Introducción a Redes Neuronales

Organización de Datos, 2021-1C

Repaso: un modelo

- Planteamos una hipótesis
- Definimos qué es un buen modelo
- Intentamos llegar a uno

Repaso: un modelo (v2)

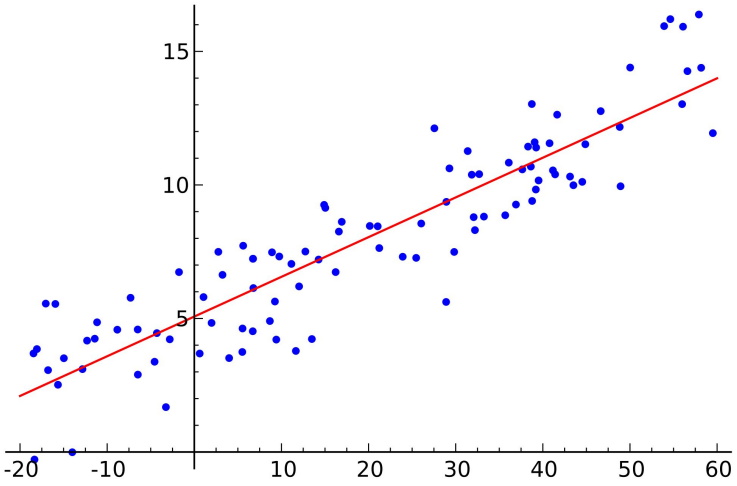
- Planteamos una función predictora
  - Tendremos *parámetros* e *hiperparámetros*
- Definimos una función de error o *de costo*
  - Según si es un problema de clasificación o regresión tenemos varias opciones.
- Minimizamos el costo con métodos numéricos
  - Esto nos encuentra los mejores *parámetros*. Los *hiperparámetros* los buscamos de otra manera.

Primer modelo:
Regresión lineal

Regresión lineal

Una regresión lineal es un modelo que intenta ajustar una línea a un conjunto de datos.
Es decir, queremos hallar el conjunto *m* y *b* que más se acerque a nuestros datos, según la ecuación de la recta:

$$y = mx + b$$



Hipótesis de la regresión lineal

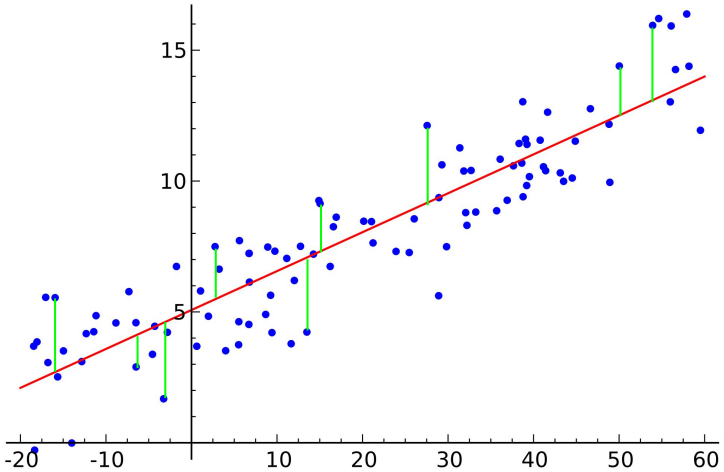
Planteamos la hipótesis de la regresión lineal:

$$h_\theta(\bar{x}) = \theta^t \bar{x}$$

x es una muestra (o evidencia) y *θ* es el parámetro que queremos aprender.

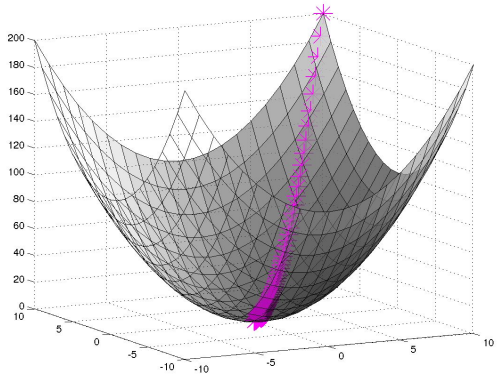
¿Cómo sabemos si una regresión es buena?

Viendo el gráfico, queremos minimizar los errores de predicción, es decir, las longitud de las líneas verdes:



Idea de optimización: descenso de gradiente

- El gradiente me dice para dónde aumenta más la función
- Si voy en sentido contrario, para dónde disminuye más
- Es la mejor opción local => Greedy



Generalmente agregamos algo de inercia al gradiente.

Objetivo de la regresión lineal

Que es equivalente a disminuir el error cuadrático:

$$J(\theta) = \frac{1}{2} \sum_{i} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Minimizar esta función significa que se predice algo muy cercano al valor real.

Optimización

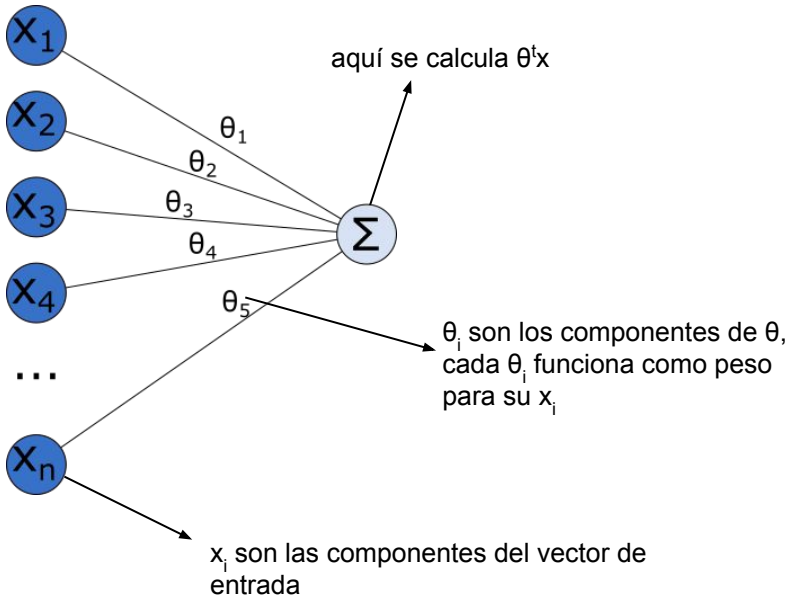
Podemos utilizar descenso por gradiente. Definimos la sucesión:

$$\theta_{n+1} = \theta_n - \gamma \bar{\nabla} J(\theta_n)$$

Aquí “γ” es un parámetro *no aprendido* y por lo tanto debemos definirlo nosotros. Se denomina “ritmo de aprendizaje” (*learning rate*) y puede ser constante, o dependiente del número de iteraciones.

