Programa de formación

MACHINE LEARNING AND DATA SCIENCE MLDS

Facultad de INGENIERÍA

UNIVERSIDAD NACIONAL DE COLOMBIA

Módulo 5

Deep learning: Introducción al Aprendizaje Profundo con Python

Unidad 2

Redes neuronales en *Keras*

Fabio Augusto González Osorio, PhD

Facultad de INGENIERÍA

UNIVERSIDAD NACIONAL DE COLOMBIA

Bienvenida

Fabio Augusto González Osorio, PhD

https://dis.unal.edu.co/~fgonza/

fagonzalezo@unal.edu.co

Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia
Sede Bogotá

UNIVERSIDAD NACIONAL DE COLOMBIA

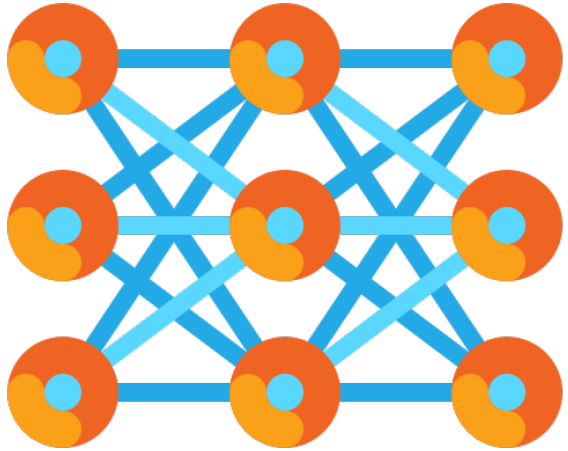
Agenda

- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

Agenda

- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

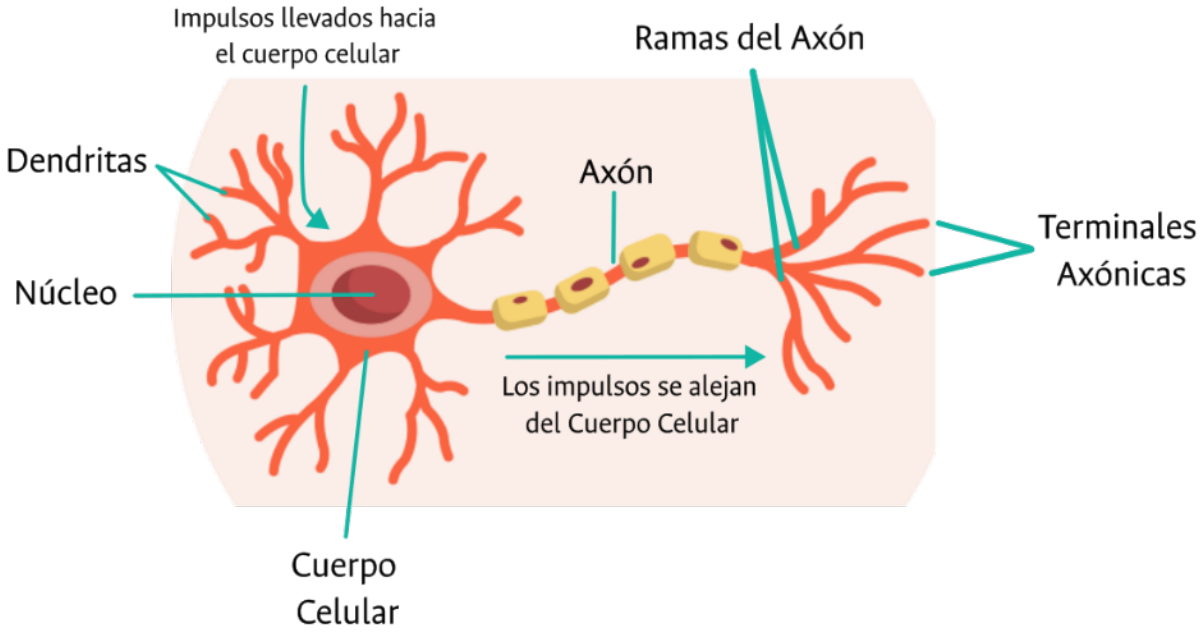
1 Introducción a redes neuronales en *Keras*



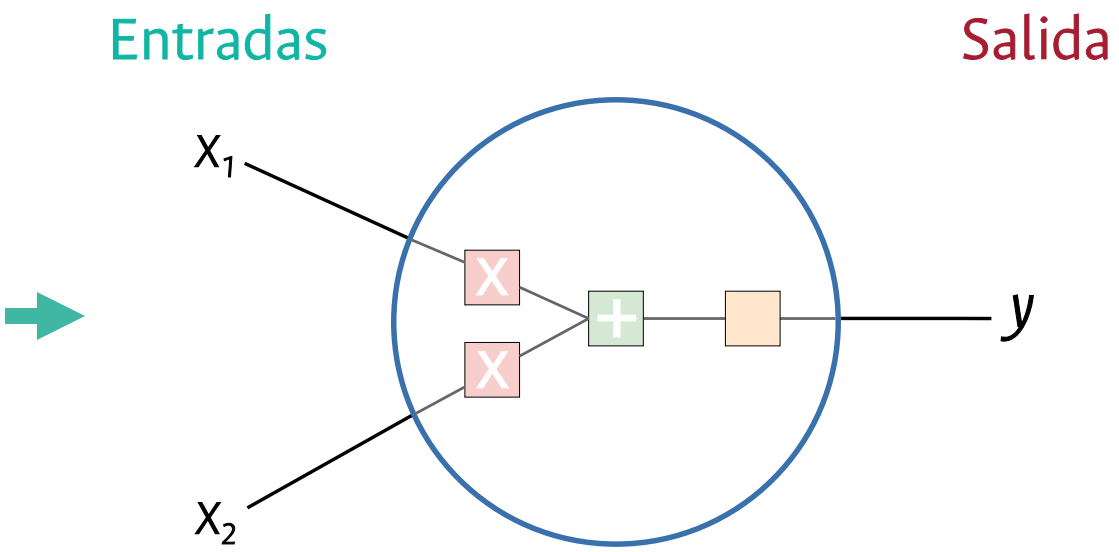
Introducción a redes neuronales en Keras

Unidad Fundamental : La Neurona

Neurona Biológica



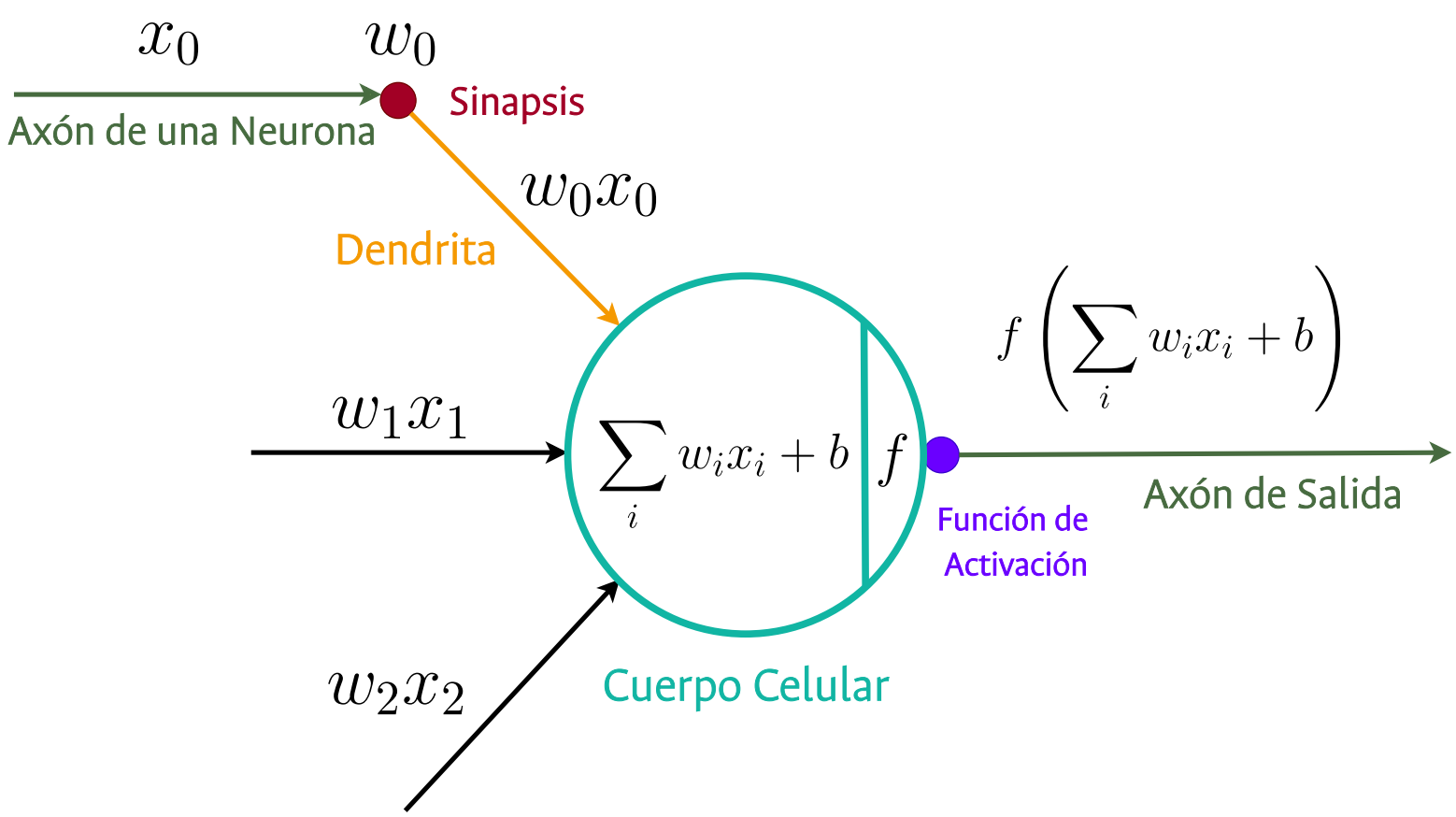
Neurona Artificial



¿Cómo funciona?

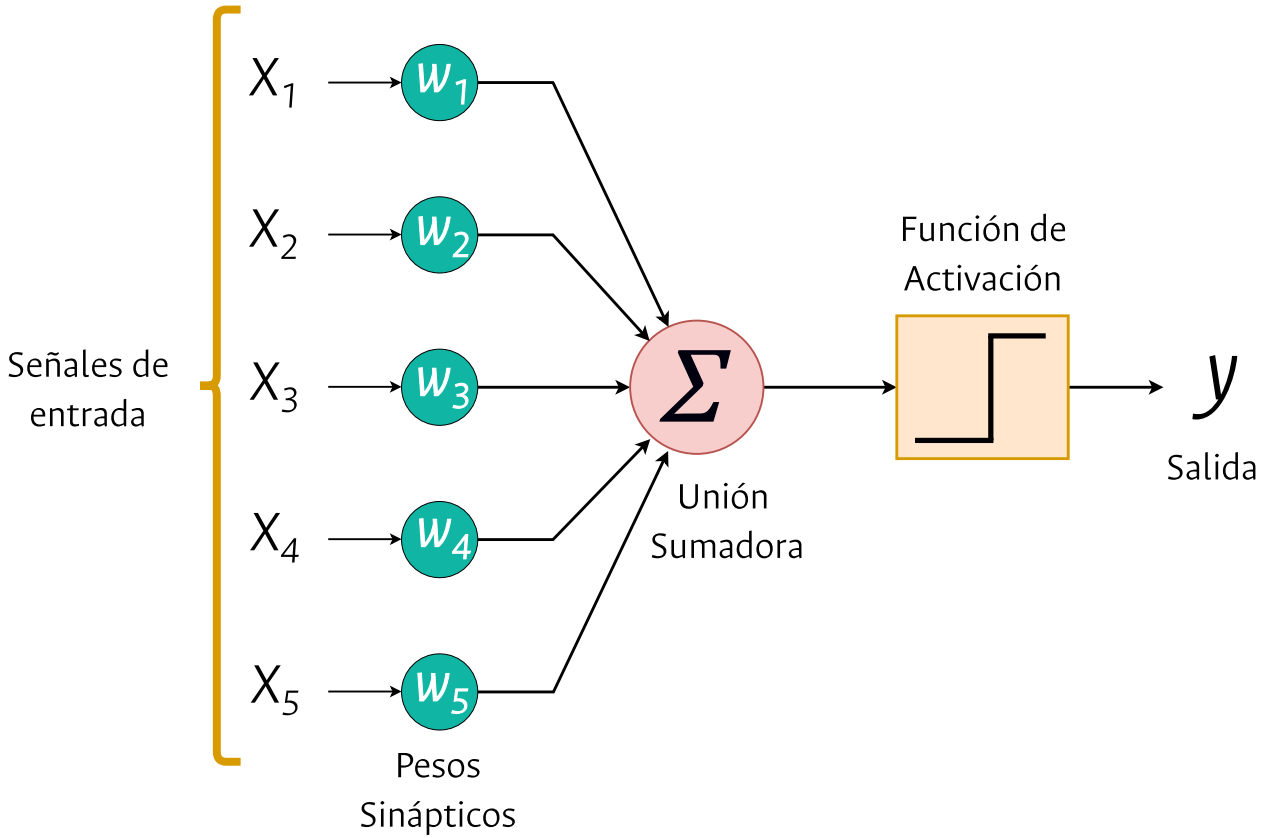
Funcionamiento

- Cada entrada se multiplica por un peso
- Todas las entradas ponderadas se suman, incluyendo un sesgo (bias).
- Finalmente, toda la sumatoria pasa a través de una función de activación



Introducción a redes neuronales en Keras

Perceptrón de Rosenblat - La red neuronal más sencilla

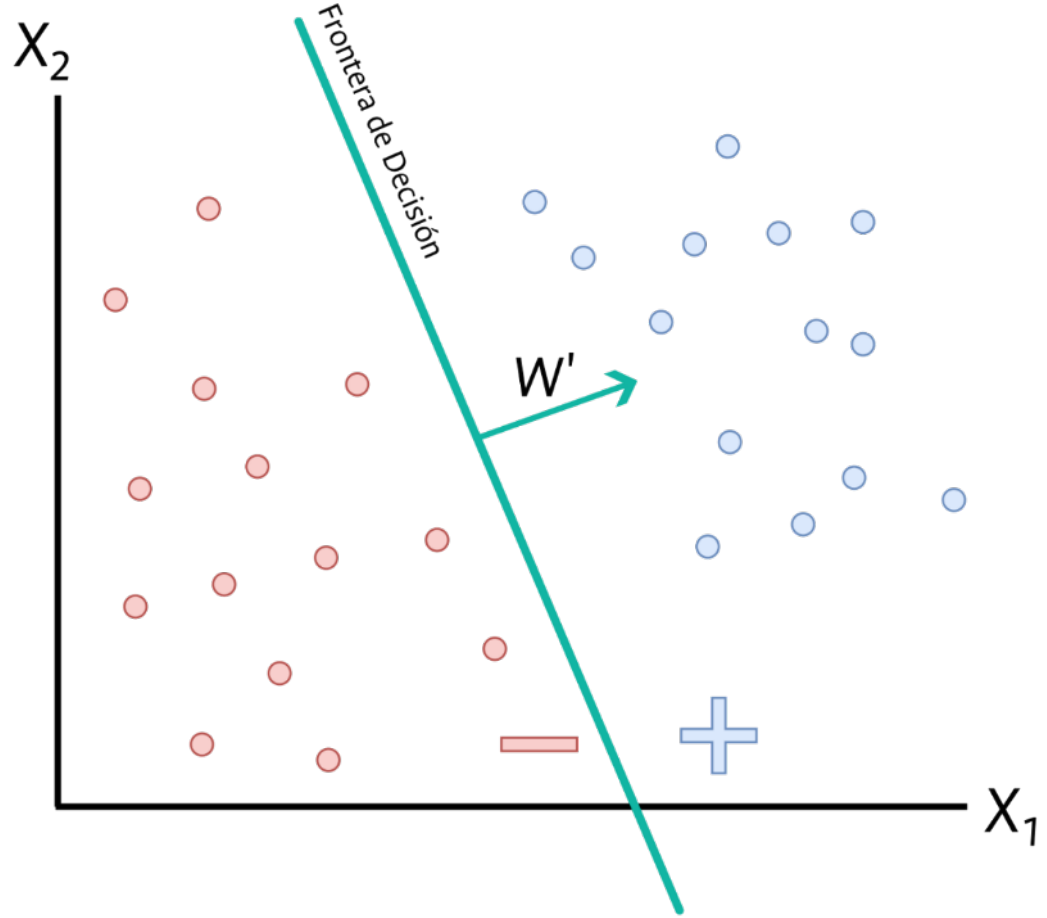


Perceptrón de Rosenblat

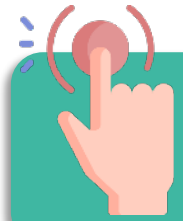
- Las entradas pasan por una sola neurona
- Se usa una función de activación *Heaviside*.

Perceptrón de Rosenblat - La red neuronal más sencilla

Para un problema de clasificación, el perceptrón de Rosenblat encuentra una **frontera lineal**.



Funciones de Activación



Función de Activación

- La función de activación permite incluir una **no-linealidad**.
- La función de activación sirve como un mecanismo de control.
  - Acota la salida de cada neurona y hace que se comporte de manera más predecible.

