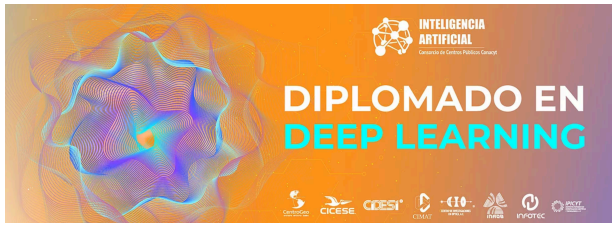


Introducción al Aprendizaje Profundo

Hugo Jair Escalante

hugojair@inaoep.mx

https://ccc.inaoep.mx/~hugojair/

@hugojair

Contenido

- Primer acercamiento a DL
- Redes neuronales profundas
- Desmenuzando a las redes neuronales
- Entrenamiento de redes neuronales

Aprendizaje profundo
un primer acercamiento

En breve…

- DL es una metodología del aprendizaje computacional que intenta resolver problemas de modelado construyendo modelos apilados en capas de parámetros de abstracción incremental

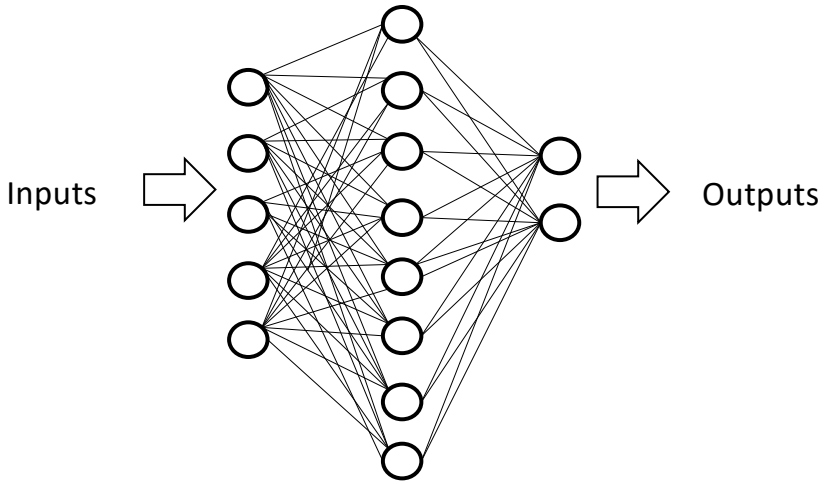
- Las capas de estos modelos capturan información descriptiva/discriminativa a partir de los datos *crudos*

- Pueden usarse para tareas de aprendizaje supervisado, no supervisado, por refuerzo, extracción de atributos, modelado probabilista, etc.

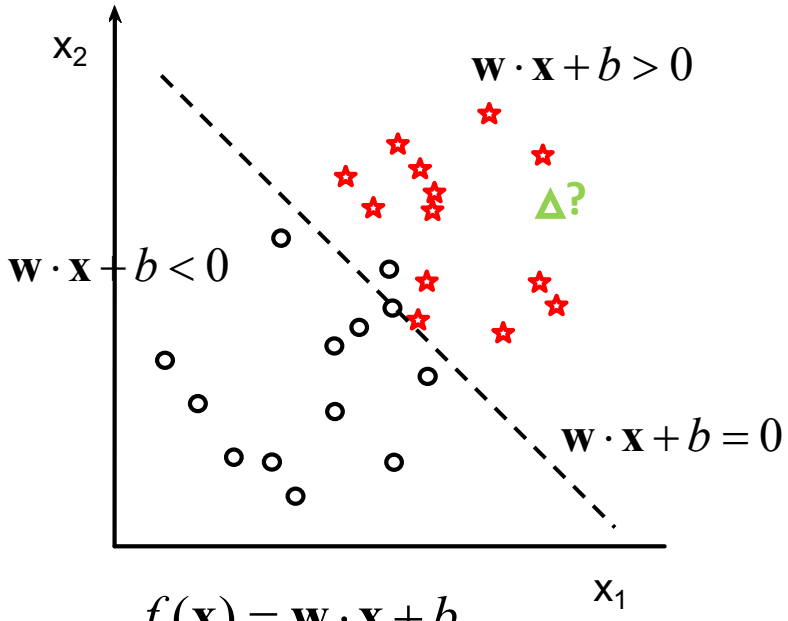
- Ejemplos: MLPs, DNNs, CNNs, DBNs, Aes, etc.

En breve…

- Cómo se ve un modelo no profundo?



$$f(\mathbf{x}) = \mathbf{w}\phi(\mathbf{x}) + b$$

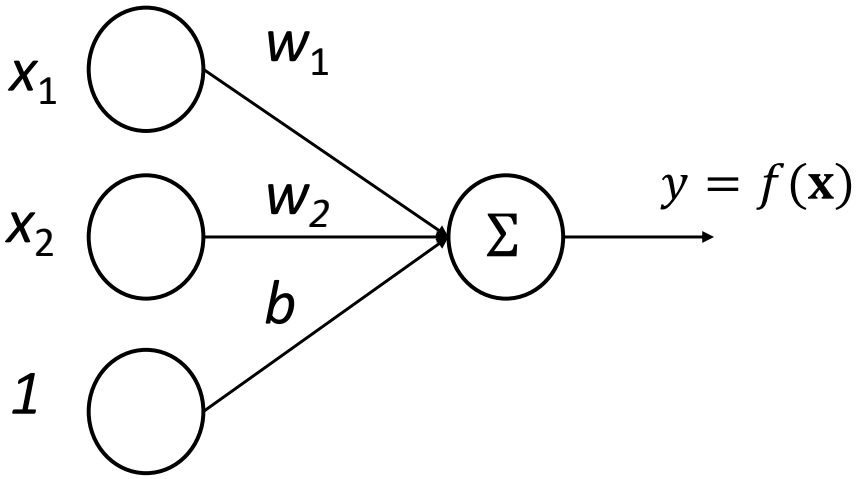


$$f(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} + b$$

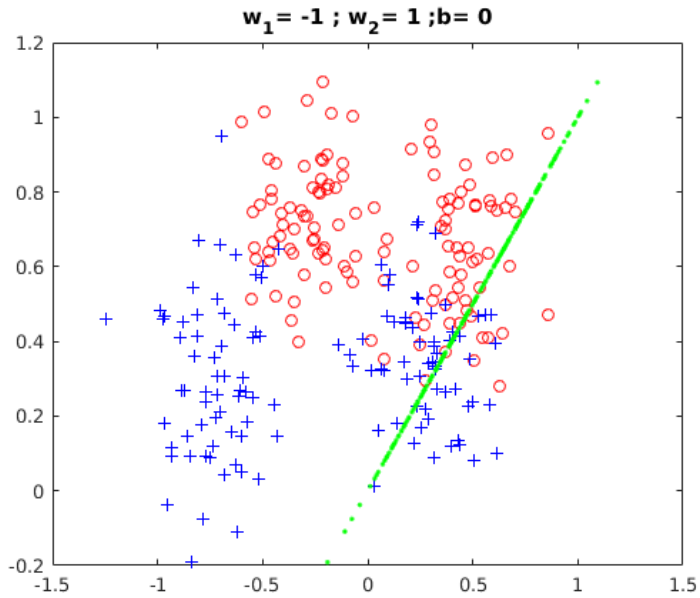
Flashback: el perceptron

- El perceptron aprende una función de decision de la forma:

$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

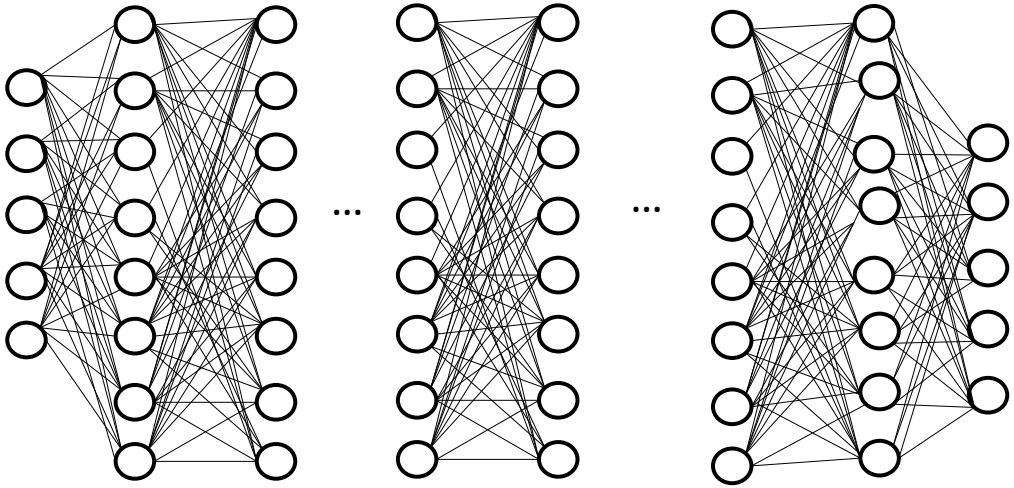


$$y = w_1 x_1 + w_2 x_2 + b$$



En breve…

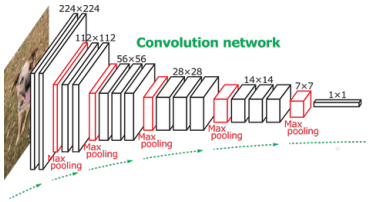
- Cómo se ve un modelo (no tan) profundo?

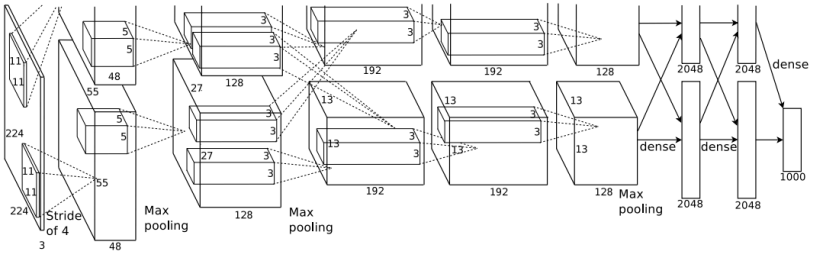


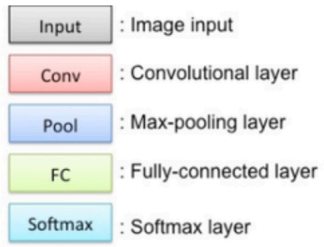
$$f(\mathrm{x}) = \mathbf{W_3}\boldsymbol{\phi_3}(\mathbf{W_2}\boldsymbol{\phi_2}(\mathbf{W_1}\boldsymbol{\phi_1}(\mathbf{X}) + b) + b_2) + b_3$$

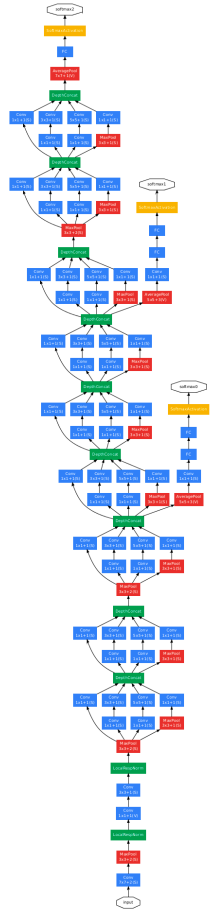
En breve…

- Más profundidad (CNNs)





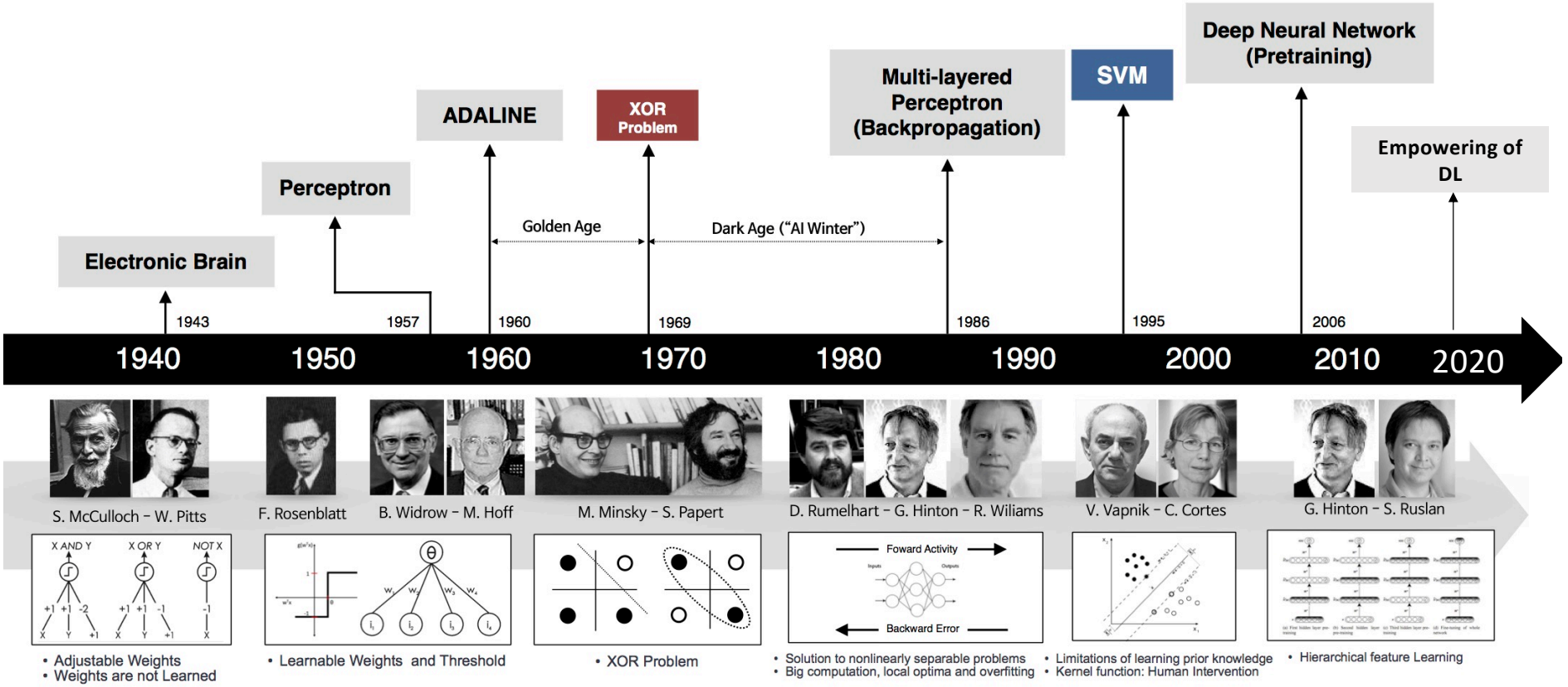




En breve…

- Características del aprendizaje profundo:
  - Un gran número de parámetros (del orden de millones)
  - Requiere de grandes cantidades de datos para ser entrenado
  - Puede aprender a extraer atributos para caracterizar datos crudos
  - Puede tomar ventaja de datos no etiquetados
  - Modelos extremadamente complejos
  - Requieren de hardware especializado para entrenamiento eficiente
  - Dominan los campos de aplicación de aprendizaje computacional (e.g., NLP, visión computacional)

El Boom de DL: breve historia



http://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

A.M. TURING

The A.M. Turing Award, sometimes referred to as the "Nobel Prize of Computing," was named in honor of Alan Mathison Turing (1912–1954), a British mathematician and computer scientist. He made fundamental advances in computer architecture, algorithms, formalization of computing, and artificial intelligence. Turing was also instrumental in British code-breaking work during World War II.

FATHERS OF THE DEEP LEARNING REVOLUTION RECEIVE ACM A.M. TURING AWARD

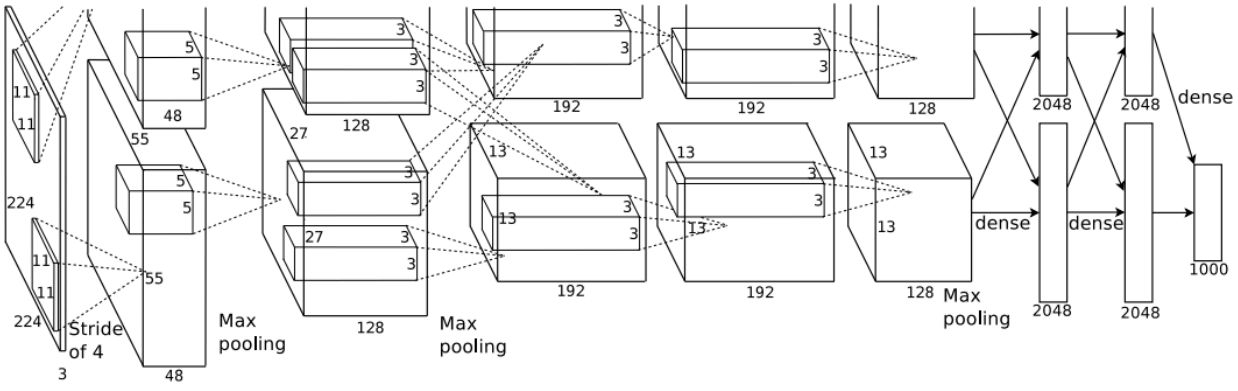
Bengio, Hinton, and LeCun Ushered in Major Breakthroughs in Artificial Intelligence

El Boom de DL: logros importantes

- Clasificación de imágenes a gran escala
- Reconocimento de voz
- Reconocimiento de rostros
- Aprendizaje profundo por refuerzo
- Otros logros
  - Image captioning
  - Word embeddings
  - Reconocimiento de gestos, acciones
  - Super resolución
  - …

Logros importantes I (ImageNET)

- En 2020, Krizhevsky et al. lograron entrenar una red convolucional usando ~1 millón de imágenes para enfrentar el concurso *ImageNET large scale classification challenge* (1000 categorías, millones de imágenes)



IMAGENET

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. **ImageNet Classification with Deep Convolutional Neural Networks.** Advances in Neural Information Processing Systems 25 (NIPS 2012)

ImageNET Challenge

- ImageNET: un recurso compuesto por millones de imágenes
- Imágenes de la Web descargadas a partir de los synsets de WordNet
- Tareas:
  - Clasificación
  - Detección de objetos
  - Localización de objetos

IMAGENET

Year	Train images (per class)	Val images (per class)	Test images (per class)
Image classification annotations (1000 object classes)			
ILSVRC2010	1,261,406 (668–3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384–1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732–1300)	50,000 (50)	100,000 (100)

The numbers in parentheses correspond to (minimum per class–maximum per class). The 1000 classes change from year to year but are consistent between image classification and single-object localization tasks in the same year. All images from the image classification task may be used for single-object localization

Logros importantes I (ImageNET)

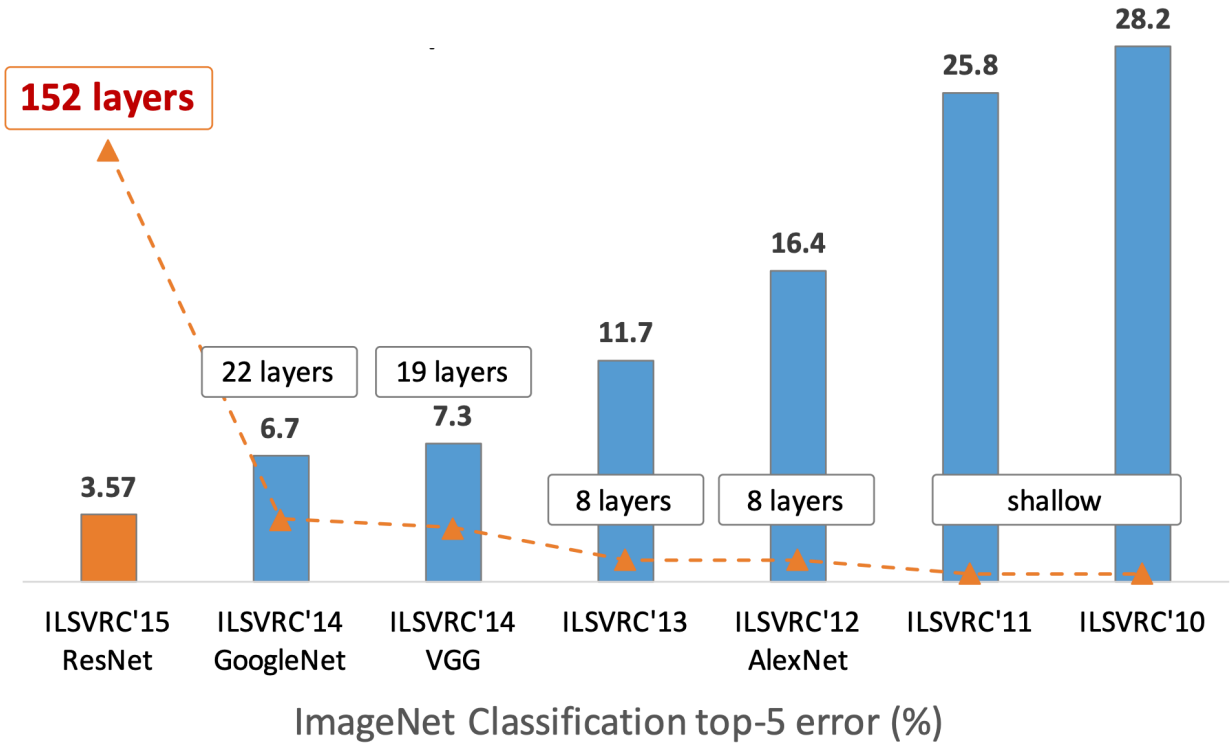
- Las mejoras de desempeño obtenidas por AlexNet fueron impresionantes
- Las claves de este logro:
  - Hardware: GPUs
  - Funciones de activación RELU
  - Regularización dropout
  - Big data / modelo complejo

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

IMAGENET

Logros importantes I (ImageNET)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016

Logros importantes I (ImageNET)

- AlexNet es hoy en día (quizá) la arquitectura más usada para clasificar imágenes, se incluye en la mayoría de las librerías de DL

- Este logro inspiró a otros en el área de visión computacional, eventualmente estableciendo a DL como *la metodología* para entender escenas

Imagenet classification with deep convolutional neural networks

Authors	Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton
Publication date	2012
Conference	Advances in neural information processing systems
Pages	1097-1105
Description	We trained a large, deep convolutional neural network to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 39.7\% and 18.9\% which is considerably better than the previous state-of-the-art results. The neural network, which has 60 million parameters and 500,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and two globally connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally connected layers we employed a new regularization method that proved to be very effective.
Total citations	Cited by 38561

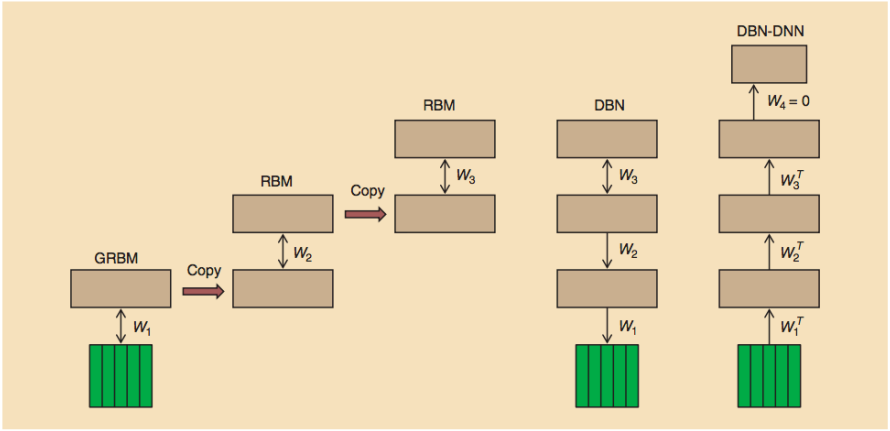
Scholar articles Imagenet classification with deep convolutional neural networks
A Krizhevsky, I Sutskever, GE Hinton - Advances in neural information processing systems, 2012
Cited by 38561 Related articles All 101 versions

Revisado, Abril 25, 2019: 38,561 citas

Re-revisado, Mayo 22, 2019: 40,022 citas

Logros importantes II (Reconocimiento del habla)

- Alrededor de 2012 las compañías más importantes en TICs convergieron hacia el uso de las *Restricted Boltzman Machines* para el reconocimiento del habla
- Idea clave: pre-entrenamiento de RBMs + fine tunning + HMM



Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012

Logros importantes II (Reconocimiento del habla)

- Resultados impresionantes se obtuvieron por dichos modelos

[TABLE 1] COMPARISONS AMONG THE REPORTED SPEAKER-INDEPENDENT (SI) PHONETIC RECOGNITION ACCURACY RESULTS ON TIMIT CORE TEST SET WITH 192 SENTENCES.

