[course site]

#DLUPC

Day 2 Lecture 2

Backpropagation

Xavier Giro-i-Nieto
xavier.giro@upc.edu

Associate Professor
Universitat Politècnica de Catalunya
Technical University of Catalonia

Acknowledgements

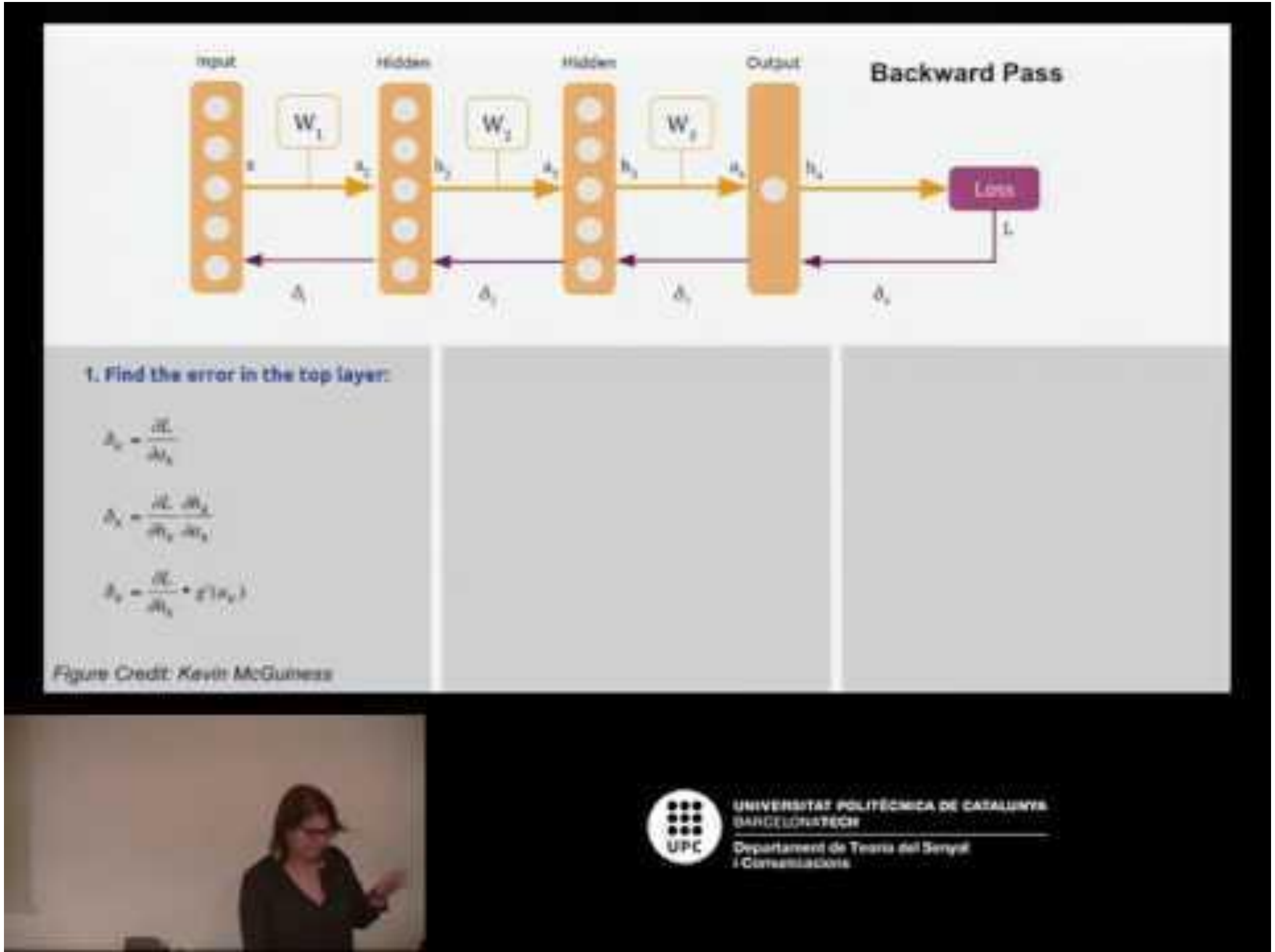
[Kevin McGuinness](#)
kevin.mcguinness@dcu.ie

Research Fellow
Insight Centre for Data Analytics
Dublin City University

[Elisa Sayrol](#)
elisa.sayrol@upc.edu

Associate Professor
ETSETB TelecomBCN
Universitat Politècnica de Catalunya

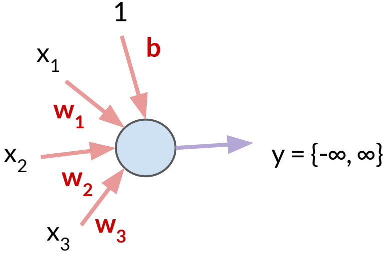
Video lecture



Loss function - L(y, ŷ)

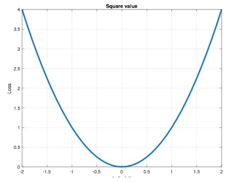
The **loss function** assesses the performance of our model by comparing its predictions (y) to an expected value (ŷ), typically coming from ground truth labels.

<u>Example</u>: the predicted price (y) and one actually paid (ŷ) could be compared with the Euclidean distance (also referred as L2 distance or Mean Square Error - MSE):



$$y = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3 + b = \mathbf{w}^\mathbf{T} \cdot \mathbf{x} + b$$

$$\mathcal{L}_2(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$



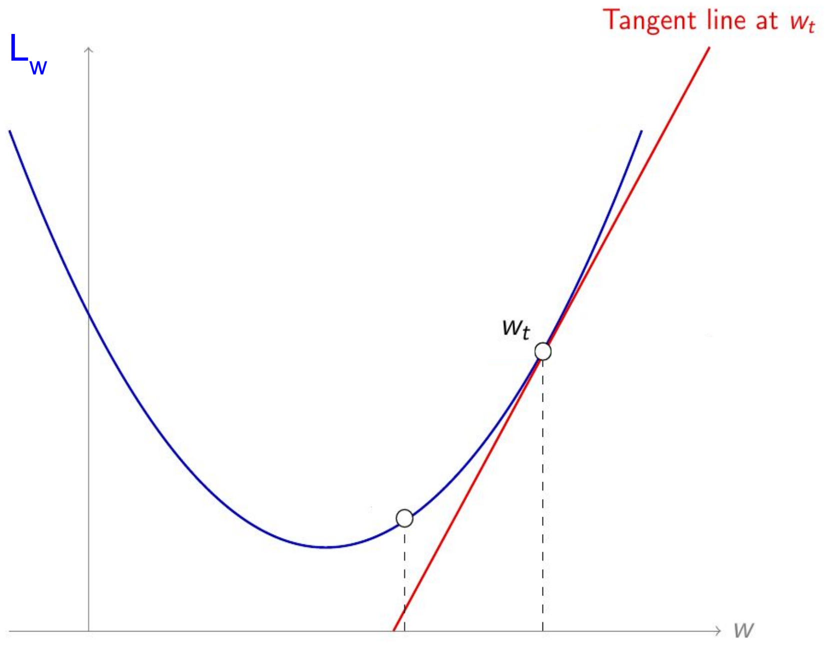
Loss function - L(y, ŷ)

<u>Discussion</u>: Consider the very simple model...

$$y = w \cdot x$$

…....and that, given a pair (y, ŷ), we would like to update the current w_t value to a new w_{t+1} based on the loss function L_w.

