Tecnológico de Monterrey

Inteligencia Artificial Avanzada para la Ciencia de Datos

Daniel Otero Fadul

Departamento de Matemáticas y Ciencia de Datos
Escuela de Ingeniería y Ciencias

Regresión Logística

Supongamos que tenemos una variable de Bernoulli Y la cual tiene una probabilidad p de ser igual a uno y una probabilidad $1 - p$ de ser igual a cero. Dicho esto, tenemos que

$$\begin{aligned} E(Y) &= p \\ \text{var}(Y) &= p(1-p). \end{aligned}$$

Si quisiéramos tener un modelo lineal de la probabilidad de que Y sea igual a uno tendríamos que

$$E(Y|x) = p = w_0 + w_1 x.$$

Sin embargo, este modelo lineal no es acotado y puede tomar cualquier valor, mientras que $0 \leq p \leq 1$.

Regresión Logística

Una mejor estrategia surge cuando se considera el "odds ratio":

$$\text{odds} = \frac{p}{1-p}.$$

Nótese que odds $\in [0, \infty)$. Este se puede interpretar como la proporción de la probabilidad de un uno sobre la probabilidad de un cero. Si tenemos que $p = 0.8$, odds $= 4$, es decir, una proporción cuatro a uno. Sin embargo, si calculamos los "odds" de $1 - p = 0.2$, obtenemos que odds $= 1/4$, así que no tenemos simetría. Para que los "odds" sean simétricos se utiliza el logaritmo natural: $\ln(1/4) = -\ln(4)$. Dado esto, se prefiere decir que el logaritmo natural de los "odds" es lineal respecto a la variable predictora x:

$$\text{logit}(p(x)) = \ln\left(\frac{p(x)}{1-p(x)}\right) = w_0 + w_1 x.$$

En general tendremos más de una variable predictora, por lo cual se tiene que

$$\text{logit}(p(x)) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_k x_k.$$

A esta expresión se le conoce como el **modelo logit**.

Regresión Logística

Es convencional pasar de la función *logit* a una una expresión que sea la probabilidad en función de las variables predictoras. Después de un poco de álgebra se puede llegar a que

$$p(x) = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_k x_k)}}.$$

Esta expresión se conoce como la **curva logística**.

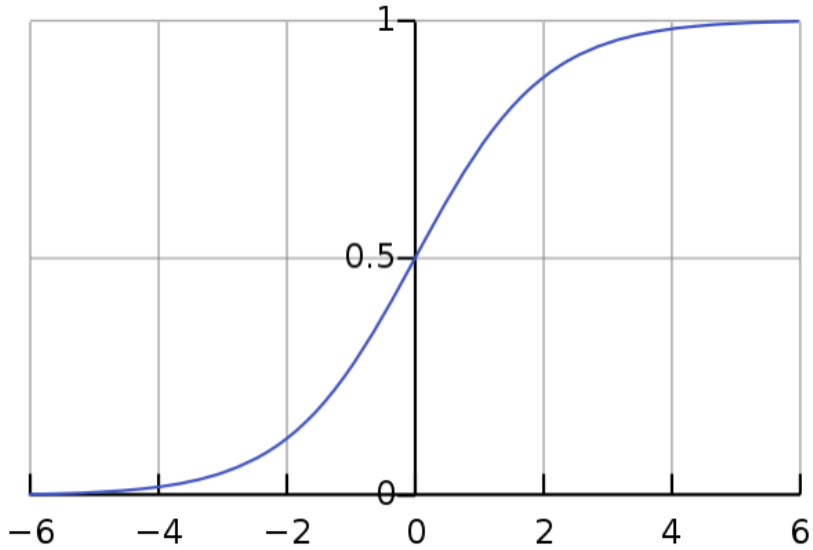

Figura: La gráfica de la curva logística como función de z: $p(z) = (1 + e^{-z})^{-1}$. Imagen tomada de `https://en.wikipedia.org/wiki/Logistic_regression`.