Grafo

Podemos representar este tipo de regresión a través de una red, de la siguiente manera:



Segundo modelo:
Perceptrón clásico

Perceptrón

- Puede ser utilizado de manera *online*
- Tiene convergencia asegurada.
- Es simpático

Usamos la siguiente función de activación:

$$f_\theta(x) = \begin{cases} 1 & \text{si} \quad \theta^t x \geq 0 \\ 0 & \text{c.c} \end{cases}$$

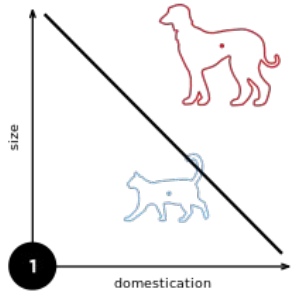
Entrenamiento

Los pesos se actualizan según:

$$\theta^{t+1} = \theta^t + \gamma(y_i - f_{\theta^t}(x_i))x_i$$

¡esto se puede hacer *online*!

Entrenamiento



Tercer modelo:
Regresión logística

Regresión logística

En el caso anterior intentábamos predecir un valor continuo a partir de un vector. Ahora, queremos predecir algo discreto.

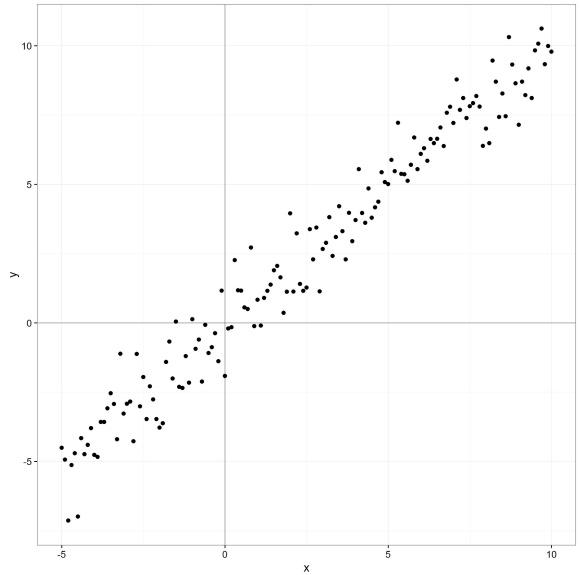
Para simplificar, la predicción será binaria: “0” o “1”.

¿Cómo transformamos el modelo anterior en un clasificador binario?

¿Qué es necesario *agregarle*?

Regresión logística

Buscamos una transformación que tome el valor de la regresión lineal y lo ponga en un intervalo [0,1]:

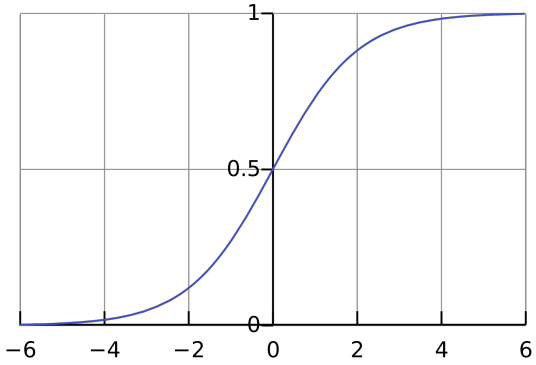


$$f : \mathbb{R} \to [0, 1]$$

[0,1]

Regresión logística

Parche: definir una transformación continua que me “aplaste” el valor a un intervalo [0,1].



$$f(x) = \frac{1}{1 + e^{-x}}$$

Esta función se llama, por su uso, “función logística”.

Hipótesis de la regresión logística

Por ello, la regresión logística hace la siguiente hipótesis:

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + \exp(-\theta^t x)}$$

Notemos que aquí incluimos el parámetro “θ” que guiará el valor, y es el parámetro aprendido.

¿Cómo estimar la calidad de predicción?

Introducimos el concepto de *verosimilitud*(likelihood):

$$\mathfrak{L}(\theta|x) \equiv P(x|\theta)$$

Se lee: “la verosimilitud del parámetro θ dadas las evidencias *X* se define como la probabilidad de las evidencias *X* dado el parámetro θ”.

¿Cómo estimar la calidad de predicción?

Entonces, la verosimilitud del parámetro nos dice cuán bien se ajusta a nuestros datos.

Nosotros queremos encontrar el parámetro que mejor prediga los “1” y los “0”.

¿Cómo estimar la calidad de predicción?

La función costo queda, entonces:

$$J(\theta) = -\sum_i \left(y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

Esta ecuación se llama log-likelihood.

Gradiente

Derivando la función costo anterior, llegamos a la siguiente expresión:

$$\bar{\nabla} J(\theta) = \sum_i x^{(i)} (h_\theta(x^{(i)}) - y^{(i)})$$

Calcular esto es tremendamente simple. ¡No es casualidad! Las funciones costo intentan lograr una expresión sencilla para su gradiente.

¿Qué es lo que hicimos?

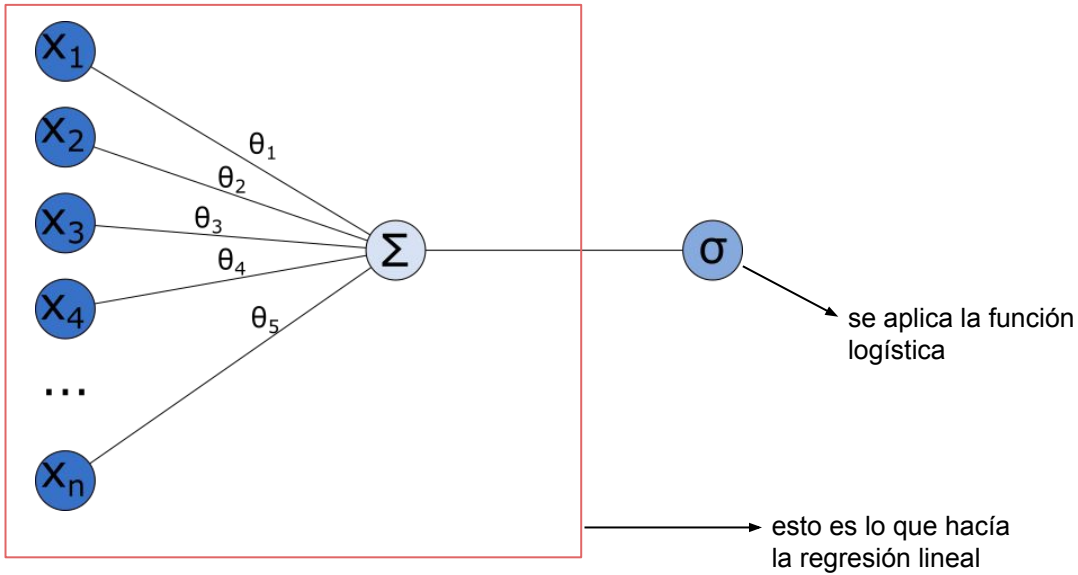
Finalmente, la regresión logística es un “parche” a la regresión lineal. Le aplicamos la función sigmoide para poder predecir clases, en vez de valores continuos.

