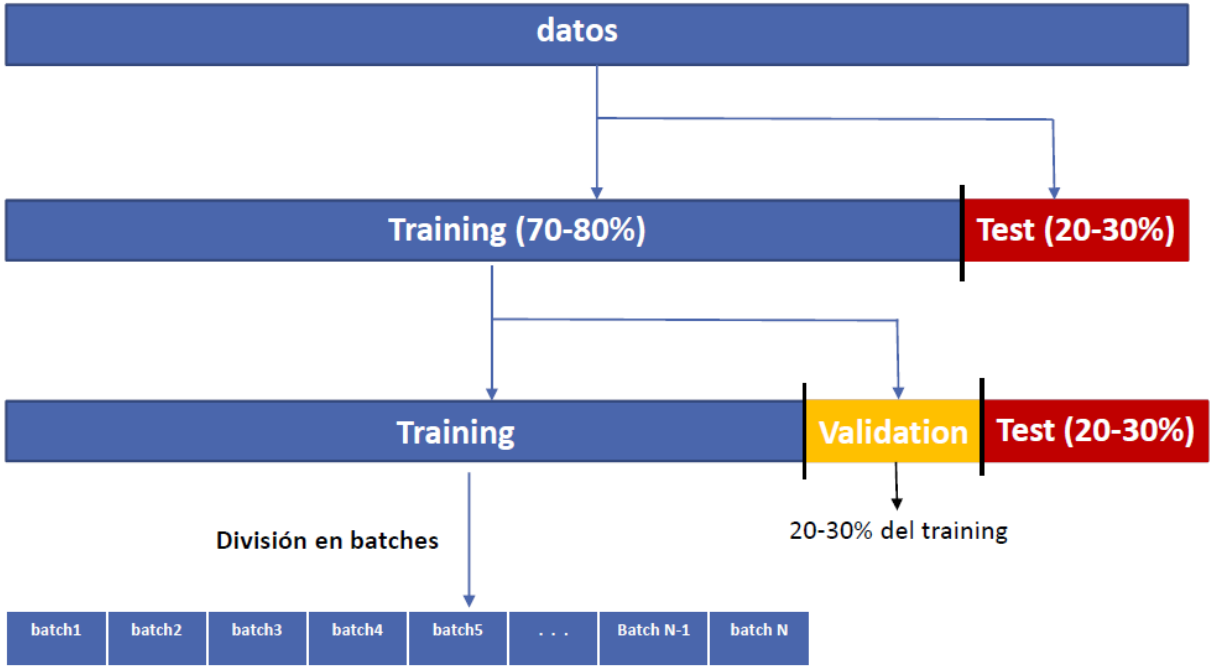
Deep Learning

Sesión 3 - Resumen

De un vistazo

- Partición de datos
- Librería keras
- Descenso del Gradiente: GD, SGD, Mini-batch SGD
- Learning rate
- Batch size
- Funciones de pérdidas
- Funciones de activación
- Inicialización de los pesos (parámetros de las redes)
- Consejos

División de datos



Librería keras

Después de trabajar durante estas sesiones con la librería de bajo nivel denominada TensorFlow, en el presente módulo se va a introducir un framework de alto nivel para el entrenamiento de redes neuronales denominado **Keras**. Esta librería fue desarrollada por **François Chollet** en 2015 con el objetivo de **simplificar la programación de algoritmos basados en aprendizaje profundo** ofreciendo un conjunto de abstracciones más intuitivas y de alto nivel. Keras hace uso de librerías de más bajo nivel o ***backend*** por detrás, concretamente se puede escoger entre **TensorFlow, Microsoft Cognitive Toolkit o Theano**. Durante las siguientes sesiones haremos uso de la librería Keras con TensorFlow como backend.

Cabe destacar que en keras, existen dos formas de generar la arquitectura de un modelo:

- **Modo (o API) secuencial**: Se instancia un objeto del tipo Model y a este se le van añadiendo las capas que conforman la arquitectura una detrás de otra.
- **Modo (o API) funcional**: Se define una entrada y a partir de las mismas se va definiendo la arquitectura indicando cuál es la entrada a cada capa) Una vez definida la arquitectura se crea el objeto modelo pasándole las entradas y las salidas (última capa definida).