(a) Would you increase or decrease w_t ?
(b) What operation could indicate which way to go ?
(c) How much would you increase or decrease w_t ?



Gradient Descent (GD)

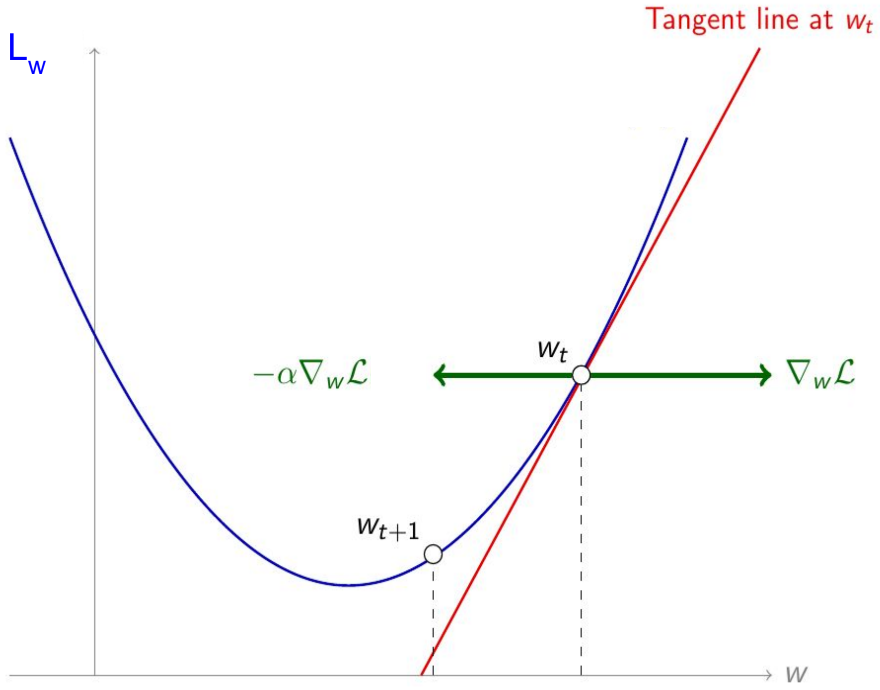
<u>Motivation for this lecture</u>:

if we had a way to compute the gradient of the loss with respect to the parameter(s), we could use gradient descent to optimize them.

Descend
(minus sign)

$$w_{t+1} \leftarrow w_t - \alpha \nabla \mathcal{L}_w(w_t)$$

Learning rate (LR)

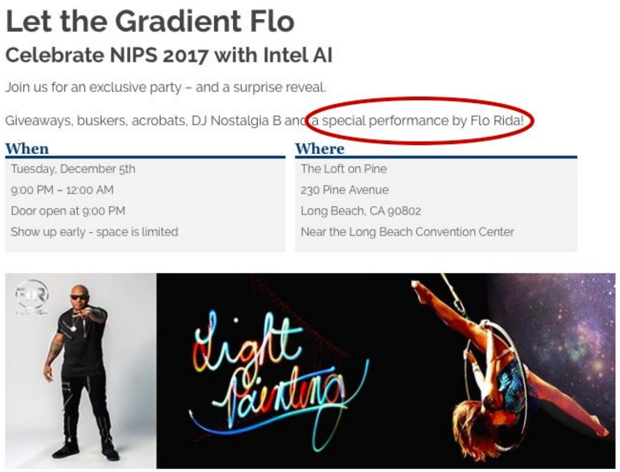


Gradient Descent (GD)

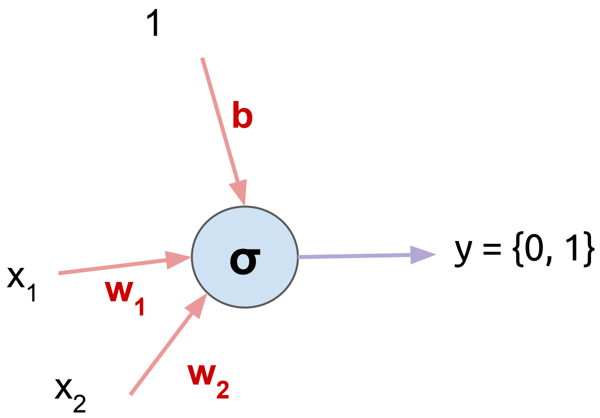
Backpropagation will allow us to compute the **gradients of the loss function** with respect to:

- all model parameters (**w**) - final goal during training
- input/intermediate data - visualization & interpretability purposes.

Gradients will **“flow”** from the output of the model towards the input (“back”).

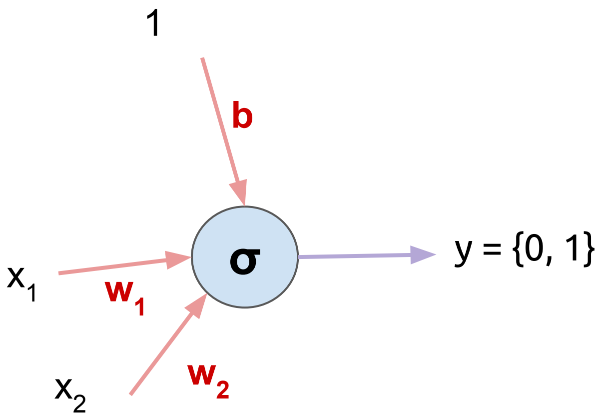


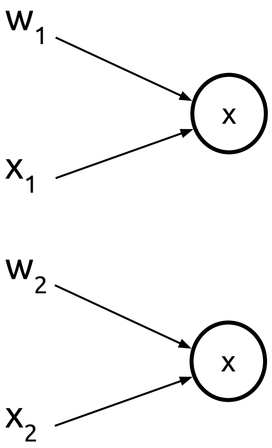
Computational graph of a simple perceptron



<u>Question</u>: What is the computational graph (operations & order) of this perceptron with a sigmoid activation ?

