ENFIN761

Business Intelligence para las Finanzas

Ayudantía 7

Profesor: *David Díaz S.*
Ayudantes: *Gabriel Cabrera G.*[1]
25 octubre 2019

Construyendo un bloque: Neurona

1. Construya una neurona con dos *input-neurons* que usan una función de activación logística o sigmoide que posee los siguientes parámetros:

$$w = [0, 1]$$
$$b = 4$$

   Donde $w = [0, 1]$ es la forma vectorial de $w_1 = 0$, $w_2 = 1$ de los *weights*. Agregue como *inputs* $x = [2, 3]$.

Construyendo una Red Neuronal

1. Construya una red neuronal que presente:
   a. 2 *inputs* ($x_1 = 2$ y $x_2 = 3$) en la *Input Layer*.
   b. 1 *Hidden Layer* con dos neuronas (h_1 y h_2), construida en la sección anterior.
   c. 1 neurona (O_1) en la *Output Layer*.

Entrenando una Red Neuronal

1. Cargue la base de datos iris desde el módulo `datasets` de la librería `sklearn` y seleccione:
   a. Como variable a predecir (*target*) aquellas con valor igual a 0 (setosa) y 1 (versicolor).
   b. Como features *sepal length* (cm) y *sepal width* (cm).
   c. Separe la muestra en 80% *Training* y 20% *Testing* de manera aleatoria. Recuerde agregar una semilla igual a 42.
2. Entrene la Red Neuronal construida en la sección anterior con la muestra de Training.
3. Gráfique la función de costo (perdida) por cada *epochs* ¿se logra la convergencia?.

[1] :gcabrerag@fen.uchile.cl

4. Seleccione de manera aleatoria una observación de la muestra de *Testing* y verifique si la red neuronal predice la clase.

Redes Neuronales con Sckit-Learn: Clasificación

1. Importe la base de datos `prediccion_quiebra.xlsx` y elimine los *missing values*.
2. Realice un histograma para la variable `Q1_NQ_0` (*target*).
3. Separe la muestra en 70% *Training* y 30% *Testing* de manera aleatoria. Recuerde agregar una semilla igual a 42.
4. Realice el preproceso necesario a la muestra de *Training* y *Testing*. Luego balance a la clase menos común.
5. Entrene una red neuronal con arquitectura MLP y que tenga una función de activación del tipo ReLu. Muestre las clases predichas y las probabilidades asociadas.
6. Muestre la matriz de confusión y el *Classification Report*.

Redes Neuronales con Sckit-Learn: Regression

El archivo `data_usd_clp.xlsx` contiene datos de divisas extranjeras, entre ellas, el dolar. Utilizando las variables de la base de datos, realice la predicción de la variación del precio del dolar utilizando una red neuronal. Considere los siguientes pasos:

1. Importe la base de datos a su espacio de trabajo (*workspace*).
2. Construya hasta el tercer rezago (*lag*) de cada variable más la variación del precio del dolar. De ser necesario, programe una función que facilite la implementación de rezagos.
3. Utilice el año `2015-06-19` como corte para generar su muestra de training y de testing. Luego realice las transformaciones pertinentes.
4. Entrene un modelo utilizando una red neuronal con arquitectura MLP. Debe tener 3 *hidden layer* con 50 neuronas c/u.
5. Use las métricas MAE (*mean absolute error*) y RMSE (*root mean square error*) para medir el accuracy de su modelo.

Apéndice

Una neurona (*neuron*) se compone a partir de *inputs* (x), a la cuál se le agrega pesos (w) más un sesgo (b o *bias*) que se insertar en una función de activación (f) para obtener un *output* (y). El proceso que siguen, visualmente es:

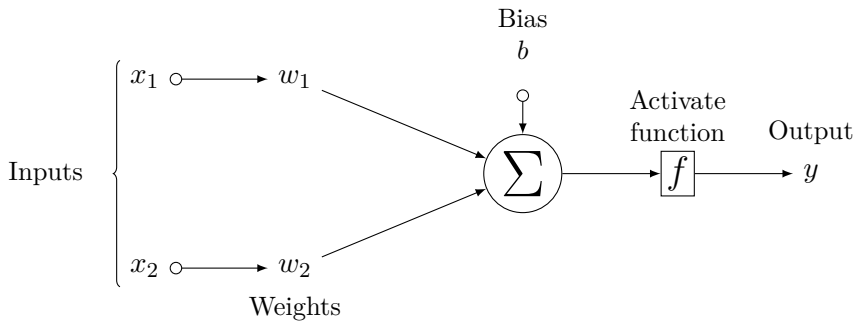

Figure 1: Esquema de una neurona

Considerando el ejercicio de la sección **Construyendo un bloque: Neurona**, su desarrollo viene dado por:

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

Donde w son los pesos o *weights* y b el sesgo o *bias*, el resultado obtenido es 7, número que debe ser "entregado" a una función de activación, en este caso a la función logística o sigmoide:

$$y = f(w \cdot x + b) = f(7) = \frac{1}{1 + exp(-7)} = 0.999$$

Si se quiere construir una red neuronal a partir de lo solicitado en **Construyendo una Red Neuronal**, la presentación visual sería:

