Deep Learning Specialization

by DeepLearning.AI

Course #1: Neural Networks and Deep Learning

# 

# 

# 

[Course Site](https://www.coursera.org/specializations/deep-learning)

Made By: [Matias Borghi](https://mattborghi.github.io/)

# 

**Table of Contents**

[**Summary**](#_heading=h.88c3te42lqc7) **3**

[**Week 1 Introduction to deep learning**](#_heading=h.co7c7dt1rihg) **5**

[Introduction to Deep Learning](#_heading=h.ko7fcucpmufx) 5

[¿Qué es una red neuronal?](#_heading=h.2z85xhm5b1kk) 5

[Aprendizaje Supervisado con Redes Neuronales](#_heading=h.njskmwhpbqny) 6

[¿Por qué está despegando Deep Learning?](#_heading=h.2ggfdn98cag5) 8

[**Week 2 Neural Networks Basics**](#_heading=h.lc8c87anagjk) **9**

[Logistic Regression as a Neural Network](#_heading=h.nepbvc6dzrx) 9

[Fundamentos de la programación de redes neuronales](#_heading=h.unb6mh5tdk5l) 9

[Clasificación binaria](#_heading=h.xhfpjscme69c) 9

[Notación](#_heading=h.y3e7lvg0uxb) 9

[Regresión logística](#_heading=h.xwuj2krg88c2) 10

[Casos extremos](#_heading=h.k2elciizk1) 11

[Función de coste de la regresión logística](#_heading=h.5m7mzj7omyut) 11

[Gradiente descendente](#_heading=h.7w6b2548ed1q) 12

[¿Cómo se minimiza J(w,b)?](#_heading=h.onzc0bcnj0oo) 12

[Gráfico computacional](#_heading=h.3ky4q6bzhohh) 13

[Derivadas en un gráfico computacional](#_heading=h.a5g4h2jb0azw) 14

[Gradiente descendente en regresión logística](#_heading=h.fbjipdkvct4b) 14

[Gradiente descendente en m ejemplos](#_heading=h.9moxlx3nstht) 15

[Python and Vectorization](#_heading=h.k82ysvnf8867) 16

[Vectorización](#_heading=h.lzj3h4j71abi) 16

[Implementación vectorial del gradiente descendente en regresión logística](#_heading=h.x0s1raoh2ct1) 16

[Broadcasting en Python](#_heading=h.8bvwcy596uyz) 17

[Python/ Numpy vectors](#_heading=h.t186acsrq2x4) 17

[**Week 3: Shallow Neural Network**](#_heading=h.r4isw3l3zn6b) **19**

[Representación de Redes Neuronales](#_heading=h.kvh07uf6dop) 19

[¿Cómo se computan las salidas?](#_heading=h.w9ekyoyp5ux9) 19

[Introduciendo un poco de notación](#_heading=h.pdtuc5uvcz67) 21

[Explicación de la implementación vectorizada](#_heading=h.hnn18bgjh1f) 22

[Funciones de activación](#_heading=h.vtbj7trgngwz) 23

[Ventajas y desventajas de las funciones de activación](#_heading=h.df593j3k23yx) 24

[Derivadas de las funciones de activación](#_heading=h.b4x9mtm7qyi3) 24

[Gradiente descendente para redes neuronales](#_heading=h.gmleajxrq0yi) 25

[Intuición de la backward propagation](#_heading=h.ks40n26kgu1d) 27

[Inicialización parámetros](#_heading=h.yq83yfk2sfb4) 28

[Inicialización aleatoria](#_heading=h.lwjnjfvlvj2) 28

[**Week 4 Deep Neural Network**](#_heading=h.c851w1b08f8r) **30**

[Redes neuronales profundas de L-capas](#_heading=h.2wkcugqu9azh) 30

[Forward and Backward propagation in deep neural networks](#_heading=h.k73cq3eao83r) 31

[Resumen](#_heading=h.aohb0u4vje27) 32

[¿Por qué las redes neuronales profundas generalmente funcionan mejor que las que no lo son?](#_heading=h.6wrsae7dydvu) 33

[¿Qué son los hiper parámetros?](#_heading=h.28rac7dwzofl) 34

[Forward y Backward propagation](#_heading=h.8dk1ckxqgvd6) 35

# 

# 

# Summary

**Week 1 Introduction to deep learning**

**Be able to explain the major trends driving the rise of deep learning, and understand where and how it is applied today.**

**Learning Objectives**

* **Discuss the major trends driving the rise of deep learning.**
* **Explain how deep learning is applied to supervised learning**
* **List the major categories of models (CNNs, RNNs, etc.), and when they should be applied**
* **Assess appropriate use cases for deep learning**

**Week 2 Neural Networks Basics**

**Learn to set up a machine learning problem with a neural network mindset. Learn to use vectorization to speed up your models.**

**Learning Objectives**

* **Build a logistic regression model structured as a shallow neural network**
* **Build the general architecture of a learning algorithm, including parameter initialization, cost function and gradient calculation, and optimization implemetation (gradient descent)**
* **Implement computationally efficient and highly vectorized versions of models**
* **Compute derivatives for logistic regression, using a backpropagation mindset**
* **Use Numpy functions and Numpy matrix/vector operations**
* **Work with iPython Notebooks**
* **Implement vectorization across multiple training examples**

**Week 3 Shallow neural networks**

**Learn to build a neural network with one hidden layer, using forward propagation and backpropagation.**

**Learning Objectives**

* **Describe hidden units and hidden layers**
* **Use units with a non-linear activation function, such as tanh**
* **Implement forward and backward propagation**
* **Apply random initialization to your neural network**
* **Increase fluency in Deep Learning notations and Neural Network Representations**
* **Implement a 2-class classification neural network with a single hidden layer**

