Introducción a las redes neuronales: Red neuronal de alimentación progresiva

M. Sc. Saúl Calderón Ramírez
Instituto Tecnológico de Costa Rica,
Escuela de Computación, bachillerato en Ingeniería en Computación,
PAttern Recongition and MAchine Learning Group (PARMA-Group)

January 31, 2018

Abstract

Este material está basado en el capítulo de redes neuronales, del libro *Pattern recognition and Machine Learning* de Cristopher Bishop[**?**].

1 Introducción

Ya hemos discutido sobre el uso de modelos lineales compuestos por combinaciones lineales de funciones base fijas. Un enfoque alternativo es utilizar un número definido de funciones básicas con parámetros personalizables, permitiendo a tales funcionales cambiar tales parámetros durante el entrenamiento. El modelo más exitoso para este tipo de contexto es la red de alimentación hacia adelante o red *feed-forward* o más comúnmente conocida como el perceptrón multi-capa.

El término de red neuronal tiene sus orígenes en los intintos de encontrar representaciones matemáticas para el procesamiento de información en los sistemas biológicos (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986)

1.1 Funciones de alimentación hacia adelante o alimentación progresiva

Para los modelos de regresión y clasificación vistos anteriormente, ya se analizó que están constituidos por combinaciones lineales de funciones base no lineales *fijas*, es decir, sin parámetros que alteren su forma, dadas por ϕ_j, con $\vec{x} \in \mathbb{R}^D$ el vector de características a clasificar, $\vec{w}$ el vector de pesos para la combinación lineal de las funciones base ϕ_j. Para plantear con una sola expresión

el problema de la regresión y clasificación podemos escribir:

$$y\left(\vec{x},\vec{w}\right)=f\left(\sum_{j=0}^{M}w_j\phi_j\left(\vec{x}\right)\right) \tag{1}$$

donde en el caso de la regresión $f\left(u\right)$ corresponde a la función identidad $f\left(u\right)=u$ y para el caso de la clasificación $f\left(u\right)$ es una función no lineal, por ejemplo la función escalón $f\left(u\right)=\begin{cases}1 & u>0\\ 0 & u\leq 0\end{cases}$. En general, a la función f se le denomina **función de activación**. Recordemos que tanto en el problema de la regresión como la clasificación, es necesario determinar el arreglo de pesos $\vec{w}\in\mathbb{R}^M$.

Una extensión que incrementa la flexibilidad del modelo es hacer que las funciones base ϕ_j presenten parámetros que les permitan ser no fijas o adaptables, por lo que las funciones bases vendrían dadas por $\phi_j\left(\vec{x},\vec{\theta}\right)$ con $\vec{\theta}$ el arreglo de parámetros ajustables en entrenamiento, lo que hace necesario encontrar los arreglos $\vec{\theta}$ y $\vec{w}$ durante el entrenamiento.

Existen muchas formas the construir esas funciones paramétricas $\phi_j\left(\vec{x},\vec{\theta}\right)$. Las redes neuronales usan como funciones base, funciones de forma similares a lo planteado en la función 1, por lo que una función base ϕ_j tiene la forma:

$$\phi_j\left(\vec{x},\vec{\theta}\right)=h\left(\sum_{i=0}^{R}\theta_i x_i\right),$$

por lo que:

$$y\left(\vec{x},\vec{w}\right)=f\left(\sum_{j=0}^{M}w_j\,h\left(\sum_{i=0}^{R}\theta_i x_i\right)\right) \tag{2}$$

donde $h\left(u\right)$ es una función no lineal y su entrada u viene dada por una combinación lineal de las entradas con los pesos $\vec{\theta}$. Veamos más detalladamente como están dadas tales funciones base. Observe de izquierda a derecha la Figura 1.

La red es escencialmente un grafo, usualmente de tres capas:

- La capa de D nodos de entrada, constituidos por los valores $x_1,x_2,\ldots,x_D$ del arreglo de entrada $\vec{x}\in\mathbb{R}^D$,
- La capa de M nodos $y_0^o,y_1^o,y_2^o,\ldots,y_M^o$, en notación vectorial dada por $\overrightarrow{y^o}\in\mathbb{R}^M$, llamada capa oculta, la cual controla el nivel de generalización de la red (suavidad de la superficie de decisión).
- La capa de salida, constituida por K nodos $y_1^s,\ldots,y_K^s$, con $\overrightarrow{y^s}\in\mathbb{R}^K$, lo cual corresponde al **número K de clases por discriminar**, usando una notación de la salida de $1-K$.

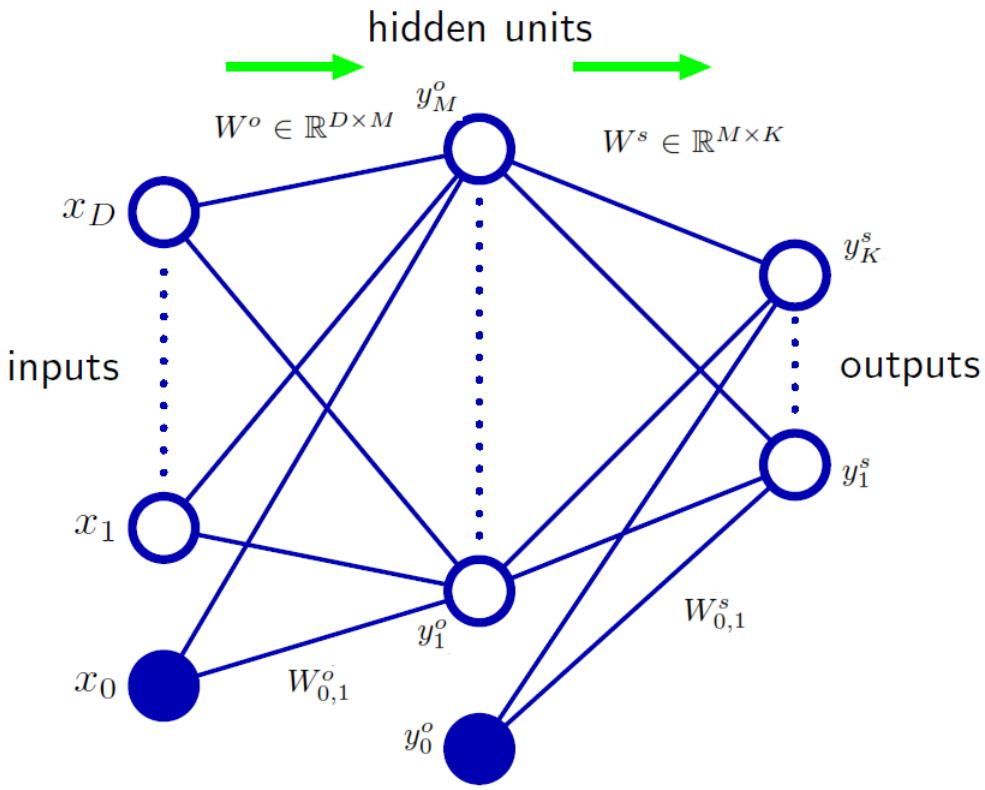

Figure 1: Diagrama de una red neuronal de dos capas (usualmente la capa de entrada no se incluye).

1.1.1 La capa oculta

El grafo está definido por dos matrices de pesos $W^o \in \mathbb{R}^{D\times M}$ y $W^s \in \mathbb{R}^{M\times K}$ (se usará una notación en mayúscula para implicar el uso de una matriz, la cual sería simétrica, aunque también se puede suponer que W es un vector con dos subindices), La primer matriz de pesos conecta a la capa de entrada con la capa oculta, y la segunda matriz conecta la capa oculta con la capa de salida.

Para cada nodo j en la capa oculta, se define el **peso neto o coeficiente de activación** p_m^o el cual está dado por la combinación lineal de los valores en los nodos de entrada:

$$p_m^o\left(\vec{x}, W^o\right) = \sum_{d=1}^{D} W_{d,m}^o x_d + W_{0,m}^o, \tag{3}$$

donde el peso $W_{0,m}^o$ comúnmente se refiere como sesgo, por lo que para expresar al peso neto de la capa oculta p_m^o como únicamente una combinación lineal sin sesgo o desplazamiento, se reescribe:

$$p_m^o\left(\vec{x}, W^o\right) = \sum_{d=0}^{D} W_{d,m}^o x_d$$

se fija a $x_0 = 1$. El peso neto p_m de la unidad o neurona m es transformado usando una **función de activación** no lineal y diferenciable en la capa oculta $g^o(\cdot)$ para resultar en:

$$y_m^o\left(\vec{x}, W^o\right) = g^o\left(p_m^o\left(\vec{x}, W^o\right)\right), \tag{4}$$

con lo cual se puede observar que la función y_m^o corresponde a la función base $\phi_m\left(\vec{x}, \vec{\theta}\right)$,

$$\phi_m\left(\vec{x}, \vec{\theta}\right) = h\left(\sum_{j=0}^{M} \theta_j x_j\right),$$

con los parámetros de la función base dados por $\vec{\theta} = W^o$, y con $h = g^o$ y . El valor de los nodos $y_0^o, y_1^o, y_2^o, \ldots, y_M^o$ se les llama las salidas de la capa oculta.

La función de activación $h = g^o$ usualmente se escoge para que sea *suave* (derivable), y denote el estado de una neurona como activada o desactivada antes una entrada específica. Como veremos, la condición de que la función de activación sea derivable, es importante para poder calcular el gradiente y facilitar la minimización del error de clasificación. Las funciones de activación comunmente utilizadas son la tangente hiperbólica o la función sigmoidal, respectivamente dadas por:

$$\begin{array}{cc} \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} & \text{sigmoid}(x) = \frac{1}{1+e^{-x}} \\ \frac{\mathrm{d}}{\mathrm{d}x}\tanh(x) = 1 - \tanh^2(x) & \frac{\mathrm{d}}{\mathrm{d}x}\text{sigmoid}(x) = \frac{e^{-x}}{(1+e^{-x})^2} \end{array}$$

Sus gráficas se muestran en la Figura 2.

