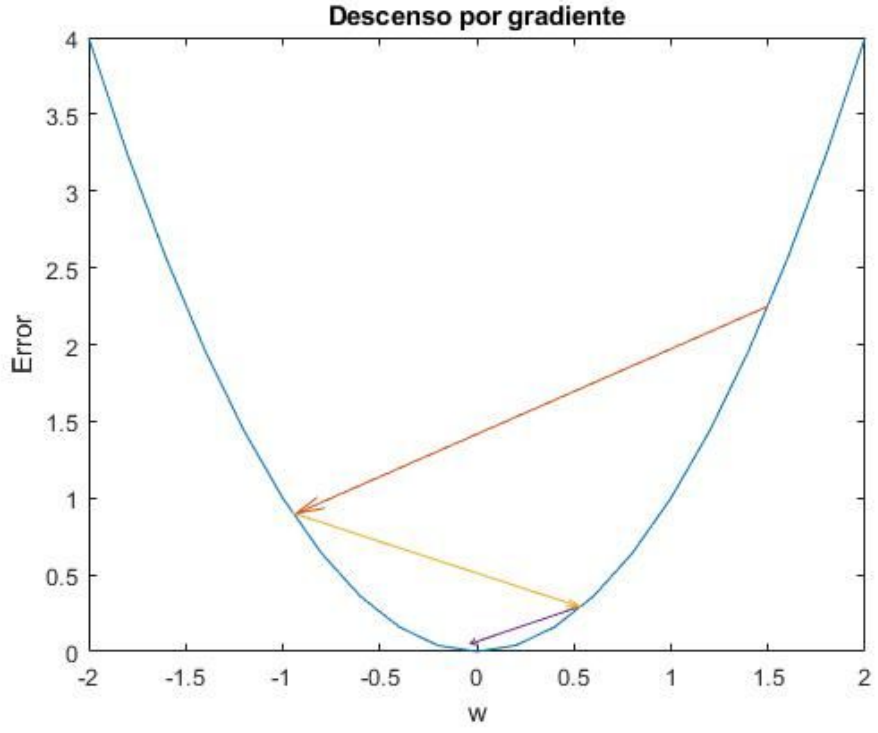
apartado. Otra opción bastante popular es añadir conexiones residuales que sumen la señal de entrada de una capa con su salida de forma que cada una afecta directamente al error de la red. $a_L = \sigma(x + a_1 + a_2 + \ldots + a_{L-1})$

Otras soluciones añaden un componente dentro del cálculo del error como las funciones L1 y L2 de regularización. Este tipo de funciones son dependientes de los pesos, penalizando los valores muy grandes o muy pequeños.

Conocidos todos los componentes del gradiente es hora de minimizar la función de error. Un método de optimización común se conoce como descenso por gradiente que mediante saltos, proporcionales a la aportación del error, en la dirección de decrecimiento, resulta en un proceso iterativo para la búsqueda de mínimos. La cantidad de salto se denomina ratio de aprendizaje y valores muy grandes pueden provocar que no llegue a converger en el mínimo (zig-zag) o incluso divergir mientras que valores muy pequeños hace que el aprendizaje sea más lento.

$$w = w_{anterior} - \alpha \times \frac{\partial E}{\partial w}$$

No suele ser necesario decrecer el ratio de aprendizaje con el número de iteraciones ya que los saltos son proporcionales al módulo del gradiente y tiende a 0 a medida que nos acercamos a un mínimo.



Ejemplo de descenso por gradiente sobre la función de error para una variable.

En caso de que el tipo de datos de entrada tenga grandes diferencias entre sí puede ocurrir que la red deshaga los cambios en los pesos al tratar de minimizar sus errores por separado. Por lo

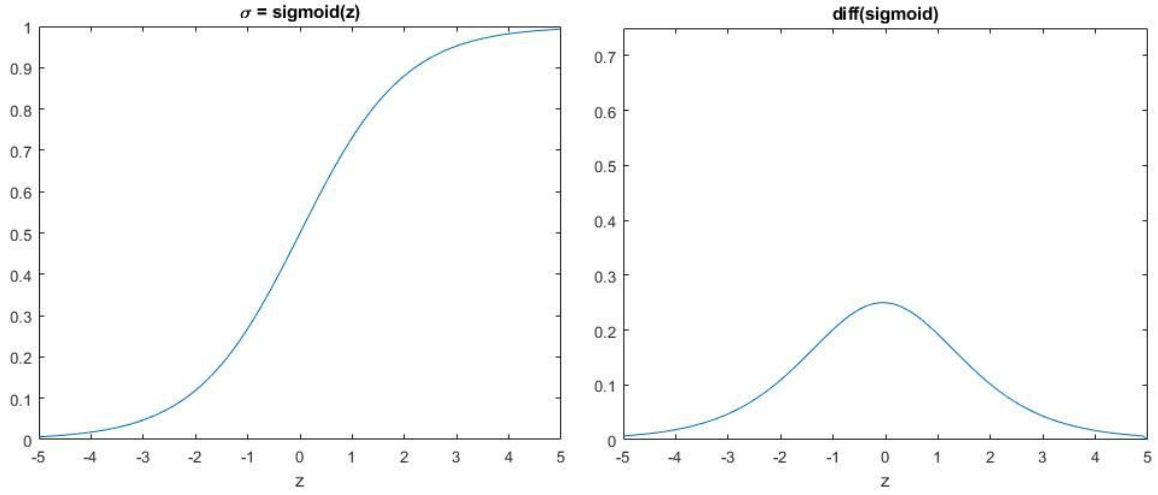
tanto, se recomiendo calcular la media del error sobre un conjunto o lote de datos que preferiblemente contenga un caso de cada tipo. En numerosas ocasiones no es posible separar en diferentes clases los datos de entrada, por lo que suele recomendarse su reordenación de manera aleatoria antes de extraer el mayor número posible para el lote.