**Week 4 Deep Neural Networks**

**Understand the key computations underlying deep learning, use them to build and train deep neural networks, and apply it to computer vision.**

**Learning Objectives**

* **Describe the successive block structure of a deep neural network**
* **Build a deep L-layer neural network**
* **Analyze matrix and vector dimensions to check neural network implementations**
* **Use a cache to pass information from forward to back propagation**
* **Explain the role of hyperparameters in deep learning**

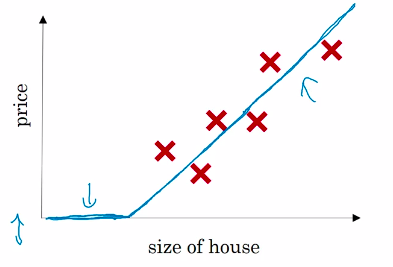
# 

# Week 1 Introduction to deep learning

## Introduction to Deep Learning

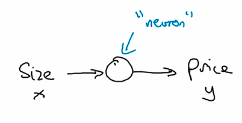
### ¿Qué es una red neuronal?

Predicción aplicado al precio de las casas.



Los datos parecen ajustarse con un ajuste lineal. Dado que los precios no pueden ser negativos podemos agregar una sección donde el ajuste sería cero.

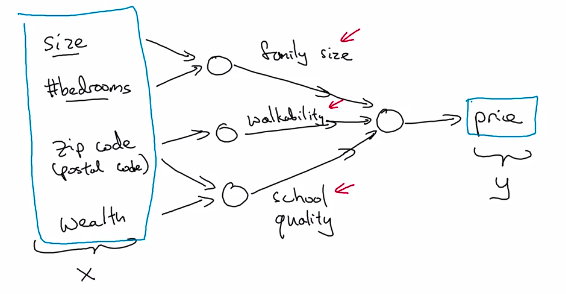
A esta función que predice los precios se la puede pensar como una red neuronal simple.



Este tipo de funciones que es cero y luego lineal es llamada RELU (Rectified Linear Unit).

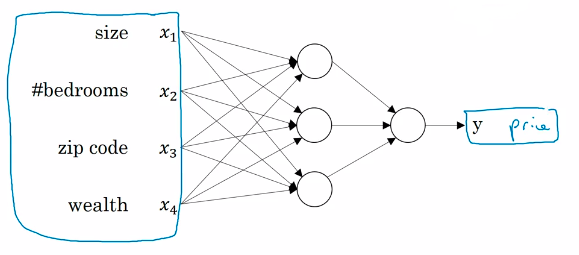
Una red neuronal más compleja se podría armar uniendo “neuronas” simples como la recién creada.

En lugar de crear una red neuronal que predice el precio de una casa en base al tamaño, también lo hace en función de otras características (features) como el número de habitaciones. También el tamaño de la familia (4 integrantes, 5 integrantes etc.).



Lo bueno es que, en un modelo de redes neuronales, no necesitamos conocer las características intermedias, sino que basta con determinar x e y.

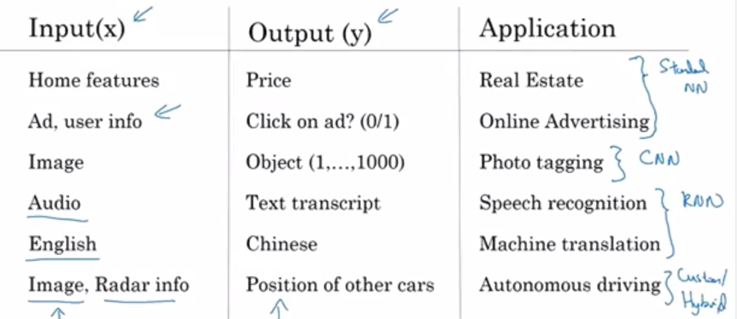
En la realidad la red neuronal se implementa de la siguiente manera:



donde las unidades ocultas se conectan con todas las unidades precedentes.

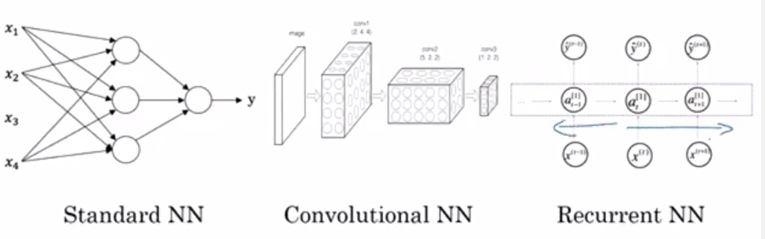
### Aprendizaje Supervisado con Redes Neuronales

Ejemplos de Aprendizaje supervisado se muestran en la tabla siguiente:

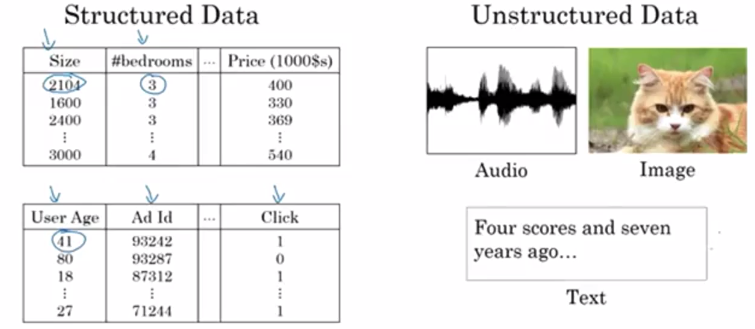


donde cada aplicación en particular requiere una red neuronal especifica ya sea CNN, RNN. híbridas, etc.

