¿Cómo se entrenan las redes neuronales?

Jocelyn Dunstan Escudero

jdunstan@uc.cl

Departamento de Ciencia de la Computación
& Instituto de Matemática Computacional

Pontificia Universidad Católica de Chile

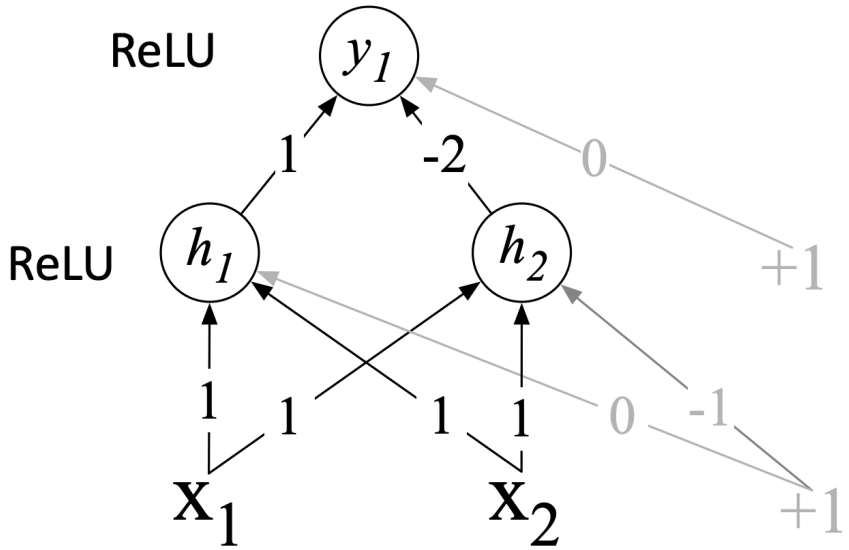
29 de octubre de 2025

Objetivos

- Distinguir entre los parámetros de una red (*weights and biases*, que se entrenan) y los hiperparámetros (que escoge el/la programador/a).
- Introducir el concepto de grafo de computación.
- Entrenar una red neuronal sencilla.

De la última clase: el XOR se requiere un perceptrón multicapa

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



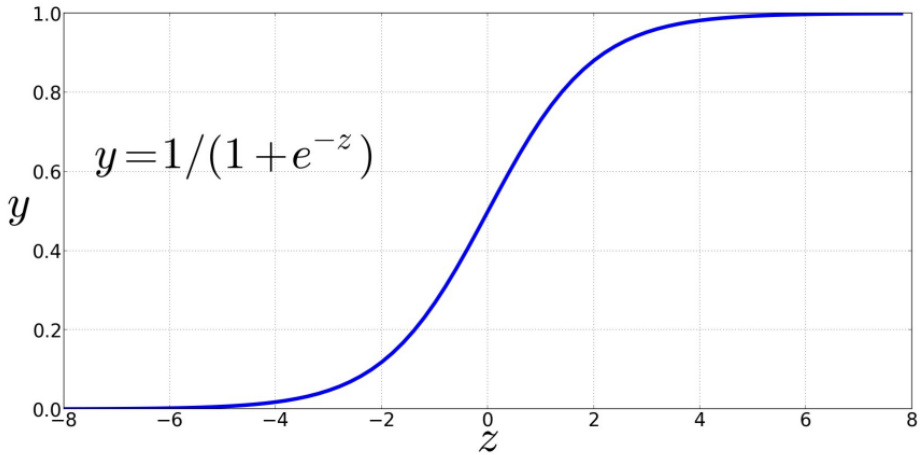
https://web.stanford.edu/~jurafsky/slp3/6.pdf

Y con ello introdujimos la función de activación

$$z = w \cdot x + b$$
$$y = a = f(z)$$

Sigmoid

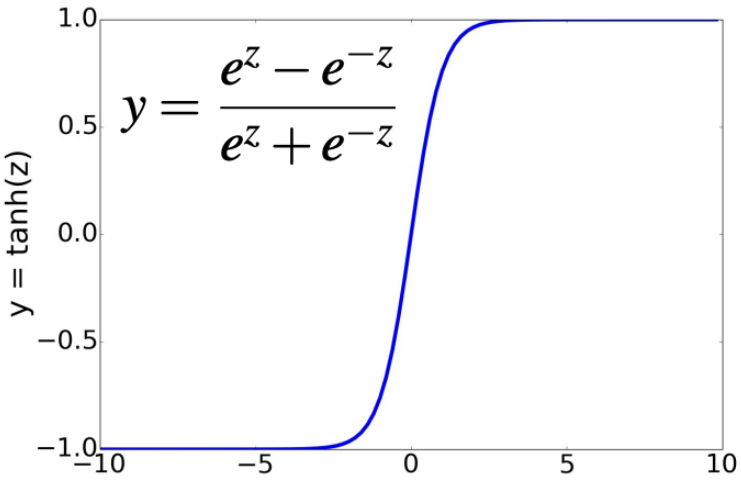
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



