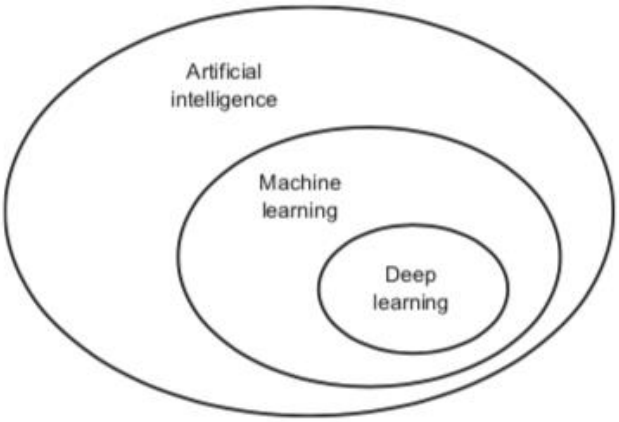
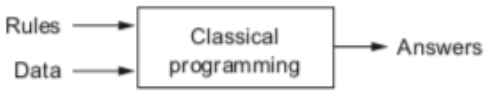
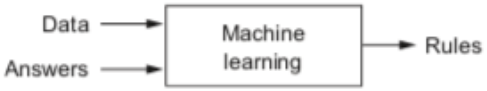
Hasta ahora…dinámica: puntos fijos… redes; feedforward…

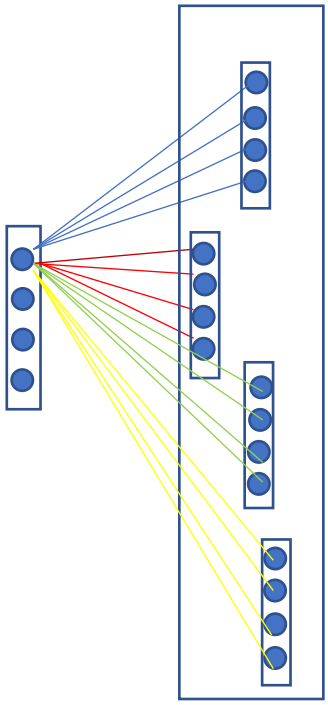
Clase III: Panorama general de redes en el mar de Inteligencia artificial



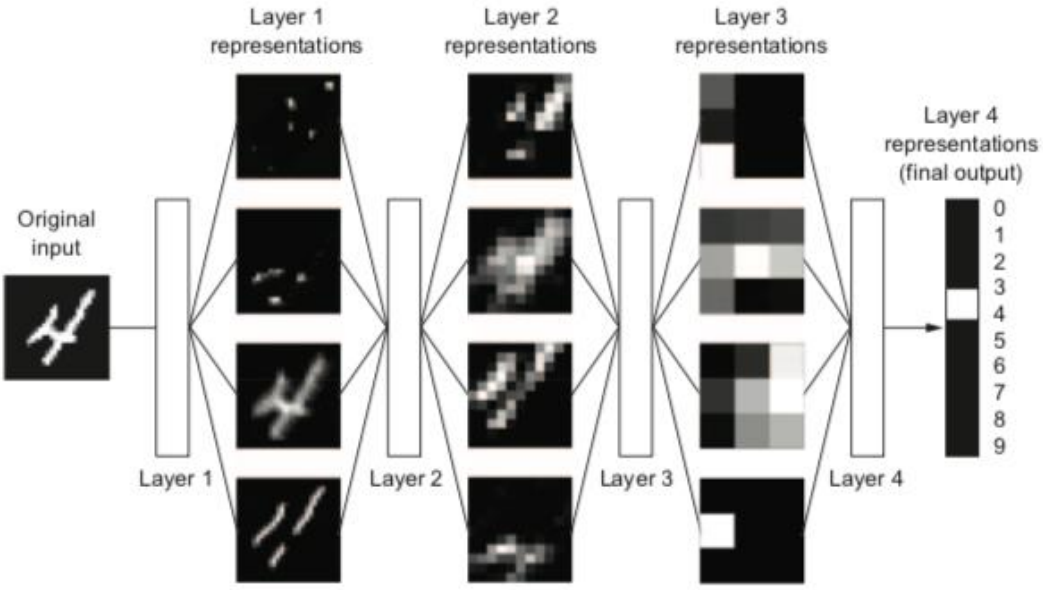




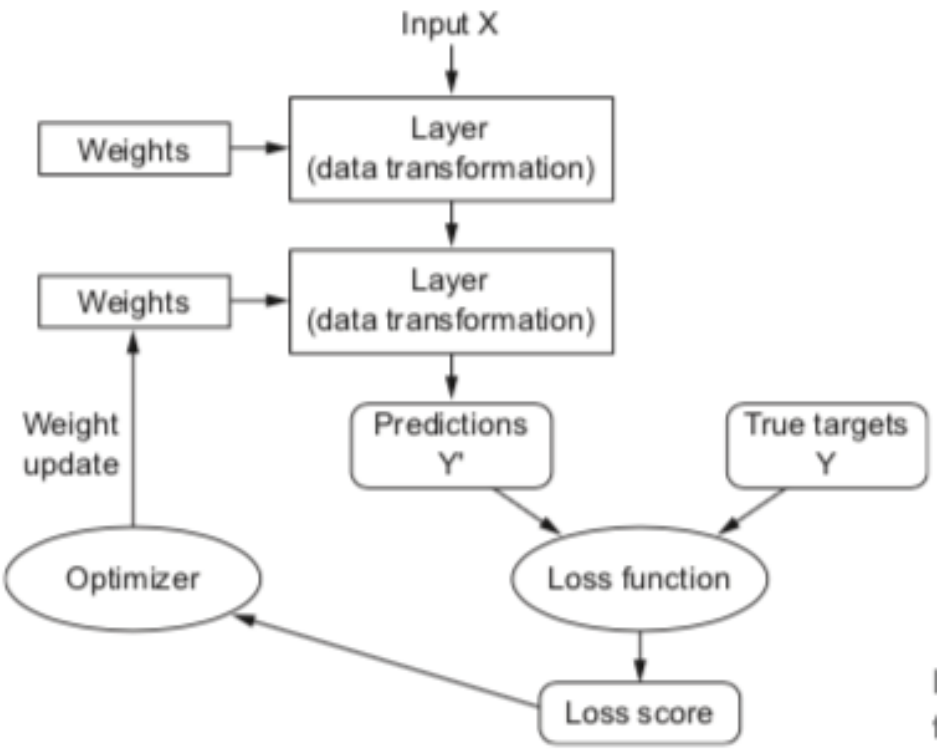
Ej. Programar las reglas del ajedrez, puertas lógicas…



