La Red Neuronal

Notas 5^{ta} Semana Curso ISML

09/11/18

1 Softmax Regression

Sabemos que la regresión logística particiona el espacio en dos grupos mediante un separador lineal. La regresión softmax usa la función Softmax en lugar de la sigmoide, para realizar una clasificación multiclase, realizando n particiones lineales del espacio.

1.1 Softmax

Definimos la función softmax derivada directamente de una distribución multinomial como:

$$Softmax(Z) = \frac{e^{Z}}{\sum_{j=1}^{C} e^{Z_j}} \tag{1}$$

$$e^{z} = exp\begin{pmatrix} z_{0,0} & z_{0,1} & \cdots & z_{0,c} \\ z_{1,0} & \cdots & \cdots & \cdots \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ z_{m,0} & z_{m,1} & \cdots & z_{m,c} \end{pmatrix}$$

$$= \frac{}{\sum_{j=0}^{C} e^{Z_j} = \begin{pmatrix} \sum_{j=0}^{C} e^{Z_{0,j}} \\ \sum_{j=0}^{C} e^{Z_{1,j}} \\ \vdots \\ \sum_{j=0}^{C} e^{Z_{m,j}} \end{pmatrix}}$$

$$= \begin{pmatrix} \frac{z_{0,0}}{\sum_{j=0}^{C} e^{Z_{0,j}}} & \frac{z_{0,1}}{\sum_{j=0}^{C} e^{Z_{0,j}}} & \cdots & \frac{z_{0,c}}{\sum_{j=0}^{C} e^{Z_{0,j}}} \\ \frac{z_{1,0}}{\sum_{j=0}^{C} e^{Z_{1,j}}} & \cdots & \cdots & \cdots \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{z_{m,0}}{\sum_{j=0}^{C} e^{Z_{m,j}}} & \frac{z_{m,1}}{\sum_{j=0}^{C} e^{Z_{m,j}}} & \cdots & \frac{z_{m,c}}{\sum_{j=0}^{C} e^{Z_{m,j}}} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{z_{0}}{\sum_{j=0}^{C} e^{Z_{0,j}}} \\ \frac{z_{1}}{\sum_{j=0}^{C} e^{Z_{1,j}}} \\ \vdots \\ \frac{z_{m}}{\sum_{j=0}^{C} e^{Z_{m,j}}} \end{pmatrix}$$

Donde:

$$Z = \theta \odot X + b$$

también llamado Logit, con dimensiones (m, c), i representa el i-ésimo caso de ejemplo de los m totales, y c es el número de clases que contiene ese vector, compuesto por la combinación lineal de θ los parámetros, b el sesgo, y X nuestra variable que representa los datos de entrada.
(El símbolo "$\odot$" simboliza multiplicación matricial)

Una vez definida la softmax, la utilizaremos para aproximar nuestra predicción, por lo tanto denotaremos $Softmax(Z) = \hat{y}$. Se puede observar por la definición que dado un vector de valores sin restricciones, lo mapea a probabilidades excluyentes, de modo que $\sum_{j=0}^{C} \hat{y}_{i,j} = 1$, sin embargo al tener dentro exponenciales pueden presentarse errores de Overflow numérico, causando que los valores se vayan a infinito y para implementaciones computacionales termine devolviendo un vector de valores *nan* (not a number).
Es por esta razón que se propone una variación de la softmax tradicional, llamada "Stable Softmax".

1.2 Stable Softmax

Definiremos la nueva función de activación como:

$$\frac{e^{Z-max(Z)}}{\sum_{j=1}^{C} e^{[Z-max(Z)]_j}} \tag{2}$$

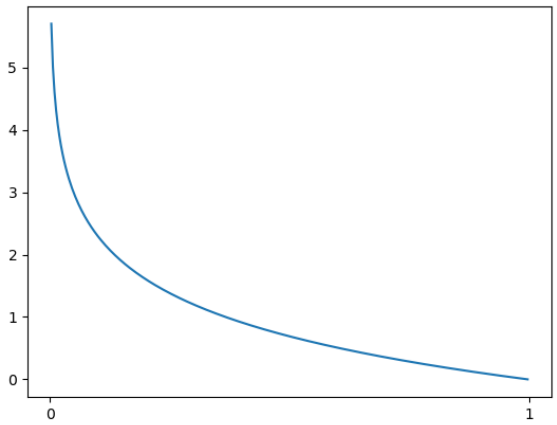
Observamos que al restarle el máximo de Z al mismo Z, convertimos todos los alores a $(-\infty, 0]$, para que de esta forma, si tenemos números enormes que desborden por el exponencial, al ser negativos, se convertirán en 0 al activar.