Regresión Logística

La regresión logística es muy útil para situaciones en que la variable dependiente solo toma dos valores. Supongamos que hemos tomado n observaciones de las variables predictoras y la variable dependiente y:

$$\begin{aligned} x_1 &= (x_{11}, x_{12}, \ldots, x_{1k}) \rightarrow y_1 \\ x_2 &= (x_{11}, x_{12}, \ldots, x_{2k}) \rightarrow y_2 \\ &\vdots \\ x_1 &= (x_{n1}, x_{n2}, \ldots, x_{nk}) \rightarrow y_n \end{aligned}$$

Además,

$$P(Y_i = y_i) = p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}, \; y_i = 0, 1.$$

Regresión Logística

Para hallar el vector de las estimaciones de los parámetros del modelo, $\hat{w} = [\hat{w}_0, \hat{w}_1, \ldots, \hat{w}_k]^T$, se maximiza la función de *máxima verosimilitud*:

$$\begin{aligned} L(w) &= \Pi_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i} \\ &= \Pi_{i=1}^{n} \frac{e^{y_i(w_0 + w_1 x_{i1} + \cdots + w_k x_{ik})}}{1 + e^{w_0 + w_1 x_{i1} + \cdots + w_k x_{ik}}}. \end{aligned}$$

Regresión Logística

Sea $h(z) = p(z)$. Entonces, maximizar la función de máxima verosimilitud es equivalente a minimizar la siguiente función de costo:

$$\begin{aligned} C(w) &= -\frac{1}{n}\log(L(w)) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(h(x_i)) + (1-y_i)\log(1-h(x_i)) \\ &= = -\frac{1}{n}(y^T \log(h(Xw)) + (1-y)^T \log(1-h(Xw))), \end{aligned}$$

donde $X \in \mathbb{R}^{n\times(k+1)}$, $w \in \mathbb{R}^{k+1}$ y $y \in \mathbb{R}^n$. Nótese que esta función de costo es convexa.

En este caso no es posible obtener una expresión para obtener los parámetros $\hat{w}$, por lo cual toca recurrir a métodos numéricos para encontrar sus valores, por ejemplo, el método de **gradiente descendente**.

Regresión Logística

Dada una condición inicial w_0, la iteración principal de la implementación más simple gradiente descendente es la siguiente:

$$w_{k+1} = w_k - \alpha \nabla C(w_k),$$

donde α se conoce como la **taza de aprendizaje**. En este caso, el gradiente de la función de costo está dado por

$$\nabla C(w) = \frac{1}{n} X^T (h(Xw) - y).$$

Regresión Logística

Lo anterior nos permite definir el siguiente algoritmo:

inicializar $w = w_0$, $\alpha \in (0, 1)$
repetir m veces
    calcular $\frac{1}{n} X^T (h(Xw) - y)$
    $w = w - \alpha \nabla C(w)$
retornar w

Redes Neuronales

Las **redes neuronales**, también conocidas como "Artificial Neural Networks" (ANN) en inglés, son sistemas computacionales que modelan una relación entre un conjunto de señales de entrada y un conjunto de señales de salida. Estas redes están conformadas por *neuronas* que se conectan a otras por medio de *sinapsis*.