1.1.3 Funciones de activación

Como vimos anteriormente las funciones de activación juegan un papel importante en el cálculo del gradiente. Es común encontrar en una arquitectura multicapa dos tipos de funciones de activación, el que utiliza las capas internas para aprender y el que utiliza la capa de salida para producir el resultado del problema tratado.

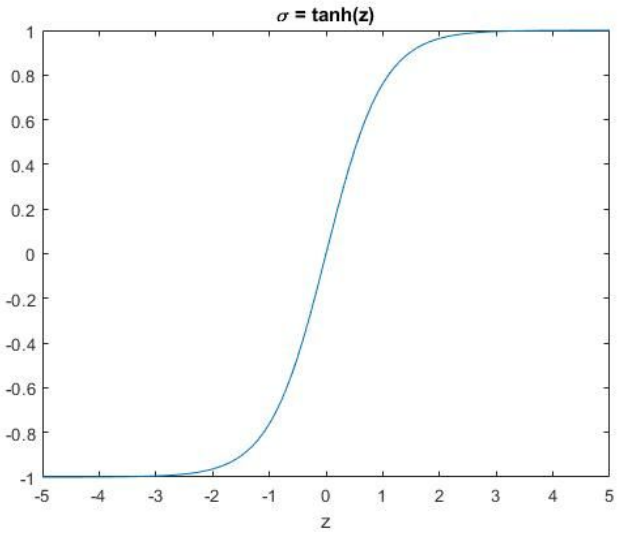
A continuación nombramos cuatro de las funciones más comunes:

Sigmoid. La función sigmoidal o logística se basa en la activación de una neurona biológica que devuelve valores comprendidos entre 0 y 1. Observando su derivada, comprobamos que agrava el problema de desvanecimiento del gradiente en los extremos, donde tiende a cero.



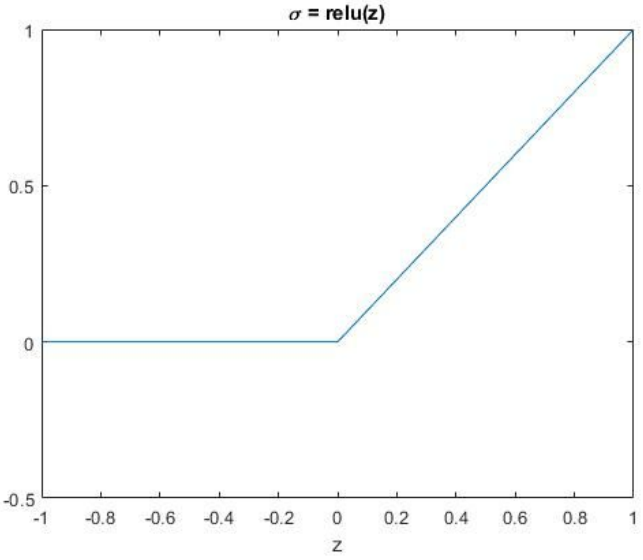
Su valor a la salida de la red se interpreta como la probabilidad de pertenecer a una clase. Cuando se tienen múltiples salidas no tienen por qué sumar uno entre sí (80% gato y 95% animal).

Tanh. Parecida a la sigmoidal pero entre -1 y 1. Tiene la ventaja de que al estar centrada en 0 tiene gradientes de mayor tamaño pero termina sufriendo los mismos problemas en sus extremos.



Softmax. Generalización de la función sigmoid para resolver problemas de clasificación multiclase. Produce una probabilidad entre todas las salidas igual a 1(80% perro, 20% gato).

RELU. La Unidad Rectificadora Lineal es una función cuyo uso se vuelve casi obligatorio en redes multicapas dejando el uso de otras funciones únicamente para la salida de la red.



Como se observa, esta función es parecida a la identidad para valores positivos de entrada, pero añade no-linealidad al devolver cero para valores negativos de entrada. Su derivada sólo puede tomar los valores constantes 1 o 0 y por tanto esta función no sufre de desvanecimiento ni explosión de gradiente.
La activaciones a 0 mejoran la convergencia de la red debido a que fuerza a un conjunto más pequeño de neuronas a aprender de manera óptima, ya que en ellas recae la aportación del gradiente.