El resultado de la sumatoria de las entradas por los pesos, es una operación lineal.

$$\sum_i w_i x_i + b$$

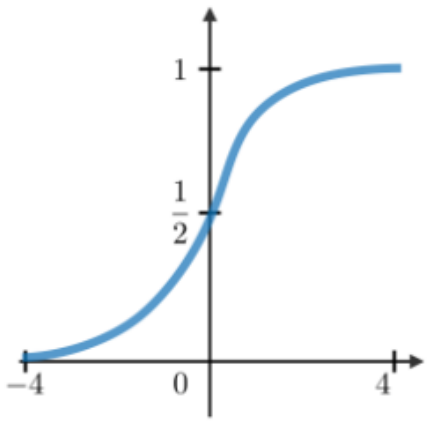
Introducción a redes neuronales en Keras

Funciones de Activación

Algunas de las funciones de activación más comunes son:

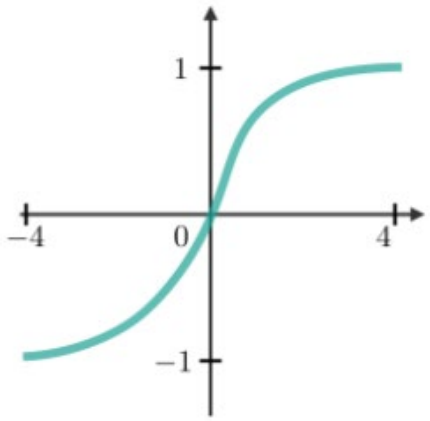
Sigmoid

$$g(z) = \frac{1}{1 + e^{-z}}$$



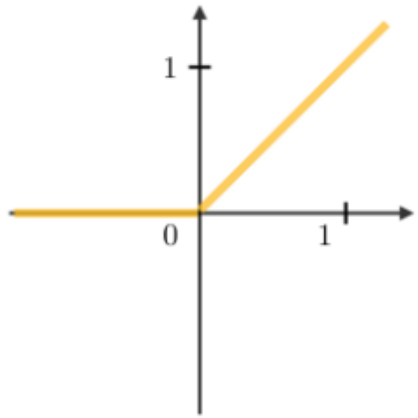
Tanh

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



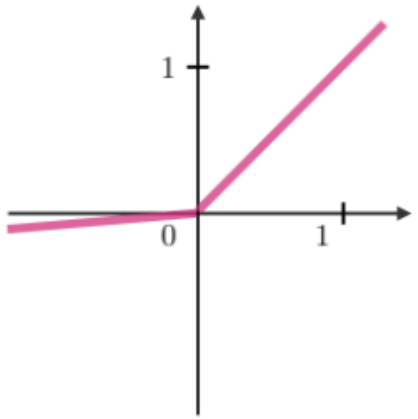
ReLU

$$g(z) = \max(0, z)$$



Leaky ReLU

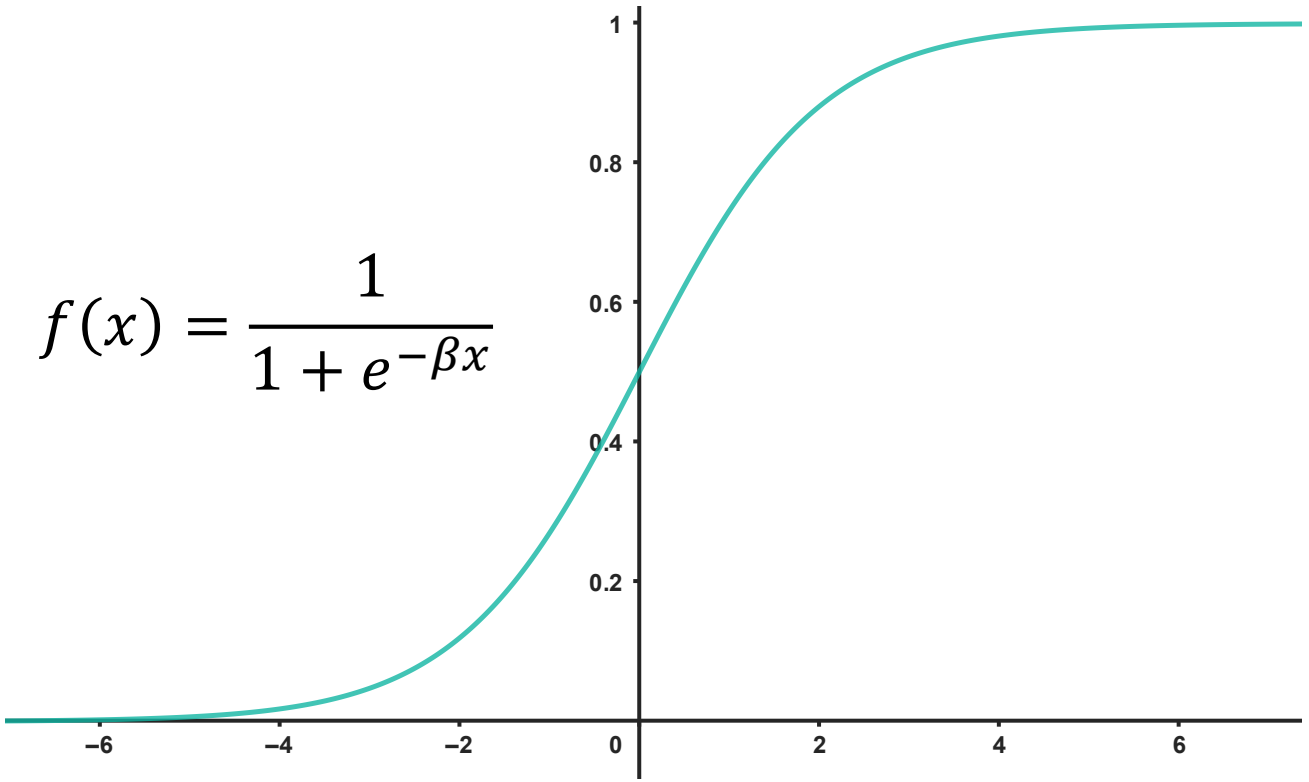
$$g(z) = \max(\epsilon z, z)$$
$$Con\ \epsilon \ll 1$$



Ejemplo : Función de activación *Sigmoide*

Sigmoid

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



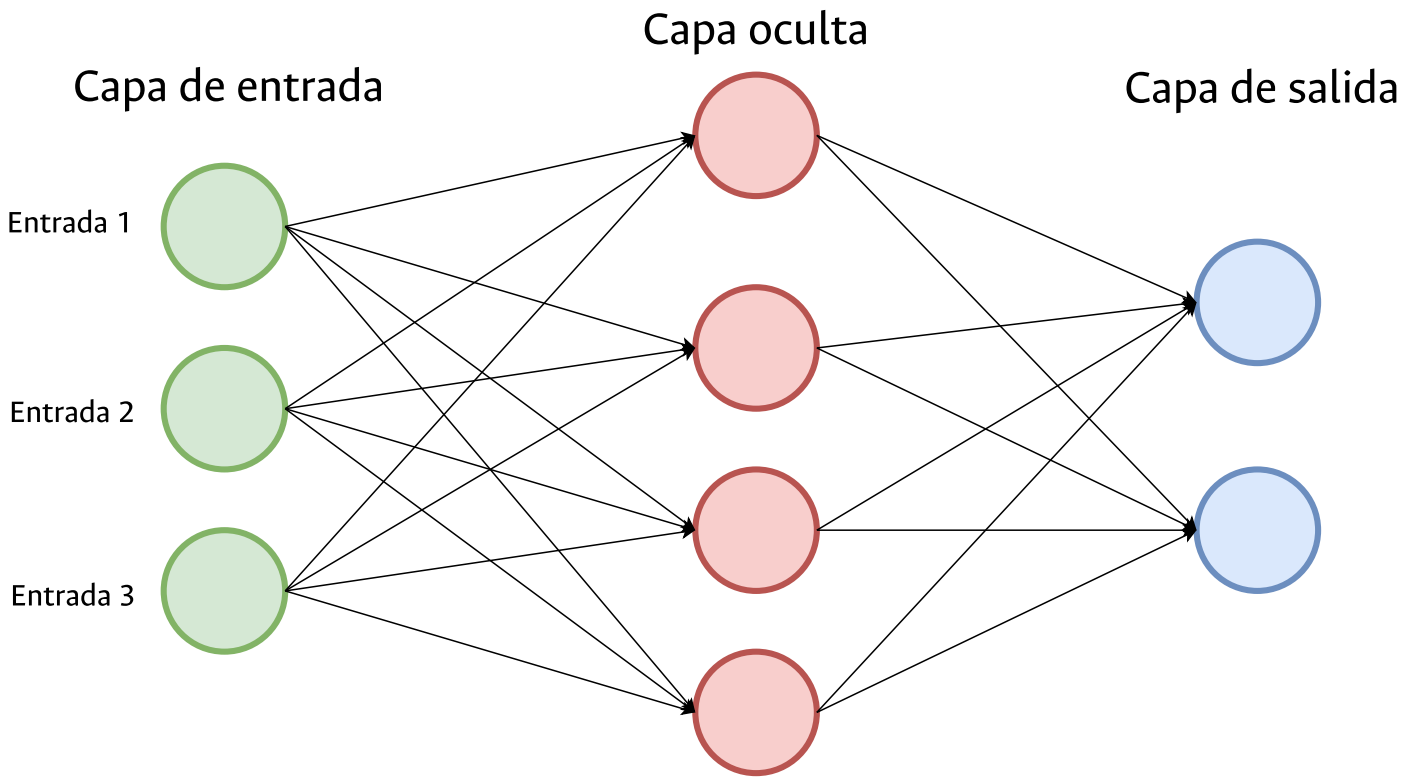
$$x = (2,3)$$
$$w = [0,1]$$
$$b = 4$$

$$(w \cdot x) + b = \big((w_1 * x_1)\big) + \big((w_2 * x_2)\big) + b$$
$$= 0 * 2 + 1 * 3 + 4$$
$$= 7$$

$$y = f(w \cdot x + b) = f(7) = 0.999$$

Introducción a redes neuronales en Keras

Redes neuronales : Perceptrón multicapa (MLP)



Una red neuronal no es más que un conjunto de neuronas conectadas en capas, una tras otra. En la imagen vemos una red que tiene:

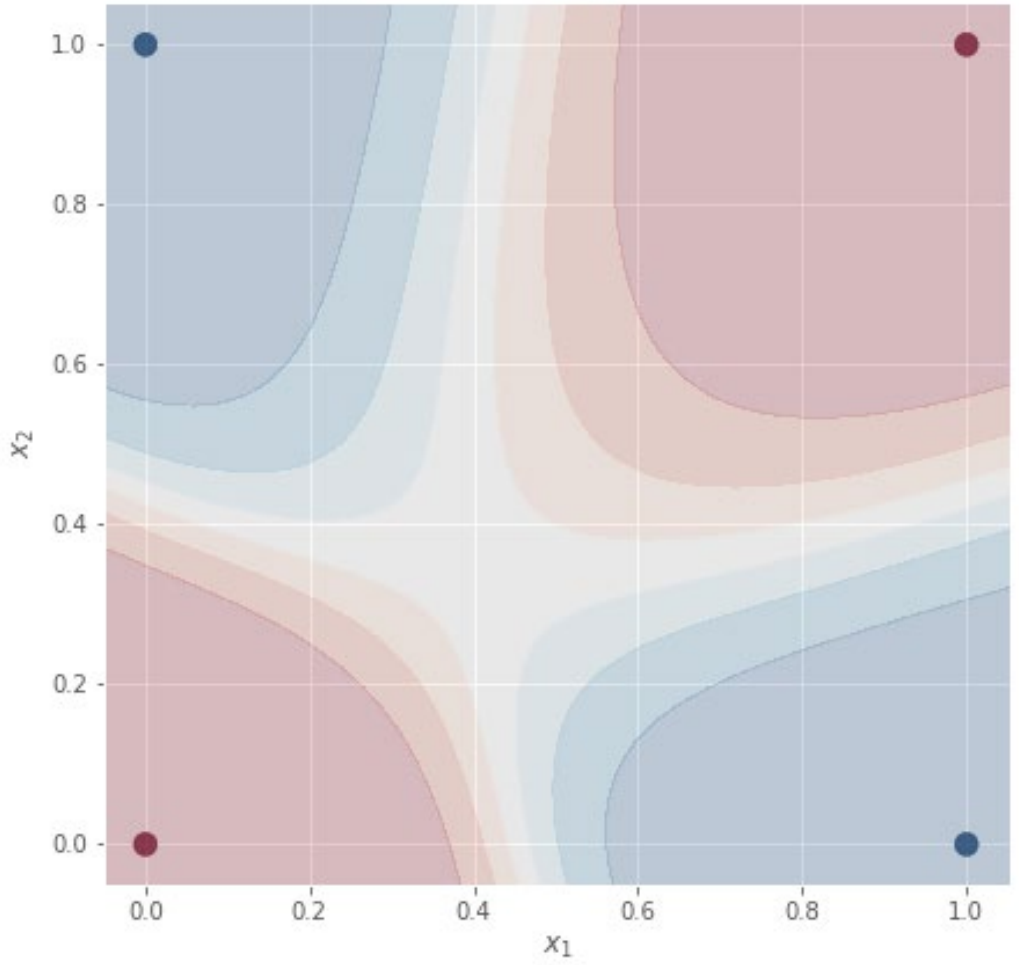
- 3 entradas
- Una capa oculta con 4 neuronas
- Una capa de salida con 2 neuronas.

Una **capa oculta** es cualquier capa de neuronas que esté entre las entradas y las salidas

Un ejemplo pequeño : La function XOR

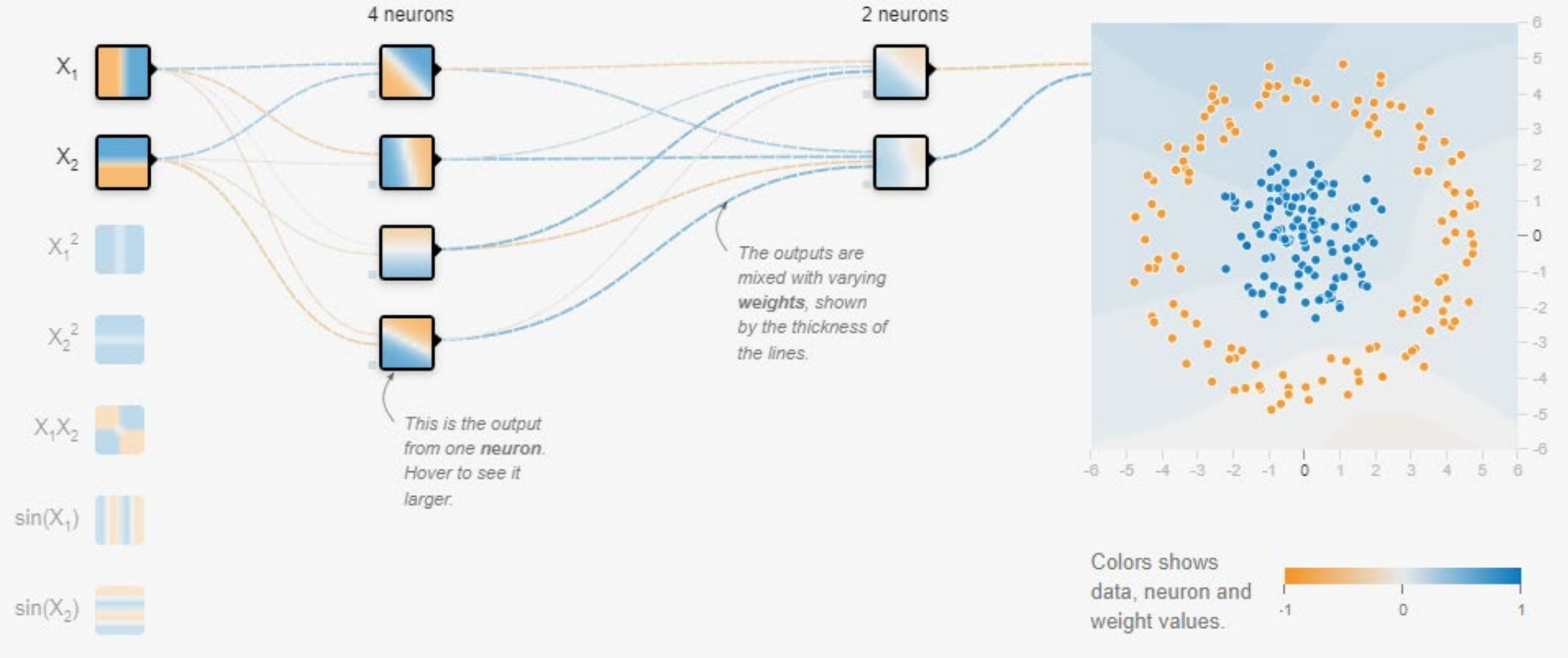
- Un perceptrón multicapa sí puede modelar regiones de decisión más complejas

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Introducción a redes neuronales en Keras

Un ejemplo didáctico: *Tensorflow Playground*



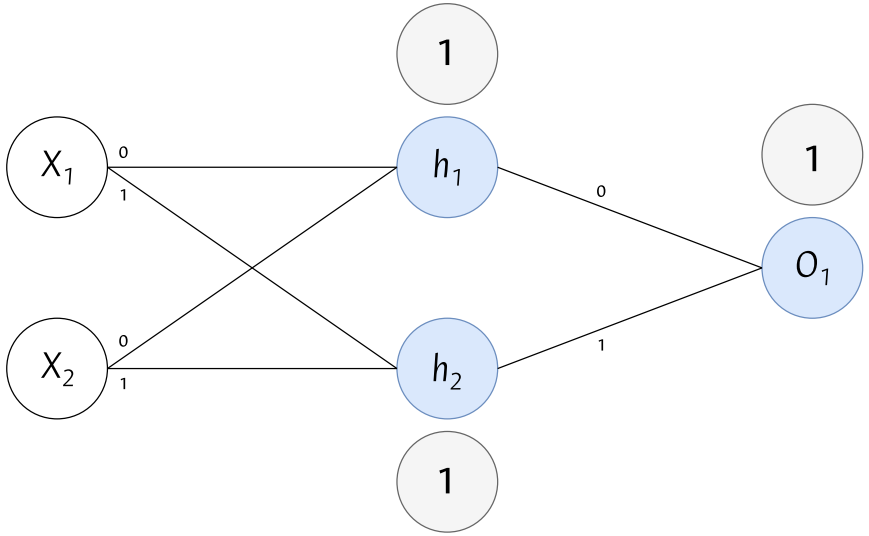
A Neural Network Playground[Click]

Tensorflow Playground es un visualizador interactivo de redes neuronales. Con el, pueden simularse pequeñas redes neuronales en tiempo real en el navegador y ver los resultados al instante.

Introducción a redes neuronales en Keras

Un ejemplo

Capa de Entrada | Capa Oculta | Capa de Salida



- Supongamos que todas las neuronas tienen los mismos pesos: [0, 1]
- Supongamos que todas las neuronas tienen el mismo bias: 1
- Función de activación f : *sigmoide*

¿Cuál es la salida del modelo si (x1, x2) = (2,3)?