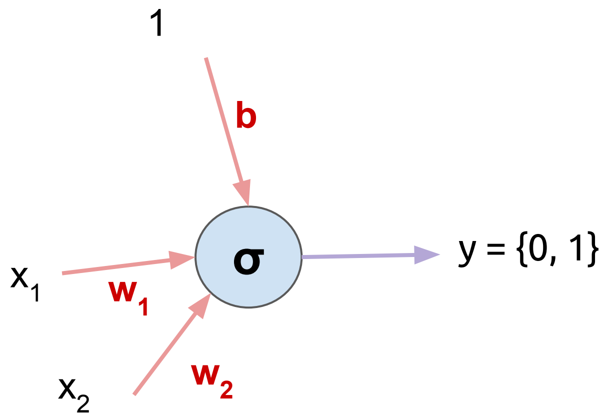
Computational graph of a perceptron

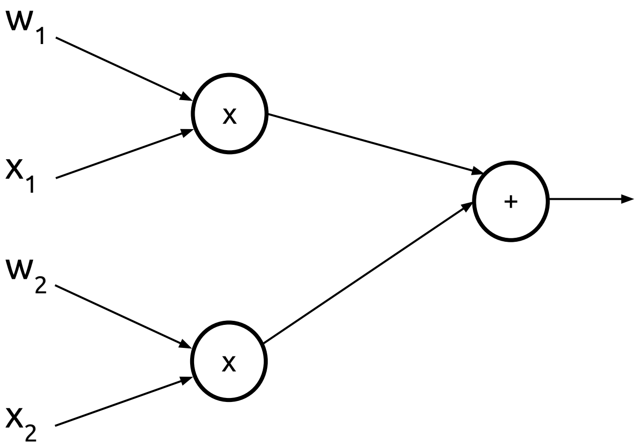




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

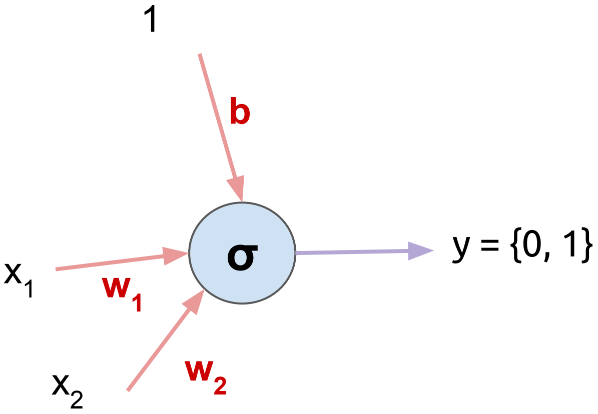
Computational graph of a perceptron

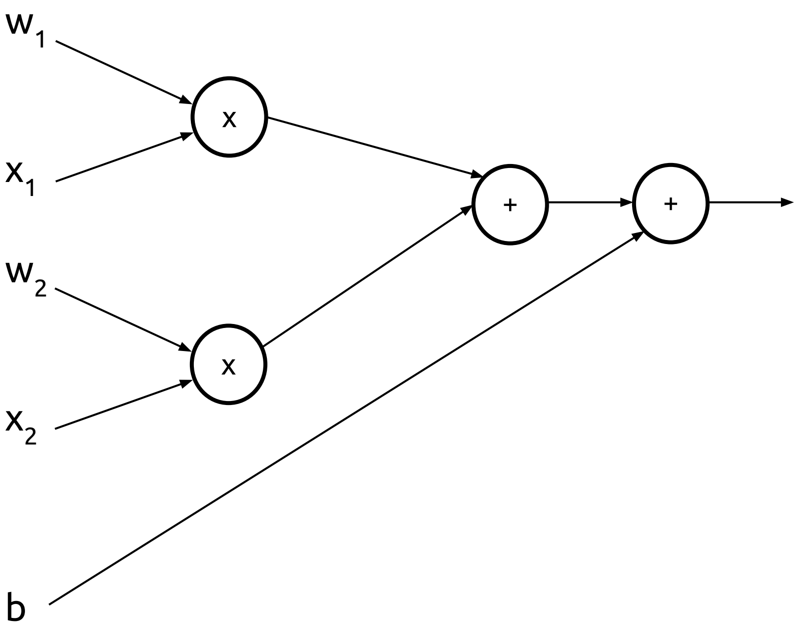




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

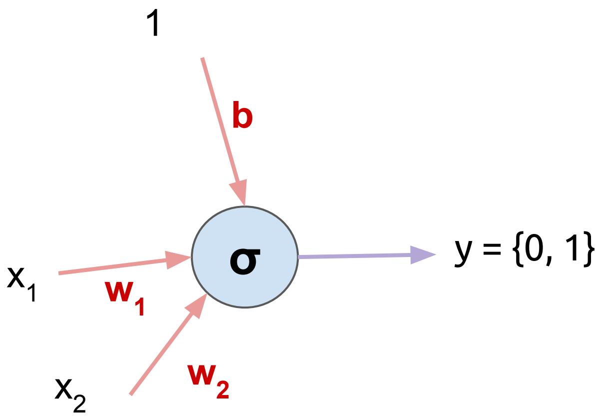
Computational graph of a perceptron

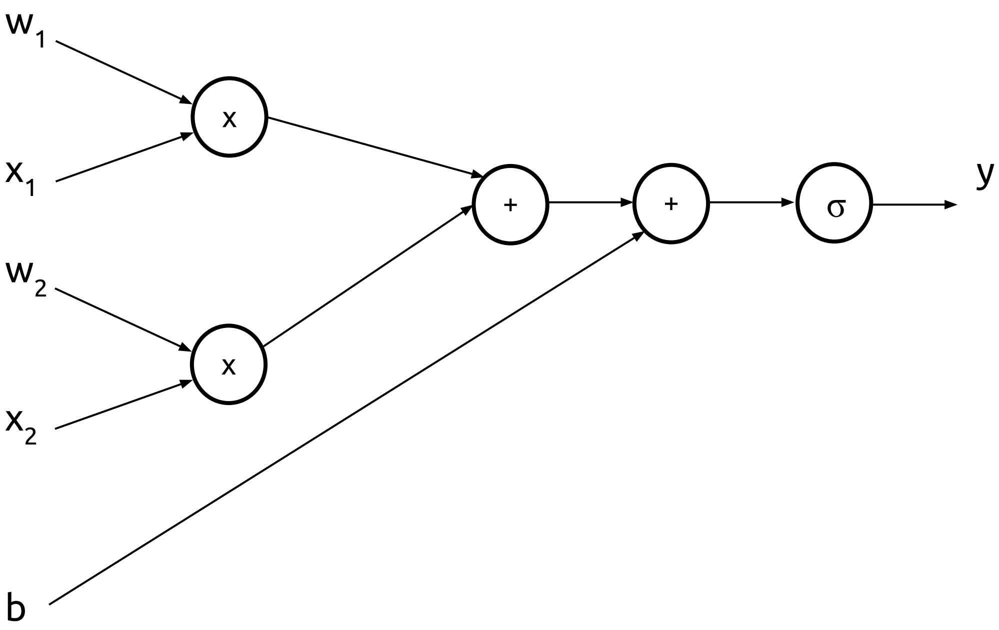




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

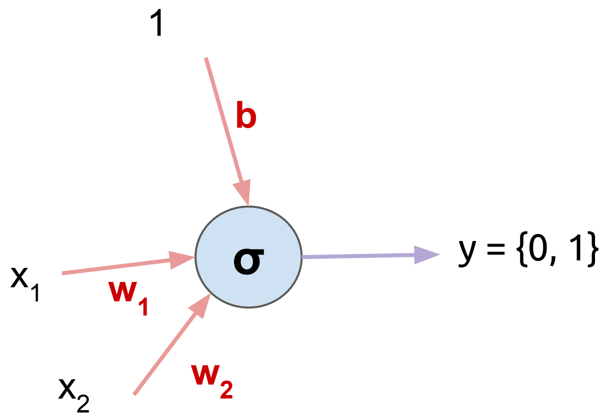
Computational graph of a perceptron

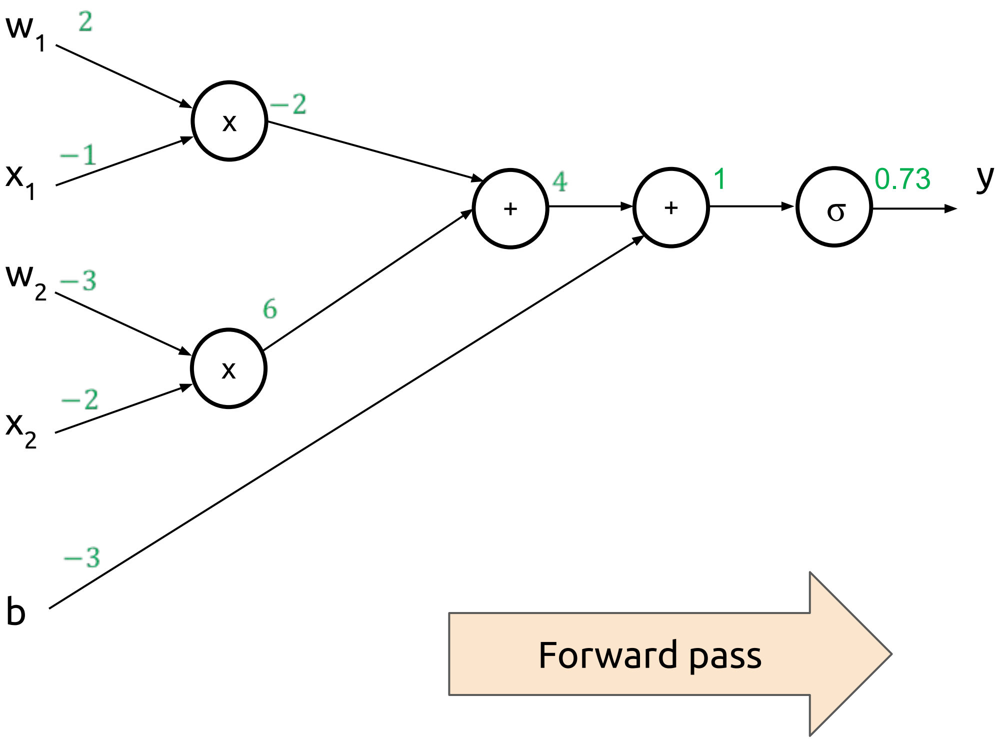




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

Computational graph of a perceptron