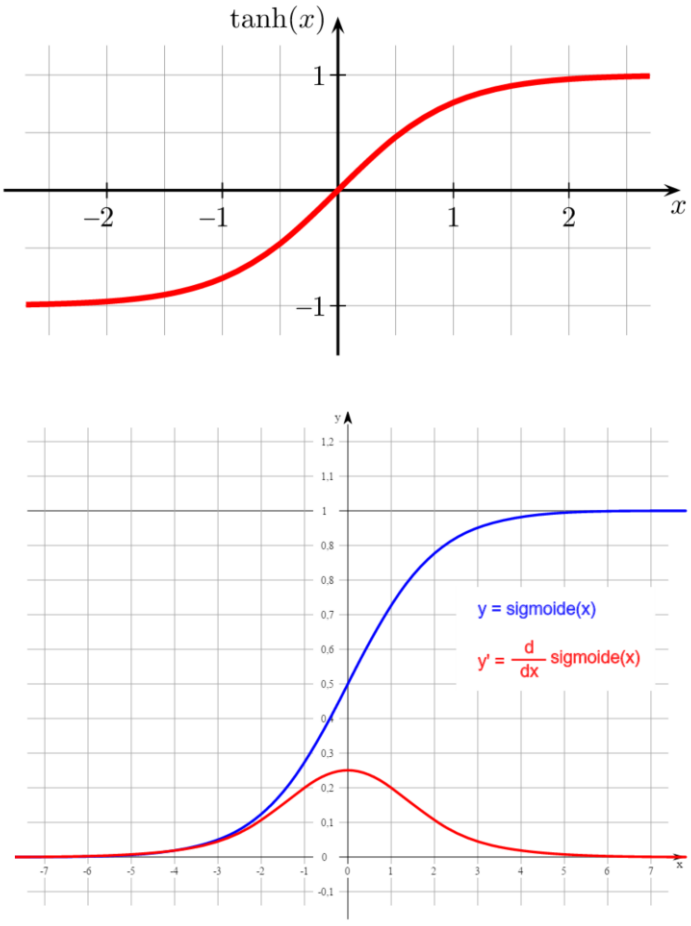

Figure 2: Funciones de activación, tangente hiperbólico y sigmoidal, de izquierda a derecha.

Dado el uso extensivo que haremos de la función *sigmoidal*, expresaremos su derivada de forma más compacta:

$$\frac{\mathrm{d}}{\mathrm{d}x}\text{sigmoid}\,(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\frac{e^{-x}}{(1+e^{-x})} \tag{5}$$

donde tomando el término derecho de tal multiplicación:

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

$$\Rightarrow \frac{1+e^{-x}}{(1+e^{-x})} - \frac{1}{(1+e^{-x})} = \left(1 - \frac{1}{(1+e^{-x})}\right),$$

por lo que entonces la ecuación 5 se puede reescribir como:

$$\frac{\mathrm{d}}{\mathrm{d}x}\text{sigmoid}\,(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})}\left(1 - \frac{1}{(1+e^{-x})}\right)$$

lo cual significa que:

$$\frac{\mathrm{d}}{\mathrm{d}x}\text{sigmoid}\,(x) = \text{sigmoid}\,(x)\,(1 - \text{sigmoid}\,(x)) \tag{6}$$

1.1.2 La capa de salida

Siguiendo el grafo de la red neuronal en la Figura 1, se define el peso neto para la unidad de salida k como:

$$p_k^s\,(\vec{x}, W^s) = \sum_{m=0}^{M} W_{m,k}^s y_m^o, \tag{7}$$

para cada unidad $k = 1, \ldots, K$, donde de manera similar para la capa anterior, $y_0^o = 1$. Observe que el peso neto de una unidad o neurona en la capa de salida está dado por la **combinación lineal de las salidas en las unidades ocultas**, usando los pesos definidos entre la capa oculta y de salida. La salida de cada unidad de la capa de salida, está dada por:

$$y_k^s\,(\vec{x}, W^s) = g^s\,(p_k^s\,(\vec{x}, W^s))$$

donde la función de activación $g^s\,(u)$ para las unidades de salida se elige usualmente según los siguientes casos:

- Para la regresión (clasificación en dominio continuo) se escoge la función identidad, de modo que $g^s\,(u) = u$ o lineal.
- Para la clasificación binaria o de dos clases donde entonces $K = 2$, se utiliza una función de activación de tangente hiperbólico o sigmoidal. Esto pues la red neuronal utiliza una codificación de la salida $1 - K$, de modo que si por ejemplo la red quiere dar a entender que una entrada $\vec{x}_a$ corresponde a la clase 1, entonces las unidades de salida deben representar lo anterior con $y_1 \approx 1$ y $y_2 \approx 0$, y en caso de corresponder a la clase 2, $y_1 \approx 0$ y $y_2 \approx 1$.
- Para la clasificación en múltiples clases $K > 2$ usualmente se utiliza la función *softmax o* la de tangente hiperbólico anteriormente vista. La función *softmax* es utilizada en el algoritmo de regresión logística *y* asocia sus entradas a un valor de salida $0 \leq y_k^s \leq 1$, contribuyendo a la noción probabilística de que la entrada se de. Más formalmente, la función softmax

denota la probabilidad de dado los valores de la muestra $\vec{x}$, la misma sea de la clase C_k, softmax $\left(\vec{p^s}, p_k^s\right) = p\left(C_k|\vec{p^s}\right)$ y está dada por:

$$\text{softmax}\left(\vec{p^s}, p_k^s\right) = \frac{e^{p_k^s}}{\sum_j e^{p_k^s}}.$$

La función *softmax* no es más que la generalización de la función *sigmoidal para* $K > 2$ clases. A continuación el siguiente código:

```
a = [3 2 1];
y(1) = exp(a(1))/ sum(exp(a));
y(2) = exp(a(2))/ sum(exp(a));
y(3) = exp(a(3))/ sum(exp(a));
```

muestra el ejemplo en el que el arreglo de coeficientes de activación está dado por $\vec{p^s} = \begin{bmatrix}3\\1\\2\end{bmatrix}$, donde en este caso el compontente p_1^s es el mayor y debe asignársele la mayor probabilidad, por lo que al hacer que $y_k\left(\vec{p^s}, p_k^s\right) =$ softmax $\left(\vec{p^s}, p_k^s\right)$ para todos los valores p_k^s resulta en $\vec{p^s} = \begin{bmatrix}0.6652\\0.2447\\00.09\end{bmatrix}$, donde por ejemplo y_1 es el valor con mayor probabilidad.

Retomando el funcional para las unidades de salida $y_k^s = g^s\left(p_k^s\right)$, reemplazando el funcional p_k^s por su definición en la Ecuación 7 $p_k^s\left(\vec{x}, W^s\right) = \sum_{m=0}^{M} W_{m,k}^s y_m^o$, se tiene que:

$$y_k^s\left(\vec{x}, W^s\right) = g^s\left(\sum_{m=0}^{M} W_{m,k}^s y_m^o\right) \tag{8}$$

donde a su vez, y_m^o corresponde al funcional de activación definido en la ecuación 4 como $y_m\left(\vec{x}, W^o\right) = g^o\left(p_m^o\left(\vec{x}, W^o\right)\right)$, por lo que reemplazando en la ecuación 8 junto con la definición de p_m^o en la ecuación ; $p_m^o\left(\vec{x}, W^o\right) = \sum_{d=0}^{D} W_{d,m}^o x_d$, se tiene que:

$$y_k\left(\vec{x}, W^o, W^s\right) = g^s\left(\sum_{m=0}^{M} W_{m,k}^s\, g^o\left(\sum_{d=0}^{D} W_{d,m}^o x_d\right)\right) = g^s\left(\sum_{m=1}^{M} W_{m,k}^s\, g^o\left(\sum_{d=1}^{D} W_{d,m}^o x_d + W_{0,m}^o\right) + W_{0,k}^s\right) \tag{9}$$

lo cual es análogo al modelo no lineal estudiado al inicio:

$$y\left(\vec{x}, \vec{w}\right) = f\left(\sum_{j=0}^{M} w_j\, h\left(\sum_{j=0}^{M} \theta_j x_j\right)\right) \tag{10}$$

donde entonces reemplazamos $\vec{w} = W^s$, $f = g^s$ y la función base está dada por $\phi_j\left(\vec{x}, \vec{\theta}\right) = g^o\left(\sum_{d=1}^{D} W_{d,m}^o x_d\right)$, con el vector de parámetros de la función base dado por $\vec{\theta} = W_{d,m}^o$. Observe que la no linealidad de la función base está

dada por la función de activación g^o (usualmente definida como una tangente hiperbólica o una función sigmoidal), y los parámetros de la función base no lineal están dadas por la matriz de parámetros $\vec{\theta} = W^o_{d,m}$.

El proceso de evaluar la ecuación 9 se conoce como la **propagación hacia adelante.** Para realizar la propagación hacia adelante, la red no debe tener ciclos, aunque puede incluir conexiones entre capas no consecutivas. Una cantidad menor de neuronas en la capa oculta respecto a la cantidad de neuronas de entrada implica una reducción de dimensionalidad, disminuyendo el riesgo de sobre-ajuste. Como se mencionó, las redes neuronales son conocidas también como **perceptrones multicapa,** con la diferencia principal de que las redes neuronales usan funciones de activación sigmoidales no lineales, mientras que el perceptrón usa una función de escalón o *Heaviside*, lo que permite que la función de salida de la red sea **diferenciable**, facilitando el proceso de entrenamiento.