0.1	0
0.091	0
0.081	0
0.96	1
0.091	0
0.091	0
0.091	0
0.091	0
0.091	0



Es importante destacar que **no se prescribe como cada capa procesara la información**, sino que **los pesos se irán ajustando para que el resultado, ante la matriz de pixels del input, sea el encendido de la cuarta celda**.



Distintos modos de cuantificar “Loss”

1. Mean sq. error. Usual en regresión. Facil de hacer cuentas por el cuadrado. Sensible a outliers

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$$

Mean Squared Error

2. Mean absolute error (como no usa cuadrados, no se arrastra por outliers. Pero es dificil de tartar analiticamente)

$$MAE = \frac{\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid}{n}$$

Mean absolute error

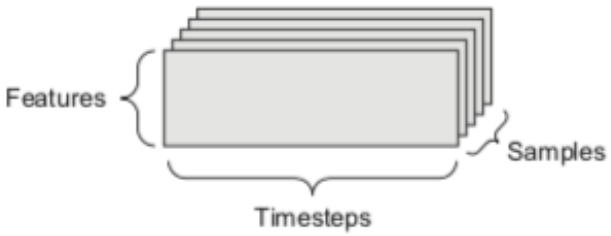
3. Cross entropy loss (**penaliza el error cometido confiadamente**) Si el label posta es y_i=1, el segundo termino es cero. Y si se predice $\hat{y}_i \approx 0$, da enorme. Lo mismo si y_i=0 y el $\hat{y}_i \approx 1$

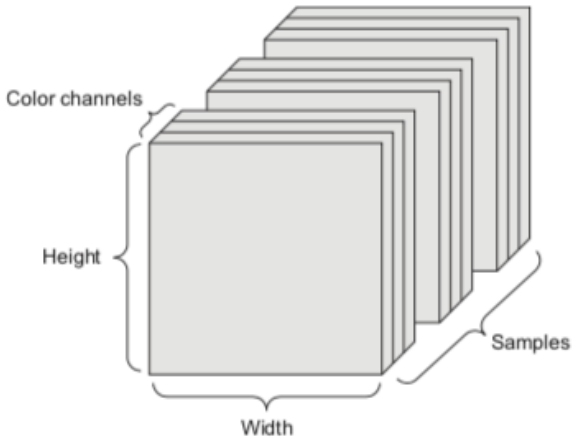
$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

Cross entropy loss

En la actualidad, existe un innumerable cantidad de software de alto nivel, que permite armar códigos para implementar técnicas de inteligencia artificial a diversos problemas, típicamente para clasificar, o realizar regresión. En este curso se discuten 3 códigos en la practica (uno para clasificar, uno para realizar predicciones, y uno para comprimir data). En general, gran parte del trabajo de un científico empleando estas herramientas consiste en **preparar los datos.**

Sobre entrada de datos





Arquitecturas en function de la estructura de los datos

(Muestras, features), capas densas

(Muestras, tiempos, features) Redes recurrentes

(Muestras, colores, alturas, anchos) Redes convolucionales

Arquitectura para clasificacion

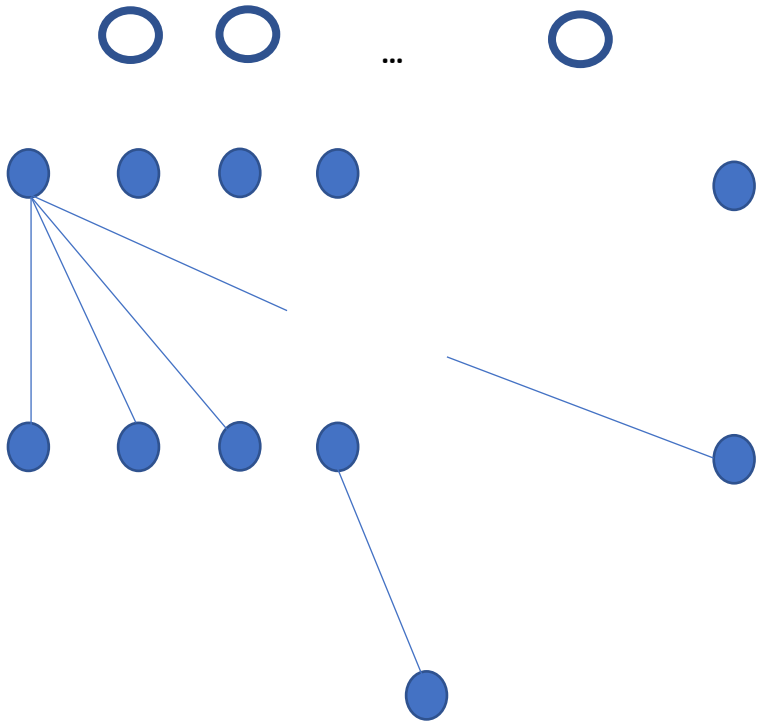
Ejemplo: recomendaciones de películas en IMDB (clasificación binaria)

Datos=[[1,54,3,10…,1],
[1,3,55,23…,2],
[…],
[],
[],
…,
[45,54,1,…,3]]

Donde cada vector es un conjunto de "**features**", dadas por un numero entero, que indican una palabra. O sea, se convierte el texto en un conjunto de enteros.