Redes Neuronales

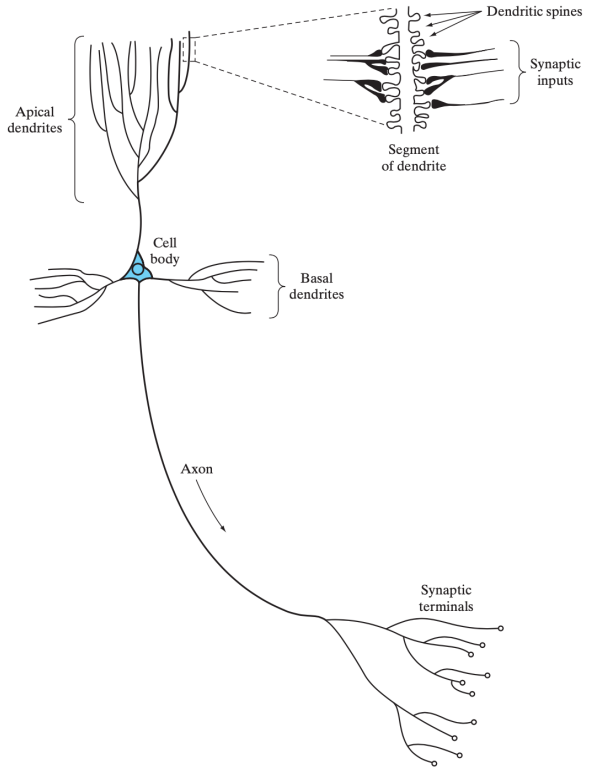

Figura: Como su nombre lo indica, las redes neuronales son sistemas que fueron desarrollados tomando como inspiración el cerebro biológico. La neurona biológica recibe señales eléctricas por medio de sus dendritas, las cuales reciben un distinto grado de importancia gracias a un proceso bioquímico. La acumulación de estas señales hace que se active una respuesta una vez se alcance un umbral, la cual se transmite a través del axón. Esta señal de salida se transmite a otras neuronas pues el axón de una neurona se conecta a las dendritas de neuronas vecinas. Imagen tomada de [2].

Redes Neuronales

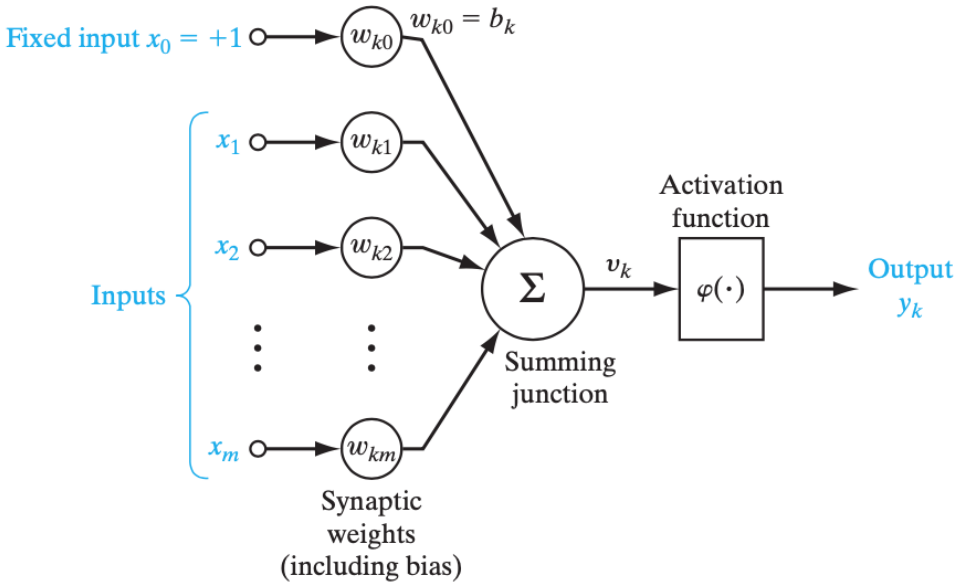

Figura: La neurona biológica se modela como una neurona artificial como la que se ve en la figura. Imagen tomada de [2].

Dado lo anterior, la expresión que relaciona las entradas y la salida de una neurona está dada por

$$\begin{aligned} y &= \phi\left(\sum_{i=0}^{n} w_i x_i\right) \\ &= \phi(w^T x), \end{aligned}$$

donde $w \in \mathbb{R}^{n+1}$ es un vector de pesos, $x_0 = 1$ y $\phi : \mathbb{R}^{n+1} \to \mathbb{R}$ es la **función de activación**.

Función de Activación

La función de activación es el mecanismo por medio del cual la neurona procesa la información de entrada y genera una señal de salida. Las funciones de activación más comunes son las siguientes:

- **Sigmoide:** $\sigma(z) = \frac{1}{1+e^{-z}}$.
- **Tangente hiperbólica:** $\phi(z) = 2\sigma(z) - 1$.
- **Unidad Lineal Rectificada (ReLU):** $\phi(z) = \max(0, z)$.
- **Identidad:** $\phi(z) = z$.