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

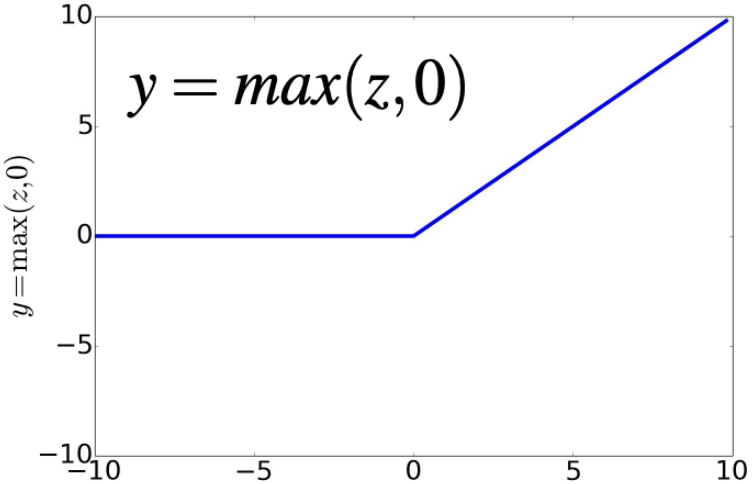
https://web.stanford.edu/~jurafsky/slp3/6.pdf

Otras funciones de activación (¡hay más!)



tanh

Most Common:



ReLU
Rectified Linear Unit

https://web.stanford.edu/~jurafsky/slp3/6.pdf

Red neuronal sencilla

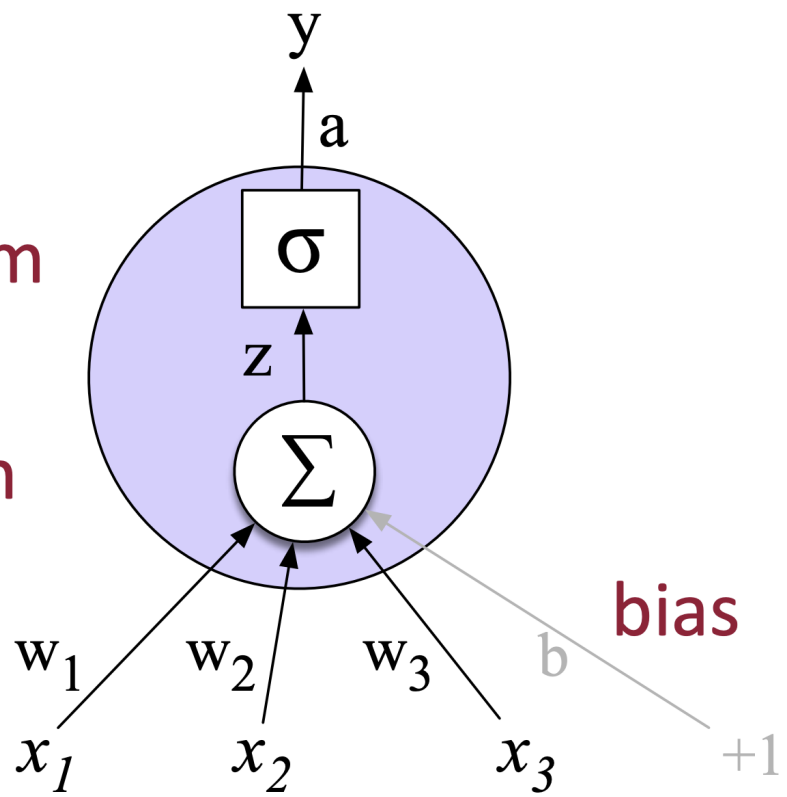
Output value

Non-linear transform

Weighted sum

Weights

Input layer



https://web.stanford.edu/~jurafsky/slp3/6.pdf

¿Cómo se entrena una red neuronal?