Las redes neuronales se dice que son *aproximadores universales* en cuanto son capaces de realizar la regresión a partir de un conjunto de puntos dados y aproximar un funcional, usando como función de activación en la capa de salida una función lineal. La Figura 3 muestra la aproximación de cuatro funciones $f(x) = x^2$, $f(x) = \sin(x)$, $f(x) = |x|$ y $f(x) = H(x)$ correspondiente a la función de Heaviside, con $N = 50$ puntos de entrenamiento. La Figura 3 muestra además las salida de 3 neuronas ocultas, ilustando como la combinación de estas resulta en la función de salida aproximada.

1.2 Entrenamiento de la red

Se seguirá con la misma terminología planteada para los problemas de regresión lineal y clasificación en dos categorías analizados anteriormente. En el problema de clasificación, se define entonces una matriz de muestras de entrada

$$X = \begin{bmatrix} | & | & | & | \\ \vec{x}_1 & \vec{x}_2 & \dots & \vec{x}_N \\ | & | & | & | \end{bmatrix}$$

(lo que corresponde a una manera de representar al conjunto de N vectores de entrenamiento $\{\vec{x}_1, \dots, \vec{x}_n\}$) y el conjunto correspondiente de vectores de etiquetas (conocido de antemano), representado en la matriz

$$T = \begin{bmatrix} | & | & | & | \\ \vec{t}_1 & \vec{t}_2 & \dots & \vec{t}_N \\ | & | & | & | \end{bmatrix}.$$

. La notación general para un modelo de clasificación está dada por $y(\vec{x}, \vec{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\vec{x})\right)$, donde en el caso de la red neuronal de dos capas, los pesos a determinar están dados por dos matrices W^o y W^s y la función de activación $f = g^s$ es una función no lineal. Tal cual se procedió anteriormente para los clasificadore lineales, el objetivo para construir un clasificador es encontrar el vector de pesos $\vec{w}$ que minimice la función de error cuadrático (lo

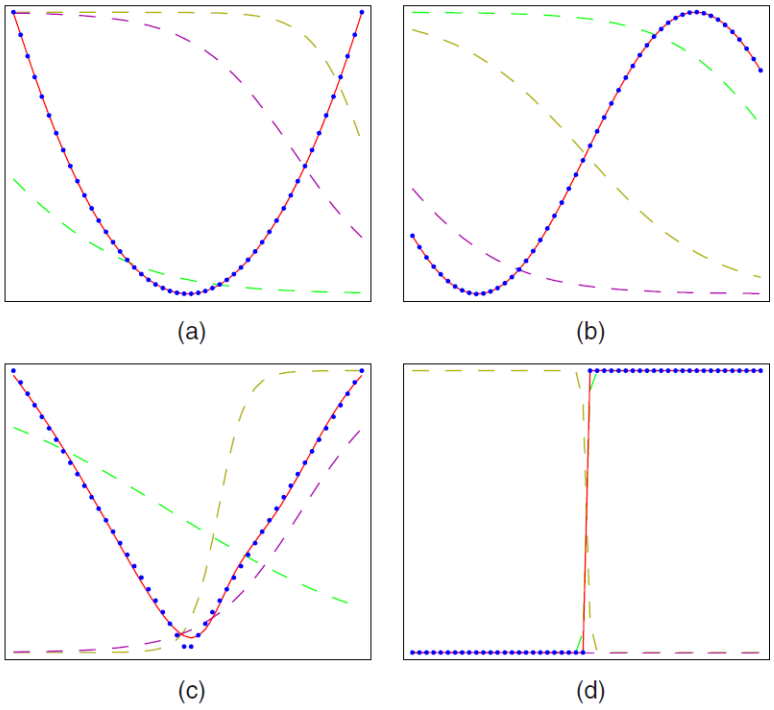

Figure 3: Aproximación de las funciones (a) $f(x) = x^2$ (b) $f(x) = \sin(x)$ (c) $f(x) = |x|$ y (d) la función escalón. En líneas punteadas las salidas de las neuronas ocultas.

que corresponde a maximizar la función de verosimilitud como se concluyó anteriormente, para más detalles sobre ello ver BISHOP, pagina 249):

$$E\left(\vec{w}\right) = \frac{1}{2}\sum_{n=1}^{N}\left\|y\left(\vec{x}_n, \vec{w}\right) - \vec{t}_n\right\|^2,$$

donde se observa que efectivamente la función $E\left(\vec{w}\right)$ tiene su dominio en $\mathbb{R}^2$ y su codominio en $\mathbb{R}$, una función multivariable para la cual podemos calcular fácilmente sus derivadas parciales. Para ilustrar el problema de encontrar el $\vec{w}_{\text{opt}}$ que minimice a la función de error $E\left(\vec{w}\right)$, imaginemos el caso en el que el vector de pesos $\vec{w} \in \mathbb{R}^2$, lo que nos permite dibujar una superficie correspondiente a la función multivariable $E\left(\vec{w}\right) = E\left(w_1, w_2\right)$ como la gráfica de la superficie en la Figura 4 muestra, con tres vectores específicos en la superficie $\vec{w}_A$, $\vec{w}_B$ y $\vec{w}_C$ y con el vector gradiente ∇E evaluado en el punto $\vec{w}_C$.

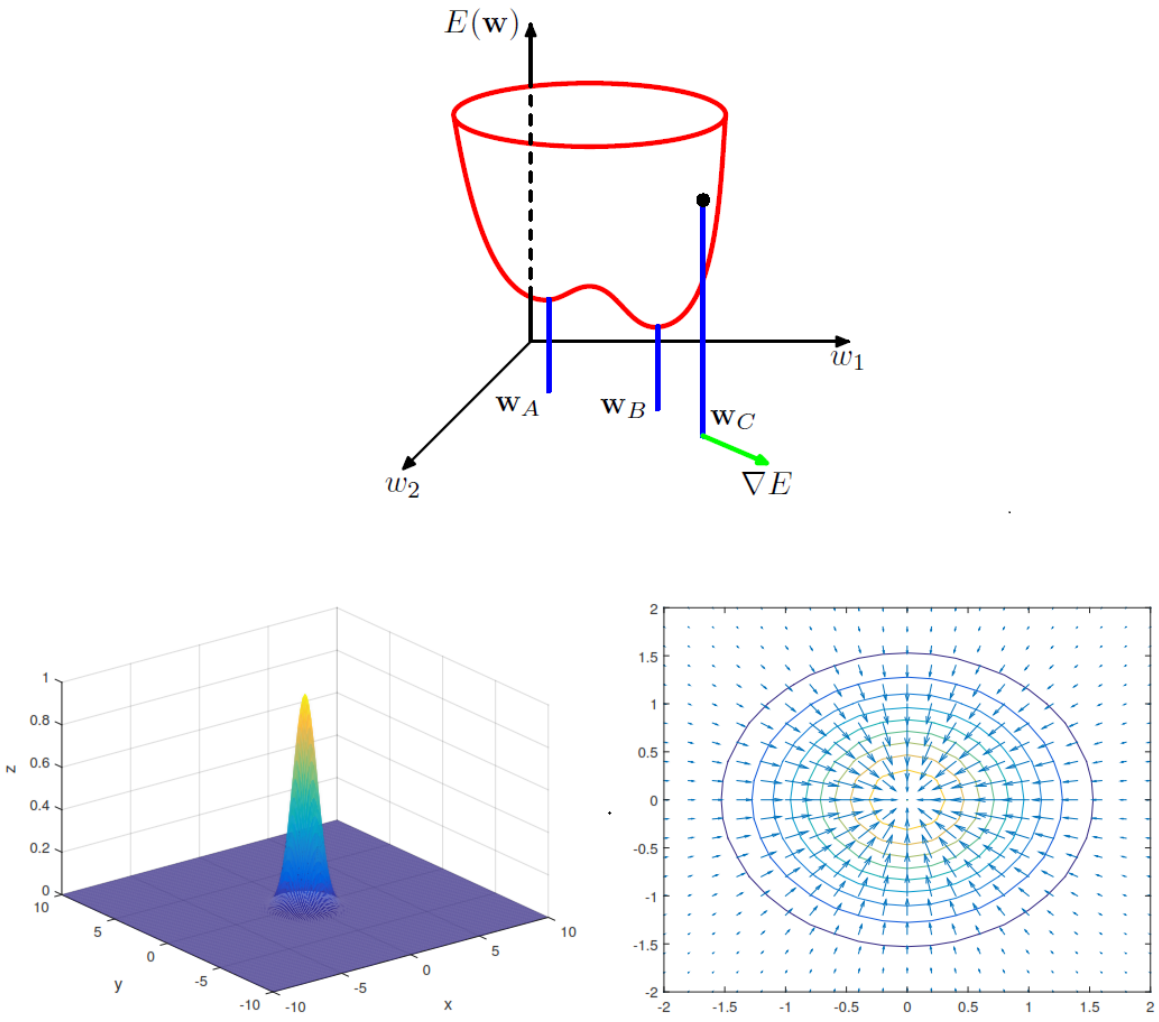

Figure 4: Superficie de la función de error $E\left(\vec{w}\right) = E\left(w_1, w_2\right)$.

Siguiendo con la Figura 4, observe que si por ejemplo la búsqueda de un mínimo en la función $E\left(\vec{w}\right)$ inicia por ejemplo en el vector $\vec{w}_C$, tal proceso de búsqueda debería, en una secuencia finita de iteraciones, encontrar un mínimo de la función de error, lo que podría coincidir con vector como $\vec{w}_B$. Observe que un pequeño cambio en el vector $\vec{w}$ por otro *vector de cambio muy pequeño* $\vec{w}_\delta$

(observe que δ no es un escalar, sino una nomenclatura para el cambio) creando un vector nuevo $\vec{w}_{\text{nuevo}} = \vec{w} + \vec{w}_\delta$, el cual genera un cambio en la función de error E_δ que correspondiente a:

$$E_\delta \simeq (\vec{w}_\delta)^T \nabla E(\vec{w}) = (\vec{w}_\delta) \cdot \nabla E(\vec{w}) = \sum_{i=0} (\vec{w}_\delta)_i \nabla E(\vec{w})_i \tag{11}$$

equivalente al producto punto entre los vectores $\delta\vec{w}$ y $\nabla E(\vec{w})$, donde el último corresponde al vector gradiente, el cual apunta hacia la dirección de mayor crecimiento de la superficie, como también se ilustró anteriormente en la parte de abajo de la Figura 4, para el caso de una función Gaussiana.

