



Argentina  
programa  
4.0



Universidad  
Nacional  
de San Martín

# Módulo 3

# Aprendizaje Automático



Argentina  
programa  
4.0



Universidad  
Nacional  
de San Martín

---

# Módulo 3

# Aprendizaje Automático

Semana 10 – Repaso

# Contenidos del módulo

## Deep Learning

## ML Clásico

- Árboles de Decisión
- Métodos de Ensemble
  - Bagging / Pasting → Random Forests
  - Boosting
- Support Vector Machines

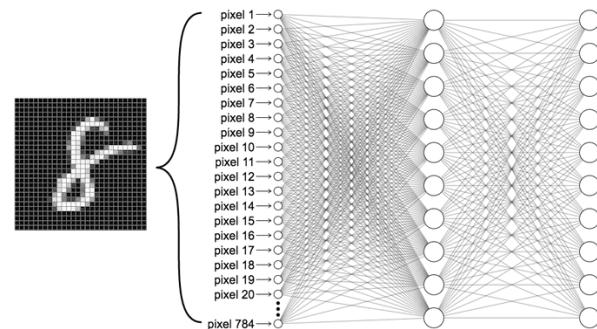
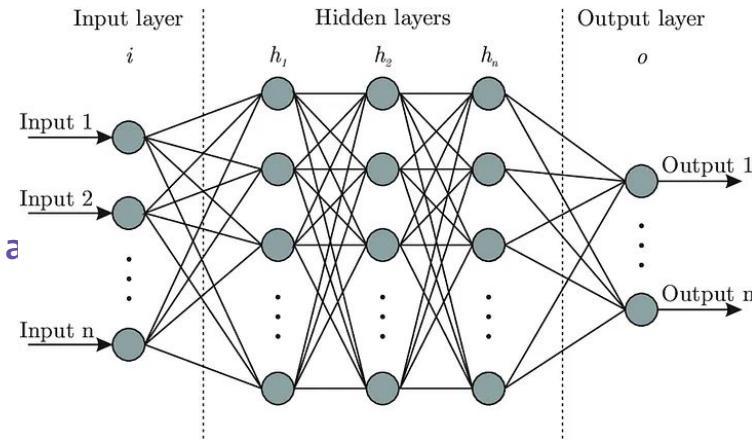
- Redes Neuronales
- Redes Neuronales Convolucionales
- Auto-Encoders / Auto-Encoders Variacionales
- Redes Neuronales Recurrentes (LSTM, otras)
- Extras:
  - Generative Adversarial Networks (GAN)
  - Reinforcement Learning

# Repaso redes neurales

## Capas visibles: input, output

### Input:

- estas son las variables (features) de los datos, que vamos a usar para aprender y hacer predicciones.
- Una neurona input por feature
  - Para datos tabulares, estas serían las columnas
  - Para imágenes, estas son las dimensiones
    - Ej  $28 \times 28 = 784$  (MNIST)
    - Ej  $224 \times 224 \times 3 = 150,528$  (ImageNet)

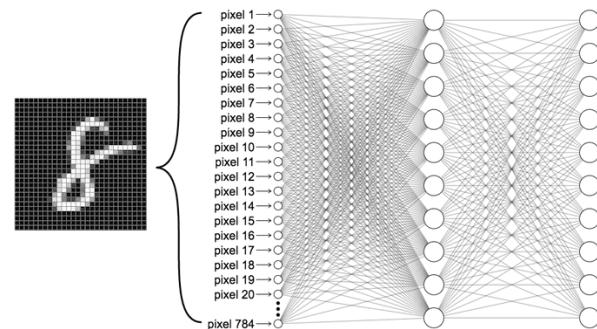
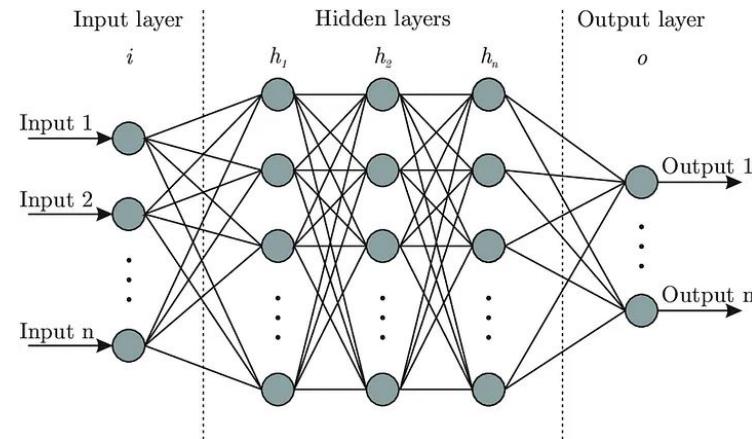


# Repaso redes neurales

## Capas visibles: input, output

### Output:

- estas son las predicciones que queremos hacer
  - Para regresión, puede ser un solo valor
    - Ej Valor de la casa (California housing)
  - Para regresión multivariada, una neurona por valor
    - Ej predecir cuadro delimitador (alto, ancho, posición x, posición y)
- Una neurona por output



# Redes neurales: keras

Se crean en forma secuencial agregando capas:

```
keras.models.Sequential()
```

- **Input** – keras.layers.InputLayer()
- **Hidden** – keras.layers.Dense()
- **Output** – keras.layers.Dense()

Se compila el modelo final antes de entrenar:

```
model.compile()
```

Acá es donde se elige:

- Número de neuronas
- Tipos de activaciones
  - Ej ReLU, SoftMax
- Tipos de capas
  - Convolutional, Pooling

Acá es donde se elige:

- Optimizador
  - Ej SGD - Stochastic Gradient Descent
- Función de Pérdida
  - Ej Cross Entropy

Se entrena el modelo:

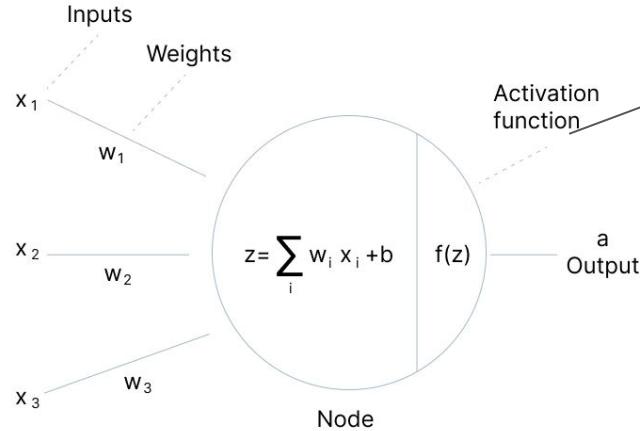
```
model.fit()
```

Acá es donde se elige:

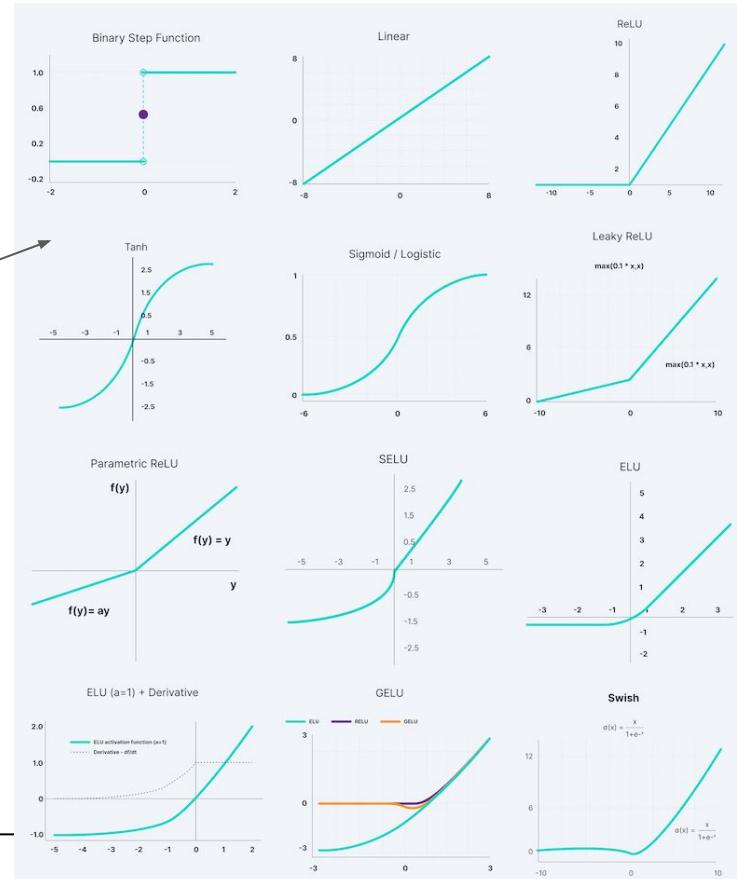
- Tamaño de batch
- Epocas

# Sobre activaciones

El propósito principal de las funciones de activación es el de **introducir no-linealidad** en el output



Activation Functions in Neural Networks [12 Types & Use Cases]  
<https://www.v7labs.com/blog/neural-networks-activation-functions>



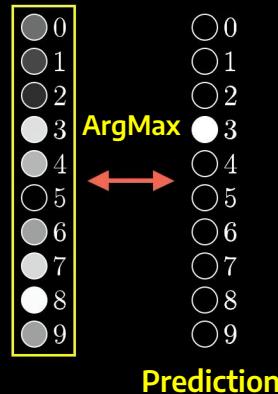
# Backpropagation

Es el algoritmo que se encarga de calcular el descenso del gradiente para disminuir la pérdida (loss).

Recorre la red hacia atrás, para averiguar cuales son los nodos (neuronas) responsables de las pérdidas y sus magnitudes.

Durante la propagación hacia atrás, se actualizan los **pesos y los sesgos** (**weights & biases**) de manera de minimizar la pérdida, dándole menor peso a los nodos que tienen mayor error.

What's the “cost” of this difference?



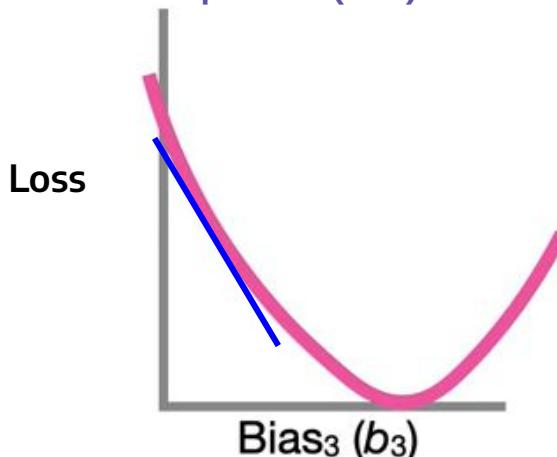
What is backpropagation? (3Blue1Brown)  
<https://www.3blue1brown.com/lessons/backpropagation>  
<https://youtu.be/Ilg3gGewQ5U>  
En español: <https://youtu.be/CyPjDPKtycM>

# Backpropagation

Como retocamos los pesos y bias?

