http://bit.ly/idl2020

Day 2 Lecture 1

Backpropagation

Xavier Giro-i-Nieto

@DocXavi
xavier.giro@upc.edu

Associate Professor
Universitat Politècnica de Catalunya

Acknowledgements

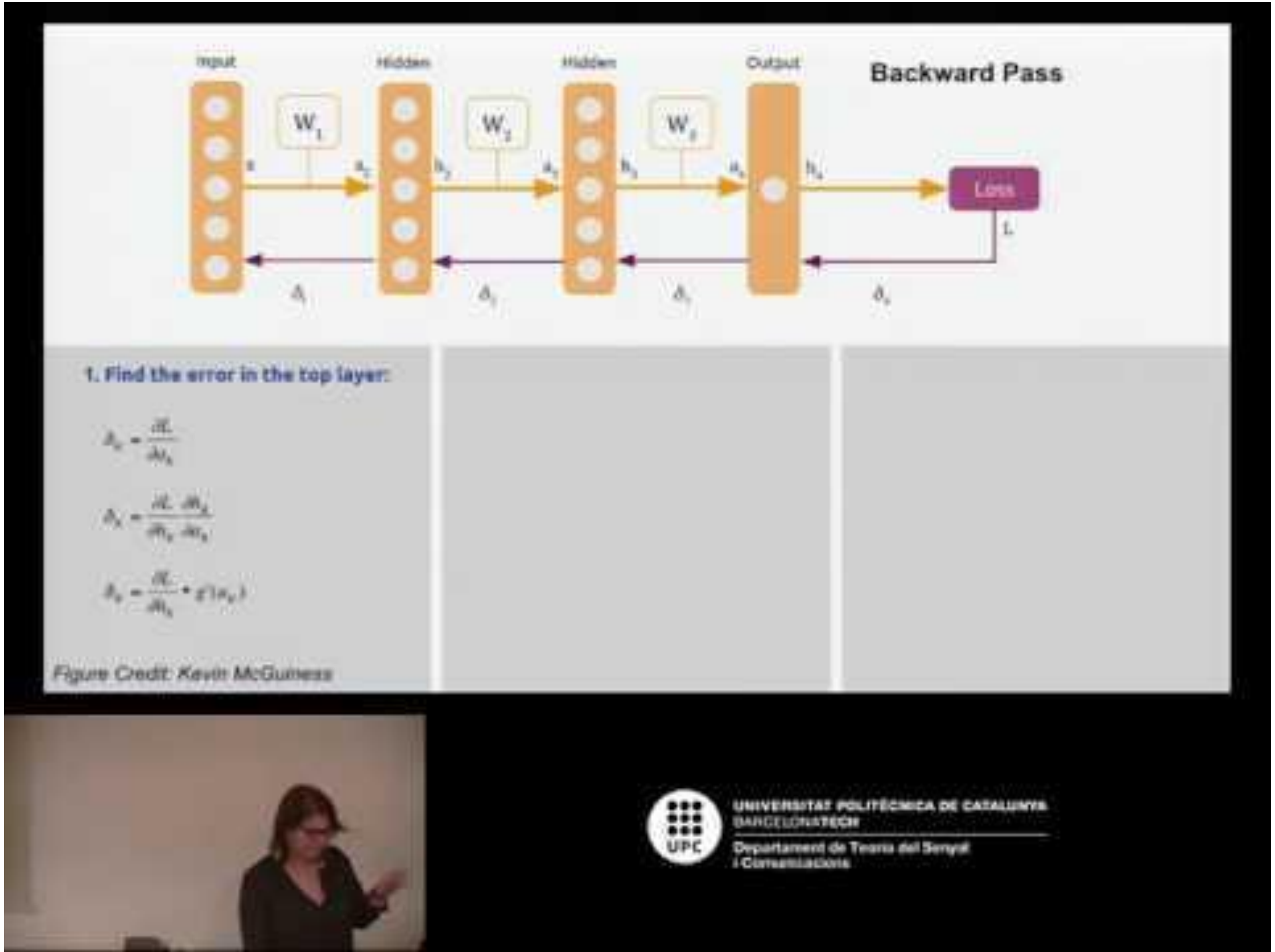
Kevin McGuinness
kevin.mcguinness@dcu.ie

Research Fellow
Insight Centre for Data Analytics
Dublin City University

Elisa Sayrol
elisa.sayrol@upc.edu

Associate Professor
ETSETB TelecomBCN
Universitat Politècnica de Catalunya

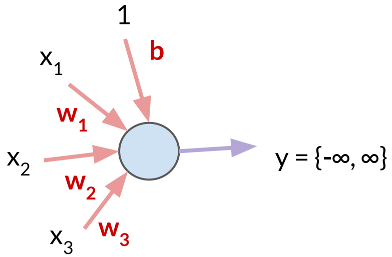
Video lecture



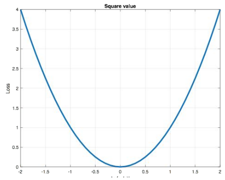
Loss function - L(y, ŷ)

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

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



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



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

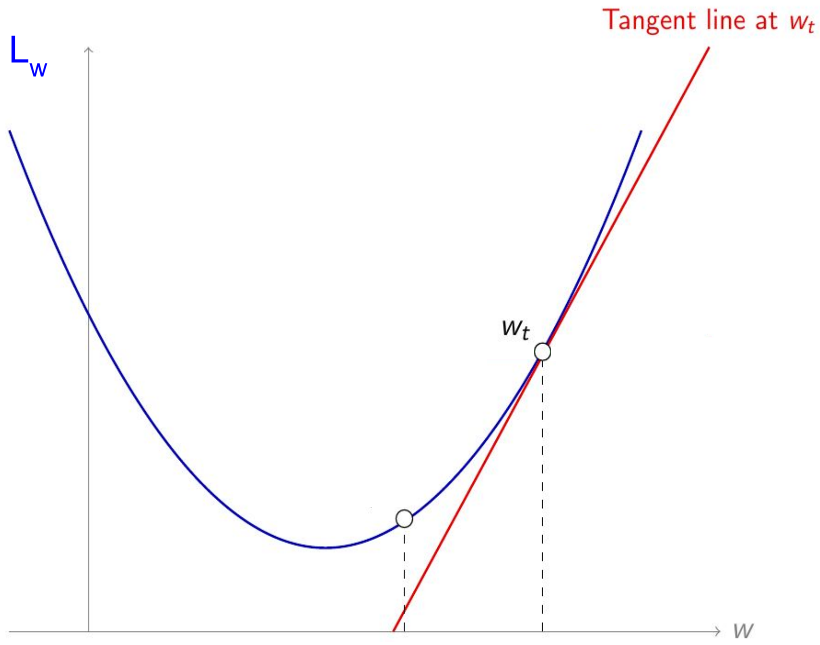
Loss function - $L(y, \hat{y})$

Discussion: Consider the single-parameter model...

$$\hat{y} = x \cdot w$$

.......and that, given a pair $(y, \hat{y})$, 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)

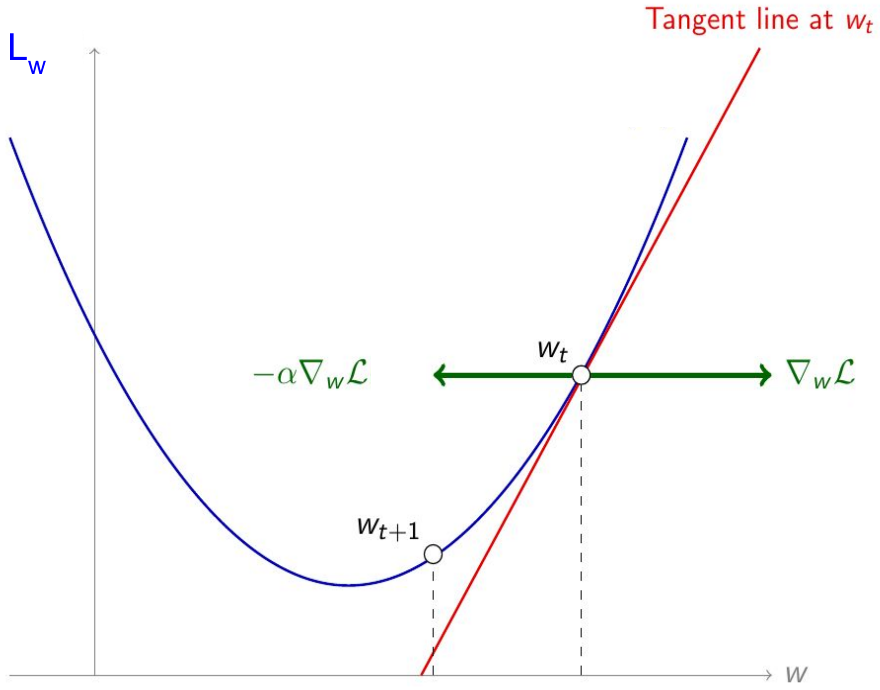
<u>Motivation for this lecture</u>:

if we had a way to estimate the gradient of the loss (∇L)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 & b**) - final goal during training
- input/intermediate data - visualization & interpretability purposes.