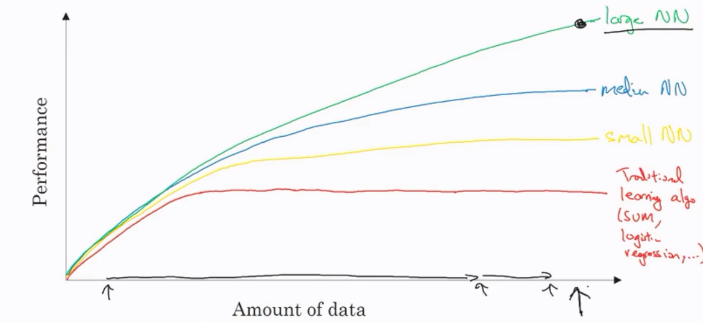
Un diagrama de estas redes neuronales podría ser el siguiente



**Aplicaciones de Machine Learning sobre datos estructurados o no estructurados**



### ¿Por qué está despegando Deep Learning?



La cantidad de información disponible ha aumentado considerablemente. Los algoritmos tradicionales de aprendizaje no mejoran su rendimiento dada la mejora de datos, cosa que las redes neuronales si lo hacen.

Para obtener un rendimiento alto habría que tener una red neuronal grande, con muchas unidades ocultas y mucha información.

# Week 2 Neural Networks Basics

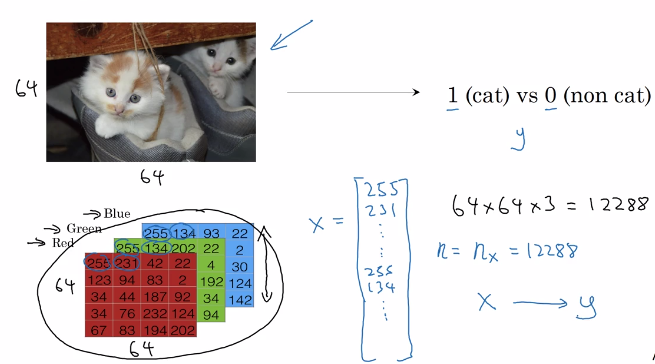
## Logistic Regression as a Neural Network

### Fundamentos de la programación de redes neuronales

### Clasificación binaria

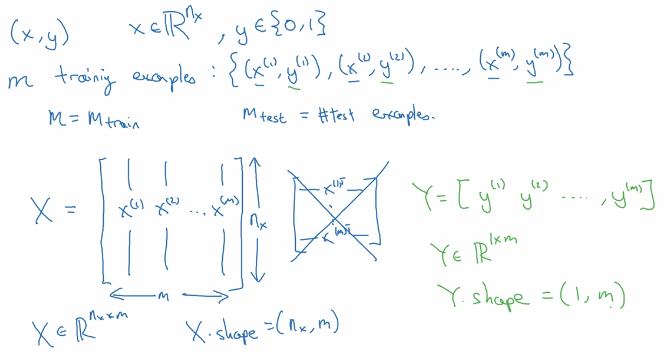
Un ejemplo sería determinar si, dada una foto, la misma pertenece a un gato o no.

Una foto color se puede representar como tres matrices RGB de, por ejemplo, 64x64. El vector de entrada (input feature) X tendrá como longitud 64x64x3 = 12288. La idea del algoritmo de clasificación binaria es dado este vector Y determinar si es gato (1) o no (0) .



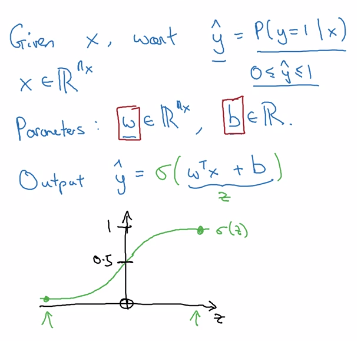
### Notación

Ejemplo de entrenamiento representado por el par (x,y). En total se tendrán m, conocido como el conjunto de ejemplos de entrenamiento (training set). La matriz X consiste en poner en columnas los distintos ejemplos de entrenamiento. Con las salidas Y se procede de la misma manera.



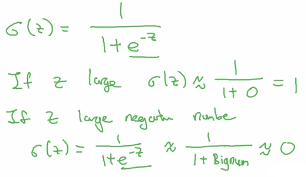
### Regresión logística

La idea es determinar w y b. Como z puede tomar cualquier valor real, se le aplica la función sigmoide para que tome valores entre 0 y 1, correspondientes a una probabilidad.



### Casos extremos

La notación del curso de Machine Learning de Andrew Ng no se utilizará.



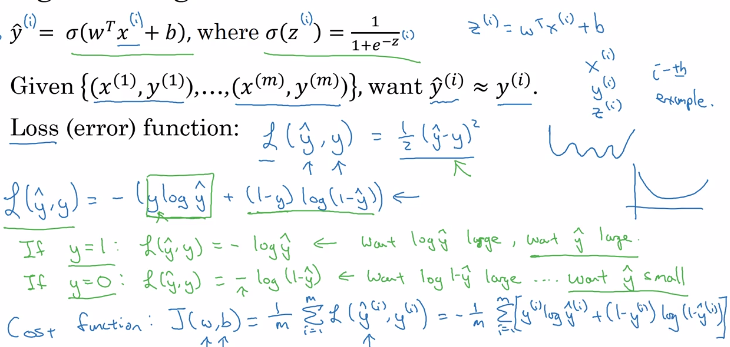
### Función de coste de la regresión logística

Para entrenar los parámetros w y b se necesita una función de coste.