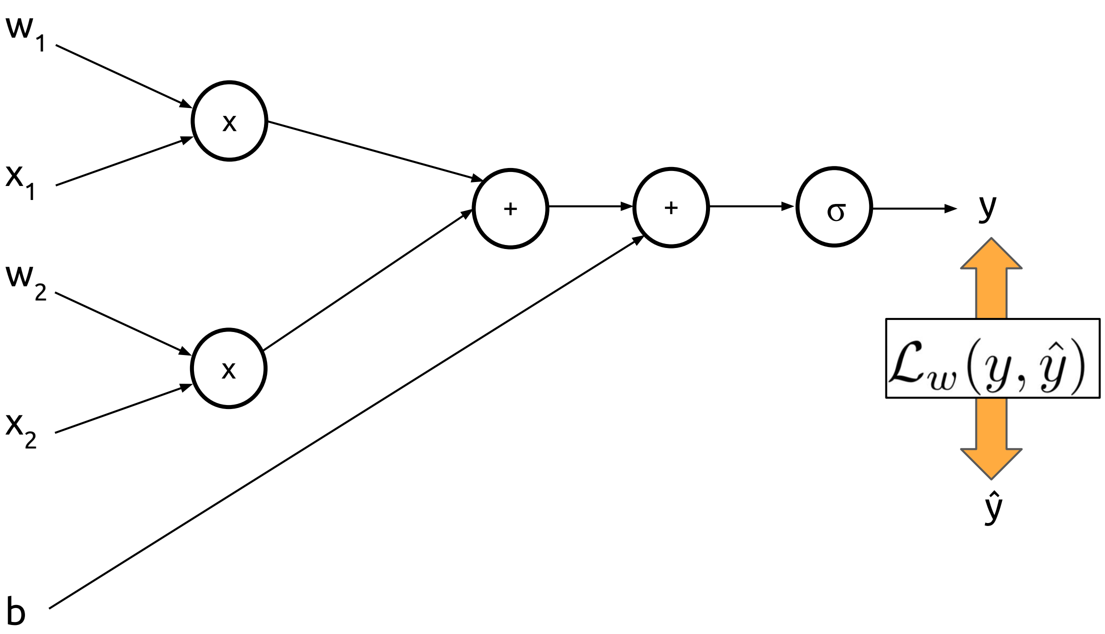


Computational graph of a perceptron

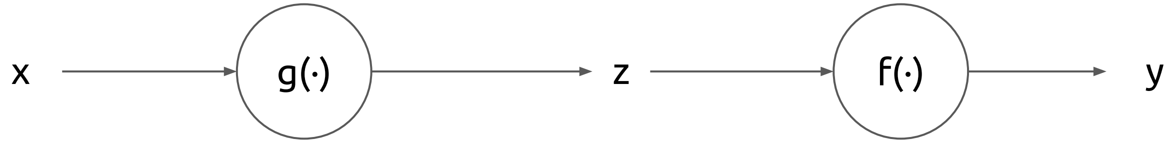
Challenge: How to compute the gradient of the loss function with respect to w_1 or w_2 ?

$$\frac{\partial \mathcal{L}_w(y, \hat{y})}{\partial w_1} = ?$$

$$\frac{\partial \mathcal{L}_w(y, \hat{y})}{\partial w_2} = ?$$



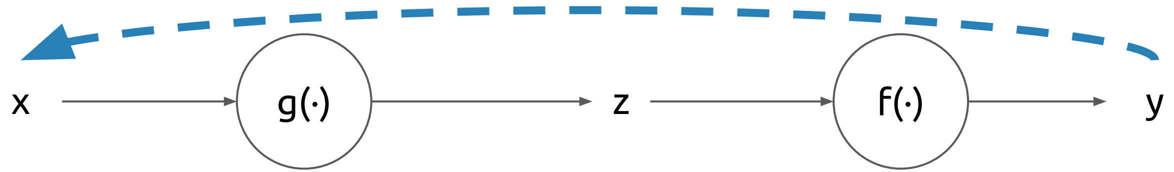
Gradients from composition (chain rule)



$$y = f(z)$$
$$z = g(x)$$

Forward pass

Gradients from composition (chain rule)



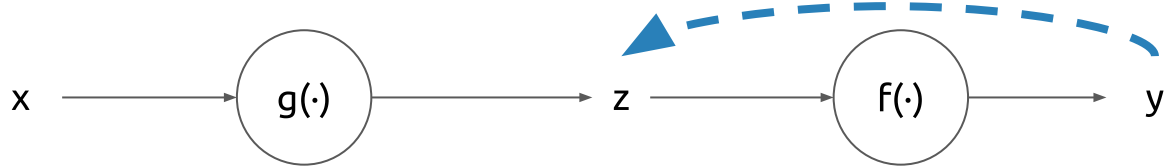
How does a variation (“difference”) on x affect y ?

$$y = f(z)$$
$$z = g(x)$$

$$\frac{dy}{dx} = \text{?}$$

Backward pass

Gradients from composition (chain rule)



How does a variation ("difference") on x affect y ?