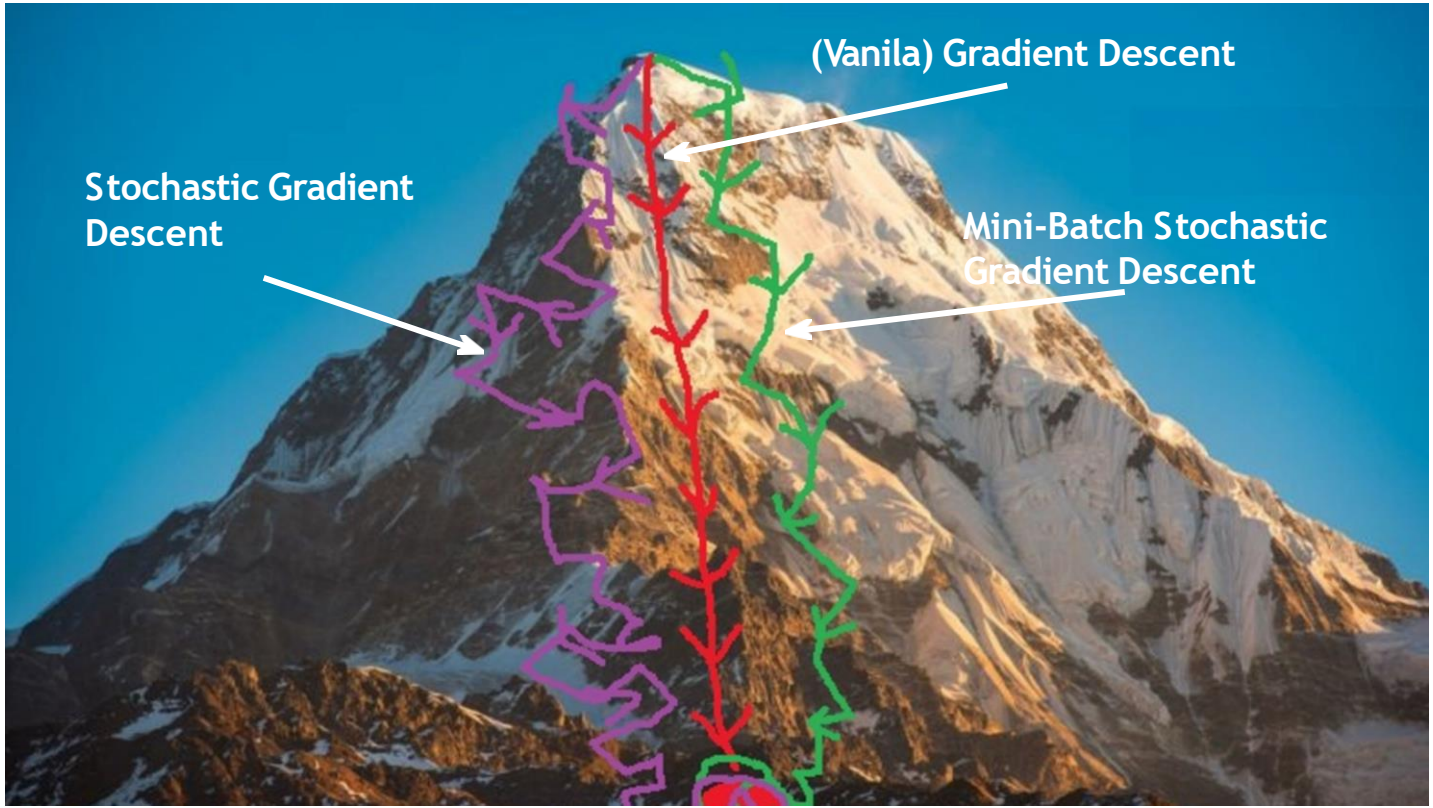
Librería Keras: Secuencial

```
15 # Implementamos la red empleando Keras
16 model = Sequential() # Instancia de modelo API secuencial #(X)
17 model.add(Flatten()) # Estiramos los datos en forma de vector como entrada a nuestro Perceptrón Simple #(X)
18 model.add(Dense(10, input_shape=(784,), activation="softmax")) # Construimos nuestro Perceptrón simple con una única capa Dense #(X)
19
20 # Compilamos y entrenamos el modelo SGD
21 print("[INFO]: Entrenando red neuronal...")
22 model.compile(loss="categorical_crossentropy", optimizer=SGD(lr), metrics=["accuracy"])  #(X)
23 H = model.fit(x_tr, y_tr, validation_data=(x_val, y_val), epochs=n_epochs, batch_size=batch_size)  #(X)
24
25 # Evaluando el modelo de predicción con las imágenes de test
26 print("[INFO]: Evaluando red neuronal...")
27 predictions = model.predict(x_te, batch_size=batch_size)  #(X)
28 print(classification_report(y_te.argmax(axis=1), predictions.argmax(axis=1)))
29
```

