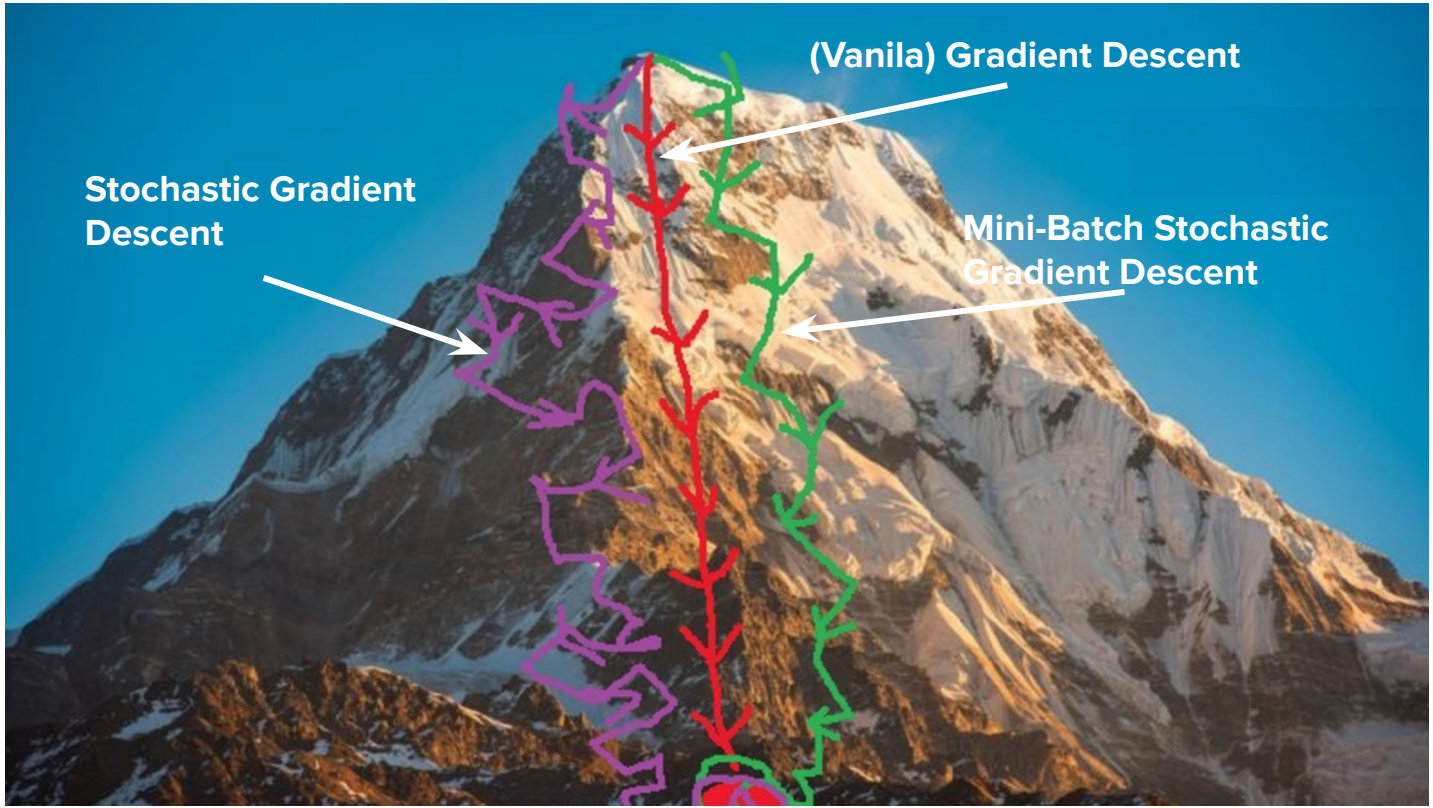
Deep Learning

Sesión 3 - Resumen

De un vistazo

- 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

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!

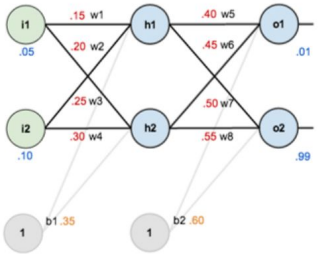
Íntimamente relacionado con el batch size

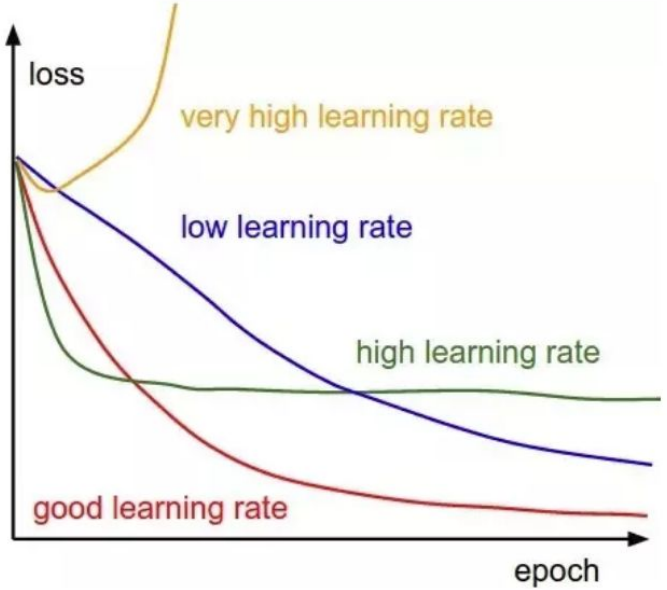
$$w_1^+ = w_1 - \eta \frac{\partial E_{total}}{\partial w_1} \qquad w_5^+ = w_5 - \eta \frac{\partial E_{total}}{\partial w_5}$$

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

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

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





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

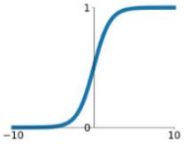
Funciones de activación

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

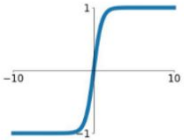
Sigmoid

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



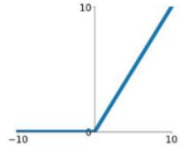
tanh

$$\tanh(x)$$



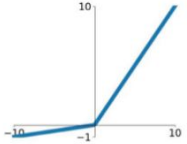
ReLU

$$\max(0, x)$$



Leaky ReLU

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

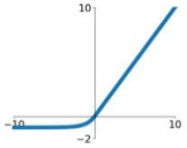


Maxout

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

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 ;)