Redes Neuronales

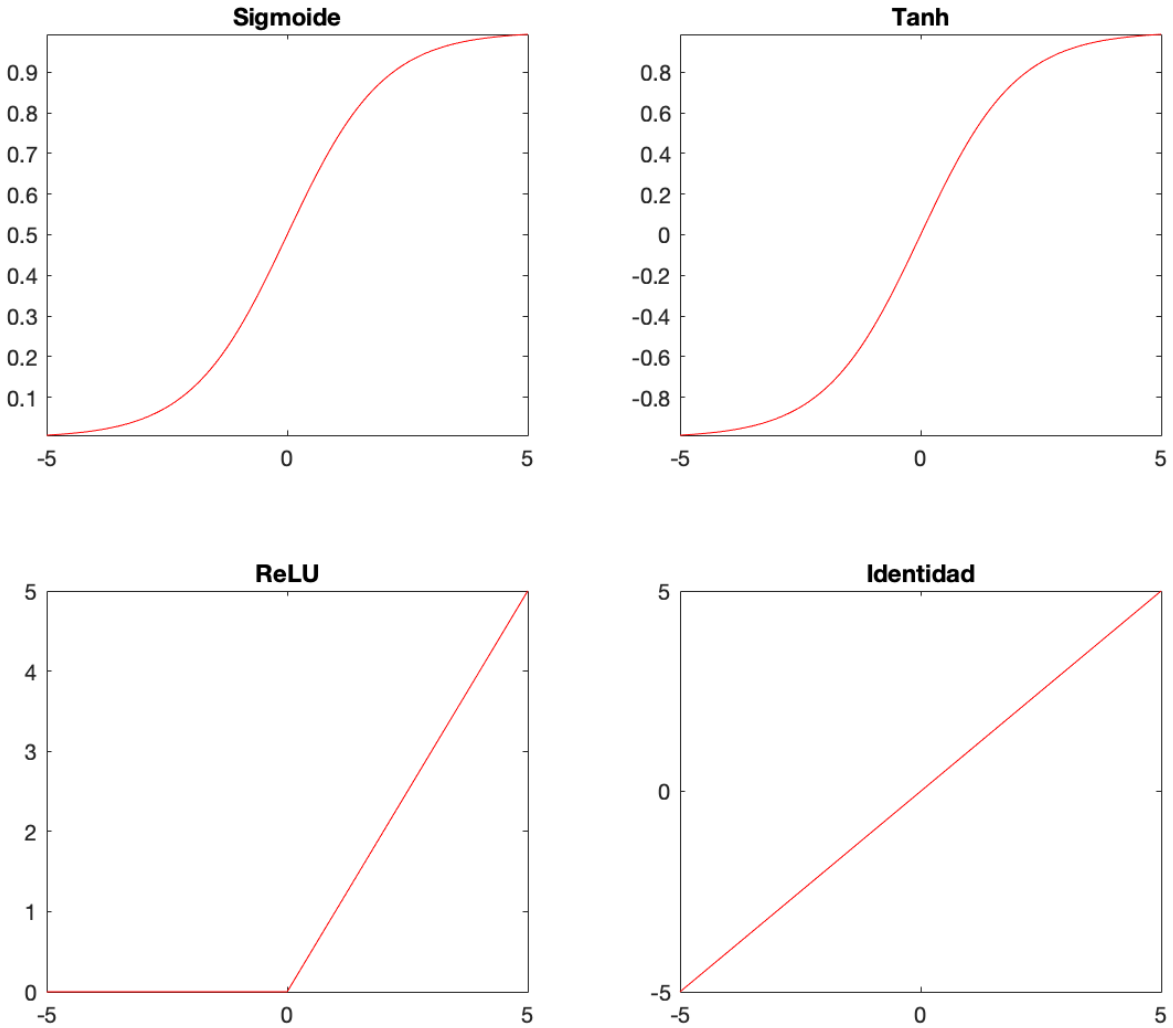

Figura: Algunas de las funciones de activación más usadas.