La función de pérdida (Loss function) está asociada a un ejemplo únicamente. Puede definirse como el error cuadrado, pero no es convexa. Por eso se define de otra manera.

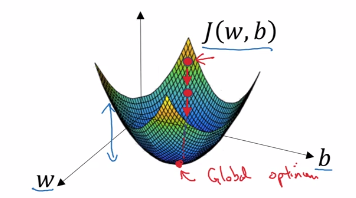
La función de coste se aplica sobre todo al conjunto de entrenamiento, minimizando la función para cada uno de estos puntos. Los valores w y b que minimizan la función serán los de interés.

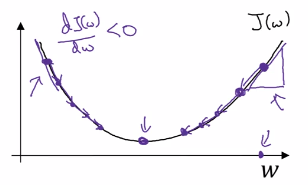
A continuación se verá aplicada a una regresión logística como una red neuronal muy pequeña.



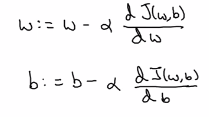
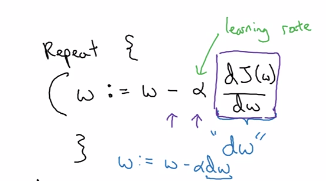
### Gradiente descendente

#### ¿Cómo se minimiza J(w,b)?

Pensando el problema para un b fijo, 



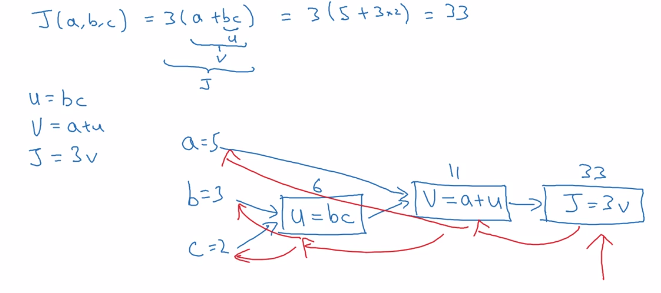
el algoritmo consiste en repetir la ecuación hasta que w o b no varíen drásticamente.

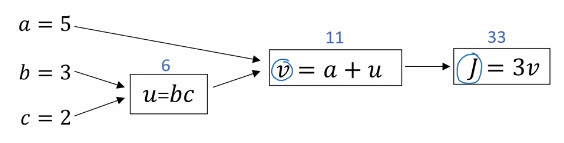


### Gráfico computacional

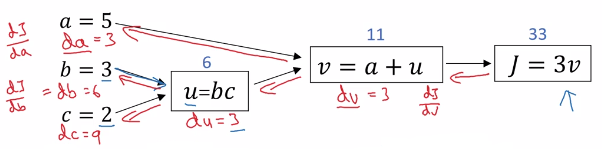
Simplemente para tener una idea de back y forward propagation, veamos con un ejemplo como una función de varias variables puede ser evaluada hacia adelante “forward” mientras que para derivarla habría que hacerlo hacia atrás (“backward”).

En un ejemplo la función J depende de tres variables a, b y c. Las flechas azules representan la propagación hacia adelante para evaluar la función (cálculo de w y b), mientras que las rojas representan las derivadas (gradiente descendente).

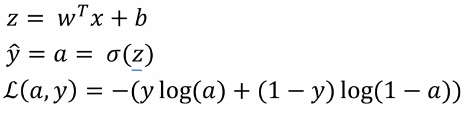




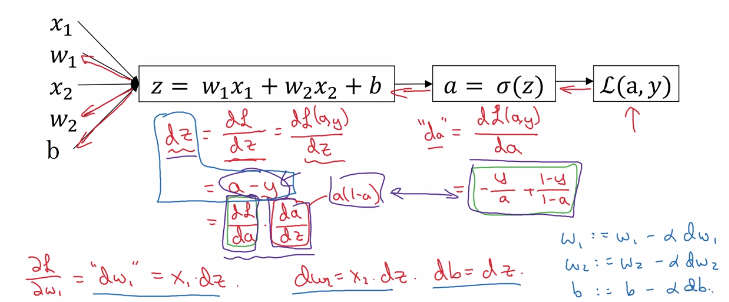
### Derivadas en un gráfico computacional



### Gradiente descendente en regresión logística



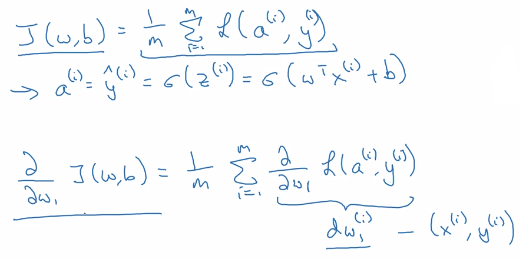
Considerando dos entradas x1,x2, para calcular w1,w2 y b primero habría que calcular dz y luego multiplicar por su correspondiente entrada.