Comparamos el vector target ( $y$ ) vs predicción ( $\hat{y}$ )

Buscamos minimizar alguna función de pérdida (loss)



What is backpropagation? (3Blue1Brown)  
<https://www.3blue1brown.com/lessons/backpropagation>  
<https://youtu.be/Ilg3gGewQ5U>  
En español: <https://youtu.be/CyPjDPKtycM>

```
from tensorflow import keras

from tensorflow.keras import layers

model = keras.Sequential()

model.add(layers.Dense(64, kernel_initializer='uniform', input_shape=(10,)))

model.add(layers.Activation('softmax'))

opt = keras.optimizers.Adam(learning_rate=0.01)

model.compile(loss='categorical_crossentropy', optimizer=opt)

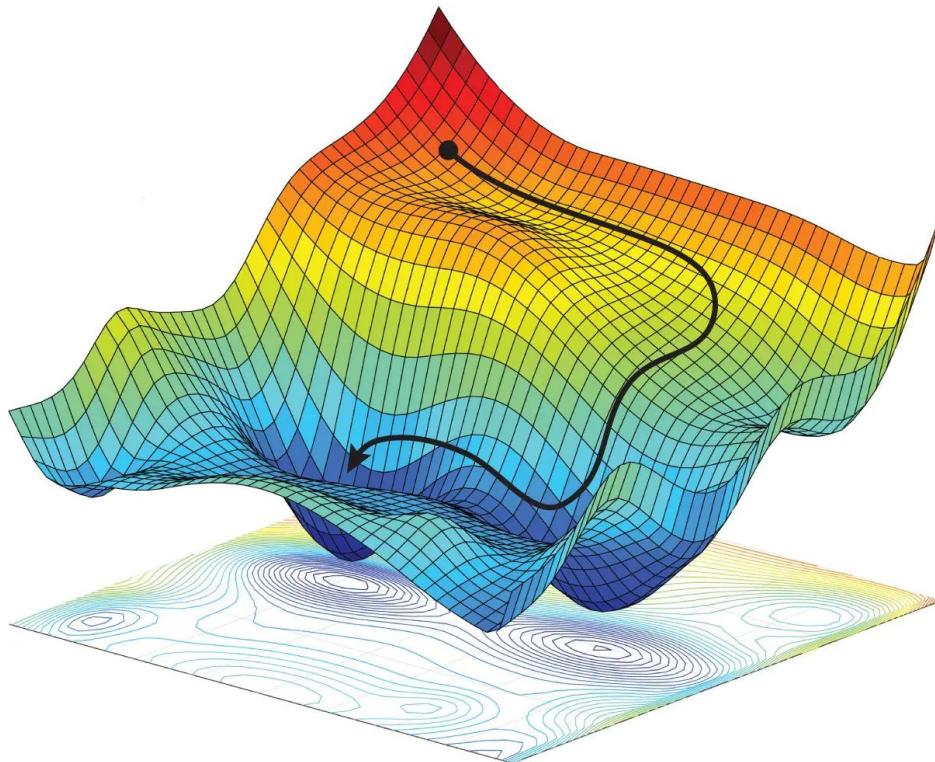
model.fit(X_train, t, epochs=250, batch_size=int(len(X_train)/4))
```

# Sobre optimizadores

Todos buscan lo mismo: en un espacio multidimensional, encontrar ***un mínimo*** (el de la función de pérdida).

Usan distintas estrategias. No los vimos en profundidad en el curso, son todas optimizaciones algorítmicas, que pueden interesarles estudiar:

Various Optimization Algorithms For Training Neural Network  
<https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>



# Sobre funciones de pérdida (loss functions)

- Una métrica de ajuste mide la pérdida (la diferencia) entre los valores reales (target) y los valores de output de la red (predicciones)
- Recordemos que usualmente trabajamos en dos tipos de tareas de aprendizaje: **REGRESION** vs **CLASIFICACION**
- En **regresión usamos, por ej**
  - Mean Squared Error (MSE), Mean Absolute Error (MAE), Cross-Entropy, Huber Loss
- En **clasificación usamos, por ej**
  - Binary Cross-Entropy, Categorical Cross-Entropy

# Pero ...

Pero en el fondo, todo problema de clasificación puede ser pensado como un problema de regresión.



There is no classification — here's why

<https://towardsdatascience.com/there-is-no-classification-heres-why-bdc8539bc898>

# MSE y MAE

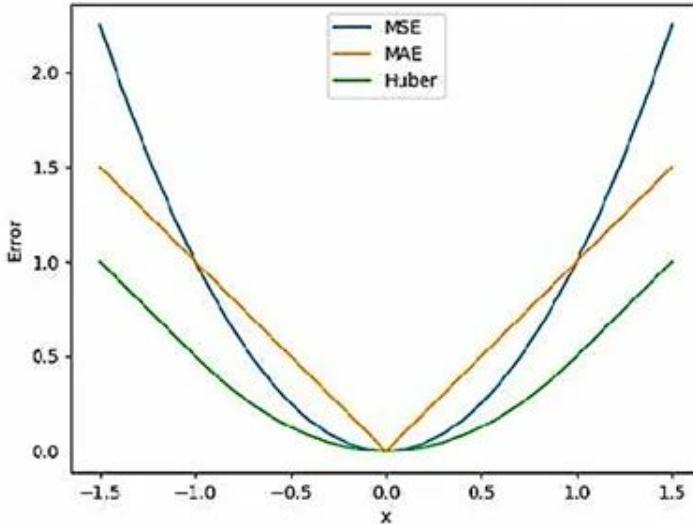
En este y otros módulos ya vimos MSE y MAE

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

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

**MSE:** las diferencias se elevan al cuadrado → los errores más grandes pagan! Se magnifican los errores (dicho de otra forma, es sensible a outliers!). Es una función convexa (se puede derivar) → ideal para Gradient Descent

**MAE:** da igual peso a errores, no es sensible a outliers pero la función no es derivable en el mínimo.

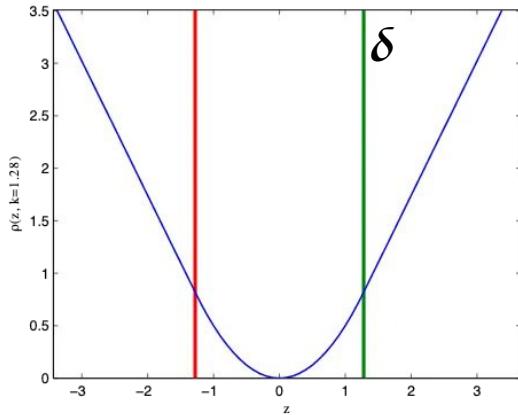


# Huber loss the best of both worlds

Y acá es donde entra Huber

$$\text{Huber Loss} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

$$\frac{1}{n} \sum_{i=1}^n \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta)$$



MSE

$$|y^{(i)} - \hat{y}^{(i)}| \leq \delta$$

$$|y^{(i)} - \hat{y}^{(i)}| > \delta$$

MAE

Por eso es que a mis amigos  
Los tengo muy escogidos,  
Son lo mejor de cada casa

*Joan Manuel Serrat  
– Las malas compañías*



# Huber loss the best of both worlds



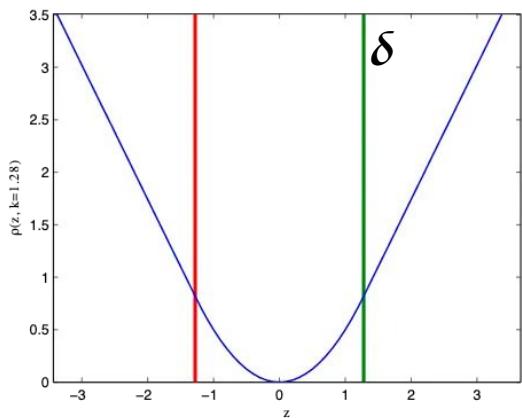
Y acá es donde entra Huber

$$\text{Huber Loss} = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

$|y^{(i)} - \hat{y}^{(i)}| \leq \delta$

$$\frac{1}{n} \sum_{i=1}^n \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta)$$

$|y^{(i)} - \hat{y}^{(i)}| > \delta$



Por eso es que a mis amigos  
Los tengo muy escogidos,  
Son lo mejor de cada casa