- Encontrando los valores de los pesos y los sesgos (*weights & biases*) . . . ¿Pero como encontramos esos valores?

¿Cómo se entrena una red neuronal?

- Encontrando los valores de los pesos y los sesgos (*weights & biases*) . . . ¿Pero como encontramos esos valores?

- ¡La entrenamos con el algoritmo de backpropagation!

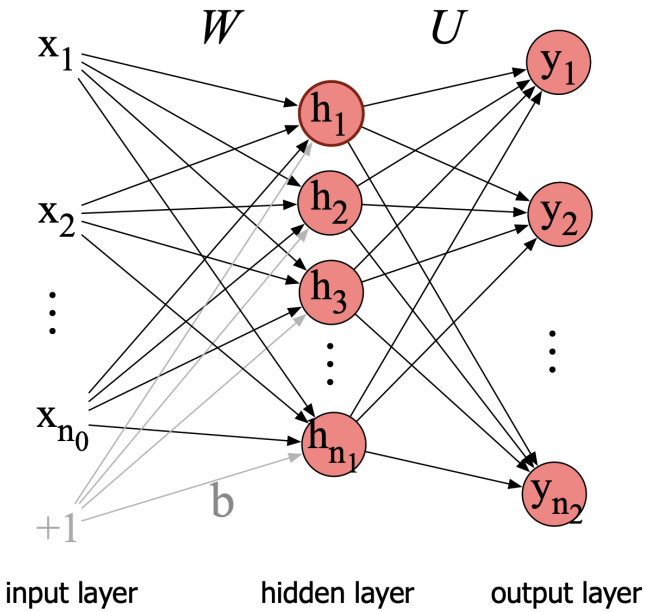
¿Cómo se entrena una red neuronal?

- Encontrando los valores de los pesos y los sesgos (*weights & biases*) . . . ¿Pero como encontramos esos valores?
- ¡La entrenamos con el algoritmo de backpropagation!
- ¿Pero qué significa entrenar en este contexto?

¿Cómo se entrena una red neuronal?

- Encontrando los valores de los pesos y los sesgos (*weights & biases*) . . . ¿Pero como encontramos esos valores?
- ¡La entrenamos con el algoritmo de backpropagation!
- ¿Pero qué significa entrenar en este contexto?
- Mostrar un conjunto de ejemplos (*batches*) y penalizar usando la función de pérdida (*loss*).

Pero antes de eso... notación matricial

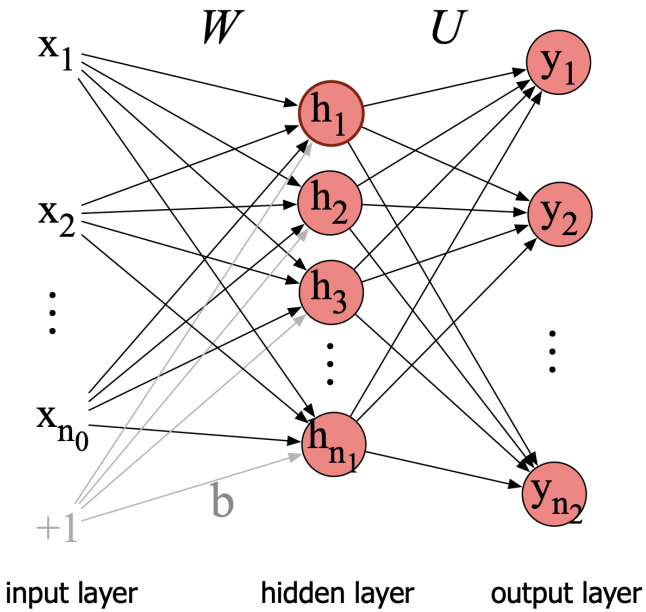


$$\mathbf{h} = f(\mathbf{W}\mathbf{X} + \mathbf{b})$$

$$\mathbf{y} = g(\mathbf{h}\mathbf{U} + \mathbf{c})$$

donde f y g son funciones de activación.

Pero antes de eso... notación matricial



$$\mathbf{h} = f(\mathbf{W}\mathbf{X} + \mathbf{b})$$

$$\mathbf{y} = g(\mathbf{h}\mathbf{U} + \mathbf{c})$$

donde f y g son funciones de activación.