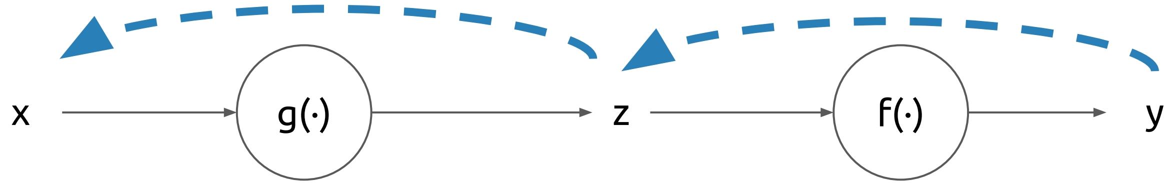
$$y = f(z)$$
$$z = g(x)$$

It will depend on the how y is affected by a variation in z...

$$\frac{dy}{dx} = \boxed{\frac{dy}{dz}}\frac{dz}{dx}$$

Backward pass

Gradients from composition (chain rule)



How does a variation (“difference”) on x affect y ?

$$y = f(z)$$
$$z = g(x)$$

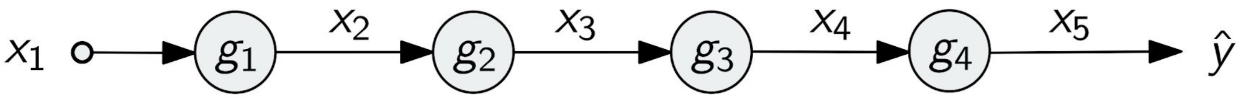
It will depend on the how y is affected by a variation in z...

$$\frac{dy}{dx} = \boxed{\frac{dy}{dz}}\boxed{\frac{dz}{dx}}$$

...scaled (multiplied) by how z is affected with a variation in x.

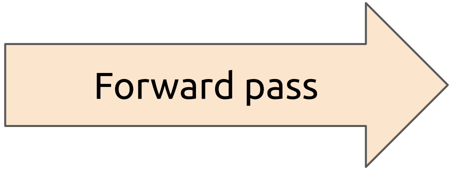
Gradients from composition (chain rule)

$$\hat{y} = g_4(g_3(g_2(g_1(x_1))))$$



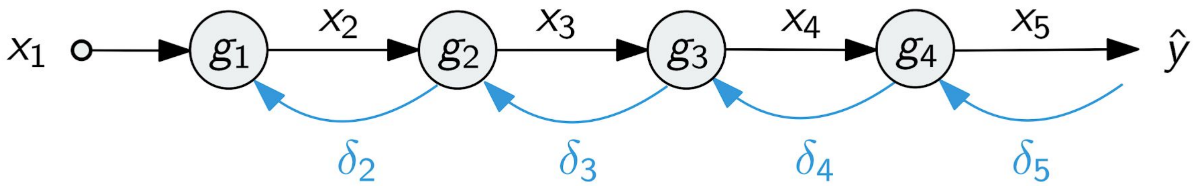
Decompose into steps (**forward propagation**):

$$\begin{aligned} x_2 &= g_1(x_1) \\ x_3 &= g_2(x_2) \\ x_4 &= g_3(x_3) \\ \hat{y} = x_5 &= g_4(x_4) \end{aligned}$$



Gradients from composition (chain rule)

$$\hat{y} = g_4(g_3(g_2(g_1(x_1))))$$



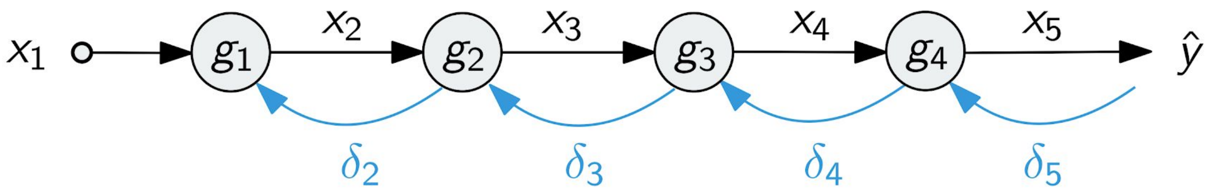
Want to find $\frac{\partial \hat{y}}{\partial x_1}$. Chain rule:

$$\frac{\partial \hat{y}}{\partial x_1} = \frac{\partial \hat{y}}{\partial x_4} \frac{\partial x_4}{\partial x_3} \frac{\partial x_3}{\partial x_2} \frac{\partial x_2}{\partial x_1}$$

How does a variation ("difference") on the input affect the prediction ?

Backward pass

Gradients from composition (chain rule)



Decompose into steps again. Let $\delta_k = \frac{\partial \hat{y}}{\partial x_k}$. **Backpropagation**:

$$\delta_5 = \frac{\partial \hat{y}}{\partial x_5} = 1$$

$$\delta_4 = \frac{\partial \hat{y}}{\partial x_4} = \frac{\partial \hat{y}}{\partial x_5}\frac{\partial x_5}{\partial x_4} = \delta_5 g_4'(x_4)$$

$$\delta_3 = \frac{\partial \hat{y}}{\partial x_3} = \frac{\partial \hat{y}}{\partial x_4}\frac{\partial x_4}{\partial x_3} = \delta_4 g_3'(x_3)$$

$$\delta_2 = \frac{\partial \hat{y}}{\partial x_2} = \frac{\partial \hat{y}}{\partial x_3}\frac{\partial x_3}{\partial x_2} = \delta_3 g_2'(x_2)$$

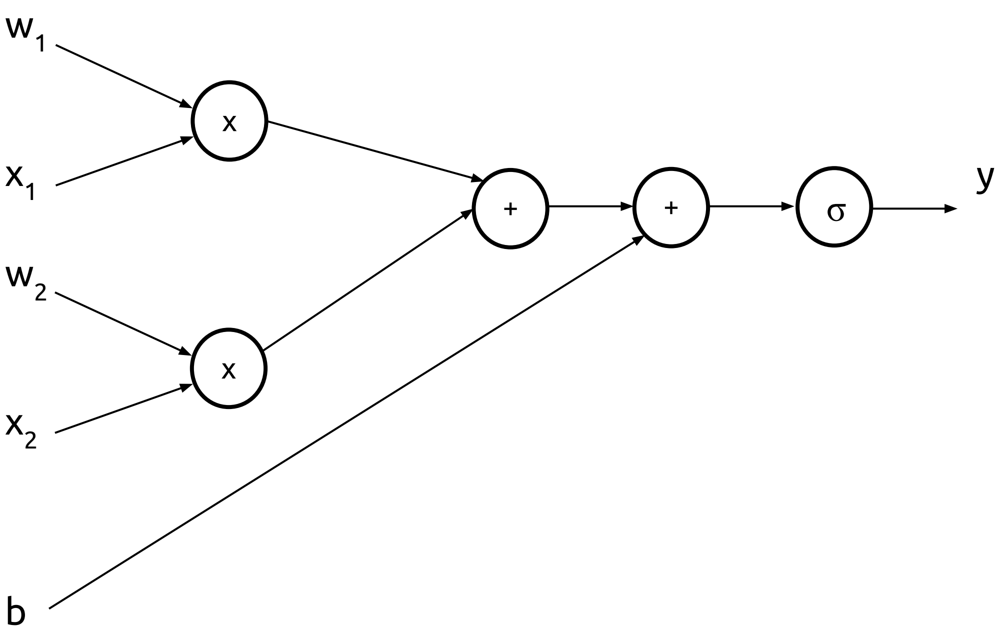
$$\delta_1 = \frac{\partial \hat{y}}{\partial x_1} = \frac{\partial \hat{y}}{\partial x_2}\frac{\partial x_2}{\partial x_1} = \delta_2 g_1'(x_1)$$