*Joan Manuel Serrat  
– Las malas compañías*

MSE

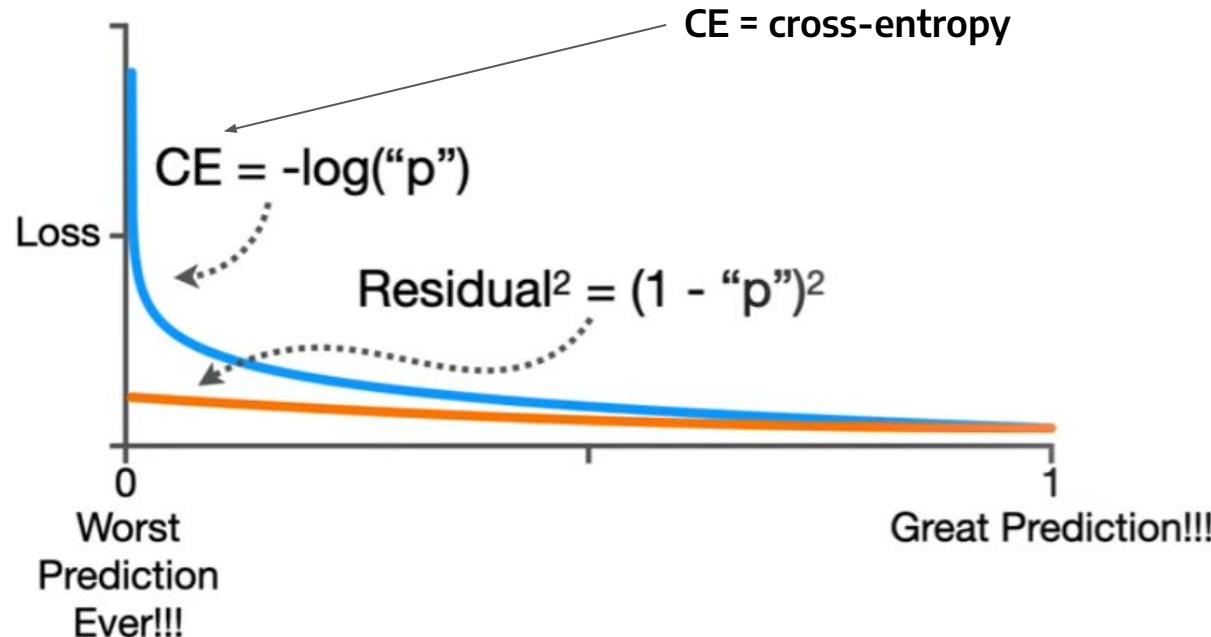
MAE



HUBER

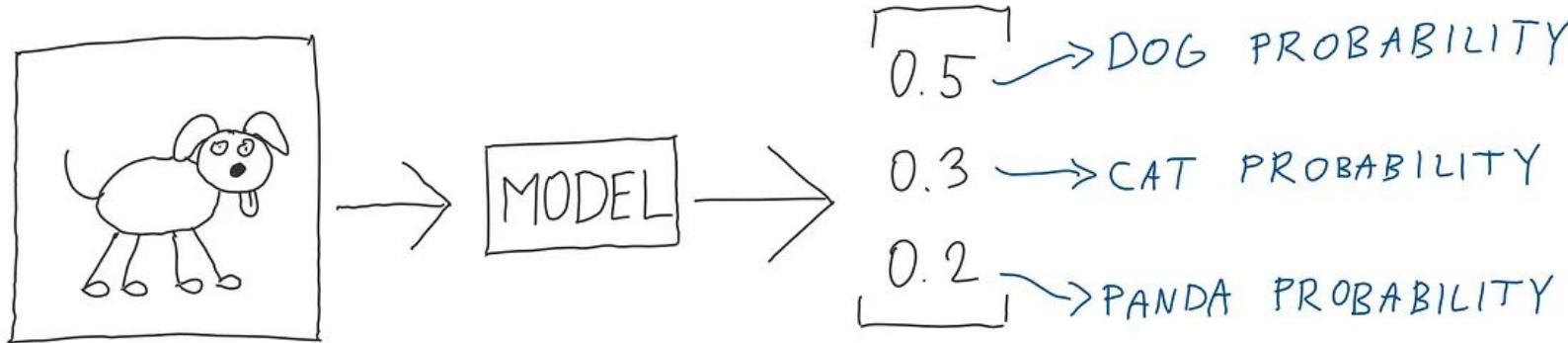
# Cross-entropy

Es una métrica para evaluar cuán bien se ajusta una red a los datos



<https://youtu.be/6ArSys5qHAU>  
StatQuest: Cross Entropy

# Cross-entropy



TARGET      PREDICTION

1	0.5
0	0.3
0	0.2

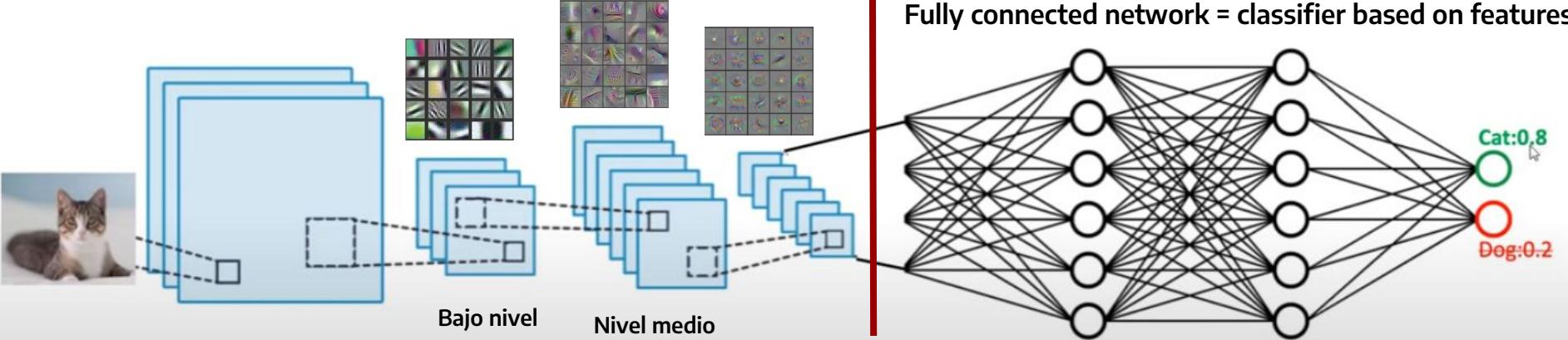
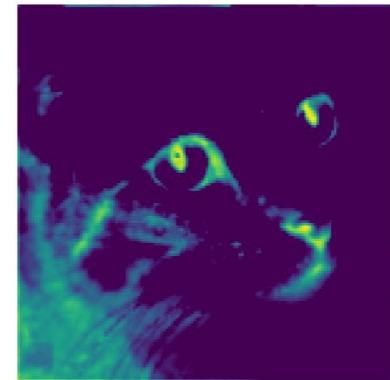
$$\text{Loss for class } X = - \underbrace{p(X)}_{\substack{\text{probability} \\ \text{of class } X \\ \text{in TARGET}}} \cdot \log \underbrace{q(X)}_{\substack{\text{probability} \\ \text{of class } X \\ \text{in PREDICTION}}}$$

# Tipos de arquitecturas de redes neuronales

## Convolutional Neural Networks

Las capas (ocultas) convolucionales lo que hacen es **detectar** y **extraer**, las features

Convolutions = feature detection + extraction



# Repaso redes neurales

Capas ocultas (*hidden layers*): muchos tipos de arquitecturas y técnicas

- **Capas convolucionales (Convolutional layers)**

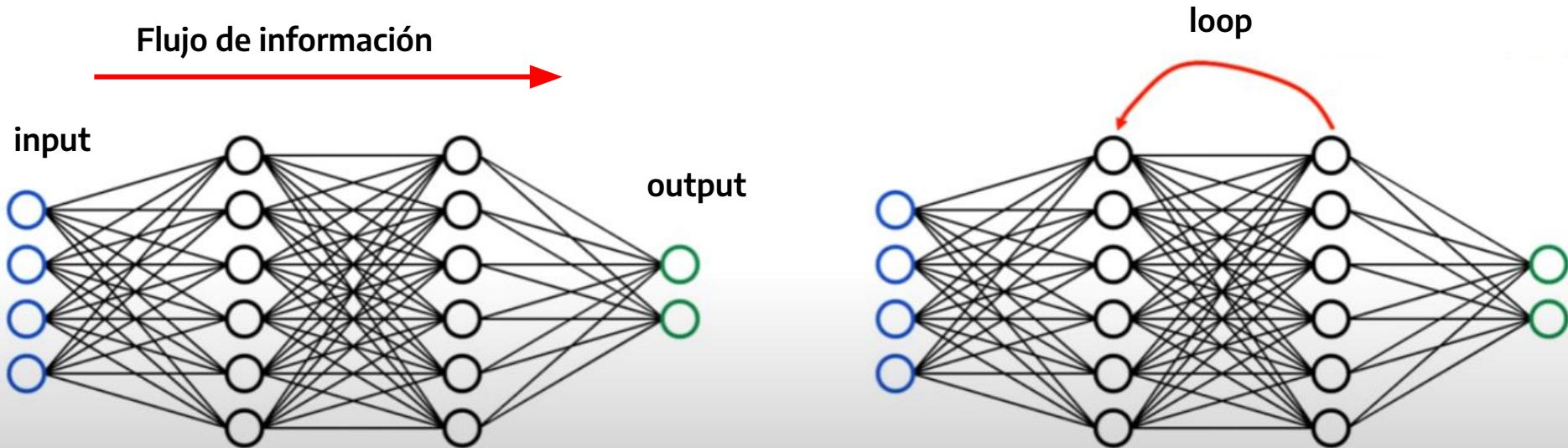
- Una **convolución** es la aplicación de un **filtro** sobre un **input** (ej una imagen)
- Ej multiplicar la señal de input \* un kernel (filtro) para obtener una señal modificada

- **Capas de pooling (agrupamiento)**

- Proceso de discretización / sub-muestreo / reducción de dimensiones
- El objetivo es submuestrear una representación de input

# Redes Neurales Recurrentes (RNNs)

Las redes neurales recurrentes (Recurrent Neural Networks, RNNs en inglés) tienen **loops**. Los loops sirven para *recordar* información que pasó por la red.



# Qué modelan las RNNs?

Domingo	Carnes rojas
Lunes	Pollo
Martes	Pasta
Miercoles	Pescado
Jueves	Carnes rojas
Viernes	Pollo
Sabado	Pasta
Domingo	Pescado
Lunes	Carnes rojas
Martes	Pollo
Miercoles	Pasta
Jueves	Pescado
Viernes	Carnes rojas

Que vamos a comer?

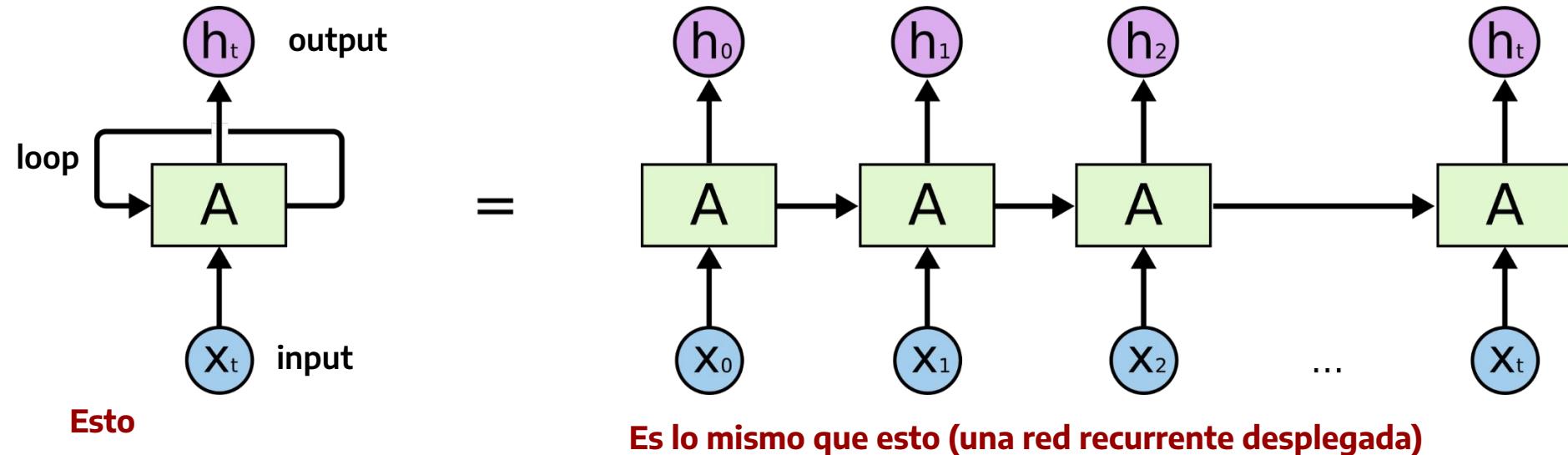
Normalmente modelamos la correlación entre **x** e **y**

Pero en este caso no hay correlación!!!

Para poder predecir lo que vamos a comer, necesitamos modelar la correlación de **y** con **y** (autocorrelation)

# Redes Neurales Recurrentes (RNNs)

Las redes neurales recurrentes (Recurrent Neural Networks, RNNs en inglés) tienen **loops**. Los loops pasan información de un paso al próximo.



# Aplicaciones de RNNs

Las RNNs se aplican en muchos problemas diversos. Pero en todos hay una secuencia de eventos (palabras, por ejemplo).