1.3 Negative Log Likehood

Sean $\hat{y} = p(\phi(x), \theta)$ las predicciones de un modelo de clasificación, queremos encontrar los mejores estimadores θ para que el error de predicción sea minimizado (Por la notación utilizada en redes neuronales, a partir de ahora llamaremos W de "Weights" al vector de parámetros θ).

Para esto utilizaremos la verosimilitud negativa logarítmica, con el fin de tener mayor estabilidad numérica y para aprovechar una propiedad muy especial del logaritmo para esta situación.

Si Observamos la gráfica de $-log_e(x)$ en el rango x = $(0, 1]$, podemos notar que cuando el logaritmo vale 0 cuando $x = 1$ e ∞ cuando $x = 0$.



Por lo tanto se entiende que cuando nuestra predicción que debería ser $y = 1$ da $\hat{y} = 0$ o un valor cercano, el costo o penalización es alto, por lo tanto la derivada con respecto al costo será igual de grande, y el modelo aprenderá a no cometer ese error, por otra parte si $\hat{y} = 1$, Entonces el error será cercano a cero, por lo que no habrá penalización.

El ajuste de los parámetros se realiza por un algoritmo de optimización no lineal de primer orden llamado Gradient Descent, es cierto que gracias a la convexidad de la función softmax (demostrada en las anotaciones previas sobre regresión logística) podemos utilizar métodos de optimización de orden superior, sin embargo es mejor acostumbrarnos a comprender el uso de Gradient Descent, ya que al extenderse a redes neuronales, es mucho más costoso optimizar usando métodos de segundo orden.

1.4 Gradient Descent

Para ajustar los pesos del modelo, primero debemos evaluar la función compuesta que detalla el modelo.

$$Z = W \odot X + b$$

$$\hat{y} = g(Z)$$

Donde g es una función de activación, Sigmoide o Softmax.
Una vez obtenidas nuestras predicciones, procedemos a evaluar un funcional que nos permita medir el error, mediante una loss function para obtener un costo total.

$$J = \sum_{i}^{m} \mathscr{L}(y_i, \hat{y}_i)$$

Donde y_i son las predicciones conocidas a aproximar para cada entrada x_i.
A partir de la obtención del error de predicción, podemos proceder a ajustar los pesos basados en el error.

$$\frac{\partial \mathscr{L}}{\partial Z} = \frac{1}{m} * \frac{\partial \mathscr{L}}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial Z}$$

Obtenemos ahora la derivada con respecto a los pesos.

$$\frac{\partial \mathscr{L}}{\partial W} = \frac{\partial \mathscr{L}}{\partial Z} * \frac{\partial Z}{\partial W}$$

Una vez obtenida la derivada podemos ajustar los pesos.

$$W_t = W_{t-1} - \alpha * \frac{\partial \mathscr{L}}{\partial W}$$

Donde "α" es la magnitud del salto que se realizará por actualización en dirección al mínimo.

1.5 Log Softmax - Softmax Loss

Siguiendo este desarrollo definiremos la función de Pérdida como la verosimilitud negativa logarítmica de la activación para las predicciones, también llamada Log Softmax o Softmax Loss.

$$\begin{aligned} \mathscr{L} &= -log(\hat{y}) \\ &= -log\left(\frac{e^{Z}}{\sum_{j=1}^{C} e^{Z_j}}\right) \\ &= -Z_{[y==1]} + log\left(\sum_{j=1}^{C} e^{Z_j}\right) \end{aligned} \tag{3}$$

Donde la sumatoria de e^{Z_j} se realiza a lo largo de las filas de la matriz para obtener un vector colúmna de dimensiones $(m, 1)$, y la condición $[y == 1]$ establece que para cada fila, extraemos el valor de la columna en donde $Y_i = 1$, obteniendo de esta forma también un vector colúmna por lo que al final se realiza la suma de ambos valores.