Los comentarios son clasificador como positivos o negativos, de modo que la salida o **target** de cada review son o unos o ceros:

Reviews=[[1],
[1],
…,
[0]]

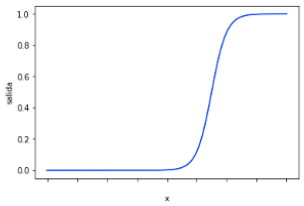


tantas unidades como features

16 unidades

16 unidades

1 unidad (sigmoidea)

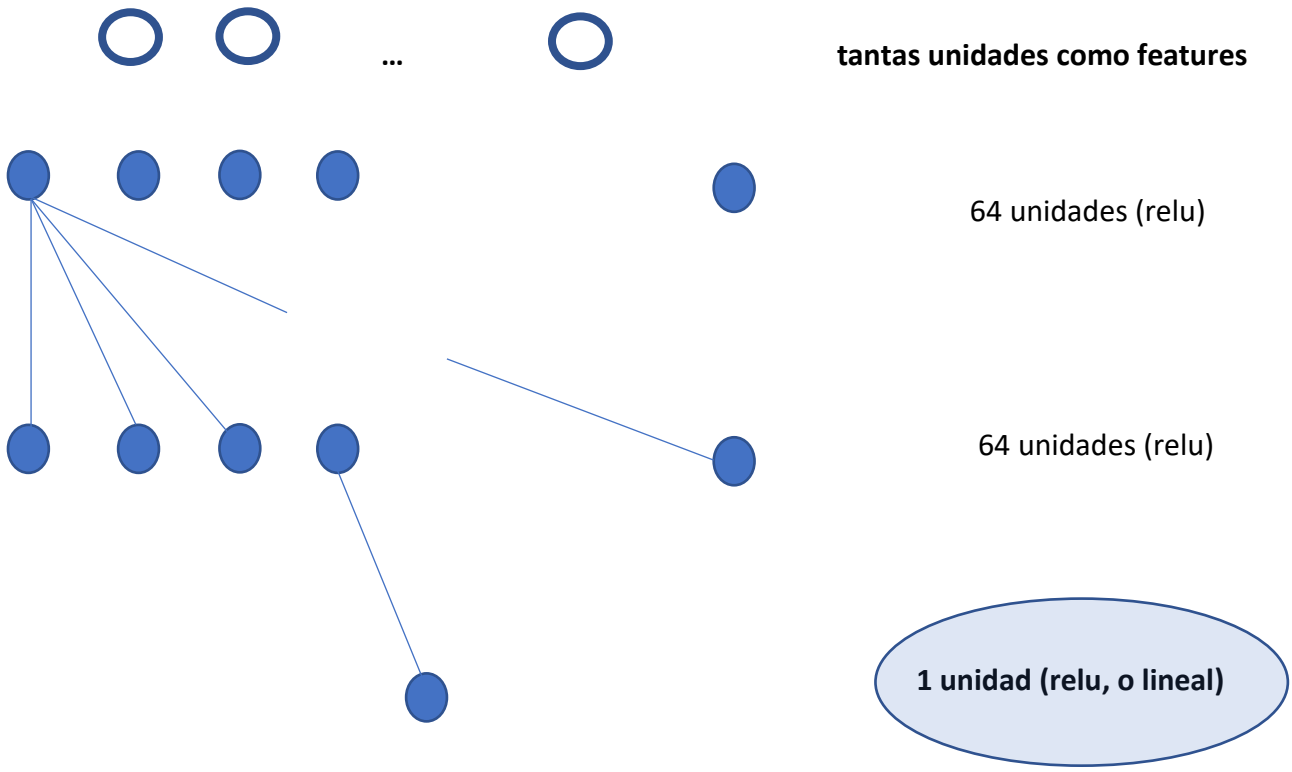


Arquitectura para regresión

Ejemplo: precio para casas en función de características.

Datos=[[124.2,1894,3,10…,1],
[1,3,55,23…,2],
[…],
[],
[],
…,
[45.3,1900,1,…,3]]

(ahora el primer real indica la superficie, el Segundo el código de área, etc…)



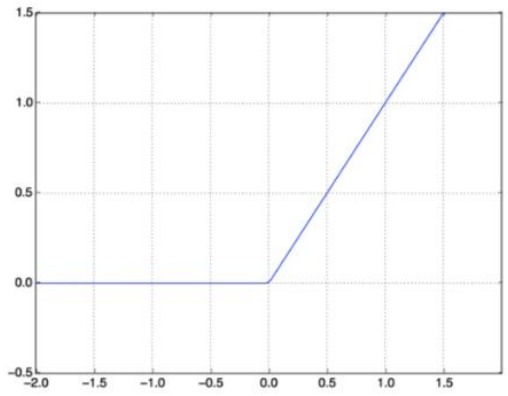


Figure 3.4 The rectified linear unit function

Que notemos una función de activación **relu** es una caricatura de lo que ocurre en un rango con una sigmoidea:

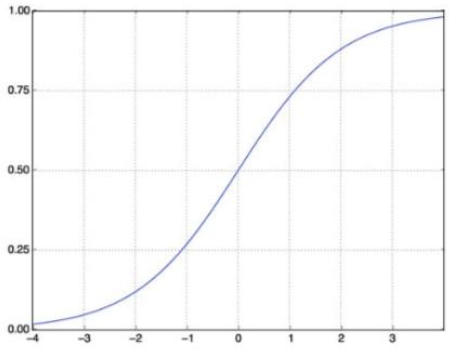
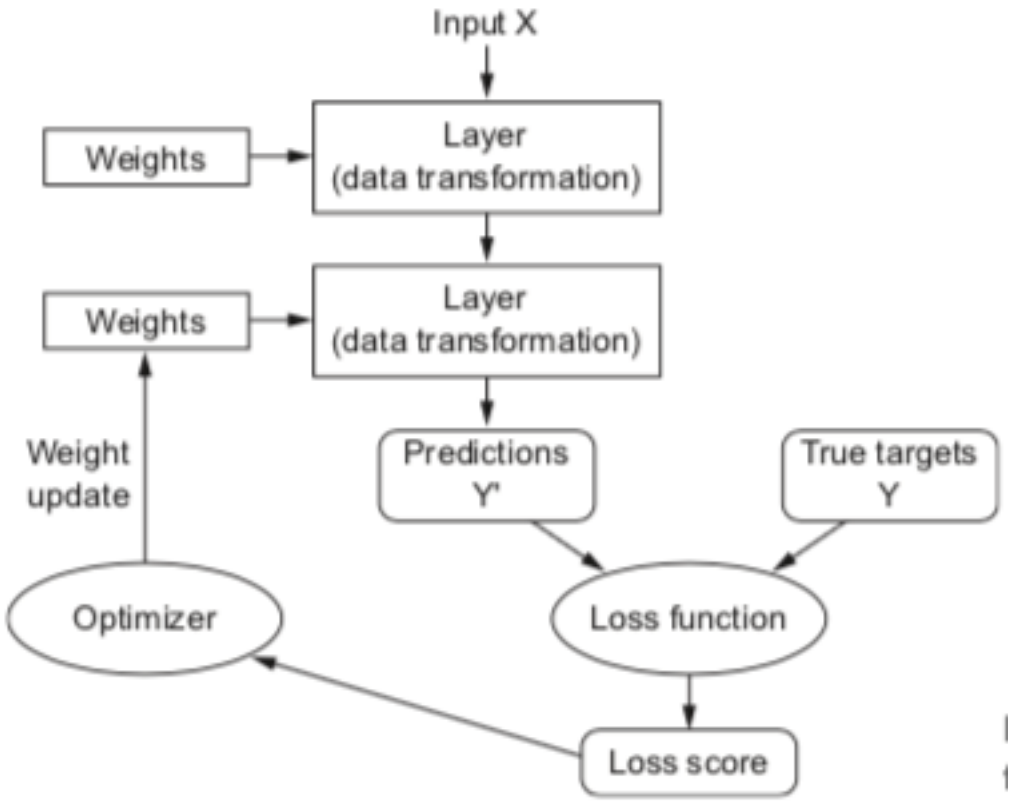


Figure 3.5 The sigmoid function

Ambos ejemplos siguen este esquema que mostramos hace un rato:



Solo difirió el tipo de **salida, o predicción**, de la red. En un caso era una pertenencia a una clase, en el otro, via un conjunto de valores continuos.

Y son, ambos, ejemplos de **aprendizaje supervisado.** Los problemas típicos de este tipo de aprendizaje son:

1. **Clasificación binaria**

   1.a. La ultima capa es una única unidad
   1.b. Esa unidad es una sigmoidea (da 0 o 1)

2. **Clasificación multi-clase**

   2.a. La ultima capa tiene tantas unidades como clases
   2.b. Las ultima unidades son sigmoideas (dan 0 o 1)

3. **Regresión**

   3.a. La ultima capa tiene una unidad (si es un escalar)
   3.b. La ultima capa tiene un numero de unidades iguales a las componentes del vector de propiedades.
   3.c. La ultima capa es una relu, que puede dar un rango continuo de valores.

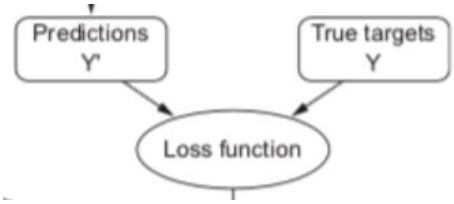
Aprendizaje no supervisado. Clustering, reducción de dimensionalidad…

Aprendizaje auto-supervisado: autoencoders (salida=entrada)

Advertencia: El que en principio sea sencillo el problema conceptualmente, no significa que la cosa funcione sin problemas. **Hablemos de los problemas.**

1. **Minimización del error**.

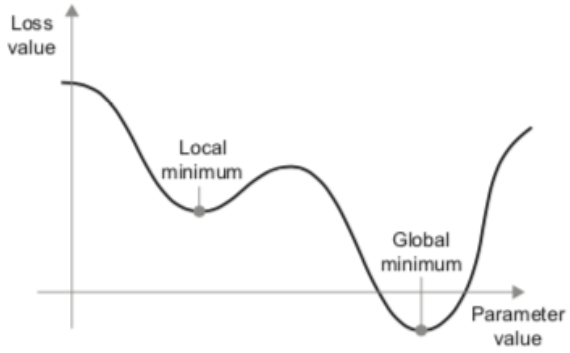
Cuando uno evalúa:



Y lo emplea para calcular los cambios en la red:

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}},$$

Con la idea de encontrar un mínimo en el cual $\Delta W_{ij} = 0$, resulta que

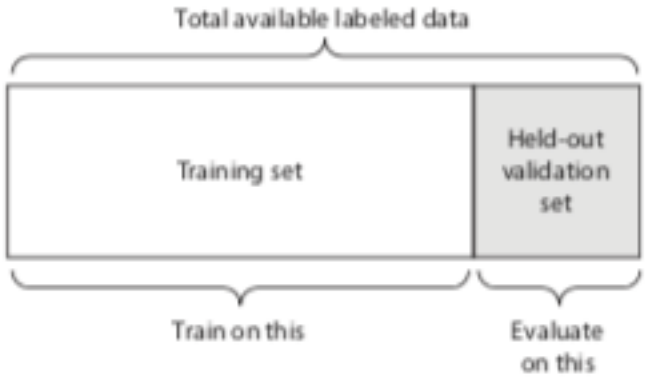


De modo que uno de los “hiper parámetros” del problema es cuanto se modifican los pesos en cada época. ¿Se sigue de largo? ¿Me quedo varado en un mínimo local?