- *Reconocimiento de Voz (lenguaje hablado)*
- *Traducción*
- *Subtitulado de imágenes (video, o descripción de fotos)*
- *Clasificación de videos (secuencia de frames)*
- *Respuestas a preguntas visuales (input = imagen)*
  - *Ej que hay enfrente de la mesa? (Rta = una silla)*
  - *En donde está sentado el gato? (Rta= en el alfeizar de la ventana)*
- *Predictión de series temporales (forecasting, pronósticos)*
  - *Predecir valores futuros (precios de activos, temperaturas, lluvias, ventas, tránsito...) a partir de datos históricos colectados y registrados en períodos de tiempo regulares (horas, días, meses, años)*

# Long Short Term Memory Networks (LSTM)

Son un tipo especial de arquitectura de RNNs  
(recurrent neural networks).

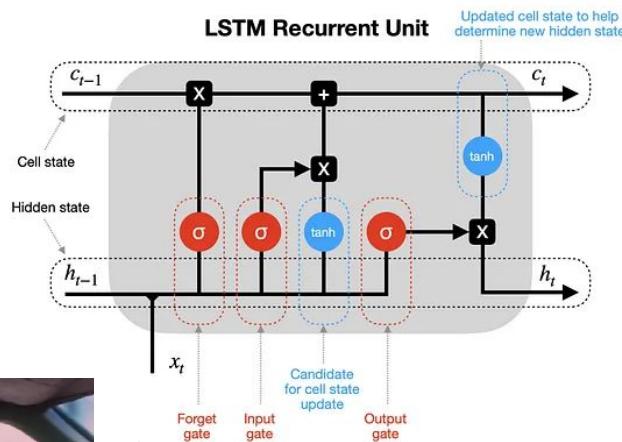
LSTMs usan una serie de compuertas (gates) que controlan cómo la información (que viene en una secuencia) entra, se almacena, y sale de la red.

Hay 3 tipos de compuertas:

- Forget (olvidar)
- Input
- Output



## LONG SHORT-TERM MEMORY NEURAL NETWORKS



# Redes neurales: modulos

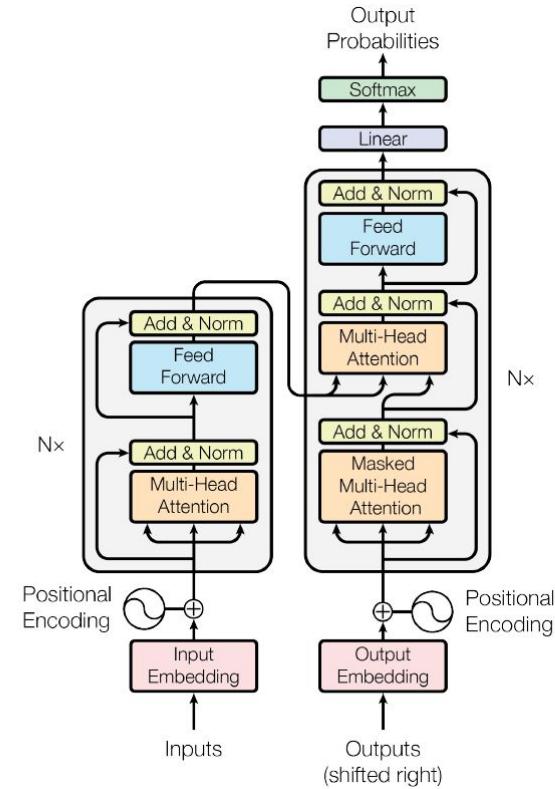
A lo largo del desarrollo y la investigación en Redes Neurales Artificiales, se van descubriendo y validando arquitecturas (maneras de conectar capas ocultas) para lograr distintos objetivos: compresión, recordar.

Esto módulos (conjuntos de capas con una arquitectura definida que funcionan

# Atención

*Attention is the most interesting recent architectural innovation in neural networks.*

```
tf.keras.layers.MultiHeadAttention(  
    num_heads,  
    key_dim,  
    value_dim=None,  
    dropout=0.0,  
    use_bias=True,  
    output_shape=None,  
    attention_axes=None,  
    kernel_initializer="glorot_uniform",  
    bias_initializer="zeros",  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```



# Redes modulares

Como ven ... podemos encapsular arquitecturas *tipo* en forma de **modulos reusables**

```
tf.keras.layers.MultiHeadAttention(  
    num_heads,  
    key_dim,  
    value_dim=None,  
    dropout=0.0,  
    use_bias=True,  
    output_shape=None,  
    attention_axes=None,  
    kernel_initializer="glorot_uniform",  
    bias_initializer="zeros",  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

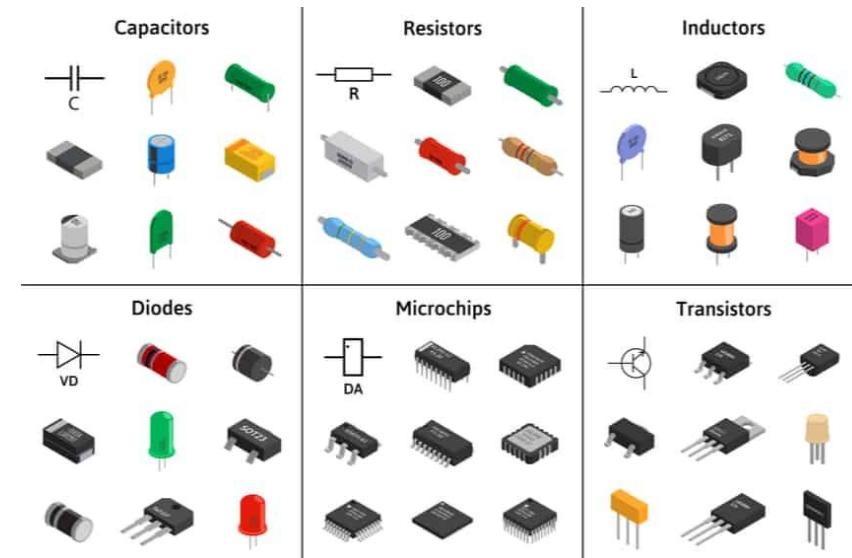
```
tf.keras.layers.LSTM(  
    units,  
    activation="tanh",  
    recurrent_activation="sigmoid",  
    use_bias=True,  
    kernel_initializer="glorot_uniform",  
    recurrent_initializer="orthogonal",  
    bias_initializer="zeros",  
    unit_forget_bias=True,  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    return_sequences=False,  
    return_state=False,  
    go_backwards=False,  
    stateful=False,  
    time_major=False,  
    unroll=False,  
    **kwargs  
)
```

# Circuitos, diseños, arquitecturas

Podemos entender como funciona cada uno de estos componentes?

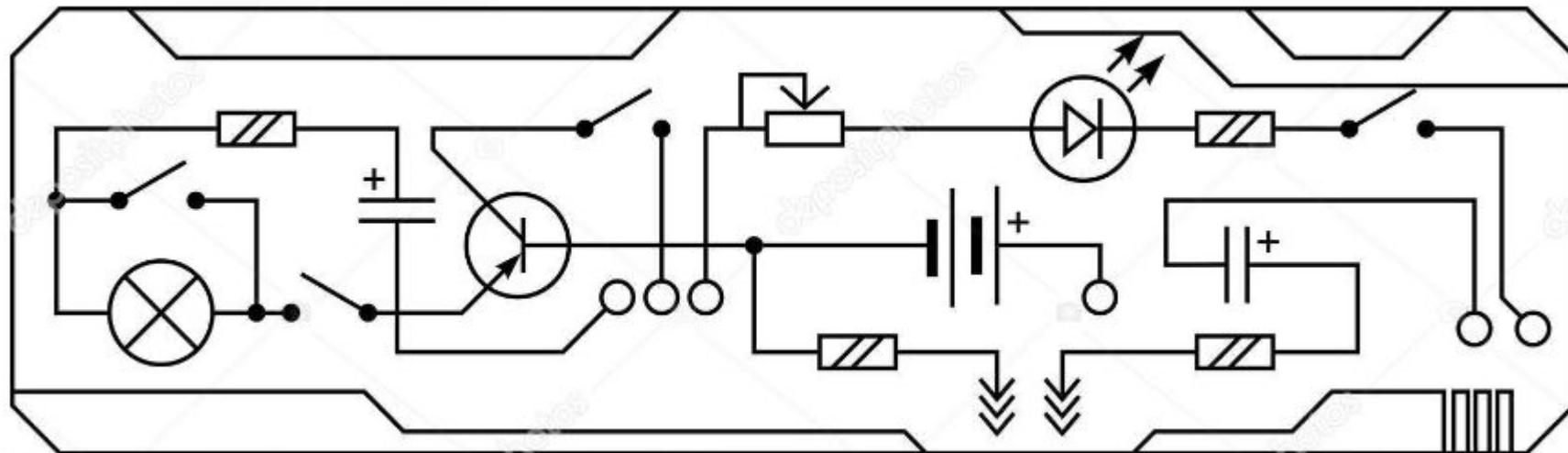
Circuit Symbols of Electronic Components

 Diode	 And gate	 Capacitor	 Inductor	 Resistor	 DC voltage source	 AC voltage source	
	Nand gate		Or gate		Xor gate		Inverter (Not gate)
 Coil	 LED	 Transistor	 Fuse	 Regulator	 Transformer		



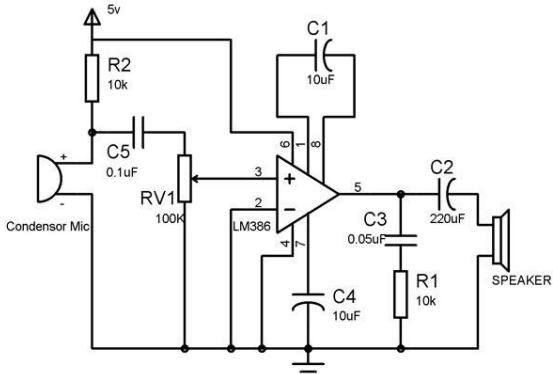
# Circuitos, diseños, arquitecturas

Podemos entender como funciona cada uno de los diseños o arquitecturas en las que estos componentes electricos y electronicos se conectan?