De manera más general, diremos que los elementos de la capa oculta i son:

$$\mathbf{h^{(i)}} = f^{(i)}(\mathbf{W^{(i)}}\mathbf{h^{(i-1)}} + \mathbf{b^{(i)}})$$

Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.

Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.
- Los parámetros por encontrar de la red son $\theta = \{\mathbf{W^{(1)}}, \mathbf{b^{(1)}}, ..., \mathbf{W^{(r)}}, \mathbf{b^{(r)}}, \mathbf{U}, \mathbf{c}\}$, donde r es el número de capas ocultas.

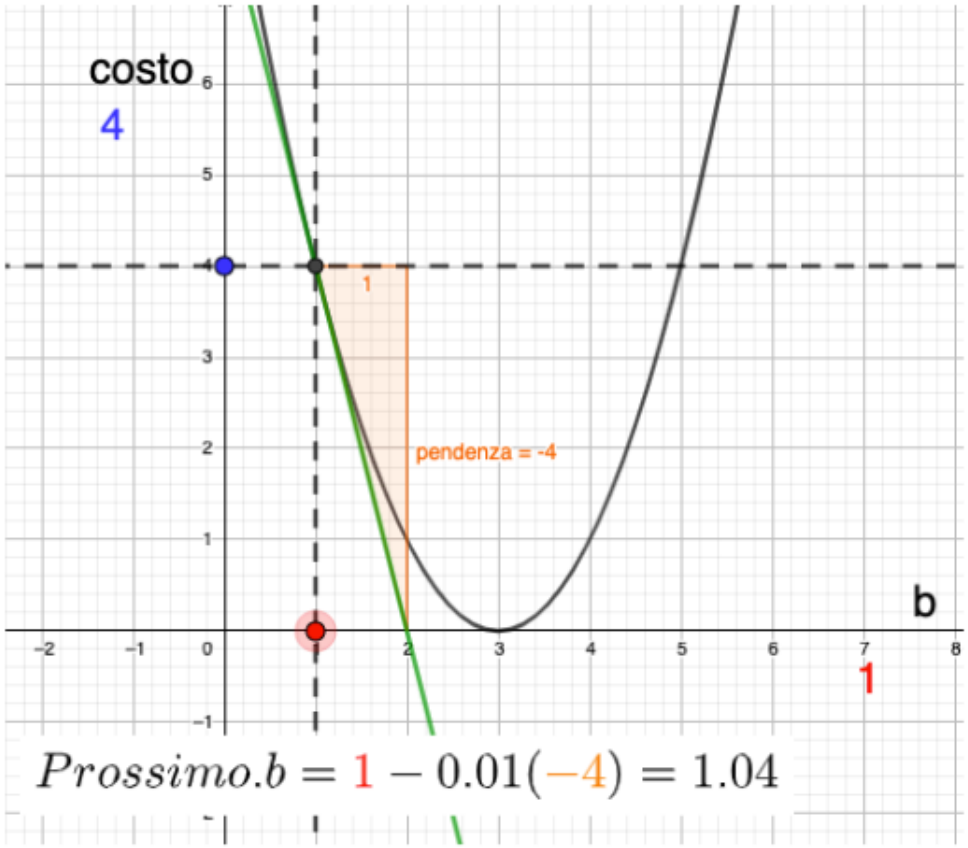
Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.
- Los parámetros por encontrar de la red son $\theta = \{\mathbf{W^{(1)}}, \mathbf{b^{(1)}}, ..., \mathbf{W^{(r)}}, \mathbf{b^{(r)}}, \mathbf{U}, \mathbf{c}\}$, donde r es el número de capas ocultas.
- La ecuación para encontrar un parámetro θ^* es:

$$\theta^* = \theta^* - \eta \frac{\partial \mathcal{L}}{\partial \theta},$$

donde η es la tasa de aprendizaje (*learning rate*) y $\mathcal{L}$ es la función de pérdida (*loss function*).

Learning rate: fracción de la pendiente que uso para acercarme al valor óptimo



https://www.geogebra.org/m/DwJZjmdy

Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.

Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.

- Los parámetros por encontrar de la red son $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, ..., \mathbf{W}^{(r)}, \mathbf{b}^{(r)}, \mathbf{U}, \mathbf{c}\}$, donde r es el número de capas ocultas.

Actualización de los parámetros

- Sea un *batch* (lote) de datos $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$.

- Los parámetros por encontrar de la red son $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, ..., \mathbf{W}^{(r)}, \mathbf{b}^{(r)}, \mathbf{U}, \mathbf{c}\}$, donde r es el número de capas ocultas.