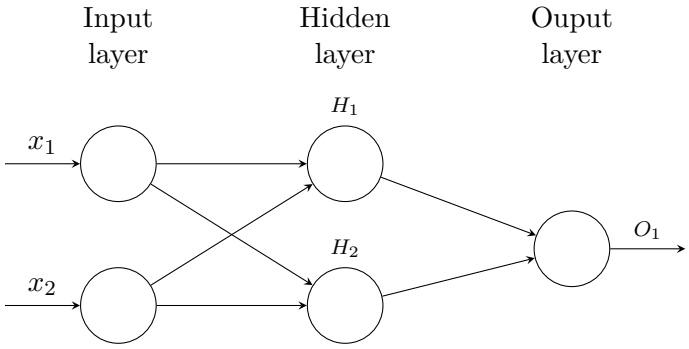

Figure 2: Esquema de la red neuronal

Donde h_1 como h_2 está compuesta por la neurona anteriormente definida y brevemente explicada. Teniendo en consideración lo anterior, se agrega a cada neurona ($h_1 = h_2$) los "ingredientes" necesario:

$$\begin{aligned} h_1 = h_2 &= f(w \cdot x + b) \\ &= f((0 * 2) + (1 * 3) + 0) \\ &= f(3) \\ &= \frac{1}{1 + exp(-3)} \\ &= 0.9526 \end{aligned}$$

Una vez que las *hidden layer* realizan su parte en la red su *output* (0.9526) que no es el final, se lo "transfiere" a la capa de salida (*output layer*):

$$\begin{aligned} o_1 &= f(w \cdot [h_1, h_2] + b) \\ &= f((0 * h_1) + (1 * h_2) + 0) \\ &= f(0.9526) \\ &= \frac{1}{1 + exp(-0.9526)} \\ &= 0.7216 \end{aligned}$$

Obteniendo el resultado final de 0.7216.

El proceso de **entrenar una red neuronal** es bastante interesante, si bien matemáticamente puede ser largo, basta con realizar una parte ($\frac{\partial L}{\partial w_1}$) para entender su todo. Recordar que este tipo de arquitectura es la más sencilla (*MLP* o *Muli-layer Perceptron*) y tiene un comportamiento **feedforward**. Antes de comenzar, hay que recordar ciertos conceptos.

Cuando se habla de "entrenar" un modelo, es simplemente minimizar la función de perdida o costo (*loss function*). Una de ella puede ser el *Mean Square Error* (*MSE*) que se define como:

$$\begin{aligned} MSE(w) &= \frac{1}{m} \sum_{i=1}^{m} (y^{true} - y^{pred})^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} (y^{true} - \underbrace{w^T \mathbf{x}}_{o_1})^2 \end{aligned}$$

Por un tema[2] de complejidad computacional se debe obtener el gradiente de $MSE(w)$:

$$\nabla_w MSE(w) = \begin{pmatrix} \frac{\partial}{\partial w_1} MSE(w) \\ \frac{\partial}{\partial w_2} MSE(w) \\ \vdots \\ \frac{\partial}{\partial w_n} MSE(w) \end{pmatrix}$$

[2]El tema es más extenso que esto.

Donde cada dirección y el paso (*step*) del gradiente se regula a través de:

$$w^{next\ step} = w + \eta \cdot \nabla_w MSE(w) = w + \eta \cdot \begin{pmatrix} \frac{\partial}{\partial w_1} MSE(w) \\ \frac{\partial}{\partial w_2} MSE(w) \\ \vdots \\ \frac{\partial}{\partial w_n} MSE(w) \end{pmatrix}$$

El proceso completo es el algorítmo de **gradiente descendiente**[3]. Abusando de la notación matemática podemos resolver para un caso general, "desarmando" la red neuronal en sus componentes tenemos:

1. $o_1 = f(w_5 \cdot h_1 + w_6 \cdot h_2 + b_3) = y^{pred}$
2. $h_1 = f(w_1 \cdot x_1 + w_3 \cdot x_2 + b_1)$
3. $h_2 = f(w_2 \cdot x_1 + w_4 \cdot x_2 + b_2)$

Luego, se puede asumir que la función de perdida tiene la forma:

$$L(w_1, w_2, w_3, w_4, w_5, w_6, b_1, b_2, b_3)$$

Por lo tanto si se quiere $\frac{\partial L}{\partial w_1}$:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y^{pred}} * \frac{\partial y^{pred}}{\partial w_1}$$

Luego, y^{pred} se define como:

$$y^{pred} = o_1 = f(w_5 * h_1 + w_6 * h_2 + b_3)$$

Cuya derivada con respecto a w_1 sería:

$$\frac{\partial y^{pred}}{\partial w_1} = \underbrace{\frac{\partial y^{pred}}{\partial h_1}}_{w_5 * f'(w5 * h_1 + w_6 * h_2 + b_3)} * \underbrace{\frac{\partial h_1}{\partial w_1}}_{x_1 * f'(w_1 * x_1 + w_2 * h_2 + b_1)}$$

Por lo tanto, el problema a resolver para $\frac{\partial L}{\partial w_1}$ es:

$$\frac{\partial L}{\partial w_1} = \underbrace{\frac{\partial L}{\partial y^{pred}} * \frac{\partial y^{pred}}{\partial h_1} * \frac{\partial h_1}{\partial w_1}}_{\textbf{backpropagation}}$$

Finalmente recordar, que la derivada de la función logística o sigmoide, es:

$$f'(x) = \frac{exp(-x)}{(1 + exp(-x))^2} = f(x) * (1 - f(x))$$

[3]El algorítmo que se usa en una red neuronal es el de gradiente descendiente estocástico con *reverse-mode autodiff*.