Ejemplo interactivo.

¿Cómo podemos representar esto?

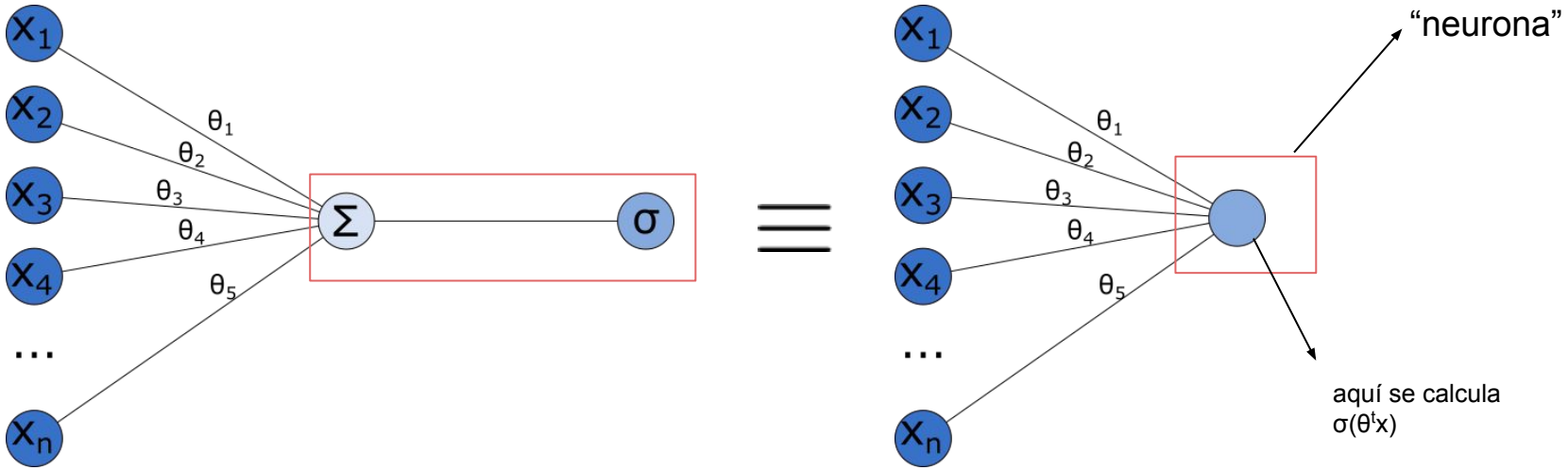
Grafo

Como antes, podemos también representar el modelo de manera más visual:



Grafo

A partir de ahora, haremos una representación más compacta:



Cuarto modelo:
Regresión softmax

Generalización

La regresión softmax es una generalización de la regresión logística para el caso de múltiples clases.

Podemos reescribir el caso binario como un vector de la siguiente manera:

$$P(x|\theta) = \begin{pmatrix} \exp(\theta^{(1)t}x) \\ \exp(\theta^{(2)t}x) \end{pmatrix} \frac{1}{\sum_{i=1}^{2} \exp(\theta^{(i)t}x)}$$

Generalización

Ahora generalizamos para el caso de *k* clases.

$$P(x|\theta) = \begin{pmatrix} \exp(\theta^{(1)t}x) \\ \dots \\ \exp(\theta^{(k)t}x) \end{pmatrix} \frac{1}{\sum_{i=1}^{k} \exp(\theta^{(i)t}x)}$$

Generalización

Observemos que ahora Θ comprende los parámetros (vectores) de cada clase, $\theta^{(1)}$, $\theta^{(2)}$, .., $\theta^{(k)}$. La podemos representar como una matriz de N filas y K columnas.

$$P(x|\Theta) = \text{Softmax}(\Theta x)$$

Función costo

Ahora queremos hacer lo mismo, para la función costo.

$$J(\theta) = -\sum_{i} \left(y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

$$J(\theta) = -\sum_{i=1}^{m} \sum_{k=1}^{2} 1\{y^{(i)} = k\} \frac{\exp(\theta^{(k)t} x^{(i)})}{\sum_{j=1}^{2} \exp(\theta^{(j)t} x^{(i)})}$$

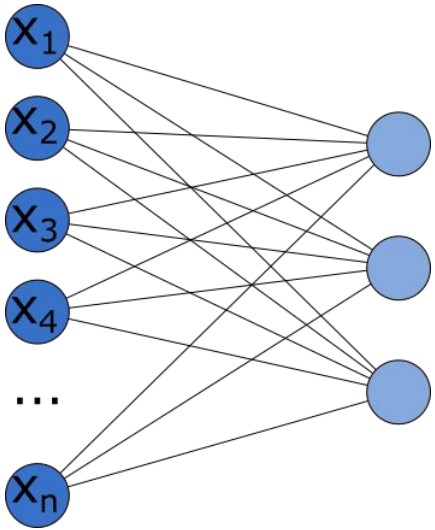
Función costo

¡Y así queda la función!

$$J(\theta) = -\sum_{i=1}^{m}\sum_{k=1}^{K} 1\{y^{(i)} = k\} \frac{\exp(\theta^{(k)t} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)t} x^{(i)})}$$

Grafo

Nuevamente, podemos graficarlo. En este caso, elegimos k=3:



Tengamos en cuenta que cada neurona tiene sus propios pesos, como tenemos 3 neuronas tendremos *3n* pesos.

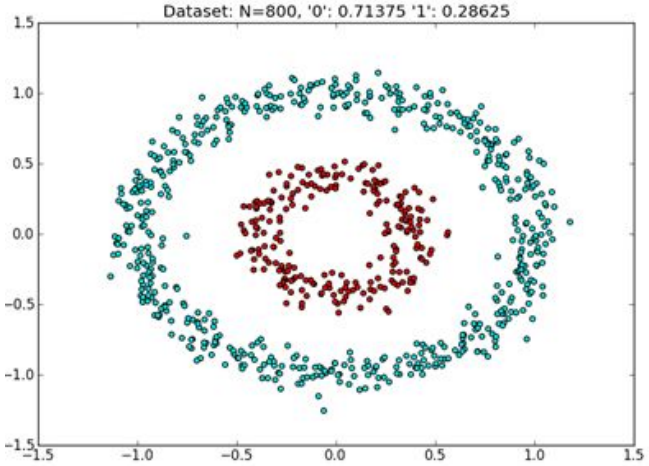
De esta manera se ve bastante claro cómo la regresión softmax es una generalización de la regresión logística.