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

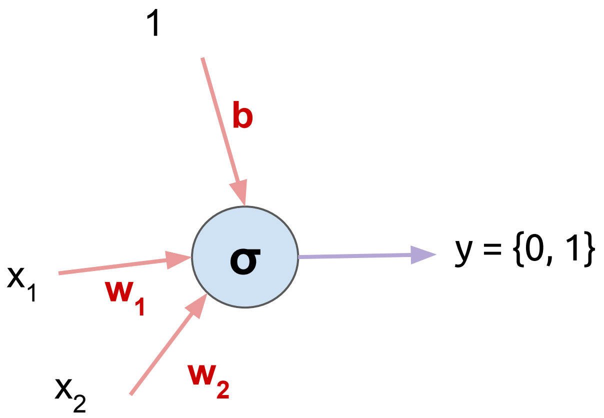
Let the Gradient Flo

Celebrate NIPS 2017 with Intel AI

Join us for an exclusive party – and a surprise reveal.

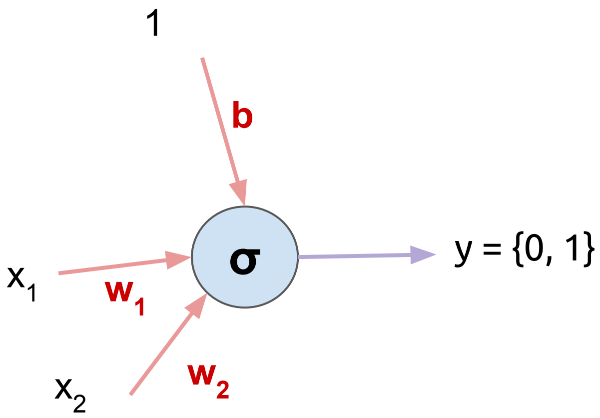
Giveaways, buskers, acrobats, DJ Nostalgia B and a special performance by Flo Rida!

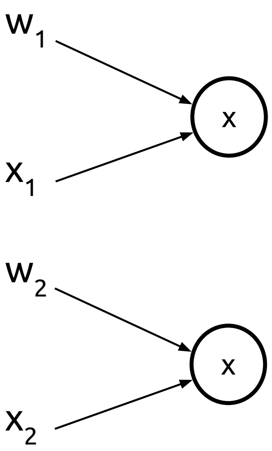
Computational graph of a simple perceptron



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

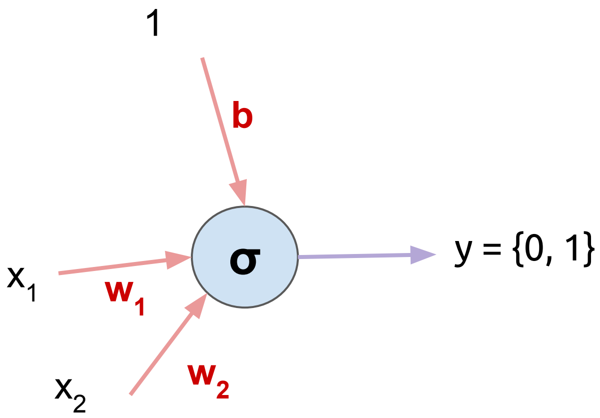
Computational graph of a perceptron

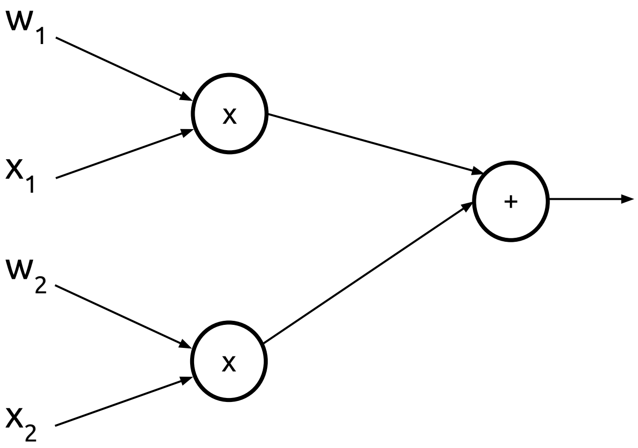




Example adapted from "CS231n: Convolutional Neural Networks for Visual Recognition", Stanford University.

