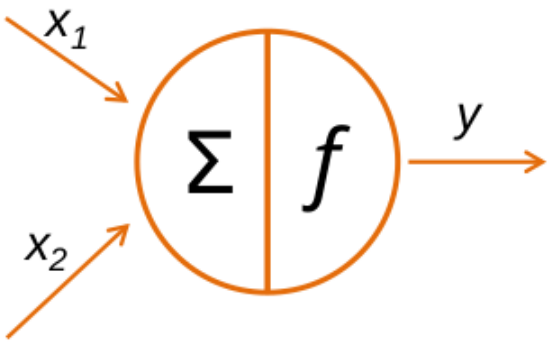
Deep Learning

02 Trabajo con Redes

Miguel A.
Castellanos

Para construir una Red neuronal necesitamos:

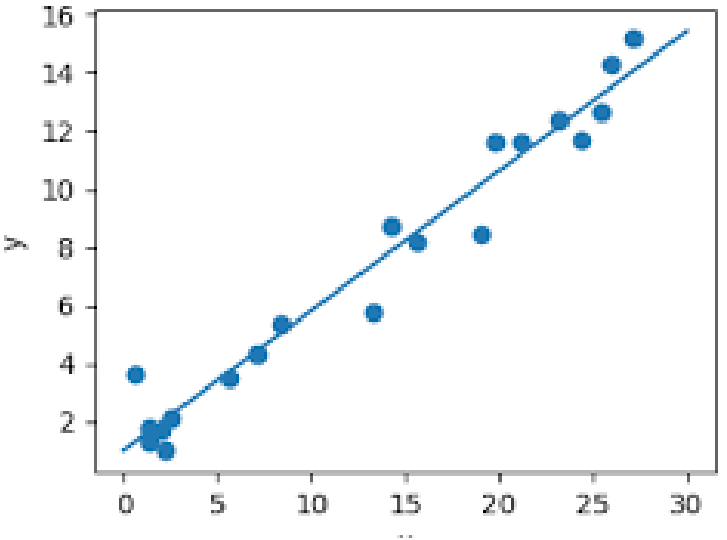
- Un conjunto de datos
- Un modelo de neurona
- Una función de activación
- Una estructura o arquitectura
- Una función de coste
- Un algoritmo de aprendizaje



La función de coste (L):

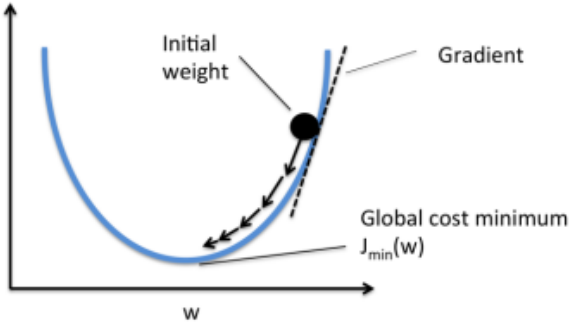
Dependiendo de la naturaleza de *y* e *y'* se utilizan distintas funciones de perdida:

- Datos continuos: *mse*
- Categóricos binarios: *entropía binaria*
- Categóricos múltiples: *entropía categórica*



El algoritmo de aprendizaje:

Todos parten de la función de perdida y van cambiando los valores de los pesos para minimizarla. Es lo que se llama el *backpropagation.*



El más conocido es el *descenso del gradiente*, todos usan la misma lógica o son derivados de él.

Un gradiente es una derivada, se busca minimizar los errores buscando los valores de W y b que hacen mínimo el valor de coste