Quinto modelo:
Perceptrón multicapa

Decision boundary

Ahora, todavía estamos trabajando con problemas separables linealmente.

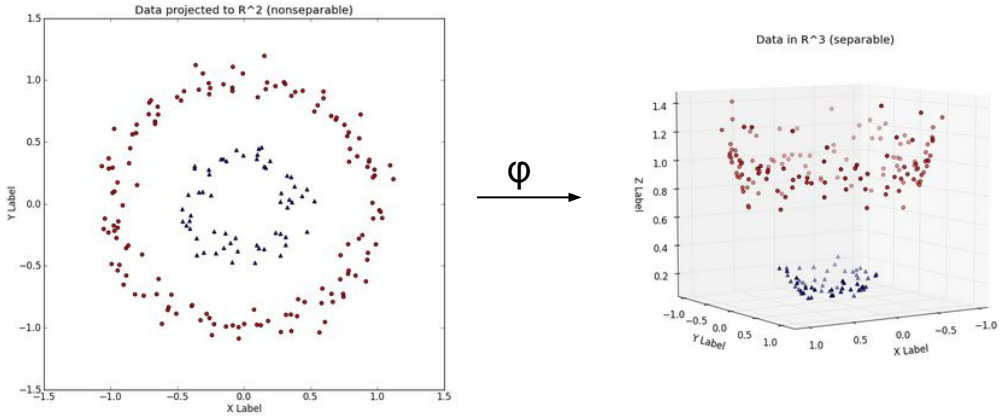
¿Qué pasa si queremos separar algo más complicado?



Decision boundary

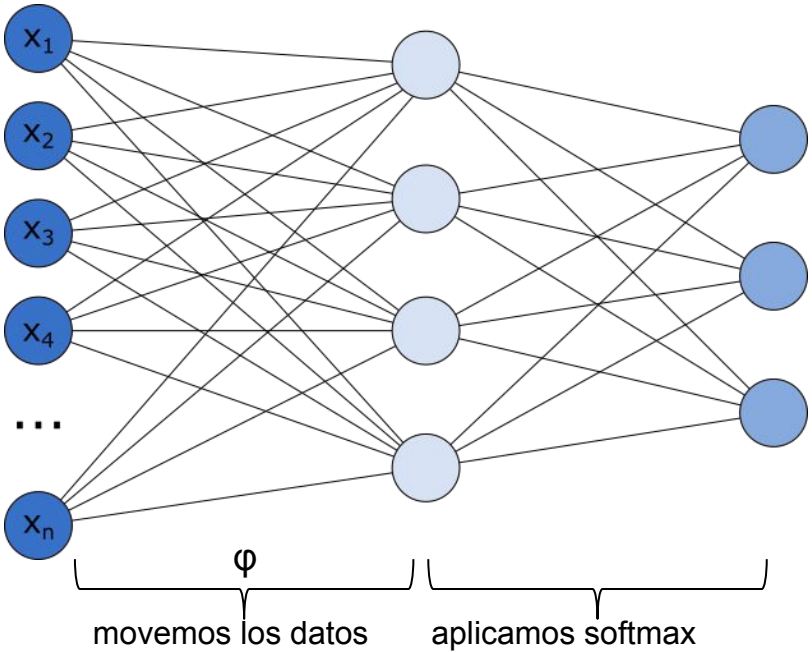
Necesitamos mover los puntos de tal manera que, luego de moverlos, podamos separarlos linealmente.

Una función φ que los mueva puede ser así:



Decision boundary

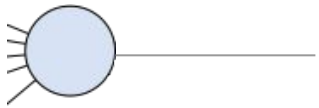
Esto lo podemos hacer agregando una capa adicional.



$$\text{softmax}(\phi(\bar{x}))$$

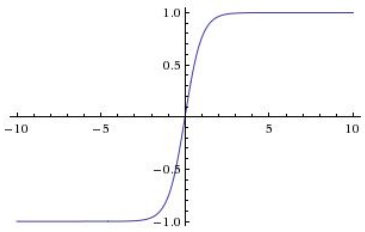
Función de activación

Una neurona tiene varias entradas y una salida.

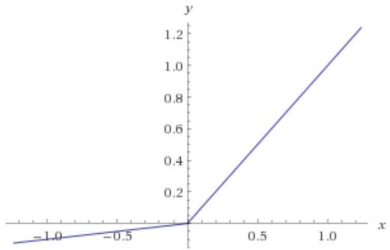


La salida es una función de la entrada: $\text{Salida} = \sigma\left(\theta^t w_{\text{Entrada}}\right)$

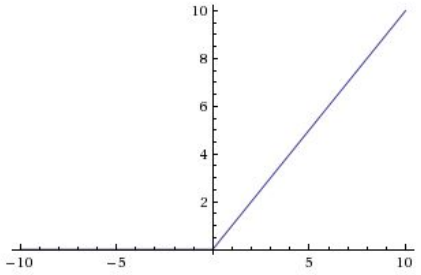
En este contexto σ se llama función de activación. Hasta ahora usamos la función sigmoide, pero hay otras:



tanh



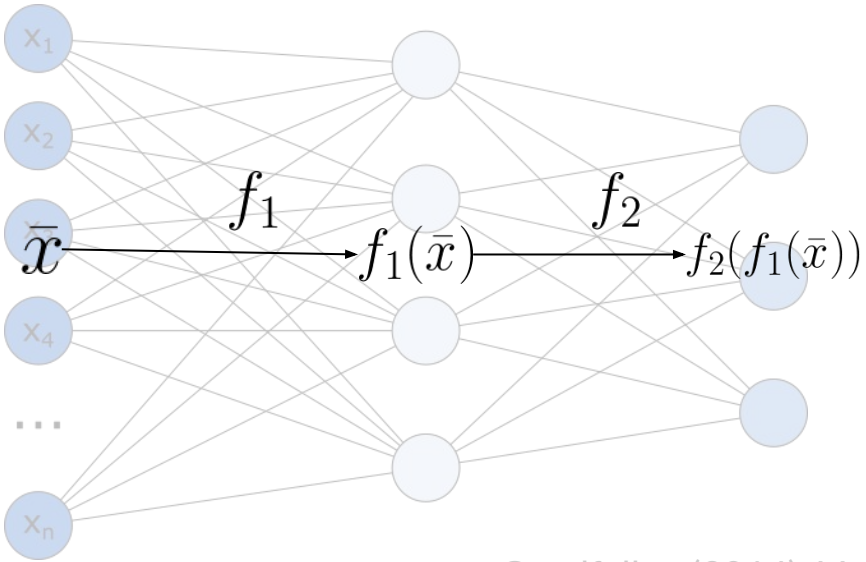
LeakyReLU



ReLU $y = \max\{0, x\}$

Multilayer perceptron

Podemos interpretarlo como una secuencia de funciones.