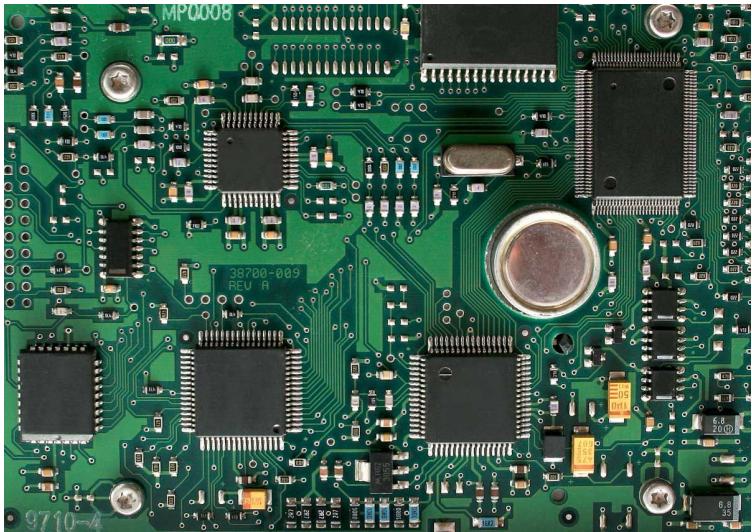
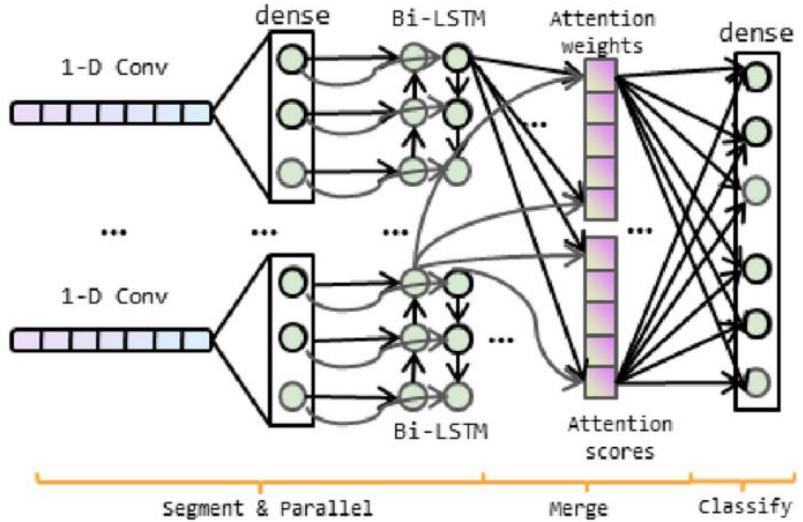


Circuito de una radio

# Audio Amplifier Circuit Using LM386



 CIRCUITS-DIY.COM  
SIMPLIFYING ELECTRONICS



# Las redes neurales como estructuras complejas

A PARTS LIST

## Inside the Brain's GPS

The neural navigation system of the human brain resides deep within a region known as the medial temporal lobe. Two areas of the medial temporal lobe—the entorhinal cortex and the hippocampus—act as key components of the brain's GPS. Networks of specialized cell types in the entorhinal cortex contribute to the complexity in the mammalian brain's pathfinding system.

Hippocampus (home of place cells)  
Entorhinal cortex (home of grid cells)

Cross section of the hippocampal region

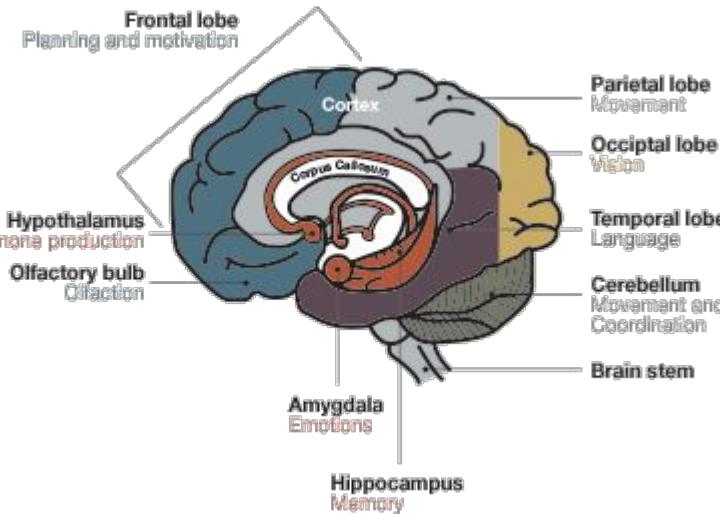
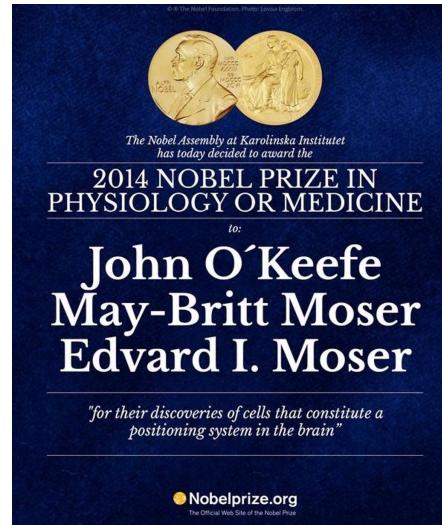
**Messaging the Hippocampus**  
The entorhinal cortex transmits information from grid cells about direction and distance traveled. It does so by sending signals along several pathways to subregions of the hippocampus (the dentate gyrus, CA3 and CA1) that produce a mental map optimized for planning future journeys (inset).

Small scale      Large scale

**A Close Look at Grid Cell Organization ...**  
... reveals that the spacing of the hexagonal elements that aid in creating a spatial map change when moving from top to bottom in the entorhinal cortex. The broader spacings correspond to larger distances the rat needs to travel to activate a vertex on the grid. At the top of the entorhinal cortex, a rat that activates a grid cell at one vertex of a hexagon will have to move 30 to 35 centimeters to an adjoining vertex. At the bottom, the animal needs to go as far as several meters.

**Other Specialized Cells Recently Discovered ...**  
... in the entorhinal cortex of rodents convey information to the hippocampus about the orientation of an individual's head, its speed of movement, and the distance to walls and other obstacles encountered. The output of these cells is combined to help create a composite map of the animal's environs.

Orientation  
Speed  
Border recognition



# Tipos de arquitecturas de redes neuronales

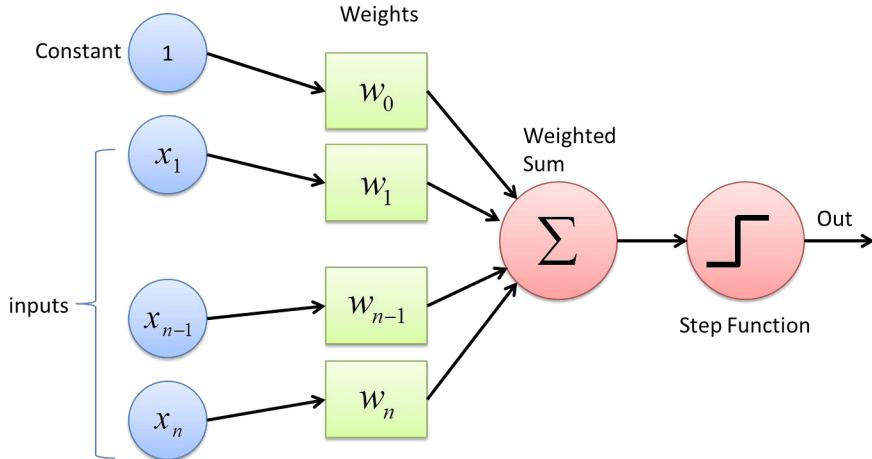
## Perceptrón

- Primera generación de redes neurales
- Modelos computacionales de única neurona

También llamadas ***feed forward networks***

La información fluye hacia adelante

La excepción es el ***backpropagation*** que durante el *entrenamiento* envía información hacia atrás (generalmente alguna forma de diferencia (loss) entre input y output.

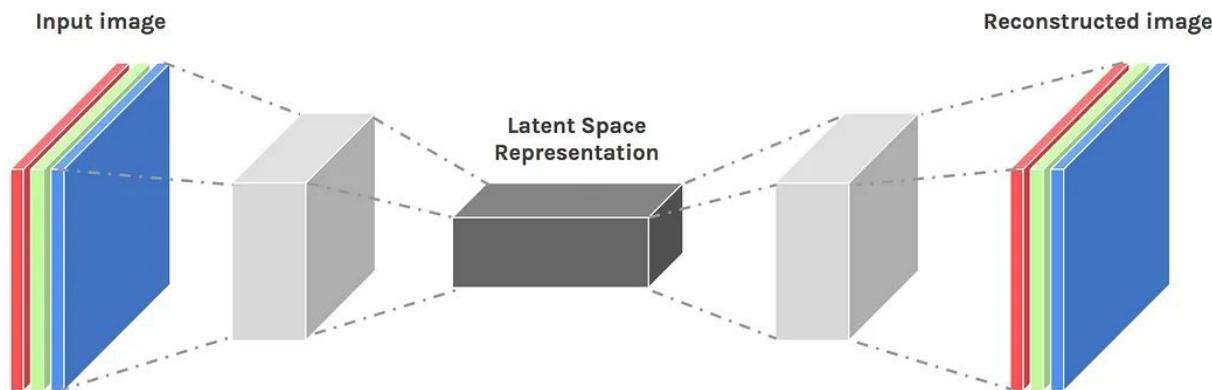


# Tipos de arquitecturas de redes neuronales

## Autoencoders

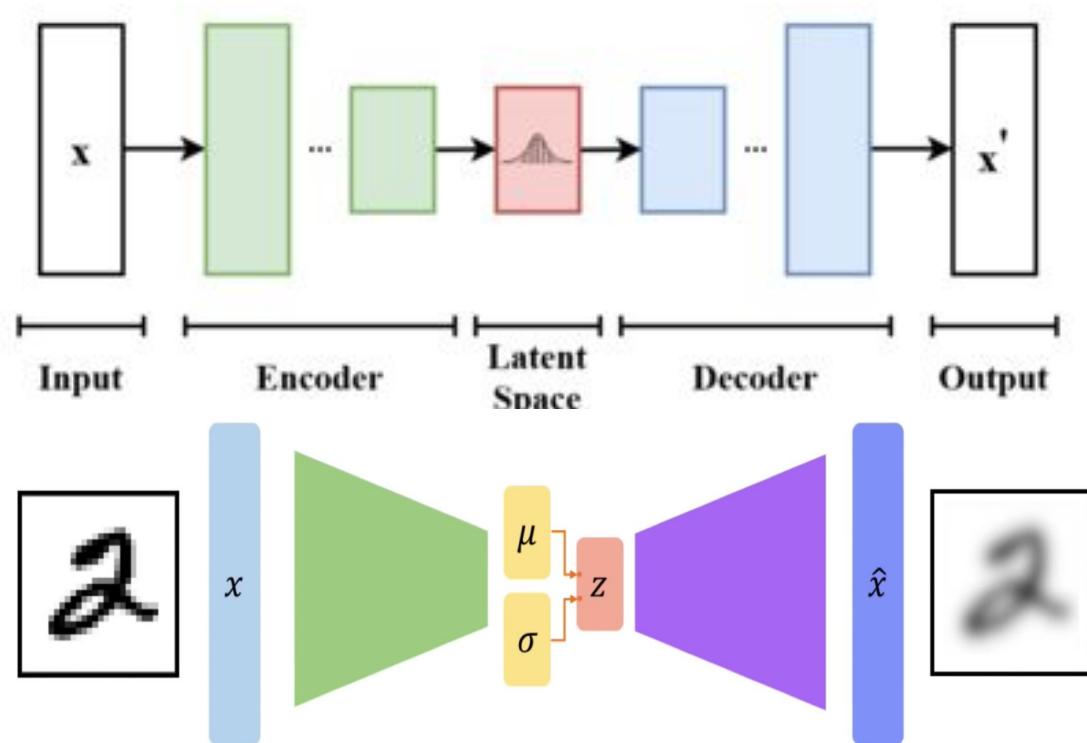
Diseñadas para aprender en forma *no supervisada*.

Como son modelos de *compresión* de datos, pueden codificar el input en una representación de menor dimensionalidad.