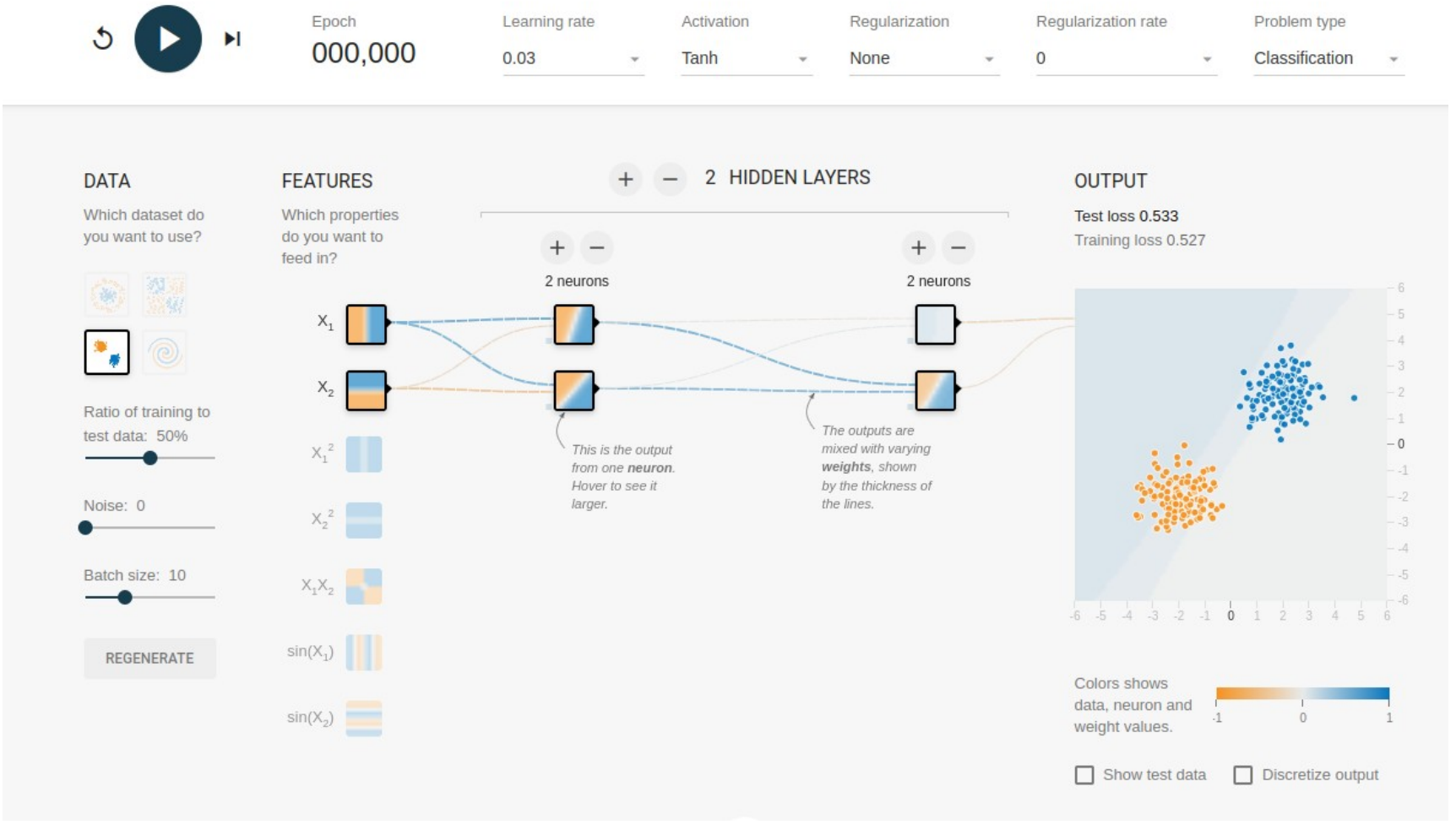
Como existen múltiples capas esa derivada se va propagando hacia atrás a través de las capas

Se basa en la **regla de la cadena** (derivadas encadenadas): Si tenemos una función de una función, se puede calcular la derivada en forma de cadena:

Si f es una combinación de g: $f(g(x))$ entonces:

$$\frac{\partial f(g)}{\partial x} = \frac{\partial f(g)}{\partial x}\frac{\partial g}{\partial x}$$