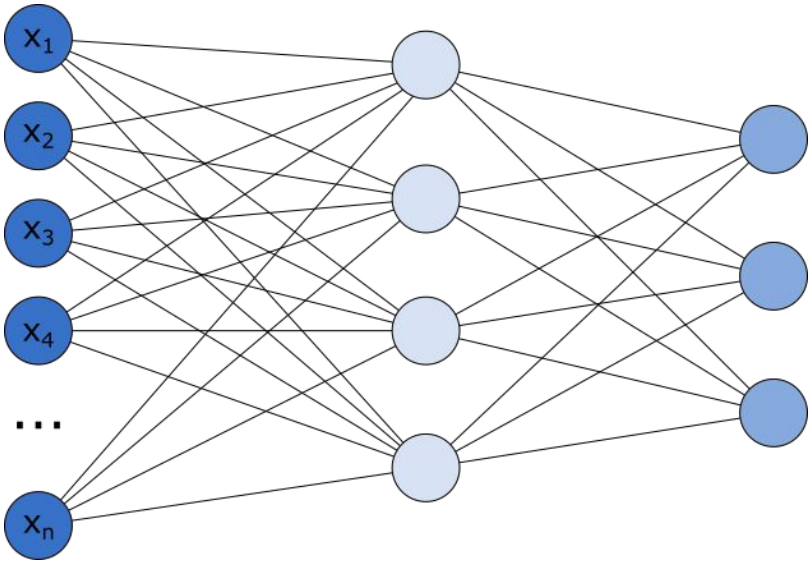


Por supuesto, también podemos tener más de dos capas: tres, cuatro, diez..

Goodfellow(2014), *Multilayer Perceptron*

Multilayer perceptron

Es una red neuronal simple, con cero o más capas ocultas.



Esto permite al algoritmo aprender bordes de decisión no-lineales.

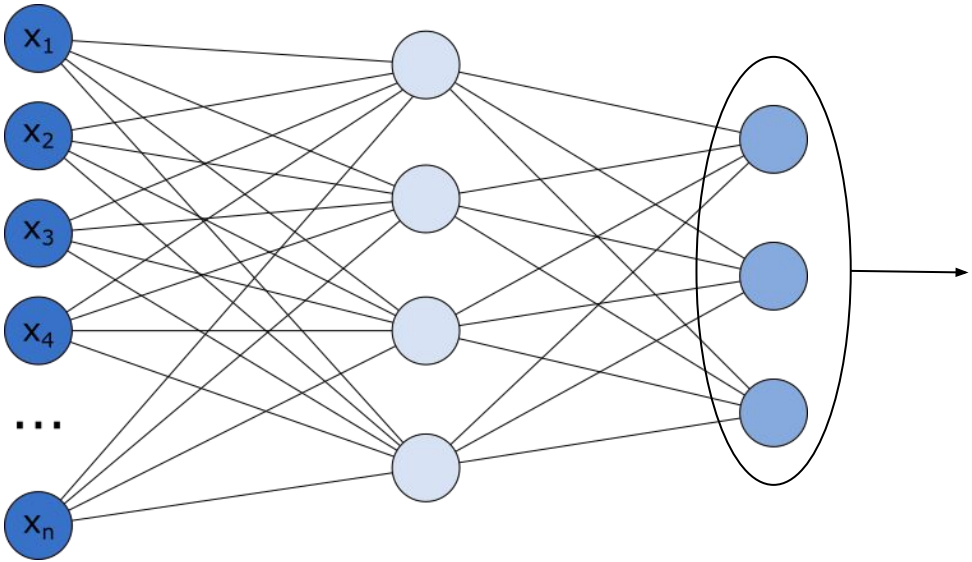
Ejemplo interactivo.

Teorema de aproximación universal

El *teorema de aproximación universal* nos garantiza que con una sola capa oculta de suficiente número de neuronas, podemos representar cualquier función.

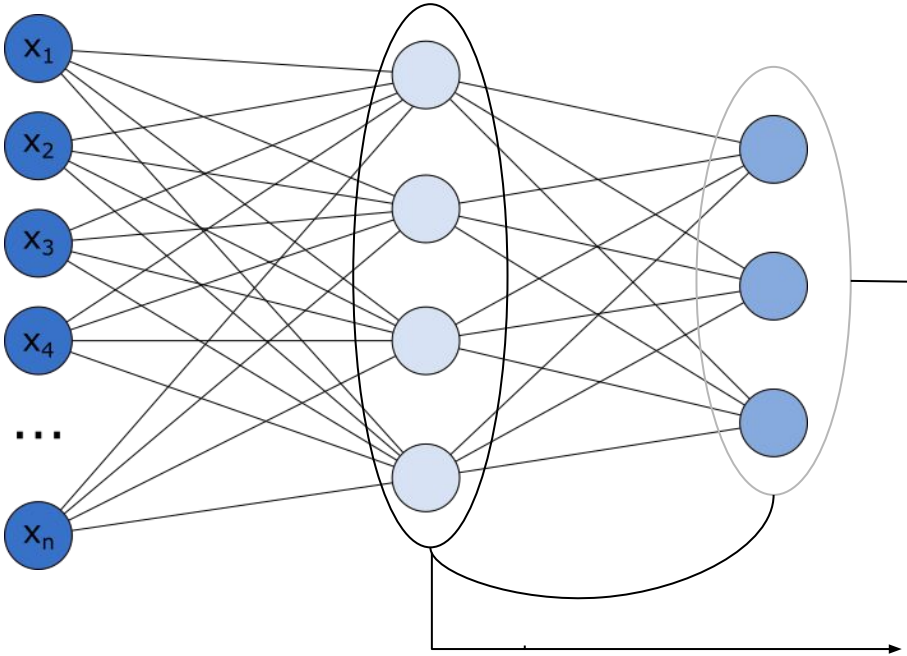
Entonces, un perceptrón multicapa podrá separar cualquier set de datos.

¿Cómo entrenamos esto?



Esta capa es la predicción final.
Entonces podemos poner el *error* o el *costo* en función de esta capa.

¿Cómo entrenamos esto?



Esta capa es la predicción final. Entonces podemos poner el *error* o el *costo* en función de esta capa.

Como la capa final depende de ésta, el error también es función de esta capa.

Backpropagation

Esto es retropropagación (*backpropagation)*. Calculamos el error para la capa final, y propagamos los errores hacia atrás.

Es una forma rápida de calcular las derivadas por regla de la cadena.