Librería Keras: Funcional

```
17 # Implementamos la red empleando Keras (modelo funcional)
18 inputs= Input (shape=(28,28))
19 x=Flatten(input_shape=(784,))(inputs)
20 x= Dense(10, activation= "softmax") (x)
21 model = Model(inputs=inputs, outputs = x)
22 # Compilamos y entrenamos el modelo SGD
23 print("[INFO]: Entrenando red neuronal...")
24 model.compile(loss="categorical_crossentropy", optimizer=SGD(lr), metrics= ["accuracy"])
25 H=model.fit (x_tr, y_tr, validation_data=(x_val, y_val), epochs=n_epochs, batch_size=batch_size)
26
27 # Evaluando el modelo de predicción con las imágenes de test
28 print("[INFO]: Evaluando red neuronal...")
29 prediction=model.predict(x_te, batch_size=batch_size)
30 print(classification_report(y_te.argmax(axis=1), predictions.argmax(axis=1)))
```

Descenso del Gradiente: GD, SGD, Mini-batch SGD



Learning rate

El learning rate es la “velocidad” con la que queremos actualizar los pesos (parámetros) de nuestro modelo (red neuronal)

Debe ser el adecuado, sino, problemas!

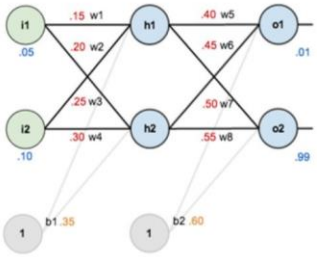
Íntimamente relacionado con el batch size

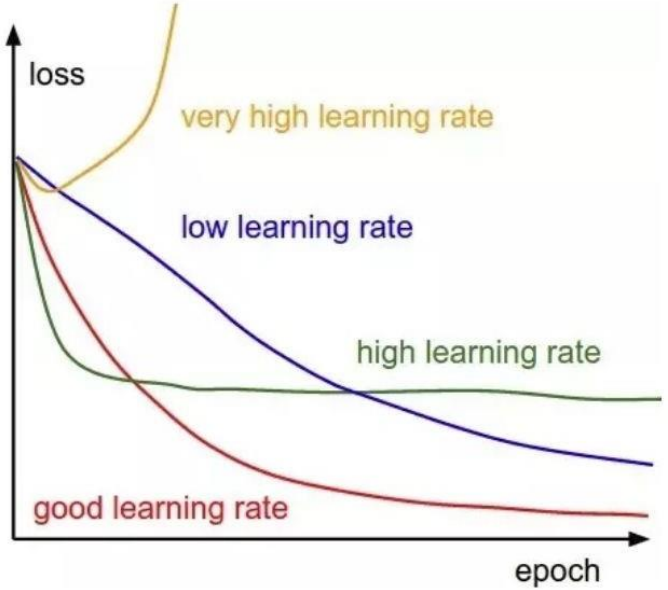
$$w_1^+ = w_1 - \eta \frac{\partial E_{total}}{\partial w_1} \qquad w_5^+ = w_5 - \eta \frac{\partial E_{total}}{\partial w_5}$$