METHOD	PER
CD-HMM [26]	27.3%
AUGMENTED CONDITIONAL RANDOM FIELDS [26]	26.6%
RANDOMLY INITIALIZED RECURRENT NEURAL NETS [27]	26.1%
BAYESIAN TRIPHONE GMM-HMM [28]	25.6%
MONOPHONE HTMS [29]	24.8%
HETEROGENEOUS CLASSIFIERS [30]	24.4%
MONOPHONE RANDOMLY INITIALIZED DNNs (SIX LAYERS) [13]	23.4%
MONOPHONE DBN-DNNs (SIX LAYERS) [13]	22.4%
MONOPHONE DBN-DNNs WITH MMI TRAINING [31]	22.1%
TRIPHONE GMM-HMMs DT W/ BMMI [32]	21.7%
MONOPHONE DBN-DNNs ON FBANK (EIGHT LAYERS) [13]	20.7%
MONOPHONE MCRBM-DBN-DNNs ON FBANK (FIVE LAYERS) [33]	20.5%
MONOPHONE CONVOLUTIONAL DNNs ON FBANK (THREE LAYERS) [34]	20.0%

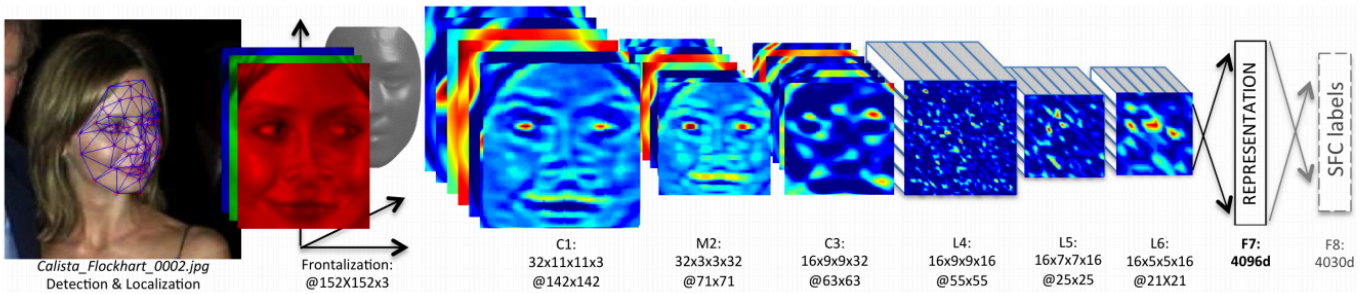
[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

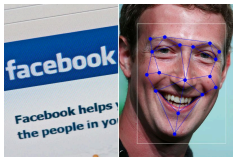
TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012

Logros importantes ++: Deepface

- En 2014 se anuncio DeepFace una red convolucional profundal entrenada en 4.4 millones de imágenes

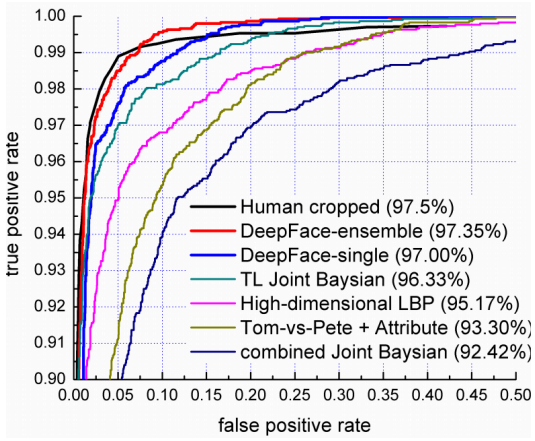


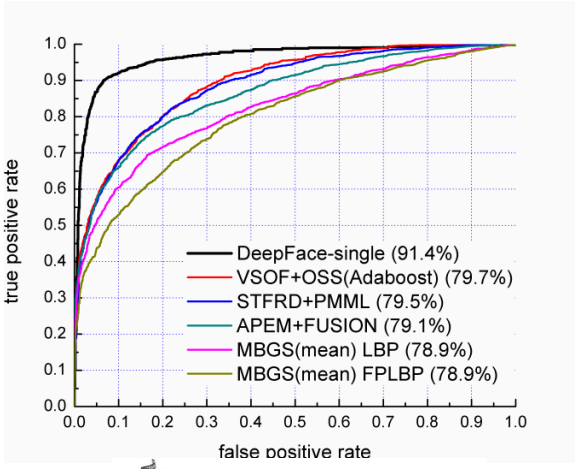


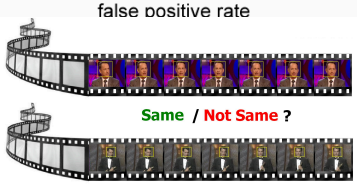
Yaniv Tagiman, Ming Yang, Marc Aurelio Ranzato, Lior Wolf**. DeepFace: Closing the Gap to Human-Level Performance in Face Verification**, CVPR 2014

Logros importantes ++: Deepface

- Deepface obtuvo una tasa de reconocimiento del 97.35% en el conjunto de datos LFW (humanos: 97.5%) y 91.4% en el conjunto de datos de Youtube

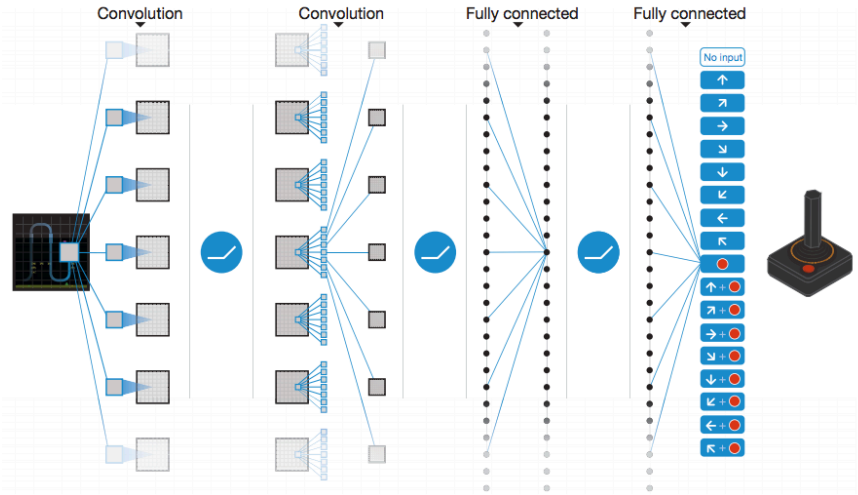






Logros importantes ++: DeepRL

- En 2015, DeepMind presentó su Deep-Q Network: una arquitectura de DL capaz de aprender a jugar Atari simplemente “mirando” los pixeles de la pantalla del juego

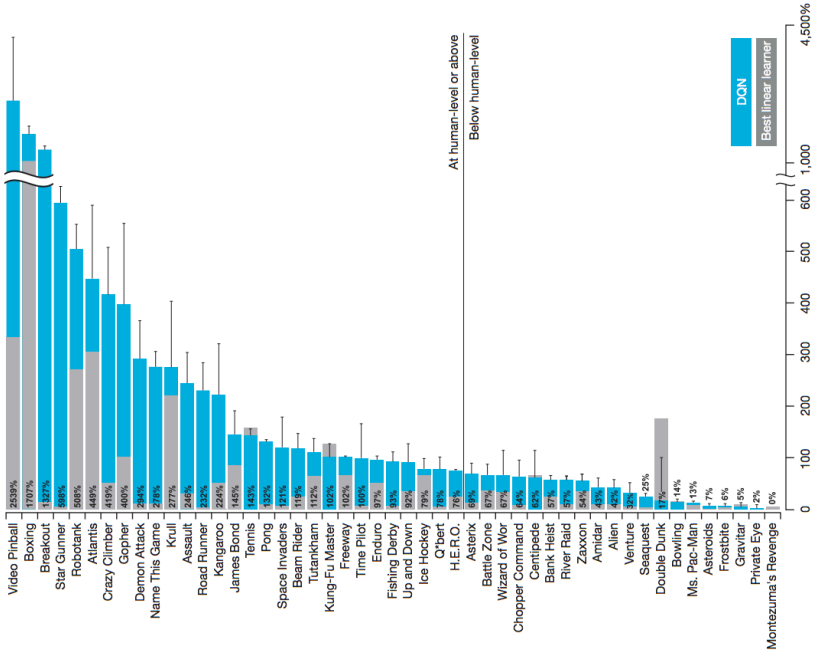


Volodymyr Mnih, et al. **Human-level Control through Deep Reinforcement Learning** In Nature, 518: 529–533, 2015.

https://www.youtube.com/watch?v=TmPfTpjtdgg

Logros importantes ++: DeepRL

- DQN superó a todas las soluciones previas al problema en 50 juegos de Atari
- Alcanzando nivel de juego de humanos en un gran número de juegos



https://deepmind.com/blog/deep-reinforcement-learning/

Logros importantes ++: Alpha Go

Captured Stones

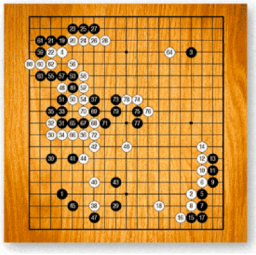
3 hours

AlphaGo Zero plays like a human beginner, forgoing long term strategy to focus on greedily capturing as many stones as possible.

Captured Stones

70 hours

AlphaGo Zero plays at super-human level. The game is disciplined and involves multiple challenges across the board.

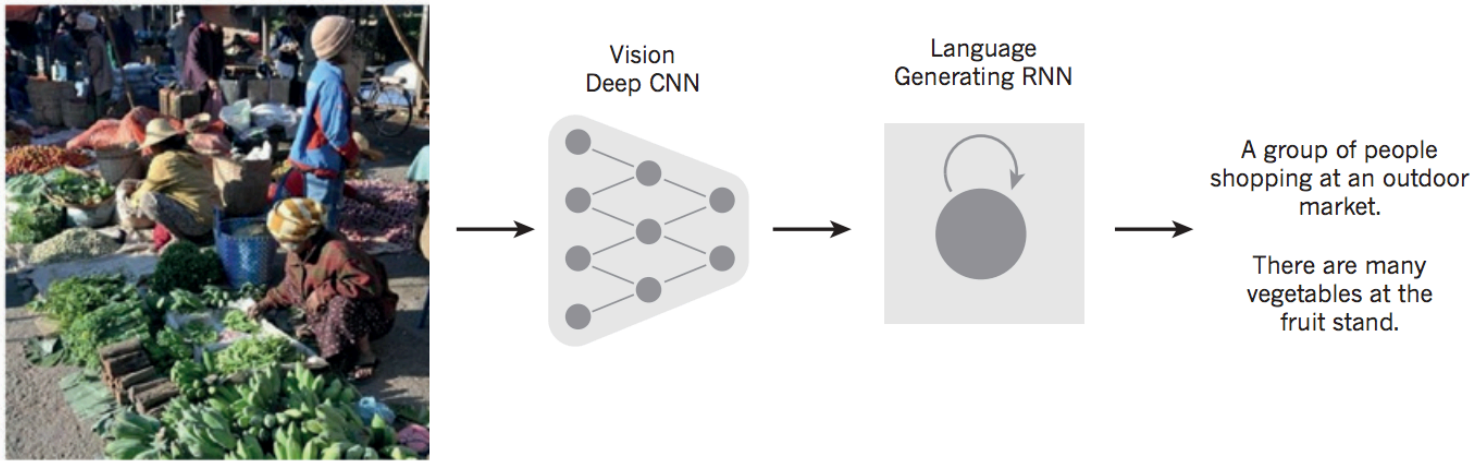


19 hours

AlphaGo Zero has learnt the fundamentals of more advanced Go strategies such as life-and-death, influence and territory.

David Silver et al. **Mastering the game of Go without human knowledge.** Nature, Vol. 550:550-559, 2017

Logros importantes ++: Image Captioning



https://pdollar.wordpress.com/2015/01/21/image-captioning/

Logros importantes ++: Image Captioning

A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background

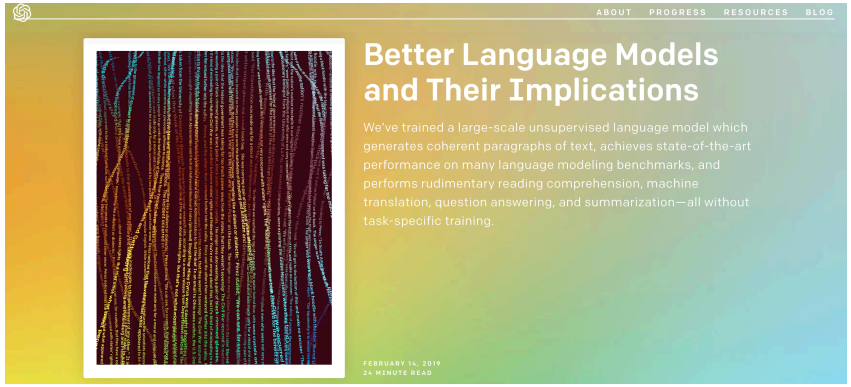
A little **girl** sitting on a bed with a teddy bear.

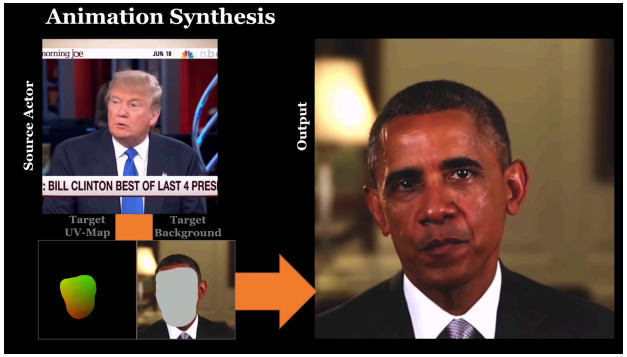
A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

https://pdollar.wordpress.com/2015/01/21/image-captioning/

Logros importantes ++ …





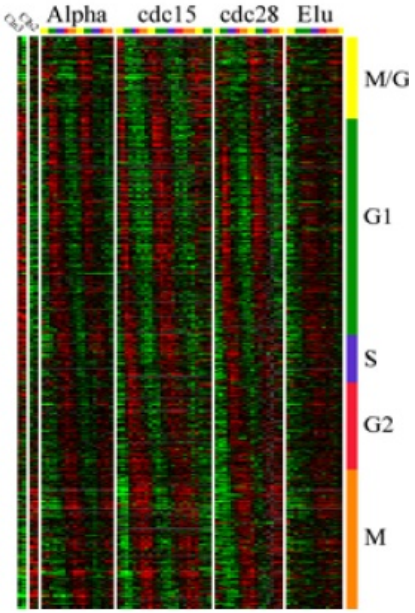
Modelos lineales: Clasificación

Modelos lineales para clasificación

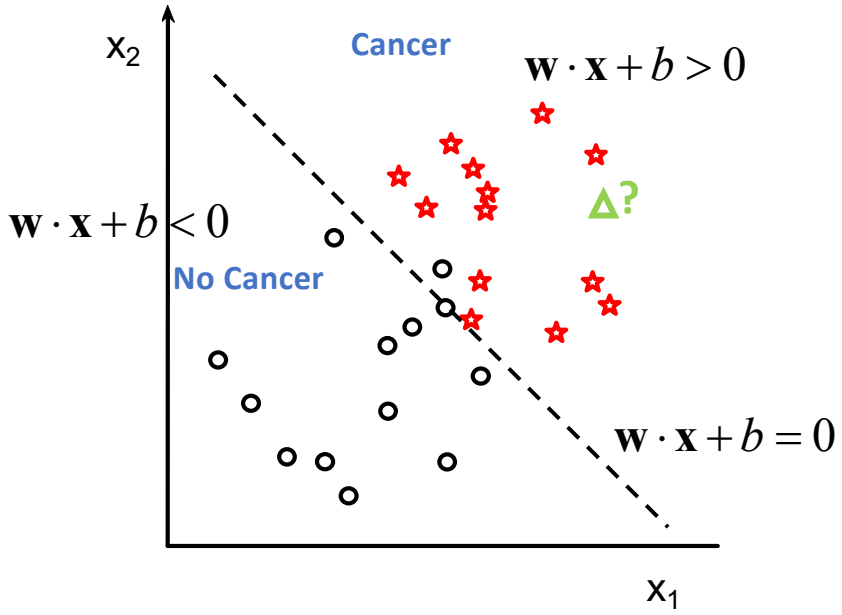
- **Idea:** aprender una función lineal (en los paramétros) que nos permita separar los datos:
  - $f(\mathbf{x}) = \mathbf{w} \bullet \mathbf{x} + b = \sum_{j=1:n} w_j x_j + b$ *(linear discriminant)*
  - $f(\mathbf{x}) = \mathbf{w} \bullet \Phi(\mathbf{x}) + b = \sum_{j} w_j \phi_j(\mathbf{x}) + b$ *(the perceptron)*
  - $f(\mathbf{x}) = \sum_{i=1:m} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b$ *(Kernel-based methods)*

Linear Discriminants and Support Vector Machines, *I. Guyon and D. Stork,* In Smola et al Eds. Advances in Large Margin Classifiers. Pages 147--169, MIT Press, 2000.

Modelos lineales para clasificación



$$\mathbf{x} = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$$



$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

Modelos lineales para clasificación

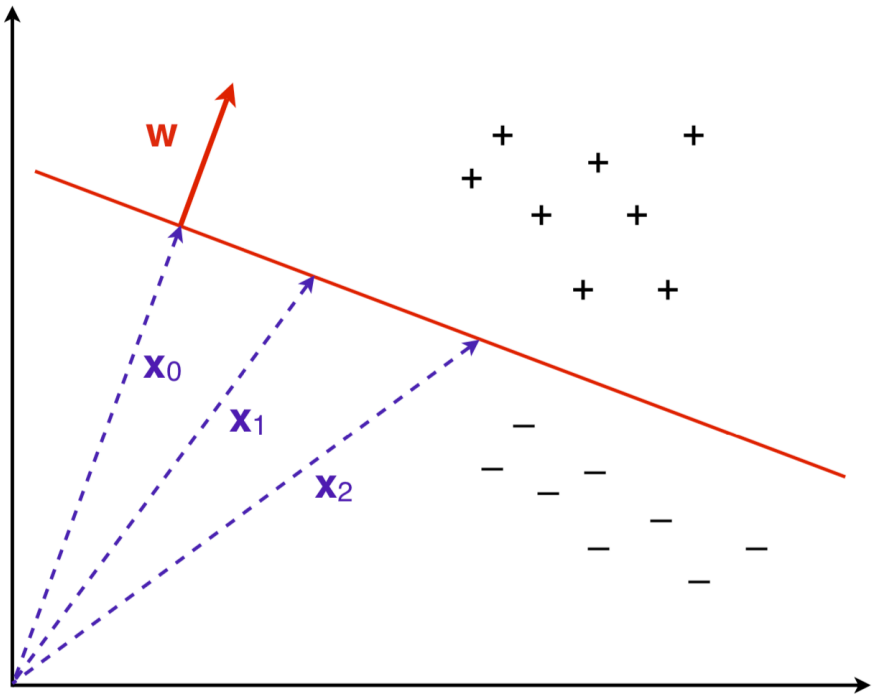
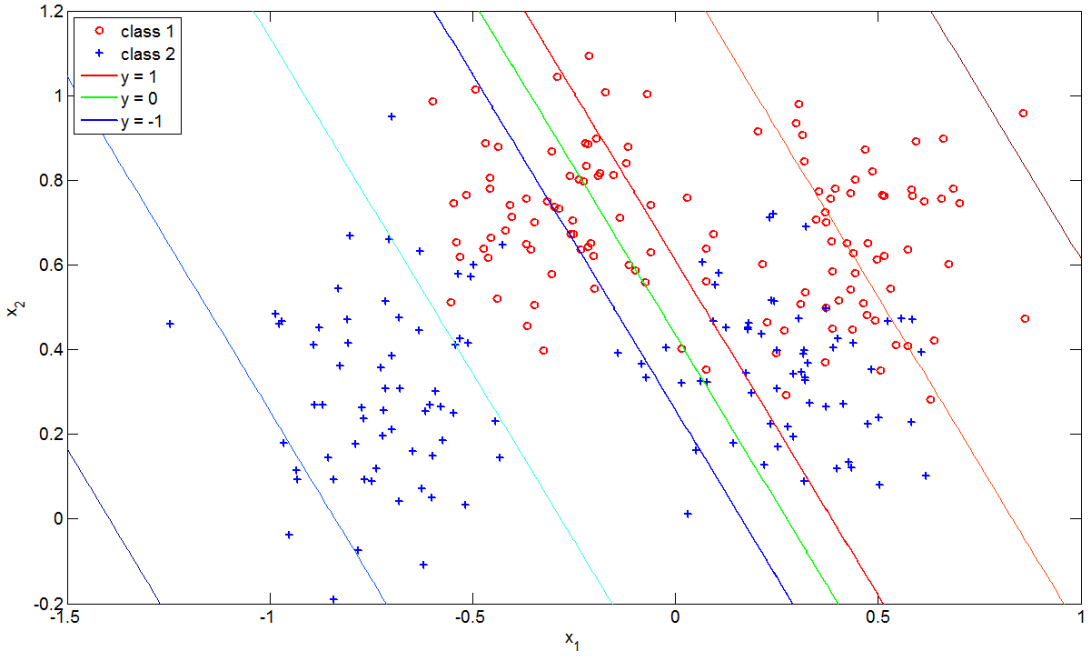


Figura de P. Flach. Machine Learning. The Art and Science of Algorithms that Make Sense of Data, Cambridge University Press, 2012

