**Introducción al aprendizaje profundo**

Esta guía es optativa, no se evalúa, pero es importante porque es lo más usado en inteligencia artificial.

Vamos a usar la librería **PyTorch**, pero hay otras como Tensorflow y demás.

Tienen implementaciones que se pueden correr en PC, celulares, incluso algunas en el navegador.

Todo lo que vemos esta subido en la carpeta de la guía 5.

También podemos ver y probar todo lo que tiene PyTorch, se puede probar en la máquina o en la nube.

La función que se usa es la tangente hiperbólica, que va de -1 a 1, es lo mismo que la sigmoidea, pero hay veces que le llaman “sigmoidea” a la sigmoide que va de 0 a 1, así que no confundir, por eso usa la tanh.

En el MLP si yo cambio la función de activación no lineal, debería cambiar todo lo que use la derivada de la función de activación, porque usamos solo para la sigmoidea.

Afortunadamente, frameworks como PyTorch implementan el gradiente automático (auto grad).

Vamos a ir armando un conjunto de operaciones a realizar, que toman ciertas entradas y generan una salida. Yo podría elegir distintas funciones de activación en cada capa o incluso distintas en cada neurona.

El framework ya trae implementadas un montón de funciones distintas que ya tienen un método para la propagación hacia delante y otro método para la propagación hacia atrás.

-Un tensor es un objeto que guarda cierta información (no recuerdo qué) y que además son derivables.

Para nosotros simplemente son matrices y vectores. Y usamos por ejemplo FloatTensor porque son de 32 bits, pero se puede cambiar.

-Al armar una arquitectura simplemente debemos ir indicando como “bloques” por dónde irá pasando la información, por ejemplo, una capa lineal que recibe 3 entradas y genera 2 salidas (Linear(3,2)), una función de activación sigmoidea por la que van a pasar esas salidas, luego otra capa y así todo lo que queremos. La ventaja además es que el gradiente se calcula solo, PyTorch ya sabe que derivada debe usar según la función de activación que usamos.

-Para el error al final podemos usar el error cuadrático pero también hay muchas otras opciones.

-Algo interesante que se explicó es la diferencia entre usar un método batch, mini batch o lotes y online.

En el batch le muestro todos los patrones, saca un promedio y se mueve según ese promedio, entonces puede ser muy lento, en mini batch por ejemplo si tengo 100 patrones tomo grupos (batches o lotes) de 10, se los paso, saca un promedio de esos 10 y actualizo o me muevo así con cada grupo, entonces voy a hacer 10 movimientos en lugar de 1, y en el online va pasando patrón por patrón (no recuerdo cuál sería la ventaja o desventaja en ese caso).

-Otra cosa interesante es que cuando hacemos el paso hacia adelante, al pasar un batch estamos pasando un grupo de patrones, va a ser una estructura como una matriz de matrices, entonces se va accediendo a distintos índices como hace Python.

-El método “squeeze” elimina todas las dimensiones que son 1, es como eliminar las dimensiones redundantes. Mientras que “flatten” te concatena todo, genera un vector “plano”.

-La “paciencia” es un término que me indica cuántas épocas voy a esperar para cortar el entrenamiento si mi modelo ya llegó al punto límite de error de validación, ya que a partir de ahí se comenzaría a sobre entrenar el modelo si tiene esa capacidad, entonces por ejemplo, si tengo paciencia = 10, si luego de 10 épocas no mejora corto y me quedo con lo que tenía guardado en ese punto 10 épocas antes que sería el mejor.