Solución :

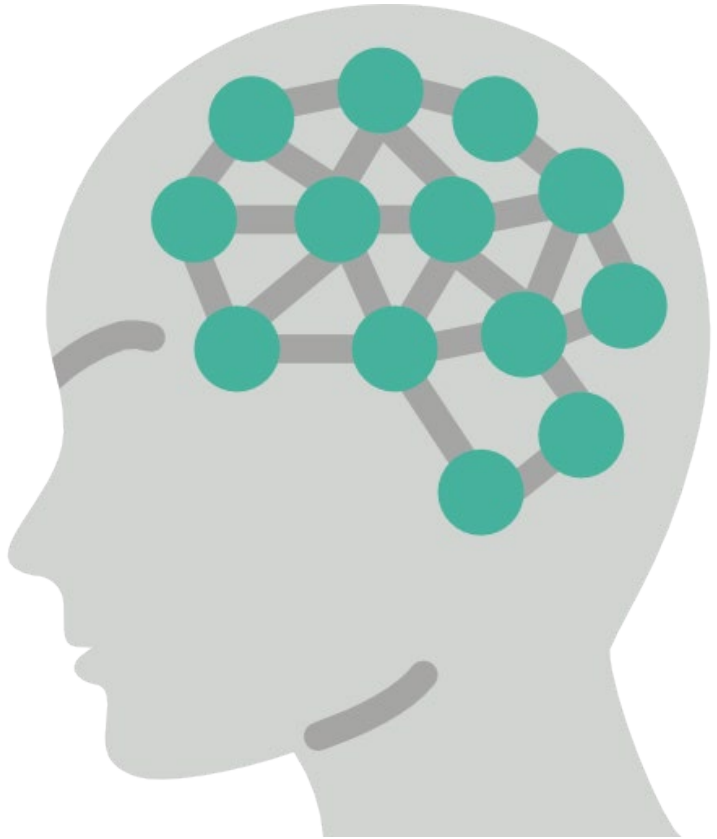
$$h_1 = h_2 = f(w \cdot x + b)$$
$$= f((0 * 2) + (1 * 3) + 0)$$
$$= f(3)$$
$$= 0.9526$$

$$O_1 = f(w \cdot [h_1, h_2] + b)$$
$$= f((0 * h_1) + (1 * h_2) + 0)$$
$$= f(0.9526)$$
$$= 0.7216$$

Agenda

- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

2 Entrenamiento de una red neuronal



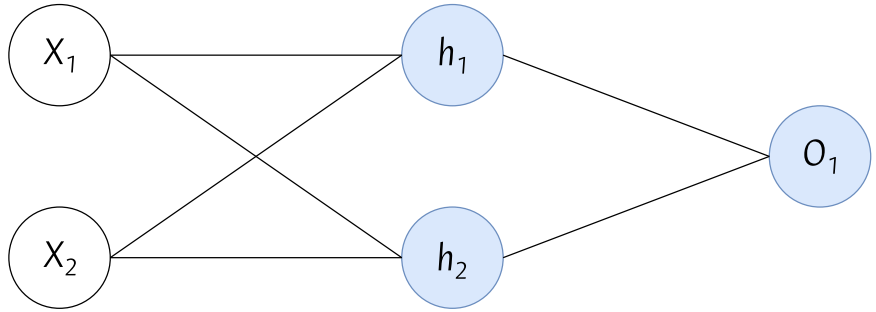
Entrenamiento de una red neuronal

Entrenamiento - ¿Qué significa en general?

- Vamos a entrenar una red neuronal para que aprenda la función **XOR**

Necesitamos un modelo con:
- Una capa de entrada de dos neuronas
- Una capa intermedia (al menos)
- Una capa de salida con una neurona:
  - Función de activación: ***sigmoid***
  - Función de pérdida: ***binary cross-entropy***

Capa de Entrada Capa Oculta Capa de Salida



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Entrenamiento de una red neuronal

Ejemplo de función de perdida

Mean – Squared - Error

El **MSE** es una función de pérdida que se usa para el entrenamiento de modelos en tareas de regresión. Supongamos que tenemos y_{true} y y_{pred} , los valores reales y los predichos por el modelo, respectivamente

y_{true}	y_{pred}	$[y_{true} - y_{pred}]$	$[y_{true} - y_{pred}]^2$
4	3	1	1
1	3	2	4
4	4	0	0
2	3	1	1
0	3	3	9

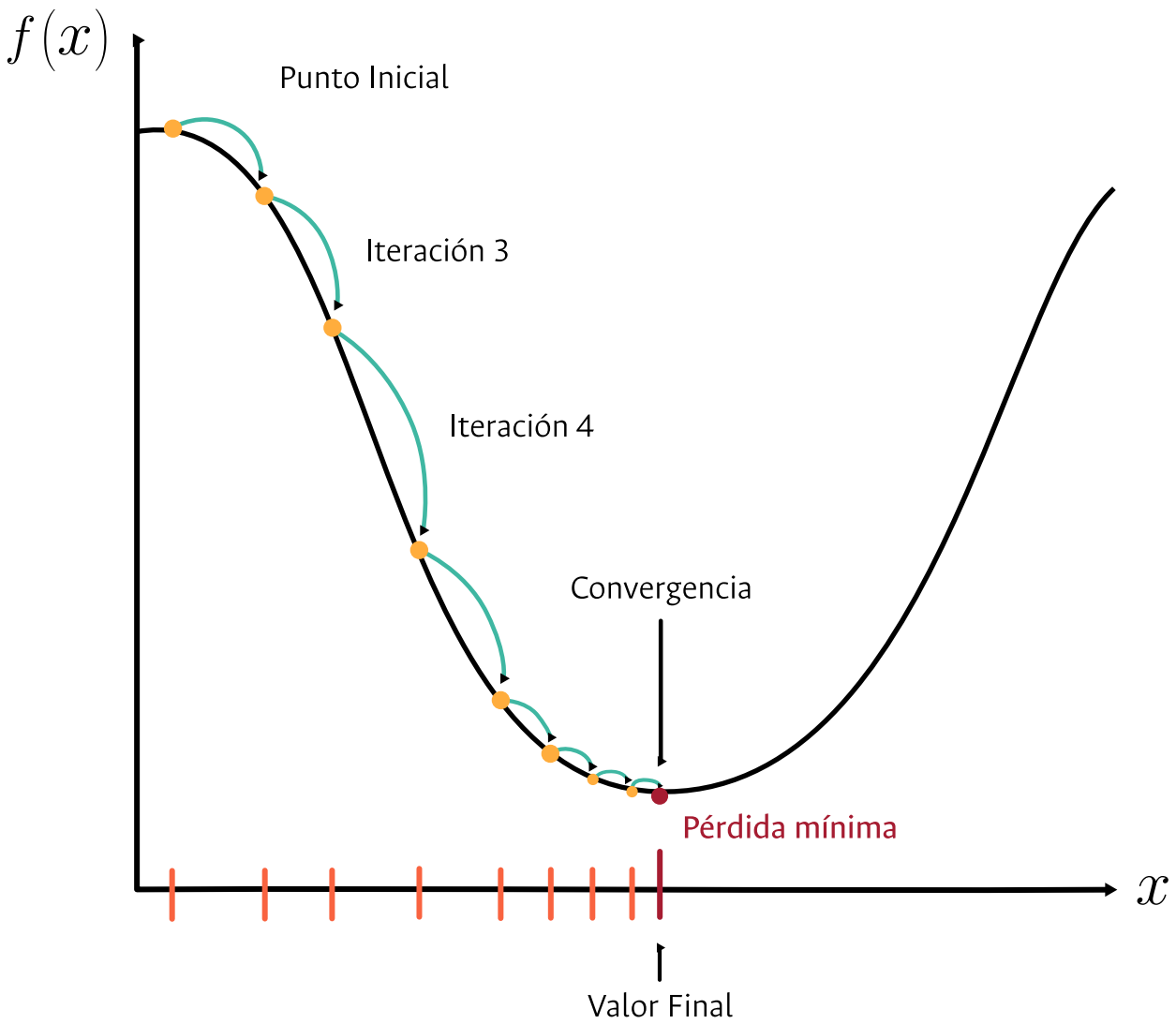
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

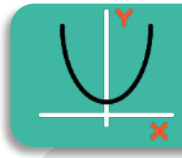
Solución :

$$\text{MSE} = \frac{1+4+0+1+9}{5} = \frac{15}{5} = 3$$

Entrenamiento de una red neuronal

Entrenamiento - ¿Cómo se ajustan los pesos del modelo?





Gradiente Descendente

Es un método iterativo. En cada iteración:

- Se calculan los gradientes de la función de pérdida.
- Cada parámetro se modifica proporcionalmente según la magnitud y en dirección opuesta a la del gradiente.
- Se calculan de nuevo las predicciones usando el modelo con los parámetros modificados.

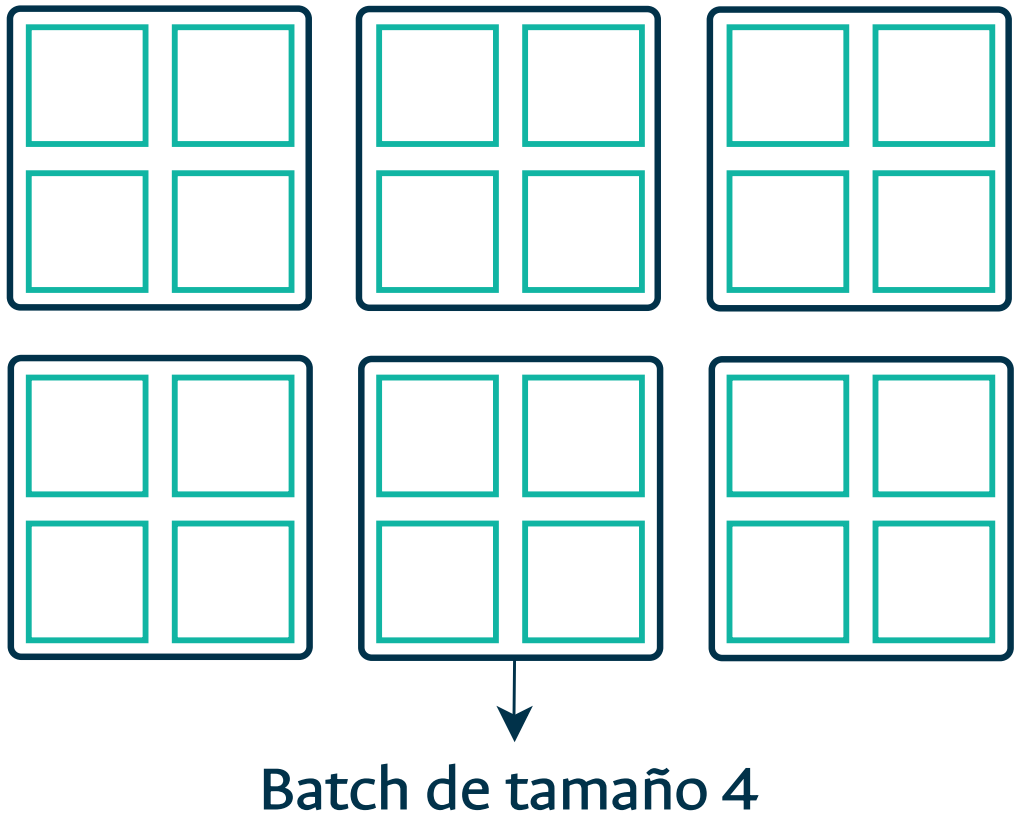
Gradiente Descendente en Batch

Problema :

- Si nuestro conjunto de datos es muy grande, la iteración del gradiente descendente se vuelve **computacionalmente costosa**.

Solución :

- Partir el conjunto de datos en grupos (batch) **más pequeños.**
- El cálculo de la función de pérdida y las correcciones se hace en cada grupo por separado, de forma secuencial



Entrenamiento de una red neuronal

Entrenamiento

Epoch

Una epoch representa una iteración sobre el conjunto de datos

Batch

No podemos pasar todo el conjunto de datos a la red neuronal de una vez. Así que dividimos el conjunto de datos en varios batches o lotes

Iteración

Si tenemos 1000 imágenes como datos y un tamaño batch de 20, entonces una epoch debería ejecutarse 1000/20 = 50 iteraciones

Gradiente Descendente Estocástico

Problema :

- Si nuestro conjunto de datos es muy grande, la iteración del gradiente descendente se vuelve computacionalmente costosa.

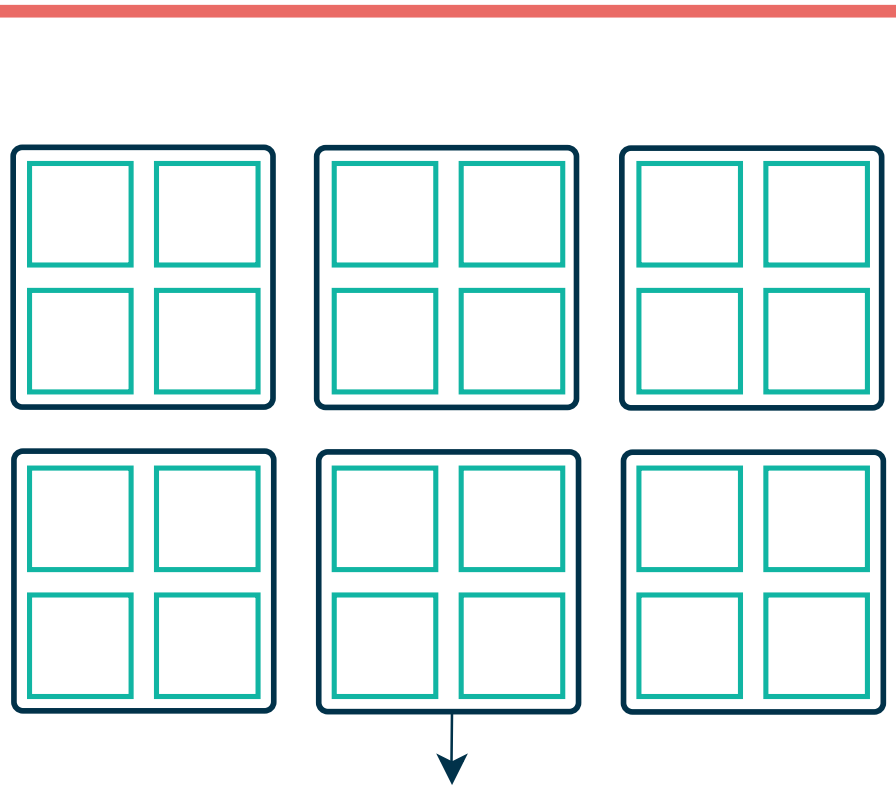
Solución :

- Partir el conjunto de datos en grupos (batch) más pequeños.
- El cálculo de la función de pérdida y las correcciones se hace en cada grupo por separado, de forma secuencial

Entrenamiento de una red neuronal

Entrenamiento

Dataset : 24 muestras



Batch de tamaño 4

6 Batches

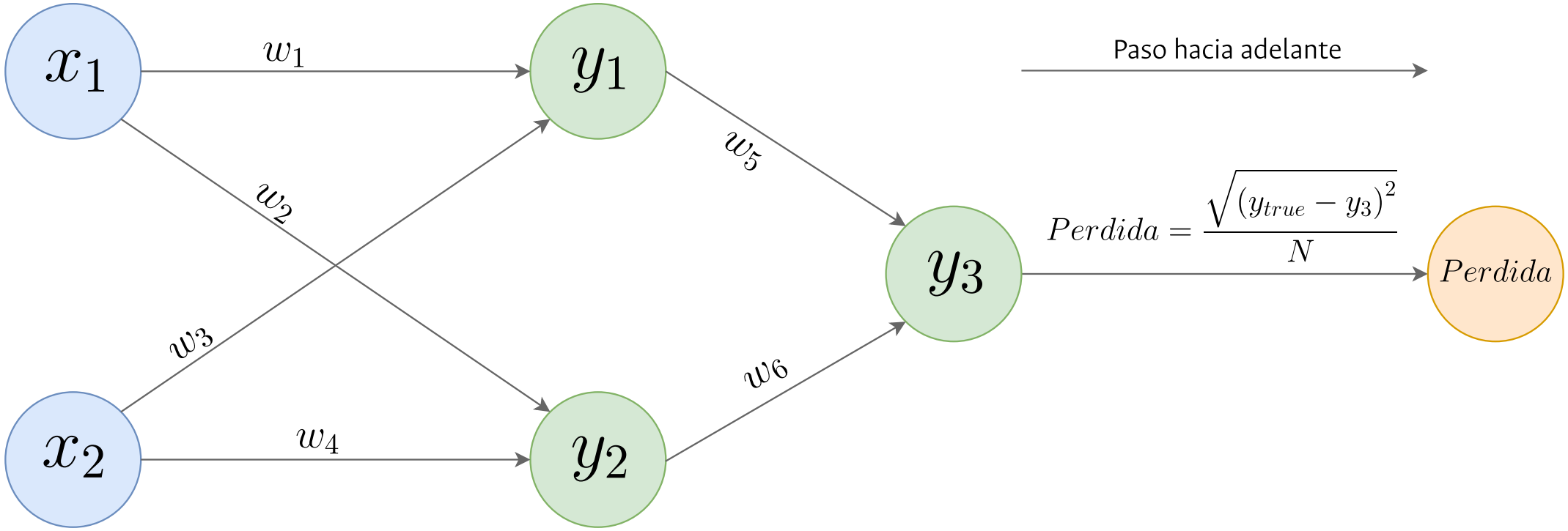
= 6 iteraciones por Epoch

1 Epoch
=
Una iteración sobre TODOS los datos

Entrenamiento de una red neuronal

Entrenamiento - ¿Cómo se ajustan los pesos del modelo?

Backpropagation : Una forma eficiente de calcular el gradiente

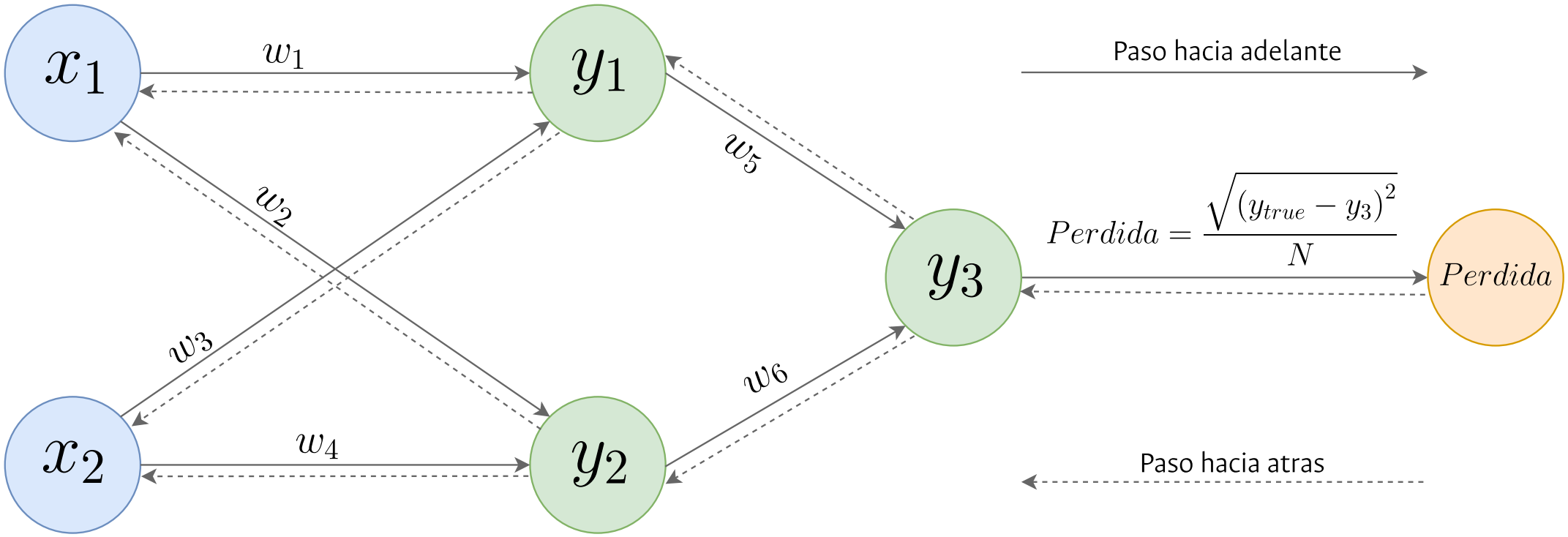


- Los ejemplos de entrenamiento se propagan hacia adelante, y se calculan predicciones.
- Con las predicciones se calcula el valor de la función de pérdida.