- La ecuación para encontrar un parámetro θ^* es:

$$\theta^* = \theta^* - \eta \frac{\partial \mathcal{L}}{\partial \theta},$$

donde η es la tasa de aprendizaje (*learning rate*) y $\mathcal{L}$ es la función de pérdida (*loss function*).

Función de pérdida

- $\mathcal{L}$ es la función de pérdida. Si y es el valor real, e $\hat{y}$ es la predicción que da la red,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} error(\hat{y}_i, y_i)$$

Función de pérdida

- $\mathcal{L}$ es la función de pérdida. Si y es el valor real, e $\hat{y}$ es la predicción que da la red,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} error(\hat{y}_i, y_i)$$

- Cuando el problema es de regresión, $\mathcal{L}$ es el error cuadrático medio:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Función de pérdida

- $\mathcal{L}$ es la función de pérdida. Si y es el valor real, e $\hat{y}$ es la predicción que da la red,

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} error(\hat{y}_i, y_i)$$

- Cuando el problema es de regresión, $\mathcal{L}$ es el error cuadrático medio:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

- Cuando la salida de la red es un vector de k clases, se usa la entropía cruzada:

$$\mathcal{L}_{CE} = -\sum_{j=1}^{K} y_j \log(\hat{y}_j).$$

Para dos clases, $\mathcal{L}_{CE} = -[y \log(\hat{y} + (1-y) \log(1-\hat{y})]$.

¿Cómo se calcula en la práctica?

Usando grafos computacionales (computational graphs)

Ejemplo sencillo de grafo computacional: forward

Suponga que $L(a, b, c) = c(a + 2b)$. Uno puede hacer esta suma y multiplicación explícita, y agregar los nodos intermedios d y e, de modo que:

$$
\begin{aligned}
d &= 2 * b \\
e &= a + d \\
L &= c * e.
\end{aligned}
$$

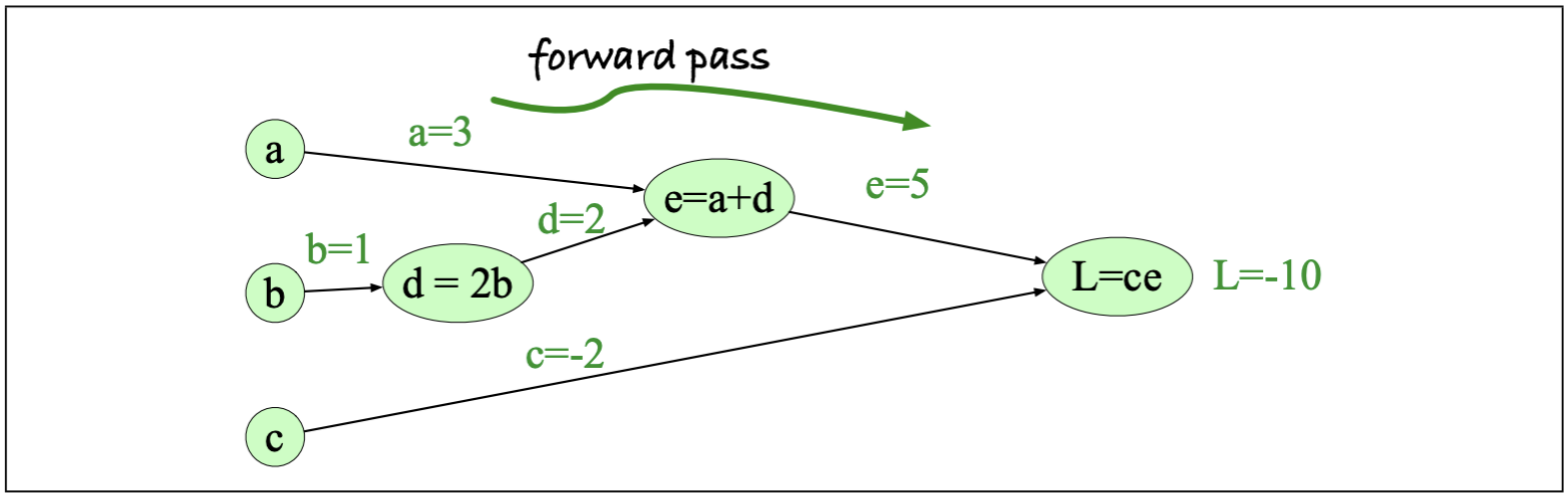


Figure 6.15 Computation graph for the function $L(a,b,c) = c(a+2b)$, with values for input nodes $a = 3$, $b = 1$, $c = -2$, showing the forward pass computation of L.