# Variational autoencoders

Son modelos probabilísticos generativos, aprenden los parámetros centrales que definen una función de densidad de probabilidades (PDF = probability density function)



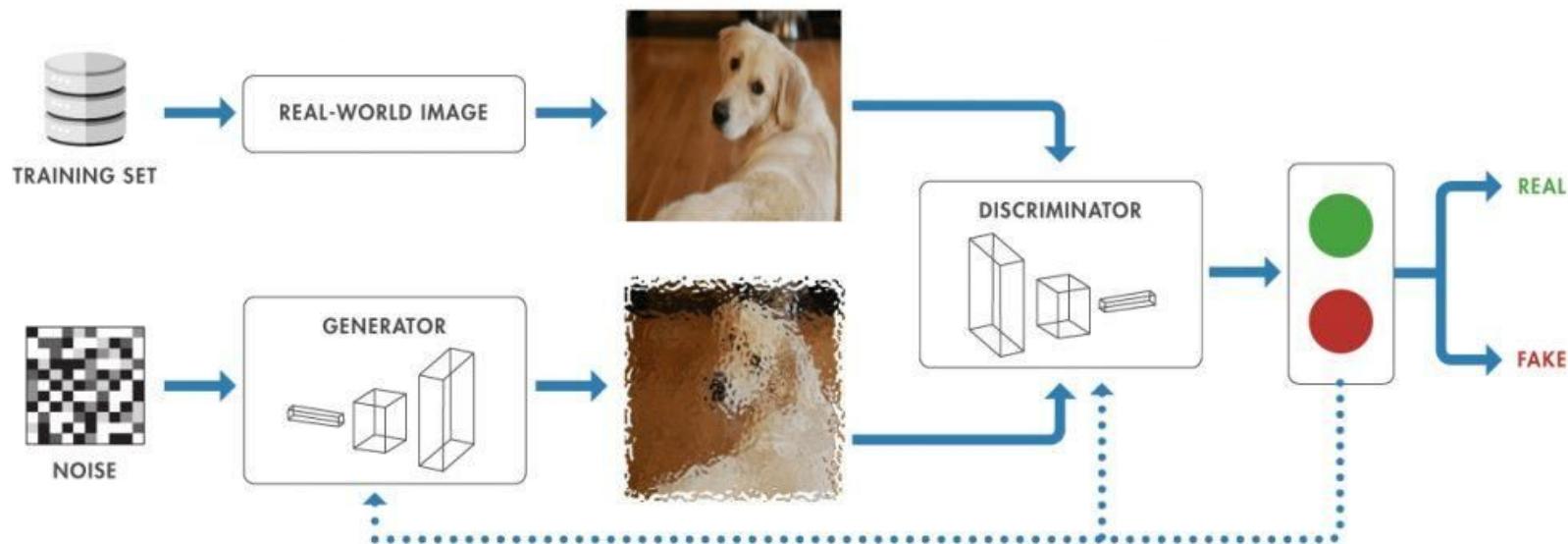
# Aplicaciones de Autoencoders

Los autoencoders se aplican por ejemplo en:

- *Reducción de ruido en el input (Image Denoising)*
- *Reducción de dimensiones*
- *Extracción de Features (foco en las más importantes)*
- *Compresión de datos*
- *Detección de anomalías*
- *Sustitución de datos faltantes (Missing Value Imputation)*

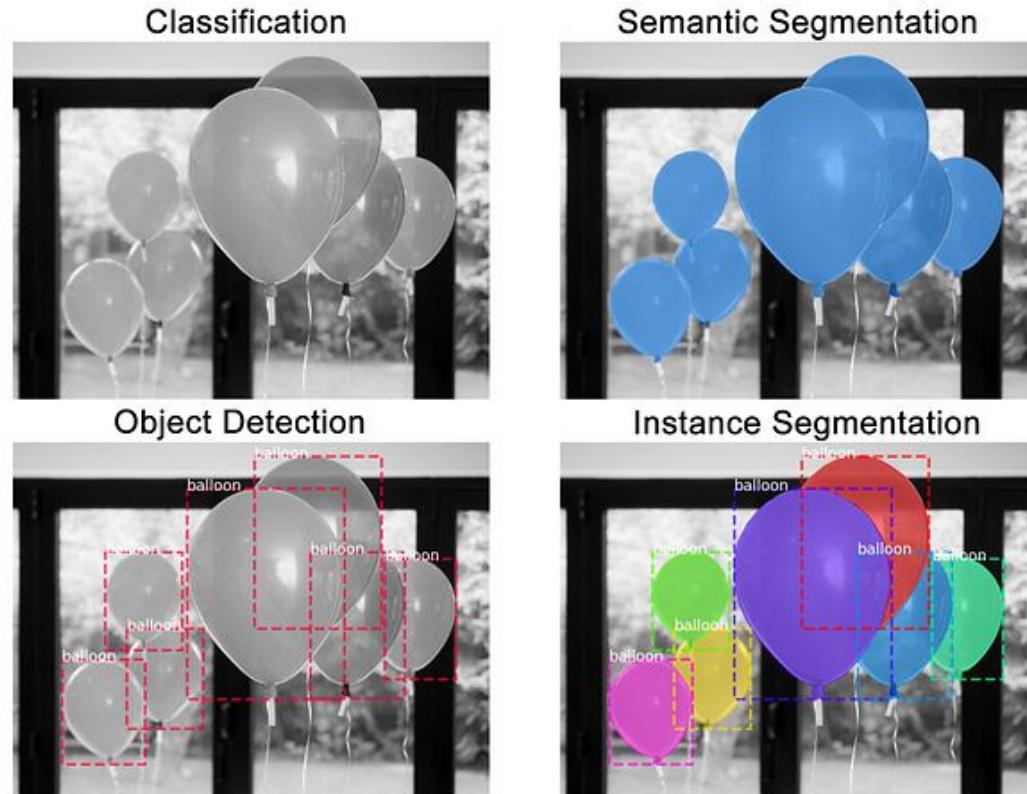
# Tipos de arquitecturas de redes neuronales

## Generative Adversarial Networks



# Region-based Convolutional Neural Networks (R-CNNs)

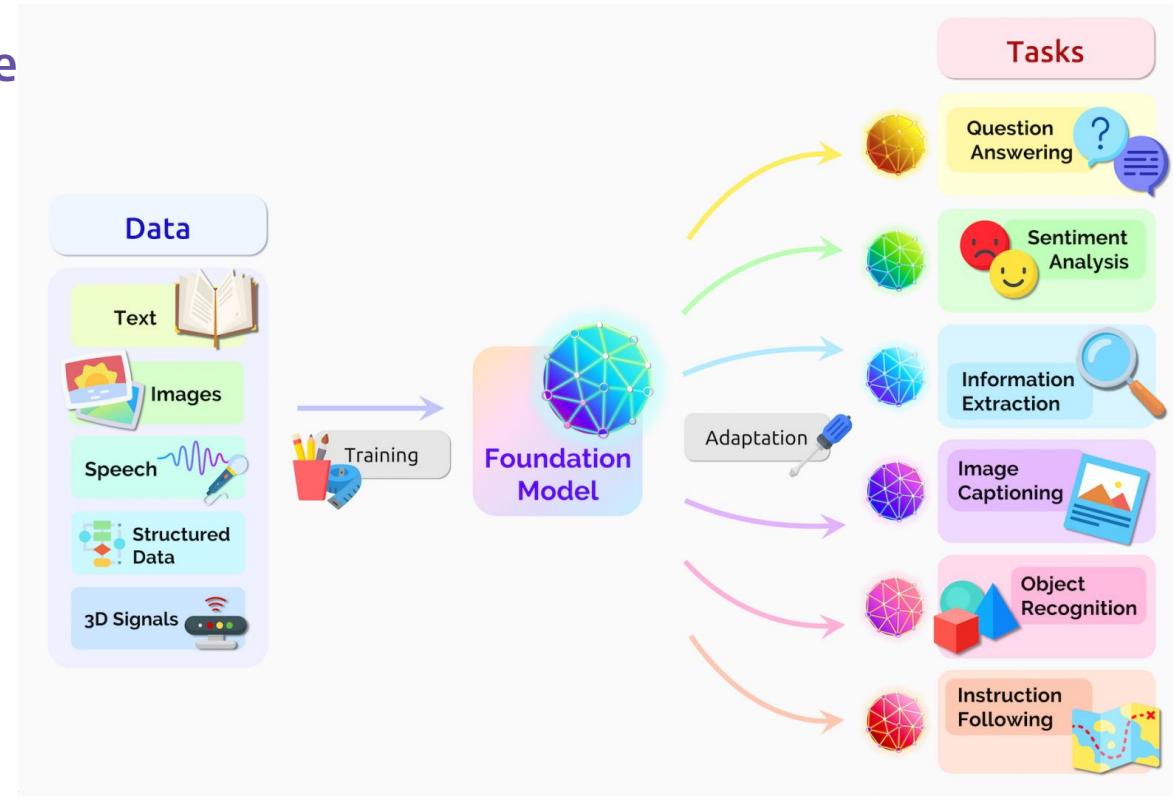
- Clasificación
  - Hay un globo en la imagen
- Segmentación semántica
  - Estos son los pixeles donde hay globos
- Detección de objetos
  - Hay 7 globos en la imagen en estas posiciones
- Segmentación de instancias
  - Hay 7 globos en estas posiciones y estos son los pixeles que corresponden a cada uno



# Foundational models

Son modelos entrenados que producen una variedad de outputs:

- Texto
- Imágenes
- Voz (audio)



# Transfer learning

Importamos el modelo **base**

```
include_top = False
```

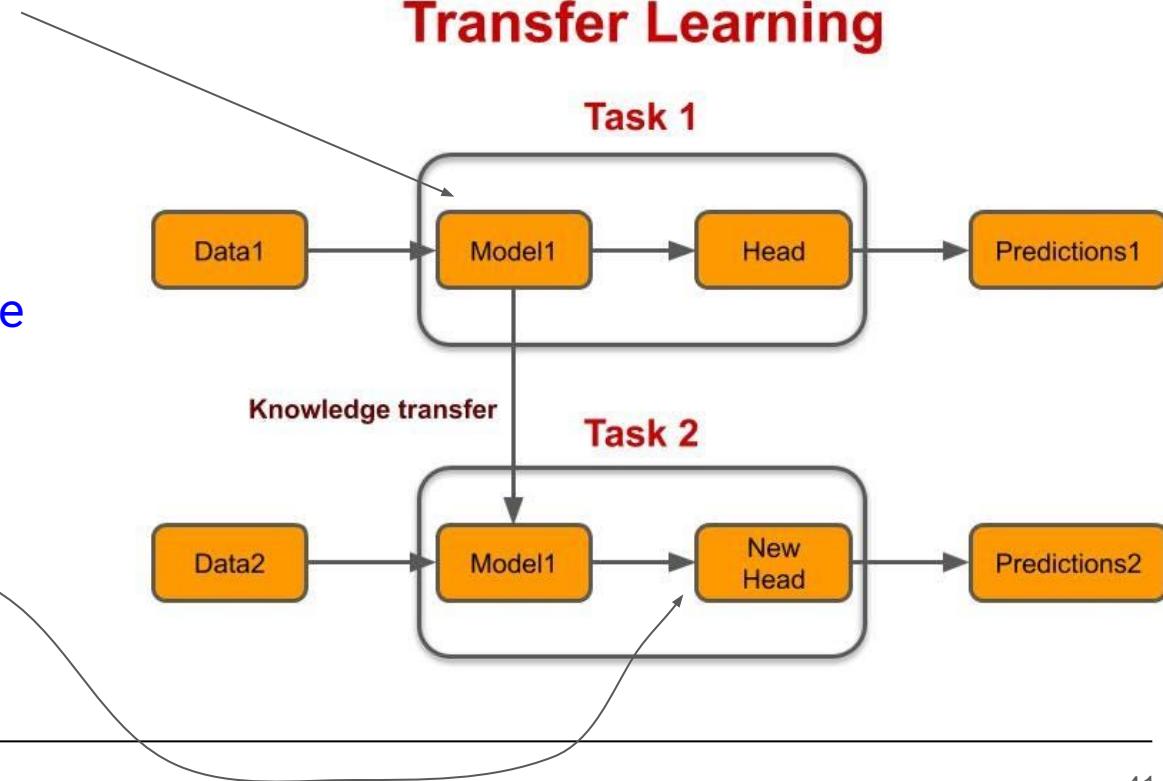
Freezamos el modelo:

```
model.trainable = False
```