Entrenamiento de una red neuronal

Entrenamiento - ¿Cómo se ajustan los pesos del modelo?

Backpropagation : Una forma eficiente de calcular el gradiente

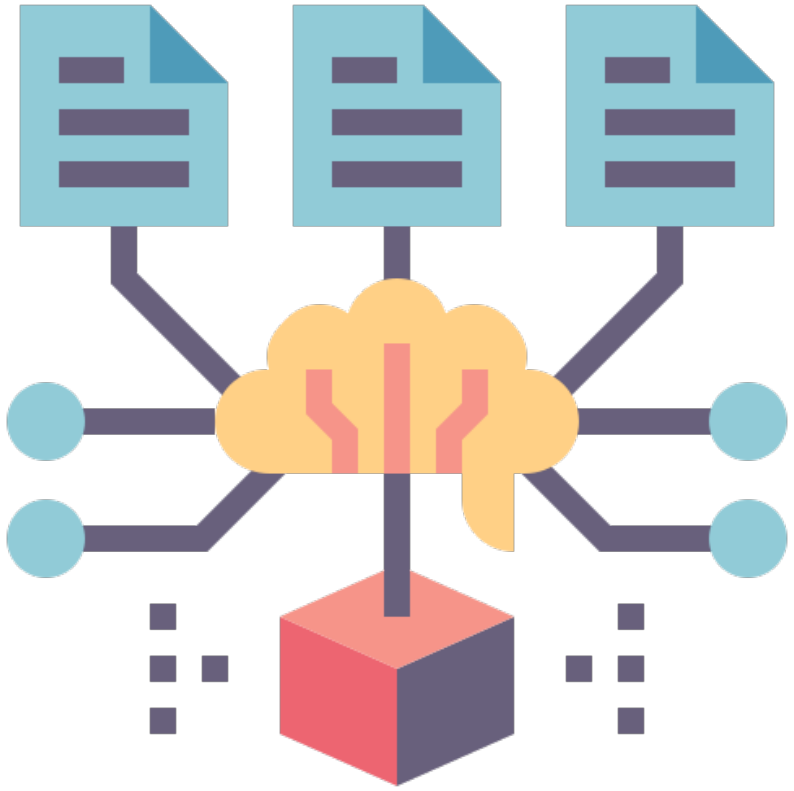


- Usando **la regla de la cadena**, se calculan las derivadas parciales de la función de pérdida respecto a cada parámetro del modelo. **Es decir, podemos calcular en dónde se están generando los errores.**

Agenda

- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

3 Proceso de Modelamiento y Entrenamiento



Proceso de Modelamiento y Entrenamiento

En resumen :

- **Neurona** : Unidad Fundamental
  - **Perceptrón / Perceptrón multicapa**
    - **Función de activación**

Modelado

- **Función de perdida**
  - **Backpropagation**
    - **Gradiente descendente**

Entrenamiento

Agenda

- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

4 *Keras* y tipos de capas en una red neuronal

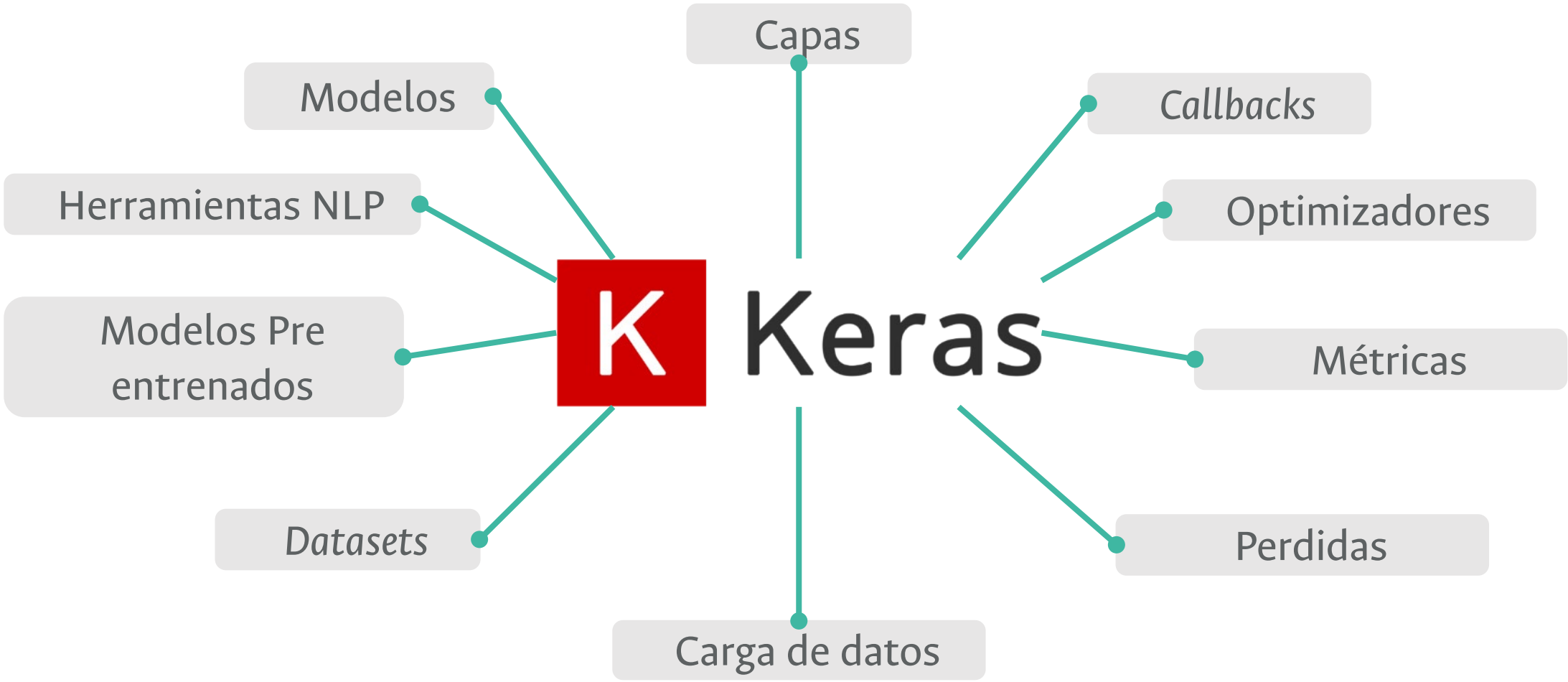
Panorama general de *Keras*

- Desarrollado por **François Chollet**
- Framework de *Python* de alto nivel capaz de ejecutarse sobre *TensorFlow*,
- Principios directores :
  - Facilidad de uso
  - Modularidad
  - Fácil extensibilidad
  - Trabajar con Python
- Muy popular
- Creación rápida de prototipos
- Fácil de extender
- Muchos modelos pre entrenados

François Chollet

Keras y tipos de capas en una red neuronal

Panorama general de *Keras*

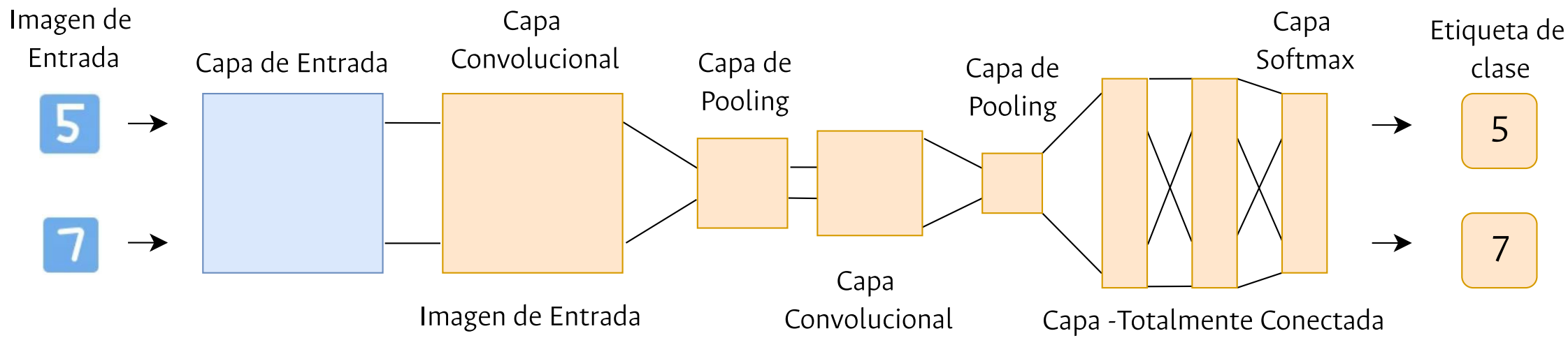


Keras y tipos de capas en una red neuronal

Capas

Capas

- Las capas son los bloques de construcción de los modelos.
- *Keras* proporciona varias capas predefinidas para construir diferentes tipos de redes.
- Las capas tienen diferentes métodos que permiten obtener y establecer sus pesos, definir una función de inicialización, controlar la regularización, la función de activación, etc.

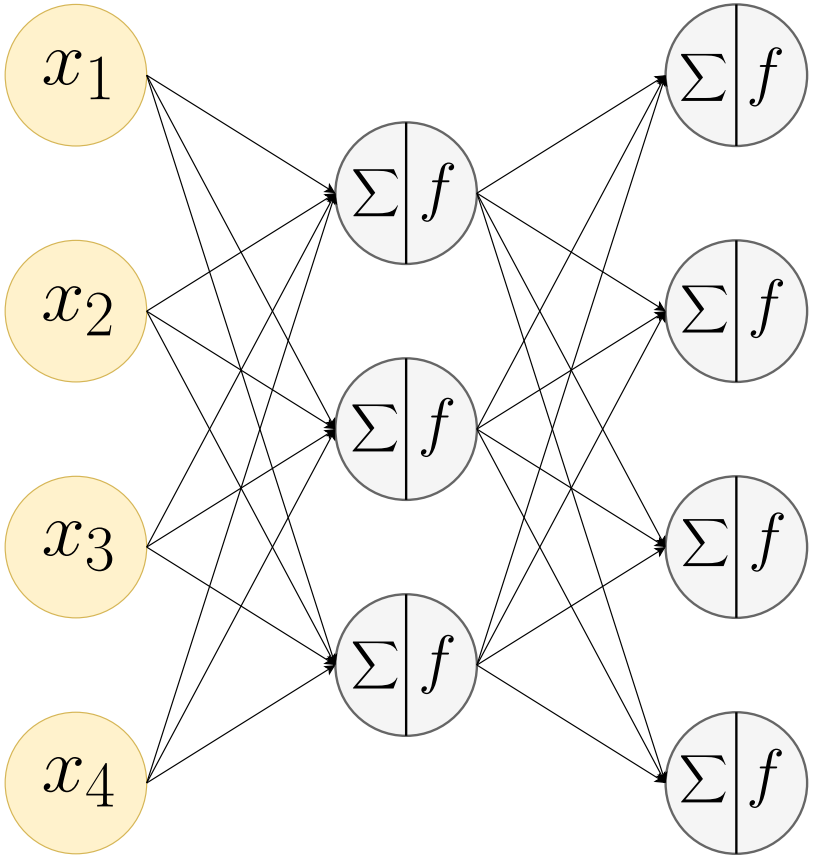


Keras y tipos de capas en una red neuronal

Tipos de Capas

Básicas o Esenciales

- Input object
- Dense layer
- Activation layer



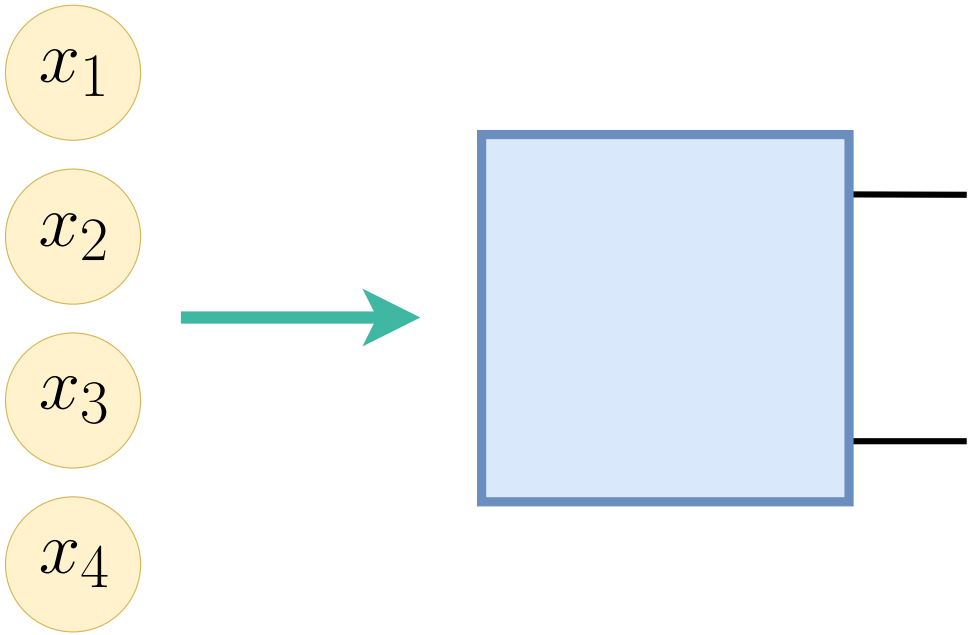
Para más información puedes dar click en cada uno de los distintos nombres de las capas

Keras y tipos de capas en una red neuronal

Input Layer

Entrada

Capa de Entrada



Input Layer

- Es la primera capa del modelo
- Sirve para definir las dimensiones y el tamaño de los datos de entrada (Parámetro *shape*)

```
tf.keras.Input(shape=(...))
```

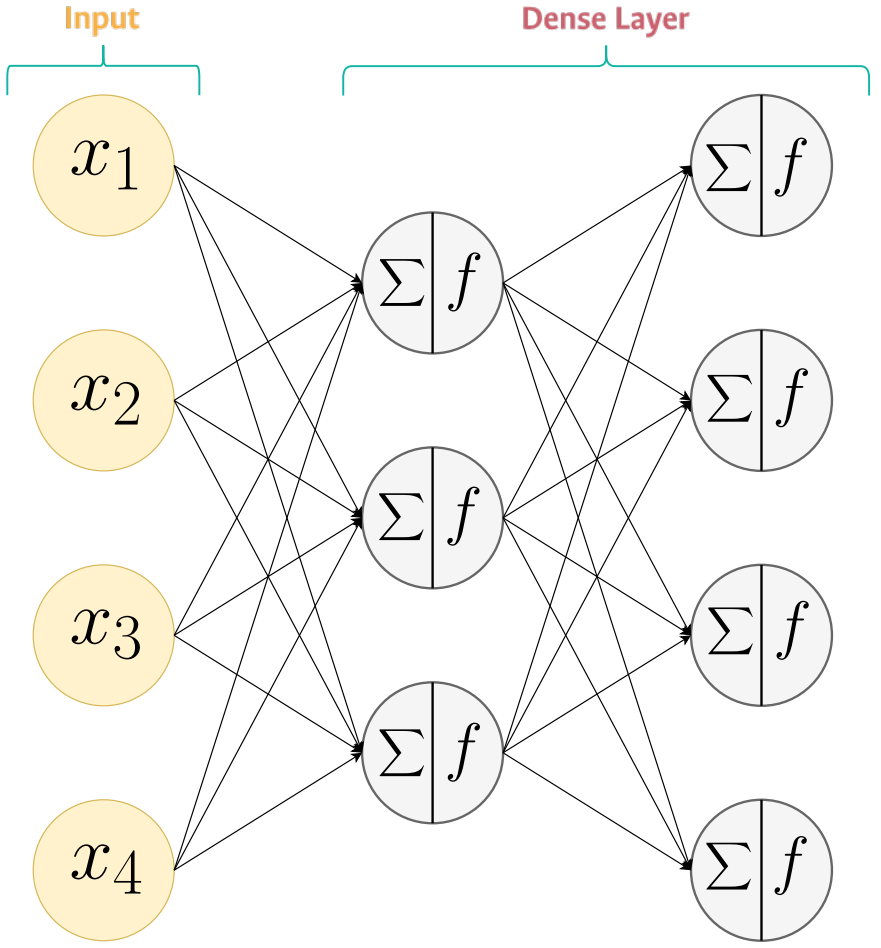
Keras y tipos de capas en una red neuronal

Dense Layer

Dense Layer

- Agrega una capa de neuronas.
- Parámetros:
  - **units** : Define cuántas neuronas
  - **activation** : Define la función de activación