Para ilustrar mejor lo anterior, tómese un ejemplo en el que el vector gradiente $\nabla E(\vec{w})$ sea perpendicular al vector de cambio de los pesos $\vec{w}_\delta$. En este caso el producto punto expresado en la ecuacion 11 resultaría en cero, por lo que entonces $E_\delta = 0$. Lo anterior resulta intuitivo, pues la función en esa dirección con $\vec{w}_\delta \to 0$ haría que el valor del funcional sea aproximadamente el mismo, $E(\vec{w}) \cong E(\vec{w}_\delta)$ (Imagine el caso en el que la función $E(\vec{w})$ correspondiera a un plano, en el caso comentado de perpendicularidad entre el vector gradiente y el desplazamiento de los pesos, el último correspondería a una **curva de nivel de tal funcional**).

La Figura 4 también ilustra el hecho de que el mínimo en una función multivariable como el caso de la función $E(\vec{w})$ ocurre cuando el gradiente tiende a cero:

$$\nabla E\left(\vec{w}_{\text{opt}}\right) = 0,$$

como se observa en los gradientes ilustrados para la superficie Gaussiana en las gráficas de la parte inferior en tal figura, o en el caso de la ilustración de la superficie $E(\vec{w})$ para el vector $\vec{w}_B$. Lo anterior significa que el objetivo de la búsqueda es encontrar un vector gradiente con magnitud lo suficientemente pequeña, determinada por ϵ:

$$\|\nabla E(\vec{w})\| < \epsilon.$$

Los puntos donde la magnitud del gradiente tiende a *desvanacerse* se conocen como **puntos estacionarios** los cuales pueden corresponder tanto a máximos como a mínimos locales de la función. Un máximo o mínimo global es un punto estacionario absolutamente máximo o mínimo en toda la función. Como se observa también en la Figura 4, el seguir la dirección contraria al gradiente $-\nabla E(\vec{w})$ nos acerca más a un mínimo local.

Dado que no es posible encontrar una solución analítica a la ecuación

$$\nabla E(\vec{w}) = 0,$$

es necesario utilizar métodos numéricos para ello. La optimización de funciones continuas no lineales es un problema muy estudiado y existen muchas técnicas para lograr tal objetivo. Muchas de las técnicas involucran la elección de un valor inicial para el vector de pesos $\vec{w}_0 \in \mathbb{R}^M$, y con una serie de iteraciones cambiar tal vector *moviéndose* por el espacio $\mathbb{R}^M$, lo cual se expresa

como:

$$\vec{w}(\tau+1) = \vec{w}(\tau) + \alpha \triangle \vec{w}(\tau), \tag{12}$$

donde τ etiqueta la iteración a la que corresponde el vector de pesos (como un funcional vectorial del tiempo), el vector $\triangle \vec{w}(\tau)$ es el vector de *actualización* y $\alpha \in \mathbb{R}$ es un coeficiente que determina la *velocidad* de la actualización. Existen diferentes enfoques para seleccionar el vector de actualización $\triangle \vec{w}(\tau)$, muchos se basan en la evaluación del gradiente $\nabla E(\vec{w}(\tau))$ para su definición.

1.3 Optimización de los pesos por descenso de gradiente y retro-propagación

El enfoque más sencillo para escoger al vector de actualización $\triangle \vec{w}(\tau)$ es hacerlo igual al negativo del vector gradiente de modo que $\triangle \vec{w}(\tau) = -\nabla E(\vec{w}(\tau))$, por lo que entonces la ecuación 12 se reescribe como:

$$\vec{w}(\tau+1) = \vec{w}(\tau) - \alpha \nabla E(\vec{w}(\tau)).$$

Es común realizar varias corridas en las que el vector inicial $\vec{w}_0$ se fija con distintos valores aleatorios.

En las redes neuronales de múltiples capas, recordemos que tenemos al menos dos matrices de pesos a optimizar (W^o y W^s) por lo que al proceso de aplicar el descenso de gradiente desde la capa de salida hacia la capa de entrada se le conoce como **retro-propagación del error**. El proceso de entrenamiento de retropropagación con descenso de gradiente se puede dividir en las siguientes etapas:

- Propagación del error para calcular las derivadas parciales desde la capa de salida hacia la entrada (hacia atrás).
- Utilizar el resultado del gradiente evaluado desde la entrada para computar los ajustes a realizar en los pesos, lo que corresponde a la aplicación de la técnica de **descenso del gradiente**.

Cabe destacar que esta técnica de retro-propagación del error puede ser combinada con cualquier otro método para fijar el valor nuevo del gradiente, por lo que es importante distinguir las dos etapas.

Retomando entonces la ecuación del error definida como:

$$E(\vec{w}) = \frac{1}{2} \sum_{n=1}^{N} \left\| y(\vec{x}_n, \vec{w}) - \vec{t}_n \right\|^2,$$

para facilitar su análisis la podemos reescribir como:

$$E(\vec{w}) = \sum_{n=1}^{N} E_n(\vec{w})$$

con $E_n(\vec{w}) = \frac{1}{2}\|y(\vec{x}_n, \vec{w}) - t_n\|^2$ definido como el error de clasificación para una muestra $\vec{x}_n$ o **función de pérdida** (*loss function*). Con el mismo afán de simplificar el análisis, examinaremos el aprendizaje de cada capa, desde la capa de salida a la oculta.

1.3.1 Aprendizaje en la capa de salida

El cambio en el vector de pesos de la capa de salida W^s, analizado por cada componente o entrada de la matriz:

$$W^s_{m,k}(\tau+1) = W^s_{m,k}(\tau) - \alpha \triangle W^s_{m,k}(\tau), \tag{13}$$

donde el vector de actualización según el método del descenso de gradiente está dado por:

$$\triangle W^s_{m,k}(\tau) = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\sum_{n=1}^{N} E_n\left(W^s_{m,k}\right)\right) \tag{14}$$

A partir de ahora, para facilitar la notación se tendrá que $W^s_{m,k}(\tau) = W^s_{m,k}$, con:

$$E_n(W^s) = \frac{1}{2}\left\|\vec{y^s}(\vec{x}_n, W^s) - \vec{t}_n\right\|^2 = \left\|\begin{bmatrix} y_1 = g^s\left(\sum_{m=0}^{M} W^s_{m,1} y^o_m\right) \\ \vdots \\ y_K = g^s\left(\sum_{m=0}^{M} W^s_{m,K} y^o_m\right) \end{bmatrix}_n - \vec{t}_n\right\|^2,$$

lo cual a su vez se puede simplificar como:

$$E_n(W^s) = \frac{1}{2}\left\|\overrightarrow{y^s}(\vec{x}_n, W^s) - \vec{t}_n\right\|^2 = \frac{1}{2}\sqrt{\sum_{k=0}^{K}\left(y^s_k(\vec{x}_n, W^s) - t_{k,n}\right)^2}^2 = \sum_{k=0}^{K} E_{k,n}(W^s),$$

con:

$$E_{k,n}(W^s) = \frac{1}{2}\left(y^s_k(\vec{x}_n, W^s) - t_{k,n}\right)^2 = \frac{1}{2}\left(g^s\left(p^s_{k,n}\right) - t_{k,n}\right)^2,$$

donde $t_{i,n}$ corresponde al componente i del vector $\vec{t}_n$, y con la función y^s_i definida en la ecuación 8, corresponde a la salida de la unidad $i = 1, \ldots, K$. Calculando la derivada parcial del error para una muestra n respecto a una entrada de la matriz de pesos $W_{m,k}$ se tiene que:

$$\frac{\mathrm{d}E_n}{\mathrm{d}W^s_{m,k}} = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\sum_{k=0}^{K} E_{k,n}(W^s)\right)$$

puesto que un cambio en el peso $W^s_{m,k}$ **solo afecta a la unidad de salida** k, la derivada del error para una muestra n respecto a un componente $W^s_{m,k}$ de

los pesos viene dada por :

$$\Rightarrow \frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^s_{m,k}} = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\frac{1}{2}\left(y^s_{k,n} - t_{k,n}\right)^2\right) = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\frac{1}{2}\left(g^s\left(p^s_{k,n}\right) - t_{k,n}\right)^2\right), \tag{15}$$

con el funcional $p^s_k\left(\vec{x}, W^s\right) = \sum_{u=0}^{M} W^s_{u,k}\, z_u$. Tal derivada parcial planteada en la ecuación 15, se puede descomponer por regla de la cadena como:

$$\frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^s_{m,k}} = \frac{\mathrm{d}E_{k,n}}{\mathrm{d}y^s_{k,n}}\,\frac{\mathrm{d}y^s_{k,n}}{\mathrm{d}p^s_k}\,\frac{\mathrm{d}p^s_{k,n}}{\mathrm{d}W^s_{m,k}}. \tag{16}$$