Ejemplo sencillo de grafo computacional: backward

Y puedo calcular las derivadas parciales y calcular el paso en reversa (o *backward)*

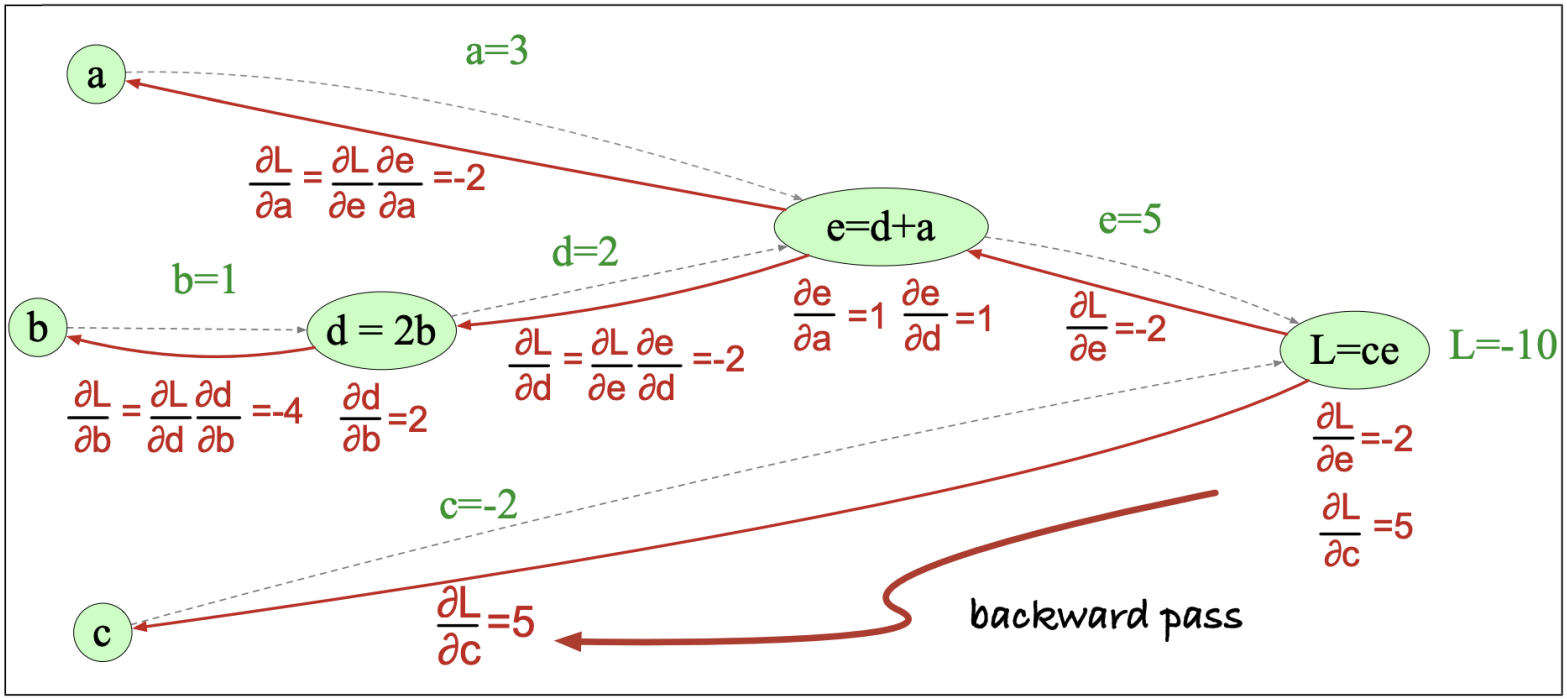
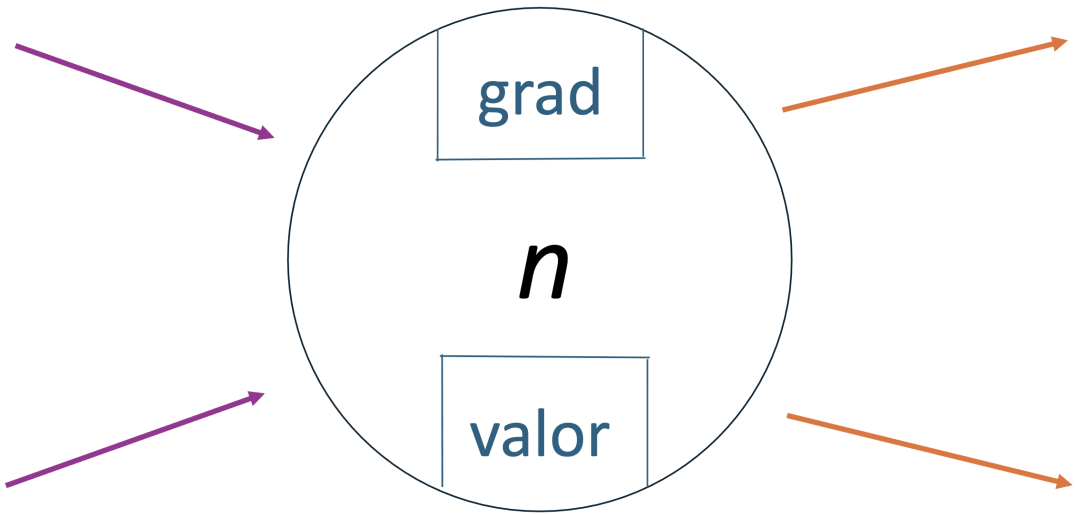