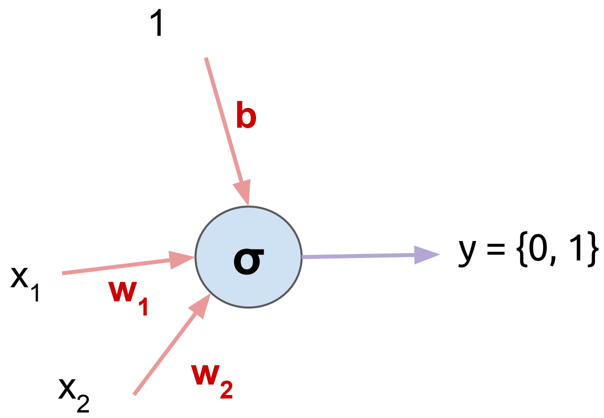
Computational graph of a perceptron

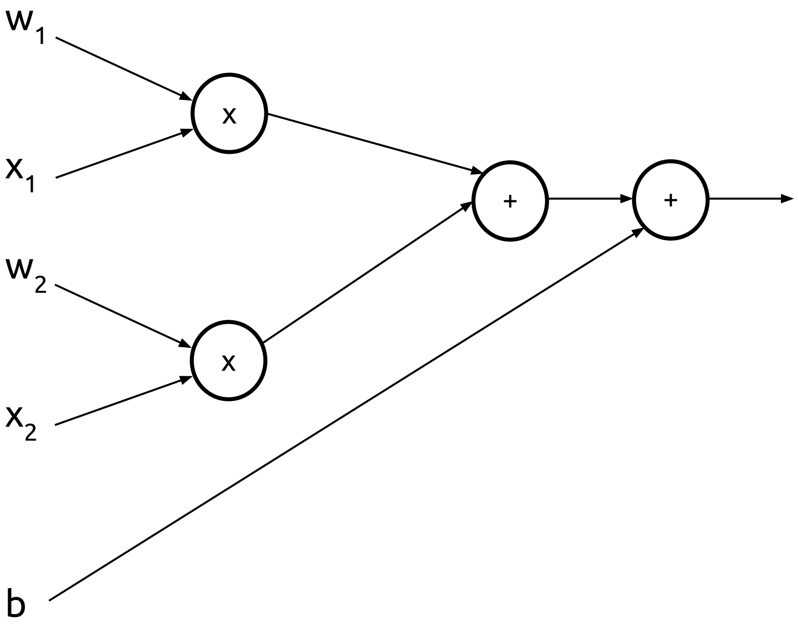




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

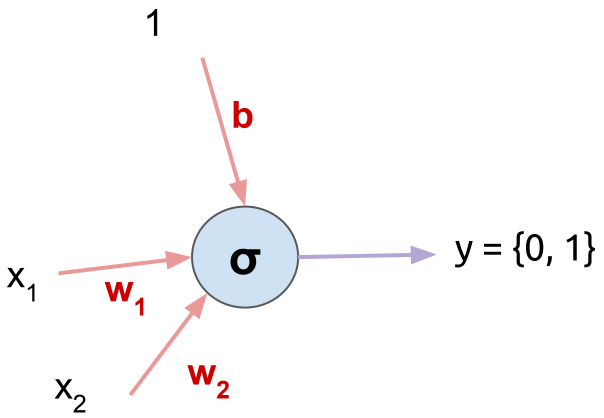
Computational graph of a perceptron

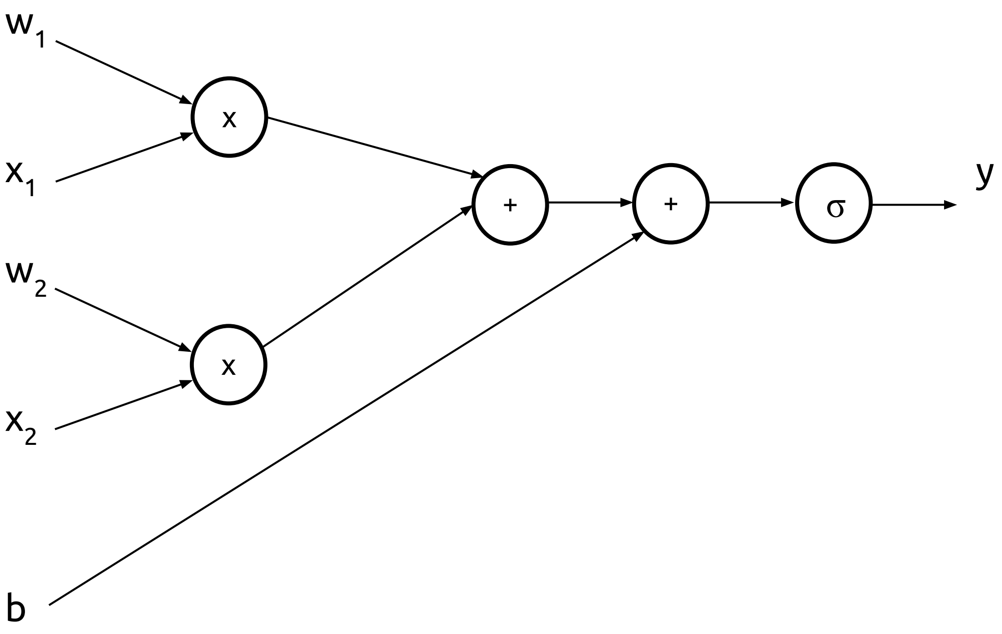




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

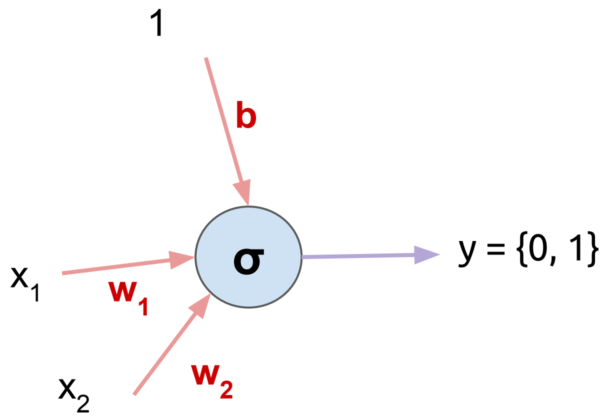
Computational graph of a perceptron

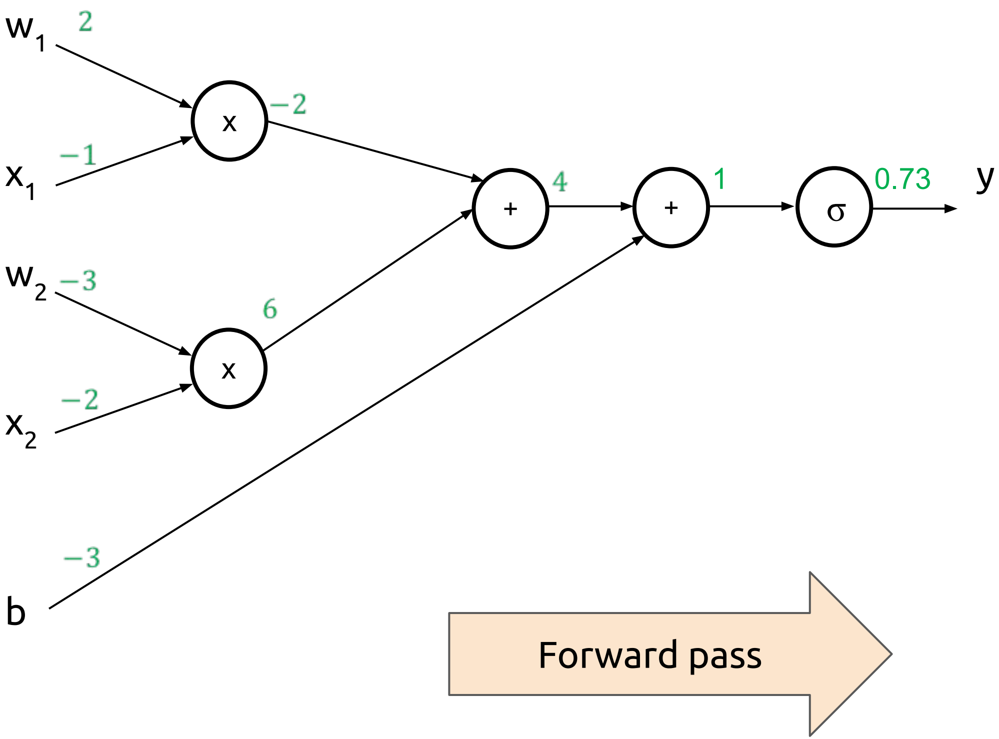




Example adapted from [“CS231n: Convolutional Neural Networks for Visual Recognition”](), Stanford University.

Computational graph of a perceptron





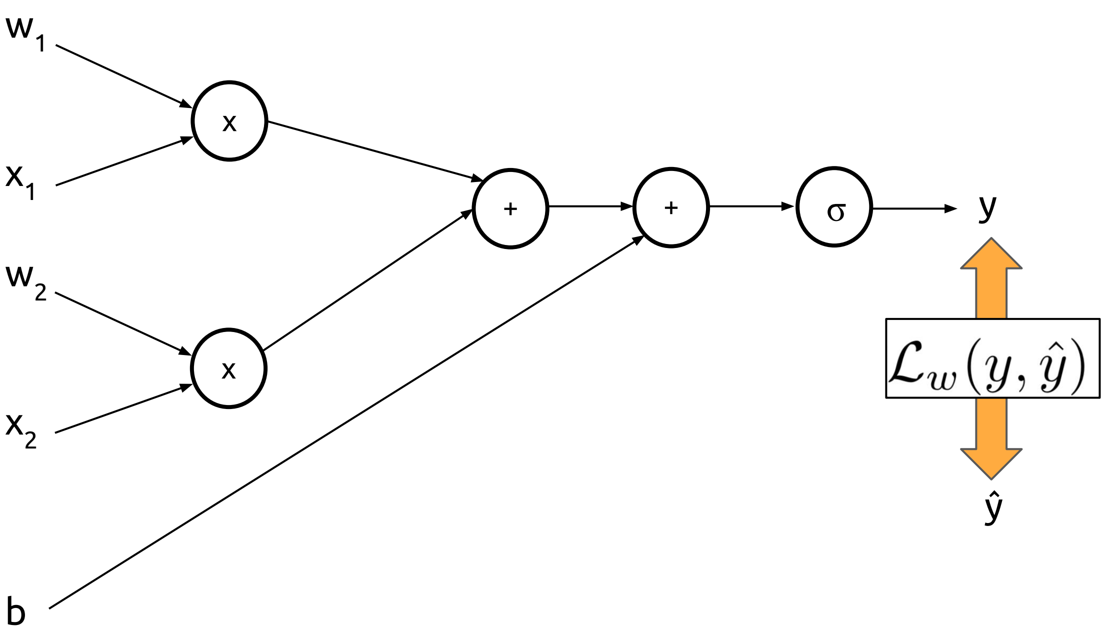
Computational graph of a perceptron

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

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

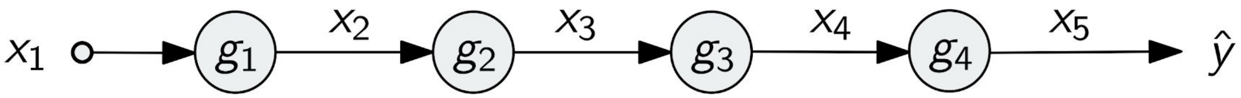
$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_2} = ?$$