Analizaremos el caso en el que la función de activación sea sigmoidal, $y^s_k = g^s\left(p^s_k\right) = \text{sigmoid}\left(p^s_k\right)$, para la cual su derivada había sido simplificada en la ecuación 6 y dada como $\frac{\mathrm{d}}{\mathrm{d}p^s_k}\text{sigmoid}\left(p^s_k\right) = \text{sigmoid}\left(p^s_k\right)\left(1 - \text{sigmoid}\left(p^s_k\right)\right)$. Cada una de las derivadas se desarrolla como sigue:

$$\frac{\mathrm{d}E_{k,n}}{\mathrm{d}y^s_{k,n}} = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\frac{1}{2}\left(y^s_{k,n} - t_{k,n}\right)^2\right) = y^s_{k,n} - t_{k,n} = g^s\left(p^s_{k,n}\right) - t_{k,n} \tag{17}$$

$$\frac{\mathrm{d}y^s_{k,n}}{\mathrm{d}p^s_{k,n}} = \text{sigmoid}\left(p^s_{k,n}\right)\left(1 - \text{sigmoid}\left(p^s_{k,n}\right)\right) = y^s_{k,n}\left(1 - y^s_{k,n}\right) \tag{18}$$

$$\frac{\mathrm{d}p^s_k}{\mathrm{d}W^s_{m,k}} = \frac{\mathrm{d}}{\mathrm{d}W^s_{m,k}}\left(\sum_{u=0}^{M} W^s_{u,k} y^o_u\right) = y^o_{m,n}, \tag{19}$$

y es por tales ecuaciones que la derivada expresada en la ecuación 16 vendría dada por:

$$\frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^s_{m,k}} = \left(y^s_{k,n} - t_{k,n}\right)\,\left(y^s_{k,n}\left(1 - y^s_{k,n}\right)\right)\,y^o_{m,n}. \tag{20}$$

lo que implica que, dado que el peso $W^s_{m,k}$ solo afecta a la neurona y^s_k, la derivada parcial del error para todas las otras $K-1$ neuronas es cero:

$$\frac{\mathrm{d}E_n}{\mathrm{d}W^s_{m,k}} = \frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^s_{m,k}} = \left(y^s_k - t_{k,n}\right)\,\left(y^s_k\left(1 - y^s_k\right)\right)\,y^o_{m,n}$$

y:

$$\frac{\mathrm{d}E}{\mathrm{d}W^s_{m,k}} = \sum_{n=1}^{N}\frac{\mathrm{d}E_n}{\mathrm{d}W^s_{m,k}} = \sum_{n=1}^{N}\left(y^s_k - t_{k,n}\right)\,\left(y^s_k\left(1 - y^s_k\right)\right)\,y^o_{m,n} \tag{21}$$

la última ecuación establece la derivada parcial $W^s_{m,k}$ del error para todas las muestras y la anterior la derivada parcial para una muestra. Una cualidad importante que se puede concluir de la ecuación 21, es que el valor del gradiente no depende de la matriz de pesos $W^s_{m,k}$.

Para simplificar la notación, para una muestra n, se define el *delta o cambio de aprendizaje* δ^s_k como:

$$\delta^s_{k,n} = \left(y^s_{k,n} - t_{k,n}\right)\,\left(y^s_{k,n}\left(1 - y^s_{k,n}\right)\right),$$

de esta manera, la actualización del peso $W^s_{m,k}$ vendría dada, según la ecuación 13 por cada muestra como:

$$W^s_{m,k}(\tau+1) = W^s_{m,k}(\tau) - \alpha \triangle W^s_{m,k}(\tau), \tag{22}$$

con

$$\triangle W^s_{m,k}(\tau) = \delta^s_{k,n}\, y^o_{m,n}.$$

Tal notación facilitará la propagación del error como se verá posteriormente.

1.3.2 Aprendizaje en la capa oculta

De manera similar, se desarrollará la ecuación del vector de actualización de los pesos, pero para los pesos que conectan la capa de entrada con la oculta:

$$W^o_{d,m}(\tau+1) = W^o_{d,m}(\tau) - \alpha \triangle W^s_{d,m}(\tau),$$

donde el vector de actualización, de forma similar a lo desarrollado anteriormente, está dado por el negativo del vector gradiente:

$$\triangle W^s_{d,m}(\tau) = \frac{\mathrm{d}}{\mathrm{d}W^s_{d,m}} \left(\sum_{n=1}^{N} E_n\left(W^s_{d,m}\right) \right)$$

Observe que el gradiente se calcula respecto a los pesos de la primer capa, para la función de **error a la salida.** De manera similar a lo realizado anteriormente, definimos el error por cada unidad k de la capa de salida:

$$E_{k,n}(W^o) = \frac{1}{2}\left(y^s_k(\vec{x}_n, W^o) - t_{k,n}\right)^2 = \frac{1}{2}\left(y^s_k\left(p^s_k(\vec{x}_n, W^o)\right) - t_{k,n}\right)^2$$

$$\Rightarrow E_{k,n}(W^o) = \frac{1}{2}\left(y^s_k\left(\sum_{u=0}^{M} W^s_{u,k}\, y^o_u\left(p^o_u(\vec{x}_n, W^o)\right)\right) - t_{k,n}\right)^2 = \frac{1}{2}\left(g^s\left(\sum_{u=0}^{M} W^s_{u,k}\, g^o\left(\sum_{v=1}^{D} W^o_{v,u} x_v\right)\right) - t_{k,n}\right)^2$$

recordando que: $y^s_{k,n} = g^s\left(p^s_{k,n}\right)$, $p^s_{k,n}(\vec{x}_n, W^s) = \sum_{u=0}^{M} W^s_{u,k}\, y^o_u$, $y^o_u(\vec{x}, W^o) = g^o(p^o_u)$, $p^o_u(\vec{x}, W^o) = \sum_{v=0}^{D} W^o_{v,u} x_v$, tenemos que la derivada del error se puede descomponer por regla de la cadena como:

$$\frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^o_{d,m}} = \frac{\mathrm{d}E_{k,n}}{\mathrm{d}y^s_{k,n}} \frac{\mathrm{d}y^s_{k,n}}{\mathrm{d}p^s_{k,n}} \frac{\mathrm{d}p^s_{k,n}}{\mathrm{d}y^o_{m,n}} \frac{\mathrm{d}y^o_{m,n}}{\mathrm{d}p^o_{m,n}} \frac{\mathrm{d}p^o_{m,n}}{W^o_{d,m}}.$$

En este caso, a diferencia del análisis del impacto de un cambio en un peso de la capa de salida, un cambio en la capa oculta ocasiona un cambio en el error de todas las unidades en la capa de salida, el cual dice que se **propaga hacia la capa de salida**, por lo que entonces:

$$\frac{dE_n}{dW^o_{d,m}} = \sum_{k=0}^{K} \frac{\mathrm{d}E_{k,n}}{\mathrm{d}W^o_{d,m}} = \sum_{k=0}^{K} \left(\frac{\mathrm{d}E_{k,n}}{\mathrm{d}y^s_{k,n}} \frac{\mathrm{d}y^s_{k,n}}{\mathrm{d}p^s_{k,n}} \frac{\mathrm{d}p^s_{k,n}}{\mathrm{d}y^o_{m,n}} \right) \frac{\mathrm{d}y^o_{m,n}}{\mathrm{d}p^o_{m,n}} \frac{\mathrm{d}p^o_{m,n}}{W^o_{d,m}}$$

Con las derivadas parciales dadas por:

$$\frac{\mathrm{d}E_{k,n}}{\mathrm{d}y_k^s} = g^s\left(p_k^s\right) - t_{k,n} = y_{k,n}^s - t_{k,n} \tag{23}$$

$$\frac{\mathrm{d}y_{k,n}^s}{\mathrm{d}p_{k,n}^s} = \text{sigmoid}\left(p_{k,n}^s\right)\left(1 - \text{sigmoid}\left(p_{k,n}^s\right)\right) = y_{k,n}^s\left(1 - y_{k,n}^s\right) \tag{24}$$

$$\frac{\mathrm{d}p_{k,n}^s}{\mathrm{d}y_m^o} = W_{m,k}^s$$

$$\frac{\mathrm{d}y_{m,n}^o}{\mathrm{d}p_{m,n}^o} = \text{sigmoid}\left(p_{m,n}^o\right)\left(1 - \text{sigmoid}\left(p_{m,n}^o\right)\right) = y_{m,n}^o\left(1 - y_{m,n}^o\right)$$

$$\frac{\mathrm{d}p_{m,n}^o}{W_{d,m}^o} = x_d,$$

por lo que entonces la derivada parcial del error respecto a un peso específico, para una muestra n, está dada por:

$$\frac{dE_n}{dW_{d,m}^o} = \sum_{k=0}^{K}\left(\left(y_{k,n}^s - t_{k,n}\right)\left(y_{k,n}^s\left(1 - y_{k,n}^s\right)\right)\left(W_{m,k}^s\right)\right)\left(y_{m,n}^o\left(1 - y_{m,n}^o\right)\right)x_d. \tag{25}$$

Para simplificar la notación, recuerde que se había definido en el aprendizaje de la capa de salida el delta $\delta_{k,n}^s = \left(y_{k,n}^s - t_{k,n}\right)\left(y_{k,n}^s\left(1 - y_{k,n}^s\right)\right)$, y tal término lo podemos encontrar en la ecuación anterior 25. Basado en lo anterior, podemos definir el delta en la capa oculta como:

$$\delta_{m,n}^o = \left(\sum_{k=0}^{K}\delta_{k,n}^s W_{m,k}^s\right)\left(y_{m,n}^o\left(1 - y_{m,n}^o\right)\right)$$

y de manera análoga para la notación en la capa anterior, respecto a la ecuación de aprendizaje $W_{d,m}^o(\tau+1) = W_{d,m}^o(\tau) - \alpha \triangle W_{d,m}^s(\tau)$, definimos:

$$\triangle W_{d,m}^s(\tau) = \delta_{m,n}^o x_d$$

1.3.3 Criterio de parada

El criterio de parada puede fijarse como un número de iteraciones P a cumplir, o un máximo porcentaje de las muestras a clasificar incorrectamente.