Redes Neuronales

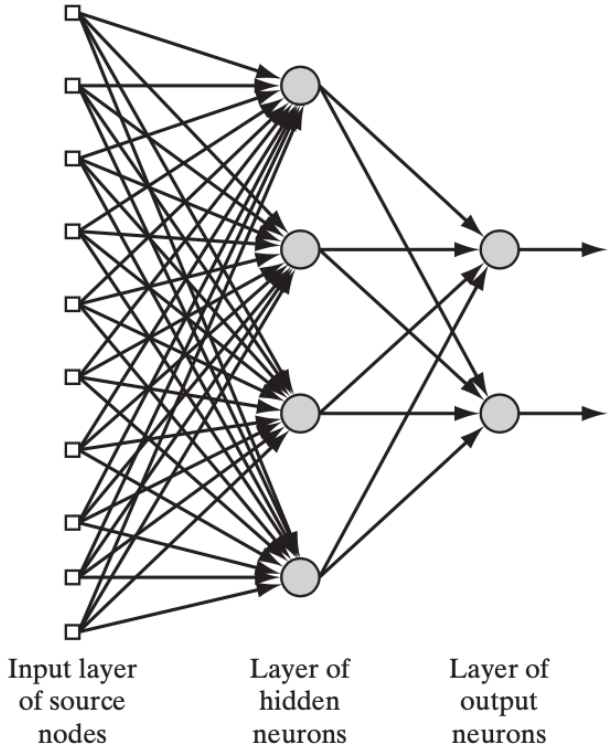

Figura: En la figura se puede ver un ejemplo de una red totalmente conectada que tiene una capa de entrada, una capa oculta y una capa de salida. Imagen tomada de [2].

Backpropagation

El entrenamiento de una red neuronal se realiza minimizando la siguiente función de costo:

$$C(W) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}(y_k^{(i)}\log(h_W(x^{(i)}))_k + (1-y_k^{(i)})\log(1-h_W(x^{(i)}))_k),$$

donde K es el número de neuronas en la capa de salida. Esta minimización es llevada a cabo utilizando el método de gradiente descendente junto con un algoritmo conocido como **"backpropagation"**. Nótese que esta función es una generalización de la función de costo de regresión logística.

Backpropagation

En otras palabras, entrenar una red neuronal consiste en obtener un conjunto de parámetros óptimos que minimizan la función de costo de la diapositiva anterior, lo cual se realiza numéricamente utilizando el método de gradiente descendente. Como sabemos, es necesario calcular este gradiente para utilizar este método, lo cual es posible gracias al algoritmo de "backpropagation". En otras palabras, "backpropagation" nos permite calcular estas derivadas parciales:

$$\frac{\partial C(W)}{\partial w_{ij}^{(l)}},$$

es decir, la derivada parcial de la función de costo respecto al parámetro $w_{ij}^{(l)}$: el peso asociado a la entrada **j** de la neurona **i** de la capa **l+1**. Los pesos $w_{ij}^{(l)}$ de la capa l se almacenan en una matriz que denotamos como $W^{(l)}$.

Backpropagation

Sea $a^{(l)}$, $l = 2, \ldots, L$, un vector que contienen las **activaciones** de las neuronas de la capa l, con $a^{(1)} = x$. Sea $\delta^{(L)} = a^{(L)} - y$ el error entre la salida esperada y y lo entregado por la última capa de la red neuronal $a^{(L)}$. Definimos $\delta^{(l)}$, $l = 2, \ldots, L-1$, como

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \circ a^{(l)} \circ (1 - a^{(l)}).$$

Nótese que $\circ$ es el producto matricial de Hadamard.