Modelos lineales para clasificación

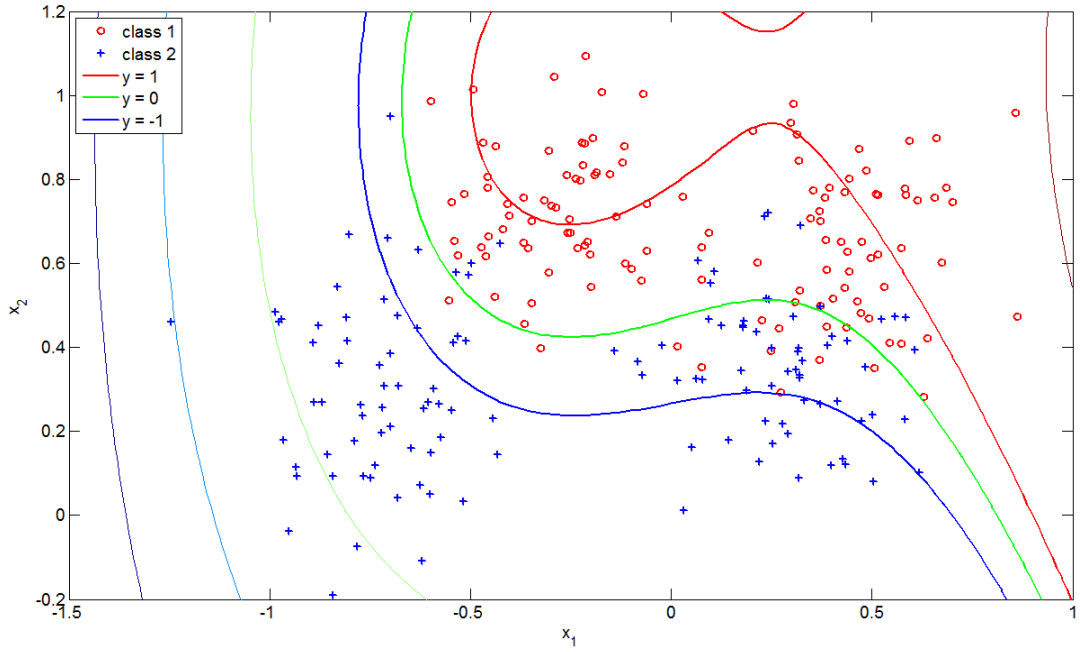
http://clopinet.com/CLOP



Linear support vector machine

Modelos lineales para clasificación

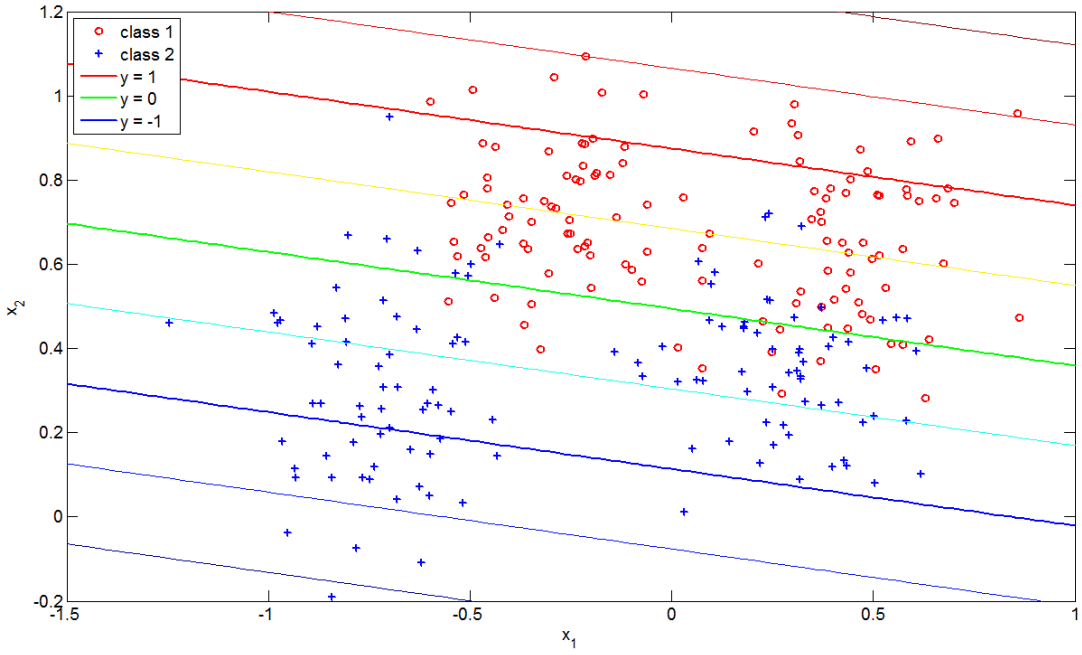
http://clopinet.com/CLOP



"Non-linear" support vector machine

Modelos lineales para clasificación

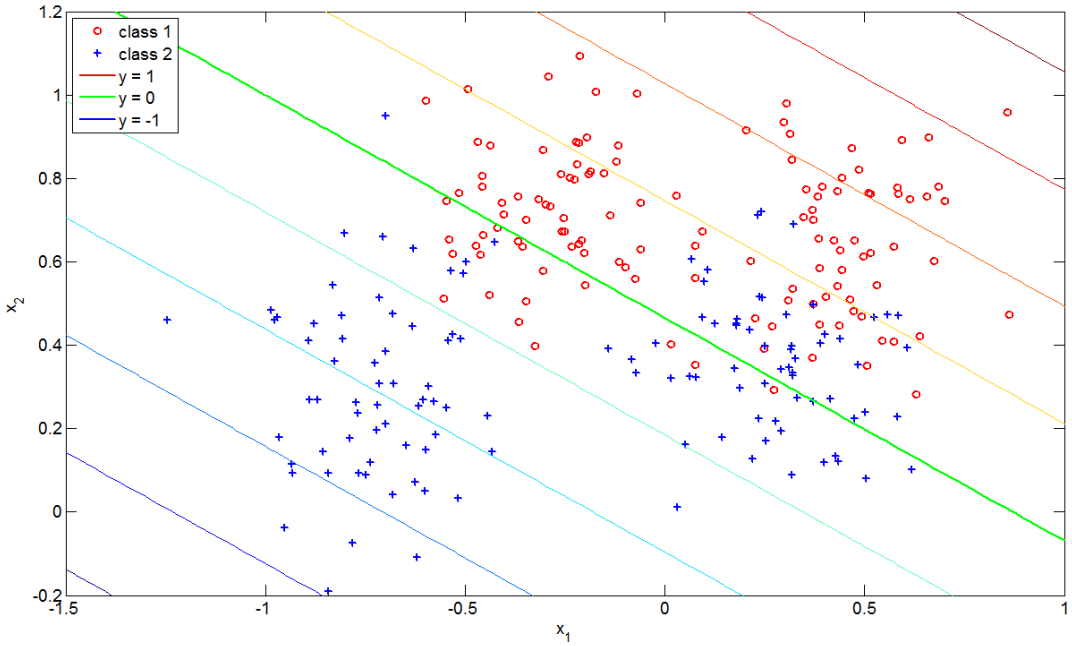
http://clopinet.com/CLOP



Kernel ridge regression

Modelos lineales para clasificación

http://clopinet.com/CLOP



Zarbi classifier

El perceptron

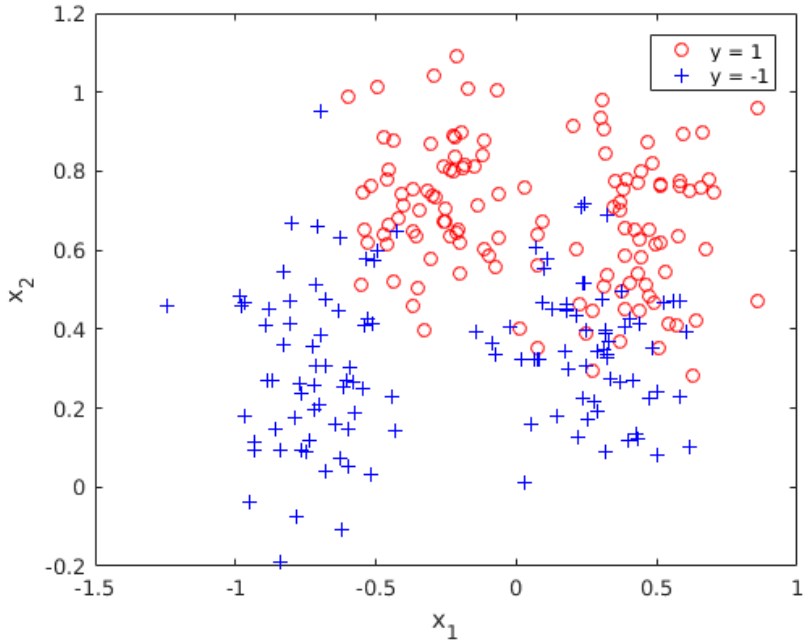
- **Perceptron:** Un clasificador lineal simple que puede resolver problemas de clasificación que son linealmente separables (abuelo de las redes neuronales y las máquinas de soporte vectorial)

- El perceptron aprende una función de decision de la forma:

$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

El perceptron

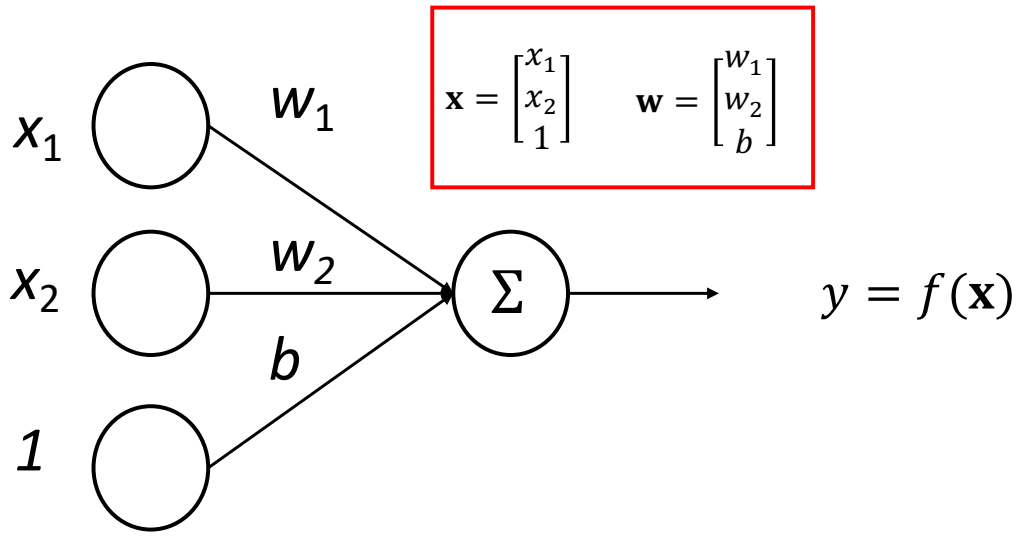
- Dado: $D = \{(\mathbf{x}_i, y_i)_{1.....N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$



El perceptron

- El perceptron aprende una función de decisión de la forma:

$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

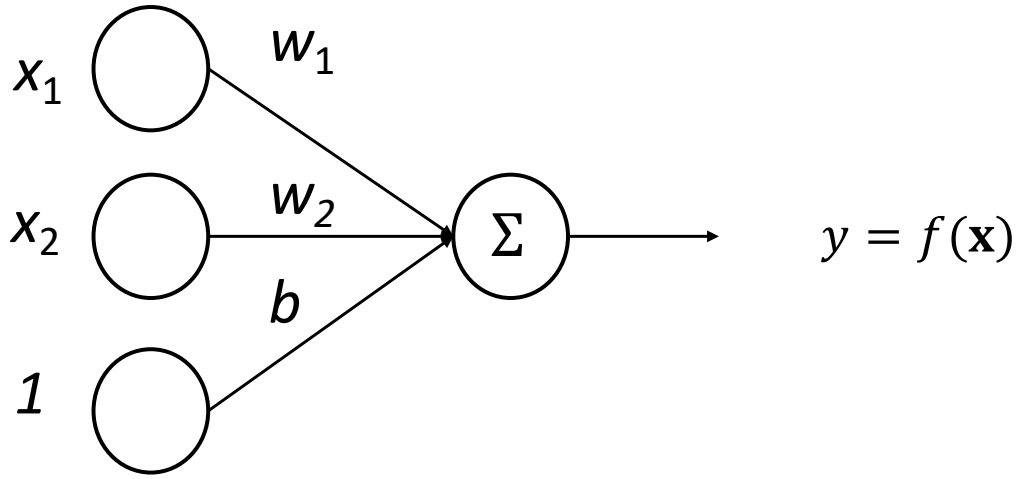


$$y = w_1 x_1 + w_2 x_2 + b \qquad y = \mathbf{w}\mathbf{x}$$

El perceptron

- El perceptron aprende una función de decision de la forma:

$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

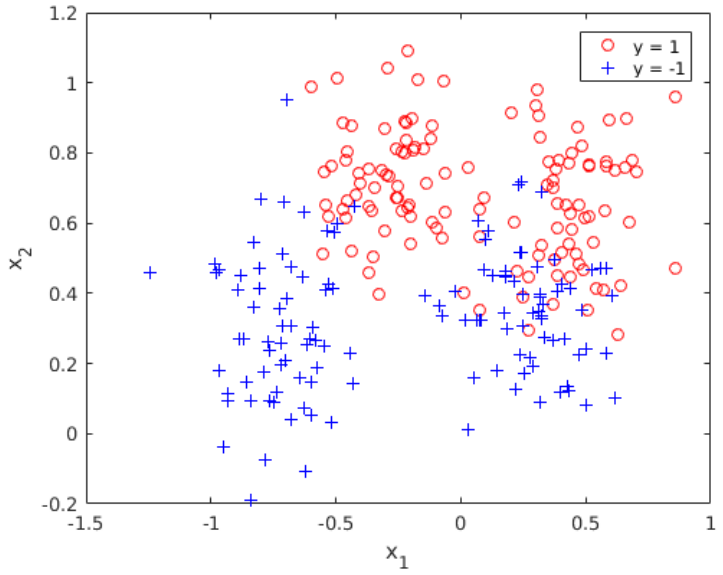


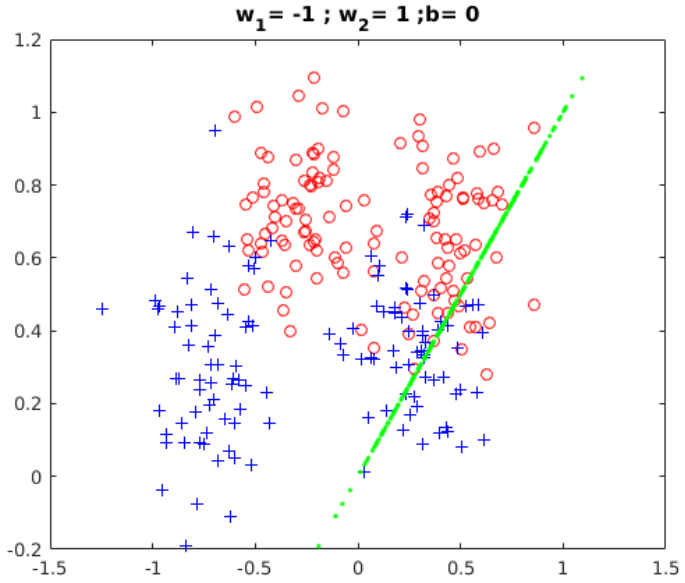
$$y = w_1 x_1 + w_2 x_2 + b$$

El perceptron

- El perceptron aprende una función de decision de la forma:

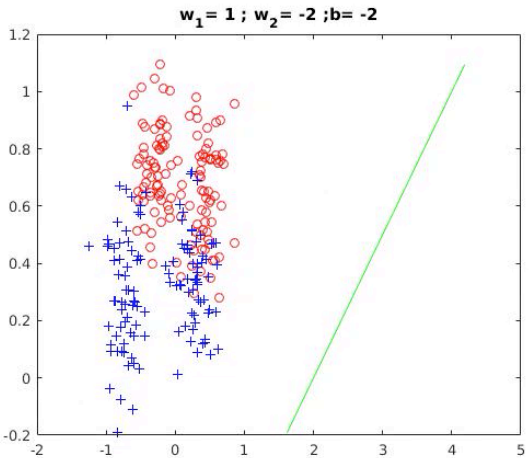
$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

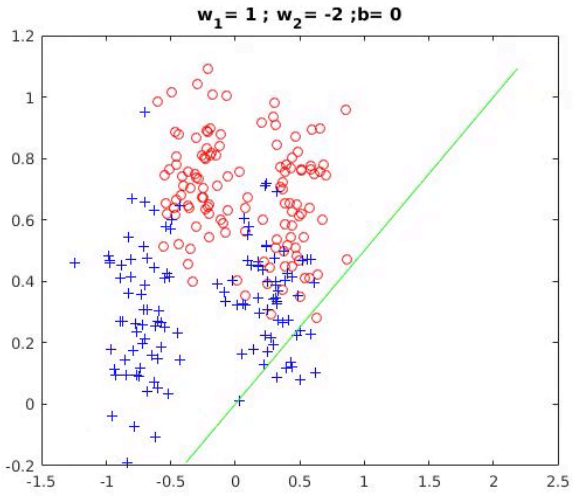


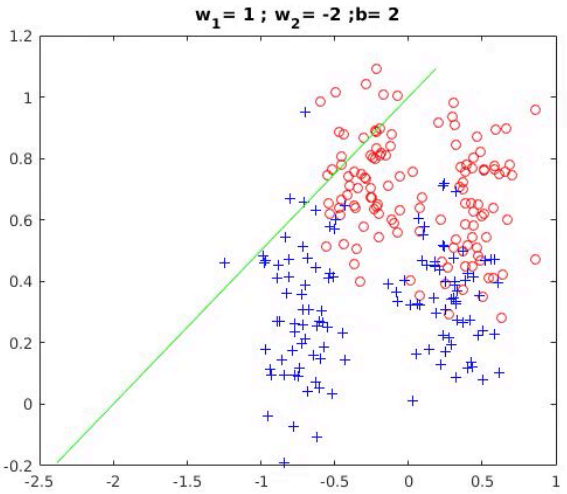


El perceptron

- Diferentes paramétros resultan en diferentes modelos







El perceptron

- Cómo estimar los pesos **w**?

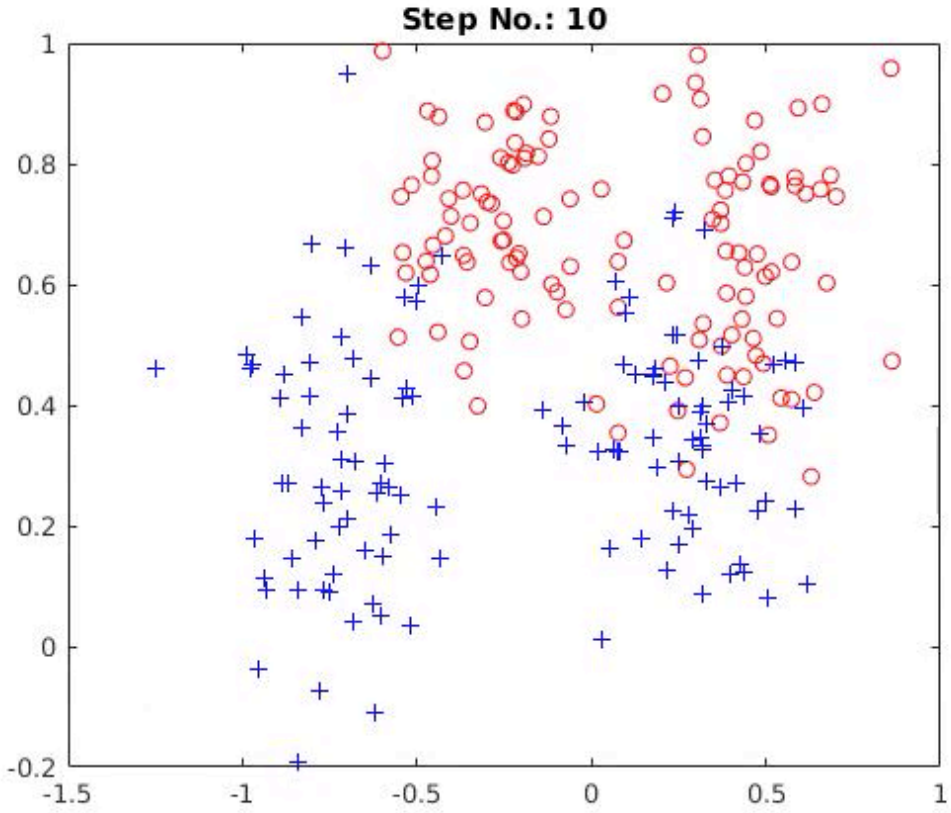
- El algoritmo de aprendizaje del perceptron

1. **w** ← inicializar con valores aleatorios
2. Repetir hasta que se alcanza criterio de paro:
   I. Por cada $\mathbf{x}_i \in D$
      a) $o_i \leftarrow \mathbf{w}\mathbf{x}_i + b$ // Estimar la predicción del perceptron
      b) $\Delta\mathbf{w} \leftarrow \eta(y_i - o_i)$ // Estimar la razón de cambio
      c) $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$ // Actualizar w
3. *Regresa* **w**

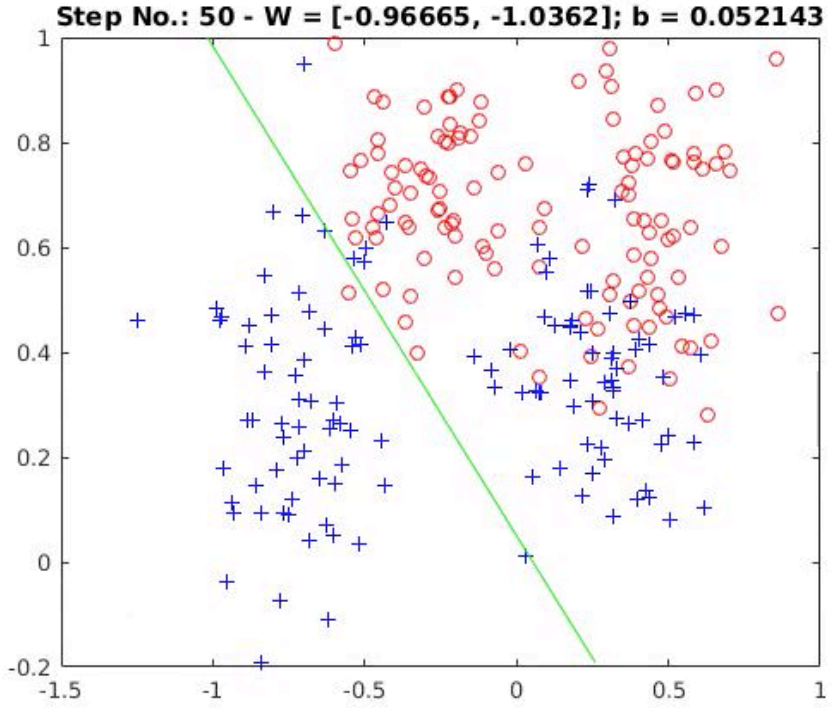
Intución ?

- Convergencia está garantizada (para problemas linealmente separables)

El perceptron en acción



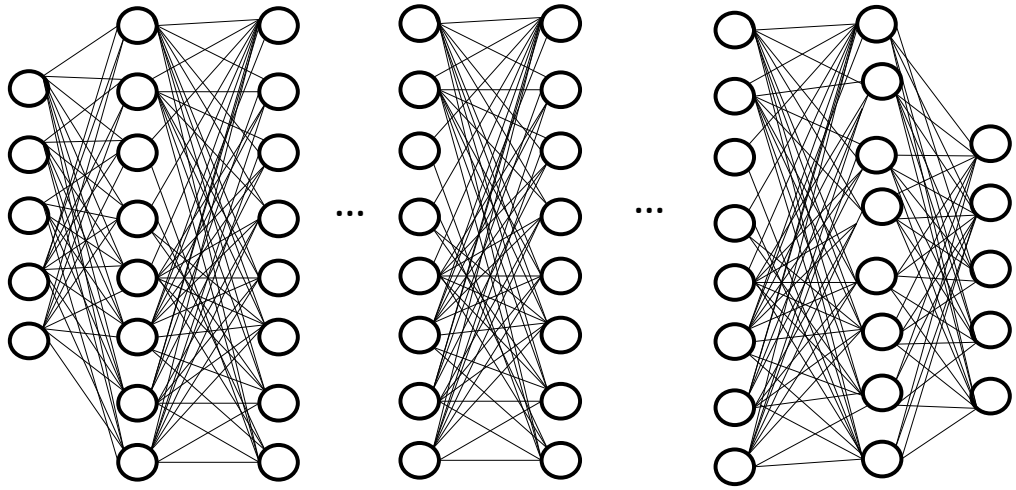
El perceptron en acción



Redes neuronales profundas

En breve…

- DL es una metodología del aprendizaje computacional que intenta resolver problemas de modelado construyendo modelos apilados en capas de parámetros de abstracción incremental



Redes neuronales profundas (DNNs)

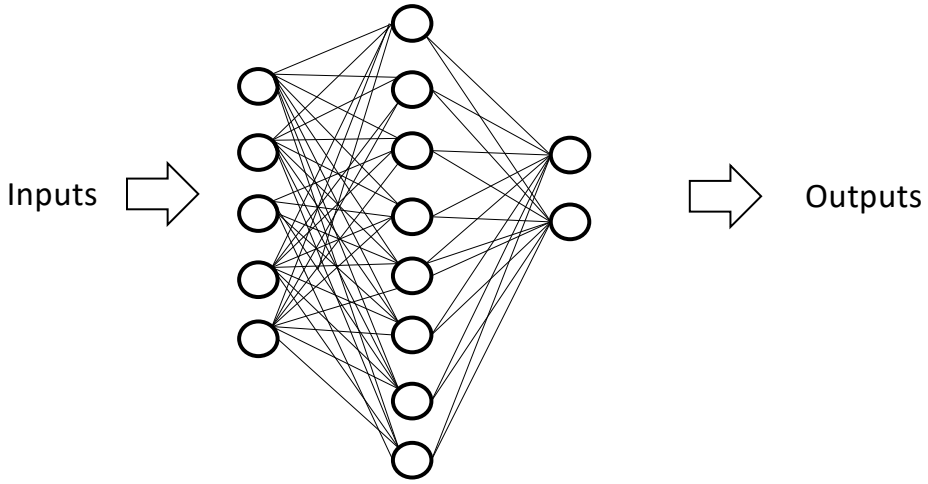
- Las redes neuronales representan el modelo *esencial* de aprendizaje profundo
- Por lo tanto, las redes neuronales *feedforward* (MLP) son la base del aprendizaje profundo
- Convencionalmente, se dice que una red es profunda si tiene al menos 2 capas ocultas

DNNs

$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

$$\phi(\mathbf{x}) = s(\mathbf{ux} + c)$$

- Una red neuronal tipo *feedforward* es un modelo que:
  - Aproxima funciones de la forma $y = f(x; \Theta)$
  - Se compone de múltiples funciones no lineales organizadas en capas
  - Las capas forman a su vez la red
  - La información fluye en una sola dirección, hacia adelante

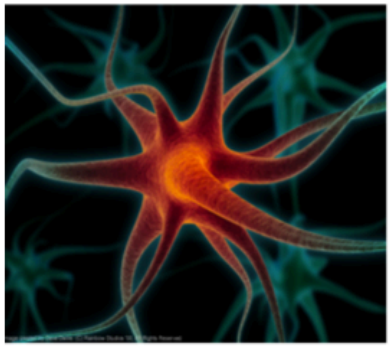


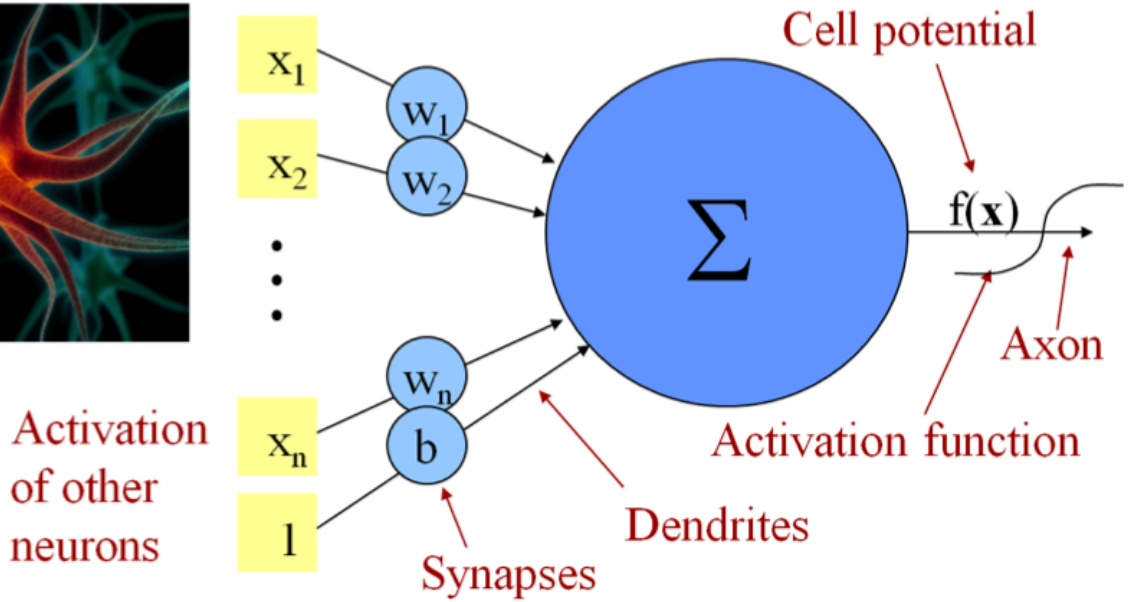
DNNs

$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

$$\phi(\mathbf{x}) = s(\mathbf{u}\mathbf{x} + c)$$

- Analogía con una neurona

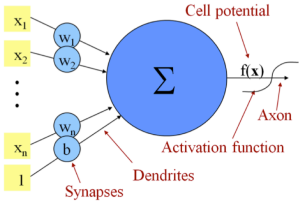


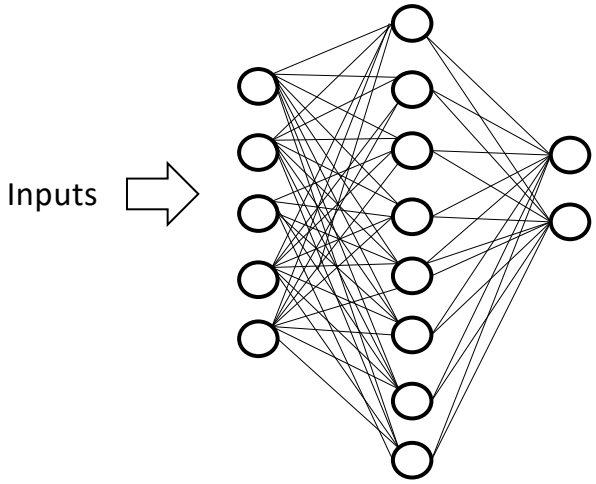


Diapositiva de I. Guyon

DNNs

- Una red neuronal tipo *feedforward* es un modelo que:
  - Aproxima funciones de la forma $y = f(x; \Theta)$
  - Se compone de múltiples funciones no lineales organizadas en capas
  - Las capas forman a su vez la red
  - La información fluye en una sola dirección, hacia adelante