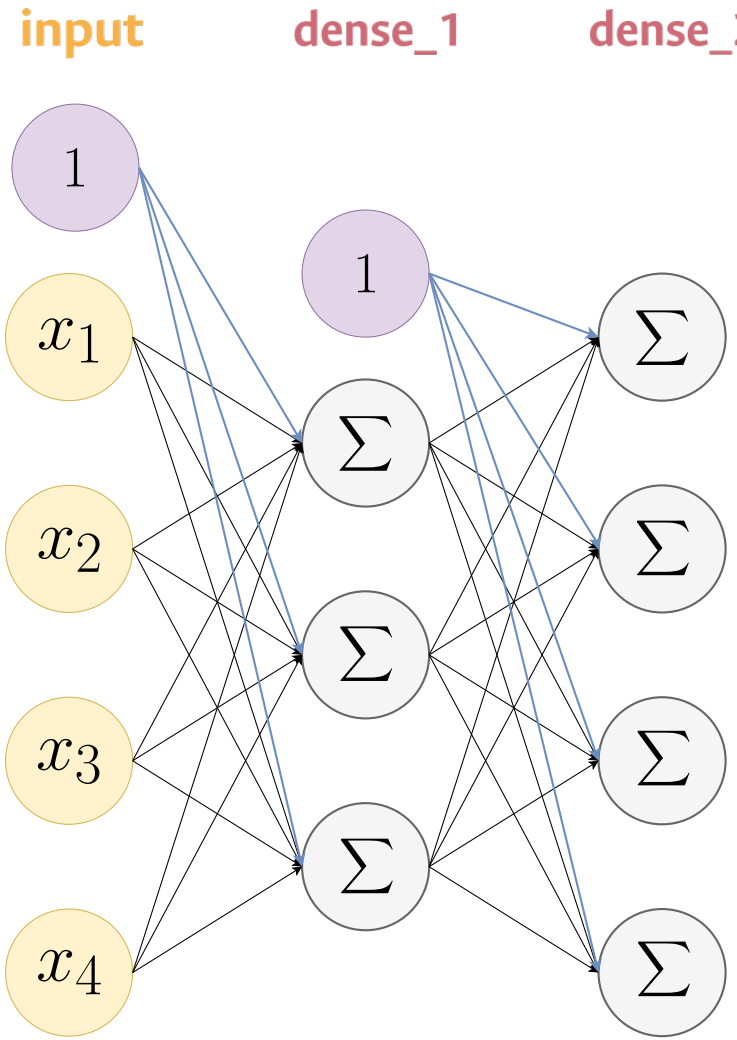
```
tf.keras.layers.Dense(
        units=...,
        activation=...)
```



f es la función de activación

Keras y tipos de capas en una red neuronal

Dense Layer - Ejemplo



Input

```
tf.keras.layers.input(shape=(4,)
```

Primera Capa Densa

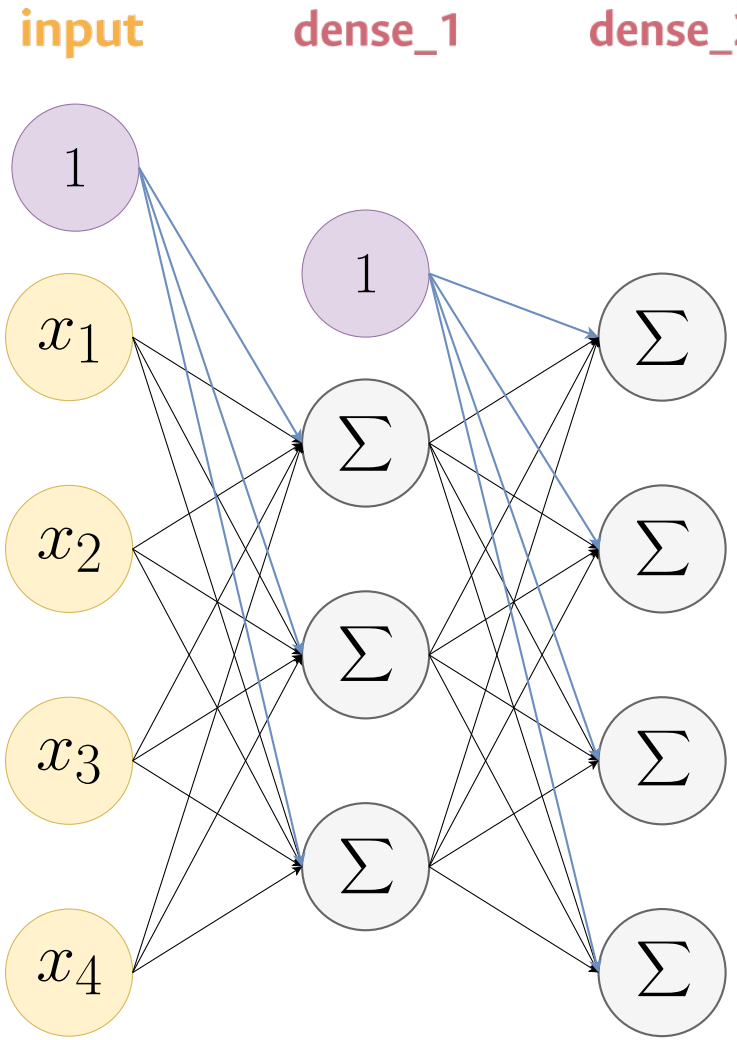
```
tf.keras.layers.dense(units=3)
```

Esta capa añade (4+1)*3 = 15 parámetros al modelo

¿Por qué (4+1)? 4 por la entrada, y 1 porque por defecto la capa incluye un parámetro de sesgo (bias)

Keras y tipos de capas en una red neuronal

Dense Layer - Ejemplo



Segunda Capa Densa

```
tf.keras.layers.dense(units=4)
```

Esta capa añade (3+1)*4 = 16 parámetros al modelo.

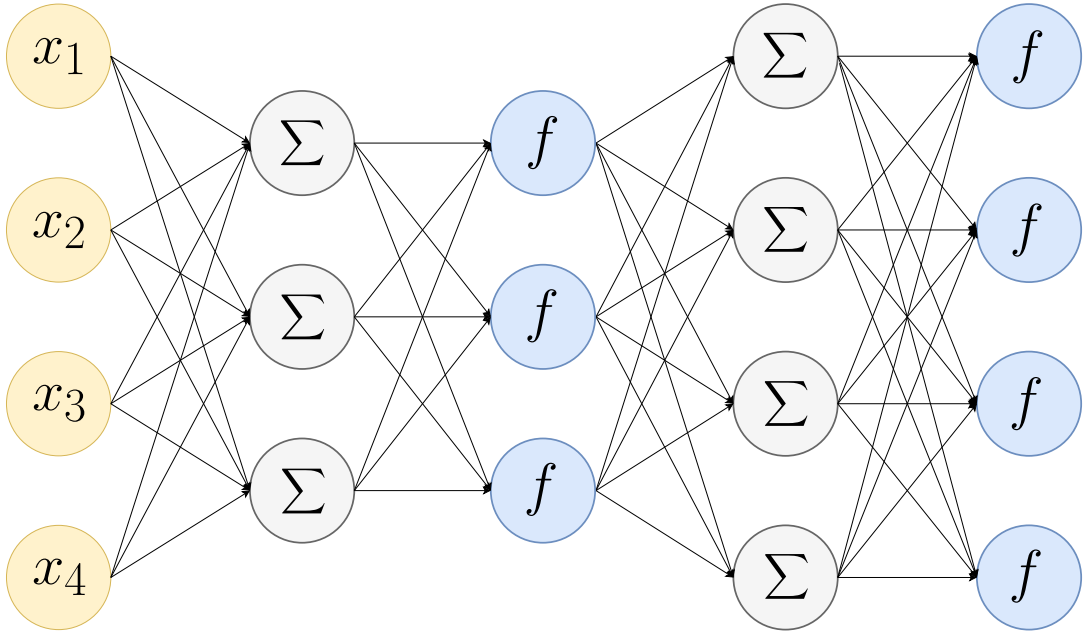
Resultado

En total el modelo tendrá **31 parámetros**, la entrada será de tamaño 4, y la salida también será de tamaño 4.

Keras y tipos de capas en una red neuronal

Dense Layer - Ejemplo

Activation Layer



Usar *tf.keras.layers.Dense*, y luego *tf.keras.layers.Activation*, es equivalente a definir la activación dentro de *tf.keras.layers.Dense*.

Pero, usar la activación por separado permite, por ejemplo, **recuperar la salida** de la capa densa antes de ser trasformada por la activación.

- Agrega una función de activación.

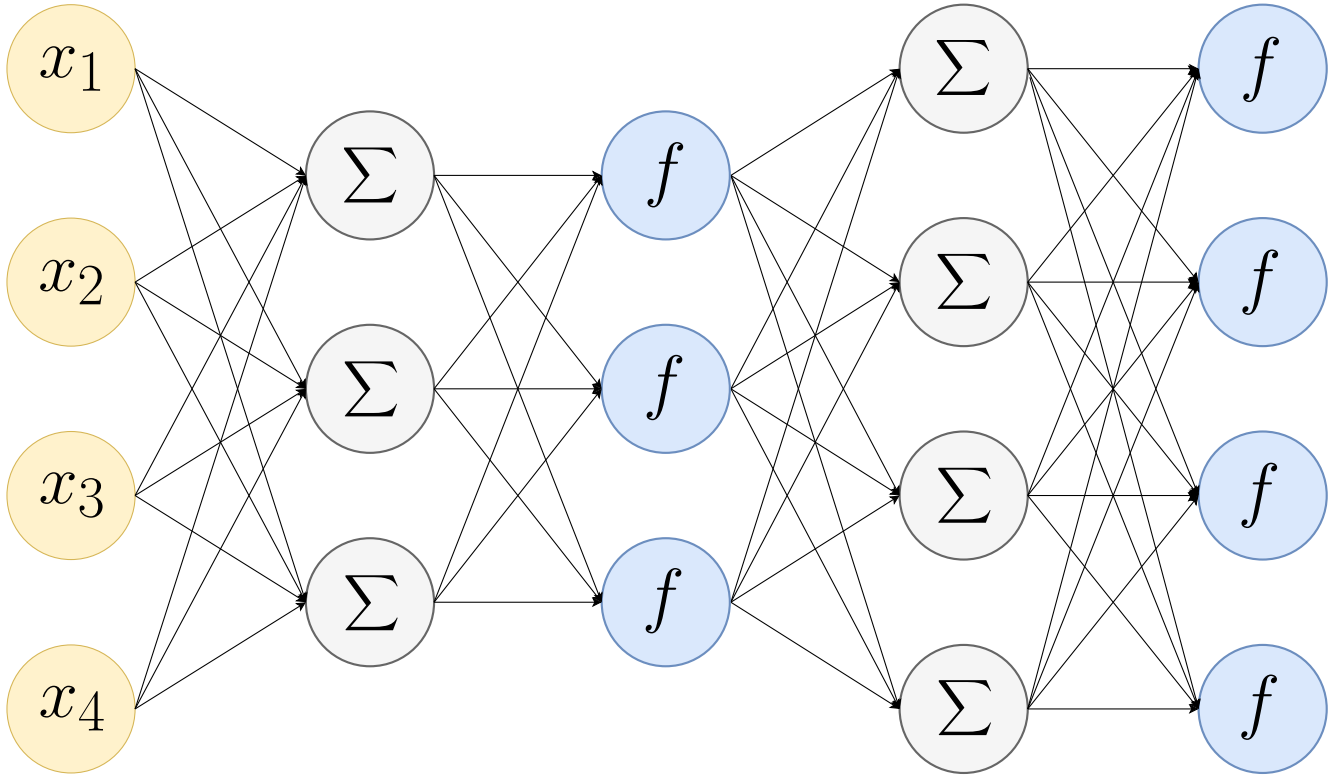
```
tf.keras.layers.Activation(
        activation)
```

Keras y tipos de capas en una red neuronal

Dense Layer - Ejemplo

Activación

- ReLU layer
- Softmax layer
- LeakyReLU layer
- PReLU layer
- ELU layer
- ThresholdedReLU layer



Para más información puedes dar click en cada uno de los distintos nombres de las capas

Keras y tipos de capas en una red neuronal

Otros tipos de Capas

Convolucionales

- Conv1D layer
- Conv2D layer
- Conv3D layer

Pooling

- MaxPooling2D layer
- AveragePooling2D layer
- GlobalMaxPooling2D layer
- GlobalAveragePooling2D layer

Regularización

- Dropout
- SpatialDropout1D layer
- SpatialDropout2D layer
- ActivityRegularization layer

Para más información puedes dar click en cada uno de los distintos nombres de las capas

Keras y tipos de capas en una red neuronal

Otros tipos de Capas

Normalización

- BatchNormalization layer
- LayerNormalization layer
- UnitNormalization layer

Merging

- Concatenate layer
- Average layer
- Maximum layer
- Minimum layer
- Add layer
- Subtract layer
- Multiply layer
- Dot layer

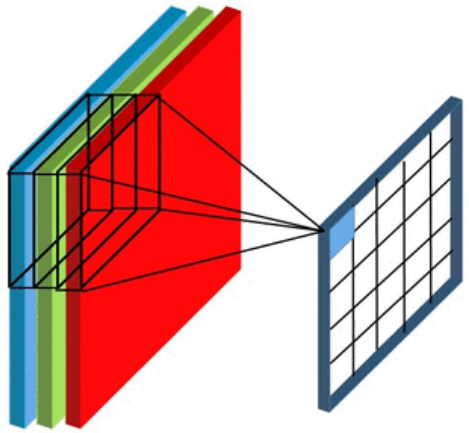
Para más información puedes dar click en cada uno de los distintos nombres de las capas

Keras y tipos de capas en una red neuronal

Otros tipos de Capas

Gráficamente

Capas Convolucionales



Pooling

Feature Map

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

Max Pooling

Average Pooling

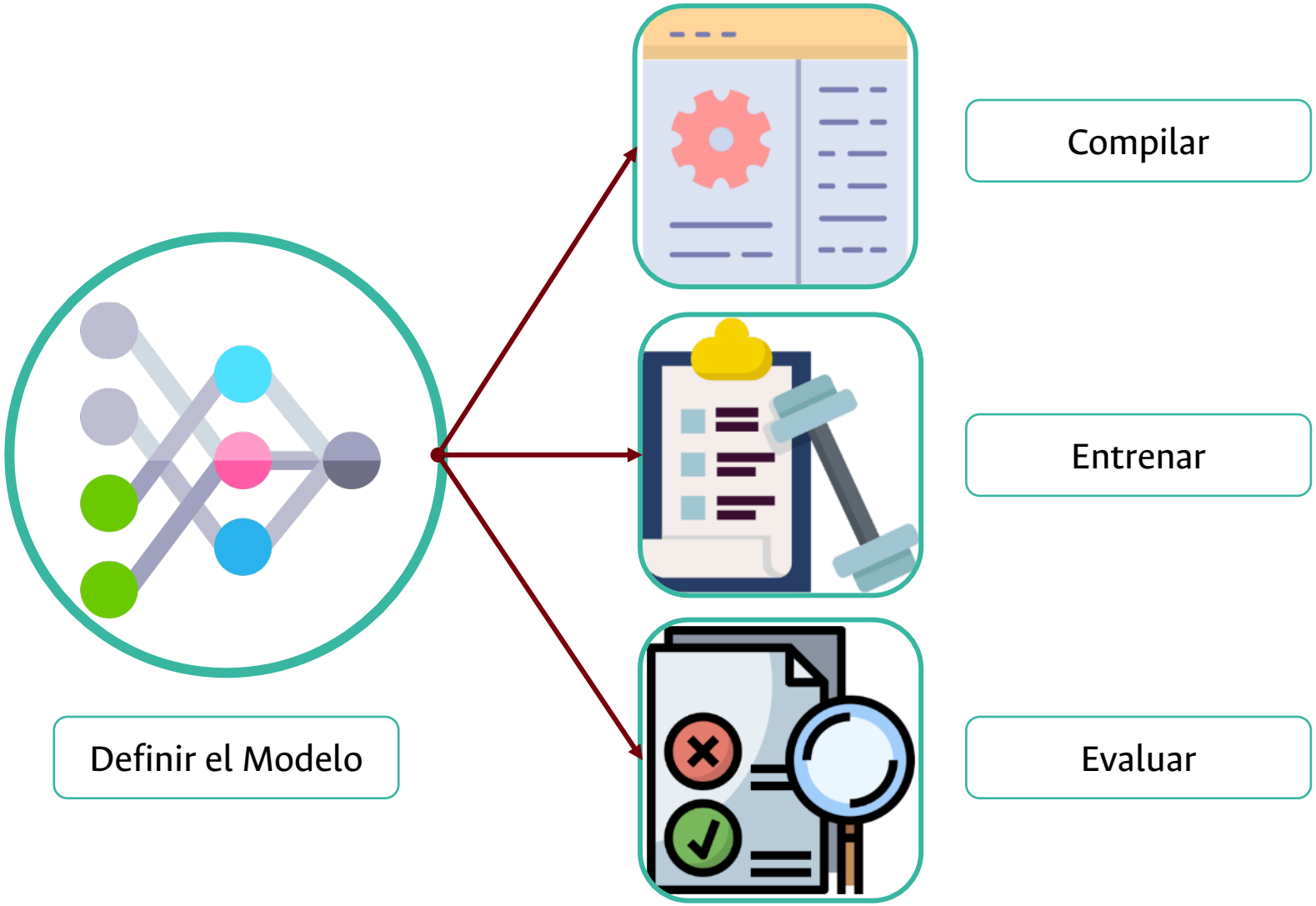
Sum Pooling

Para mejor visualización de las animaciones, dar Click en cada una de las imágenes

Agenda

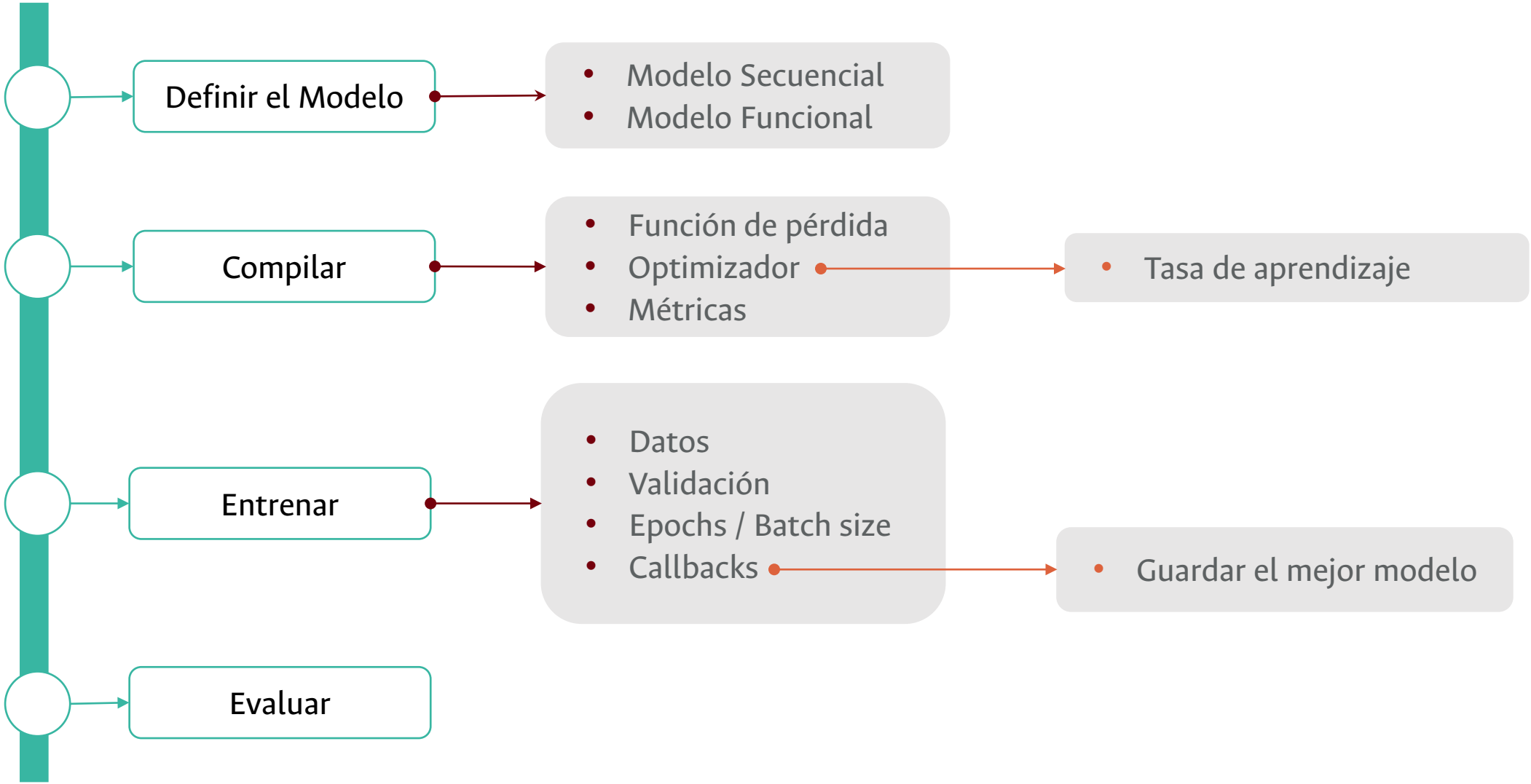
- Introducción a redes neuronales en *Keras*
- Entrenamiento de una red neuronal
- Proceso de modelamiento y entrenamiento
- *Keras* y tipos de capas en una red neuronal
- Flujo de trabajo en redes neuronales

