# ANEXOS

# ANEXO A

## Gradiente estocástico y 'política de aprendizaje'

El proceso de aprendizaje de una red neuronal pasa por hacer un pase hacia delante ('forward-pass') de todos los datos del set de training, para después ejecutar un pase hacia atrás de los gradientes ('backward-pass') donde se actualizan todos los parámetros de la red en la dirección del gradiente negativo. Este proceso debe repetirse un elevado número de veces hasta un número máximo de iteraciones o hasta que el error de la red disminuya en una iteración por debajo de un valor ('threshold') escogido a priori.

Este proceso puede llegar a ser excesivamente lento si se aplica a redes neuronales con numerosas capas y a datos complejos con un número muy elevado de dimensiones. Esto es justo lo que sucede en nuestro caso. Por ello, se debe recurrir a algoritmos que aceleren este proceso, aunque posean ciertos inconvenientes. Es el caso del gradiente descendiente estocástico: la diferencia con el gradiente descendiente original es que para cada iteración no pasa por todos los datos del dataset, sino que solo lo hace sobre un grupo de ellos cada vez. Esto acelera la convergencia, aunque da algo de inestabilidad al gradiente especialmente cuando se acerca al mínimo. Además, el gradiente estocástico hace que la actualización de los parámetros varíe mucho entre iteraciones cuando se acerca a un mínimo. Para moderar estos efectos negativos, se emplea el impulso o 'momentum', donde la actualización se calcula como el gradiente actual más una fracción entre 0 y 1 de la actualización anterior.

La inestabilidad cerca del mínimo provoca que el error del modelo empiece a oscilar según los datos escogidos para la iteración actual. Por ello, ya no se puede detener el entrenamiento del algoritmo cuando la variación del error disminuya por debajo de un sesgo. En el software Caffe [23] el entrenamiento se detiene cuando llega a un número máximo de iteraciones. Por tanto, se debe observar la evolución del error y decidir si continuar con el entrenamiento a-posteriori.

| | Modelo 1 | Modelo 2 | Modelo 3 |
|---|---|---|---|
| Batchsize | 200 | 200 | 32 |

| $f_{base}$ | *momentum* | $\gamma$ | $\beta$ |
|---|---|---|---|
| 0,01 | 0,9 | 0,0001 | 0,75 |

Valores empleados en el entrenamiento de las RNAs

Un último añadido más es que se permite variar el factor de aprendizaje a lo largo de las iteraciones. En este caso se ha utilizado una política inversa, es decir, el factor de aprendizaje disminuye conforme se realizan las iteraciones. De este modo, la actualización es mayor al principio del proceso cuando se encuentra lejos de cualquier mínimo local. Al final del proceso, cuando la red neuronal se encuentra cerca del mínimo, el factor de aprendizaje

disminuye para aumentar la estabilidad de la convergencia. Esto es muy útil para suavizar las fluctuaciones propias del gradiente descendiente estocástico cuando se encuentra cerca de un mínimo.

El factor de aprendizaje evoluciona según la siguiente fórmula:

$$f.a. = f_{base} \cdot \frac{1}{(1 + \gamma \cdot iter)^{\beta}}$$

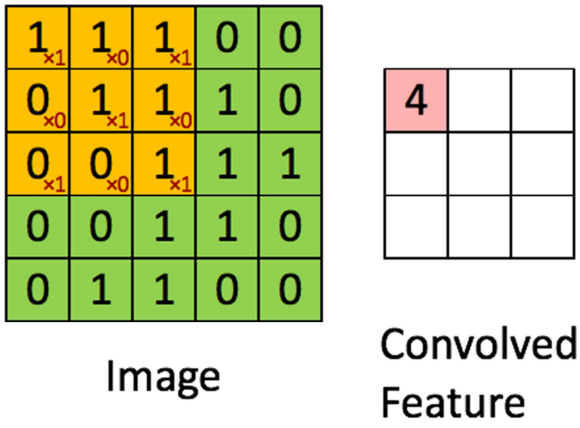
Donde $f_{base}$ es el factor de aprendizaje inicial, *iter* es el número de la iteración actual, y $\beta, \gamma$ son dos parámetros del método a escoger.