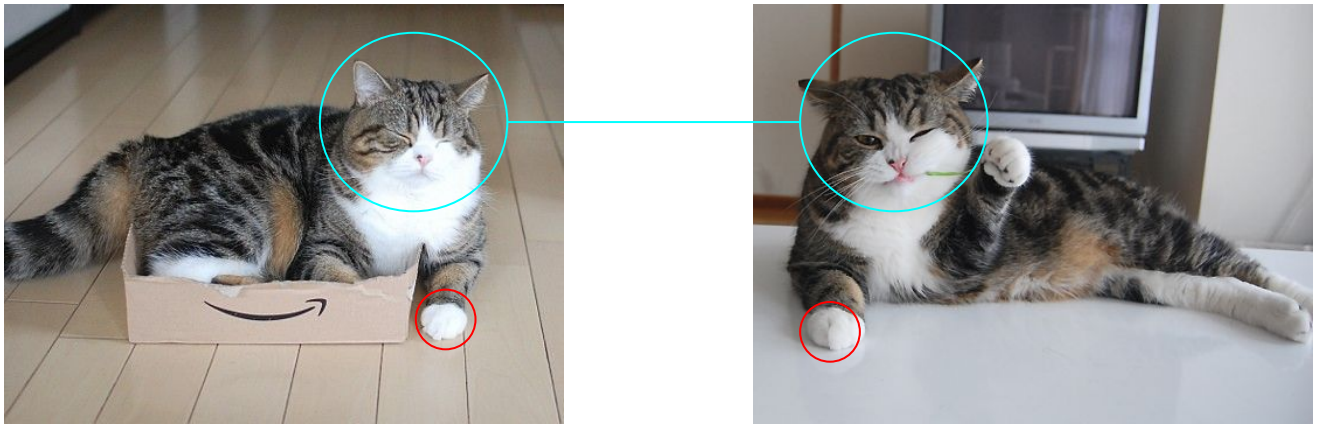
$$\frac{\partial \text{Costo}}{\partial \text{Capa}_{n-1}} = \frac{\partial \text{Costo}}{\partial \text{Capa}_{n}} \times \frac{\partial \text{Capa}_{n}}{\partial \text{Capa}_{n-1}}$$

Nielsen (2015), *Neural Networks and Deep Learning*

Redes Neuronales Convolucionales

Filtros

Tal vez nos interese cambiar algunas de estas funciones intermedias, según la aplicación que le queramos dar.



Filtros

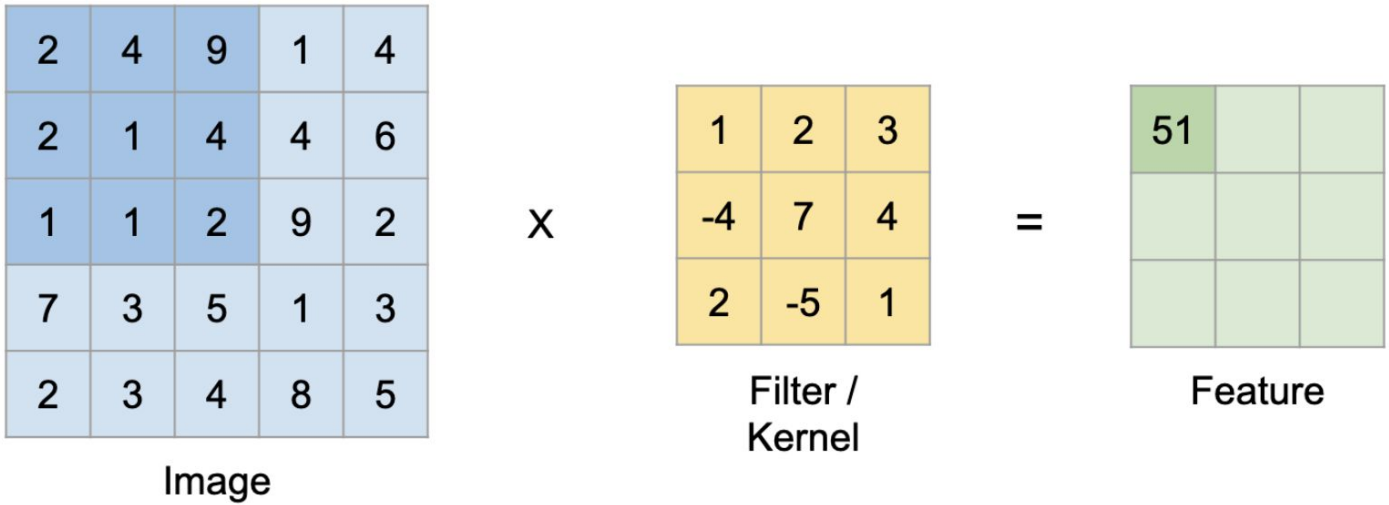
Podríamos usar funciones (o *filtros*) especiales para este caso.

Una función podría ser una *convolución de imágenes*.

Y otra podría ser una función de *max-pooling*.

Filtros

Un filtro es simplemente una convolución de una zona de la imagen..



Filtros

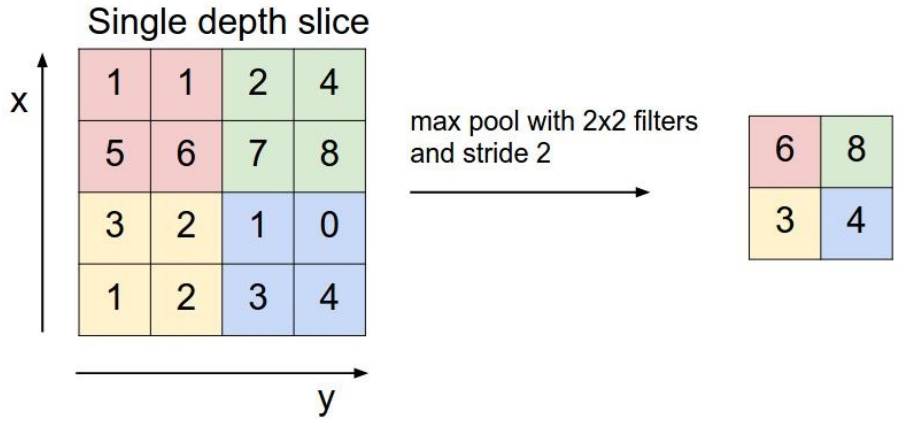
Hiper parámetros a definir:

- Cantidad de filtros a usar.
- Tamaño del filtro (ej 3x3)
- Stride (paso)
- Padding

Los parámetros son los coeficientes del filtro y eso lo aprende la red!