5 Flujo de trabajo en redes neuronales



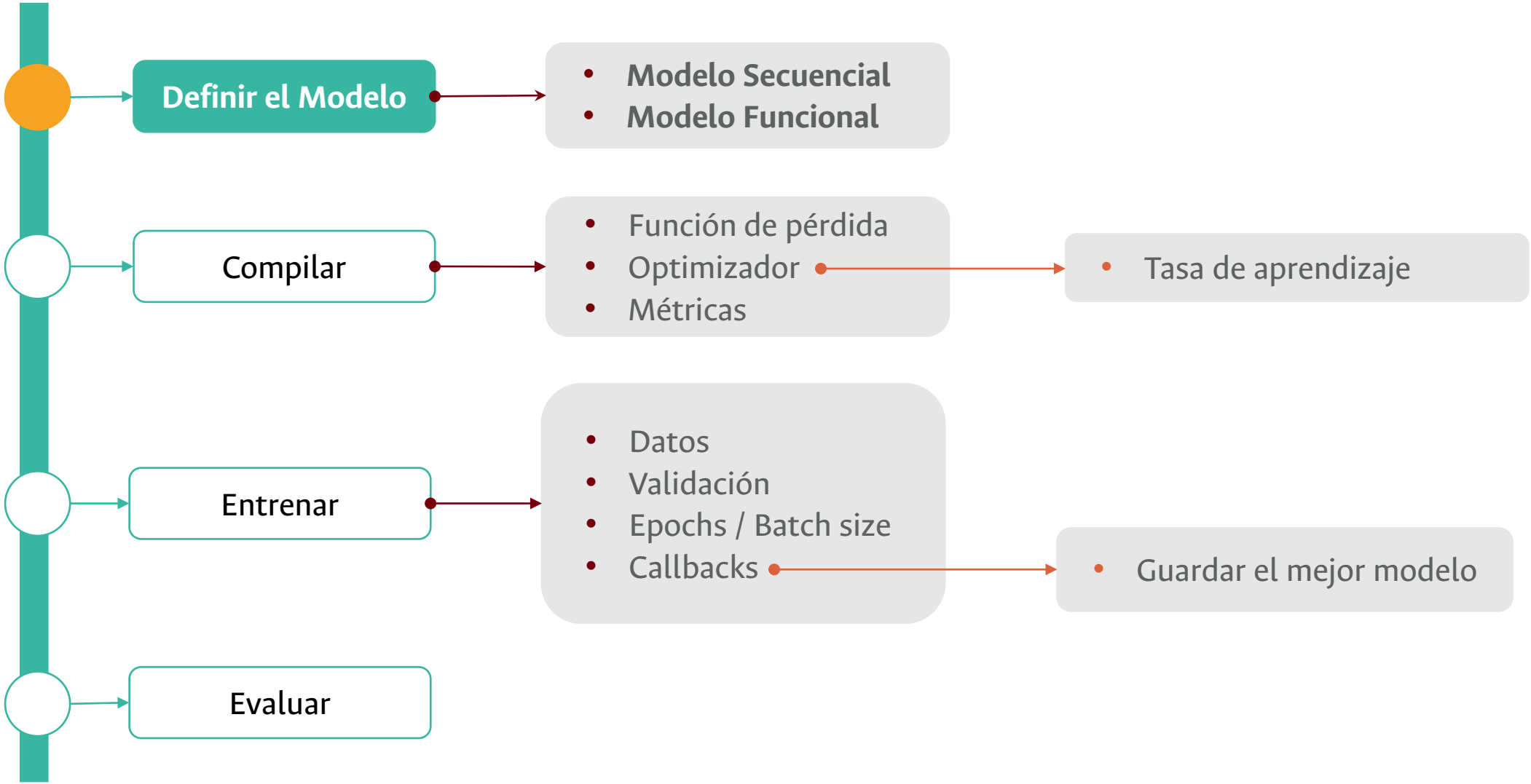
Flujo de trabajo en redes neuronales

Flujo de trabajo



Flujo de trabajo en redes neuronales

Flujo de trabajo – Definir el Modelo

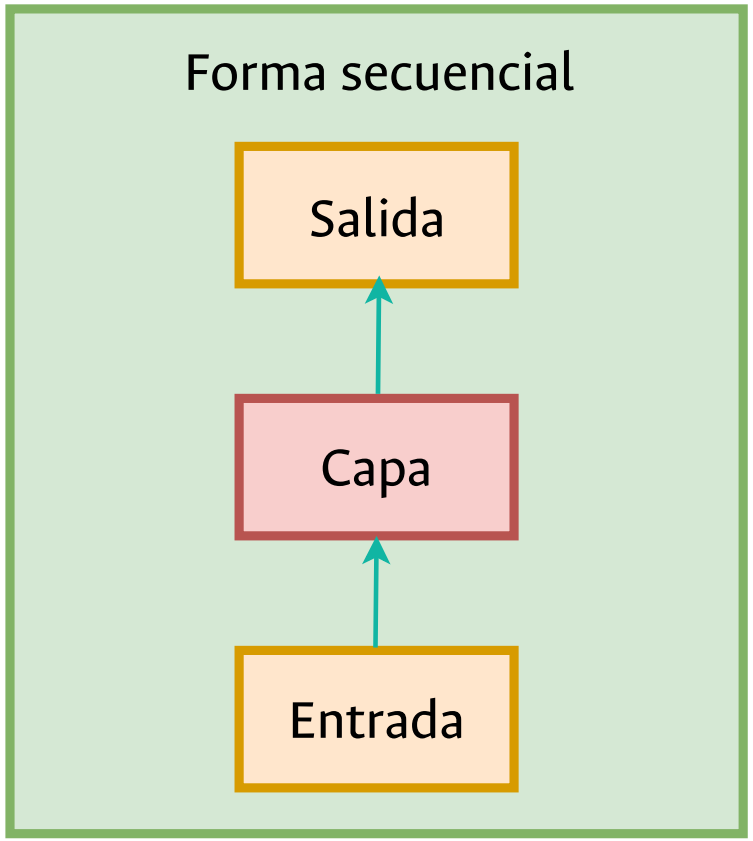


Flujo de trabajo en redes neuronales

Dos tipos de Modelos

Modelo Secuencial

- Cada capa se ubica una encima de otra.
- El orden de las capas va determinado por el orden en el código.

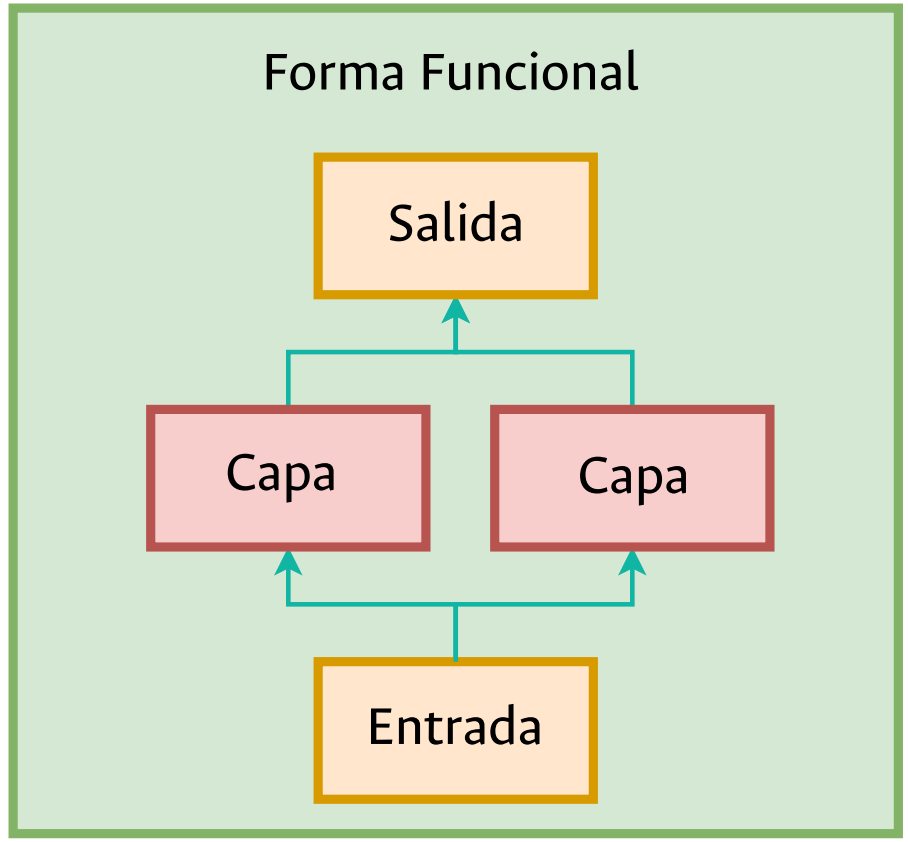


Flujo de trabajo en redes neuronales

Dos tipos de Modelos

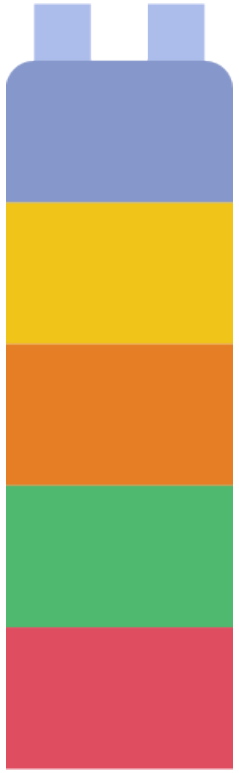
Modelo Funcional

- Las capas son funciones que reciben como argumento otras capas; el orden es flexible
- Hay más versatilidad de construcción.



Flujo de trabajo en redes neuronales

Modelo Secuencial



Modelo Secuencial

- El modelo más sencillo es el secuencial
- Las capas se apilan unas sobre otras
- El proceso de aprendizaje se configura con compilación
- El entrenamiento se realiza con una línea
- El modelo entrenado puede evaluarse fácilmente
- Y aplicado a nuevos datos

Flujo de trabajo en redes neuronales

Modelo Secuencial - Código

```
model_seq = tf.keras.models.Sequential()
```

Se define el modelo, usando la función *tf.keras.models.Sequential()*

Se agregan dos capas Densas

```
model_seq.add(
    tf.keras.layers.Dense(
        units=64,
        input_shape=(100,),
        activation="relu"
        )
    )
```

La primera compuesta por 64 neuronas con una función de activación **ReLU**

```
model_seq.add(
    tf.keras.layers.Dense(
        units=2,
        activation="softmax")
    )
```

Finalmente agregamos la capa de salida, otra capa densa de dos neuronas con una activación **SoftMax**

Flujo de trabajo en redes neuronales

Modelo Secuencial - Código

```python
@tf.function
def my_activation(x):
    return 1 / (1 + tf.exp(x))
```

Se define la función de activación como

$$f(x) = \frac{1}{1 + \exp(x)}$$

tf.keras.models.Sequential () nos permite pasar como parámetro una lista de capas para utilizar en la definición del modelo.

```python
model_seq = tf.keras.models.Sequential(
    [
        tf.keras.layers.Dense(units = 8,
                              input_shape=(2, ),
                              ),
        tf.keras.layers.Activation(my_activation),
        tf.keras.layers.Dense(2, activation=my_activation)
    ]
)
```

Primera capa de 8 neuronas

El input es de tamaño (2,)

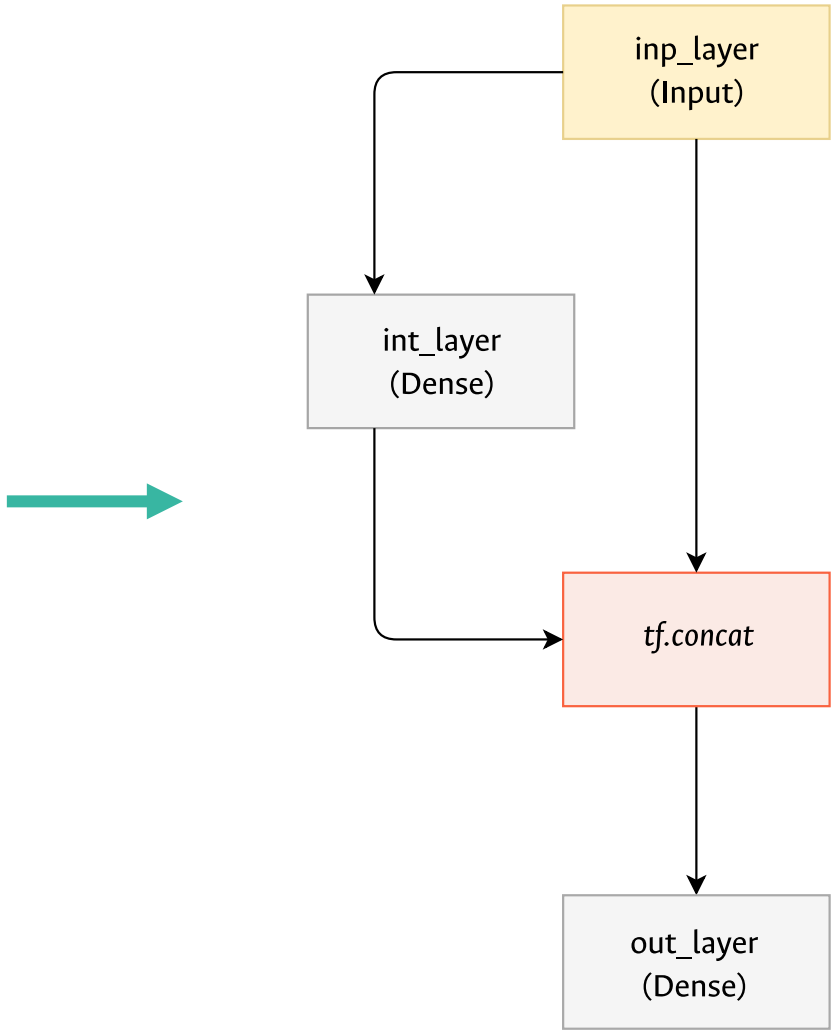
Función de activación

Salida : Capa de dos neuronas con activación Softmax

Flujo de trabajo en redes neuronales

Modelo Funcional – Código – Diagrama

El diagrama de la red que se construye del ejemplo anterior es el siguiente :



Flujo de trabajo en redes neuronales

Modelo Funcional – Código

```
inp_layer = tf.keras.layers.Input(shape=(2,))
int_layer = tf.keras.layers.Dense(units=1,
                                  activation="tanh")
out_layer = tf.keras.layers.Dense(units=1,
                                  activation="sigmoid")
```

1

Se define las dos neuronas de entrada

Se define la neurona intermedia

Se define la neurona de salida

```
int_out = int_layer(inp_layer)
y_prime = out_layer(
        tf.concat([inp_layer, int_out], axis=1)
        )
```

2

Se conecta la neurona intermedia con la capa de entrada

Se conecta la capa de salida con la neurona intermedia y la capa de entrada

Se define el modelo con *tf.keras.models.Model()*, la cual recibe dos argumentos : **inputs y outputs**

Inputs : Una lista de las capas de entrada
Outputs : Una lista con las capas de salida

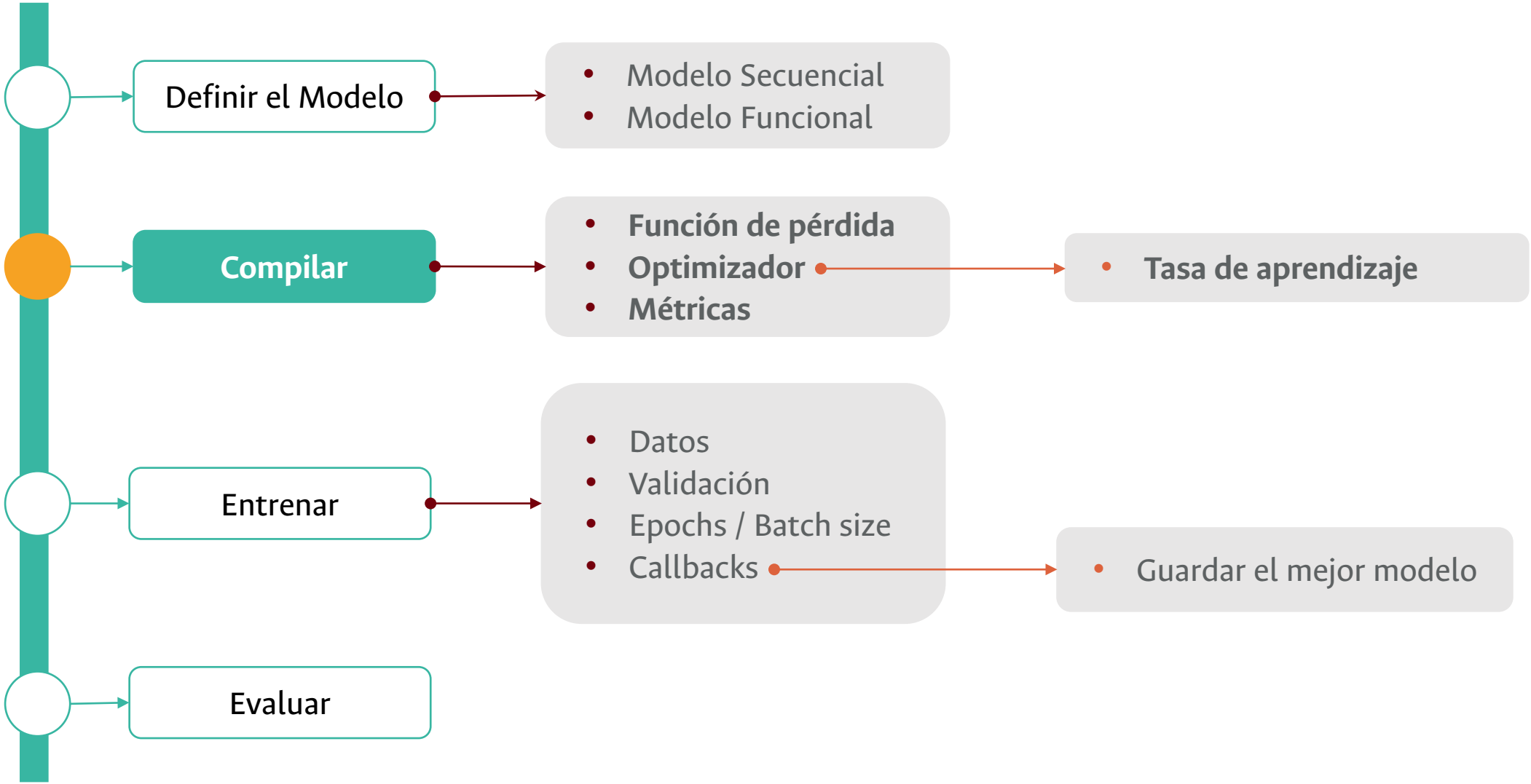
```
model = tf.keras.models.Model(
        inputs=[inp_layer],
        outputs=[y_prime]
        )
```

3

Definición del modelo

Flujo de trabajo en redes neuronales

Flujo de trabajo – Compilar



Flujo de trabajo en redes neuronales

Función de perdida

Para mas información puedes dar click en cada uno de los distintos nombres de las funciones de perdida

Entropía Cruzada

$$CE = -\sum_{i=1}^{C} y_i \log(\tilde{y}_i)$$

- C corresponde al número de clases.
- $\tilde{y}_i$ corresponde a la probabilidad de pertenencia a la clase i dado por el modelo.