Finalmente definimos la función de costo a optimizar como la media de la pérdida para todos los casos de ejemplo.

$$\begin{aligned} J(W) &= \frac{1}{m}\sum_{i=1}^{m} \mathscr{L}_i \\ &= \frac{1}{m}\sum_{i=1}^{m}\left[-Z_{[y_i==1]} + log\left(\sum_{j=1}^{C} e^{Z_{i,j}}\right)\right] \end{aligned} \tag{4}$$

1.6 Derivada de la Softmax Loss

Para optimizar los pesos o parámetros W del modelo, debemos calcular la derivada con respecto a W de todo el costo, pero para poder obtener esta derivada, primero debemos derivara el costo con respecto a los "Logits" Z. Para esto se puede llegar al mismo resultado de dos maneras distintas, el Loss desarrollado (3b), que es más directo, o desde el Loss sin desarrollar. Empezaremos por (3b).
Por fines de simplificar la notación, denotaremos $J = \mathscr{L}$

1.6.1 Derivada 1^{er} desarrollo

Aplicando sobre **(3)**

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial Z} &= \frac{1}{m}\left[-1 + \frac{1}{\sum_{j=1}^{C} e^{Z_j}} * e^{Z}\right] \\ &= \frac{1}{m}\left(-1 + \hat{y}\right) \\ &= \frac{1}{m}\left(-OneHotEncoded + \hat{y}\right) \\ &= \frac{1}{m}\left(-y + \hat{y}\right) \\ &= \frac{1}{m}\left(\hat{y} - y\right)\end{aligned}$$

Podemos observar que al aparecer un 1 en la ecuación y utilizarse una resta, se debe mantener el tamaño de la matrix $\hat{y} = (m, c)$, por lo que al sólo importar el error respecto a las predicciones que deberían ser correctas, ese 1 representa el vector de "labels" y en formato "One Hot Encoding".

1.6.2 One Hot Encoding

podemos ilustrar el One Hot Encoding, con el siguiente ejemplo:
Sea un vector de 7 elementos:

$$\mathrm{v} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 2 \\ 1 \\ 2 \end{pmatrix}$$

Su One Hot Encoding será una matriz de tamaño $(7, c = 3)$ con valor 1 en la colúmna $j = v[i]$ en la fila i :

$$\mathrm{M} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Está será el formato en que el modelo recibirá los labels y.

1.6.3 Derivada 2^{do} desarrollo

Esta forma es más tradicional y sigue la política de calcular las derivadas por regla de la cadena.

$$\frac{\partial \mathscr{L}}{\partial Z} = \frac{1}{m} * \frac{\partial \mathscr{L}}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial Z}$$

Podemos obtener fácilmente:

$$\frac{\partial \mathscr{L}}{\partial \hat{y}} = -\frac{1}{m} * \frac{1}{\hat{y}}$$

Ahora debemos obtener:

$$\begin{aligned}
\frac{\partial \hat{y}_j}{\partial Z_i} &= \frac{\left(e^{Z_i}\right)' * \sum_{j=1}^{C} e^{Z_j} - e^{Z_i} * \left(\sum_{j=1}^{C} e^{Z_j}\right)'}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2} \\
&= \frac{e^{Z_i} * \sum_{j=1}^{C} e^{Z_j} - e^{Z_i} * e^{Z_j}}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2}
\end{aligned}$$

Ahora tenemos dos situaciones que pueden ocurrir:

Caso $i = j$

$$\begin{aligned}
\frac{\partial \hat{y}_j}{\partial Z_i} &= \frac{e^{Z_i} * \sum_{j=1}^{C} e^{Z_j} - e^{Z_i} * e^{Z_j}}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2} \\
&= \frac{e^{Z_i} * \left(\sum_{j=1}^{C} e^{Z_j} - e^{Z_j}\right)}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2} \\
&= \frac{e^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}} * \frac{\left(\sum_{j=1}^{C} e^{Z_j} - e^{Z_j}\right)}{\sum_{j=1}^{C} e^{Z_j}} \\
&= \frac{e^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}} * \left(\frac{\sum_{j=1}^{C} e^{Z_j}}{\sum_{j=1}^{C} e^{Z_j}}^{1} - \frac{e^{Z_j}}{\sum_{j=1}^{C} e^{Z_j}}\right) \\
&= \frac{e^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}} * \left(1 - \frac{e^{Z_j}}{\sum_{j=1}^{C} e^{Z_j}}\right) \\
&= \hat{y}_i * (1 - \hat{y}_j) \\
&= \hat{y}_i * (1 - \hat{y}_i)
\end{aligned}$$