1.3.4 Ejemplo de entrenamiento

La Figura 5 muestra una red neuronal con una entrada $\vec{x} \in \mathbb{R}^2$ (con $D = 2$, sin incluir la neurona con valor unitario), con $\vec{y^o} \in \mathbb{R}^2$ (con $D = 2$, sin incluir la neurona con valor unitario) y $\vec{y^s} \in \mathbb{R}^1$ ($K = 1$ denotando la pertenencia a una clase $C = 1$ y la no pertenencia a la misma). A continuación se definen

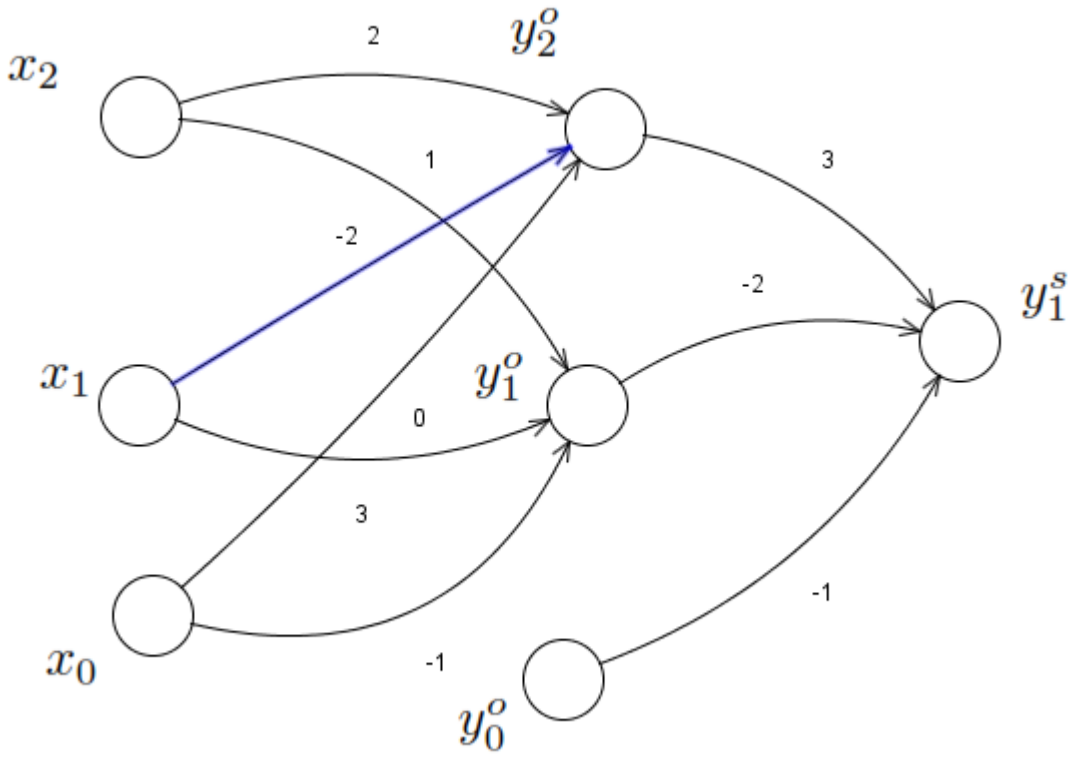

Figure 5: Ejemplo de una red neuronal de dos capas.

los valores de la entrada y los pesos en la capa oculta y de salida (los cuales se pueden suponer fueron inicializados aleatoriamente):

$$\begin{array}{llll} x_0 = 1 & t_1 = 0 & W^o_{0,1} = -1 & W^s_{0,1} = -1 \\ x_1 = 0 & & W^o_{0,2} = 0 & W^s_{1,1} = -2 \\ x_2 = 1 & & W^o_{1,1} = 3 & W^s_{2,1} = 3 \\ & & W^o_{1,2} = -2 & \\ & & W^o_{2,1} = 1 & \\ & & W^o_{2,2} = 2 & \end{array}$$

Suponga que $\alpha = 1$.

Para actualizar los pesos de la red para la iteración $\tau = 1$, realizamos las dos etapas: pasada hacia adelante y retropropagación del error

Pasada hacia adelante A partir de los datos anteriores, calculamos primero los pesos netos para la capa oculta:

$$\begin{aligned} p_1^o = &\quad W_{0,1}^o\, x_0 + W_{1,1}^o\, x_1 + W_{2,1}^o\, x_2 = -1 * 1 + 3 * 0 + 1 * 1 = 0 \\ p_2^o = &\quad W_{0,2}^o\, x_0 + W_{1,2}^o\, x_1 + W_{2,2}^o\, x_2 = 0 * 1 + -2 * 0 + 2 * 1 = 2 \end{aligned}$$

para posteriormente calcular la salida de cada unidad oculta:

$$\begin{aligned} \Rightarrow y_1^o = g^o\left(p_1^o\right) = \tfrac{1}{(1+e^0)} = 0.5 \\ \Rightarrow y_2^o = g^o\left(p_2^o\right) = \tfrac{1}{(1+e^{-2})} = 0.8808 \end{aligned}$$

Respecto a la capa de salida se tiene que:

$$p_1^s = \quad W_{0,1}^s\, y_0^o + W_{1,1}^s\, y_1^o + W_{2,1}^s\, y_2^o = -1 * 1 + -2 * 0.5 + 3 * 0.8808 = 0.6424$$

$$\Rightarrow y_1^s = g^s\left(p_1^s\right) = \tfrac{1}{(1+e^{-0.64})} = 0.6553$$

Pasada hacia atrás y actualización de los pesos Para la capa de salida, calculamos el delta de la única unidad como:

$$\delta_1^s = \left(y_1^s - t_{1,1}\right)\, \left(y_1^s\left(1 - y_1^s\right)\right) = \left(0.6553 - 0\right)\left(0.6553\ \left(1 - 0.6553\right)\right) = 0.148$$

Con base al cálculo del delta para la capa de salida, se actualizan los pesos en la capa de salida según la ecuación $W_{m,k}^s\left(\tau + 1\right) = W_{m,k}^s\left(\tau\right) - \alpha\, \delta_k^s\, y_m^o$:

$$\begin{aligned} W_{0,1}^s\left(\tau + 1\right) = W_{0,1}^s\left(\tau\right) - 1\, \delta_1^s\, y_0^o = -1 - 0.148 * 1 = -1.148 \\ W_{1,1}^s\left(\tau + 1\right) = W_{1,1}^s\left(\tau\right) - 1\, \delta_1^s\, y_1^o = -2 - 0.148 * 0.5 = -2.074 \\ W_{2,1}^s\left(\tau + 1\right) = W_{2,1}^s\left(\tau\right) - 1\, \delta_1^s\, y_2^o = 3 - 0.148 * 0.8808 = 2.8696 \end{aligned}$$

Y para la capa oculta tenemos de forma similar, el delta para las dos unidades:

$$\delta_1^o = \left(\sum_{k=1}^{K=1} \delta_k^s\, W_{1,k}^s\right)\left(y_1^o\left(1 - y_1^o\right)\right) = \left(0.148 * -2\right) * \left(0.5 * \left(1 - 0.5\right)\right) = -0.0740$$

$$\delta_2^o = \left(\sum_{k=1}^{K=1} \delta_k^s\, W_{2,k}^s\right)\left(y_2^o\left(1 - y_2^o\right)\right) = \left(0.148 * 3\right) * \left(0.8808 * \left(1 - 0.8808\right)\right) = 0.0466$$

Por lo que los pesos nuevos para la capa oculta según la ecuación $W_{d,m}^o\left(\tau + 1\right) = W_{d,m}^o\left(\tau\right) - \alpha\, \delta_m^o\, x_d$, vienen dados por:

$$\begin{aligned} W_{0,1}^o\left(\tau + 1\right) = W_{0,1}^o\left(\tau\right) - 1\, \delta_1^o\, x_0 = -1 + 0.074 * 1 = -0.9260 \\ W_{1,1}^o\left(\tau + 1\right) = W_{1,1}^o\left(\tau\right) - 1\, \delta_1^o\, x_1 = 3 + 0.074 * 0 = 3 \\ W_{2,1}^o\left(\tau + 1\right) = W_{2,1}^o\left(\tau\right) - 1\, \delta_1^o\, x_2 = 1 + 0.074 * 1 = 1.074 \\ W_{0,2}^o\left(\tau + 1\right) = W_{0,2}^o\left(\tau\right) - 1\, \delta_2^o\, x_0 = 0 - 1 * 0.0466 * 1 = -0.0466 \\ W_{1,2}^o\left(\tau + 1\right) = W_{1,2}^o\left(\tau\right) - 1\, \delta_2^o\, x_1 = -2 - 1 * 0.0466 * 0 = -2 \\ W_{2,2}^o\left(\tau + 1\right) = W_{2,2}^o\left(\tau\right) - 1\, \delta_2^o\, x_2 = 2 - 1 * 0.0466 * 1 = 1.9534 \end{aligned}$$

1.4 Descenso de gradiente estocástico

En la ecuación general de actualización de los pesos:

$$\vec{w}(\tau+1) = \vec{w}(\tau) - \alpha \nabla E(\vec{w}(\tau)) \tag{26}$$

se estableció que el gradiente de error se calcula sobre las N muestras que conforman el conjunto de muestras de entrenamiento:

$$E(\vec{w}) = \frac{1}{2}\sum_{n=1}^{N} \left\| y(\vec{x}_n, \vec{w}) - \vec{t}_n \right\|^2,$$

para facilitar su análisis la podemos reescribir como:

$$E(\vec{w}) = \sum_{n=1}^{N} E_n(\vec{w})$$

donde $E_n = \frac{1}{2}\left\| y(\vec{x}_n, \vec{w}) - t_n \right\|^2$, correspondiente al error en una sola muestra. Con esto la Ecuación 26 se re-escribe como:

$$\vec{w}(\tau+1) = \vec{w}(\tau) - \alpha \frac{1}{Q}\left(\sum_{n=1}^{Q} \nabla E_n(\vec{w})\right) \tag{27}$$

Se recomienda normalizar el aporte de cada muestra al cambio del gradiente (multiplicando por $\frac{1}{Q}$), para evitar que los *saltos* en la superficie de error dependan de la cantidad de muestras en el conjunto de datos.