Clases binarias
$y \in \{0,1\}$

Binary Cross Entropy

$$BCE = \sum -y_i \log(\tilde{y}_i) - (1 - y_i) \log(1 - y_i)$$

Más de dos clases

One-Hot Encoding

La salida es de **C** neuronas

Class Categorical Crossentropy

Clases enteras
$\{0,1,2, \dots\}$

La salida es de **una** neurona

Class Sparse Categorical Crossentropy

Flujo de trabajo en redes neuronales

Función de perdida

Funciones de perdida

Class Mean Squared Error

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

Class Mean Absolute Error

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

Class Mean Absolute Percentage Error

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y - \tilde{y}_i|}{y}$$

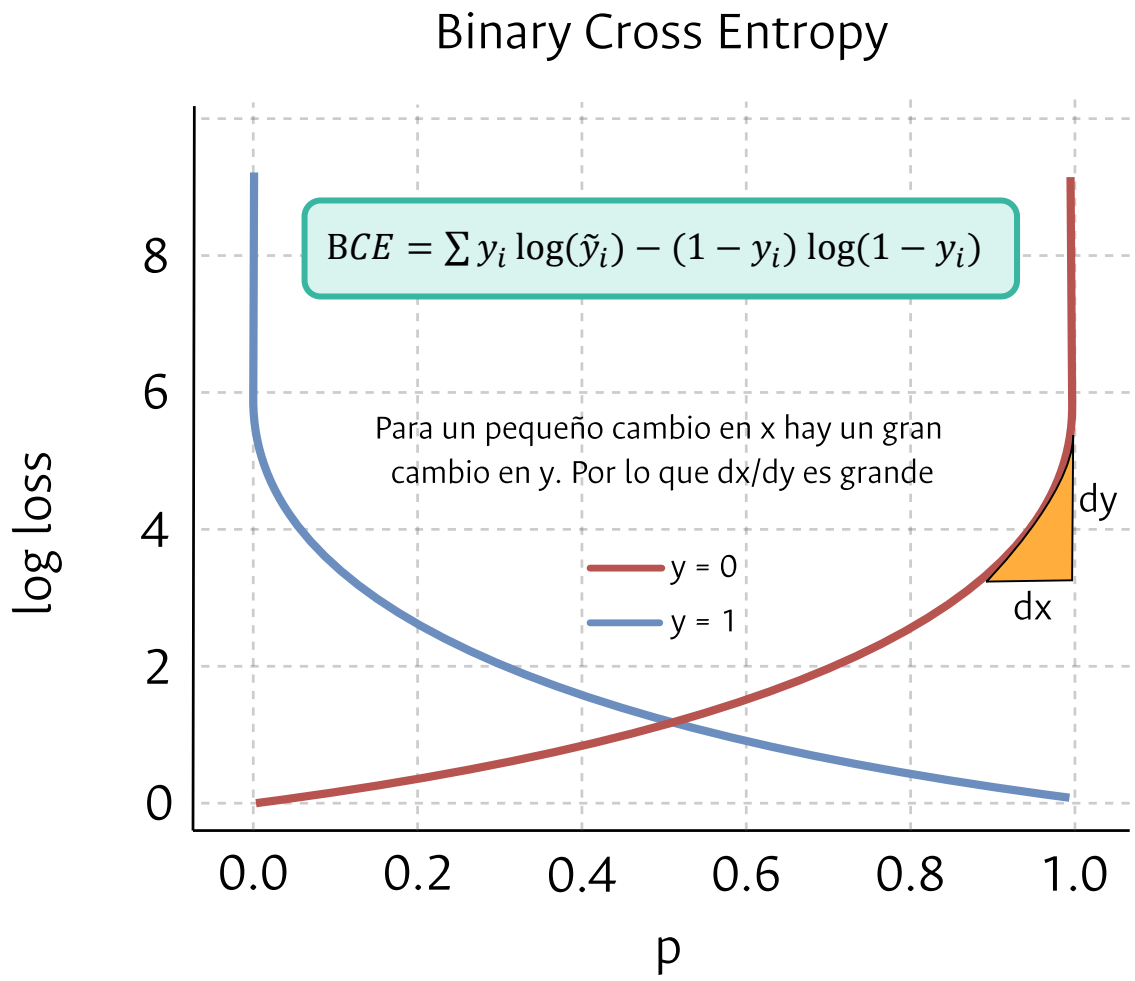
Para mas información puedes dar click en cada uno de los distintos nombres de las funciones de perdida

Flujo de trabajo en redes neuronales

Función de perdida

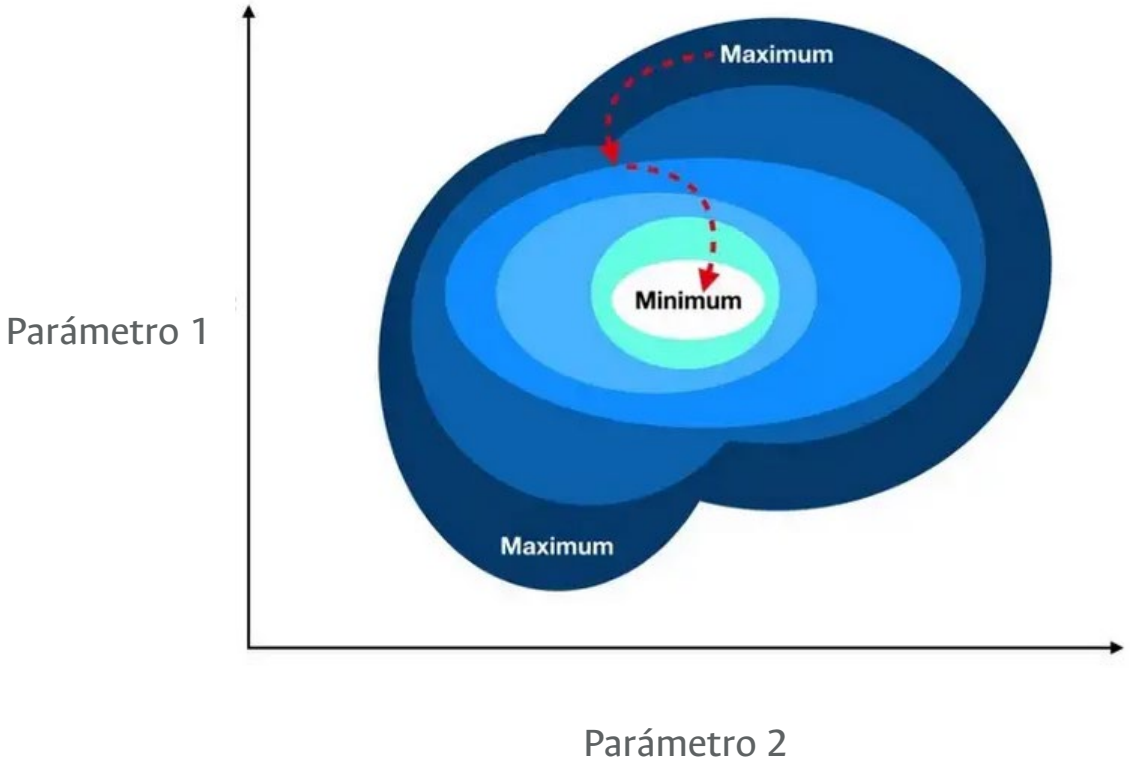
La función de perdida la escogemos dependiendo de la arquitectura y de la aplicación del modelo

- **Clasificación**
  - Log loss
  - Focal Loss
  - KL Divergence / Relative Entropy
  - Exponential Loss
  - Hinge Loss
- **Regresión**
  - Mean Square Error / Quadratic Error
  - Mean Absolute Error
  - Huber Loss / Smooth Mean Absolute Error
  - Log cosh Loss
  - Quartile Loss



Flujo de trabajo en redes neuronales

Optimizadores

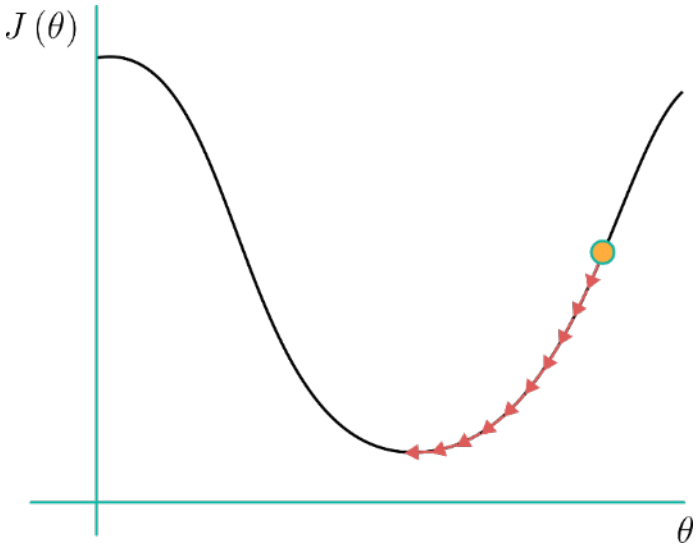


- **Class Adadelta** : Optimizador que implementa el algoritmo Adadelta.
- **Class Adagrad** : Optimizador que implementa el algoritmo Adagrad.
- **Class Adam** : Optimizador que implementa el algoritmo Adam.
- **Class Adamax** : Optimizador que implementa el algoritmo Adamax.
- **Class Ftrl** : Optimizador que implementa el algoritmo FTRL.
- **Class Nadam** : Optimizador que implementa el algoritmo NAdam.
- **Class Optimizer** : Clase base para los optimizadores de Keras.
- **Class RMSprop** : Optimizador que implementa el algoritmo RMSprop.
- **Class SGD** : Optimizador de gradiente descendiente (con momentum).

Optimizadores – Tasa de aprendizaje

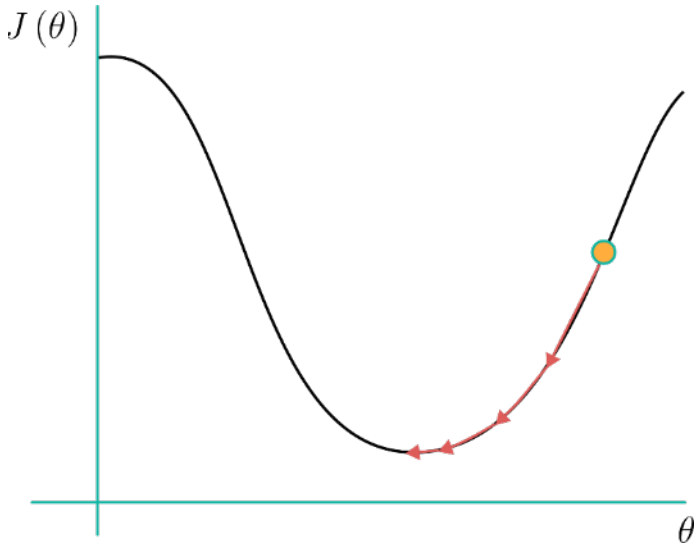
- Una vez escogido el optimizador, también hay que escoger el ***learning rate*** o tasa de aprendizaje, que controla el tamaño de paso en el proceso iterativo.

Muy bajo



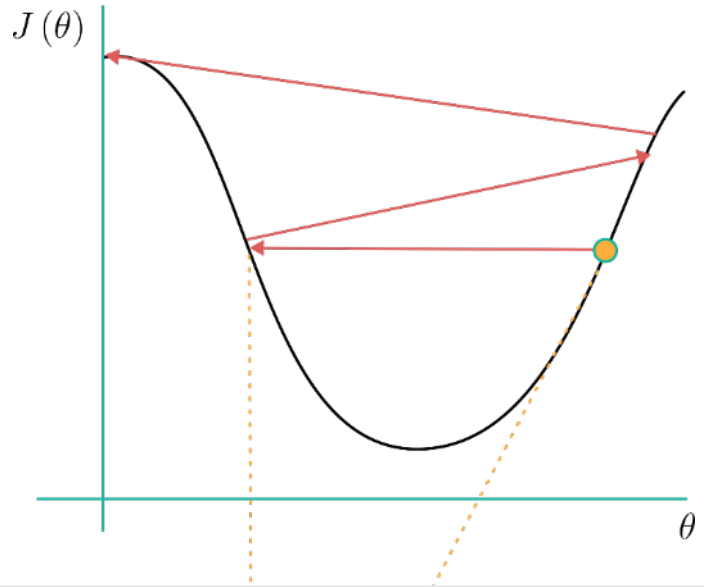
- Una tasa de aprendizaje pequeña requiere muchas actualizaciones antes de alcanzar el punto mínimo

Optimo



- La tasa óptima de aprendizaje alcanza rápidamente el punto mínimo

Muy alta



- Una tasa de aprendizaje demasiado alta provoca actualizaciones drásticas que conducen a comportamientos divergentes

Flujo de trabajo en redes neuronales

Optimizadores – Código

keras requiere un optimizador para entrenar la red neuronal

```
opt = tf.keras.optimizers.Adam(learning_rate=1e-1)
```

Se utilizará un optimizador Adam y se define la tasa de aprendizaje que se hace en cada iteración.

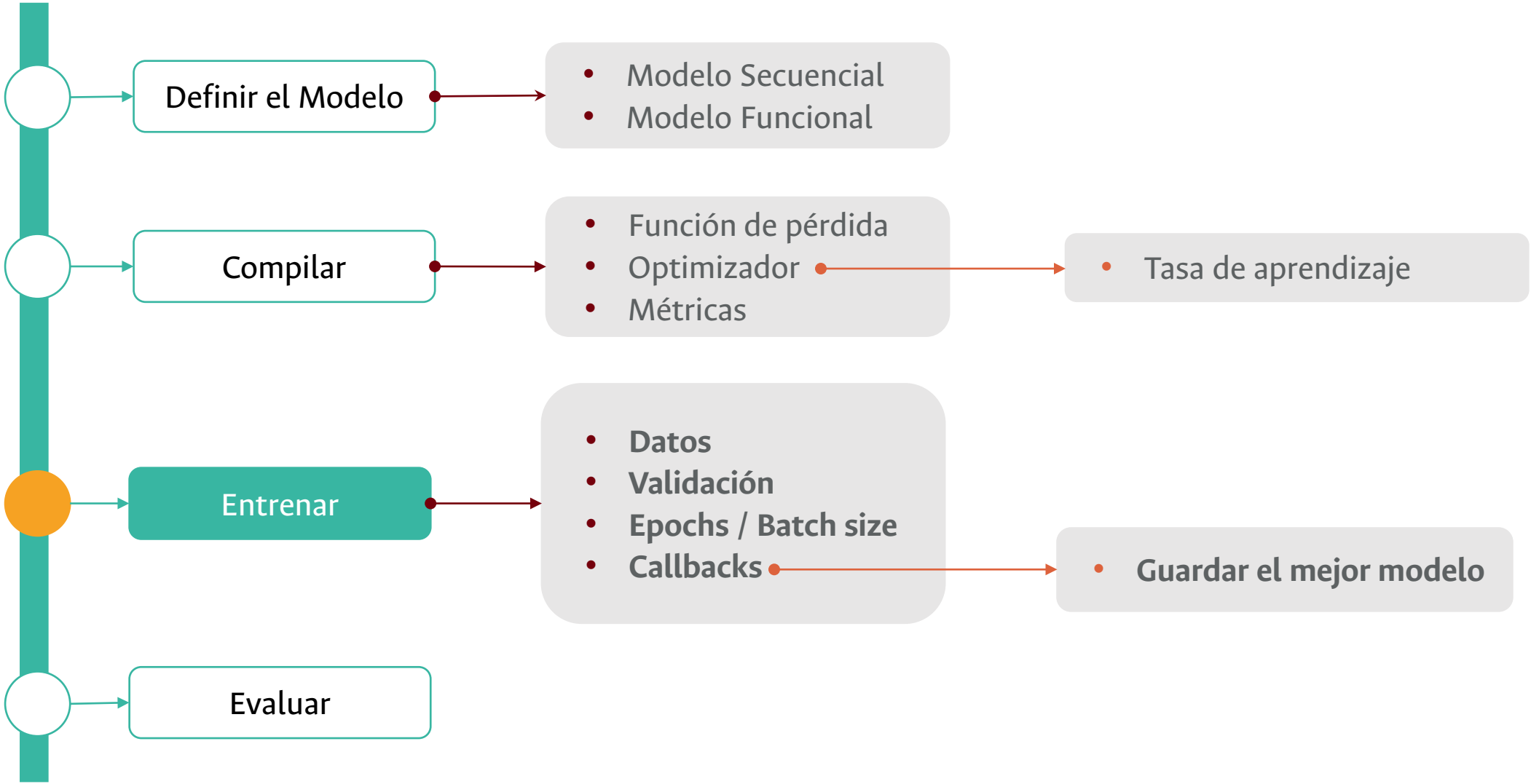
```
model.compile(loss=tf.losses.binary_crossentropy,
              optimizer=opt,
              metrics=['acc'])
```

De este modo, para compilar el modelo con la función *compile()* definimos la función de perdida y el optimizador, que en esta ocasión es *opt*, el cual fue definido anteriormente.

En este modelo se utiliza el ***accuracy*** para medir el desempeño durante el entrenamiento.

Flujo de trabajo en redes neuronales

Flujo de trabajo – Entrenar



Flujo de trabajo en redes neuronales

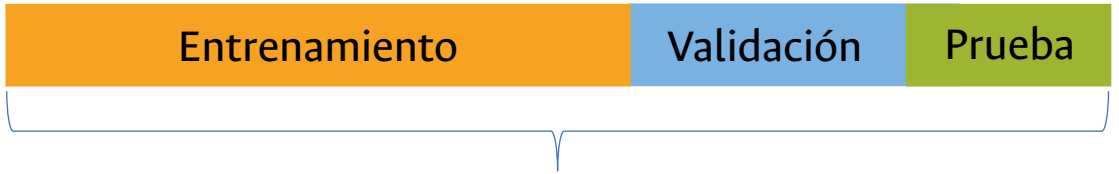
Parámetros para el Entrenamiento

Antes de empezar a entrenar hay que definir:

- ¿Cuántas **epochs** va a durar el entrenamiento?
- ¿Qué tamaño de **batch** usar?
- ¿Qué conjunto de **datos** utilizar?
- ¿Con que datos se **valida** el entrenamiento?