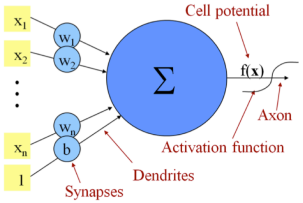


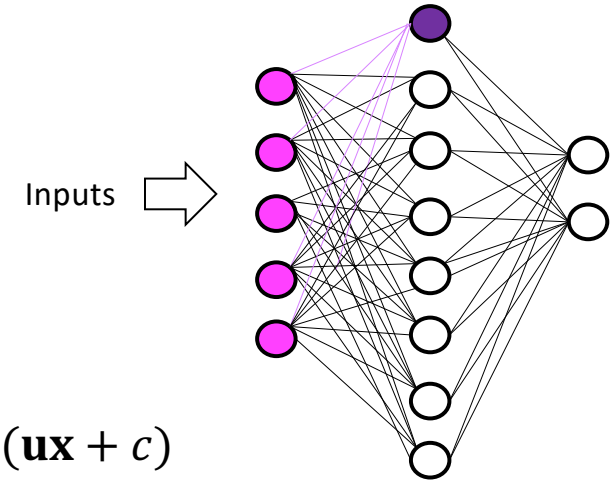


$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Una red neuronal tipo *feedforward* es un modelo que:
  - Aproxima funciones de la forma $y = f(x; \Theta)$
  - Se compone de múltiples funciones no lineales organizadas en capas
  - Las capas forman a su vez la red
  - La información fluye en una sola dirección, hacia adelante



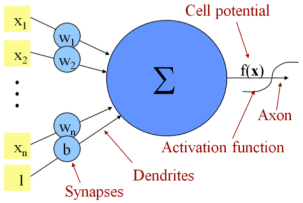


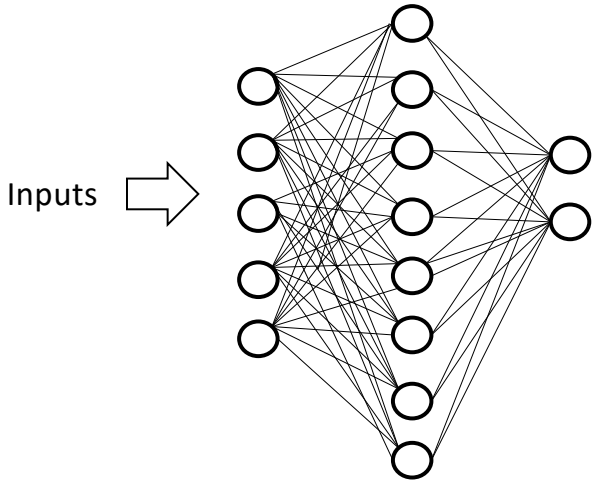
$$\phi(\mathbf{x}) = s(\mathbf{u}\mathbf{x} + c)$$

$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Una red neuronal tipo *feedforward* es un modelo que:
  - Aproxima funciones de la forma $y = f(x; \Theta)$
  - Se compone de múltiples funciones no lineales organizadas en capas
  - Las capas forman a su vez la red
  - La información fluye en una sola dirección, hacia adelante

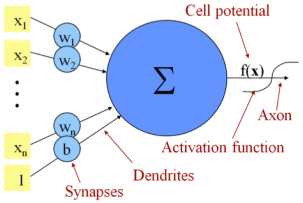


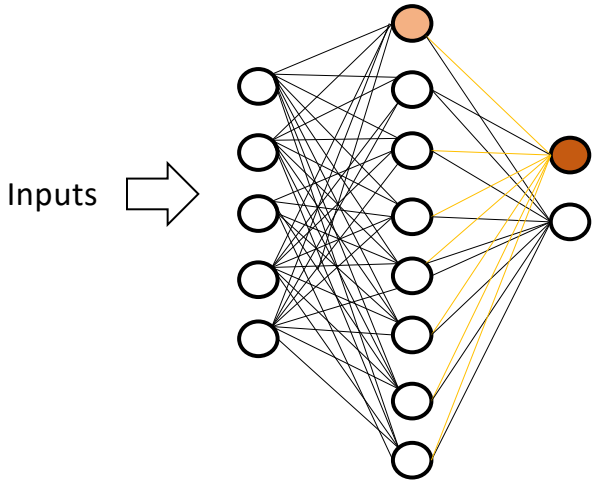


$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Una red neuronal tipo *feedforward* es un modelo que:
  - Aproxima funciones de la forma $y = f(x; \Theta)$
  - Se compone de múltiples funciones no lineales organizadas en capas
  - Las capas forman a su vez la red
  - La información fluye en una sola dirección, hacia adelante

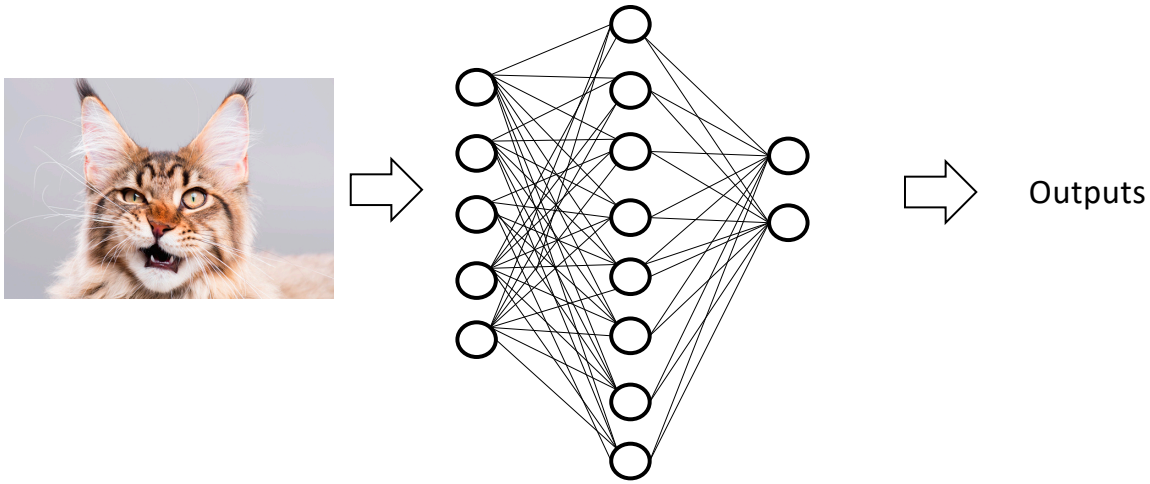




$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

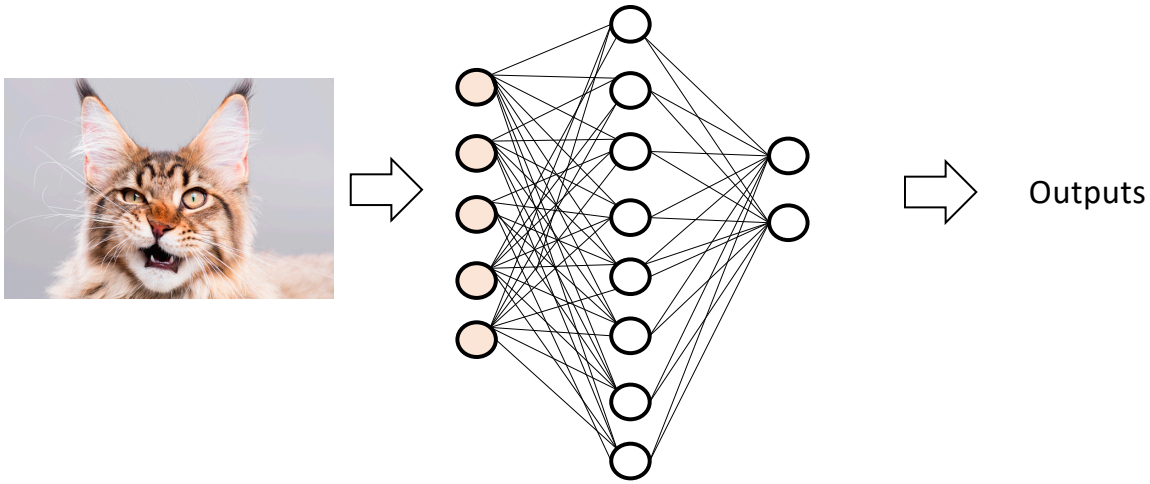
- Redes neuronales *feedforward*



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Redes neuronales *feedforward*

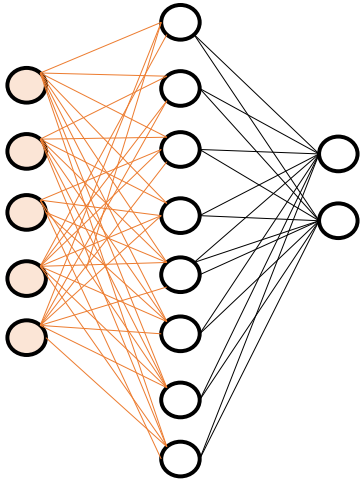


$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Redes neuronales *feedforward*

$$\phi(\mathbf{x}) = s(\mathbf{u}\mathbf{x} + c)$$



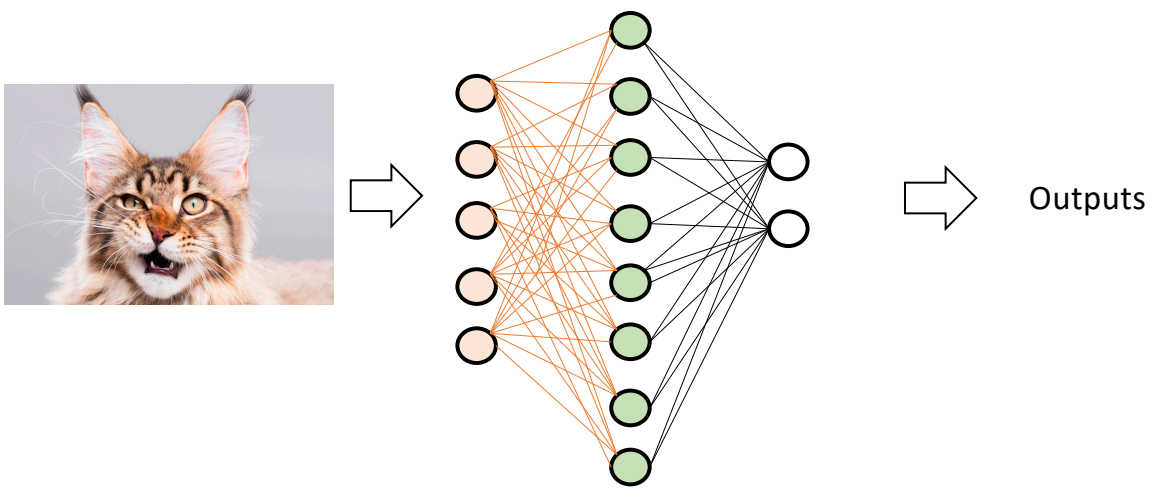
Outputs

$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Redes neuronales *feedforward*

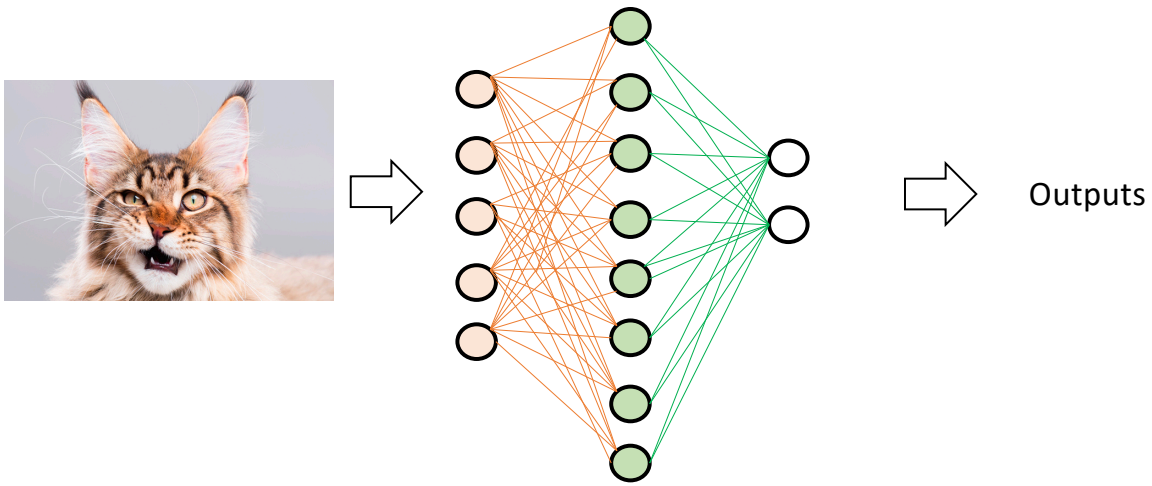
$$\phi(\mathbf{x}) = s(\mathbf{u}\mathbf{x} + c)$$



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

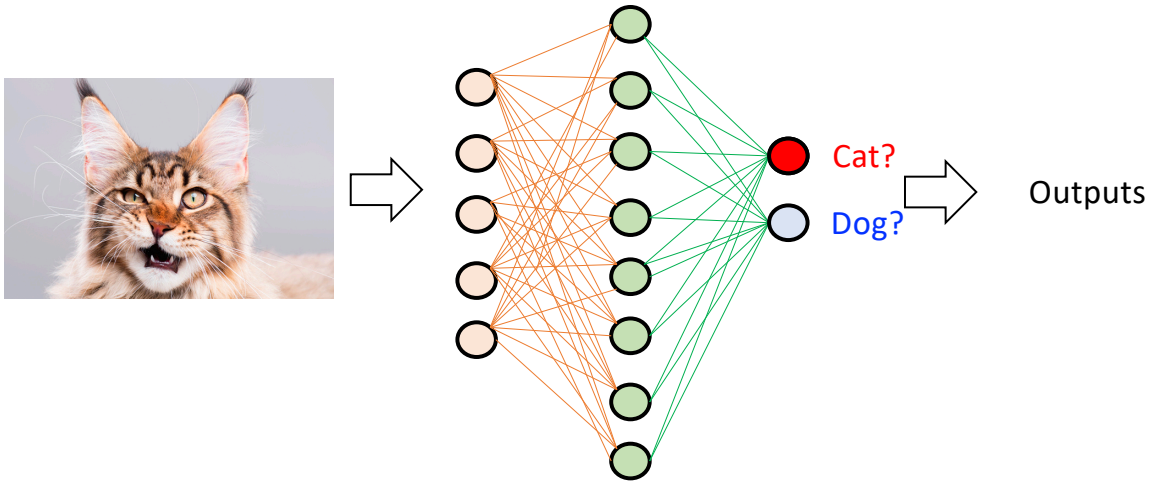
- Redes neuronales *feedforward*



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

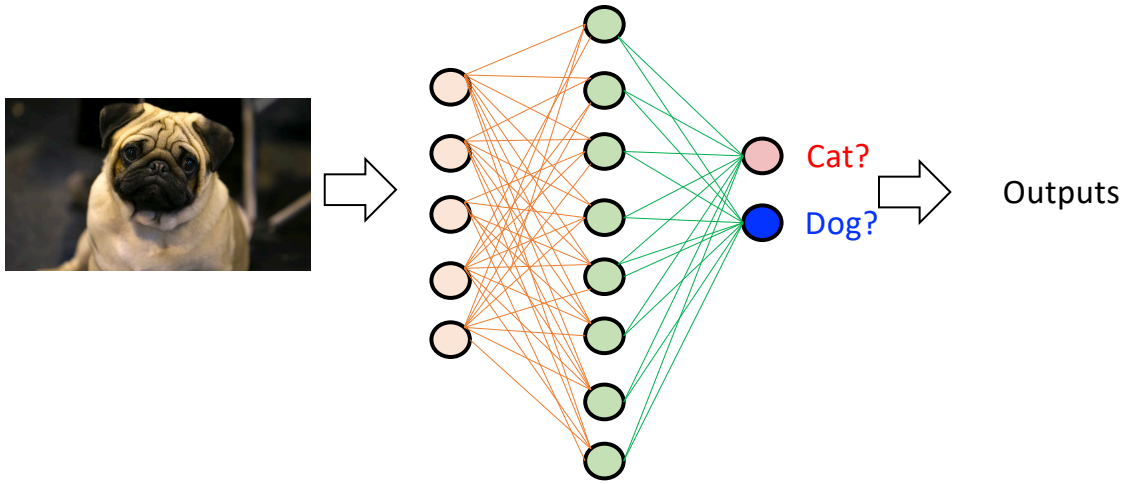
- Redes neuronales *feedforward*



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

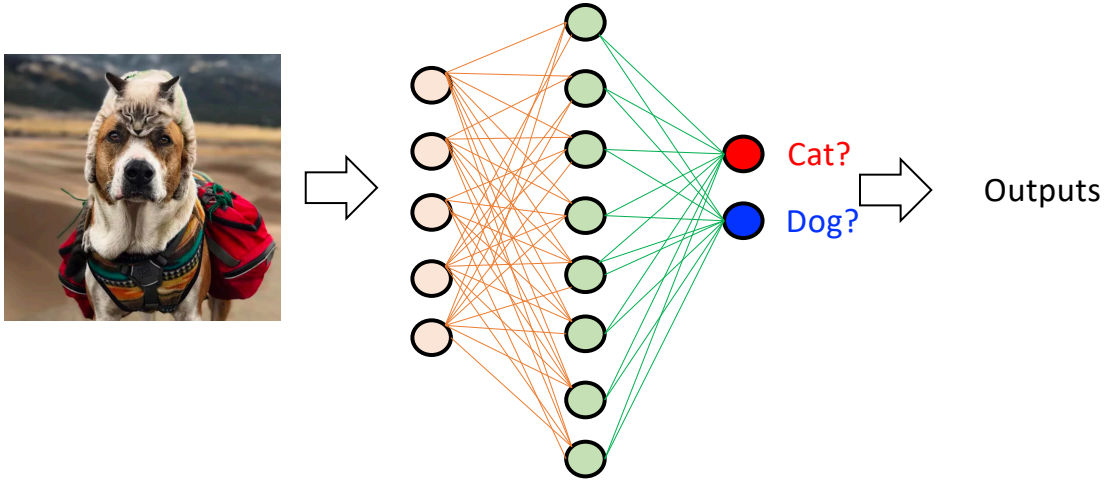
- Redes neuronales *feedforward*



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

- Redes neuronales *feedforward*



$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs

En este ejemplo la red neuronal está definida por las aristas del grafo, esto es, por matrices de pesos: **U** (5x8) y **W** (8x2)

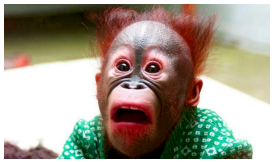
- Redes neuronales *feedforward*

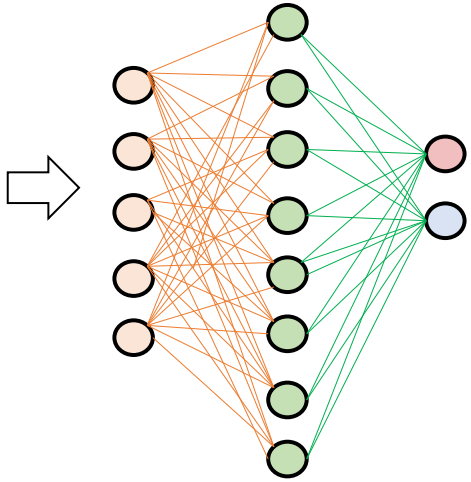
Variables - U

U

5x8 double

	1	2	3	4	5	6	7	8	9
1	-0.7203	1.1714	-1.5029	3.3133	0.0338	-0.3660	-0.2886	-1.0836	
2	-0.5102	0.2660	0.3880	0.8725	-0.5068	-1.6706	1.5959	1.0228	
3	-1.6597	-0.5933	-0.4178	1.8309	1.0331	0.1132	-0.3820	1.0903	
4	0.3005	0.3909	-0.2304	-2.3196	3.2656	-0.0441	0.4254	0.0461	
5	0.2737	-0.6014	0.0314	0.5855	0.3692	2.1029	1.5813	-0.9019	
6									
7									
8									
9									
10									
11									





Outputs

Variables - W

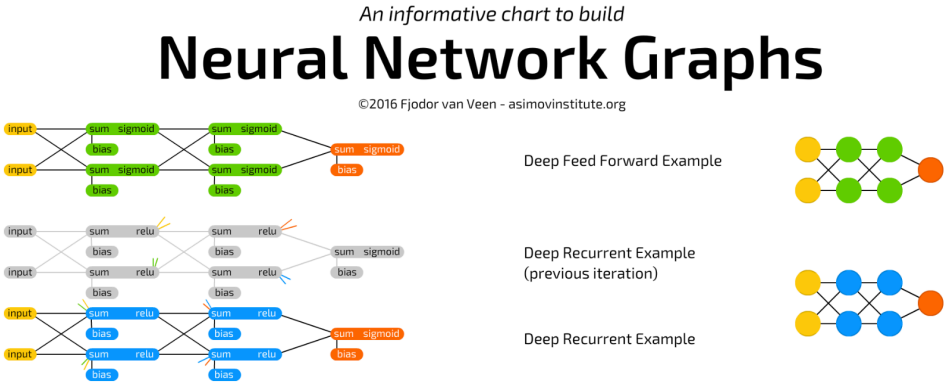
U W

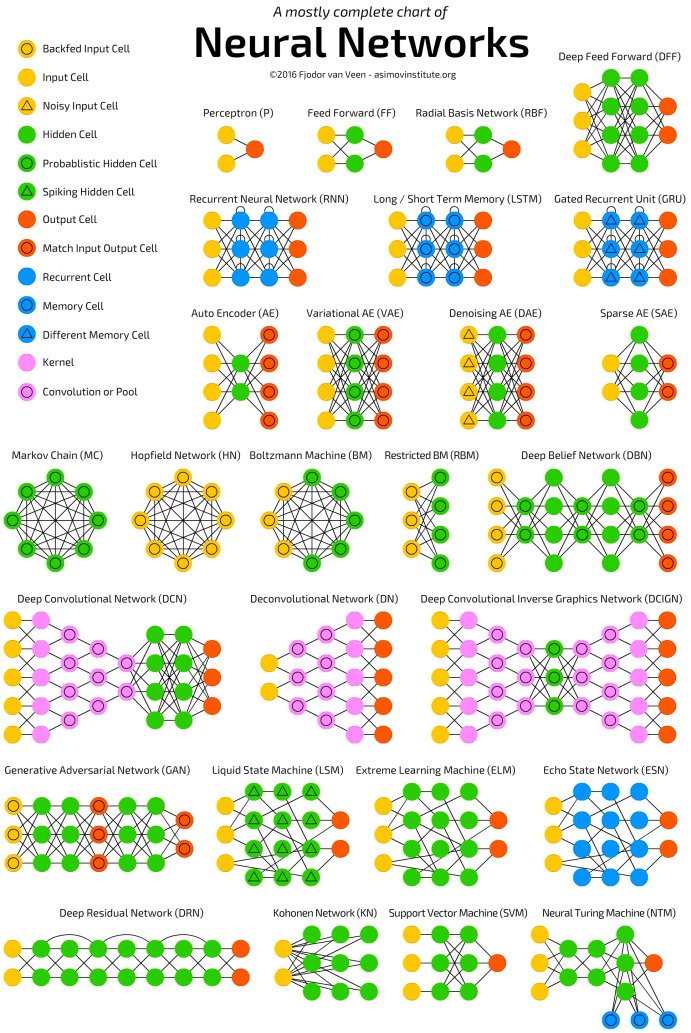
8x2 double

	1	2	3
1	0.5884	0.1427	
2	0.9775	0.6085	
3	0.2137	0.6286	
4	0.6291	0.0238	
5	0.7810	0.7787	
6	0.3571	0.4912	
7	0.9407	0.4116	
8	0.9616	0.7775	
9			
10			
11			
12			
13			

$$f(\mathbf{x}) = s(\mathbf{w}\phi(\mathbf{x}) + b)$$

DNNs





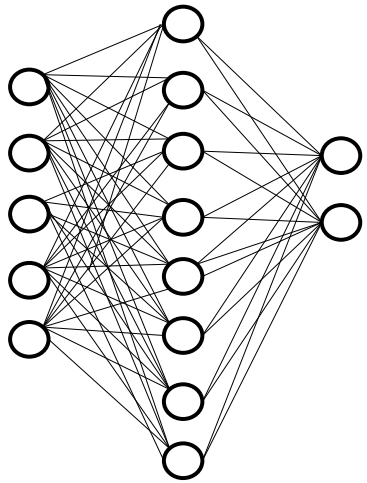
https://www.asimovinstitute.org/author/fjodorvanveen/