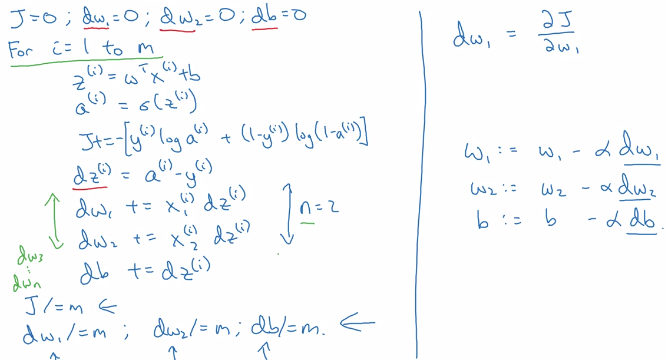
### Gradiente descendente en m ejemplos

En lugar de hacer lo anterior sobre la función de pérdida, debemos hacerlo sobre la función de coste.

Pero las derivadas toman una forma simple como el promedio sobre todos los ejemplos.



El algoritmo se puede escribir de la siguiente manera

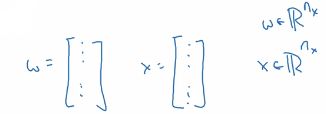


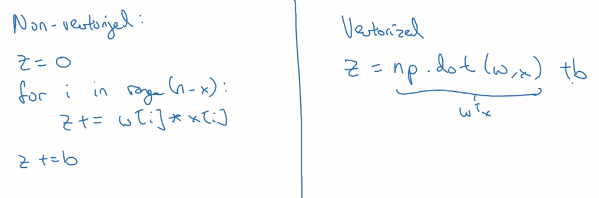
representando un único paso. Esto se repetirá hasta que tanto w como b converjan a un valor constante.

En este algoritmo tenemos dos for loops. Para que sea más eficiente se debería vectorizar.

## Python and Vectorization

## Vectorización



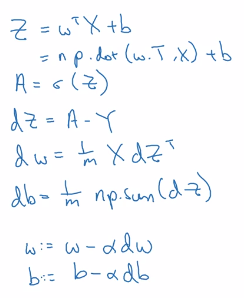




Un ejemplo hecho por Andrew Ng demuestra la diferencia entre vectorizar o no el algoritmo.

## Implementación vectorial del gradiente descendente en regresión logística

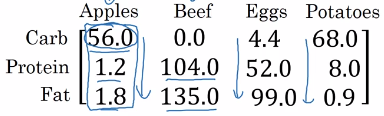
Todavía se necesitaría un for-loop sobre el número de iteraciones que se quieren para minimizar la función de coste.



### Broadcasting en Python

Es una técnica de Python para hacer funcionar el código más rápido.

A modo de ejemplo se quiere pasar la matriz a una matriz de porcentajes. Para ello primero conviene sumar sobre cada columna los valores y dividir ese valor total sobre cada elemento de la matriz (y multiplicar por 100).

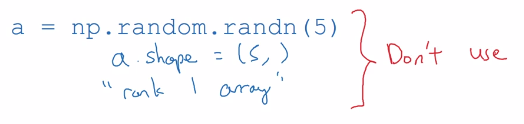




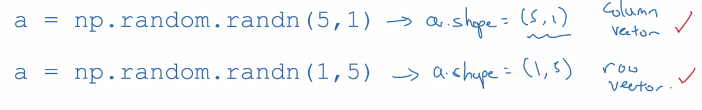
donde axis=0 indica sumar las por columna.

En porcentaje se dividió A que tiene dimensión 3x4 por cal que tiene dimensión 1x4.

### Python/ Numpy vectors



No son recomendables los arrays de rank 1 porque, por ejemplo, al usar np.dot() el comportamiento no es el esperado.

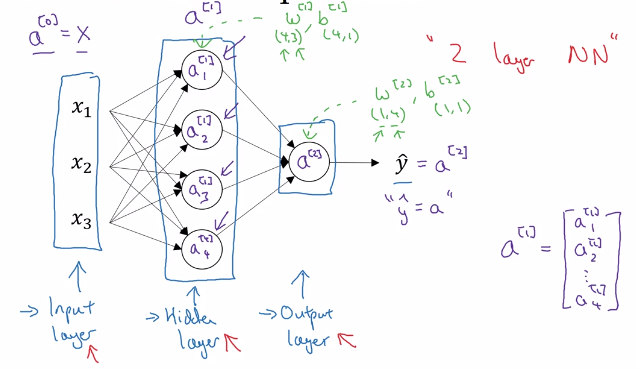


## 

# Week 3: Shallow Neural Network

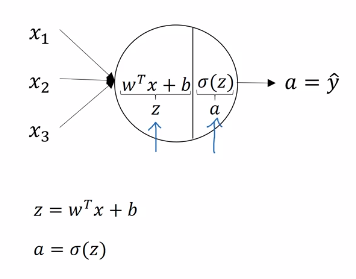
## Representación de Redes Neuronales

Una red neuronal con una capa oculta.

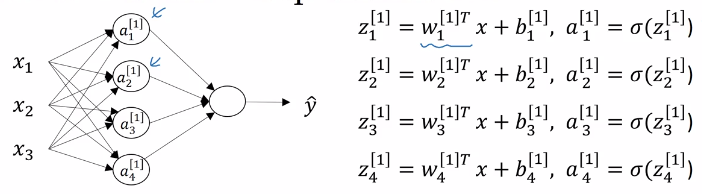


## ¿Cómo se computan las salidas?

Un nodo en una red neuronal consiste en dos pasos de una regresión logística.