Respecto a la cantidad y las muestras del conjunto de entrenamiento Q, se definen tres enfoques de entrenamiento distintos:

1. Descenso de gradiente en el lote: $Q = N$, en este caso, se toman en cuenta todas las muestras del conjunto de muestras de entrenamiento N, sin ningún criterio estocástico.
2. Descenso de gradiente estocástico en línea: $Q = 1$. Se escoge una sola muestra aleatoria para posteriormente actualizar los pesos.
3. Descenso de gradiente estocástico por mini-lotes: $1 < Q < N$. En este caso se construye un sub-conjunto de las N muestras de entrenamiento para realizar el entrenamiento. Cuando $Q \to 1$, la señal de error de la función tenderá a ser más ruidosa, e inestable, mientras que si $Q \to N$, la señal será más estable, pero tardará más en converger. Q corresponde a la **cantidad de muestras del mini-lote**.

De esta forma, el algoritmo de entrenamiento generalizado, para R pasadas o epochs, está dado por:

1. $\tau \leftarrow 0$
2. Mientras $\tau \leftarrow R$:
   (a) $\vec{w}(\tau+1) \leftarrow \vec{w}(\tau) - \alpha \frac{1}{Q}\left(\sum_{n=1}^{Q} \nabla E_n(\vec{w})\right)$
   (b) $\tau \leftarrow \tau + 1$

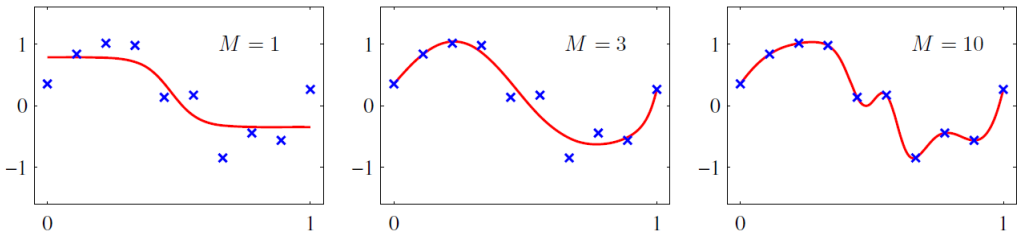

Figure 6: Redes neuronales con distintos valores de M para el problema de la regresión.

1.5 Ajuste de parámetros y regularización en redes neuronales

El número de neuronas en la capa de entrada D y las K neuronas en la capa de salida son generalmente determinadas por la dimensionalidad de las entradas y la cantidad de clases, respectivamente. La cantidad de neuronas en la capa oculta M es un parámetro libre, el cual controla el número de parámetros del modelo de la red, por lo que es usual buscar una cantidad M de neuronas que generen un modelo con un bajo sobre-ajuste, maximizando su generalización. La Figura 6 muestra como la red neuronal se comporta al ajustarse a 10 muestras para el problema de regresión, con distintos valores de M.

Sin embargo, el error de generalización no es una simple función de M, por la existencia de mínimos locales, por lo que se recomienda realizar una gráfica del error en un **conjunto de datos de validación** para evaluar el mejor modelo en función de M. Otra alternativa consisten en implementar una regularización como la analizada para el caso de la regresión polinomial, con una ecuación del error modificado como y fijando un M relativamente grande:

$$\widetilde{E}\left(\vec{w}\right) = E\left(\vec{w}\right) + \frac{\lambda}{2}\vec{w}^T\vec{w}$$

Una heurística común es fijar el número de neuronas en la capa oculta M en el siguiente rango:

$$\frac{D}{2} \leq M \leq 2D$$

donde usualmente se consiguen los menores errores en el conjunto de datos de validación con valores menores.

1.6 Invariantes

Como se discutió en el capítulo de preprocesamiento, la predicción de una clase para una nueva muestra $\vec{x}$ debe ser *invariante* bajo una o más transformaciones de tal muestra de entrada $\vec{x}$. Por ejemplo, para la clasificación de imágenes digitales de firmas, para una entrada $\vec{x}_i$, la clase asignada por el clasificador debe ser la misma sin importar cambios en la traslación o la escala de tal muestra.

Si existe una suficiente cantidad de muestras, un modelo adaptativo como una red perceptrón multi-capa puede llegar a *aprender* o generalizar tal invarianza, incluyendo entonces una cantidad grande de los ejemplos con las distintas transformaciones, por ejemplo, incluyendo muchas muestras rotadas para el caso de las firmas. Sin embargo, esto puede ser impráctico si el número de posibles transformaciones es muy alto, o existen pocas muestras en el conjunto original de datos. Existen entonces 3 alternativas básicas para lidiar con las variaciones en las muestras de entrada:

1. Aumentar el conjunto de datos usando réplicas del conjunto de datos originales, de acuerdo a las invariantes deseadas.

2. Implementar una etapa de preprocesamiento y extracción de características, donde estas últimas sean invariantes a las características deseadas, lo que requiere también la aplicación de ambas etapas para las nuevas muestras a clasificar.

3. Construir las propiedades invariantes dentro de la misma red, usando lo que que se conoce como espacios receptivos y pesos compartidos, conceptos implementados en las redes convolucionales.

4. Combinar los enfoques 1 y 3.

2 Redes profundas o Deep Learning

El aprendizaje profundo provee un marco de trabajo muy utilizado en la industria para el aprendizaje supervisado. Al agregar más capas y más unidades en una capa, la red profunda puede representar funciones de complejidad mayor.

Las redes de alimentación progresiva o perceptrón multicapa (MLP), son también llamadas **redes profundas de alimentación progresiva.** Como se vió anteriormente, estas redes están diseñadas para aproximar una función f^*, que permita asociar una entrada multivariable $\vec{x} \in \mathbb{R}^n$ a una categoría y en el caso de la clasificación, o un valor continuo, en el caso de la regresión:

$$y = f\left(\vec{x}, \vec{\theta}\right)$$

aprendiendo los parámetros $\vec{\theta}$ para encontrar la mejor aproximación a la función f^*, de modo que $f \rightarrow f^*$. Estos modelos se les llama de alimentación progresiva o *feedforward* porque la información fluye desde la evaluación de $\vec{x}$ hasta la definición de los pesos en la capa más externa que define a y. No existen conexiones hacia capas anteriores. Cuando estas conexiones existen, se le llaman **redes recurrentes**, las cuales se presentan más adelante.

La **cantidad de capas** en una red MLP define define su **profundidad**. Las capas ocultas son construídas a lo largo del entrenamiento para lograr la mejor aproximación de f^*. Las redes MLP se le llaman redes neuronales por la analogía entre las unidades en una red y las neuronas en el tejido cerebral, y

las conexiones entre tales unidades y los enlaces sinápticos entre las neuronas. Cada neurona recibe múltipes entradas de las neuronas en la capa anterior, y computa su valor de activación, mediante la **función de activación**, constituyendo una función $\mathbb{R}^n \to \mathbb{R}$. La cantidad de neuronas en cada capa se asocia con el **ancho** de la capa.

Una forma de entender las redes MLP es iniciar con los modelos lineales, y considerar como superar sus limitaciones. Algunos modelso lineales como la regresión logística y la regresión lineal, los cuales son muy utilizados pues pueden encajarse en un conjunto de datos usando expresiones cerradas u optimización convexa.

Para extender modelos lineales y representar funciones no lineales de $\vec{x}$, se puede aplicar tal modelo lineal a la entrada transformada:

$$\phi\left(\vec{x}\right)$$

donde ϕes una transformación no lineal, la cual podemos pensar que provee un conjunto de características de $\vec{x}$, o una nueva representación de tal vector en un nuevo espacio. Las siguientes son opciones para escoger ϕ:

1. Usar una función genérica ϕ, implicitamente usadas en en las máquinas de núcleos, de infinitas dimensioens, para más bien generar un espacio de más alta dimensión. Sin embargo, al usar esta opción, la generalización es pobre y el sobre ajuste alto.

2. Manualmente diseñar ϕ, enfoque utilizado en el reconocimiento de patrones, lo cual ha involucrado décadas de esfuerzo en áreas especializdas como visión artificial y reconocimiento del habla, con poca transferencia de conocimiento entre dominios.

3. La estrategia del aprendizaje profundo es aprender ϕ, por lo que se tiene un modelo que combina linealmente con los pesos $\vec{w}$ las salidas de al función $\phi\left(\vec{x}, \vec{\theta}\right)$, donde ahora $\vec{\theta}$ corresponde a los parámetros de la transformación no lineal ϕ, por lo que entonces:

   $$y = f\left(\vec{x}, \vec{\theta}, \vec{w}\right) = \phi\left(\vec{x}, \vec{\theta}\right)^T \vec{w}$$

   donde en una red MLP, como ya vimos, los parámetros $\vec{\theta}$ corresponden a la capa oculta y deben ser aprendidos. Para capturar los beneficios del primer enfoque, se escogen familias de $\vec{\theta}$ generales. Para combinar con el enfoque 2, se puede escoger la función ϕ con formas conocidas para representar el conocimiento a priori, resultado de la investigación en el dominio específico, de modo que el entrenamiento de la red encuentre los parámetros óptimos $\vec{\theta}$ para tal funcional.

3 Redes convolucionales

3.1 Introducción al a arquitectura de una red convolucional

Material tomado de `http://cs231n.github.io/convolutional-networks/`

Sea una imagen de entrada, o matriz $X \in \mathbb{R}^{M_1 \times M_2}$ y un filtro $F_1 \in \mathbb{R}^{U_1 \times U_2}$, la convolución es básicamente el computo del producto punto en los puntos de la imagen. Un filtro se puede interpretar como un conjunto de pesos que conecta la entrada con la salida, pero conectan solo unidades de entrada locales, con lo cual podemos interpretar un filtro como un arreglo de pesos $F_1 = W_1$. Es por ello que la red, a lo largo de su entrenamiento, cambiará los coeficientes del filtro, por lo que entonces los filtros se ajustarán en tiempo de entrenamiento.