2. Overfitting

Este es el gran enemigo de un problema de redes neuronales. Aprenderse los resultados del training set, y no poder generalizar. No poder resolver problemas nuevos. **¿Como se expresa eso, y como se corrige?**

En todo problema de entrenamiento de redes, lo primero que hacemos es partir los datos en un conjunto de entrenamiento, y uno de validación:

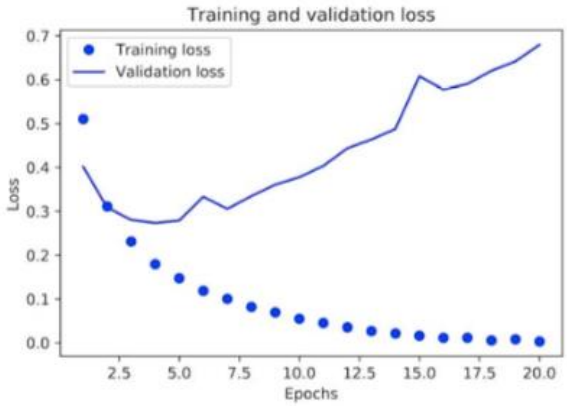


Entonces lo primero, es ir sistemáticamente a buscar que podamos acomodar, algorítmicamente a los parámetros de la red, los pesos, como para que los errores, se vayan minimizando. En la siguiente figura, eso se ve en el que la línea de puntos vaya decreciendo con las épocas. Pero además de hacer esa corrida que cambia parámetros, en cada época se evalúa en los elementos apartados, llamado conjunto de validación. Este conjunto puede, o no, ir también experimentando una disminución de los errores. Si esto no ocurre, decimos que sistema sobre-ajusta (existe overfitting).

Esta tensión entre

optimización (buen ajuste de los datos) y generalización (capacidad de predecir)

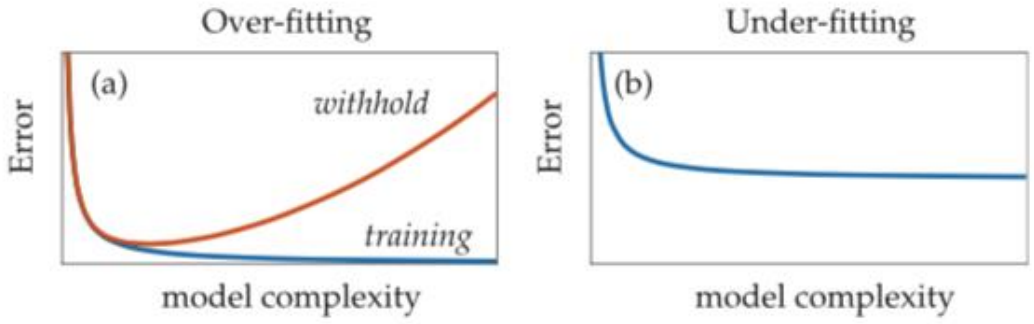
es la clave (o una de las claves de) de Machine Learning



Modos de evitar esto:

1. **Hacer chico al modelo**. Un modelo con muchos parametros tiene lugar para aprender cada ejemplo sin necesidad de generalizar. Hay que empezar con uno chico, que no ajuste al training set, y empezar a agrandar. Nunca al revés.
2. **Hay que regularizar**. Los modelos, para que sean sencillos y no ajusten caprichosamente a los datos, deben tener pocos pesos, y de tamaños comparables. Una red con un parámetro desproporcionadamente mas grande que el resto probablemente este haciendo contorsiones para ajustar algún conjunto particular de datos. Para esto se penaliza tener muchos parámetros distintos de cero.
3. Hay trucos, como el **dropout** (setear a cero algunos parámetros al azar)

Concepto: todo ML se basa en optimización, pero con una mirada en el **overfitting**, y el **underfitting**:



Flujo universal del trabajo en machine learning

1. **Tener en claro que entrada va a tener el problema**

(fragmentos de señales temporales, imágenes, películas, conjuntos de caracteristicas)

2. **Tener en claro el tipo de problema**

(vamos a hacer una regresión, una clasificación binaria, multi clase, una regresión vectorial)

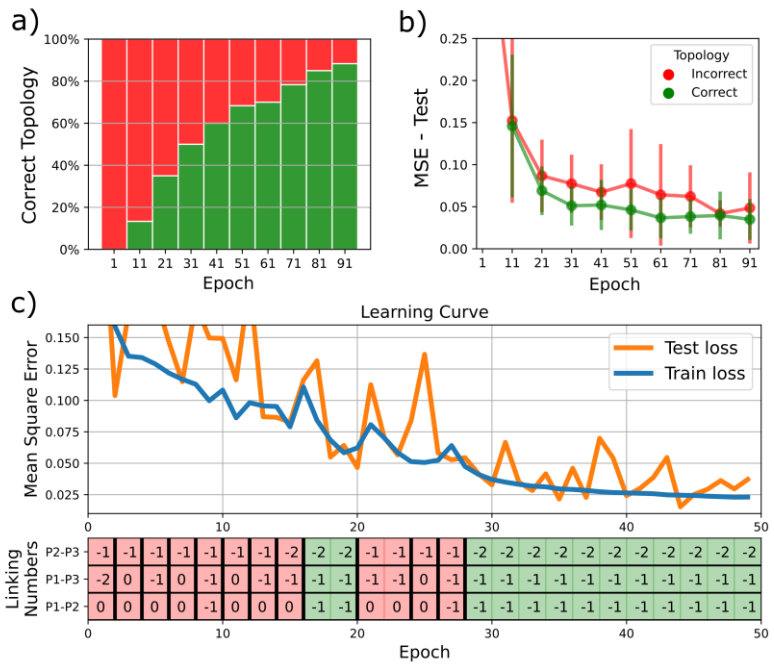
3. **Tener en claro las hipótesis del problema**

(que existe información en los segmentos de datos como para predecir el futuro, que el conjunto de features permite clasificar, etc.) Evita culpar a la red de no poder hacer lo imposible, o de matarse buscando hiper parametros (numero de capas, neruronas, etc)

4. **Definir el protocolo de evaluación**

(cuantos elementos son los que nos guardamos para validar)

5. **Definir la medida del éxito**



(¡puede no se el MSE! ¡Puede ser la topología!)