Backpropagation

Dado un conjunto de entrenamiento $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$, el algoritmo de "backpropagation" es el siguiente:

inicializar $\Delta^{(l)} = 0,\ l = 2, 3, \ldots, L-1,\ i = 1$
repetir n veces
  $a^{(1)} = x^{(i)}$
  calcular $a^{(l)},\ l = 2, 3, \ldots, L$
  $\delta^{(L)} = a^{(L)} - y^{(i)}$
  calcular $\delta^{(l)},\ l = L-1, L-2, \ldots, 2$
  calcular $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T,\ l = 1, 2, \ldots, L-1$
  $i = i + 1$
retornar $D^{(l)} = \frac{1}{n}\Delta^{(l)},\ l = 1, 2, \ldots, L-1$

En este caso,

$$\frac{\partial C(W)}{\partial w_{ij}^{(l)}} = D_{ij}^{(l)}.$$

Backpropagation

Teniendo en cuenta lo anterior, el entrenamiento de una red neuronal totalmente conectada se puede llevar a cabo ejecutando el siguiente algoritmo:

inicializar $W = W_0$, $\alpha \in (0, 1)$
repetir
    calcular $\nabla C(W)$ utilizando **backpropagation**
    $W = W - \alpha \nabla C(W)$
hasta cumplir con criterio de parada
retornar W

Backpropagation

La elección de la taza de aprendizaje α es un factor importante del entrenamiento: un α muy pequeño volvería el entrenamiento muy lento, mientras que un α muy grande podría hacer que el método de gradiente descendente no coverja. En cualquier caso, el conjunto de parámetros obtenido durante la fase de aprendizaje no sería útil. Dicho esto, es conveniente escoger una taza de aprendizaje adecuada.

Backpropagation

Una alternativa para el ajuste de la taza de aprendizaje es implementar un algoritmo que permita que esta taza se adapte de manera adecuada en cada iteración del método de gradiente descendente:

inicializar $W = W_0$, $\alpha = \alpha_0 \in (0, 1)$
repetir
    calcular $\nabla C(W)$ utilizando **backpropagation**
    $W = W - \alpha \nabla C(W)$
    calcular α
hasta cumplir con criterio de parada
retornar W

Backpropagation

"Vanishing Gradient"

Este problema se presenta cuando se entrenan redes neuronales artificiales con métodos de aprendizaje basados en el gradiente y "backpropagation". En estos métodos, durante cada iteración del entrenamiento, cada uno de los pesos de la red neuronal se ajusta de manera proporcional a la derivada parcial de la función de error con respecto al peso actual. El problema es que, en algunos casos, el gradiente será "desvanecido, lo que impedirá que el peso cambie de valor. Es más, esto podría impedir que la red neuronal siguiera aprendiendo.

Las siguientes opciones pueden ayudar a mitigar el "desvanecimiento del gradiente":

- Cambiar las funciones de activación (ReLU).
- "Batch normalization".
- Cambio en la arquitectura de la red: "Residual networks".

"Exploding Gradient"

Los "gradientes explosivos" son un problema en el que se acumulan grandes errores y dan lugar a actualizaciones muy grandes de los pesos del modelo de la red neuronal durante el entrenamiento. Esto hace que el modelo sea inestable y no pueda aprender de los datos de entrenamiento.

Las siguientes alternativas pueden ayudar a resolver este problema:

- "Gradient Clipping".
- Regularización.
- Reducir el tamaño del "lote" ("batch") durante el entrenamiento.

BIBLIOGRAFÍA

1 Dangeti, P. *"Statistics for machine learning,"* Packt Publishing Ltd., 2017.

2 Haykin, S., *"Neural networks and learning machines"*, Pearson Education India, 2010.

3 Stuart R., Norvig P, *"Artificial intelligence: a modern approach,"* 2002.

4 J. Berner et al., *"The modern mathematics of deep learning,"* arXiv preprint arXiv:2105.04026, 2021.

5 Y. Wang et al., *"Multiclassification of endoscopic colonoscopy images based on deep transfer learning"*, Computational and Mathematical Methods in Medicine, 2021.

6 Andrey Kurenkov, *"A Brief History of Neural Nets and Deep Learning,"* Skynet Today, 2020.