El reducido número de activaciones producido por las salidas a 0 introduce velocidad y un menor uso de memoria en la red. También mejoran la convergencia forzando a las neuronas activadas a aprender de manera óptima, ya que en ellas recae la aportación del gradiente. Por tanto RELU simplifica el modelo y lo vuelve más ligero en comparación con sigmoid o tanh, donde la cantidad de activaciones es más densa y el cálculo del gradiente se vuelve más complejo.

Existen otras funciones parecidas como LRELU y PRELU que tratan de evitar la activaciones a 0 en caso de que se vuelva un problema para el aprendizaje de la red (RELU moribundo), pero no suelen resultar en mejoras significativas de los resultados[3].

Otra función que está obteniendo éxito es la unidad exponencial lineal o ELU que ha mostrado en alguno problemas ser mejor que la unidad rectificadora[4]

6. Anexo

6.1 Bibliografía

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.: 2014.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting.

[2] Sergey Ioffe, Christian Szegedy.: 2015.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[3] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng.: 2013.: Rectifier Nonlinearities Improve Neural Network Acoustic Models.

[4] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter.: 2015.: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.: 2015.: Deep Residual Learning for Image Recognition.

[6] Sergey Ioffe.: 2017.: Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models.

[7] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller.: 2015.: Striving For Simplicity: The All Convolutional Net.

[8] Fisher. Fisher Yu, Vladlen Koltun.: 2016.: Multi-Scale Context Aggregation By Dilated Convolutions.

[9] Karen Simonyan, Andrew Zisserman.: 2014.: Very Deep Convolutional Networks for Large-Scale Image Recognition.

[10] Evan Shelhamer, Jonathan Long, and Trevor Darrell.: 2016.: Fully Convolutional Networks for Semantic Segmentation.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio.: 2014.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

[12] Md Atiqur Rahman and Yang Wang.: 2016.: Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation.

[13] Diederik P. Kingma, Jimmy Ba.: 2017.: Adam: A Method for Stochastic Optimization.

[14] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, Christoph Bregler.:2015.: Efficient Object Localization Using Convolutional Networks.

[15] Olaf Ronneberger, Philipp Fischer, Thomas Brox.: 2015.: U-Net: Convolutional Networks for Biomedical Image Segmentation.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.: 2015.: Road Extraction by Deep Residual U-Net.

[17] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam.: 2017.: Rethinking Atrous Convolution for Semantic Image Segmentation.

[18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille.: 2017.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.: 2015.: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition.

[20] Andrew Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, Wenzhe Shi.: 2017.: Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize.

[21] Eunbyung Park.: Groupout: A Way to Regularize Deep Convolutional Neural Network.

[22] Fausto Milletari, Nassir Navab, Seyed-Ahmad Ahmadi.: 2016.: V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.

[23]Md Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M. Taha, Vijayan K. Asari.: 2018.: Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation.

[24] Nicolas Ballas, Li Yao, Chris Pal, Aaron Courville.: 2017.: Delving Deeper into Convolutional Networks for Learning Video Representations.

[25] Kurugol S1, San Jose Estepar R, Ross J, Washko GR.: 2013.: Aorta segmentation with a 3D level set approach and quantification of aortic calcifications in non-contrast chest CT.

[26] Jan Egger, Bernd Freisleben, Randolph Setser, Rahul Renapuraar, Christina Biermann, Thomas O'Donnell.: 2009.: Aorta Segmentation for Stent Simulation.

[27] Zhiwei Fan, Chen Wang, Yuqing Zhu, Tianrun Li, Zhen Zhang.: Automate Aorta Segmentation in Thoracic Aortic Aneurysm CT Images Via Convolutional Neural Network.

[28] Karen López-Linares, Nerea Aranjuelo, Luis Kabongo, Gregory Maclair, Nerea Lete, Mario Ceresa, Ainhoa García-Familiar, Iván Macía, Miguel A. González Ballester.: 2018.: Fully automatic detection and segmentation of abdominal aortic thrombus in post-operative CTA images using deep convolutional neural networks.

6.2 Enlaces de interés

[a] MRICron - http://people.cas.sc.edu/rorden/mricron/index.html
[b] ImageJ - https://imagej.nih.gov/ij/
[c] Theano - http://deeplearning.net/software/theano/
[d] CNTK - https://www.microsoft.com/en-us/cognitive-toolkit/
[e] Tensorflow - https://www.tensorflow.org/
[f] Keras - https://keras.io/
[g] TFLearn - http://tflearn.org/
[h] FloydHub - https://www.floydhub.com/
[i] 2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5
[j] Nibabel - https://github.com/nipy/nibabel
[k] Centro de Tecnologías de la Imagen - http://www.ctim.es