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



Gradients from composition (chain rule)

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



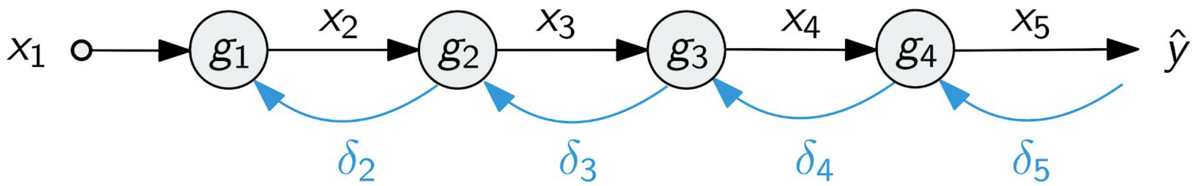
Decompose into steps (**forward propagation**):

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

Forward pass

Gradients from composition (chain rule)

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



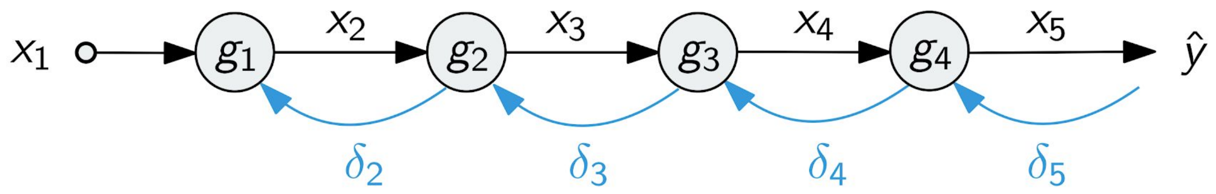
Want to find $\frac{\partial \hat{y}}{\partial x_1}$. Chain rule:

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

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

Backward pass

Gradients from composition (chain rule)

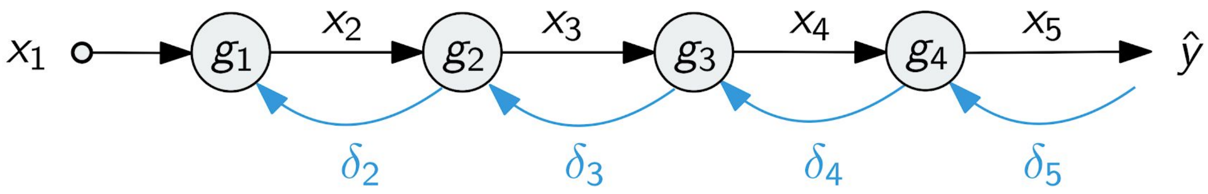


Decompose into steps again. Let $\delta_k = \frac{\partial \hat{y}}{\partial x_k}$. **Backpropagation**:

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

A variation in x_5 directly affects on ŷ with a 1:1 factor.

Gradients from composition (chain rule)



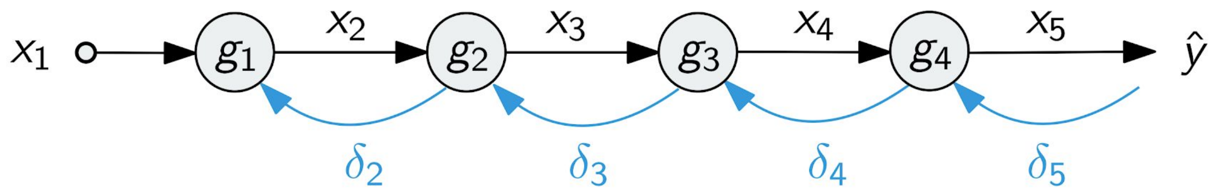
Decompose into steps again. Let $\delta_k = \frac{\partial \hat{y}}{\partial x_k}$. **Backpropagation**:

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

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

How does a variation on x_4 affect the predicted $\hat{y}$?

Gradients from composition (chain rule)



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

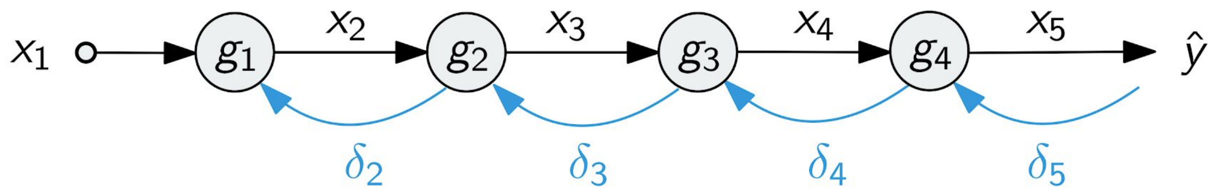
$$\delta_5 = \frac{\partial \hat{y}}{\partial x_5} = 1$$

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

How does a variation on x_4 affect the predicted $\hat{y}$?

It corresponds to how a variation of x_5 affects $\hat{y}$...

Gradients from composition (chain rule)



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

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

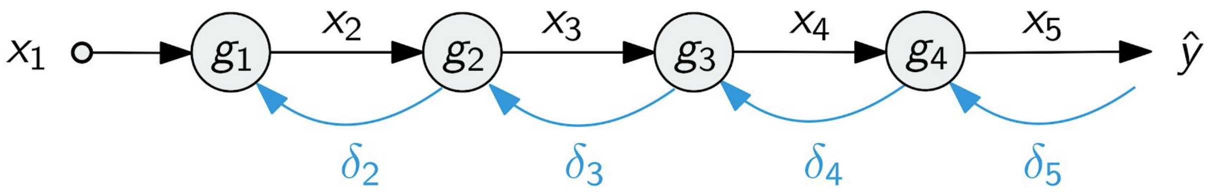
$$\delta_4 = \frac{\partial \hat{y}}{\partial x_4} = \frac{\partial \hat{y}}{\partial x_5}\frac{\partial x_5}{\partial x_4} = \delta_5 g_4'(x_4)$$

How does a variation on x_4 affect the predicted $\hat{y}$?

It corresponds to how a variation of x_5 affects $\hat{y}$ ...

...<u>**multiplied**</u> by how a variation near the input x_4 affects the output $g_4(x_4)$.

Gradients from composition (chain rule)



The same reasoning can be iteratively applied until reaching $\dfrac{\partial \hat{y}}{\partial x_1}$:

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

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

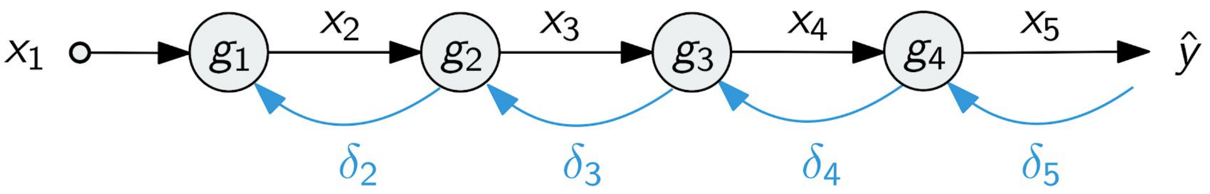
$$\delta_3 = \frac{\partial \hat{y}}{\partial x_3} = \frac{\partial \hat{y}}{\partial x_4}\frac{\partial x_4}{\partial x_3} = \delta_4 g_3'(x_3)$$

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

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

Backward pass

Gradients from composition (chain rule)



In order to compute $\delta_k = \frac{\partial \hat{y}}{\partial x_k}$, we must:

1) Find the derivative function → $g'_i(\cdot)$
2) Evaluate $g'_i(\cdot)$ at x_i → $g'_i(x_i)$
3) Multiply $g'_i(x_i)$ with the backpropagated gradient (δ_k).