6. **Tener claro el baseline que queremos batir.**

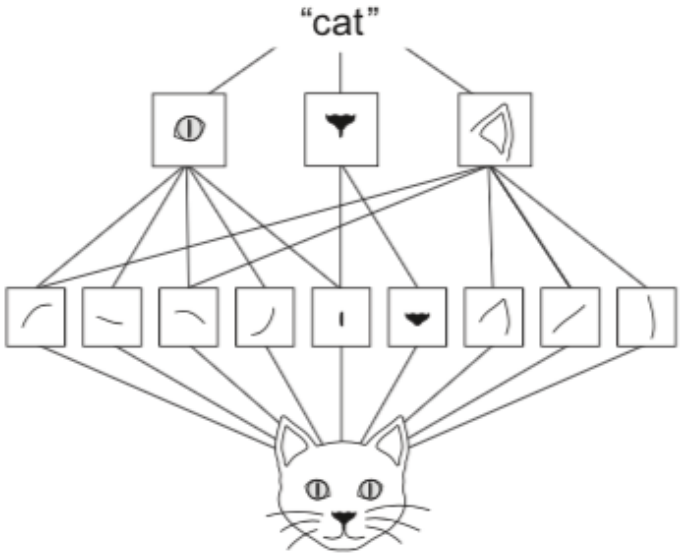
(si tenemos dos categorías, tener mas que 50-50, por ejemplo)

7. **Implementar arquitecturas cualitativamente viables**

(si vamos a hacer regresión, poner una ultima capa lineal, si vamos a hacer clasificación, una ultima capa sigmoidea, que las funciones de error sean las optimas (mse para regresiones, pero crossentropies para clasificaciones, por ejemplo))

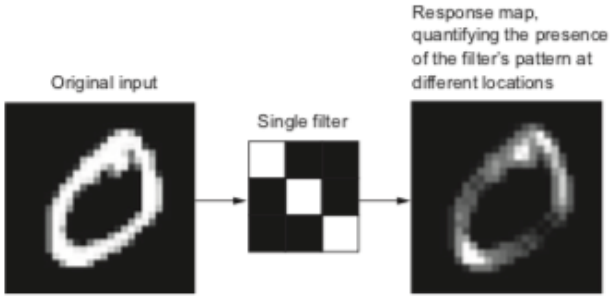
Una red estelar: la red de neuronal de convoluciones (CNN)

Asi como las **redes densas** aprenden **globalmente**, las redes **convolucionales** aprenden **patrones locales**. La identificación de esos elementos es clave en el proceso de clasificar.