Backward pass

Gradients from composition (chain rule)

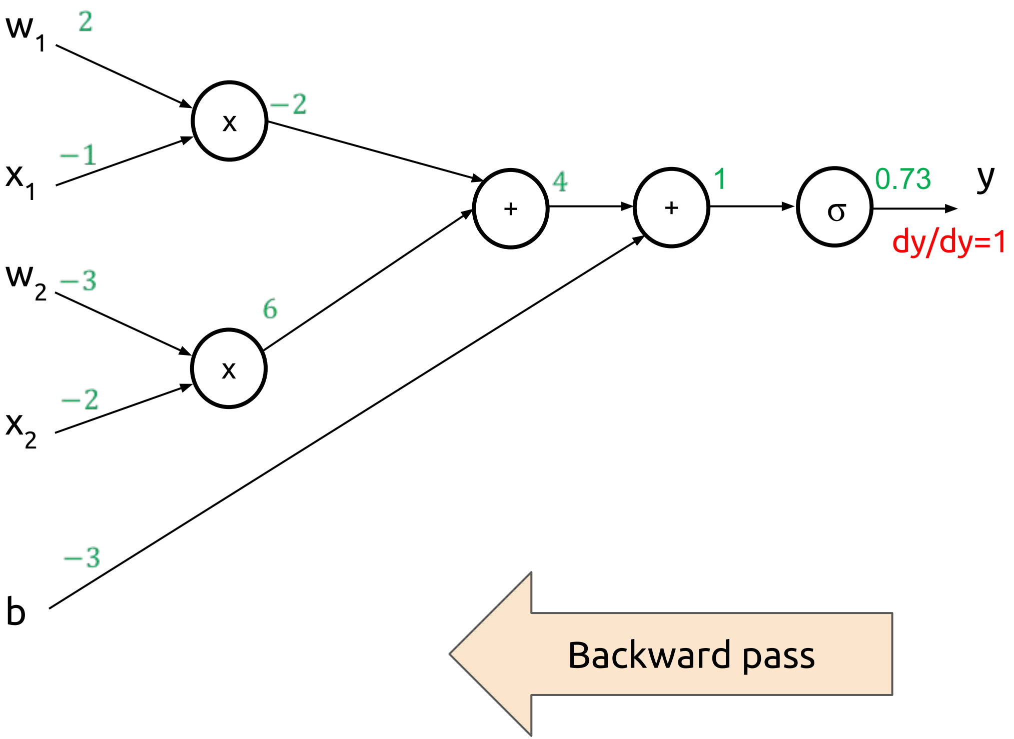
<u>Question</u>: How can gradients be backpropagated for the operations involved in a perceptron ?

- PRODUCT (x)
- SUM (+)
- SIGMOID (σ)



Gradient backpropagation in a perceptron

We can now estimate the sensitivity of the output y with respect to each input parameter w_i and x_i.



Example extracted from [Andrej Karpathy's notes](#) for CS231n from Stanford University.

Gradient weights for sigmoid σ

(*)

$$\frac{\partial \sigma(x)}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{1+e^{-x}}\right) = \frac{-1}{(1+e^{-x})^2}\frac{\partial(1+e^{-x})}{\partial x} = \frac{-1}{(1+e^{-x})^2}\frac{\partial(e^{-x})}{\partial x}$$

(*) $f(x) = \frac{g(x)}{h(x)} \qquad f'(x) = \frac{g'(x)h(x) - g(x)h'(x)}{h^2}$

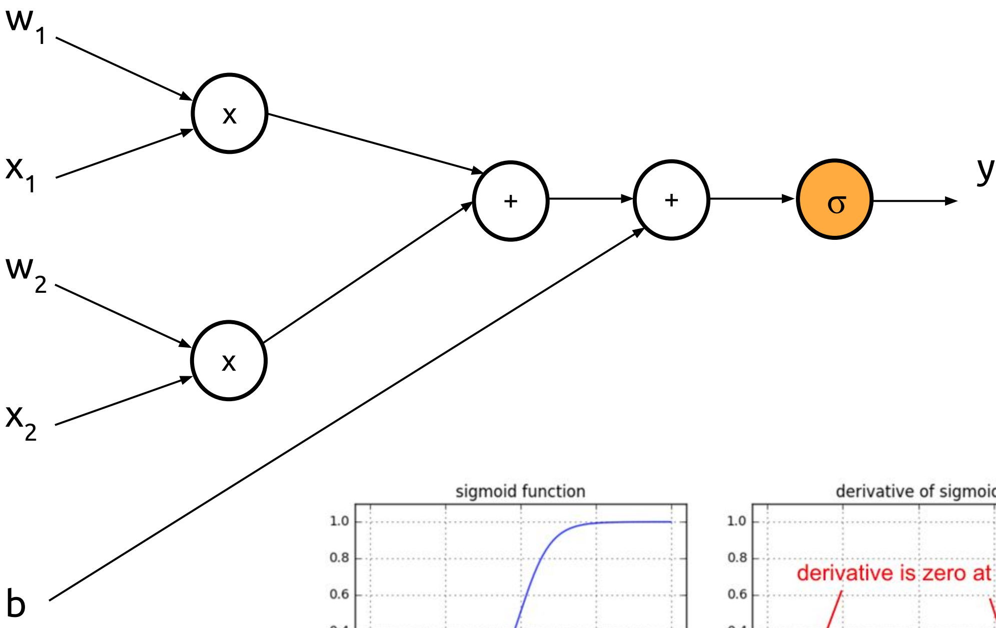
$$\frac{\partial \sigma(x)}{\partial x} = \frac{-1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

...which can be re-arranged as...

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})}\frac{1}{(1+e^{-x})}$$

$$\frac{\partial \sigma(x)}{\partial x} = \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)\sigma(x)$$