Agregamos capas:

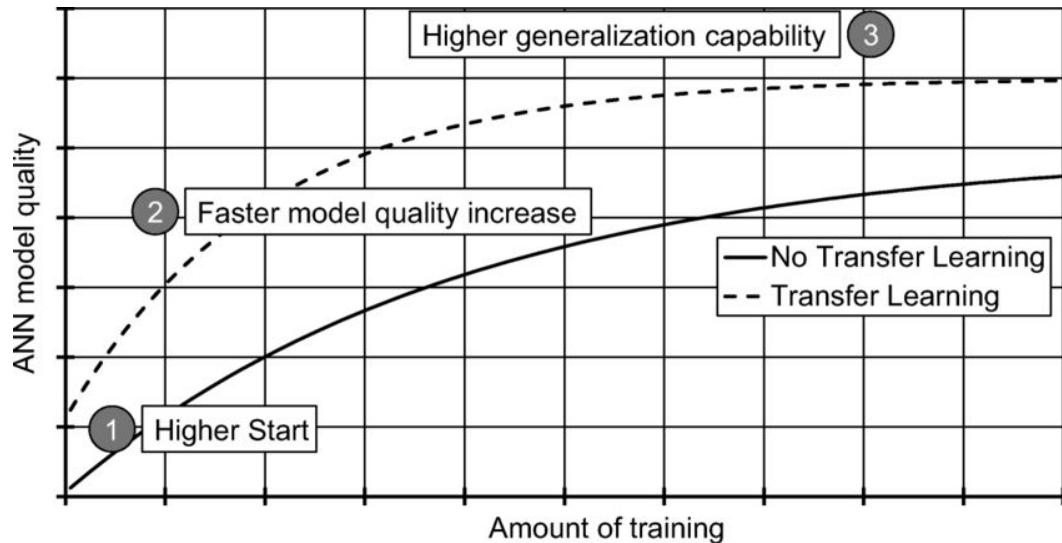
pooling , dropout, dense

## Transfer Learning



# Beneficios de transfer-learning

En el Notebook: ResNet50 model entrenado con >14 millones de imágenes, de 20 categorías



Transfer learning & fine-tuning

[https://keras.io/guides/transfer\\_learning/](https://keras.io/guides/transfer_learning/)

# Fine-tuning

Después de aplicar un proceso de *transfer-learning*  
(importar, modelo, freezar, agregar capas, entrenar, etc.)  
podemos hacer *sintonía fina* y ajustar los **pesos y los sesgos (weights & biases)** del modelo completo.

Unfreeze | Descongelar (todo o parte)! ← esa es la cuestión

Transfer learning & fine-tuning  
[https://keras.io/guides/transfer\\_learning/](https://keras.io/guides/transfer_learning/)

UNFREEZE ALL  
OR PART OF THE MODEL



imgflip.com

# Parametros, parámetros, parámetros

Batch size?

Dropout?

Learning rate?

Epochs?

# Batch size

El **batch size** define el número de muestras que van a pasar por la red **antes** de actualizar los parámetros.

Un batch (bloque) de muestras completa un proceso de *forward + backward propagation*.

Ejemplo:

Total de imágenes de entrenamiento = 3000

Batch size = 64

Epochs = 500

Entonces:

- 64 imágenes se toman cada vez para entrenar la red
- Para procesar las 3000 imágenes necesitamos 47 iteraciones ( $3000/64 \rightarrow 1$  epoch)
- Este proceso continua 500 veces (epochs)

```
model.fit(X_train,  
          y_train,  
          batch_size = 64,  
          epochs = 500,...)
```

# Tamaño del batch size?

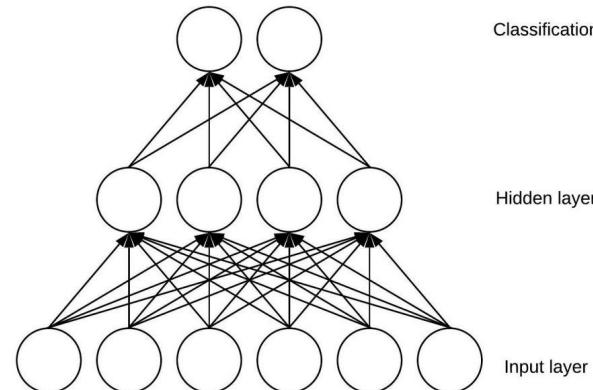
- La limitación principal está dada por el hardware (RAM, GPU)
  - Y por el tamaño de las imágenes! No es lo mismo imágenes de 512 x 512 x 3 (RGB) que imágenes de 28 x 28 x 1 (MNIST)
- Batch size pequeño puede hacer al descenso del gradiente menos preciso, pero el algoritmo puede ser más rápido en *converger*.
- Batch size muy grande puede haber degradación del modelo, sobre todo en su capacidad de *generalización*. Pero son más lentos en

# Dropout

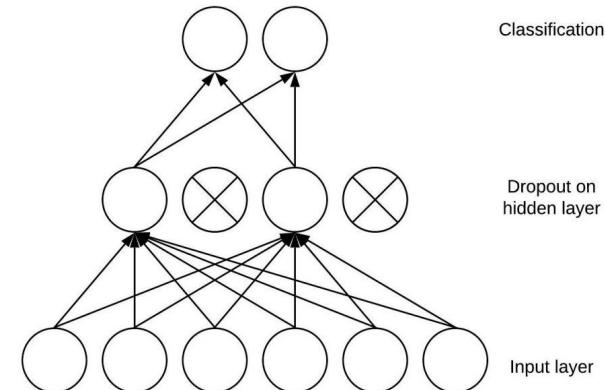
**Dropout** es el proceso por el cual se desconectan algunas neuronas aleatoriamente, durante el proceso de **entrenamiento**.

Es un método de **regularización** para reducir **overfitting**.

*Es una aproximación al proceso de entrenar un gran número de redes con arquitecturas distintas.*



Without Dropout



With Dropout

# Argmin / Argmax

Qué son Argmax y Argmin?



- Argmax es una operación que encuentra *el argumento* que da el valor máximo de una función objetivo (target)
- Suele usarse `argmax()` de NumPy.
- En machine learning se usa para encontrar las mejores predicciones
  - `yhat_probs = model.predict(X_validation)`
  - `yhat_classes = np.argmax(yhat_probs, axis=1)`
- ArgMin es lo mismo (pero al revés)



# ArgMax vs SoftMax

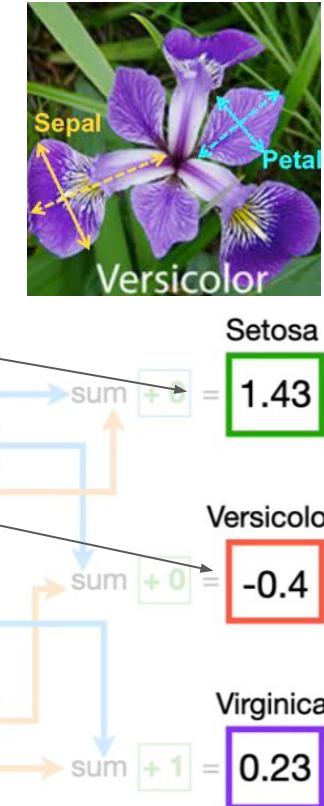
Red para clasificar flores (Iris dataset)

Raw output > 1

Raw output < 0



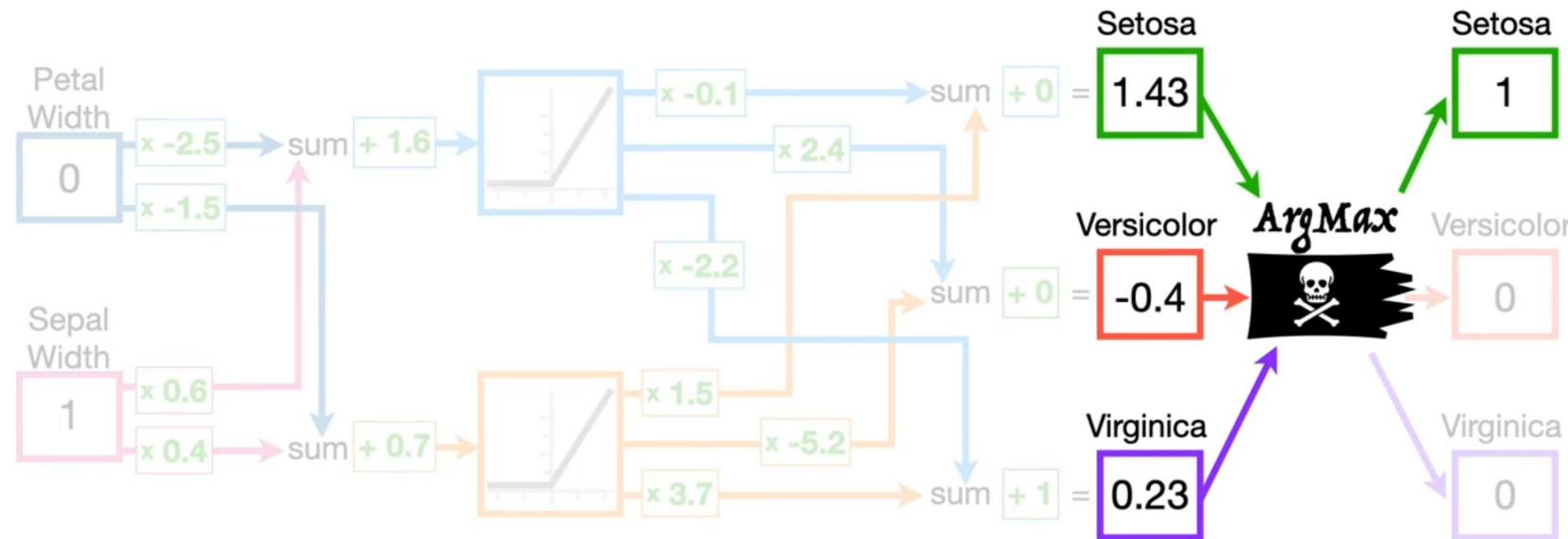
<https://youtu.be/KpKog-L9veg>  
StatQuest: ArgMax and SoftMax