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

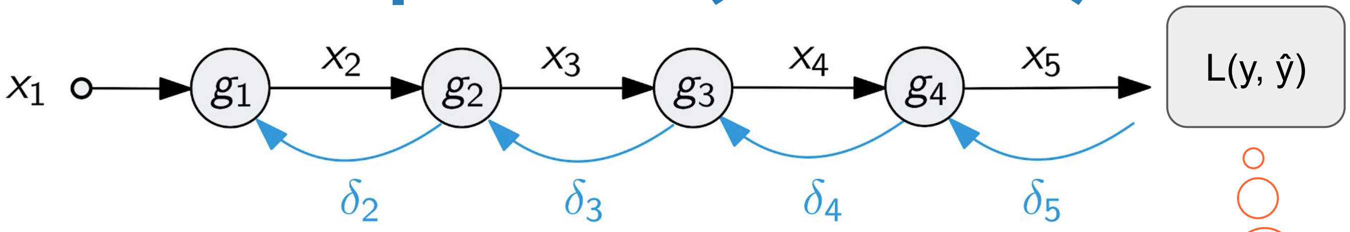
$$\delta_4 = \frac{\partial \hat{y}}{\partial x_4} = \frac{\partial \hat{y}}{\partial x_5}\frac{\partial x_5}{\partial x_4} = \delta_5 g_4'(x_4)$$

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

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

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

Gradients from composition (chain rule)



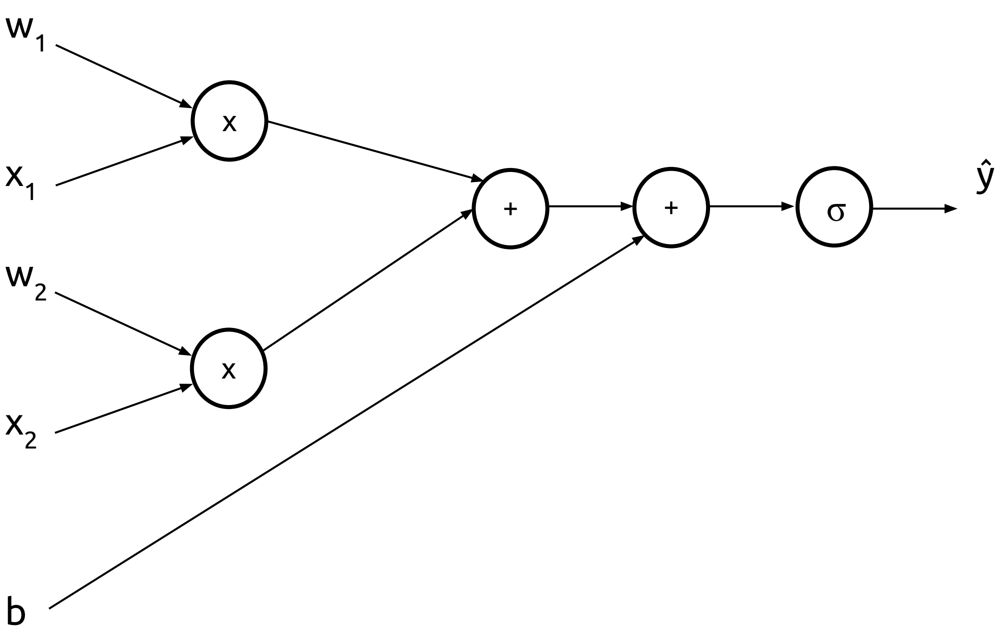
When training NN, we will actually compute the derivative over the loss function, not over the predicted value ŷ.

Backward pass

Gradients from composition (chain rule)

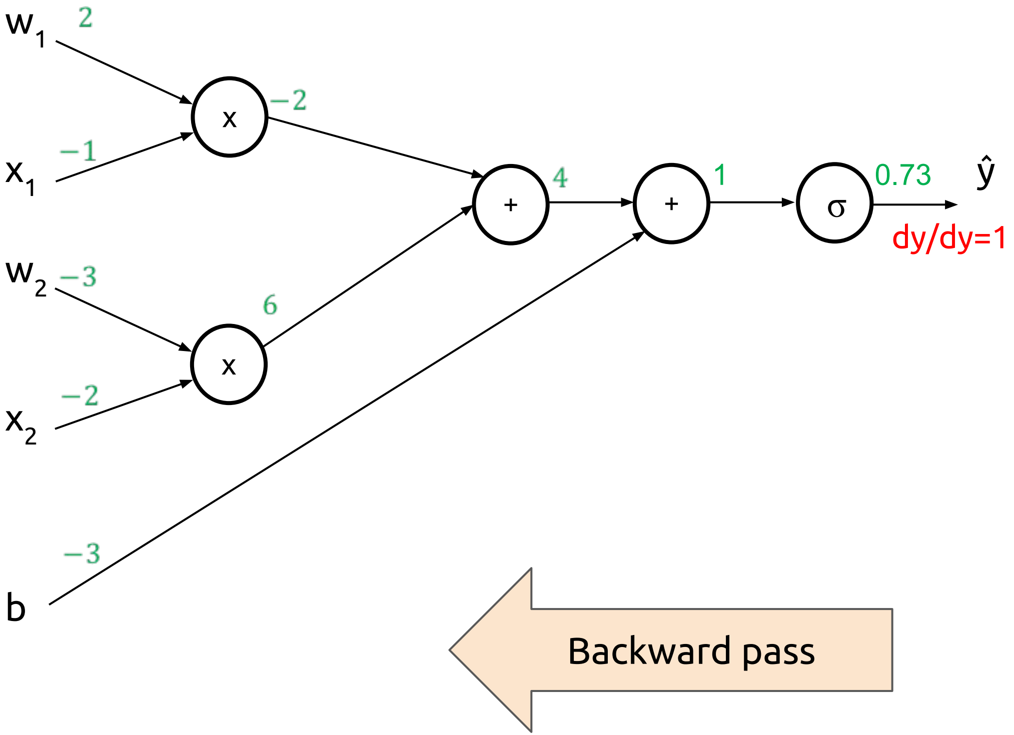
Question: What are the derivatives of the function involved in the computational graph of a perceptron ?

- SIGMOID (σ)
- SUM (+)
- PRODUCT (x)



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

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

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

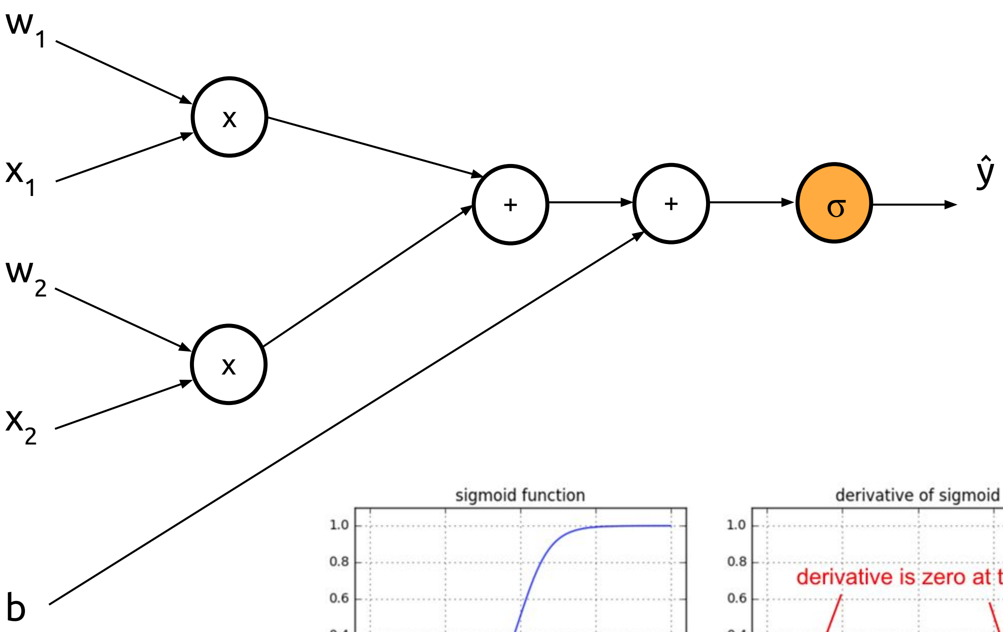
$$\frac{\partial \sigma(x)}{\partial x} = \frac{-1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

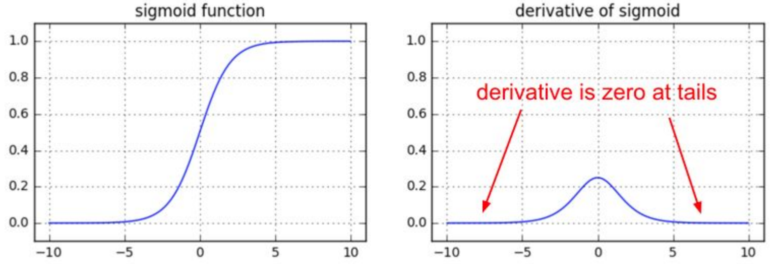
...which can be re-arranged as...