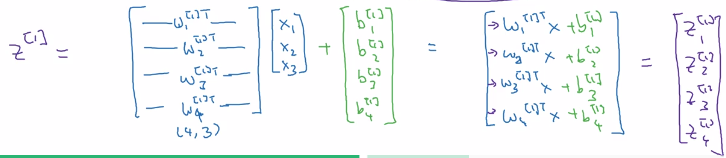
Una red neuronal hace esto mismo, pero más veces. Esto es

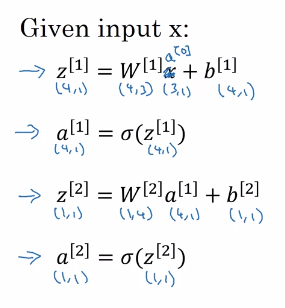


donde la notación viene dada por



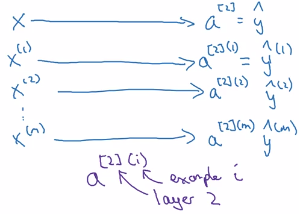
Veamos cómo hacer para vectorizar estas ecuaciones



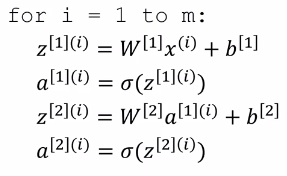


Veamos ahora cómo predecir la salida de una red neuronal pero dadas varias entradas, no solo un ejemplo.

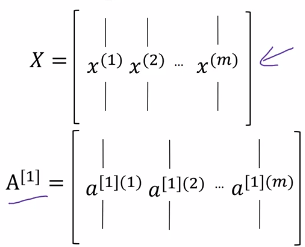
## Introduciendo un poco de notación

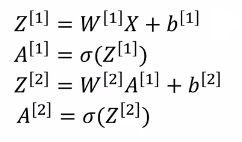


Loopeando sobre todos los ejemplos

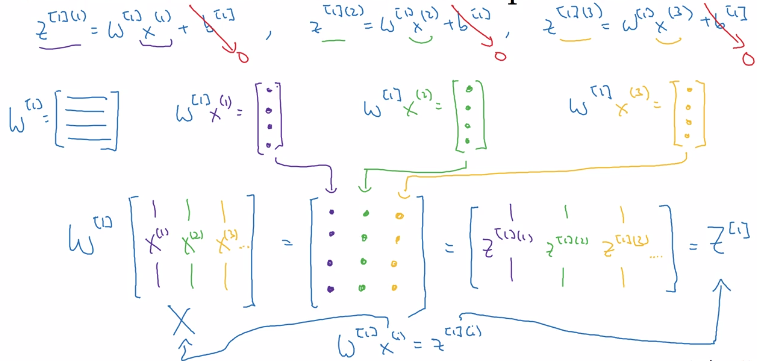


Lo que queremos hacer ahora es vectorizarlo. Escribiendo las entradas en columnas,

La vectorización resulta (Forward propagation)

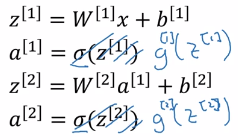


## Explicación de la implementación vectorizada



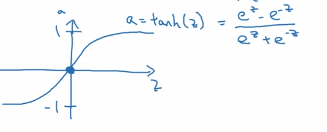
La función sigmoide no es la mejor elección en redes neuronales. Estudiemos otras funciones de activación más útiles.

## Funciones de activación

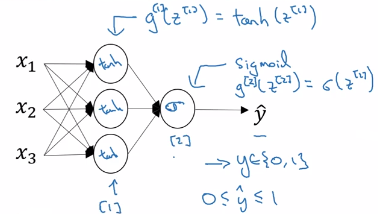


Reemplacemos la función sigmoide por una función de activación general g().

En general la tangente hiperbólica funciona mejor.

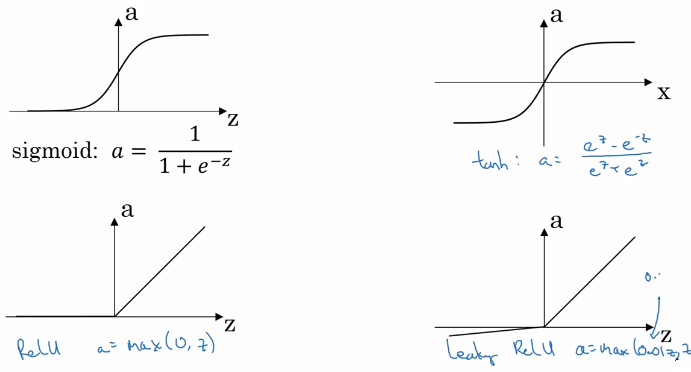


Pero se pueden utilizar distintas funciones de activación en distintas capas. Es conveniente utilizar en la salida la función sigmoide para clasificación binaria, ya que entrega valores entre 0 y 1. Pero en las capas ocultas conviene utilizar la tangente hiperbólica.



## Ventajas y desventajas de las funciones de activación

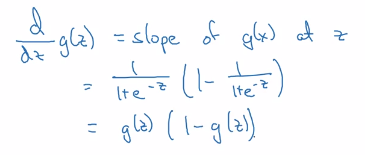
En resumen,



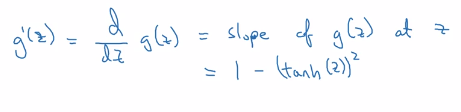
## Derivadas de las funciones de activación

Para backpropagation necesitamos conocer las derivadas de las funciones de activación.