# ArgMax vs SoftMax

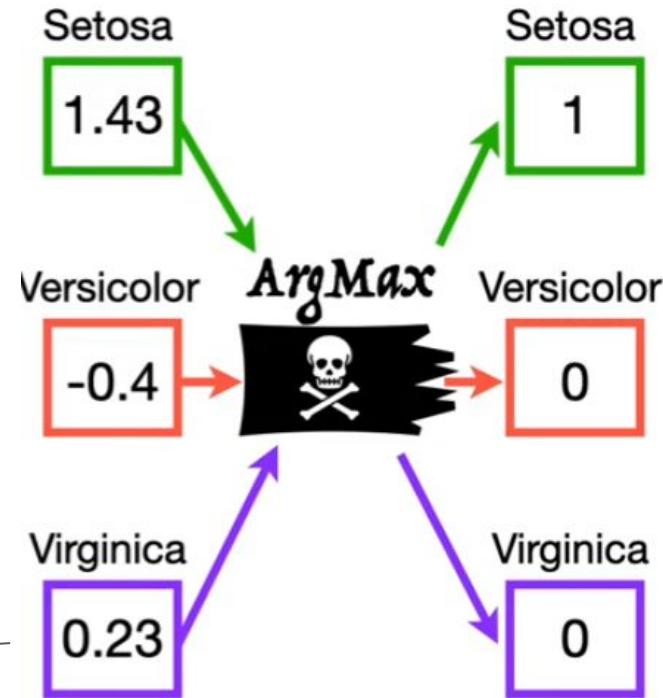
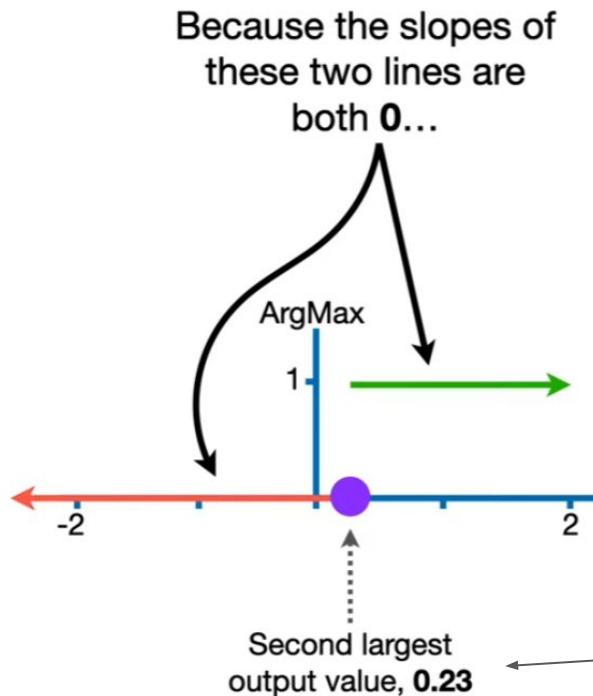
ArgMax – facilita *interpretación* del output de una red (**predicciones!**)

Pero no se puede usar para backpropagation! (optimizar weights and biases!)



# Las derivadas del output de ArgMax son cero

No se pueden usar en Gradient Descent!



# ArgMax vs SoftMax

## SoftMax

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

$\sigma$  = softmax

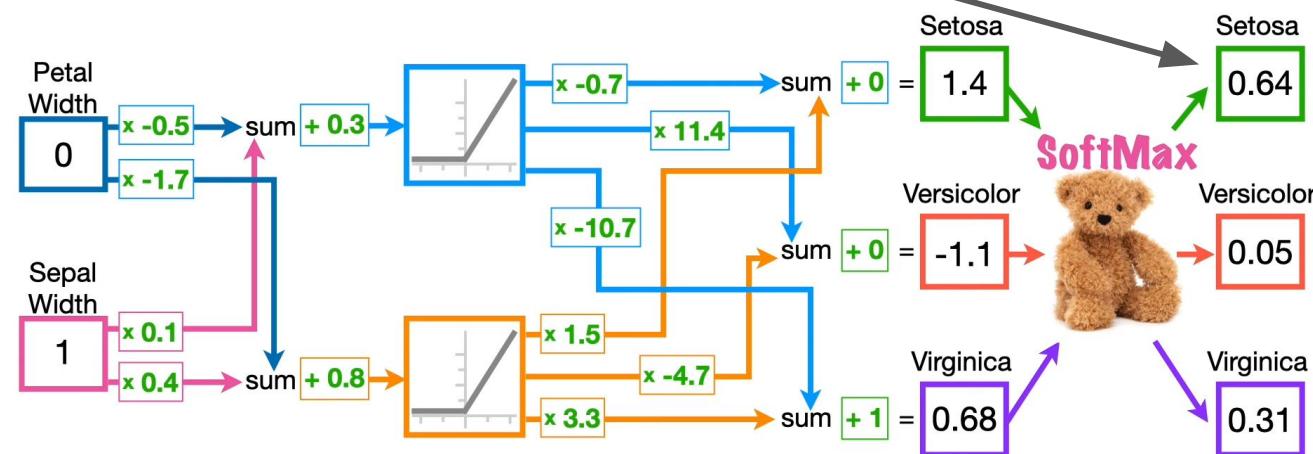
$\vec{z}$  = input vector

$e^{z_i}$  = standard exponential function for input vector

$K$  = number of classes in the multi-class classifier

$e^{z_j}$  = standard exponential function for output vector

$$\text{SoftMax}_{\text{Setosa}}(\text{Output Values}) = \frac{e^{\text{Setosa}}}{e^{\text{Setosa}} + e^{\text{Versicolor}} + e^{\text{Virginica}}}$$



# ArgMax vs SoftMax

- ArgMax y SoftMax devuelven valores en un rango entre 0 y 1
- ArgMax devuelve un único valor como 1 (el máximo) y el resto como 0
- SoftMax devuelve valores intermedios para todos los datos

**Argmax**

#1
#2
#3
#4
#5

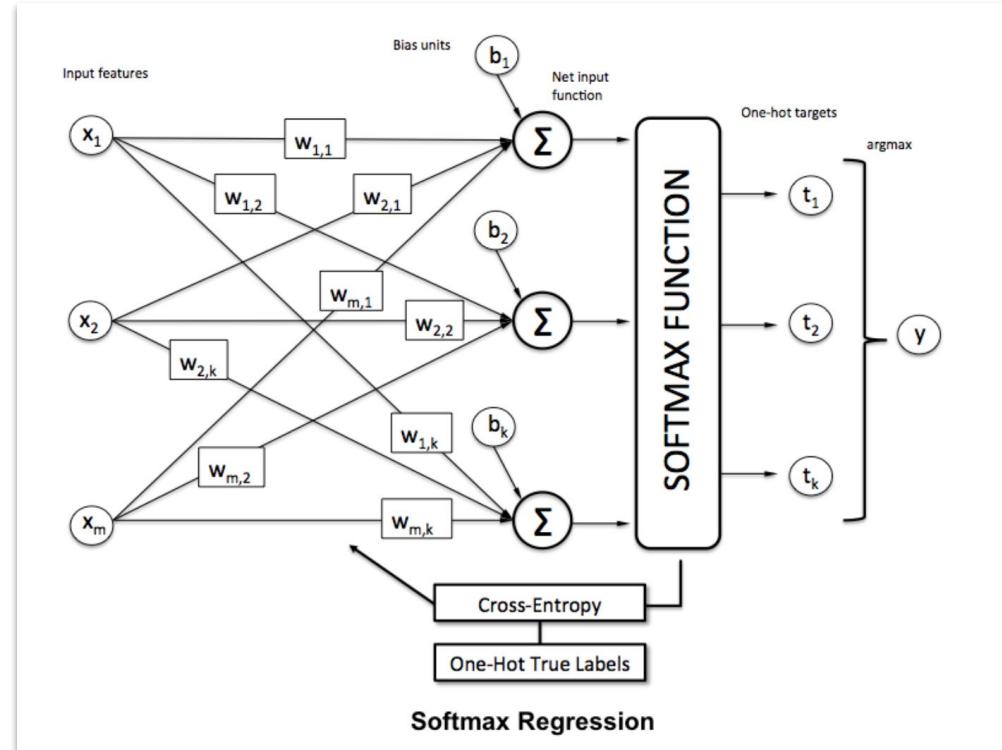
**Softmax**

#1
#2
#3
#4
#5

# ArgMax vs SoftMax

## Recapitulando

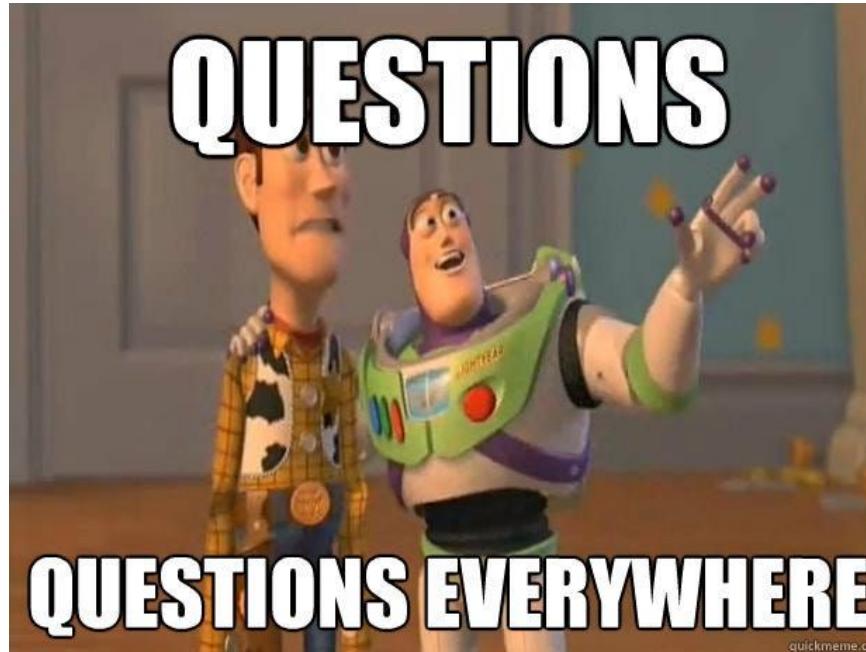
- ArgMax nos ayuda a **decidir**
- Encontrar cuál es **la mejor predicción entre varias**
  - No sirve en capas intermedias, los valores de output de ArgMax no pueden usarse para optimizar pesos y bias en backpropagation!
- SoftMax nos ayuda a **convertir** los valores de output de la red en valores uniformes
- **Parecidos** a probabilidades
- Que pueden usarse en backpropagation!



# Y ahora el meme tiene sentido



# Preguntas?



# Buena suerte en el examen!



imgflip.com