Figure 6.17 Computation graph for the function $L(a,b,c) = c(a+2b)$, showing the backward pass computation of $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$.

https://web.stanford.edu/~jurafsky/slp3/6.pdf

Grafo computacional

En cada nodo guardo el valor que toma en la pasada hacia adelante, y el valor analítico del gradiente de $\mathcal{L}$ con respecto a la variable n.



https://www.youtube.com/watch?v=1EUAoM1EhM0&list=PLBjZ-ginWc1e0_Dp4heHglsjJmacV_F20&index=5

Grafo para una red neuronal sencilla

$$
\begin{aligned}
h &= w_1 x + b_1 \\
y &= w_2 h + b_2 \\
L &= (y - t)^2.
\end{aligned}
$$

Suponga $x = 2$, $t = 5$, y que inicializamos la red con $w_1 = 1$, $w_2 = 2$ y $b_1 = b_2 = 0$.

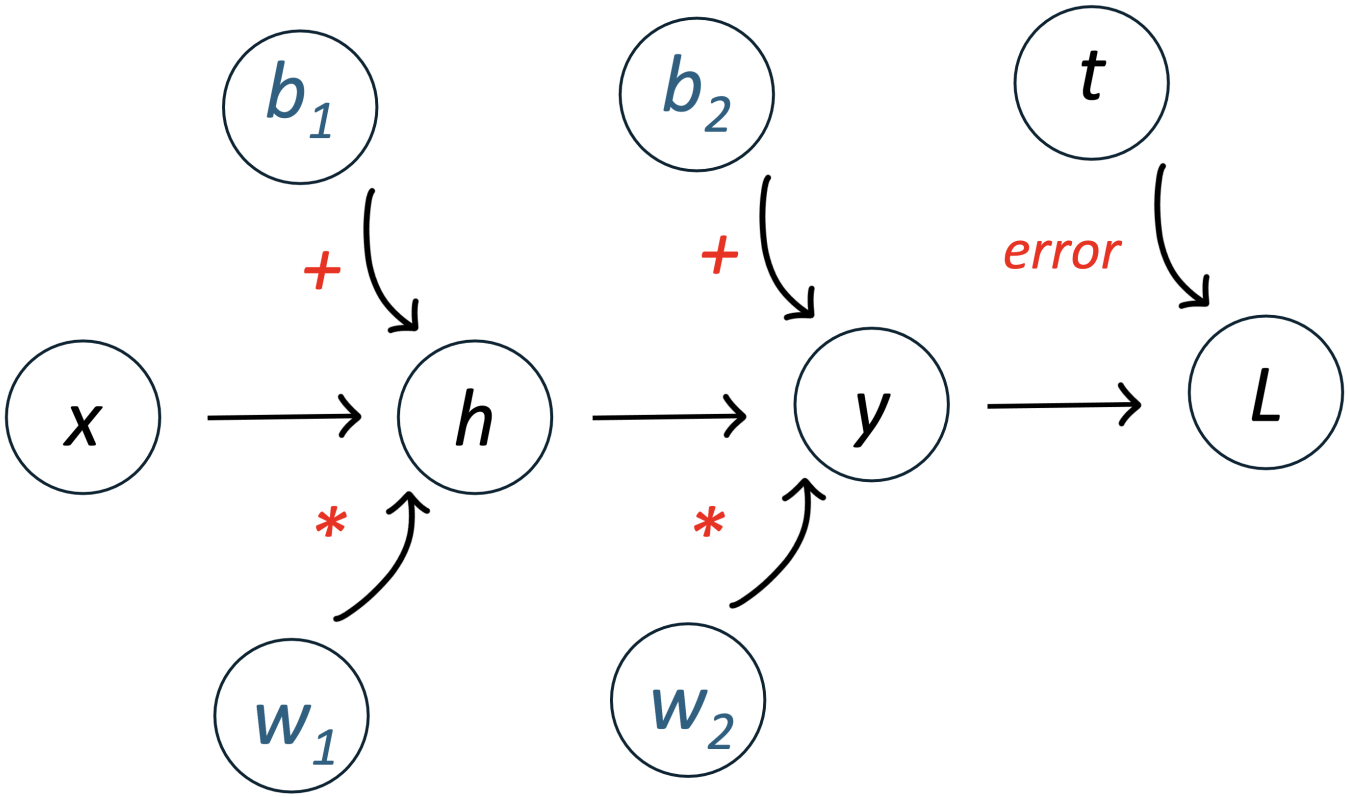
Grafo para una red neuronal sencilla

$$
\begin{aligned}
h &= w_1 x + b_1 \\
y &= w_2 h + b_2 \\
L &= (y - t)^2.
\end{aligned}
$$

Suponga $x = 2$, $t = 5$, y que inicializamos la red con $w_1 = 1$, $w_2 = 2$ y $b_1 = b_2 = 0$.

Dibuje el grafo computacional y calcule derivadas parciales.

Grafo para una red neuronal sencilla



Forward pass

$$
\begin{aligned}
h &= 1 * 2 + 0 = 2 \\
y &= 2 * 2 + 0 = 4 \\
L &= (4 - 5)^2 = 1.
\end{aligned}
$$

Backward pass: cálculo de derivadas parciales

$$\frac{\partial L}{\partial y} = 2(y - t)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w_2} = 2(y - t)\, h$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial b_2} = 2(y - t)$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial h} = 2(y - t)\, w_2$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial w_1} = 2(y - t)\, w_2\, x$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial b_1} = 2(y - t)\, w_2$$