Desmenuzando una red neuronal

DNNs

- En general, las redes neuronales están construidas por muchas unidades similares al perceptron (unidades lineales activadas por una función no lineal que es diferenciable)

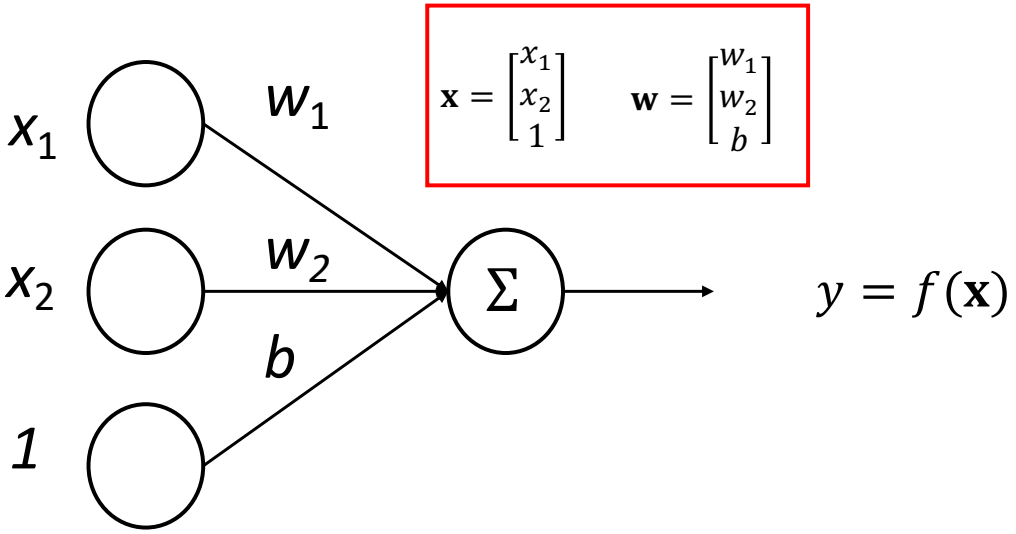
- **Perceptron:** Clasificador lineal simple que puede resolver problemas de clasificación linealmente separables (ancestro de DNNs y de la SVM)



El perceptron

- Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\ldots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$
- El perceptron aprende una función de decisión de la forma:

$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$

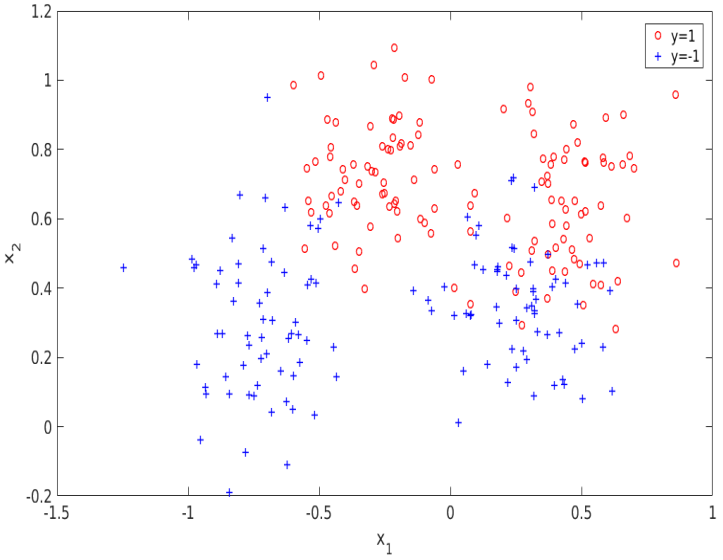


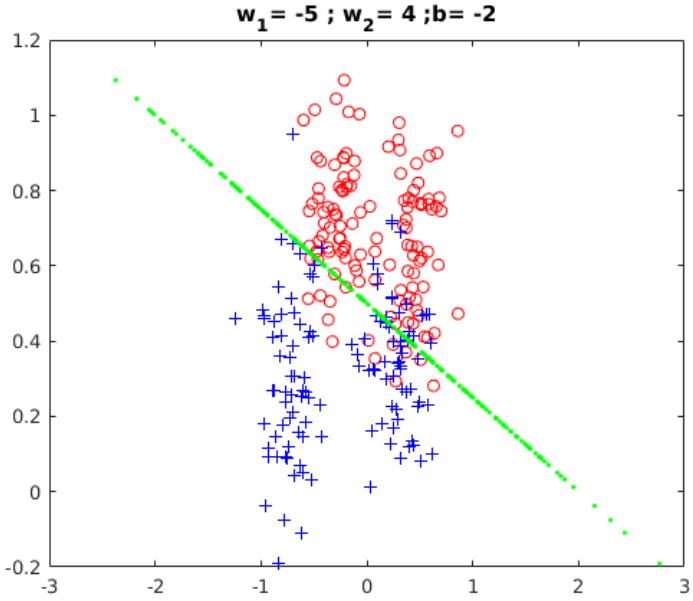
$$y = w_1 x_1 + w_2 x_2 + b$$

El perceptron

- Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\ldots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$
- El perceptron aprende una función de decisión de la forma:

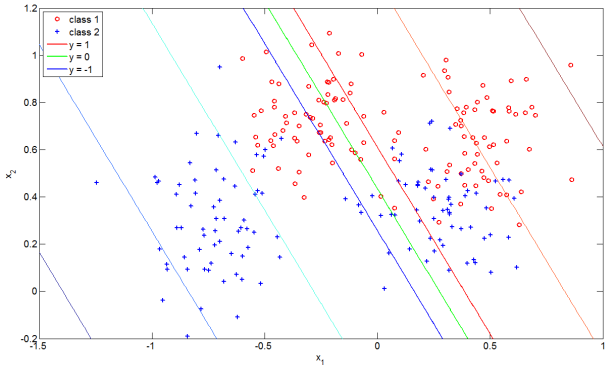
$$f(\mathbf{x}) = sign(\mathbf{w}\mathbf{x} + b), \text{ with } \mathbf{w} \in \mathbb{R}^d$$



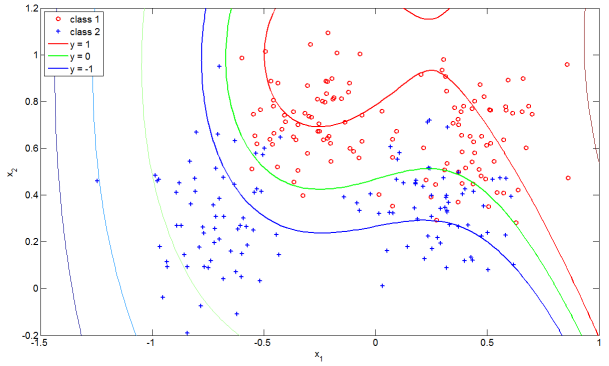


Unidades no lineales

- El problema con estos modelos lineales es que solamente pueden aprender funciones de este tipo



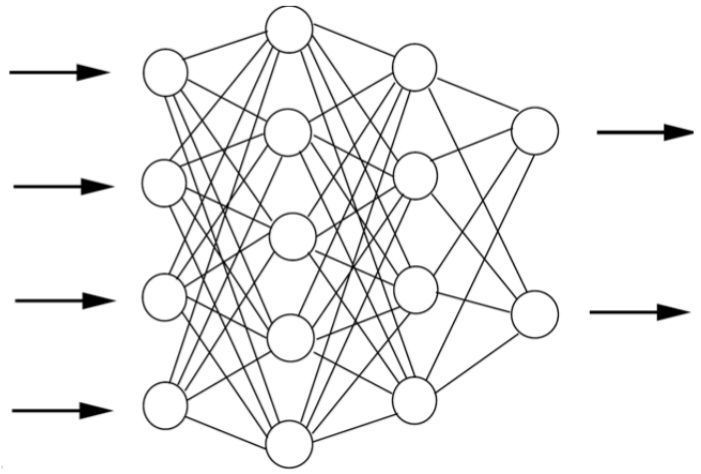
- Alternativas:
  - Mapear los datos a otro espacio (no lineal) donde el problema sea linealmente separable



Unidades no lineales

- Una opción es apilar múltiples capas de unidades lineales
  - Aún así, solo es posible aprender funciones lineales

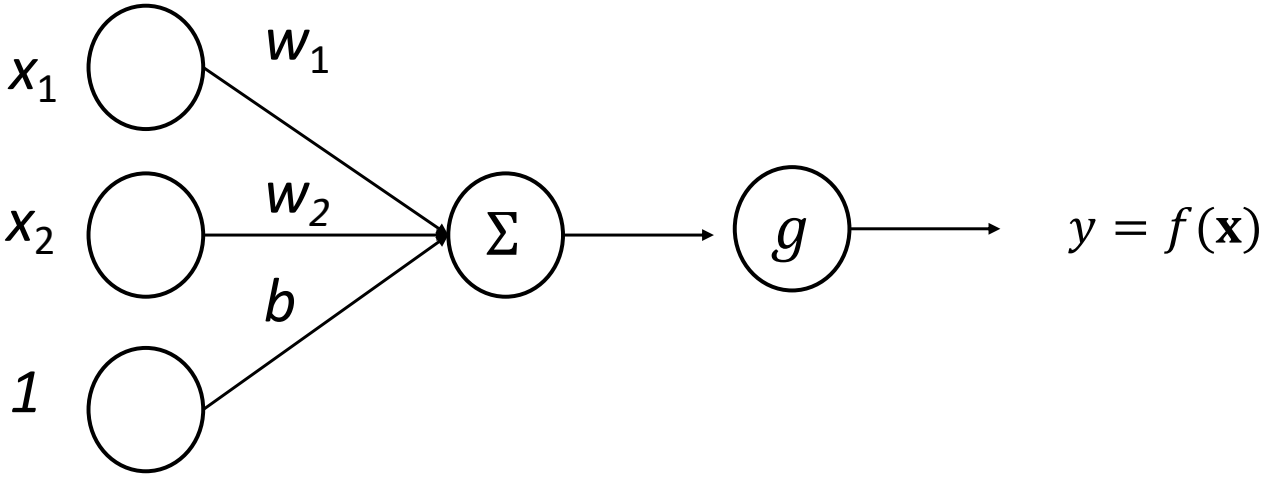
- Idea: Apilar múltiples capas de unidades lineales activadas con funciones no lineales



Unidades no lineales

- Apilar múltiples capas de unidades lineales activadas con funciones no lineales

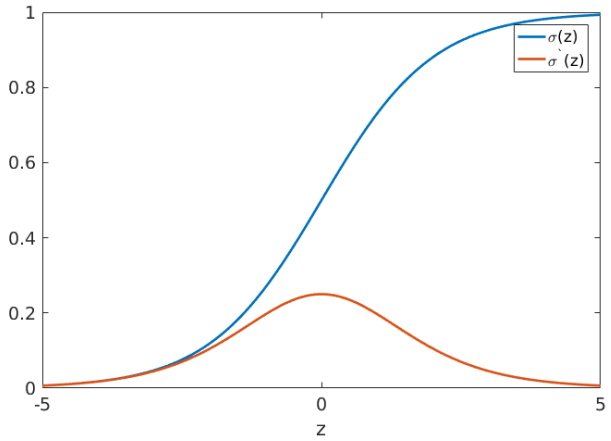
Se añade una función no lineal a la salida



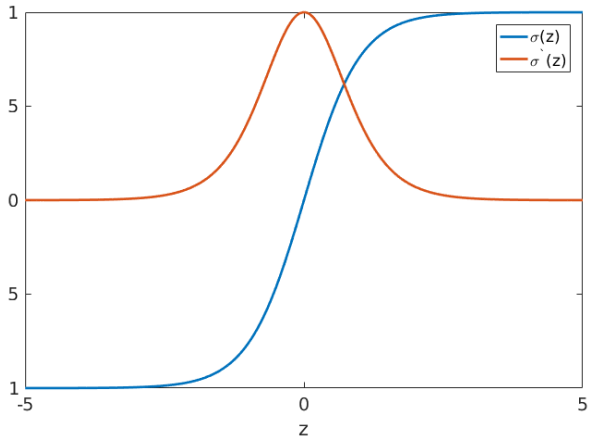
$$y = g(w_1 x_1 + w_2 x_2 + b)$$

Unidades no lineales

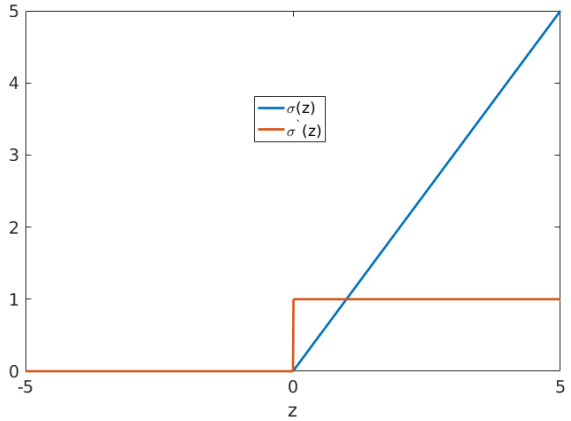
- Algunas funciones de activación:



Sigmoide

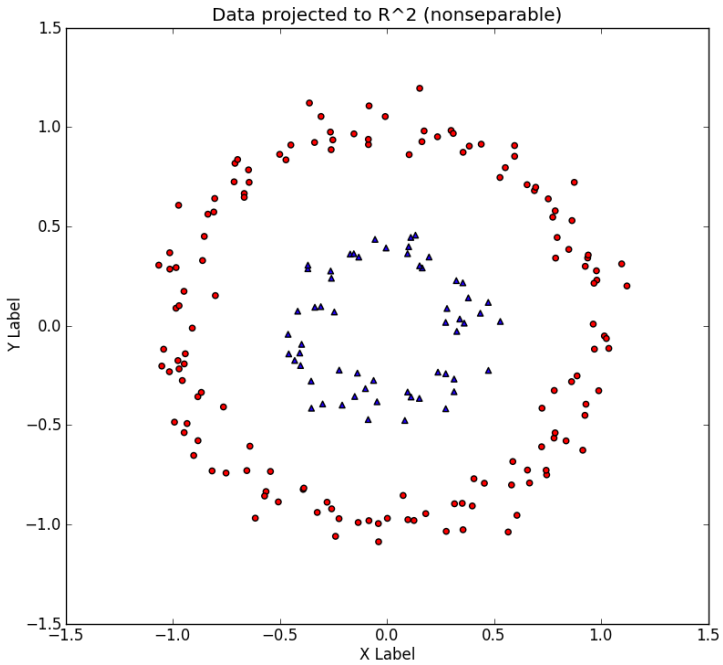


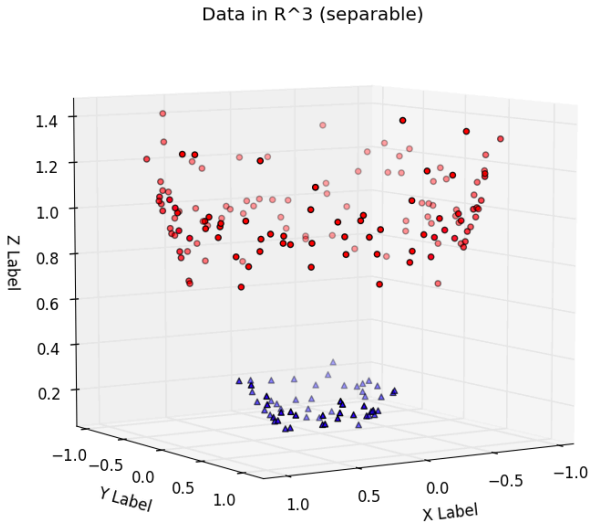
Tangente hiperbólica



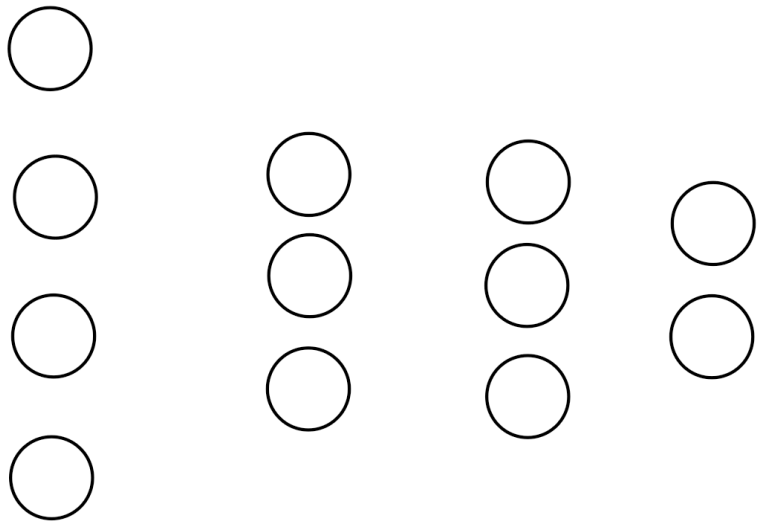
ReLU

Unidades no lineales

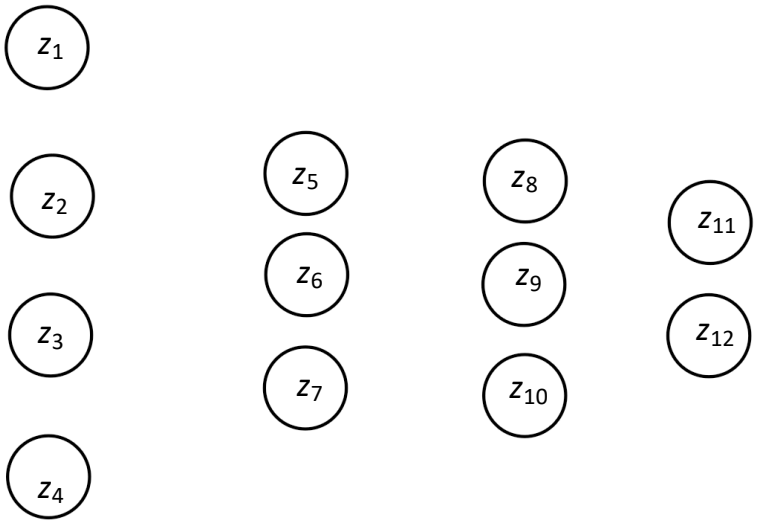




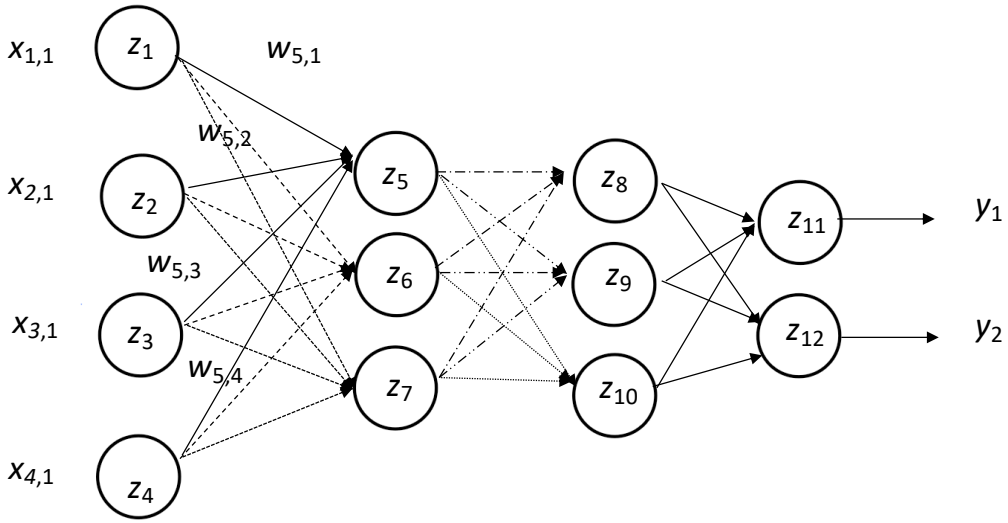
DNNs



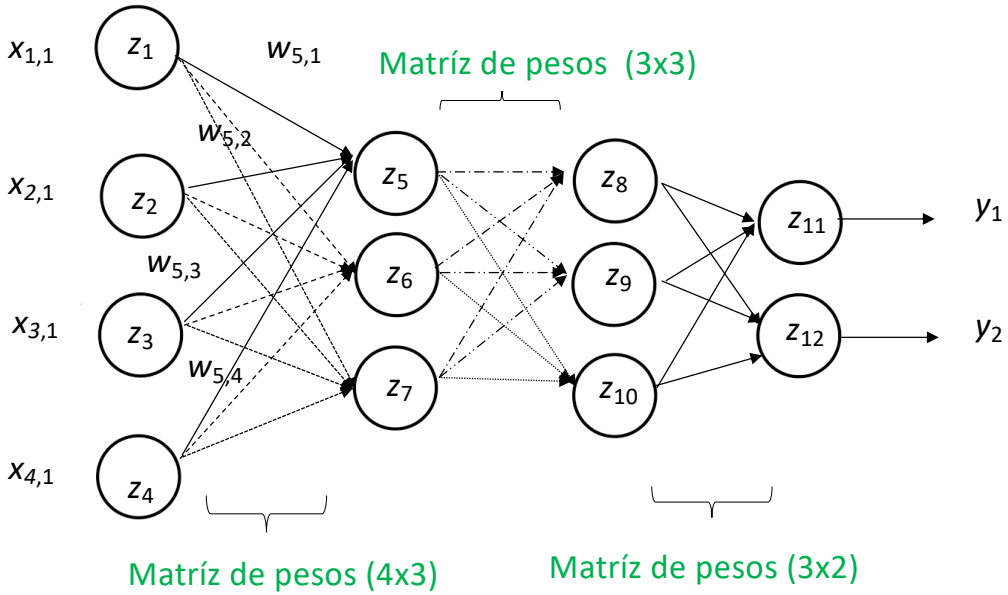
DNNs



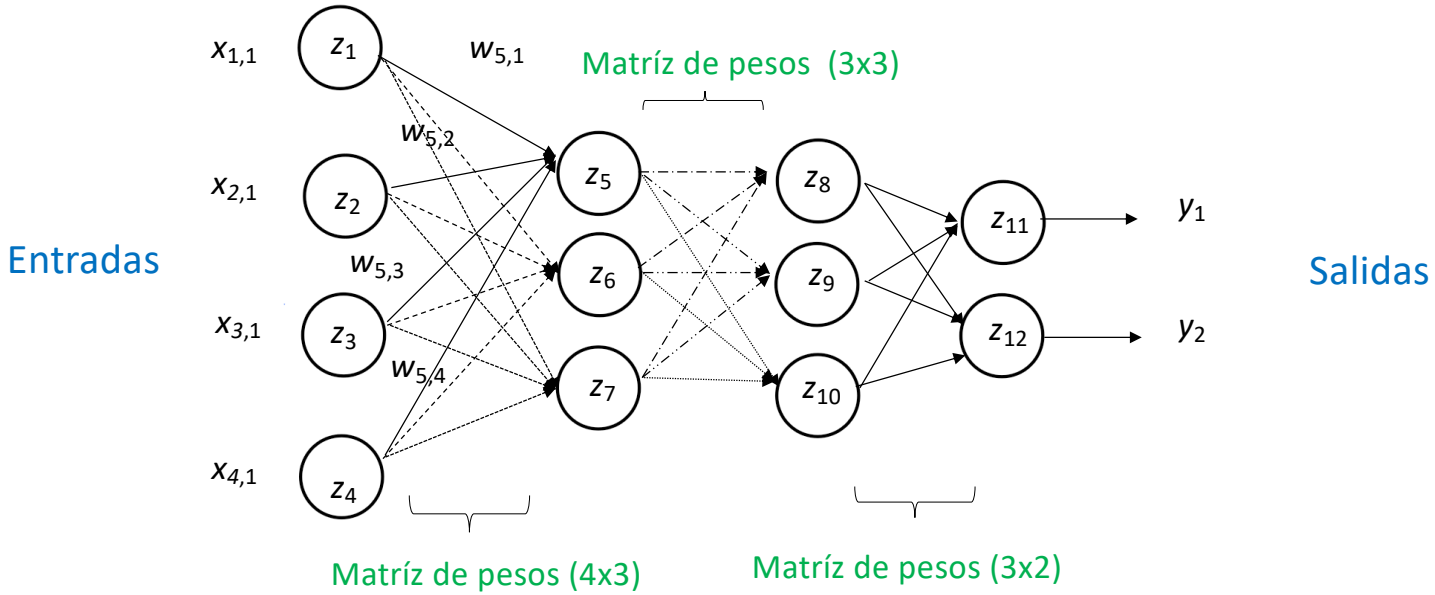
DNNs


$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
y_1
y_2

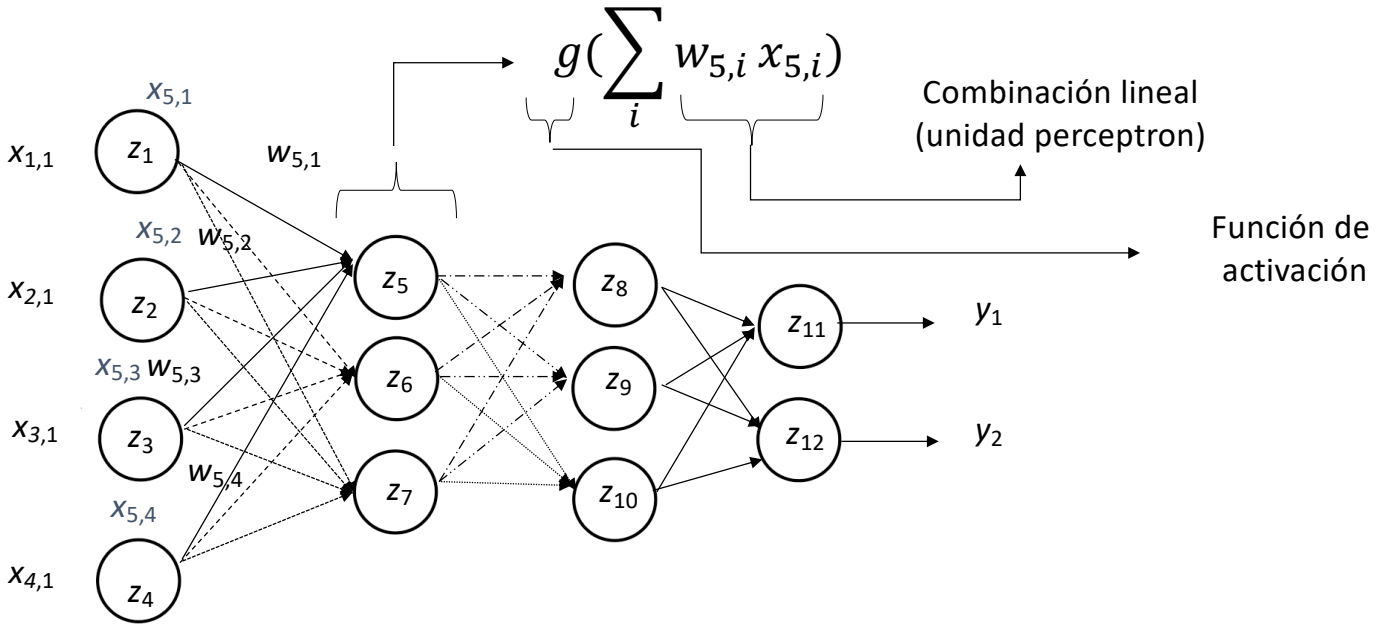
DNNs


$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
Matríz de pesos (3x3)
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
y_1
y_2
Matríz de pesos (4x3)
Matríz de pesos (3x2)

DNNs

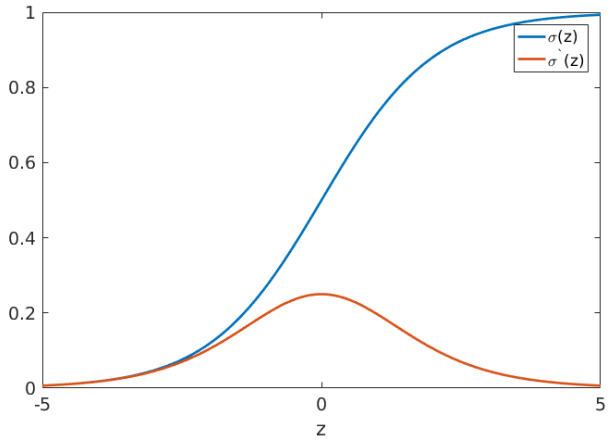

Entradas
$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
Matríz de pesos (4x3)
z_5
z_6
z_7
Matríz de pesos (3x3)
z_8
z_9
z_{10}
Matríz de pesos (3x2)
z_{11}
z_{12}
y_1
y_2
Salidas

DNNs

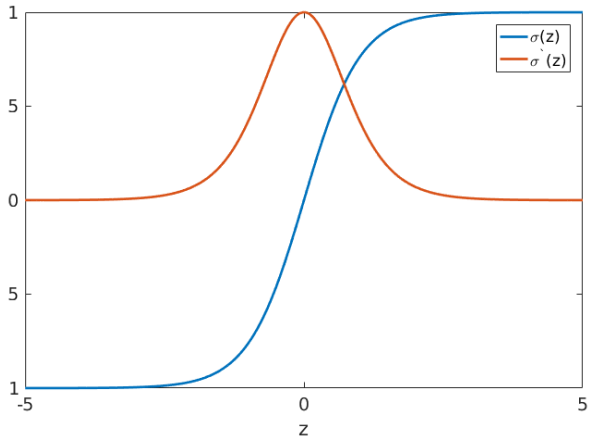

$g(\sum_i w_{5,i}\, x_{5,i})$
Combinación lineal
(unidad perceptron)
Función de
activación
$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
$x_{5,1}$
$x_{5,2}$
$x_{5,3}$
$x_{5,4}$
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
z_1
z_2
z_3
z_4
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
y_1
y_2

Unidades no lineales

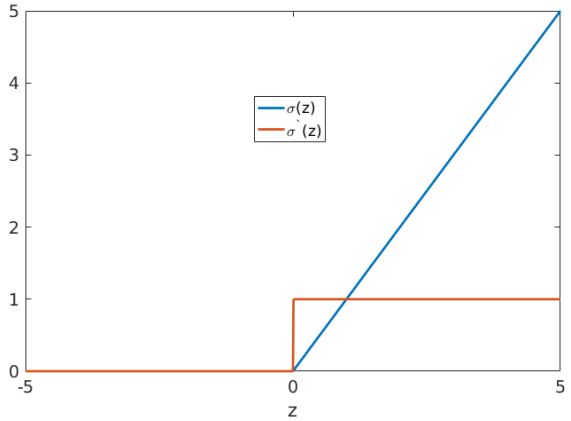
- Algunas funciones de activación:



Sigmoide

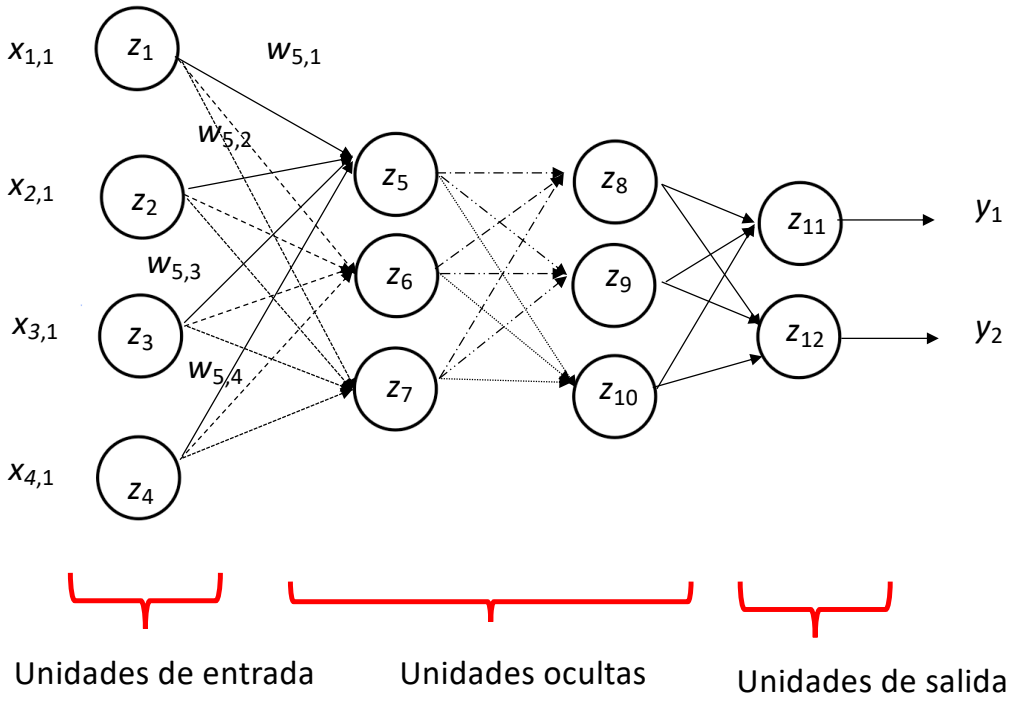


Tangente hiperbólica

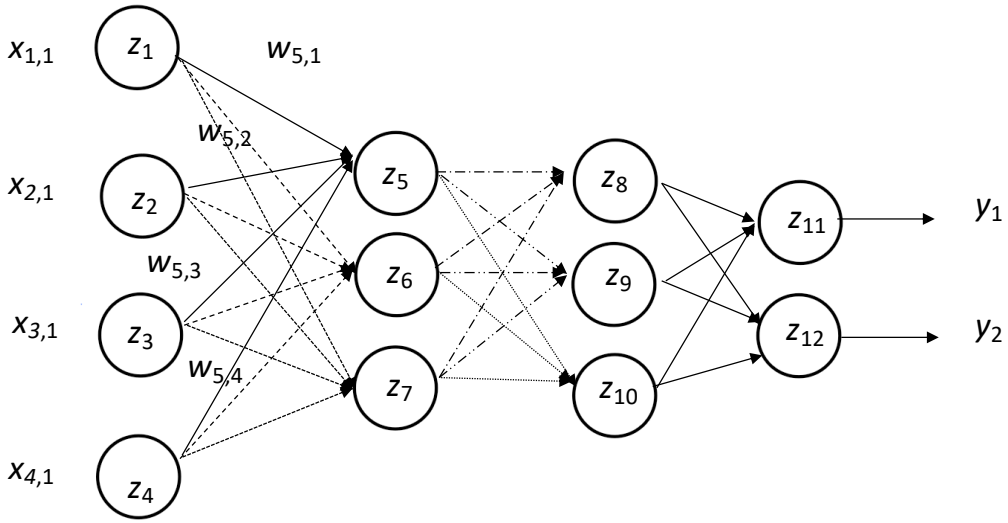


ReLU

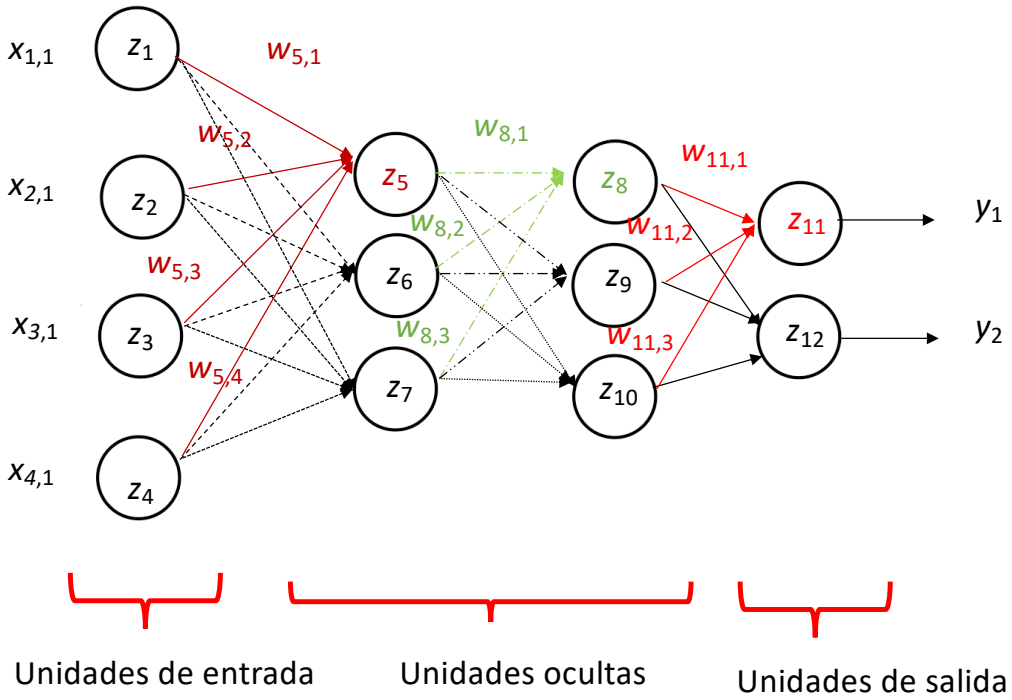
DNNs


$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
y_1
y_2
Unidades de entrada
Unidades ocultas
Unidades de salida

DNNs


$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
y_1
y_2

DNNs


$x_{1,1}$
$x_{2,1}$
$x_{3,1}$
$x_{4,1}$
z_1
z_2
z_3
z_4
z_5
z_6
z_7
z_8
z_9
z_{10}
z_{11}
z_{12}
$w_{5,1}$
$w_{5,2}$
$w_{5,3}$
$w_{5,4}$
$w_{8,1}$
$w_{8,2}$
$w_{8,3}$
$w_{11,1}$
$w_{11,2}$
$w_{11,3}$
y_1
y_2
Unidades de entrada
Unidades ocultas
Unidades de salida

DNNs

$$z_5 = g(\sum_i w_{5,i}\, x_{5,i})$$

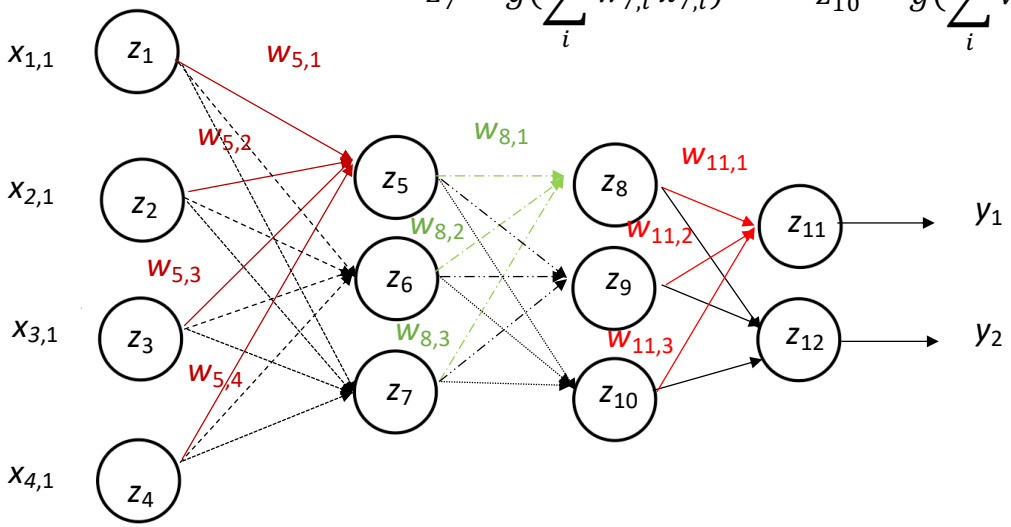
$$z_6 = g(\sum_i w_{6,i}\, x_{6,i})$$

$$z_7 = g(\sum_i w_{7,i}\, x_{7,i})$$

$$\boxed{z_8 = g(\sum_i w_{8,i}\, x_{8,i})}$$

$$z_9 = g(\sum_i w_{9,i}\, x_{9,i})$$

$$z_{10} = g(\sum_i w_{10,i}\, x_{10,i})$$



$$z_{11} = g(\sum_i w_{11,i}\, x_{11,i})$$

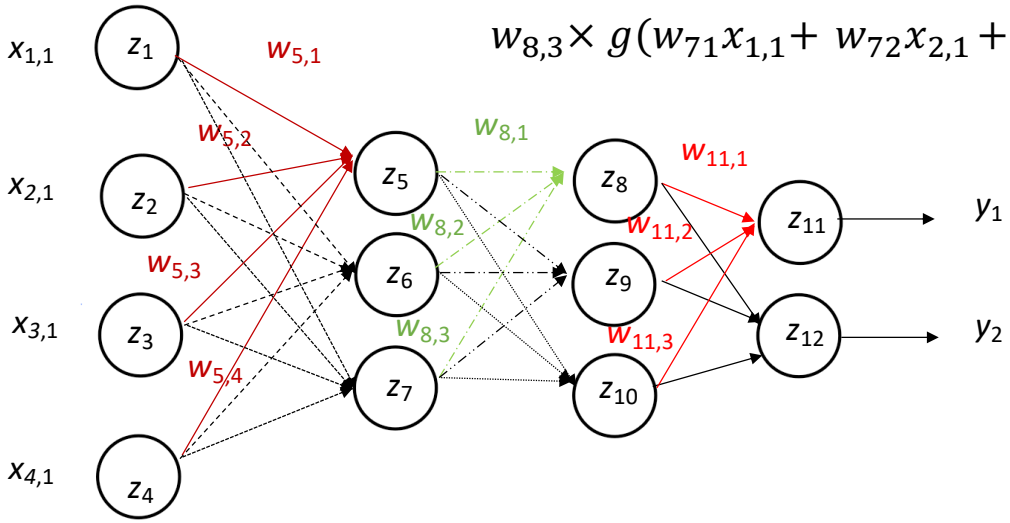
$$z_{12} = g(\sum_i w_{12,i}\, x_{12,i})$$

DNNs

$$z_8 = g(w_{8,1}z_5 + w_{8,2}z_6 + w_{8,3}z_7)$$

$$z_8 = g\big(w_{8,1}\times g(w_{51}x_{1,1}+ w_{52}x_{2,1} + w_{53}x_{3,1} + w_{54}x_{4,1}\big) \dots$$
$$w_{8,2}\times g(w_{61}x_{1,1}+ w_{62}x_{2,1} + w_{63}x_{3,1} + w_{64}x_{4,1}) + \cdots$$
$$w_{8,3}\times g(w_{71}x_{1,1}+ w_{72}x_{2,1} + w_{73}x_{3,1} + w_{74}x_{4,1}))$$

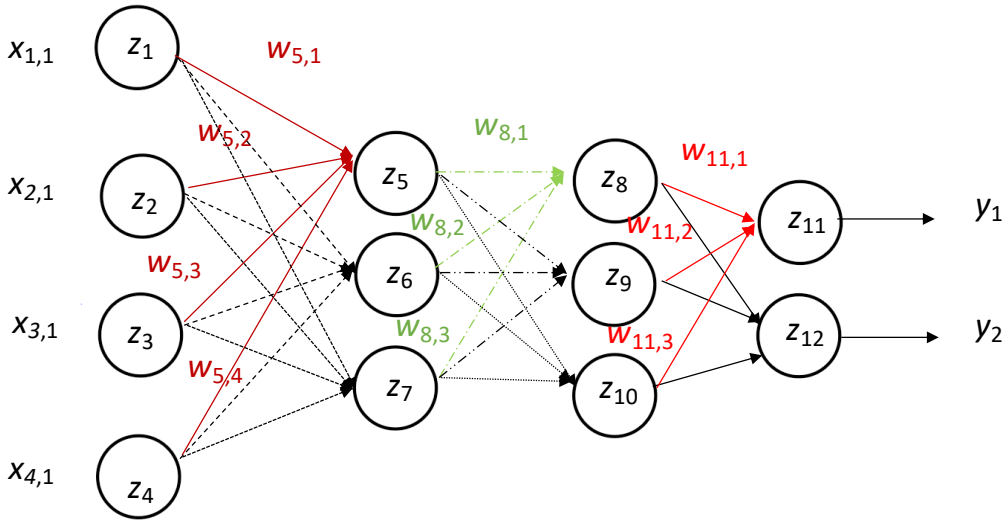
$$z_8 = g(\sum_i w_{8,i}\, x_{8,i})$$



$$z_8 = g(\sum_i w_{8,i}\, g(\sum_{j\in inputs - z_8} z_j))$$

$$inputs - z_i = \{z_j | \exists\, z_j \rightarrow z_i\}$$

DNNs



$$z_8 = g(\sum_i w_{8,i}\, g(\sum_{j \in inputs - z_8} z_j))$$

$$inputs - z_i = \{z_j | \exists\ z_j \rightarrow z_i\}$$

Entrenamiento de una red neuronal

DNNs

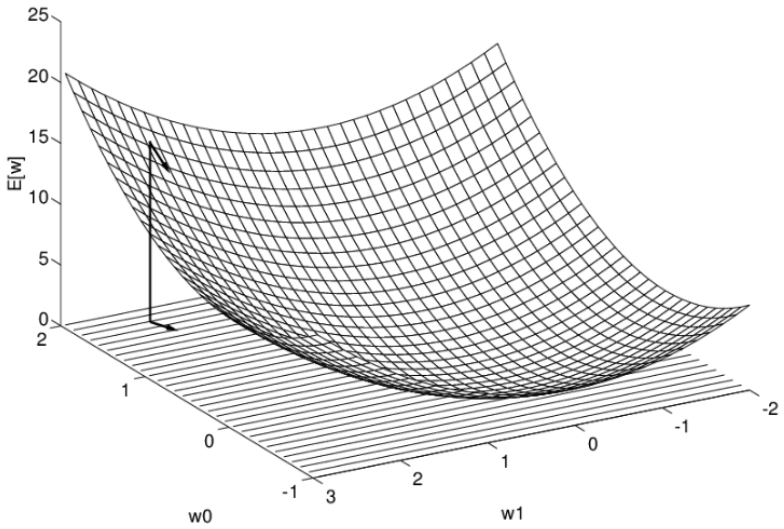
- Observaciones:
  - La red debe aprender a mapear entradas a salidas, a partir de ejemplos
  - La información siempre fluye hacia adelante
  - Las funciones de activación podrían ser diferentes para cada unidad
  - Las unidades de entrada son fijas (dadas por el problema)
  - Las unidades de salida están fijas para datos de entrenamiento y son desconocidas para cualquier otra muestra
  - Los parámetros $\boldsymbol{W}$ deben estimarse
  - El número de capas y unidades en cada capa son definidas por el diseñador
  - Las redes neuronales son aproximadores universales de funciones

DNNs

- El entrenamiento de la red consiste en aprender los pesos de la red que minimizan un estimado del error
  - Cuántos parámetros?
  - Cómo determinar el valor de los pesos?
  - Qué criterio adoptar?