Flujo de trabajo en redes neuronales

Calbacks : Buenas prácticas para el Entrenamiento

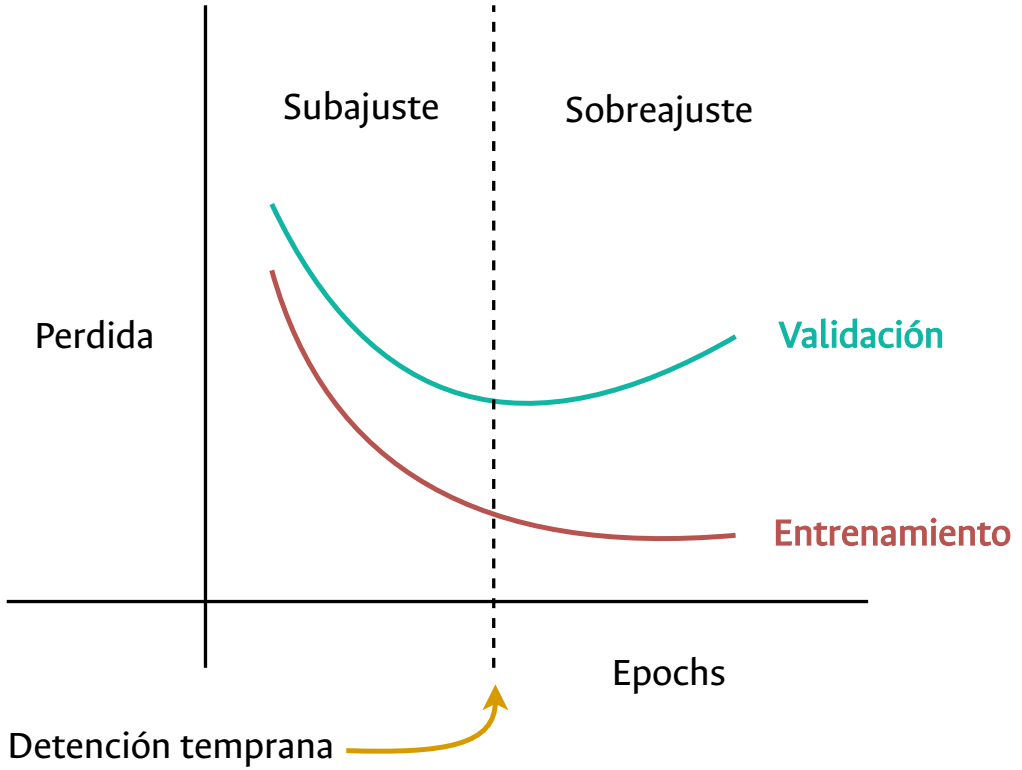


- Conjunto único de datos

Es imposible saber de antemano cuántas epochs van a ser necesarias

Algunos consejos :

- Partir el conjunto de datos en grupos (batch) más pequeños.
- El cálculo de la función de pérdida y las correcciones se hace en cada grupo por separado, de forma secuencial



Flujo de trabajo en redes neuronales

Calbacks : Buenas practicas para el Entrenamiento

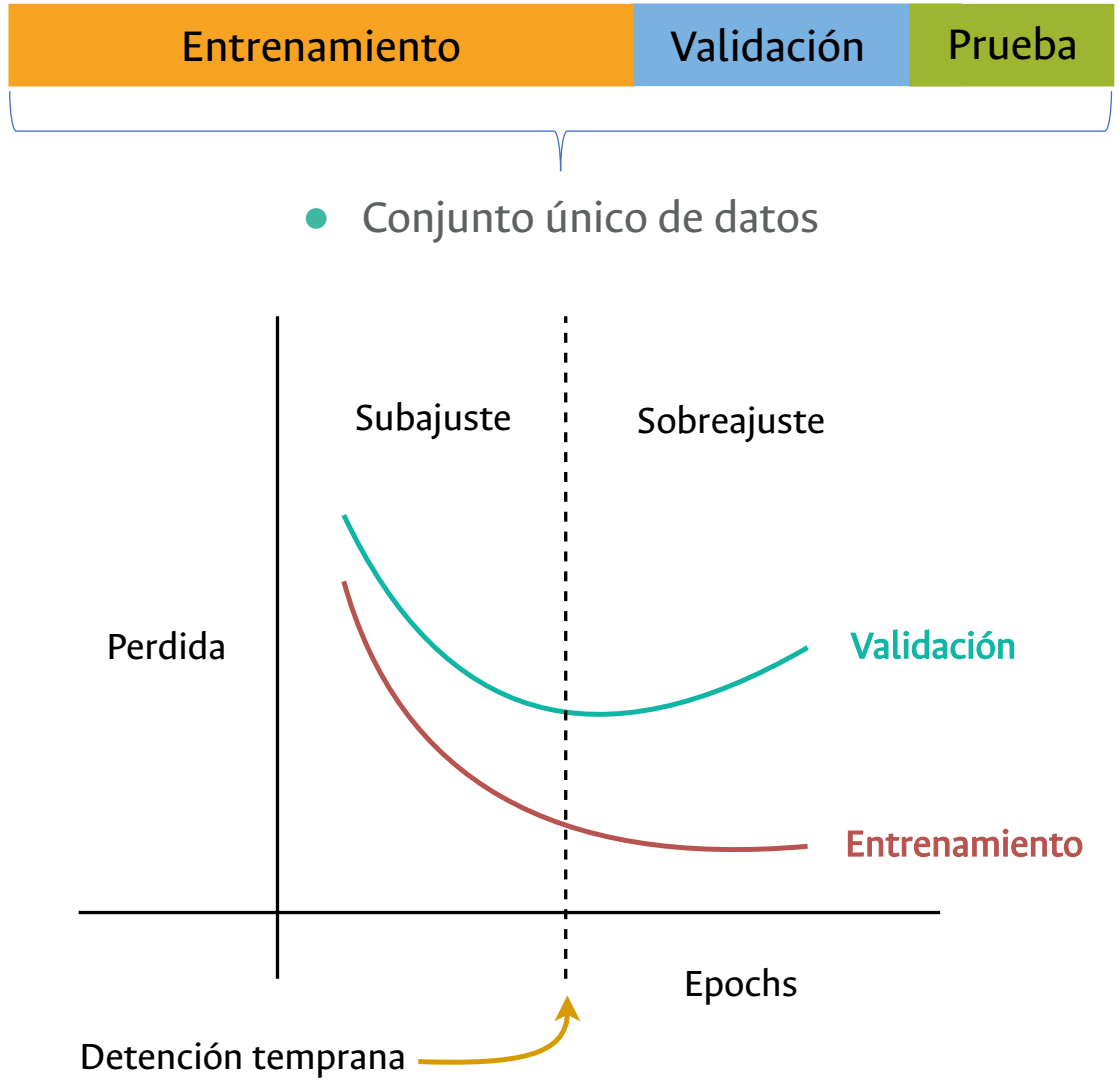
Keras ofrece herramientas: ***Callbacks***

Checkpoint

Permite guardar el modelo que presentó la menor pérdida en el conjunto de validación.

EarlyStopping

Permite detener el entrenamiento si se detecta que en las últimas epochs la función de validación solo va subiendo.



Flujo de trabajo en redes neuronales

Calbacks - Código

Checkpoint

```
checkpoint = tf.keras.callbacks.ModelCheckpoint(
                filepath='best_weights.h5',
                monitor="acc",
                mode="max",
                save_best_only=True,
                save_weights_only=True
            )
```

- Path del archivo donde se guardarán los pesos o el modelo
- acc será la métrica que se va a monitorear
- Se quiere guardar el modelo que reporte el *accuracy* máximo es decir mode = *max*.
- Si se define True, se guarda el mejor modelo
- Si se define True, solo se guardan los pesos, no la arquitectura

EarlyStopping

```
stopping = tf.keras.callbacks.EarlyStopping(
                monitor="acc",
                patience=50,
                mode="max",
                restore_best_weights=True
            )
```

- acc será la métrica que se va a monitorear
- Si después de 50 *epochs* la métrica no mejora, se detiene el entrenamiento.
- Se quiere guardar el modelo que reporte el *accuracy* máximo es decir mode = *max*.
- Si se define como True, automáticamente se cargan al modelo los mejores pesos.

Flujo de trabajo en redes neuronales

Entrenamiento - Código

Para entrenar el modelo utilizamos la función *fit()*

Se puede utilizar como argumento X y Y arreglos de numpy o tensores siempre y cuando se tengan datasets pequeños que quepan en memoria. Su entrenamiento será por batch.

```
model.fit(x=X, y=Y, epochs, batch_size, callbacks)
```

Entrenamiento durante 200 *epochs*

Entrenamiento con un *batch size* de 4

Se utilizan los *callbacks* definidos anteriormente

```
hist = model.fit(x=X,
                 y=y_XOR,
                 epochs=200,
                 batch_size=4,
                 callbacks=[checkpoint, stopping])
```

Flujo de trabajo en redes neuronales

Entrenamiento - Código

Definición del modelo

```
inp_layer = tf.keras.layers.Input(shape=(2,))
int_layer = tf.keras.layers.Dense(units=1,
                                  activation="tanh")
out_layer = tf.keras.layers.Dense(units=1,
                                  activation="sigmoid")
```

```
int_out = int_layer(inp_layer)
y_prime = out_layer(
        tf.concat([inp_layer, int_out], axis=1)
        )
```

```
model = tf.keras.models.Model(
        inputs=[inp_layer],
        outputs=[y_prime]
        )
```

Compilación

```
opt = tf.keras.optimizers.Adam(learning_rate=1e-1)
```

```
model.compile(loss=tf.losses.binary_crossentropy,
              optimizer=opt,
              metrics=['acc'])
```

Entrenamiento

```
checkpoint = tf.keras.callbacks.ModelCheckpoint(
                filepath='best_weights.h5',
                monitor="acc",
                mode="max",
                save_best_only=True,
                save_weights_only=True
            )
```

```
stopping = tf.keras.callbacks.EarlyStopping(
               monitor="acc",
               patience=50,
               mode="max",
               restore_best_weights=True
           )
```

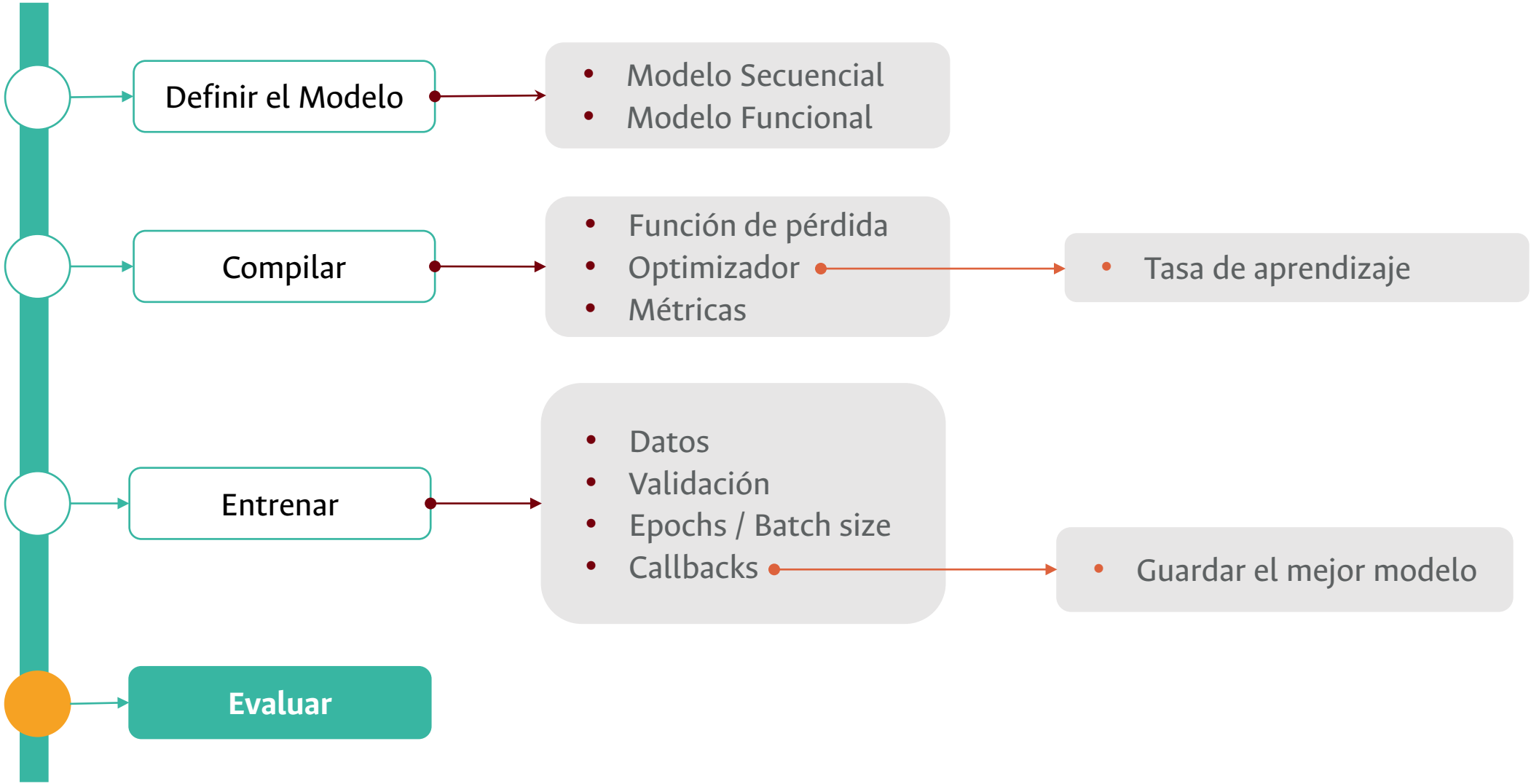
```
model.fit(x=X, y=Y, epochs, batch_size, callbacks)
```

En resumen...

```
hist = model.fit(x=X,
                 y=y_XOR,
                 epochs=200,
                 batch_size=4,
                 callbacks=[checkpoint, stopping])
```

Flujo de trabajo en redes neuronales

Flujo de trabajo – Evaluar



Flujo de trabajo en redes neuronales

Evaluación - Código

1

```
model.evaluate(X_test, y_true)
```

- Devuelve los valores de la función de pérdida y de las métricas calculadas en la partición de prueba.

2

```
y_pred = model.predict(X_test)

accuracy = tf.keras.metrics.Accuracy()
accuracy.update_state(y_true, y_pred)
acc = accuracy.result().numpy()
```

- Devuelve predicciones (*y_pred*) calculadas sobre la partición de prueba.
- Luego se puede usar cualquier métrica de evaluación.

Despedida

¡Gracias por su atención!

UNIVERSIDAD NACIONAL DE COLOMBIA

Fabio Augusto González Osorio, PhD

https://dis.unal.edu.co/~fgonza/

fagonzalezo@unal.edu.co

Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia
Sede Bogotá

Referencias

- 1.17. Neural network models (supervised). (s. f.). scikit-learn. https://scikit-learn.org/stable/modules/neural_networks_supervised.html
- Team, K. (s. f.). Keras documentation: Introduction to Keras for Engineers. https://keras.io/getting_started/intro_to_keras_for_engineers/
- Briega, L. R. E. (2017, 5 junio). Raul E. Lopez Briega - Redes Neuronales. https://relopezbriega.github.io/tag/redes-neuronales.html
- Tanner, G. (2021, 7 diciembre). Introduction to Deep Learning with Keras - Towards Data Science. Medium. https://towardsdatascience.com/introduction-to-deep-learning-with-keras-17c09e4f0eb2
- Carter, D. S. A. S. (s. f.). Tensorflow — Neural Network Playground. https://playground.tensorflow.org/

Derechos de Imágenes

- Team, K. (s. f.). Keras: the Python deep learning API. [icono] https://keras.io/img/logo.png
- Flaticon. (s.f.). Neural Network free icon. [Icono]. https://www.flaticon.com/free-icon/neural-network_6551651
- Flaticon. (s.f.). Neuron free icon. [Icono]. https://www.flaticon.com/free-icon/neuron_4860630
- Flaticon. (s.f.). Functions free icon. [Icono]. https://www.flaticon.com/free-icon/functions_2959069
- Flaticon. (s.f.). Artificial Intelligence free icon. [Icono]. https://www.flaticon.com/free-icon/artificial-intelligence_910490
- Flaticon. (s.f.). Artificial Intelligence free icon. [Icono]. https://www.flaticon.com/free-icon/artificial-intelligence_910513
- Flaticon. (s.f.). Parabolic free icon. [Icono]. https://www.flaticon.com/free-icon/parabolic_227936
- Flaticon. (s.f.). Predictive Models free icon. [Icono]. https://www.flaticon.com/free-icon/predictive-models_2103652
- colaboradores de Wikipedia. (2008, 6 agosto). Archivo:Python-logo-notext.svg - Wikipedia, la enciclopedia libre.[imagen] https://upload.wikimedia.org/wikipedia/commons/c/c3/Python-logo-notext.svg
- FranÃ§ois Chollet â. (s. f.). Medium. [Imagen]. https://miro.medium.com/max/2400/1*1gexYTX55RNk-mezNIe3uQ.jpeg
- Flaticon. (s.f.). Layers free icon. [Icono]. https://www.flaticon.com/free-icon/layers_802033
- Nadeem Qazi. The window incorporates the depth, but it only moves along two dimensions of the image [GIF]. https://miro.medium.com/max/738/1*Q7NXeOlDkm4xlNrNQOS67g.gif
- Bouvet. A Pooling example with a stride of 2 and a filter size of 2x2 [GIF]. https://www.bouvet.no/bouvet-deler/understanding-convolutional-neural-networks-part-1/_/attachment/inline/e60e56a6-8bcd-4b61-880d-7c621e2cb1d5:6595a68471ed37621734130ca2cb7997a1502a2b/Pooling.gif
- Flaticon. (s.f.). Neural Network free icon. [Icono]. https://www.flaticon.com/free-icon/neural-network_6969098
- Flaticon. (s.f.). Workout free icon. [Icono]. https://www.flaticon.com/free-icon/workout_2117237

Derechos de Imágenes

- Flaticon. Compiler free icon[PNG]. https://www.flaticon.com/free-icon/compiler_6461557
- Flaticon. Document free icon [PNG]. https://www.flaticon.com/free-icon/document_2051947
- Flaticon. Question free icon[PNG]. https://www.flaticon.com/free-icon/question_3748253
- Flaticon. Blocks free icon [PNG]. https://www.flaticon.com/free-icon/lego_302607
- Flaticon. Blocks free icon [PNG]. https://www.flaticon.com/free-icon/blocks_2466955
- Flaticon. Push free icon[PNG]. https://www.flaticon.com/free-icon/push_3062462

Créditos

Facultad de
INGENIERÍA

Profesor
Fabio Augusto González Osorio, PhD

Asistente docente
Santiago Toledo Cortés, PhD (C)

Coordinador de virtualización
Edder Hernández Forero, Ing

Diagramador PPT
Mario Andres Rodriguez Triana

Diseño gráfico
Clara Valeria Suárez Caballero
Milton R. Pachón Pinzón

2023