Caso $i \neq j$

$$\begin{aligned}
\frac{\partial \hat{y}_j}{\partial Z_i} &= \frac{e^{Z_i} * \cancelto{0}{\sum_{j=1}^{C} e^{Z_j}} - e^{Z_i} * e^{Z_j}}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2} \\
&= \frac{0 - e^{Z_i} * e^{Z_j}}{\left(\sum_{j=1}^{C} e^{Z_j}\right)^2} \\
&= -\frac{e^{Z_i}}{\sum_{j=1}^{C} e^{Z_j}} * \frac{e^{Z_j}}{\sum_{j=1}^{C} e^{Z_j}} \\
&= -\hat{y}_i * \hat{y}_j
\end{aligned}$$

Entonces tenemos que:

$$\begin{cases} \hat{y}_i * (1 - \hat{y}_i) & \text{Si: } i = j \\ -\hat{y}_i * \hat{y}_j & \text{Si: } i \neq j \end{cases}$$

Nótese que $-\hat{y}_i * \hat{y}_j = \hat{y}_i * (0 - \hat{y}_j)$, por lo que podemos reescribir la fórmula como:

$$\frac{\partial \hat{y}_j}{\partial Z_i} = \hat{y}_i * (1 - \hat{y}_j)$$

Siendo este 1 la matriz identidad, o como también se la puede representar $\delta_{i,j}$

$$\frac{\partial \hat{y}_j}{\partial Z_i} = \hat{y}_i * (\delta_{i,j} - \hat{y}_j)$$

$$\frac{\partial \hat{y}}{\partial Z} = \hat{y} * (\delta_{i,j} - \hat{y}) \tag{5}$$

Similar al caso binario utilizando una Sigmoide en vez de una Softmax.

Finalmente tenemos:

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial Z} &= \frac{1}{m} * \frac{\partial \mathscr{L}}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial Z} \\
&= -\frac{1}{m} * \frac{1}{\hat{y}} * \hat{y} * (\delta_{i,j} - \hat{y}) \\
&= \frac{1}{m} * (\hat{y} - 1) \\
&= \frac{1}{m}(\hat{y} - y)
\end{aligned}$$

1.7 Derivada Respecto a los Parámetros

Calculamos la derivada Z con respecto de los parámetros

$$\frac{\partial Z}{\partial W} = X$$

Podemos obtener las derivadas por regla de la cadena fácilmente una vez obtenidas las derivadas previas.

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial W} &= \frac{\partial \mathscr{L}}{\partial Z} * \frac{\partial Z}{\partial W} \\ &= \frac{1}{m} X^T \odot (\hat{y} - y)\end{aligned}$$

1.8 Derivada Respecto a los Sesgos

Al ser un vector colúmna para cada uno de los W_i la derivada se convierte en:

$$\frac{\partial Z}{\partial b} = 1$$

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial b} &= \frac{\partial \mathscr{L}}{\partial Z} * \frac{\partial Z}{\partial b} \\ &= \sum \frac{\partial \mathscr{L}}{\partial Z}\end{aligned}$$

Suma por filas para obtener b.