$$\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\,\sigma(x)$$



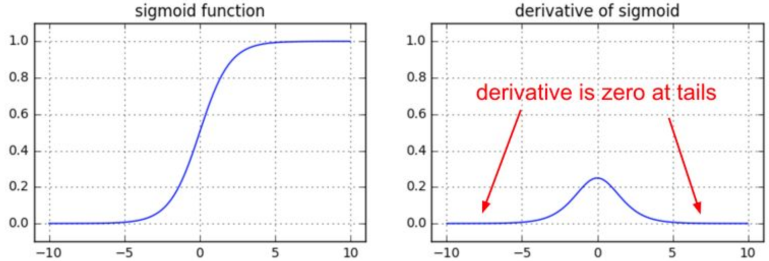


Figure: Andrej Karpathy

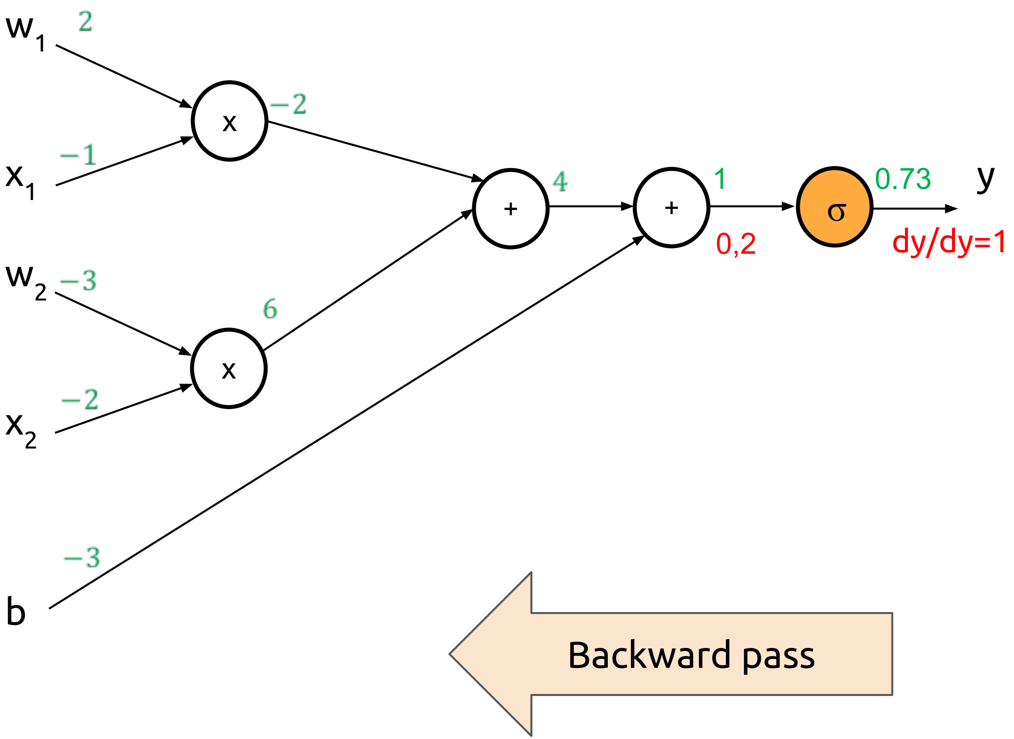
Even more details: Arunava, "Derivative of the Sigmoid function" (2018)

Gradient backpropagation in a perceptron

$$\frac{\partial \sigma(x)}{\partial x} = (1 - \sigma(x))\, \sigma(x)$$

```python
import math

dot=1

# sigmoid function
f = 1.0 / (1 + math.exp(-dot))

# gradient on dot variable,
ddot = (1 - f) * f

print(ddot)
```

```
0.19661193324148185
```



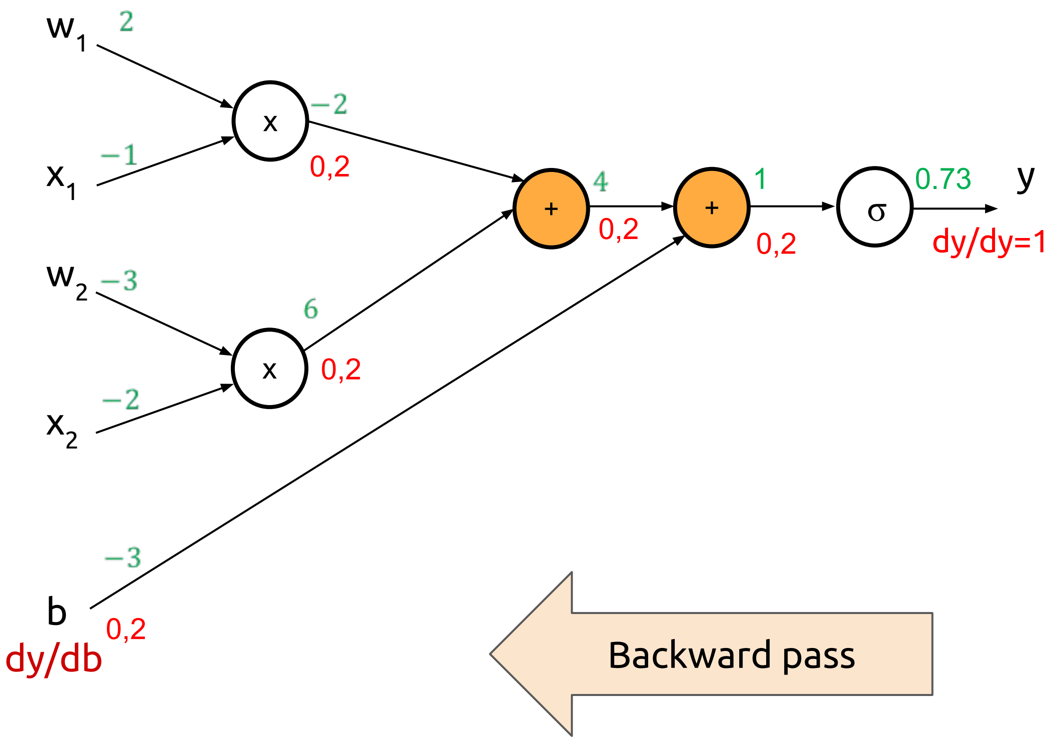
Example extracted from [Andrej Karpathy's notes](#) for CS231n from Stanford University.

Gradient backpropagation in a perceptron

Sum: Distributes the gradient to both branches.

$$\frac{\partial(a+b)}{\partial a} = 1$$

$$\frac{\partial(a+b)}{\partial b} = 1$$



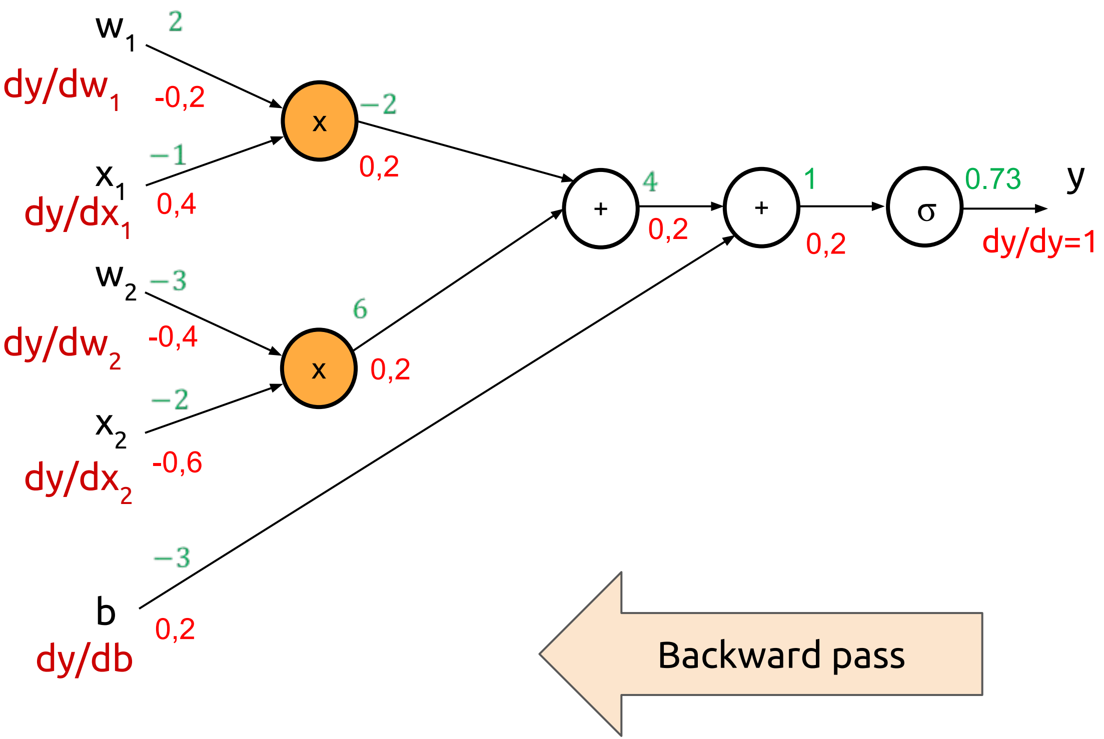
Example extracted from Andrej Karpathy's notes for CS231n from Stanford University.