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

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

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





Even more details: Arunava, [“Derivative of the Sigmoid function”] (2018)

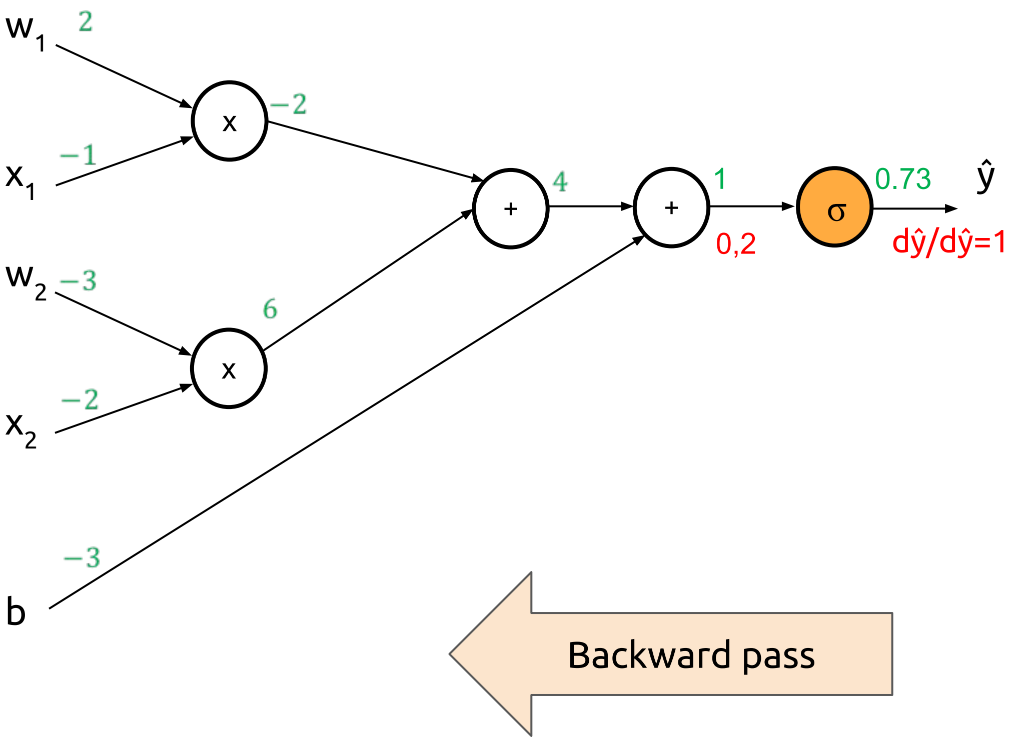
Figure: [Andrej Karpathy]