Google Playground

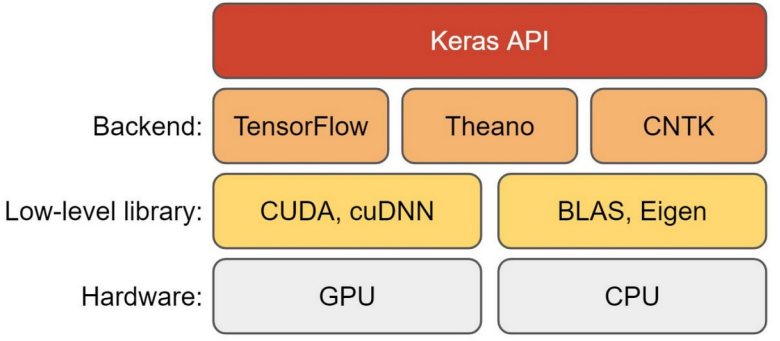


Cómo se trabaja en Deep Learning

Se programan en Python

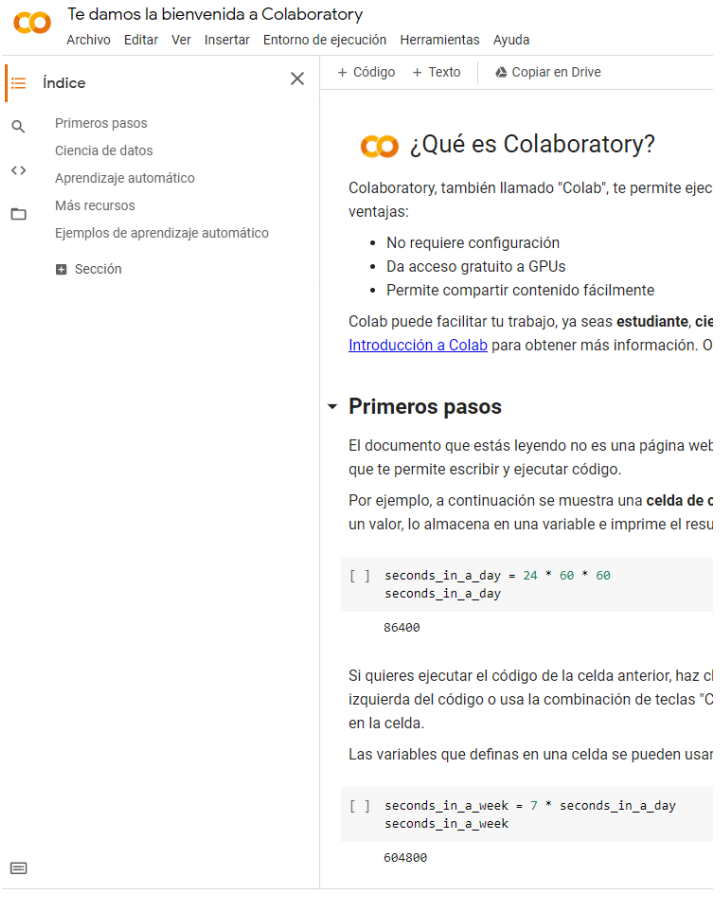
Para poder trabajar eficientemente se necesita *hardware* específico (GPUs y otras arquitecturas que están apareciendo), en concreto las tarjetas Nvidia

Controlar directamente ese *hardware* es complejo, por eso se han desarrollado APIs (application programming interface)



Colab (google) como entorno de desarrollo

- Vamos a trabajar con *tensorflow2* y google, usando herramientas para notebooks basados en el entorno de *colab: https://colab.research.google.com*
- Podemos hacer de todo con colab pero hay poca potencia de cálculo
- La instalación en un ordenador del entorno necesario puede ser compleja pero el que quiera puede seguir las instrucciones en https://www.tensorflow.org/install
- En general, todo esto funciona mejor si usas ***Linux***



- Abrimos ***colab*** (http://colab.research.google.com/)
- Necesitamos una cuenta de google
- *Colab* tiene preinstalado todo lo que necesitamos: GPU, tensorflow2, Keras, Python…
- Abrimos nuestro primer cuaderno: *Archivo → nuevo cuaderno*
- Y escribimos nuestro primer código en python

  print(“hola mundo”)
- y ejecutamos (flecha o entorno de ejecución)
- Este tipo de cuadernos (*notebooks*) que mezclan código con explicaciones son fundamentales en ciencia de datos (como *jupyter*)

Añadimos una ventana de texto y explicamos lo que vamos a hacer y en una de código escribimos lo siguiente

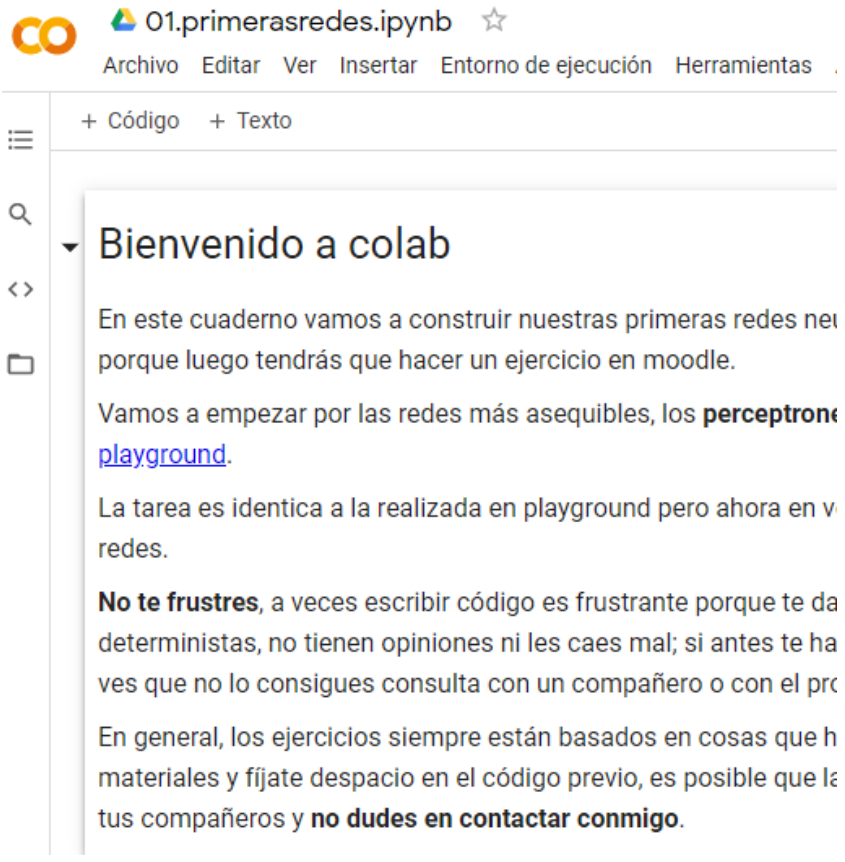
```
# crear una variable
x = 10
y = x + 15 * 3

# Mezclar texto con cadenas
print("El numero es: ", y)

# importar un paquete de python y usarlo
import random
z = random.randint(-10,10)
print(z)
```

Pues ya está, ya sabemos python

Vamos a abrir un cuaderno creado por mí:
Archivo -> abrir cuaderno -> Github

Buscamos mi usuario mcstllns y un repositorio llamado DeepLearning
Cargamos el fichero:
01.primerasredes.ipynb

Seguimos las explicaciones dentro del notebook abierto



Ejemplo 01. Regresión con datos de felicidad

Se analizan unos datos obtenido de *Kaggle.com* sobre la encuesta de felicidad de 2021

Una vez limpiados y preparados la base de datos consta de 9 variables predictoras y una variable criterio:

Criterio: Life.Ladder.