Gradient backpropagation in a perceptron

Product: Switches gradient weight values.

$$\frac{\partial(a \cdot b)}{\partial a} = b$$

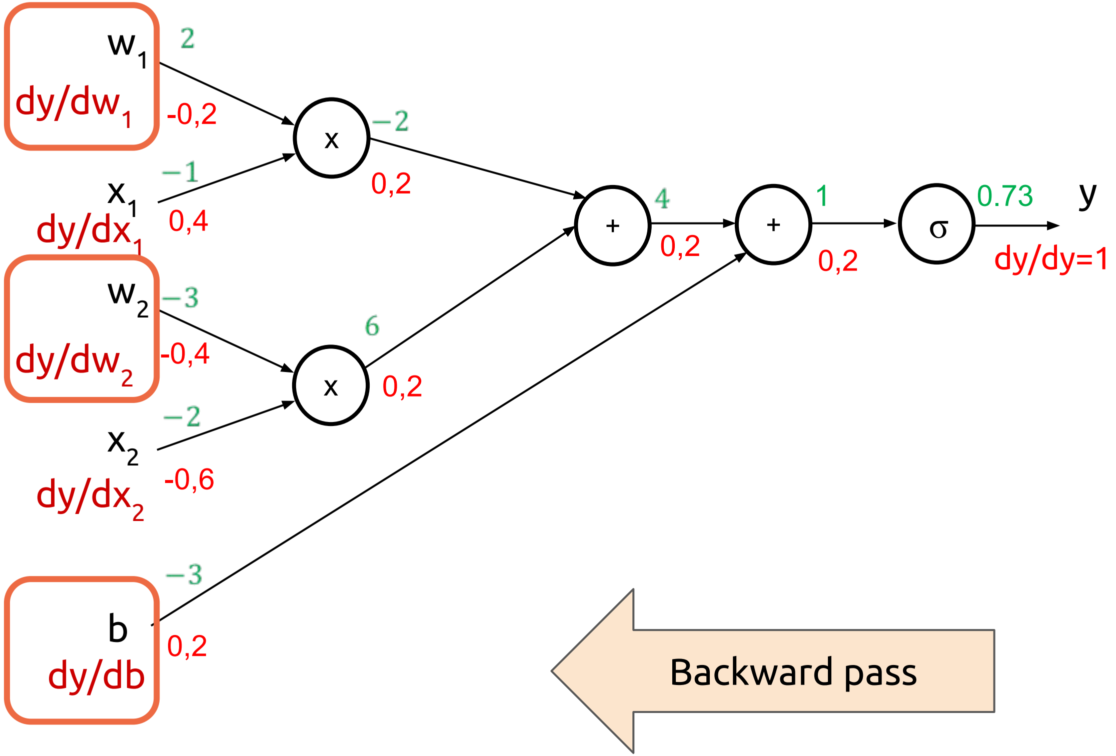
$$\frac{\partial(a \cdot b)}{\partial b} = a$$



Example extracted from [Andrej Karpathy's notes](#) for CS231n from Stanford University.

Gradient backpropagation in a perceptron

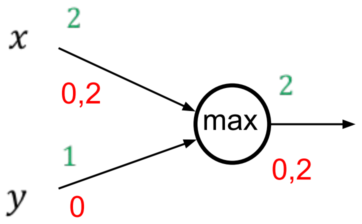
Normally, we will be interested only on the weights (w_i) and biases (b), not the inputs (x_i). The weights are the parameters to learn in our models.



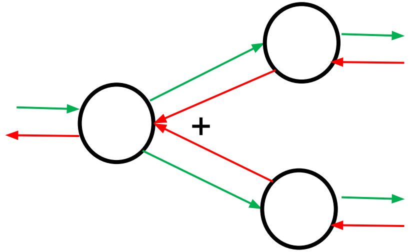
Example extracted from [Andrej Karpathy's notes](#) for CS231n from Stanford University.

(bonus) Gradients weights for MAX & SPLIT

Max: Routes the gradient only to the higher input branch (not sensitive to the lower branches).

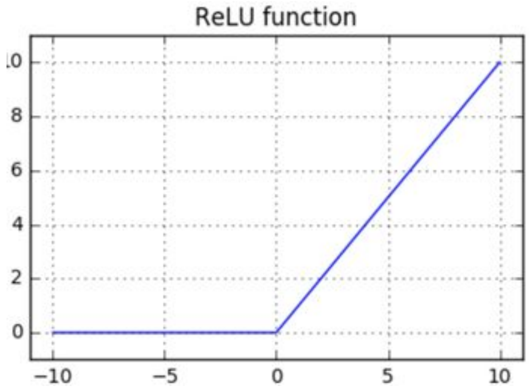


Split: Branches that split in the forward pass and merge in the backward pass, add gradients

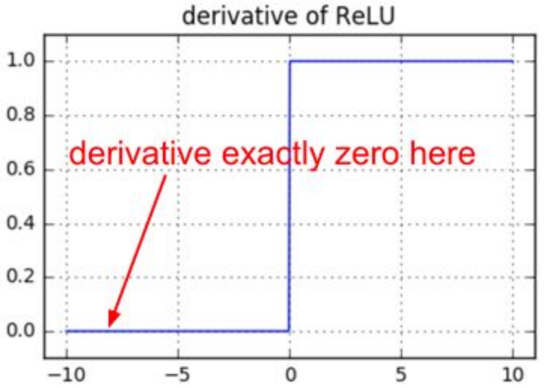


(bonus) Gradient weights for ReLU

$$ReLU(x) = \left\{ \begin{array}{ll} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{array} \right\}$$



$$\frac{\partial ReLU(x)}{\partial x} = u(x) = \left\{ \begin{array}{ll} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{array} \right\}$$



Figures: Andrej Karpathy

Learn more

READ

- Chris Olah, [“Calculus on Computational Graphs: Backpropagation”](#) (2015).
- Andrej Karpathy,, [“Yes, you should understand backprop”](#) (2016), and his [“course notes](#) at Stanford University CS231n.

WATCH

- Gilbert Strang, [“27. Backpropagation: Find Partial Derivatives”](#). MIT 18.065 (2018)

Undergradese

What undergrads ask vs. what they're REALLY asking