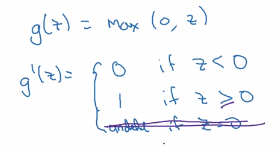
* Función de activación sigmoide



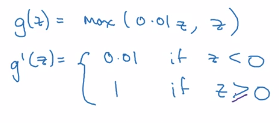
* Función de activación tangente hiperbólica



* RELU

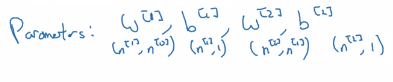


* Leaky RELU

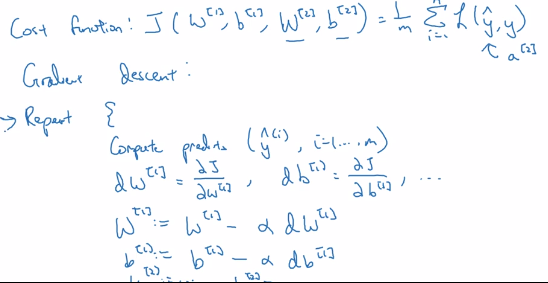


## Gradiente descendente para redes neuronales

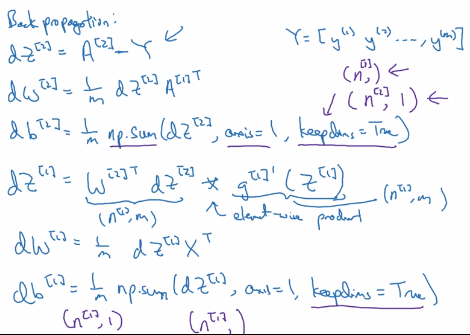
Consideremos primero una capa oculta nomás.

Previamente habíamos visto como calcular las predicciones y\_hat, ahora las derivadas de la función de coste.

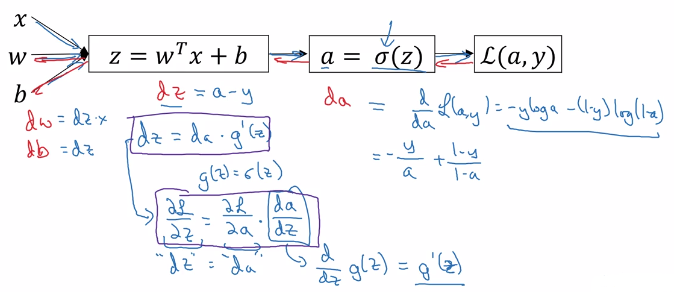


Fórmulas para calcular derivadas (Backward propagation)

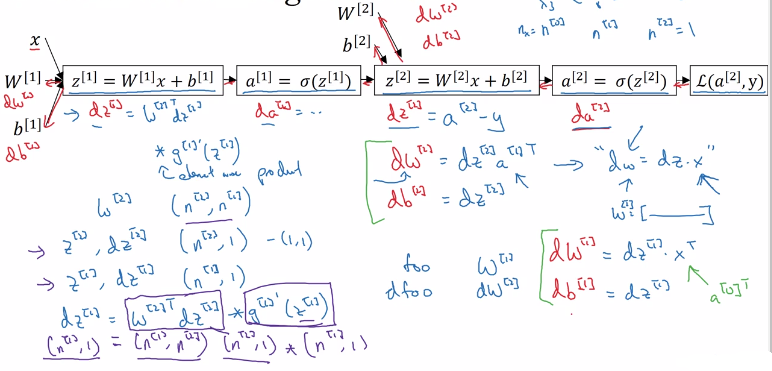


## Intuición de la backward propagation

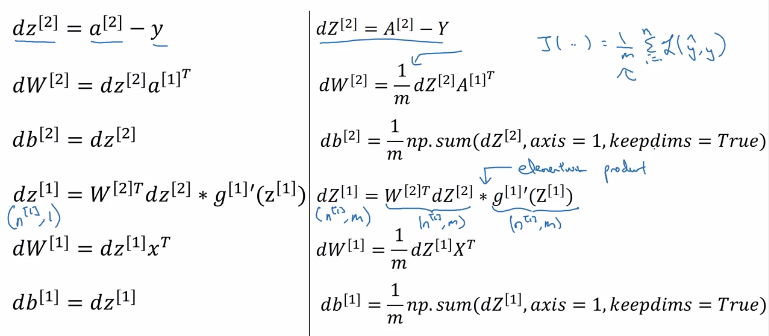
* Regresión logística



* Gradiente en red neuronal



De manera más prolija, las 6 ecuaciones de interés para el backpropagation

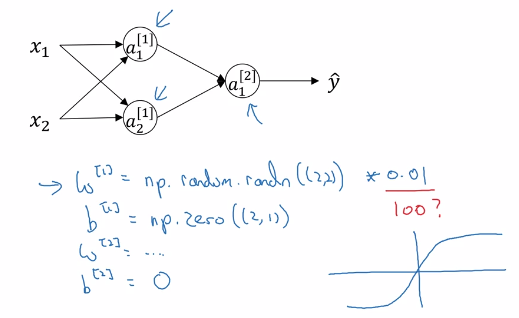


## Inicialización parámetros

* En regresión logística se puede inicializar los parámetros con valor cero.
* Pero la red neuronal hay que inicializarla aleatoriamente. Aplicar gradiente descendente a una red inicializada con ceros, no funcionará.

### Inicialización aleatoria

El factor 0.01 en w es para que se inicialicen con valores pequeños. Si tomasen valores grandes la función de activación saturaría. Por otra parte, a los parámetros b no les afecta inicializarlos con valores nulos.



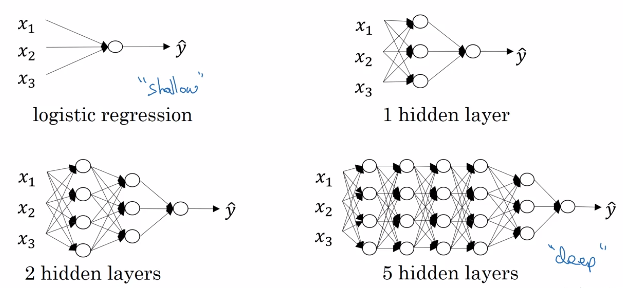
Cuando se entrene alguna red neuronal más profunda sería más razonable usar otro valor que no fuera 0.01, pero se verá más adelante.