Predictoras: year, Log.GDP.per.capita, Social.support, Healthy.life.expectancy.at.birth, Freedom.to.make.life.choices, Generosity, Perceptions.of.corruption, Positive.affect, Negative.affect.

El mejor modelo calculado con estadística convencional (modelo lineal) da un **MSE de 0.295**

Ejemplo 01. Regresión con datos de felicidad

```
Call:
lm(formula = Life.Ladder ~ ., data = d2)

Residuals:
     Min       1Q   Median       3Q      Max
-1.69545 -0.28756  0.03302  0.29949  1.59799

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                      0.01031    0.01181   0.873 0.382913
year                            -0.04468    0.01337  -3.341 0.000853 ***
Log.GDP.per.capita               0.38808    0.02638  14.713  < 2e-16 ***
Social.support                   0.19408    0.01856  10.458  < 2e-16 ***
Healthy.life.expectancy.at.birth 0.19166    0.02353   8.144 7.31e-16 ***
Freedom.to.make.life.choices     0.06656    0.01735   3.836 0.000130 ***
Generosity                       0.05631    0.01346   4.183 3.03e-05 ***
Perceptions.of.corruption       -0.11795    0.01469  -8.030 1.80e-15 ***
Positive.affect                  0.18592    0.01634  11.377  < 2e-16 ***
Negative.affect                  0.02524    0.01458   1.731 0.083694 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4872 on 1698 degrees of freedom
  (241 observations deleted due to missingness)
Multiple R-squared:  0.7725,    Adjusted R-squared:  0.7713
F-statistic: 640.7 on 9 and 1698 DF,  p-value: < 2.2e-16
```

```
> (mse <- mean(full.model$residuals^2))
[1] 0.2359338
>
```

Tipo de problema	Función de activación (última capa)	Función de coste
Regresión	linear	mse
Clasificación binaria	sigmoid	binary_crossentropy
Clasificación multiple con one-hot	softmax	categorical_crossentropy
Clasificación multiple con índice de categoría	softmax	sparse_categorical_crossentropy

Conjuntos de entrenamiento, desarrollo y prueba

- Se suelen crear tres conjuntos de datos: training (train), development (dev) y test
- El conjunto *train* sirve para entrenar la red
- El *dev* se utiliza para ir haciendo pruebas de ajuste y ver como se comporta el algoritmo cada ciertas etapas,
- Cuando todo el aprendizaje ha terminado se pone a prueba el modelo final con el test

Explicar la diferencia entre test y train

- En principio se suele hacer un reparto proporcional (pe: 80%, 10%, 10%) pero si hay muchos datos no hace falta que sean tan grandes
- Los tres deben provenir de la misma distribución de datos (es decir que sean del mismo conjunto).
- La red solo reproduce lo que aprende, no tiene sentido si provienen de distribuciones diferentes (salvo para estudiar la capacidad de generalización)
- Lo mínimo es tener al menos train y test, y lo bueno tener los tres.

Ejemplo 02. Clasificación con datos de infidelidad

Se analizan unos datos obtenido de Kaggle.com sobre una encuesta de infidelidad

Las variables son:

- **Criterio**: affairs.
- **Predictoras**: gender, age, yearsmarried, children, religiousness, education, occupation, rating.

La variable criterio es de tipo dicotómico 0 = no, 1 = sí por lo que el análisis convencional nos lleva a un modelo binomial. La precisión (*accuracy*) en la clasificación obtenida con este modelo es de 0.72

Ejemplo 02. Clasificación con datos de infidelidad

```
Call:
glm(formula = affairs ~ ., family = binomial, data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5963  -0.7369  -0.5512  -0.2721   2.4767

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    1.561062   1.089795   1.432 0.152019
gender        -0.183842   0.269388  -0.682 0.494960
age           -0.027035   0.019982  -1.353 0.176064
yearsmarried   0.062982   0.036032   1.748 0.080469 .
children       0.465497   0.329965   1.411 0.158319
religiousness -0.359633   0.100006  -3.596 0.000323 ***
education      0.008901   0.055727   0.160 0.873103
occupation     0.050717   0.080355   0.631 0.527934
rating        -0.500006   0.102111  -4.897 9.75e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Modelo entrenado con *train* y evaluado con *test*