# ANEXO B

## Convoluciones y deconvoluciones

El operador convolución aplica, dadas dos matrices (la imagen y el filtro), las cuales pueden ser de tamaños distintos, el producto escalar de Frobenius (sumatorio del producto elemento a elemento) entre la matriz de menor tamaño (el filtro) y una submatriz de la de mayor tamaño (imagen), de forma que el orden de esta submatriz sea el mismo que el del filtro. Teóricamente se ha de invertir el filtro para que se trate de una auténtica convolución, pero por claridad especialmente en la visualización, esto no se realiza en esta área en particular. De este modo se obtiene un escalar para cada combinación de filtro y ventana de la imagen. Al igual que en el operador de convolución continuo, se produce un desplazamiento relativo entre los dos elementos: el filtro se mueve a lo largo y ancho de la imagen (se está operando en 2D), produciendo para cada posición un escalar distinto ya que la ventana de la imagen cambia. Dichos escalares se almacenan en una nueva matriz donde conservan el orden espacial en el que se han calculado, de modo similar a la convolución unidimensional pero trabajando en las dos dimensiones del problema.

En realidad, las imágenes RGB no son tensores de dos dimensiones, sino de tres (altura, anchura, canal). Por ello, la cuenta explicada anteriormente es en realidad algo más complicada. Lo que se hace en realidad es el mismo proceso, pero aplicando un filtro de a su vez tres dimensiones. De este modo, el cálculo es exactamente el mismo sólo que con el triple de elementos.




Esquema del modo de funcionamiento de la capa convolucional en imágenes. [8]

Por tanto, dada una imagen y un filtro, al aplicar la conexión convolucional se obtiene una matriz donde se almacena espacialmente los distintos escalares obtenidos por el 'inner product' (Frobenius) del filtro por la ventana local de la imagen. Es decir, se obtiene una nueva imagen bidimensional, solo que en vez de colores contiene características obtenidas espacialmente por el filtro. Señalar que dichas características son todas obtenidas por el mismo filtro, pero moviéndolo a lo largo y ancho de la imagen, de modo similar a un escáner.

Para aumentar el poder de representación de la capa, se pueden emplear tantos filtros distintos como se quiera: se almacenan los resultados de cada filtro a lo largo de una tercera dimensión, obteniendo de nuevo un tensor de tres dimensiones. Cada una de las matrices pertenecientes al tensor se corresponde al 'escaneo' obtenido a partir de la imagen y del filtro correspondiente. Además, al tratarse de nuevo de un tensor de tres dimensiones, implica que es posible volver a aplicar una capa convolucional, permitiendo construir redes convolucionales profundas.

Como se ha indicado brevemente, para el entrenamiento de un autoencoder basado en convoluciones se necesita de una operación que deshaga el efecto de la convolución. Esta es la deconvolución, la cual es una capa con una difusión mucho menor que la primera. En la convolución, una ventana quedaba reducida a un escalar a través del producto de Frobenius; entonces ahora se busca obtener la ventana (matriz) a partir de un escalar. Ello se obtiene directamente con el producto de un escalar por una matriz o filtro, que es lo que aplica la deconvolución: sustituye el producto de Frobenius por un producto escalar. El resto es idéntico a las capas convolucionales, donde el filtro se mueve a lo largo de la matriz o imagen para almacenar sus resultados en otra matriz de forma ordenada.

# ANEXO C

## RMSE (error de la raíz cuadrada de la media) lineal

Esta métrica es empleada comúnmente por la comunidad científica para comparar resultados en el problema de reconstrucción de profundidad a partir de una sola imagen.

$$RMSE\ (lineal) = \frac{1}{|N|}\sum_{i=1}^{N}\sqrt{\frac{1}{T}\|{x^1}_i - {x^2}_i\|^2}$$

Donde $N$ es el número de datos o imágenes; $T$ es el número de píxeles por imagen; ${x^1}_i$ es la matriz de profundidad verdadera del ejemplo $i$; y ${x^2}_i$ es la matriz de profundidad computada por la red neuronal del ejemplo $i$.