Backward pass: Evaluando las derivadas

Parámetro	**Gradiente**
w_1	-8
b_1	-4
w_2	-4
b_2	-2

Backward pass: Evaluando las derivadas

Parámetro	Gradiente
w_1	-8
b_1	-4
w_2	-4
b_2	-2

Actualizando los parámetros con $\eta = 0{,}1$:

$$w_1' = 1 - 0{,}1(-8) = 1{,}8, \qquad w_2' = 2 - 0{,}1(-4) = 2{,}4,$$

$$b_1' = 0 - 0{,}1(-4) = 0{,}4, \qquad b_2' = 0 - 0{,}1(-2) = 0{,}2$$

Algunos comentarios finales

- Los parámetros de la red son los pesos y los sesgos, y estos se encuentran por decenso de gradiente usando el algoritmo de *backpropagation*.

Algunos comentarios finales

- Los parámetros de la red son los pesos y los sesgos, y estos se encuentran por decenso de gradiente usando el algoritmo de *backpropagation*.

- Hiperparámetros incluyen la tasa de aprendizaje η, el tamaño del batch, el número de capas, el número de nodos de cada capa, las funciones de activación, como regularizar... El decenso del gradiente también tiene elecciones, siendo Adam una muy común.

Algunos comentarios finales

- Los parámetros de la red son los pesos y los sesgos, y estos se encuentran por decenso de gradiente usando el algoritmo de *backpropagation*.

- Hiperparámetros incluyen la tasa de aprendizaje η, el tamaño del batch, el número de capas, el número de nodos de cada capa, las funciones de activación, como regularizar... El decenso del gradiente también tiene elecciones, siendo Adam una muy común.

- Usar grafos computacionales permite la paralelización usando unidades gráficas de procesamiento (GPUs). Lo más común es usar PyTorch o TensorFlow.