DNNs

- El método de retro propagación con gradiente descendente se usa para aprender pesos
  - Usar gradiente descendente para aprender los pesos que mejor se ajustan a los datos *D*
  - Uso de la regla de la cadena para distribuir el gradiente del error con respecto a los pesos

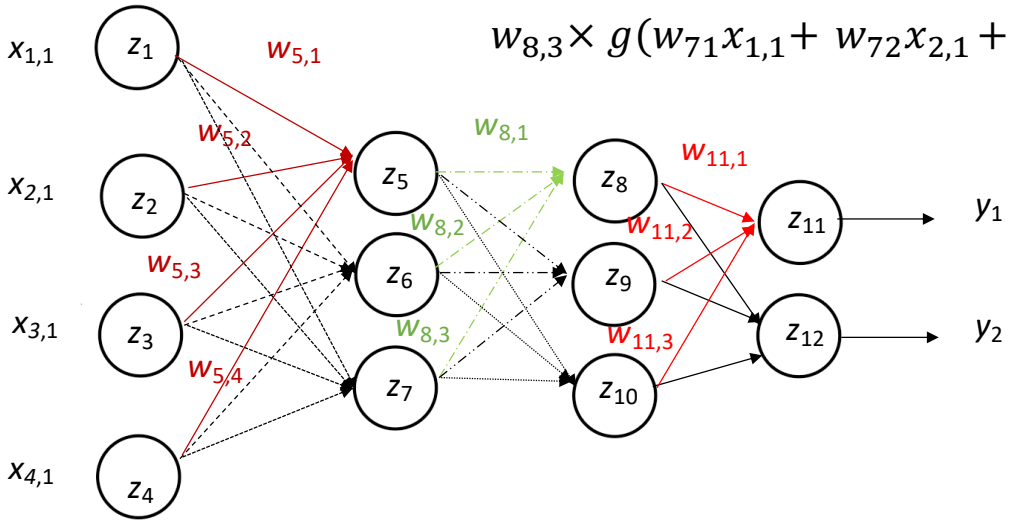


DNNs

$$z_8 = g(w_{8,1}z_5 + w_{8,2}z_6 + w_{8,3}z_7)$$

$$z_8 = g\big(w_{8,1}\times g(w_{51}x_{1,1}+ w_{52}x_{2,1} + w_{53}x_{3,1} + w_{54}x_{4,1}\big) \dots$$
$$w_{8,2}\times g(w_{61}x_{1,1}+ w_{62}x_{2,1} + w_{63}x_{3,1} + w_{64}x_{4,1}) + \cdots$$
$$w_{8,3}\times g(w_{71}x_{1,1}+ w_{72}x_{2,1} + w_{73}x_{3,1} + w_{74}x_{4,1}))$$

$$z_8 = g(\sum_i w_{8,i}\, x_{8,i})$$

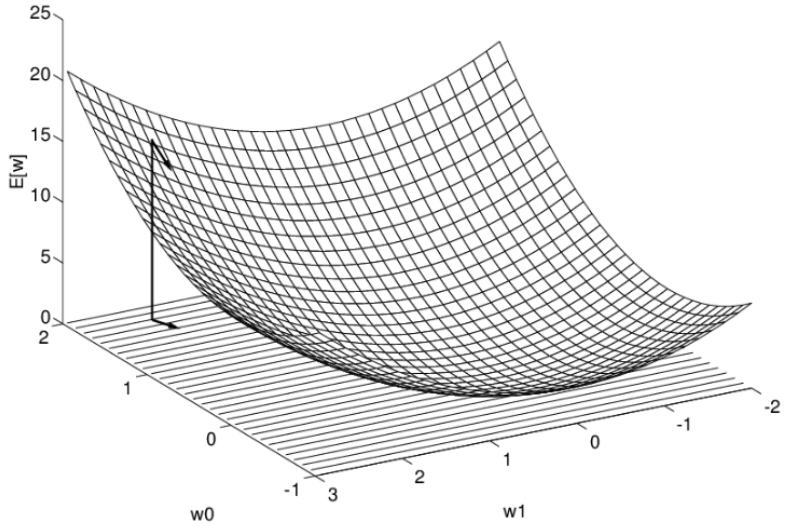


$$z_8 = g(\sum_i w_{8,i}\, g(\sum_{j\in inputs - z_8} z_j))$$

$$inputs - z_i = \{z_j | \exists\, z_j \rightarrow z_i\}$$

DNNs

- En términos generales:
  1. Inicializar todos los pesos de la red de manera aleatoria
  2. Alimentar los ejemplos de entrenamiento a la red y calcular el error
  3. Calcular el gradiente de la función de error con respecto a cada peso
  4. Actualizar el valor de pesos de acuerdo a (3)
  5. Repetir 2-4 hasta convergencia o alcanzar criterio de paro



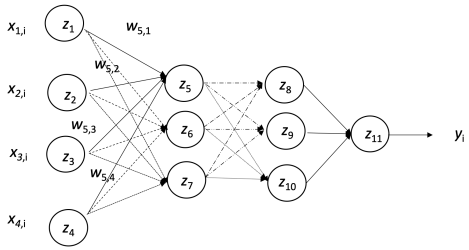
DNNs

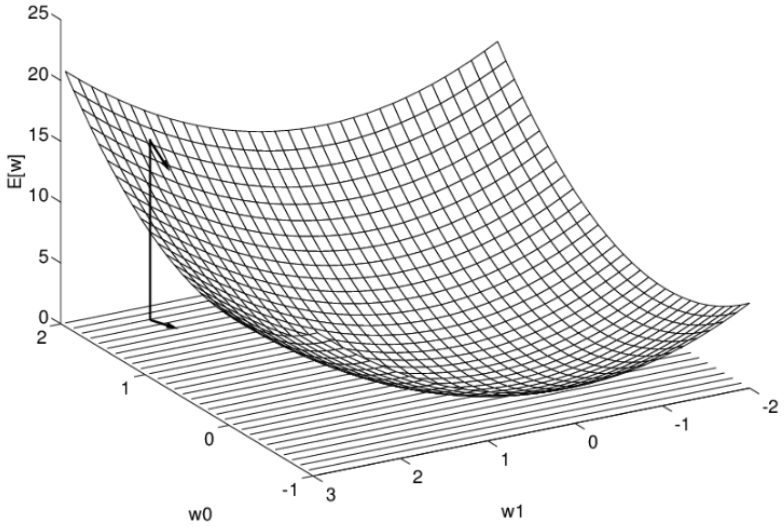
- Requerimos:
  - Definir función de error *E* (dependiente del problema)

$$E(\mathbf{w}) = \min \frac{1}{2} \sum_{i=1}^{m} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

  - Calcular el gradiente de E con respecto a W

$$\frac{d\,E(\mathbf{w})}{d\mathbf{w}}$$





DNNs

- Queremos encontrar los pesos **w** que minimizan:

$$E(\mathbf{w}) = \min \frac{1}{2} \sum_{i=1}^{m} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

- Usando gradiente descendente estocástico queremos actualizar los pesos con cada instancia *k* de la siguiente forma:

$$w_{j,i} \leftarrow w_{j,i} + \Delta w_{j,i} \qquad \Delta w_{j,i} = -\eta \frac{E_k(w_{j,i})}{dw_{j,i}}$$

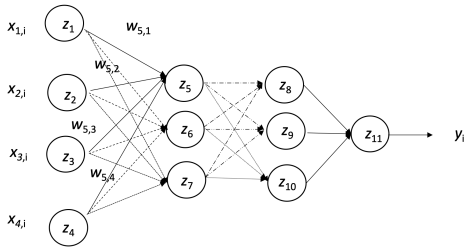
DNNs

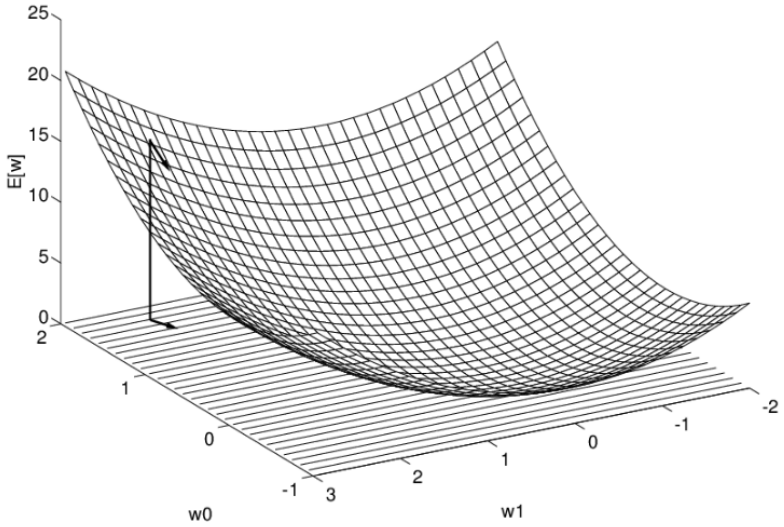
- Requerimos:
  - Definir función de error *E* (dependiente del problema)

$$E(\mathbf{w}) = \min \frac{1}{2} \sum_{i=1}^{m} (y_i - f_{\mathbf{w}}(\mathbf{x}_i))^2$$

  - Calcular el gradiente de E con respecto a W

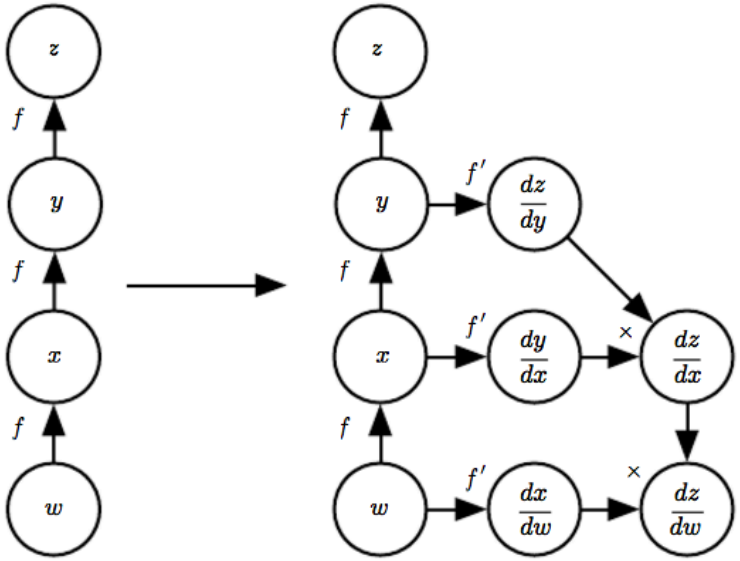
$$\frac{d\,E(\mathbf{w})}{d\mathbf{w}}$$



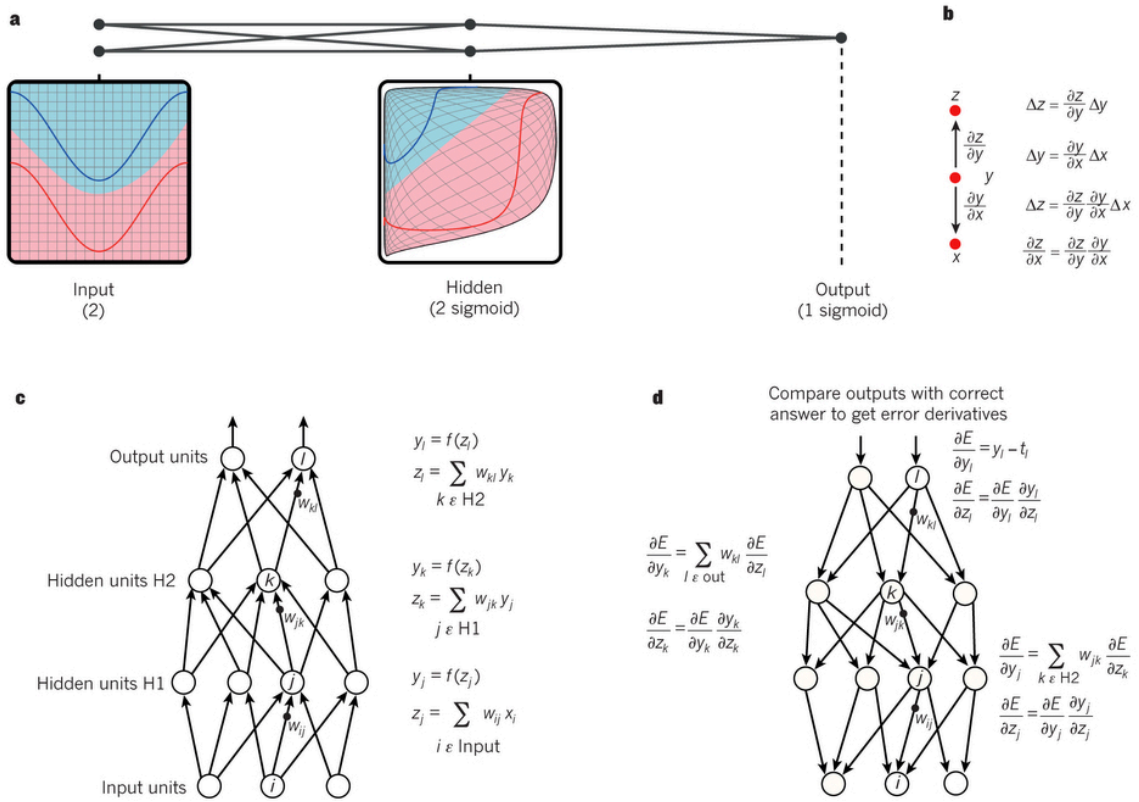


Regla de la cadena

Regla de la cadena

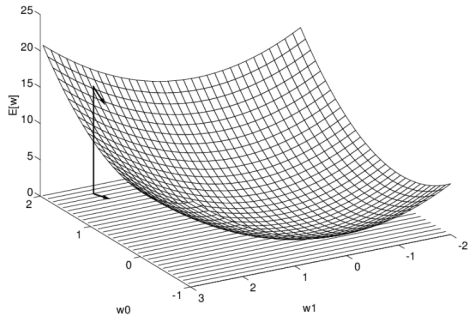


DNNs y la regla de la cadena



DNNs - entrenamiento

- Retro propagación y gradiente descendente



T. Mitchell. **Machine Learning**, McGrawHill 1997,

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i *into unit* j *is denoted* x_{ji}, *and the weight from unit* i *to unit* j *is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in $training\_examples$, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \qquad \text{(T4.3)}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \qquad \text{(T4.4)}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

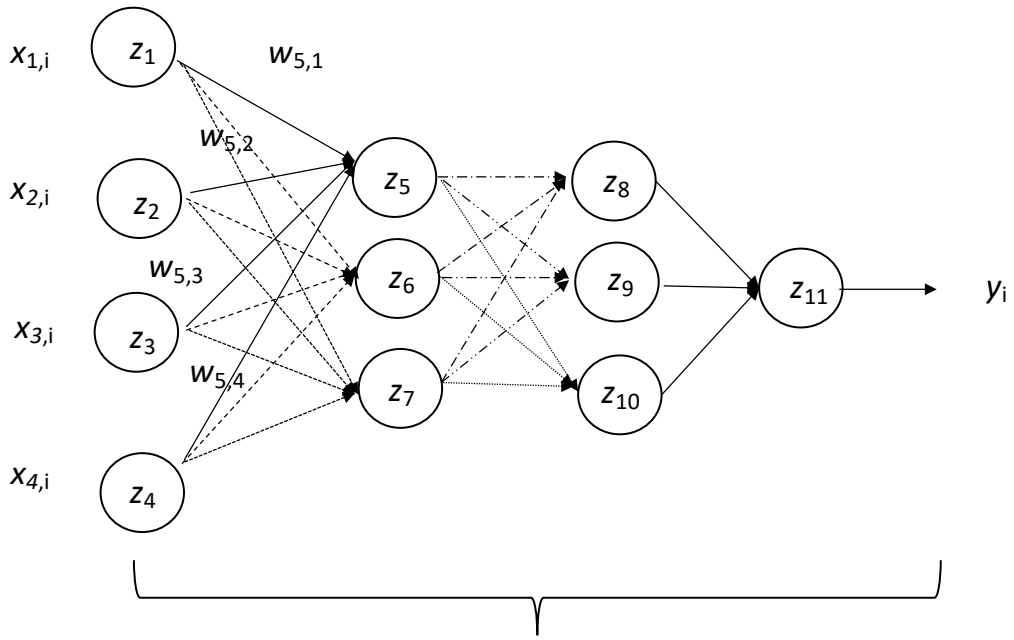
where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \qquad \text{(T4.5)}$$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