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
```



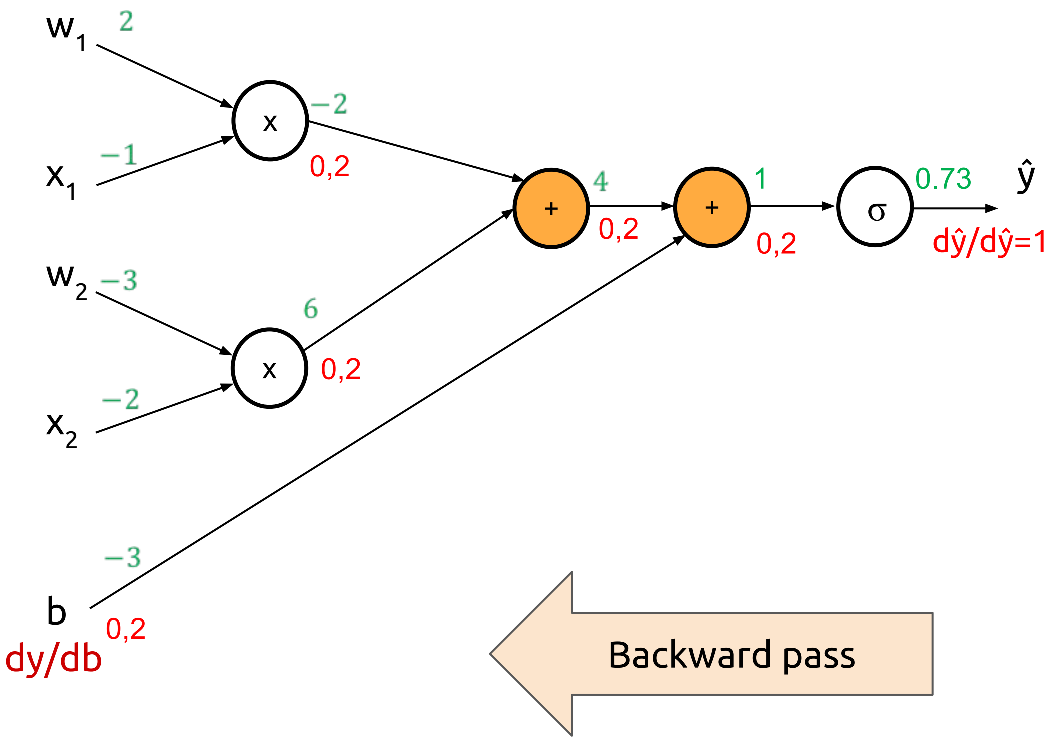
Example extracted from [Andrej Karpathy's notes]for CS231n from Stanford University.

Gradient backpropagation in a perceptron

Sum: Distributes the gradient to both branches.

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

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



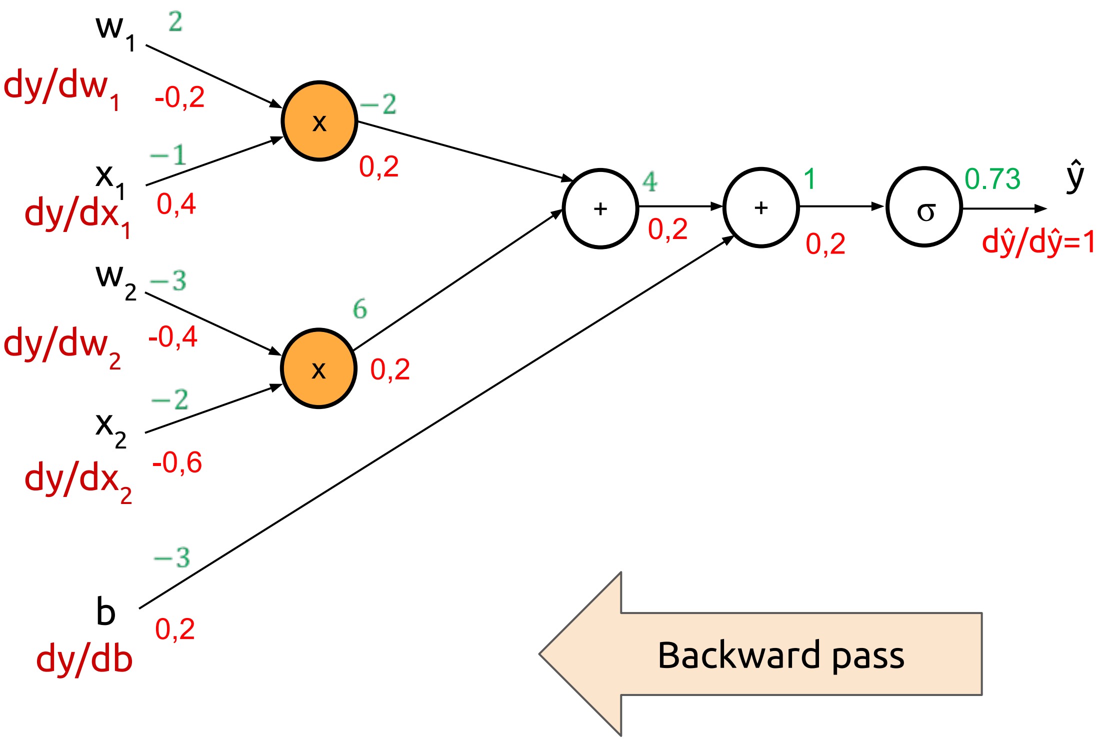
Example extracted from [Andrej Karpathy's notes]for CS231n from Stanford University.

Gradient backpropagation in a perceptron

Product: Switches gradient weight values.

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

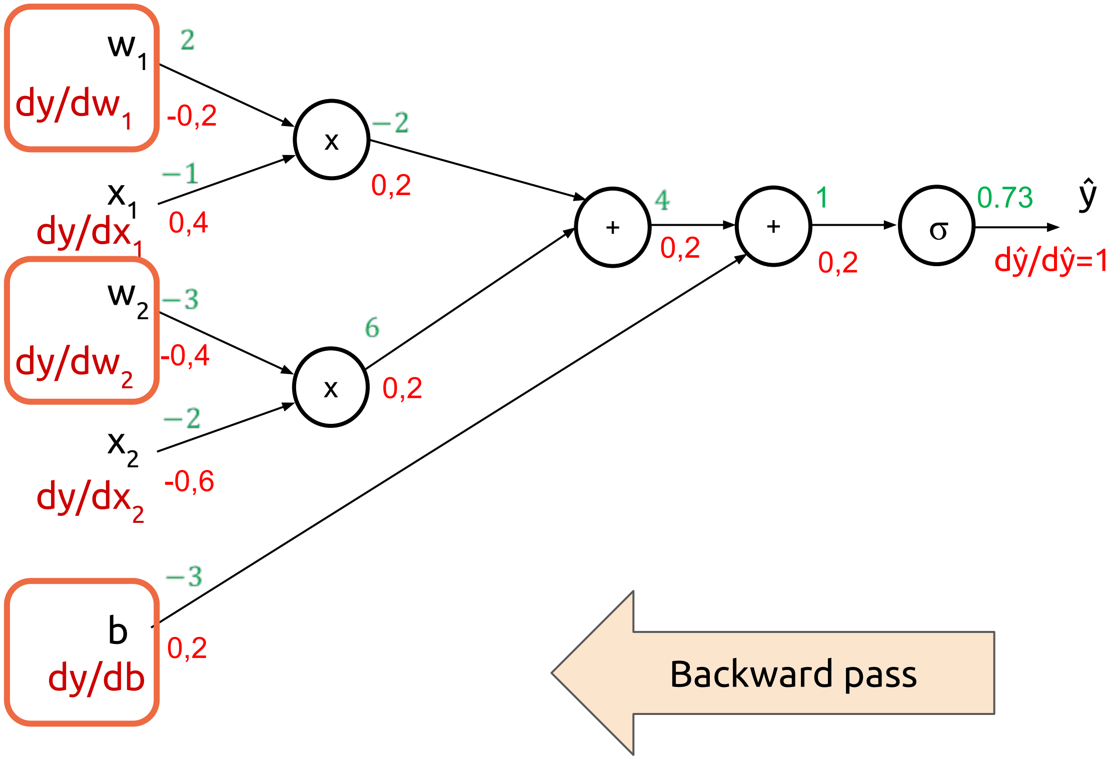
$$\frac{\partial(a \cdot b)}{\partial b} = a$$



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

Gradient backpropagation in a perceptron

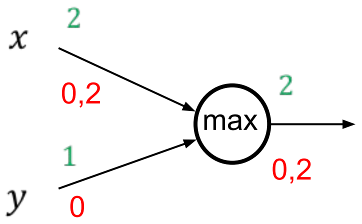
Normally, we will be interested only on the weights (w_i) and biases (b), not the inputs (x_i). The weights are the parameters to learn in our models.



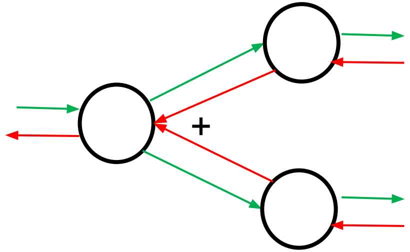
Example extracted from [Andrej Karpathy's notes](#) for CS231n from Stanford University.

(bonus) Gradients weights for MAX & SPLIT

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

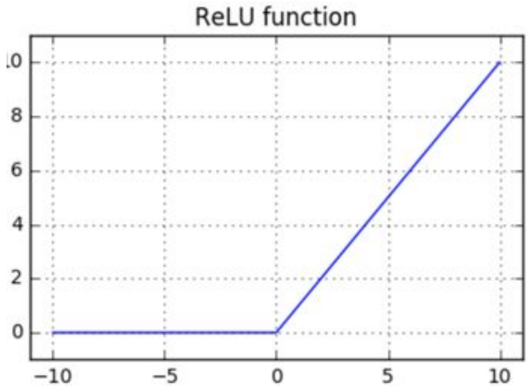


Split: Branches that split in the forward pass and merge in the backward pass, add gradients

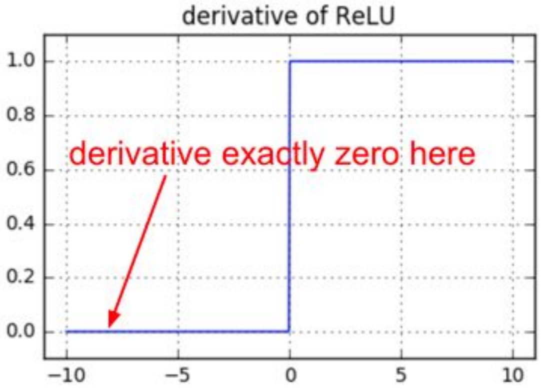


(bonus) Gradient weights for ReLU

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



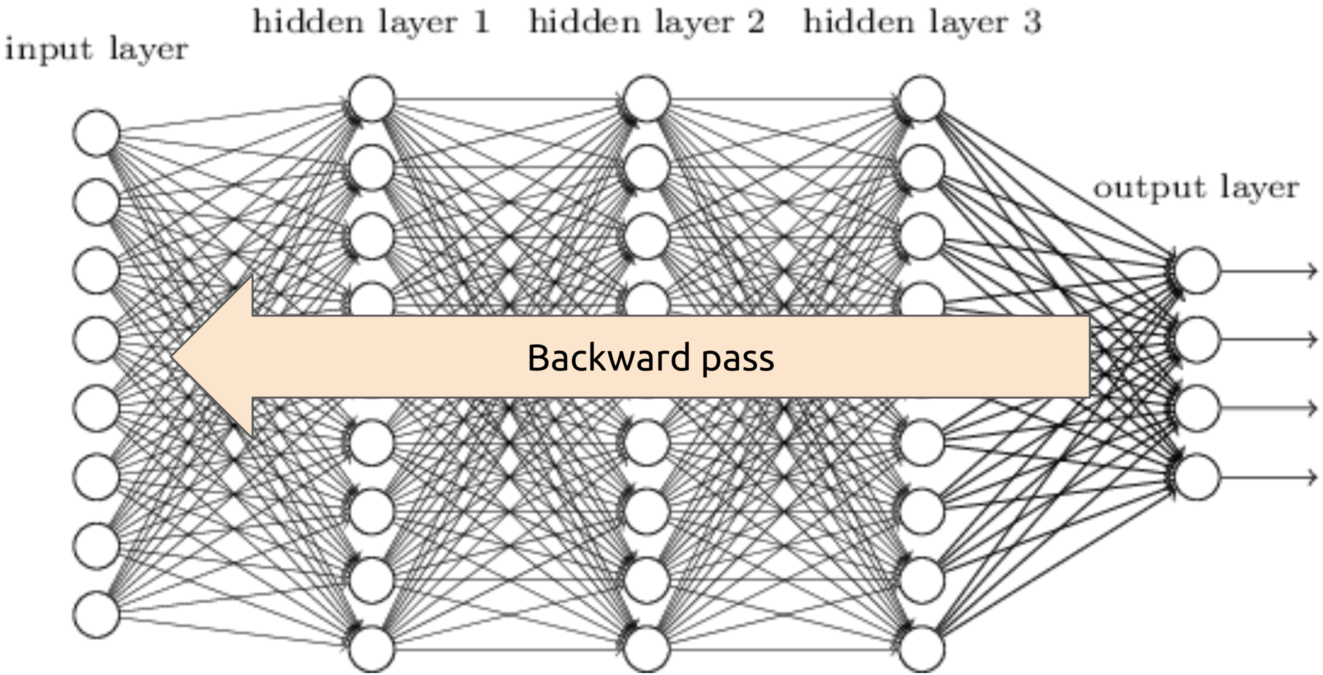
$$\frac{\partial ReLU(x)}{\partial x} = u(x) = \left\{ \begin{array}{ll} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{array} \right\}$$



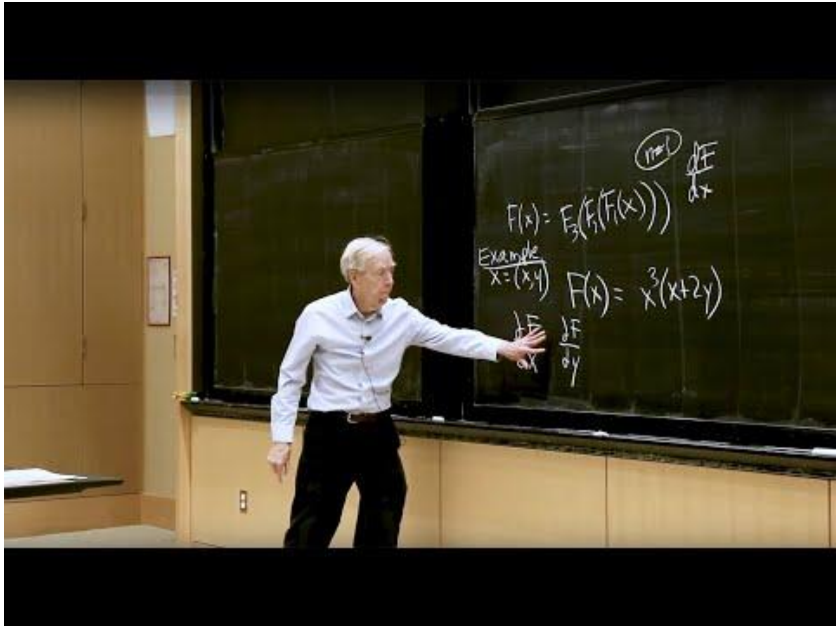
Figures: Andrej Karpathy

Backpropagation across layers

Gradients can flow across stacked layers of neurons to estimate their parameters.



Watch more



Gilbert Strang, "27. Backpropagation: Find Partial Derivatives". MIT 18.065 (2018)

Creative Commons, "Yoshua Bengio Extra Footage 1: Brainstorm with students" (2018)

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.

THREAD

Josh Gordon
@random_forests

What are the clearest explanations of backprop on the web? The two that I happily point students to are...