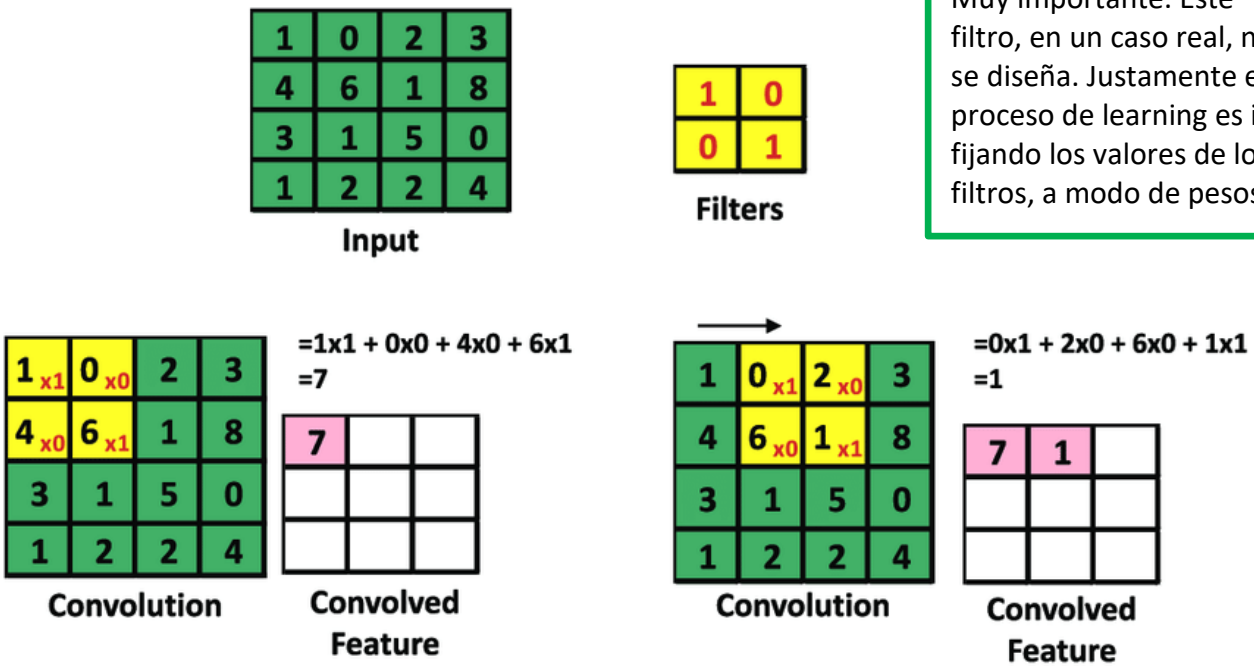


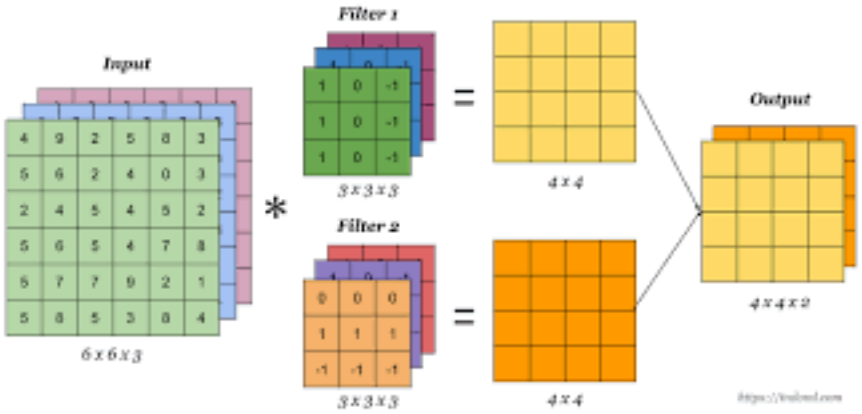
La operación clave en el proceso de identificar propiedades **locales** es la **convolucion** con un **filtro pequeño.**

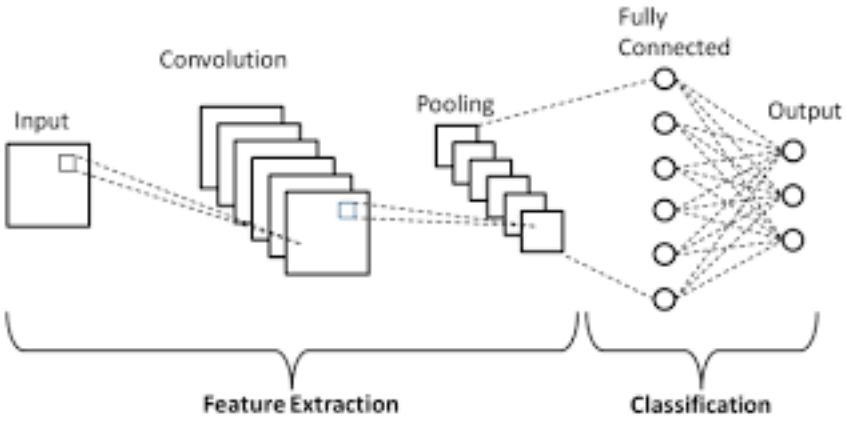
Cada capa es un nivel de representación de los datos. Típicamente la primera detecta bordes, la segunda motivos de bordes, la tercera los ajusta en partecitas significativas…



Matemáticamente, operan así:







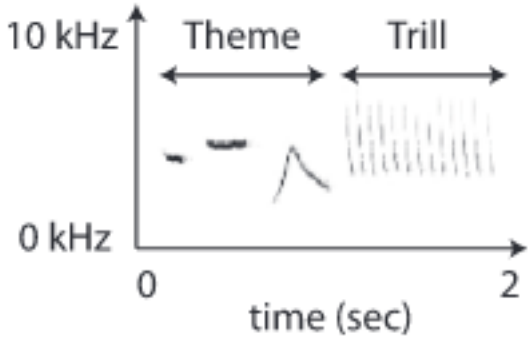
Pooling

El objetivo del pooling es reducir la escala espacial de la representación, de modo de reducir el numero de parámetros de la representación de los datos. Con esto

1. Se controla el overfitting
2. Se acota la memoria necesaria

Muy típico es hacer un max pooling, que toma el valor máximo en ventanas de 2x2, con un stride de 2 (saltando dos casilleros al mover la ventana). Asi se subsamplea por 2.

Ejemplo: identificación de chingolos por su voz.



$$\frac{dx}{dt} = y$$

$$\frac{dy}{dt} = \kappa\gamma^2 x - \gamma x^2 y + \beta\gamma y,$$

where x stands for the midpoint labial position, κ, β are system parameters, and γ is the temporal scaling of the problem. To generate sound with this labial dynamics, the pressure at the tracheal input p_i is computed as:

$$p_i(t) = \mathrm{A}\, x(t) + p_{back}\left(t - \frac{L}{c}\right)$$

$$p_{back}(t) = -r\; p_i\left(t - \frac{L}{c}\right)$$

Y asi generamos con un modelo, una base de datos para entrenar a la red. Partiendo del canto de 6 individuos, estimamos su variación, y generamos cantos sintéticos que consisten en variaciones de las pocas grabaciones conseguidas. O sea, usamos dinámica no lineal para generar cantos sinteticos que entrenaran a la red.