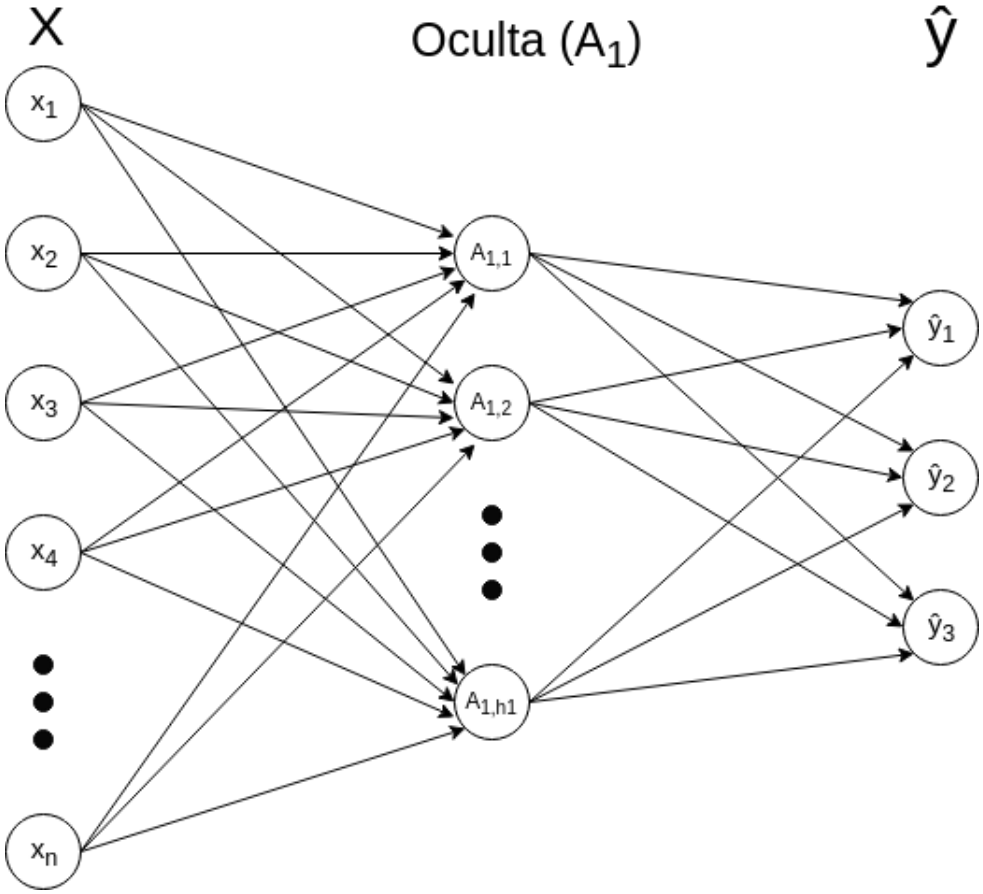
1.9 Derivada de la Entrada (Anterior Activación)

Si bien para la regresión logística, podemos observar que sólo necesitamos derivada con respecto a los parámetros, al extender a una red neuronal será necesario calcular la derivada con respecto a la activación anterior, o en este caso, la entrada "X".

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial X} &= \frac{\partial \mathscr{L}}{\partial Z} * \frac{\partial Z}{\partial X} \\ &= \frac{1}{m} * (\hat{y} - y) \odot W^T\end{aligned}$$

2 La Red Neuronal

Para una red neuronal de una capa oculta como la de la imágen tenemos que se extiende la idea de la regresión logística en una especie de concatenación de regresiones logísticas con una pequeña diferencia en la activación de las capas intermedias.



Siendo:

- La variable de entrada X de dimensiones (m, n)
- Primera capa oculta A_1 de dimensiones (m, h_1)
- Capa de salida $\hat{Y}$ de dimensiones (m, C) dónde C es el número de clases a clasificar.

Podemos expresar la red neuronal de una capa oculta A_1 mediante la siguiente composición de funciones.

$$Z_1 = X \odot W_1 + b_1$$

$$A_1 = Relu(Z_1)$$

$$Z_2 = A1 \odot W_2 + b_2$$

$$\hat{Y} = softmax(Z_2)$$

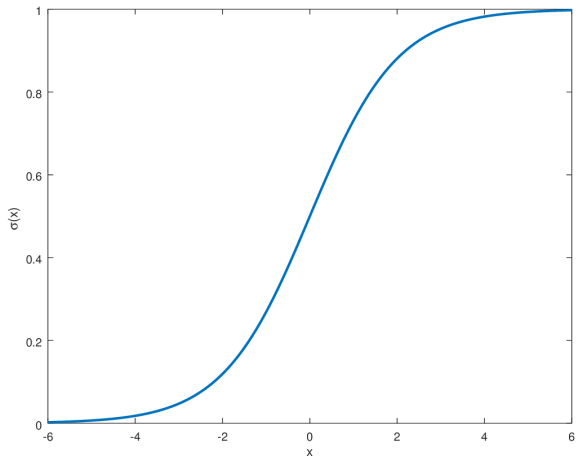
$$J = \sum_{i}^{m} \mathscr{L}(y_i, \hat{y}_i)$$