cs231n.github.io/optimization-2/
colah.github.io/posts/2015-08-...

Any others?

Tradueix el tuit
9:47 p. m. · 16 jul. 2019 · Twitter Web App

Advanced discussion

Greg Brockman
@gdb

Seguint

For differentiable problems, there's backpropagation. For everything else, there's RL.

Tradueix el tuit

18:11 - 31 de gen. de 2019

Yann LeCun
@ylecun

Seguint

En resposta a @gdb

Not quite right.
A more accurate statement would be "for everything else, there is gradient-free (zeroth-order) optimization."
RL is when there is a sequential decision process and what you see depends on previous actions you took.

Tradueix el tuit

2:38 - 1 de febr. de 2019

Problem

Consider a perceptron with a ReLU as activation function designed to process a single-dimensional inputs x.

a) Draw the computational graph of the perceptron, drawing a circle around the parameters that need to be estimated during training.

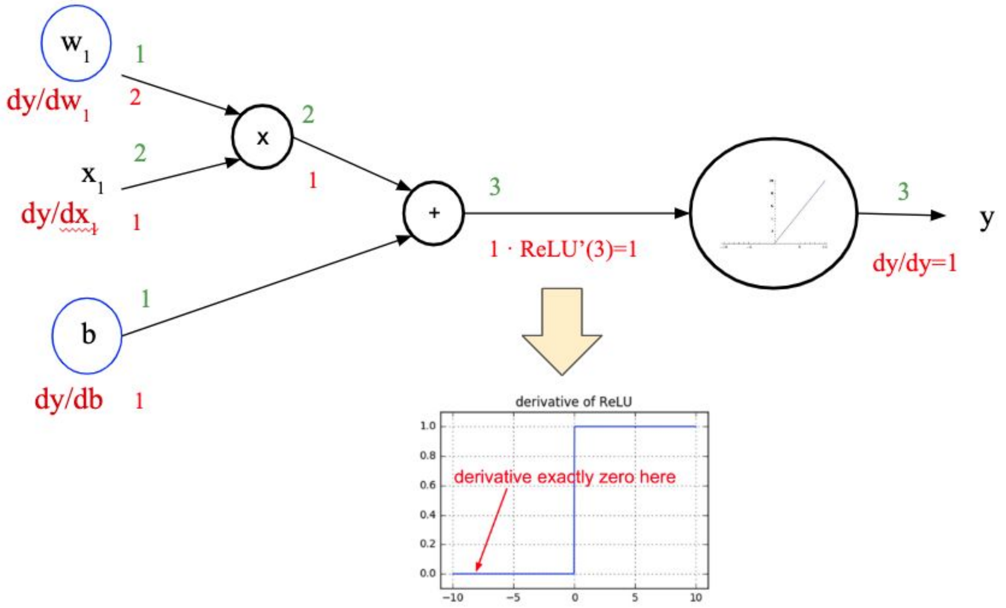
b) Compute the partial derivative of the output of the perceptron (y) with respect to each of its parameters for the input sample x=2. Consider that all the trainable parameters of the perceptron are initialized to 1.

c) Modify the results obtained in b) for the case in which all the trainable parameters of the perceptron are initialized to -1.

d) Briefly comment and compare the results obtained in b) and c).

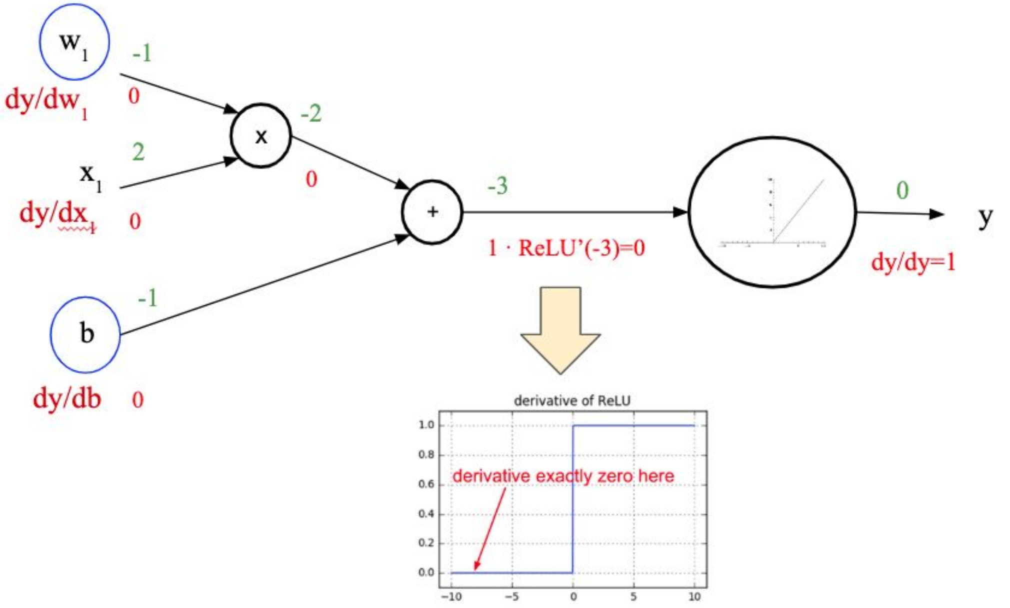
Problem (solved)

a) Draw the computational graph of the perceptron, drawing a circle around the parameters that need to be estimated during training.
b) b) Compute the partial derivative of the output of the perceptron (y) with respect to each of its parameters for the input sample x=2. Consider that all the trainable parameters of the perceptron are initialized to 1.



Problem (solved)

c) Modify the results obtained in b) for the case in which all the trainable parameters of the perceptron are initialized to -1.
d) Briefly comment and compare the results obtained in b) and c).



d) While in case b) the gradients can flow until the trainable parameters w_1 and b, in case c) gradients are "killed" by the ReLU.

Undergradese

What undergrads ask vs. what they're REALLY asking