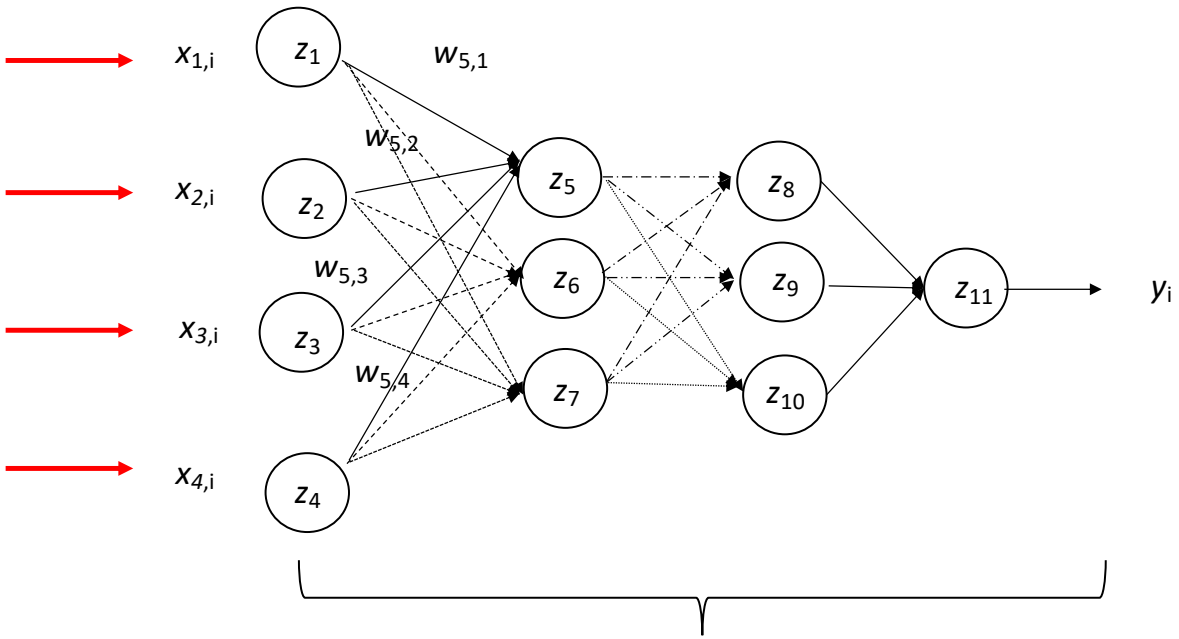


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

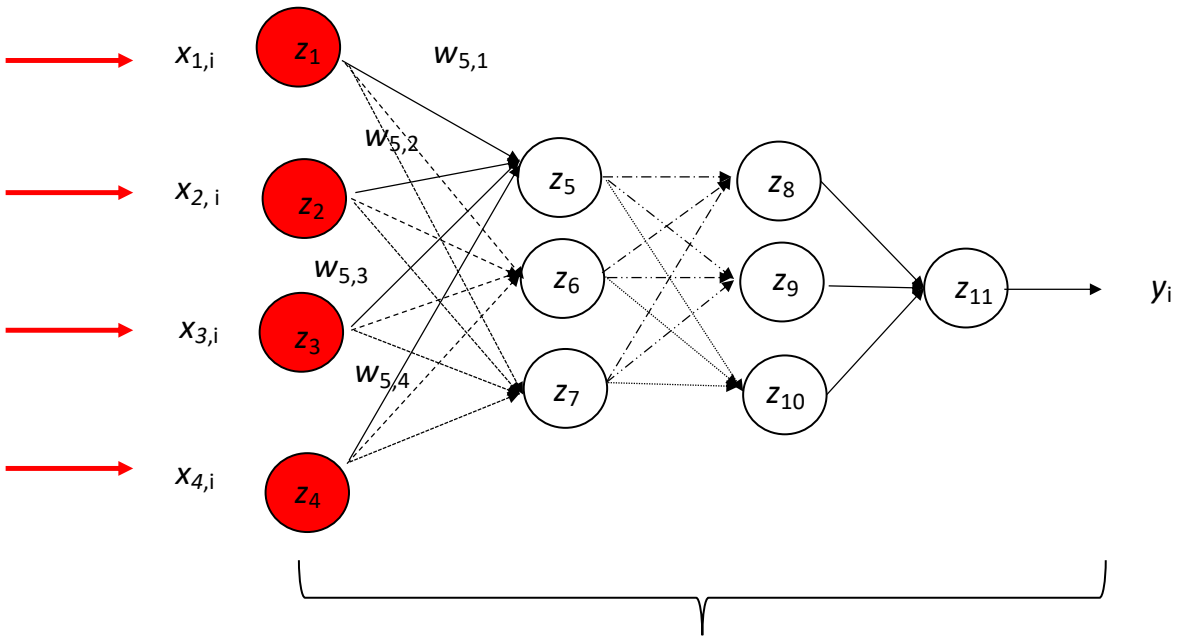


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

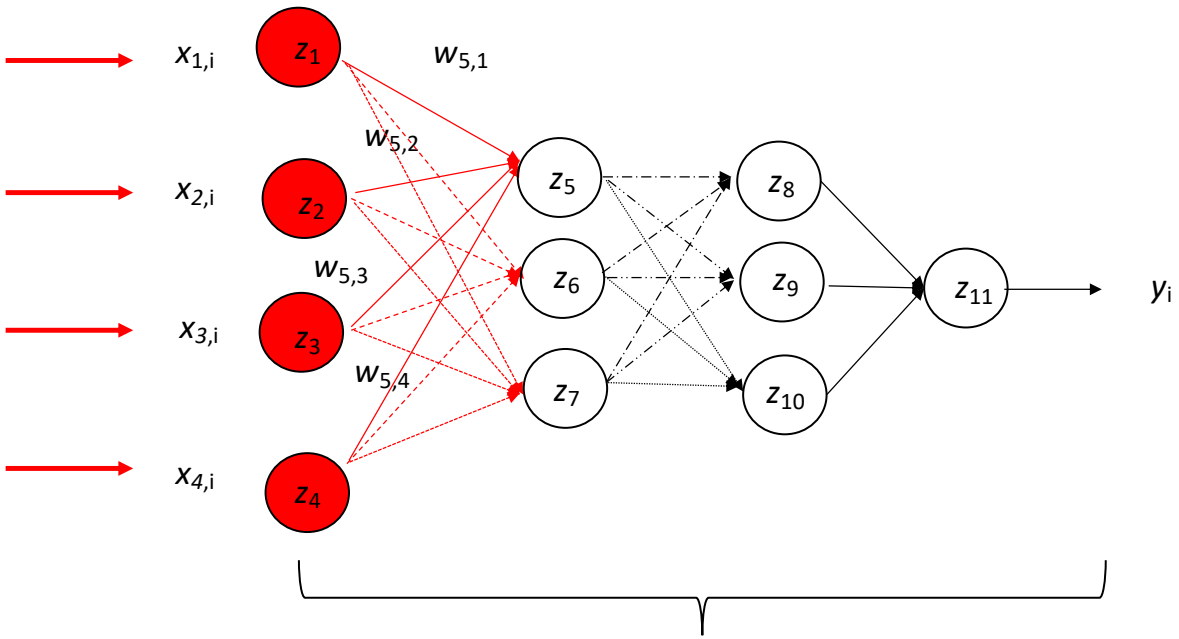


$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

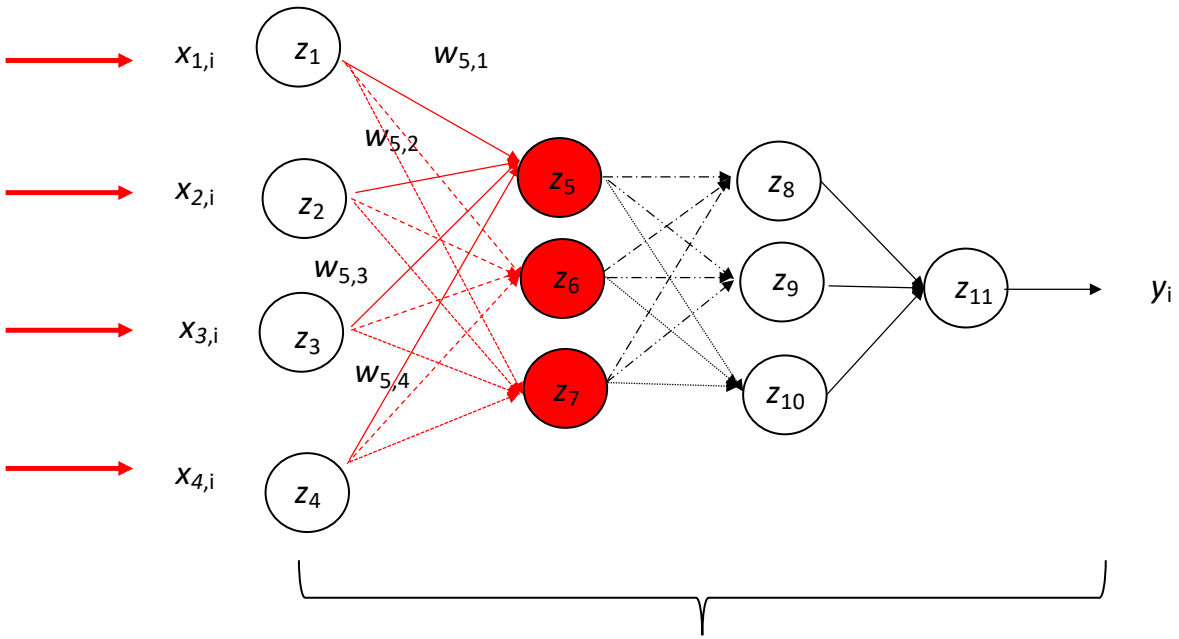


$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

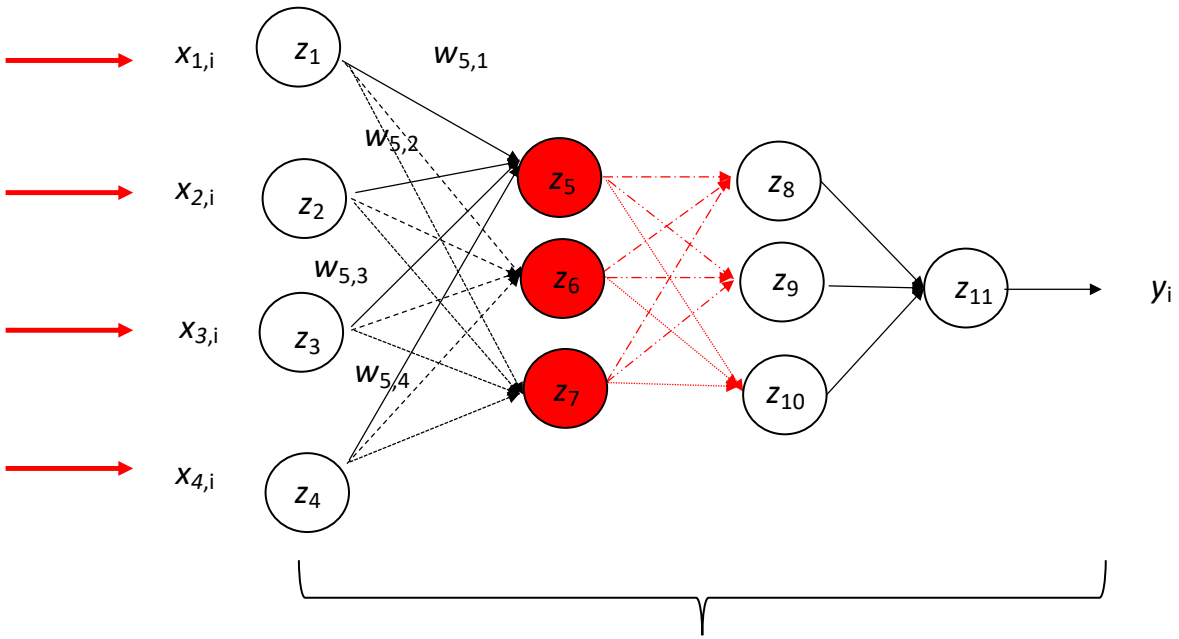


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

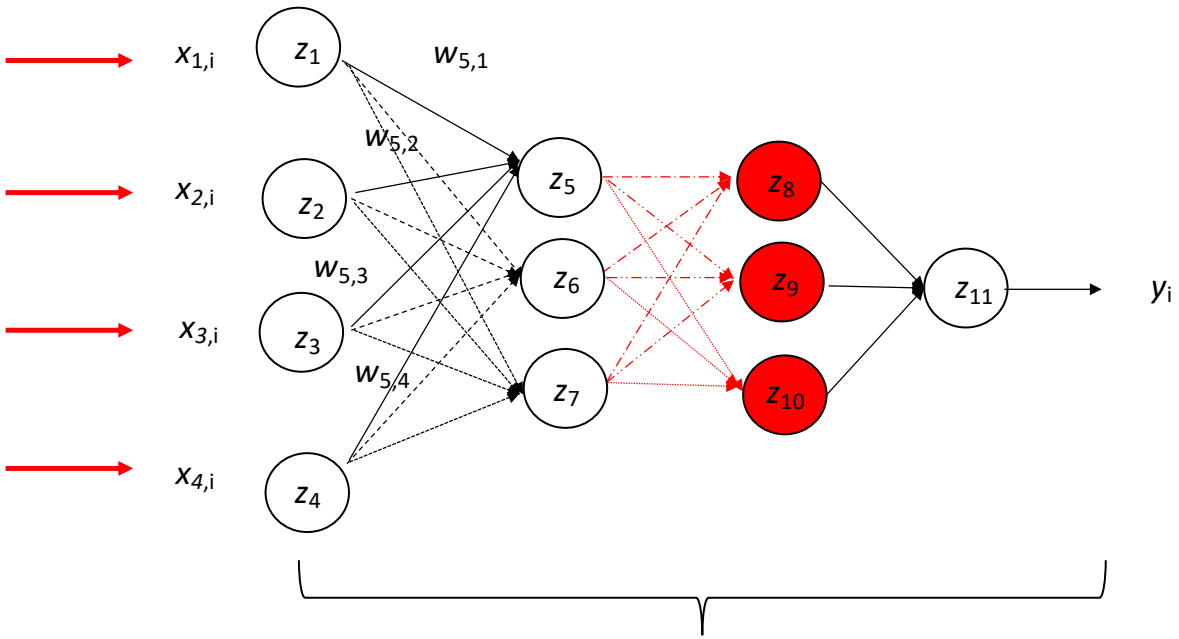


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

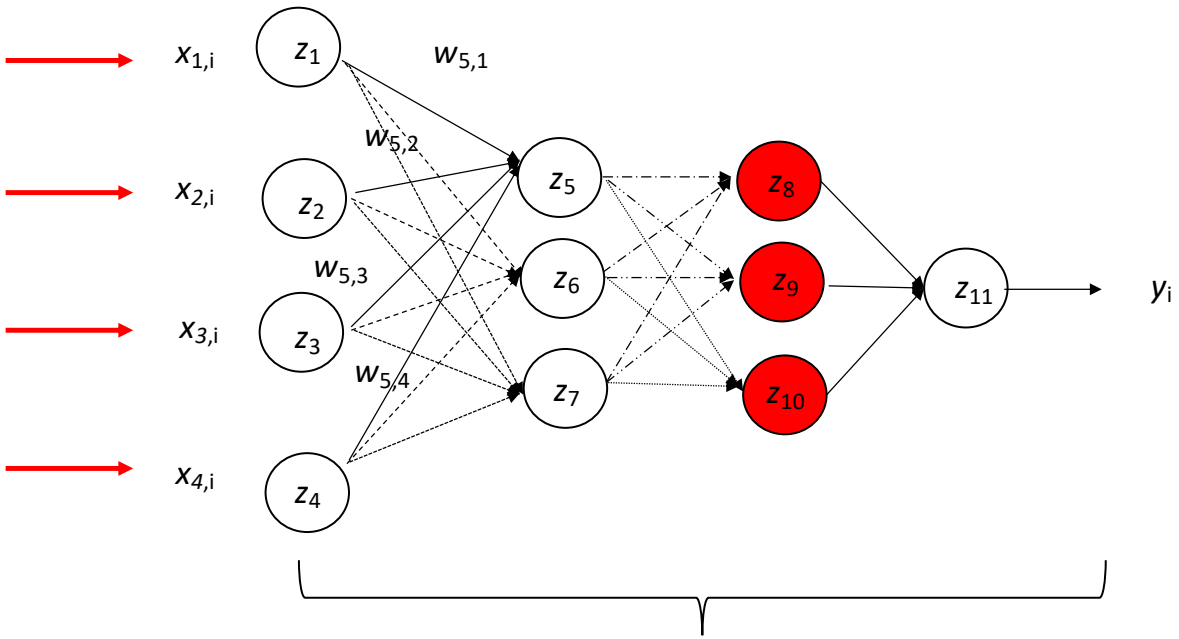


$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

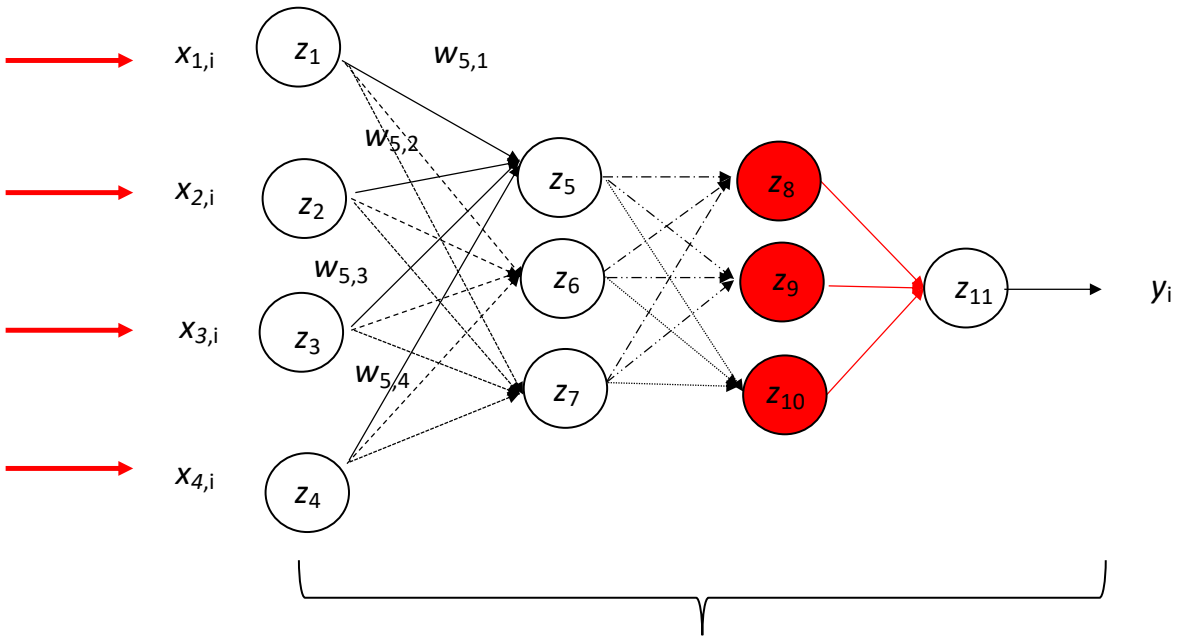


$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

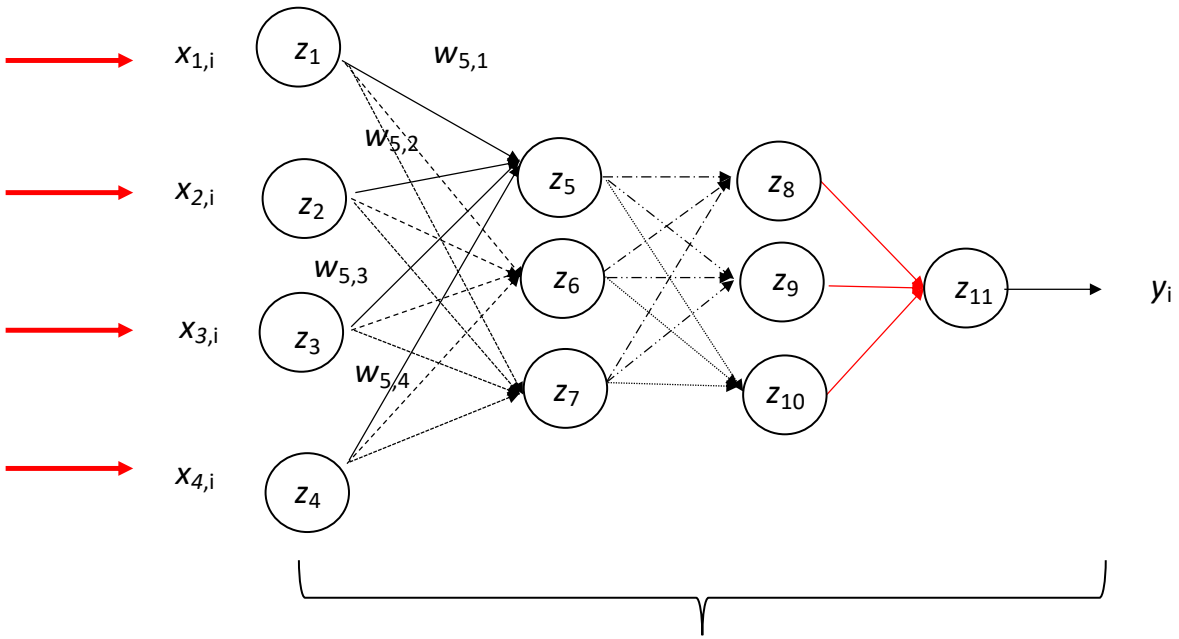


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$

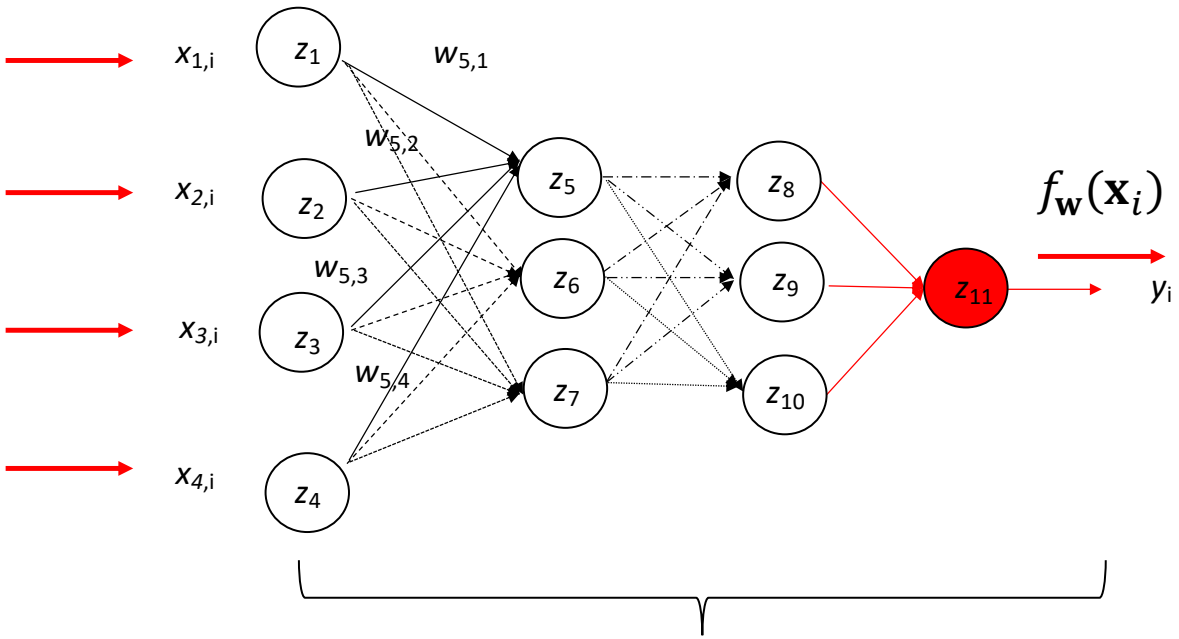


$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Dado: $D = \{(\mathbf{x}_i, y_i)_{1,\dots,N}\}$, with $x_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$



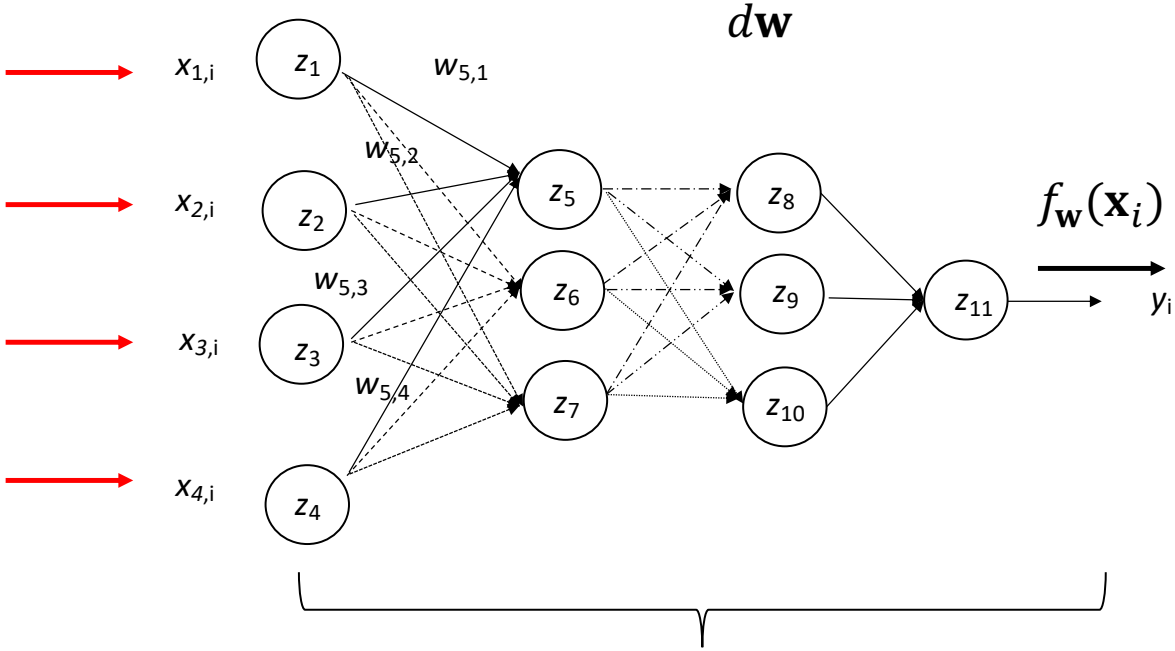
$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$

DNNs – entrenamiento

Para unidades de salida

$$\frac{dE(\mathbf{w})}{d\mathbf{w}}$$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$



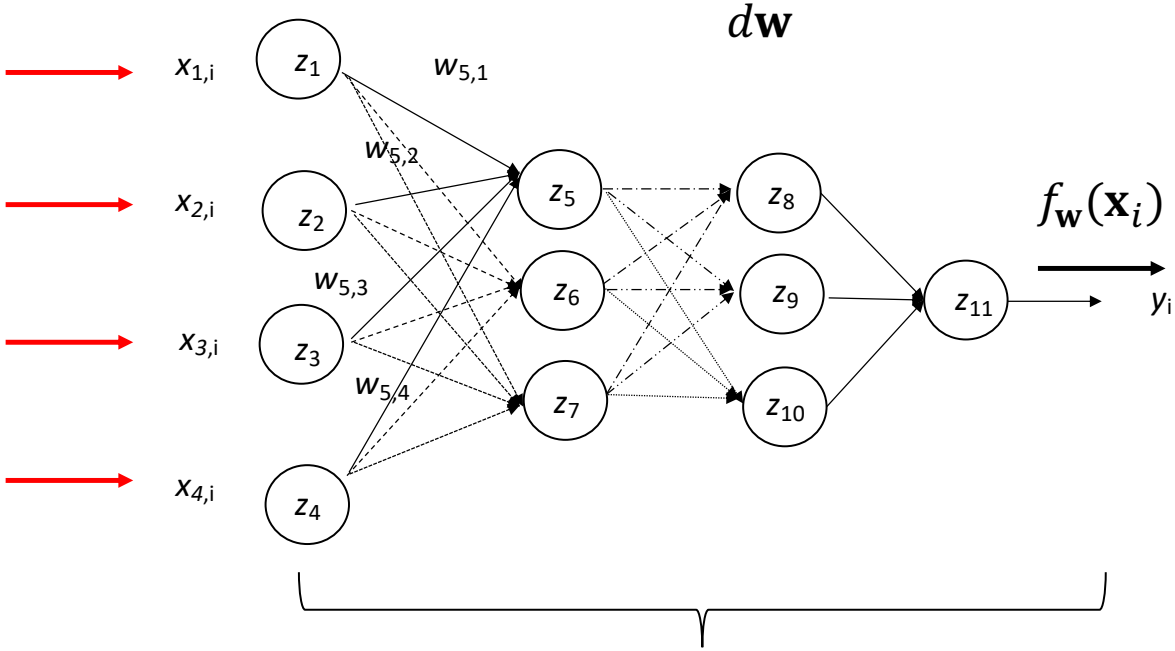
$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs – entrenamiento

Para unidades intermedias

$$\frac{dE(\mathbf{w})}{d\mathbf{w}}$$

$$\mathbf{x}_i = \begin{matrix} x_{1,i} \\ x_{2,i} \\ x_{3,i} \\ x_{4,i} \end{matrix}$$



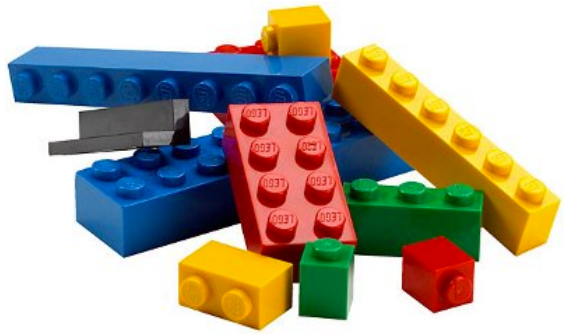
$$f(\mathbf{x}), f: \mathbb{R}^4 \rightarrow \mathbb{R}$$

DNNs

- Las redes neuronales profundas son simplemente redes neuronales con varias capas ocultas (el número de capas indican la profundidad)
- Es posible ver a una DNN como la anidación de funciones:
  - $f^n(f^{n-1}(f^{n-2}(\dots f^1(\mathbf{x})))$
- La capa oculta es usualmente lo que conecta directamente a la tarea con la red
- Las capas intermedias/ocultas capturan información de los datos de manera indirecta con respecto al objetivo
- Usualmente hay millones de parámetros que deben ser aprendidos, aun así retro propagación funciona

DNNs

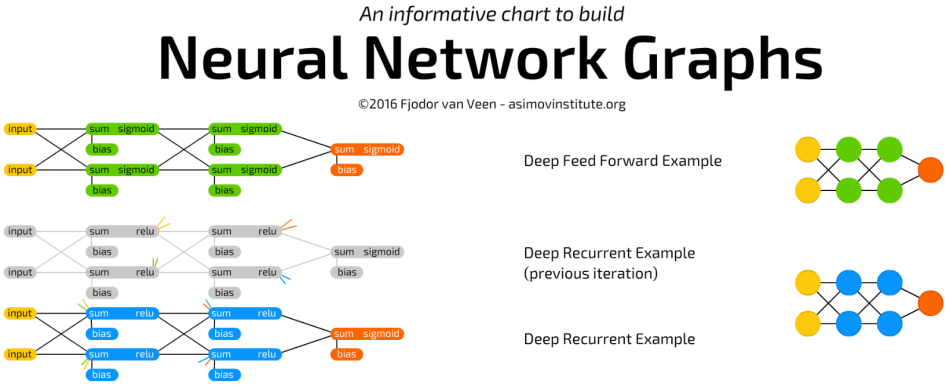
- Componentes de una DNN
  - Unidades de salida
  - Unidades ocultas
  - Estrategia de aprendizaje
  - Función de costo
  - …

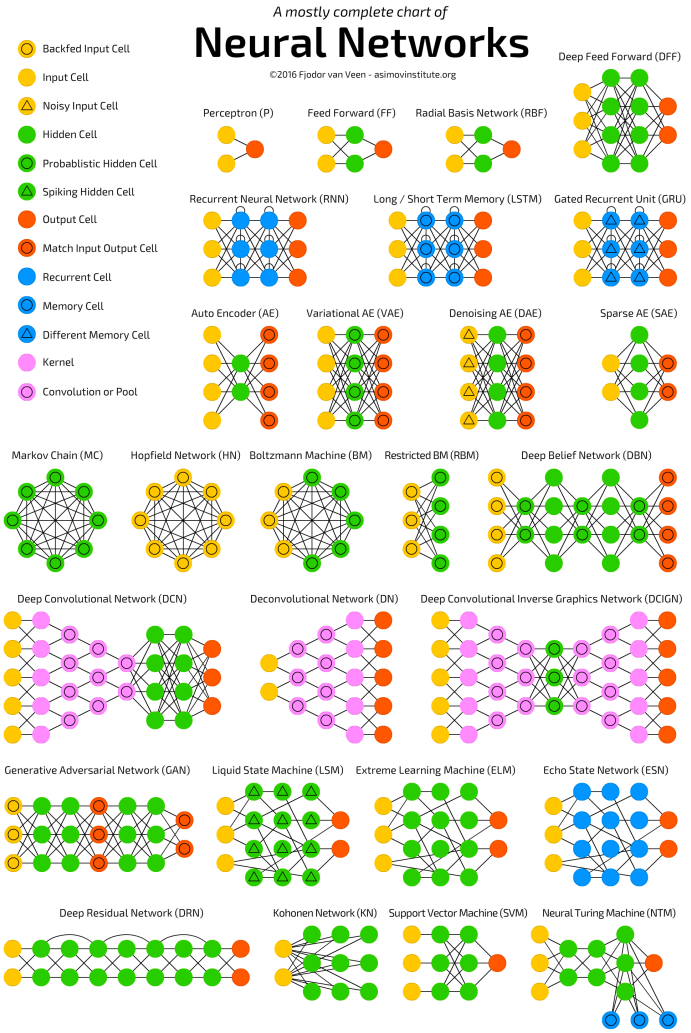


DNNs – consideraciones

- **Diseño de la arquitectura:** Un arte! (apilamiento de capas, Reducción de unidades en capas de niveles más profundos, entre más profundo mejor, usualmente)
- **Entrenamiento de DNNs:** Retro propagación con SGD, más momentum, mini-batches, …
- **Regularización:** Dropout, entrenamiento con adversario, early stopping,
- **Implementación:** GPU

DNNs





https://www.asimovinstitute.org/author/fjodorvanveen/

Recap.

- Introducción al aprendizaje profundo
- Redes neuronales *feedforward*
- Redes neuronales profundas
- Componentes y aprendizaje de redes neuronales

Referencias

- Book:
  - Ian Goodfellow, Yoshua Bengio, Aaron Courville. **Deep Learning.** MIT Press, 2016
- Overviews:
  - Yann LeCun, Yoshua Bengio & Geoffrey Hinton. **Deep learning**. Nature 521, 436–444 (28 May 2015)
  - Andrew L. Beam. Deep Learning 101 - Part 1: History and Background, Blog post, Feb 23, 2017

- Breakthrough:
  - Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. **ImageNet Classification with Deep Convolutional Neural Networks.** Advances in Neural Information Processing Systems 25 (NIPS 2012)
  - Yaniv Tagiman, Ming Yang, Marc Aurelio Ranzato, Lior Wolf. **DeepFace: Closing the Gap to Human-Level Performance in Face Verification**, CVPR 2014
  - Volodymyr Mnih, et al. **Human-level Control through Deep Reinforcement Learning** In Nature, 518: 529–533, 2015.
  - Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. **ImageNet Large Scale Visual Recognition Challenge.** IJCV, 2015.
  - Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. **Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.** IEEE Signal Processing Magazine, Vol 29(6):82 - 97, 2012
  - https://pdollar.wordpress.com/2015/01/21/image-captioning/
  - David Silver et al. **Mastering the game of Go without human knowledge.** Nature, Vol. 550:550-559, 2017