```
> (tab <- table(test$affairs, p))
   p
     0  1
  0 85  4
  1 29  2
> (accuracy <- sum(diag(tab)) / sum(tab))
[1] 0.725
```

Sesgo (*bias*) y varianza (*variance*)

- El primero se refiere a la precisión obtenida con el *training set*
- El segundo a la diferencia entre la precisión del train y el test
- En general siempre el Error es mayor en el test que en el training

Explicar overfit

Training	Test	
0,5%	1%	Perfecto
15%	30%	Desastre
15%	16%	***Bias***: No estás entrenando bien
1%	15%	***Overfitting***: No estás generalizando bien

A veces, para comparar la actuación de la red, se suele comparar con la actuación humana, diciendo cuánto es mejor o peor en el test que un grupo de humanos

Overfitting

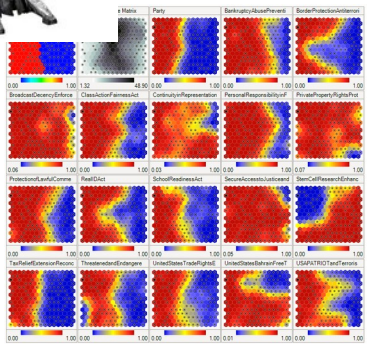
- Lo normal es que vayas detectando que se va a producir usando el dev-set
- Es muy común y hay que pensar siempre en él
- Es uno de los problemas más serios del Deep Learning

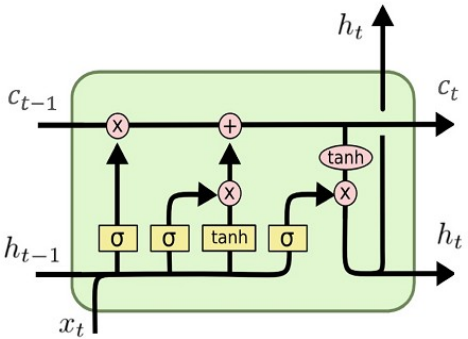
Siguientes cuestiones

Se puede solucionar cambiando algunas cosas de la red, arquitectura, algoritmos, etc. Las soluciones más utilizadas:

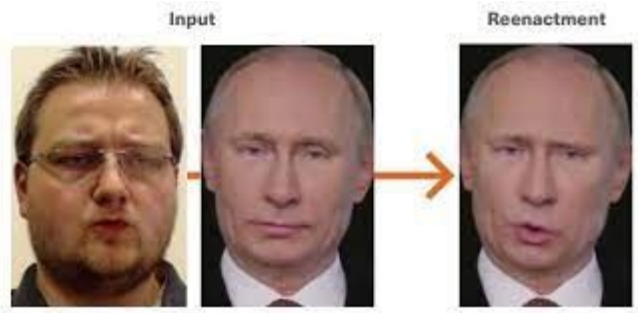
- Regularización L1 y L2
- Decaimiento de los pesos
- *Dropout*
- Normalización por lotes (*Batch normalization*)
- *Early stop*
- Normalización de los *inputs*
- Reducir el tamaño del modelo
- *Data Augmentation*

Siguientes pasos en el aprendizaje

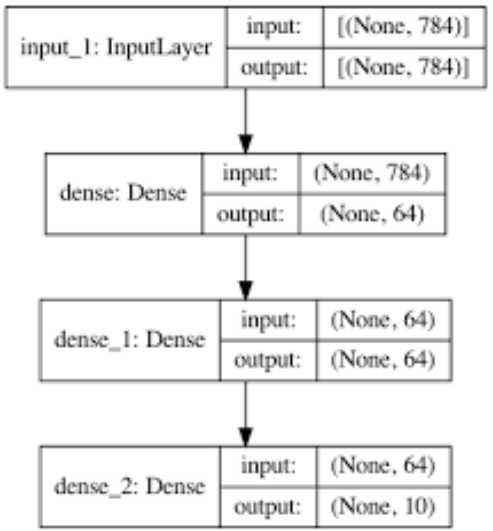


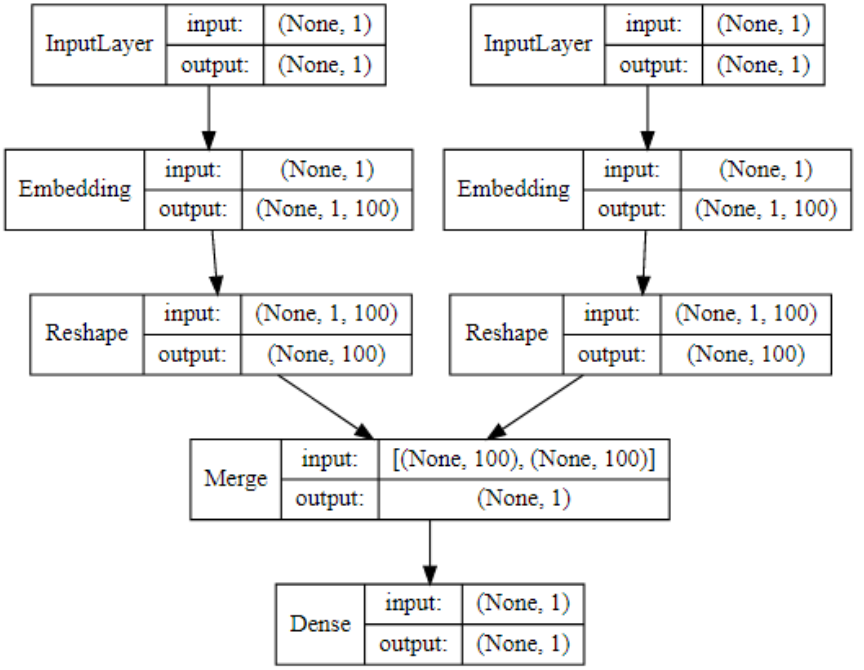


LSTM
(Long-Short Term Memory)

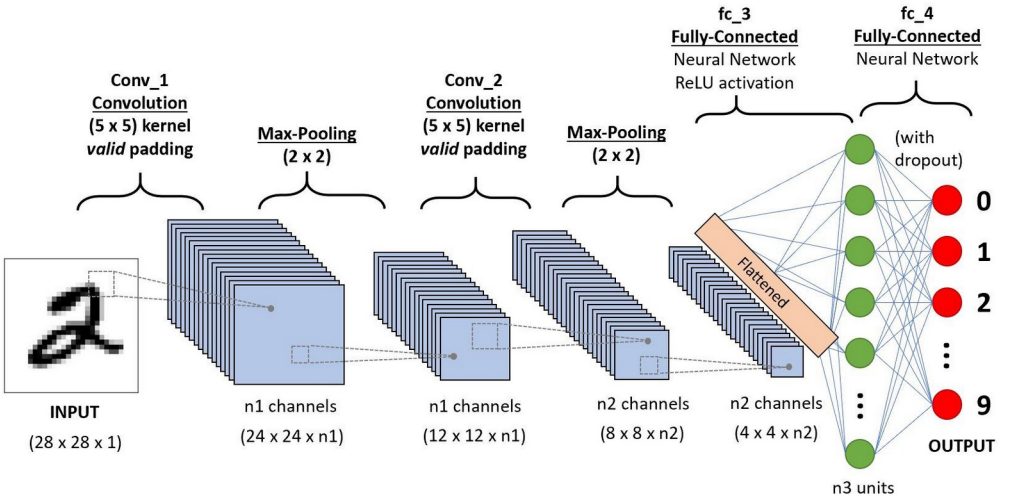