## 

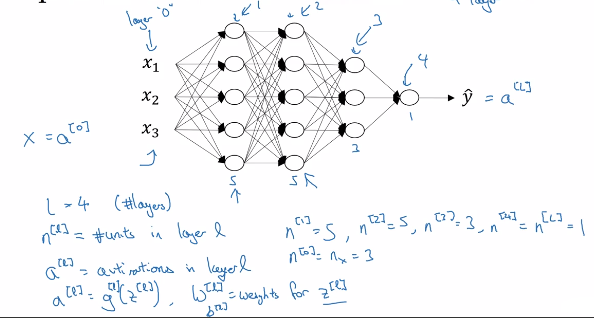
# Week 4 Deep Neural Network

## Redes neuronales profundas de L-capas

Ejemplos de redes neuronales.

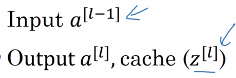


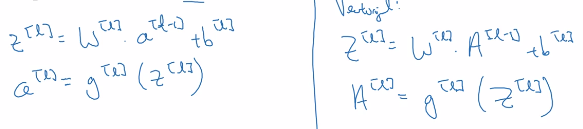
Veamos la notación a utilizar



## Forward and Backward propagation in deep neural networks

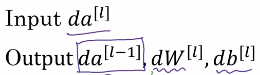
* Forward propagation

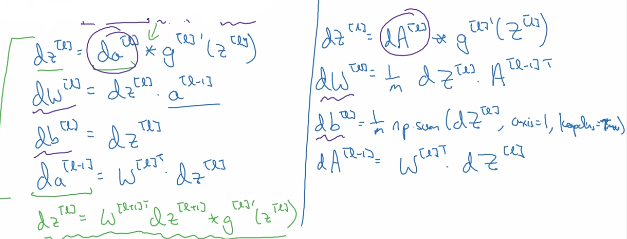




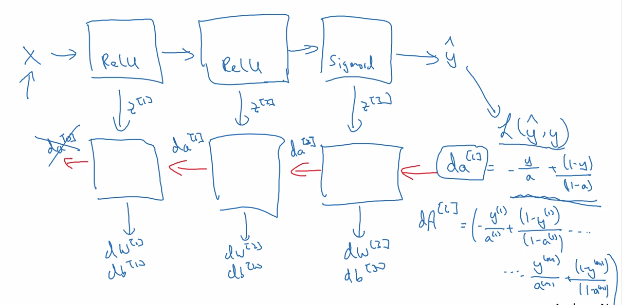
Estas dos ecuaciones se encuentran en un for-loop, el cual no parecería que se puede vectorizar.

* Backward propagation

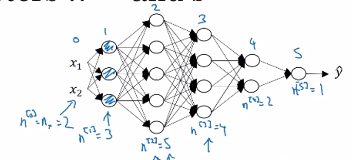




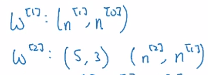
## Resumen



Método de debugging: dimensiones de las matrices



Los parámetros w para un ejemplo:



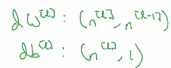
De manera general.



Para b

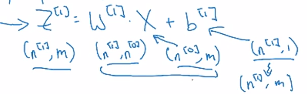


Para backpropagation dw y db tienen la misma dimensión que W y b respectivamente.



Otras cantidades de interés: z = g(a), y a tienen la misma dimensión,

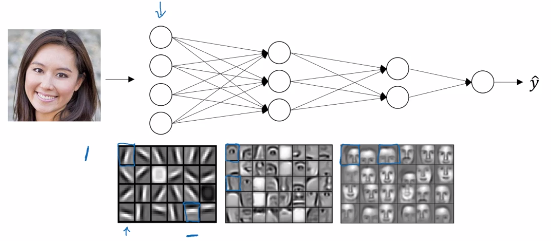
Considerando los m ejemplos a entrenar



En Python b podría seguir siendo de la misma dimensión, pero por Broadcasting es redimensionada al tamaño correcto.

## ¿Por qué las redes neuronales profundas generalmente funcionan mejor que las que no lo son?

Como ejemplo en la identificación de una cara. Se podría pensar a la primera capa de la red neuronal como la encargada de identificar bordes. A la siguiente para identificar ojos, o narices mediante el encajado de bordes. Mediante el rejunte de ojos, narices la red puede detectar caras.



Para un audio, a bajo nivel la red neuronal podría detectar los cambios de sonido, luego unidades básicas de sonido, como fonemas, luego poder reconocer palabras, para luego reconocer oraciones o frases.



## ¿Qué son los hiper parámetros?

Dados los parámetros de una red neuronal,

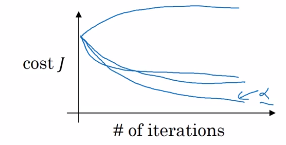


los hiper parámetros son aquellos que controlan y determinar el valor de los parámetros.

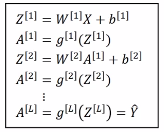
Ellos son:

* Learning ratio alfa
* número de iteraciones
* número de capas ocultas L
* número de unidades ocultas n^[1],n^[2],..
* elección de funciones de activación
* etc.

*Deep Learning es un proceso empírico. El valor de los hiper parámetros se calcula mediante la experimentación.*



## Forward y Backward propagation

**** 