Max-Pooling

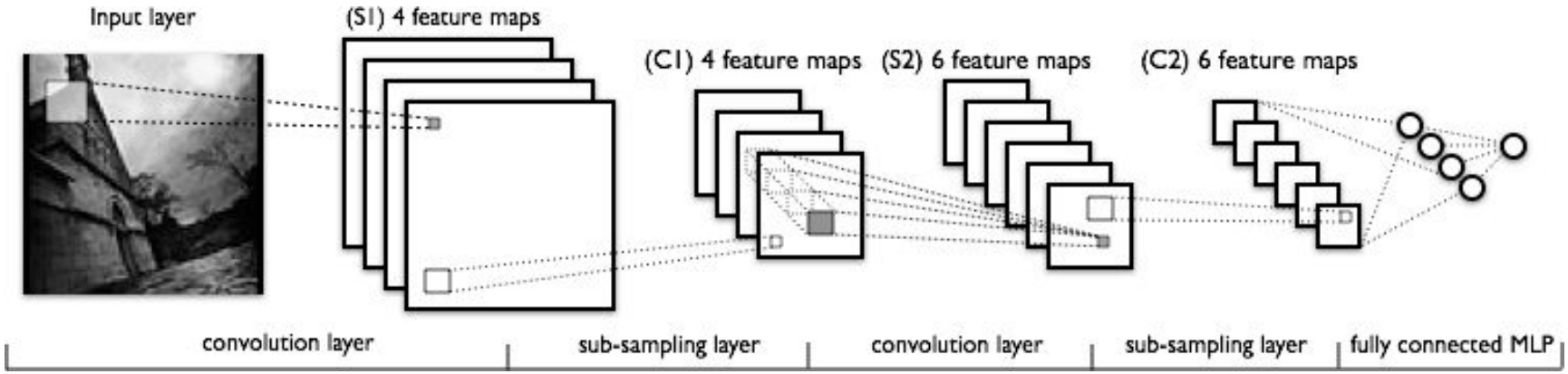
En max-pooling, para secciones de la imagen se obtiene la *feature* más importante. Es otra manera de obtener información local.



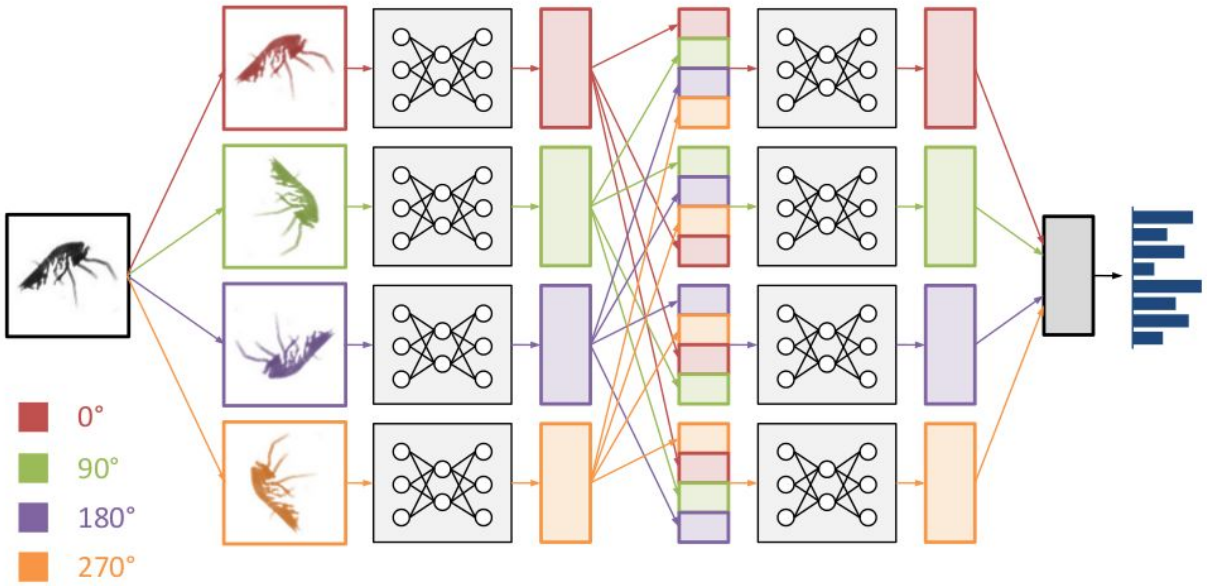
Andrej Karpathy (2016) CS231n, Stanford University

Convolutional neural network

La red quedaría de esta manera:



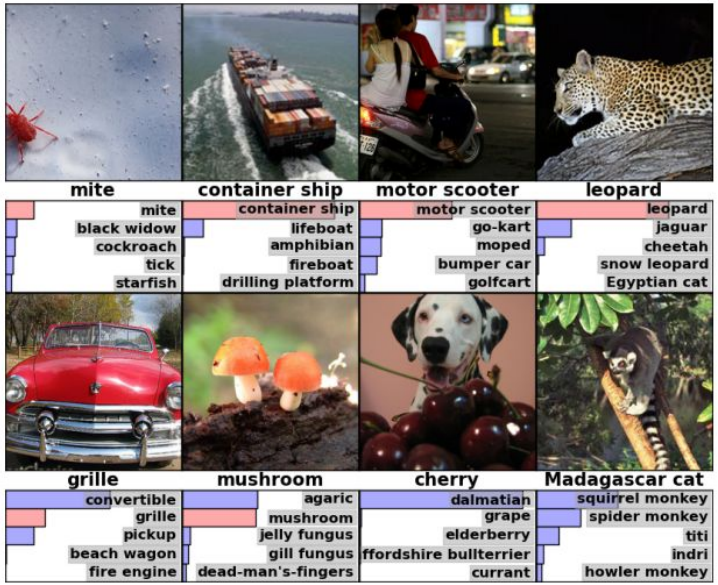
Convolutional neural network



Resultados

ImageNet 2012

Reconoce 1000 objetos distintos



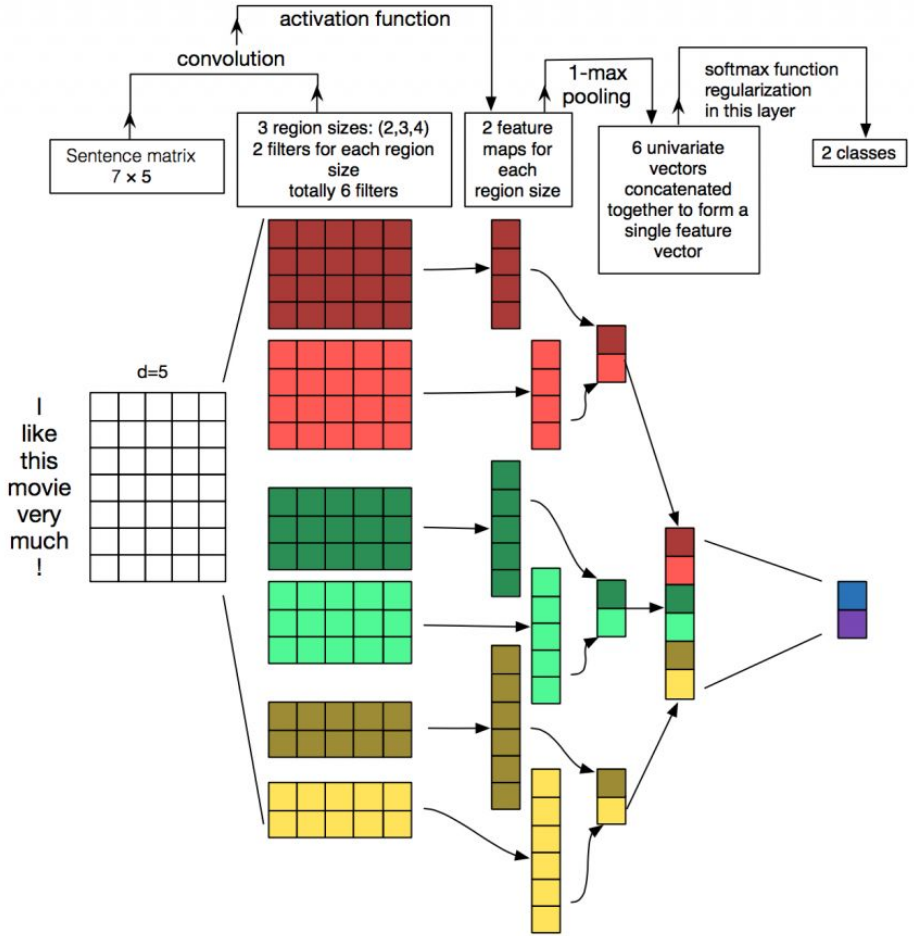
Krizhevsky et al. (2012) ImageNet Classification with Deep Convolutional Neural Networks

Conv1d (Convolucionando Texto)

Es posible aplicar el mismo concepto que usamos para imágenes en textos.

Para eso en primer lugar tenemos que representar a cada palabra del texto como un vector de “n” dimensiones mediante un embedding.

Word2vec, GloVe son formas populares de crear embeddings o podemos usar embeddings ya creados para un determinado idioma.

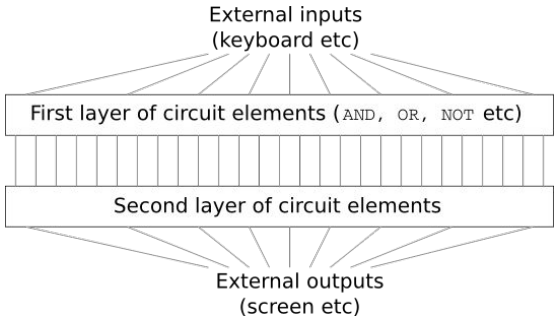

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Redes Neuronales Profundas

Anchura vs profundidad

Imaginemos que nos piden armar un circuito lógico complejo.

Pero nos pide hacerlo en *sólo dos capas*.



Nielsen (2015), *Neural Networks and Deep Learning*

Anchura vs profundidad

La ventaja de usar varias capas es la *abstracción*.

Lo mismo pasa con las redes neuronales: cada capa se abstrae más de los datos iniciales, obteniendo mejores representaciones de los datos.

Es posible hacerlo con sólo dos capas, pero ¿es lo mejor?

A veces necesitamos exponencialmente más neuronas en una sola capa para representar funciones más complejas.

Nielsen (2015), *Neural Networks and Deep Learning*

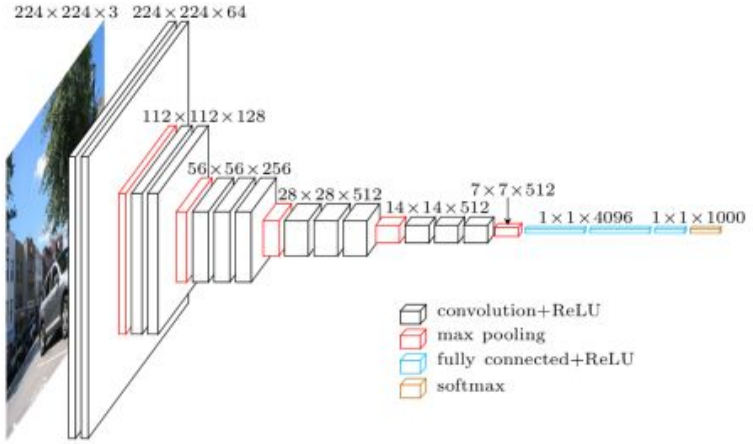
Deep Learning

Este es el modelo del *aprendizaje profundo* (deep learning).

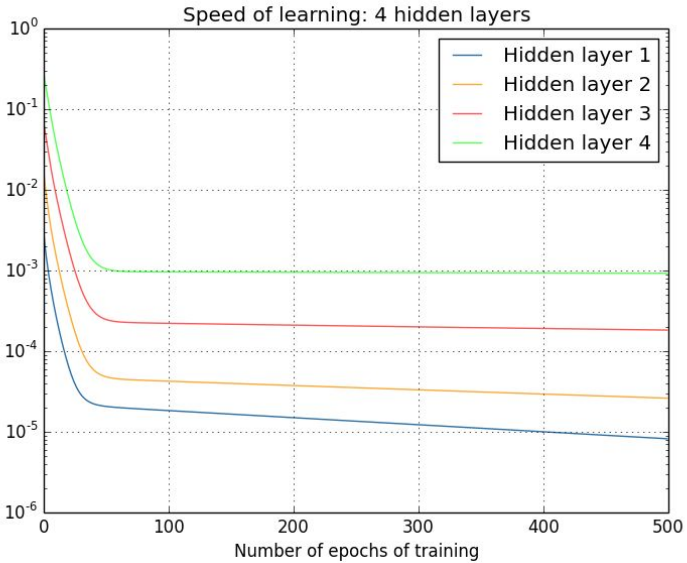
Se presentan algunas dificultades al entrenar si las capas no aprenden al mismo ritmo (*vanishing gradient problem*), pero si se las entrena bien, pueden llegar a tener muchos mejores resultados.

Nielsen (2015), *Neural Networks and Deep Learning*

Arquitectura



Vanishing gradient problem



Nielsen (2015), *Neural Networks and Deep Learning*

Técnicas de entrenamiento

- Drop out
- Batch normalization
- Weight normalization
- Regularization
- Transfer Learning

Herramientas

TensorFlow

· Librería en Python.

· Nos permite definir “grafos computacionales”, en nuestro caso las capas de una red neuronal.

· Puede correr sobre CPU o GPU.

· Ejemplo sobre Keras

Referencias

- Algoritmo de Backpropagation: Nielsen, M.: NN&DL Cp. 2
- Teorema de aproximación: Nielsen, M.: NN&DL Cp. 4
- Softmax, logística, MLP: Stanford, UFLDL Tutorial
- Backprop como regla de la cadena: Olah, C.: Blog
- Perceptrón: apunte de la materia.

Playground interactivo de Tensorflow.

Basado en la charla dada en el Laboratorio de Ciencia de Datos (2017)