bird 1

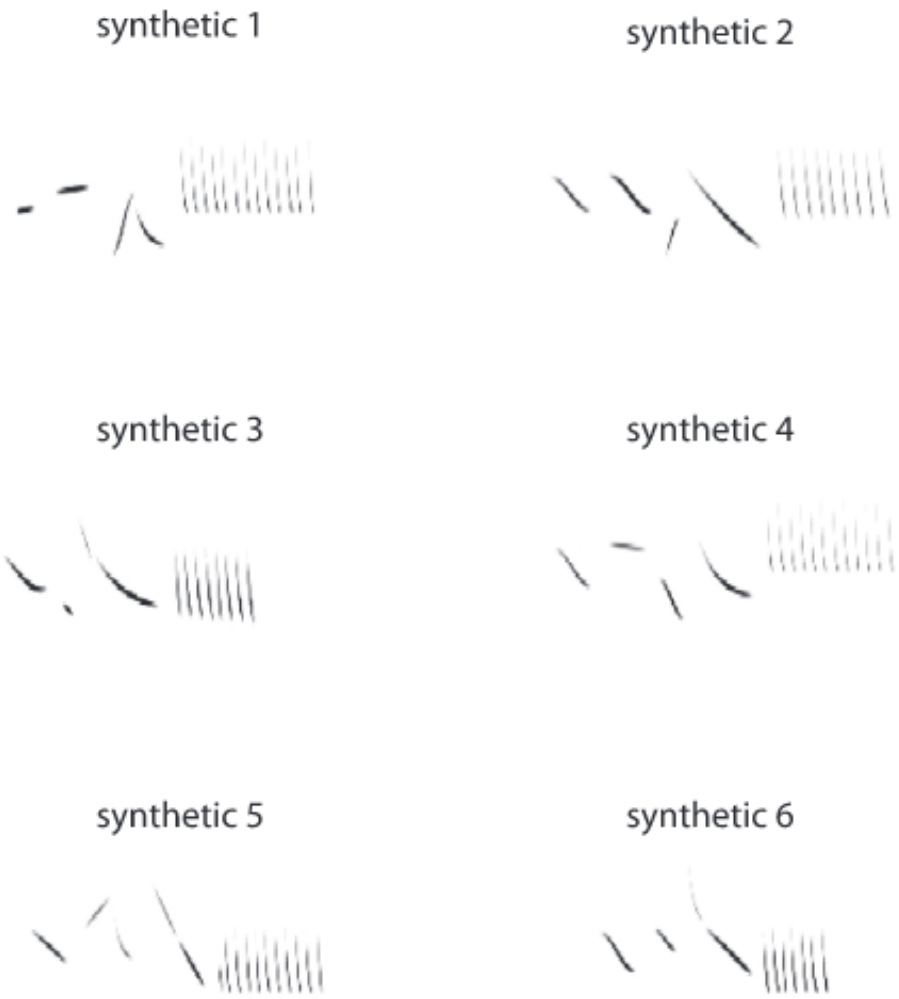
bird 2

bird 3

bird 4

bird 5

bird 6



Con varios miles de ejemplos sintéticos para cada clase, entrenamos una red convolucional de 8,16,16,32 filtros, de (3,3), cada capa alternando con MaxPooling, y terminando en una capa doble densa de 1024 x 6 unidades. Las ultimas 6 unidades dan cuenta de que queríamos distinguir entre 6 individuos.

El punto clave es que no haya que diseñar, por la propia experiencia, cual es un conjunto de propiedades que sean. Clave (ej, numero de silabas introductorias, o las frecuencias iniciales de las mismas), si no confiar en que el ajuste de los parámetros encuentre esos “filtros” a partir de los datos crudos.

Confusion matrix, computed for one numerical experiment.

Actual\predicted	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	9	0	0	0	1
3	0	0	8	0	0	1
4	0	0	2	3	1	0
5	0	0	1	0	8	0
6	1	0	0	0	0	8

Table 3
Precision, recall and f1_score, computed from a numerical experiment.

Class	*P*	*R*	*f1_score*	Support
1	1.00	1.00	1.00	1
2	1.00	1.00	1.00	10
3	0.57	0.89	0.70	9
4	1.00	0.5	0.67	6
5	1.00	0.67	0.80	9
6	0.9	1.00	0.95	9
Avg./tot	0.89	0.84	0.84	44

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Observación sobre ajustes de parámetros.

En el fondo, estamos hablando de encontrar parámetros que minimizan la diferencia entre un objetivo y una funcion de características de una entrada.

$$f(x) = \beta_1 x + \beta_2$$

to this data, the error E_2 is found by minimizing the sum

$$E_2(f) = \sum_{k=1}^{n} |f(x_k) - y_k|^2 = \sum_{k=1}^{n} (\beta_1 x_k + \beta_2 - y_k)^2 .$$

$$\frac{\partial E_2}{\partial \beta_1} = 0 : \quad \sum_{k=1}^{n} 2(\beta_1 x_k + \beta_2 - y_k) x_k = 0$$

$$\frac{\partial E_2}{\partial \beta_2} = 0 : \quad \sum_{k=1}^{n} 2(\beta_1 x_k + \beta_2 - y_k) = 0 .$$

Upon rearranging, a 2×2 system of linear equations is found for A and B

$$\begin{pmatrix} \sum_{k=1}^{n} x_k^2 & \sum_{k=1}^{n} x_k \\ \sum_{k=1}^{n} x_k & n \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^{n} x_k y_k \\ \sum_{k=1}^{n} y_k \end{pmatrix} \longrightarrow \mathbf{Ax} = \mathbf{b}.$$

Y si se propone una expresión **polinómica**, la extensión es muy sencilla.

$$f(x) = \beta_1 x^2 + \beta_2 x + \beta_3 \tag{4.14}$$

where now the three constants β_1, β_2, and β_3 must be found. These can be solved for with the 3×3 system resulting from minimizing the error $E_2(\beta_1, \beta_2, \beta_3)$ by taking

$$\frac{\partial E_2}{\partial \beta_1} = 0 \tag{4.15a}$$

$$\frac{\partial E_2}{\partial \beta_2} = 0 \tag{4.15b}$$

$$\frac{\partial E_2}{\partial \beta_3} = 0\,. \tag{4.15c}$$

Pero ni bien nos alejamos de expresiones **lineales o polinómicas**…

Although a powerful method, the minimization procedure for general fitting of arbitrary functions results in equations which are nontrivial to solve. Specifically, consider fitting data to the exponential function

$$f(x) = \beta_2 \exp(\beta_1 x)\,. \tag{4.16}$$

The error to be minimized is

$$E_2(\beta_1, \beta_2) = \sum_{k=1}^{n} (\beta_2 \exp(\beta_1 x_k) - y_k)^2\,. \tag{4.17}$$