2.1 Funciones de Activación

Si sólamente compusieramos las operaciones $A_{l-1} \odot W_l + b_l$ para obtener A_l (siendo l la capa o layer actual) sería exáctamente lo mismo a tener una simple regresión logística con múchos parámentros inútiles, es por esto que para aprovechar realmente el poder que nos brinda una red neuronal por sobre una regresión logística, debemos aplicar una función no lineal, para agregar no linealidad a nuestro predictor, para de esta forma resolver problemas de clasificación mucho más difíciles que sólo ajustando una línea o hiper plano al espacio para separar nuestros datos no sea suficiente.

A estas funciones no lineales se les llama función de activación, y existen muchas, desde la sigmoide que se utilizó desde los inicios, hasta las actuales Relu, Elu y PRelu.

Se dejó de utilizar la sigmoide como función de activación intermedia debido a un problema que se observa en la misma gráfica de la función.



Al calcular derivadas sobre esta función tendremos el problema de que en números muy grandes o muy pequeños, nuestro gradiente tiende a cero, por lo tanto llegará un punto donde nuestro entrenamiento puede llegar a estancarse, a parte que la derivada de la función es muy costosa de calcular debido a la existencia de exponenciales.

Debido a estos problemas es que se decidió utilizar una función mucho menos costosa llamada Rectified Linear Unit o Relu, que se define de la siguiente forma:

$$Relu = Max(0, Z) \ \forall \ z_j \ / \ j : 0, 1, ..., h_l \tag{6}$$

Donde h_l representa el tamaño de la capa "l".
Como aprendimos hasta ahora, la red neuronal se entrenará mediante el Gradient Descent, para este necesitamos derivadas de cada uno de los parámetro en la composición de funciones por regla de la cadena, por lo tanto necesitaremos la derivada de la función relu, la cual es la siguiente.

$$\frac{\partial Relu}{\partial Z} = \begin{cases} 1 \text{ si } z_j > 0 \\ 0 \text{ en otro caso} \end{cases} \tag{7}$$

2.2 Backpropagation

Para entrenar la red neuronal, al tener más capas por las que pasar para obtener todos los gradientes, debemos derivar a través de cada una de ellas, a este algoritmo se le llama Backpropagation o Retropropagación, que es simplemente como su nombre dice, propagar los gradientes de reverso a través de la red neuronal.

Primero obtenemos cada una de las derivadas en base al procedimiento de composición de funciones dado anteriormente y luego exploraremos usando un grafo computacional cómo se observa la jerarquía de operaciones.

$$\frac{\partial \mathscr{L}}{\partial \hat{y}} = \frac{1}{m} * \frac{1}{\hat{y}}$$

$$\frac{\partial \mathscr{L}}{\partial Z_2} = \frac{1}{m}(\hat{y} - y)$$

$$\frac{\partial \mathscr{L}}{\partial W_2} = \frac{1}{m} A_1^T \odot (\hat{y} - y)$$

$$\frac{\partial \mathscr{L}}{\partial b_2} = \sum \frac{\partial \mathscr{L}}{\partial Z_2}$$

$$\frac{\partial \mathscr{L}}{\partial A_1} = \frac{1}{m} * (\hat{y} - y) \odot W_2^T$$

$$\frac{\partial \mathscr{L}}{\partial Z_1} = \frac{\partial \mathscr{L}}{\partial A_1} * \frac{\partial A_1}{\partial Z_1} = \frac{1}{m} * (\hat{y} - y) \odot W_2^T * \begin{cases} 1 \text{ si } z_1^j > 0 \\ 0 \text{ en otro caso} \end{cases}$$

$$\frac{\partial \mathscr{L}}{\partial W_1} = \frac{\partial \mathscr{L}}{\partial Z_1} * \frac{\partial Z_1}{\partial W_1} = \frac{1}{m} * X^T \odot \left[(\hat{y} - y) \odot W_2^T * \begin{cases} 1 \text{ si } z_1^j > 0 \\ 0 \text{ en otro caso} \end{cases} \right]$$

$$\frac{\partial \mathscr{L}}{\partial b_1} = \sum \frac{\partial \mathscr{L}}{\partial Z_1}$$