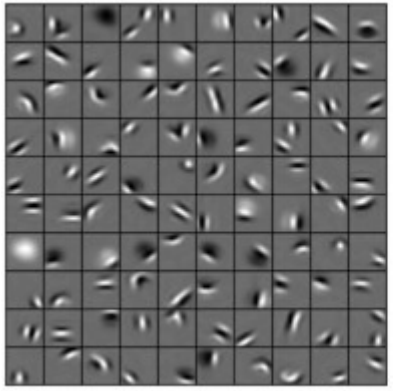
Modelos funcionales

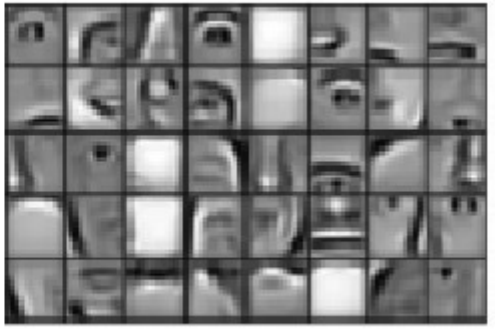




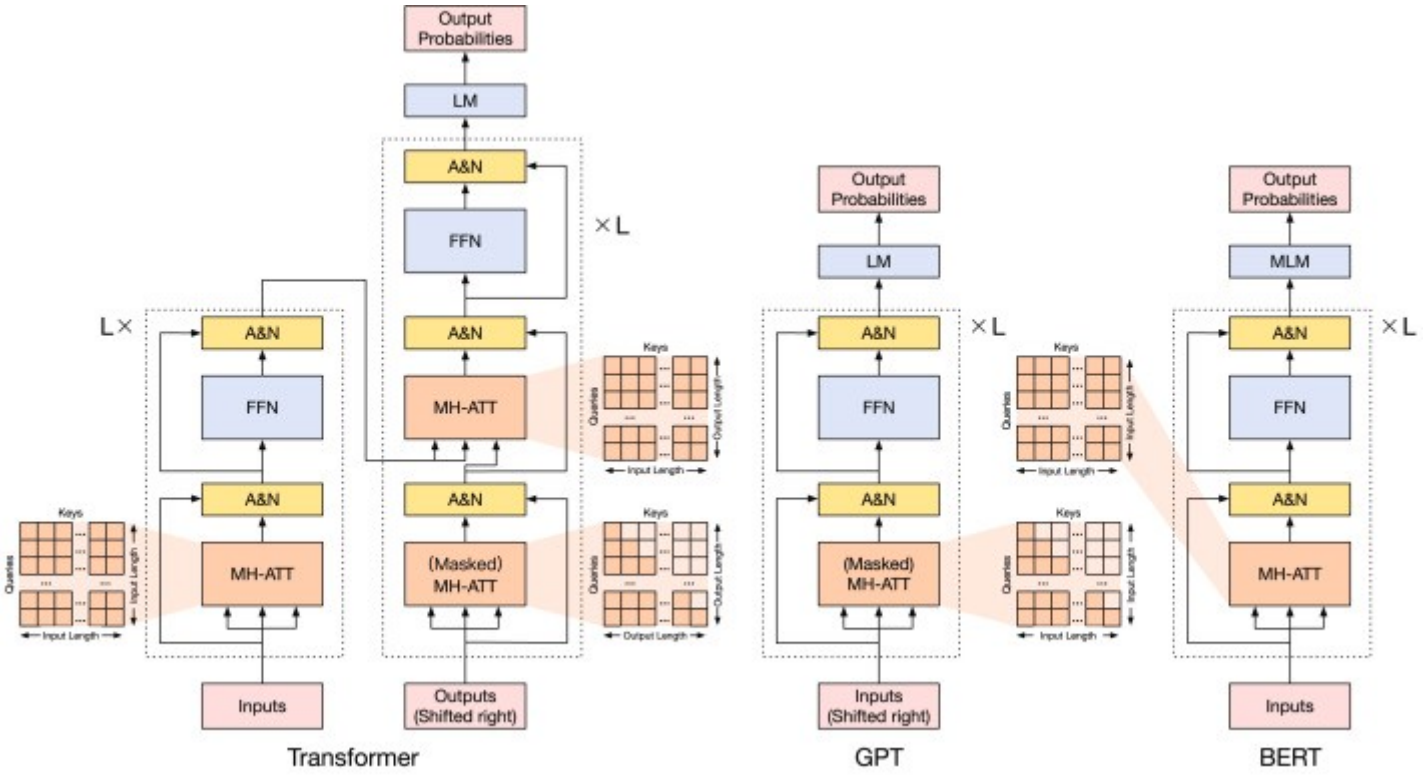
Modelos convolucionales



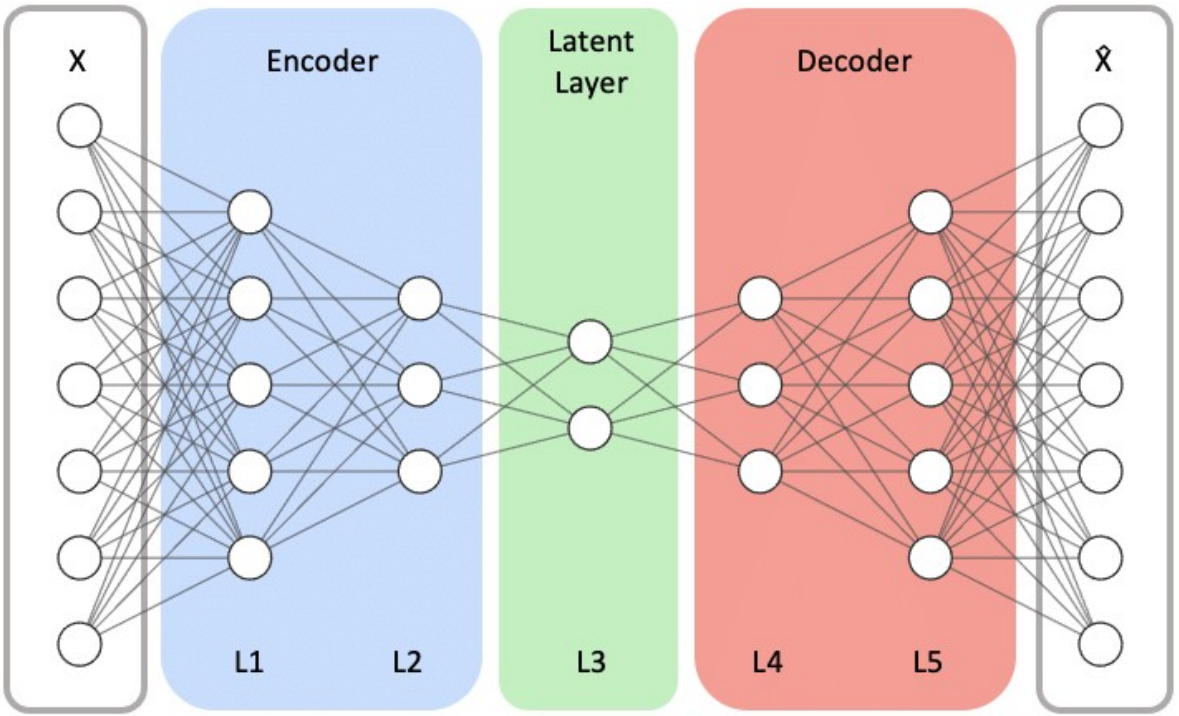




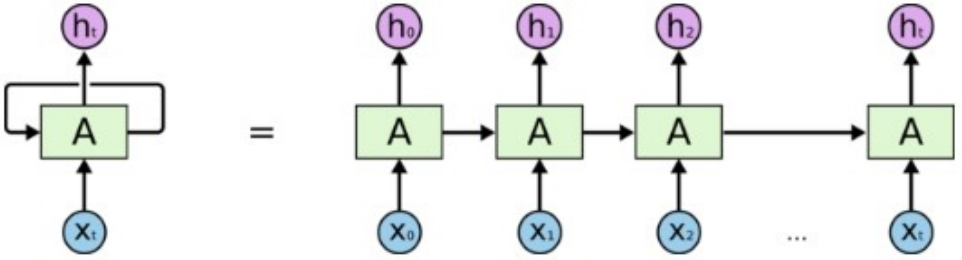
Modelos preentrenados



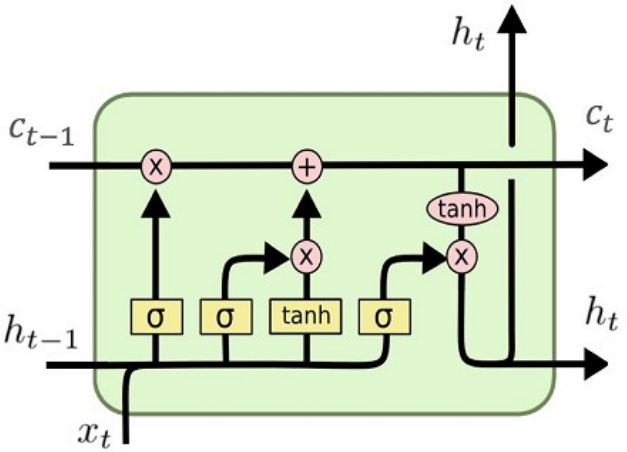
Autoencoders



Recurrent Neural Network y LSTM

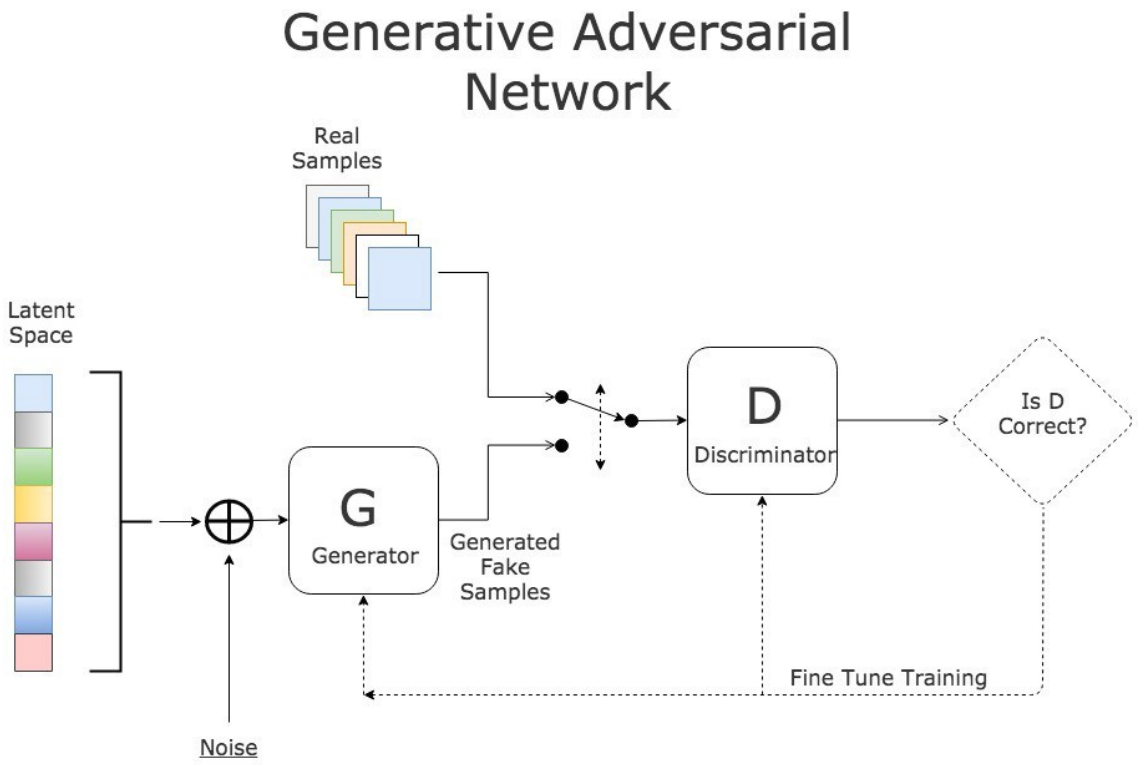


An unrolled recurrent neural network.



LSTM
(Long-Short Term Memory)

Redes Antagónicas (GAN)



Transformers (y mil cosas más)