$$w_2^+ = w_2 - \eta \frac{\partial E_{total}}{\partial w_2} \qquad w_6^+ = w_6 - \eta \frac{\partial E_{total}}{\partial w_6}$$

$$w_3^+ = w_3 - \eta \frac{\partial E_{total}}{\partial w_3} \qquad w_7^+ = w_7 - \eta \frac{\partial E_{total}}{\partial w_7}$$

$$w_4^+ = w_4 - \eta \frac{\partial E_{total}}{\partial w_4} \qquad w_8^+ = w_8 - \eta \frac{\partial E_{total}}{\partial w_8}$$





Batch size

Número de instancias que le introducimos a la red en cada iteración para que realice el forward y el backward pass (calcula predicciones, errores, gradientes, y modifica los pesos de acuerdo a esos gradientes)

Íntimamente ligado al learning rate

Lo mejor es que sea lo suficientemente grande para aprovechar las capacidades de nuestra GPU y para que represente la distribución de datos (clases) de nuestro dataset lo más fielmente posible

Se suelen emplear potencias de 2: 4, 8, 16, 32, 64...

Funciones de pérdidas

Calculan el error que comete nuestro modelo al hacer una predicción.

Existen muchísimas, pero las más comunes son:

- Problemas de **regresión**
  - Mean Squared Error
  - Mean Absolute Error
- Problemas de **clasificación**
  - Binary Cross-Entropy
  - Categorical Cross-Entropy

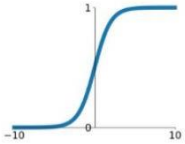
Funciones de activación

Introducen las **no-linealidades** necesarias para que nuestro modelo sea capaz de mapear (trasladar) los datos de un espacio muy complicado y enredado a otro más sencillo de abordar

Basado en la biología humana: nuestro cerebro funciona de forma similar

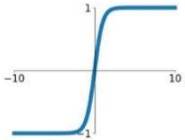
Sigmoid

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



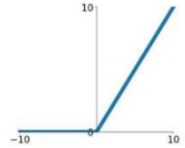
tanh

$$\tanh(x)$$



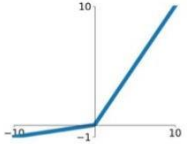
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

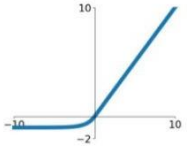


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Funciones de activación

Para capas intermedias (ocultas), utilizar siempre de tipo ReLU (ReLU, Leaky ReLU, parametric ReLU, ect)

Para capas finales:

- En **clasificación**:
  - **Softmax** cuando trabajemos con problemas multiclase en los que nuestra instancia solo puede pertenecer a una de las clases
  - Sigmoide cuando trabajemos en un problema binario (aquí la softmax puede dar mejores resultados) o cuando trabajemos con problemas multiclase en los que nuestra instancia puede pertenecer a **varias** de las clases
- En **regresión**: función de activación **lineal**, es decir, **no tiene**

Inicialización de los pesos

Nuestra red, en el instante 0, no tiene los parámetros (pesos) ajustados para dar una buena predicción.

Se pueden inicializar de muchas maneras distintas:

- A unos
- A ceros
- Utilizando distribuciones de probabilidad (normal, uniforme, normal truncada…)
- Utilizando métodos avanzados (LeCun, He, Glorot, etc)

Por lo general, una inicialización **glorot uniforme** suele dar buenos resultados.

Consejos

El batch size y el learning rate van íntimamente ligados, cuidado al elegir los valores!

La función de activación y la inicialización de los pesos van íntimamente ligadas, cuidado al elegirlas o puede que vuestra red no comience a entrenar nunca

Por lo general, mejor utilizar modelos pre-definidos con las inicializaciones y funciones de activación por defecto ;)

Aún así, se aprende mucho haciendo pruebas y jugando ;)