Por ejemplo tómese el caso en que $M_1 = M_2 = 32$ y $U_1 = U_2 = 5$, como se ilustra a continuación.

La convolución resulta en un mapa de activación:

$$X * F_1 = A_1$$

el cual tiene dimensiones $A_1 \in \mathbb{R}^{V_1 \times V_2}$ con $V_1 = M_1 - U_1$ y $V_2 = M_2 - U_2$. Las redes convolucionales están compuestas en su **capa convolucional** por un conjunto o banco de n filtros $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}$, con lo cual, al **convolucionar cada filtro** con la imagen de entrada X, se forma un conjunto de n mapas de activación $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Tal conjunto de matrices conforma la entrada de la **capa de reducción** o capa ReLU, la cual aplica una función de activación, como por ejemplo la función

$$\max(0, x)$$

(correspondiente a una umbralización por cero). Observe que esta capa no tiene parámetros por ser ajustados.

Posteriormente se define la **capa piscina** o *pool,* la cual realiza una operación de sub-muestreo, resultando en un conjunto de matrices submuestreadas $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$, con $S_i \in \mathbb{R}^{W_1 \times W_2}$. El submuestreo puede implementarse con distintas políticas, por ejemplo, una política de **submuestreo máxima** en una ventana *de* 2×2, toma, por cada ventana de 2×2, el valor máximo, como muestra la Figura 8. Otra alternativa es el **submuestreo de promediado**. Ello hace que a la salida de la capa de *pooling*, las dimensiones de la matriz a la salida sean sustancialmente menores que las dimensiones de la matriz a la entrada, por lo que entonces $W_1 < V_1$ y $W_2 < V_2$. Esta capa tiene la función incrementar el nivel de abstracción, tal cual sucede en algoritmos de determinación automática de características como SIFT u ORB.

Finalmente, se define la **capa completamente conectada** la cual está compuesta por K unidades, cada una correspondiente a una de las K clases en las que se clasificarán los datos de entrada. Como su nombre lo implica, cada una de las neuronas tendrá conexión con todos los pixeles resultantes de la *capa pooling*, por lo que se comporta como una capa en una red perceptrón multicapa. Es importante señalar que el método del gradiente descendente ajustará

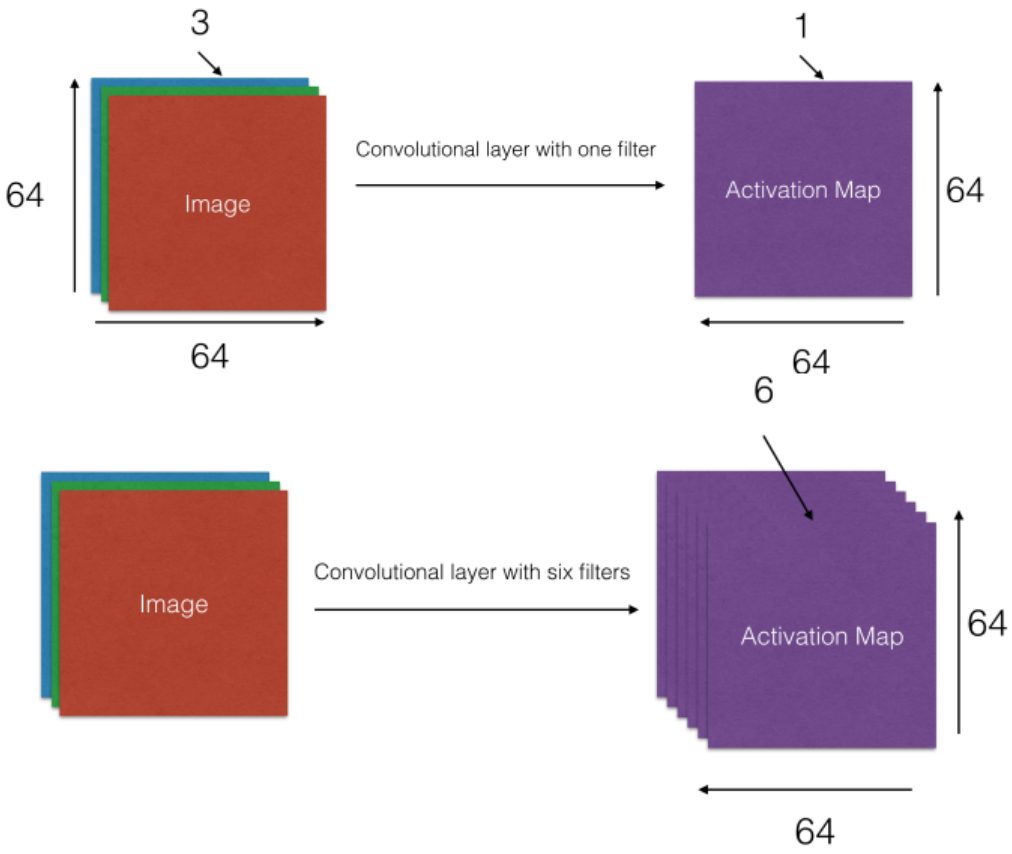

Figure 7: La salida de la capa convolucional es el mapa de activación.

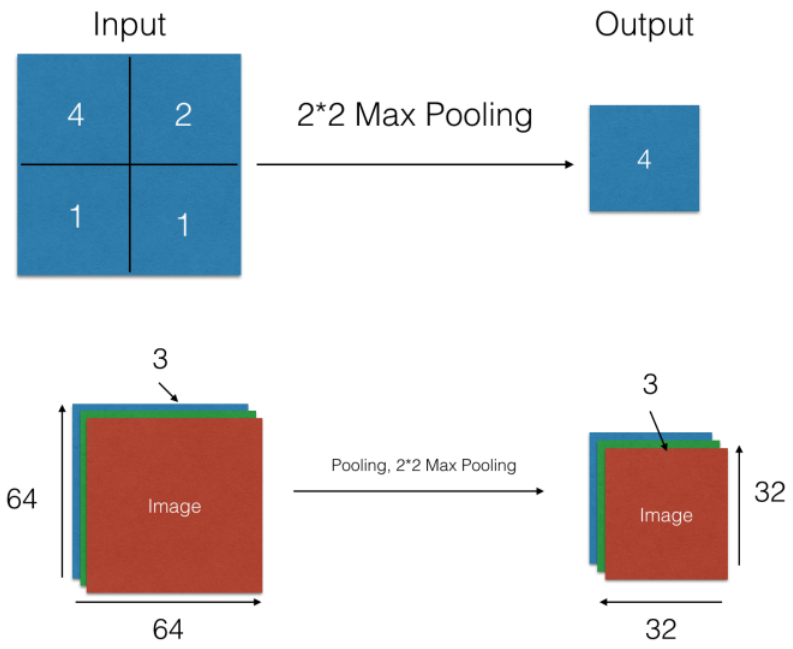

Figure 8: Ilustraciónd el muestreo en la capa de *pooling*.

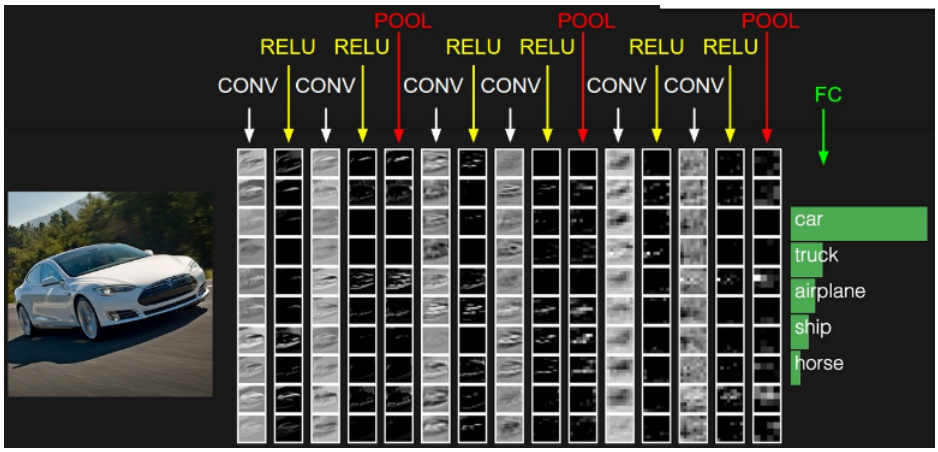

Figure 9: Una red convolucional típica.

únicamente parámetros en la capa completamente conectada y en la convolucional.

La capa completamente conectada implementa usualmente como funciones de activación funciones del tipo:

$$\max(0, x)$$

La Figura 9 ilustra una red convolucional completa, en la que se apilan múltiples series de capas convolucional-ReLU-pooling, en la cual se puede observar que los filtros en las primeras capas aprenden características de más bajo nivel.

3.1.1 Las unidades de rectificación lineal y sus generalizaciones

Como se señaló anteriormente, las redes convolucionales utilizan usualmente la función de activación de rectificación lineal, la cual está dada por:

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

y gráficamente corresponde a lo diagramado en la Figura 10. Típicamente $z = \vec{w}^T\vec{x} + b$ para una red neuronal. Como se puede observar en la Figura 10, $z = 0$ existe una discontinuidad, que hace que la función g no sea derivable en tal punto. Esto hace que parezca que la función no pueda usarse para la activación de una unidad, pues la optimización de los pesos $\vec{w}$ de la red se basa en el calculo del gradiente de tal funcional. Para evitar este problema, definimos la derivada en dos tramos, la derivada izquierda de la función $g(z)$, correspondiente a este caso a cero y la derivada derecha de la función $g(z)$ correspondiente a 1, lo que matemáticamente corresponde a:

$$g'(z) = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{si } z < 0 \end{cases}$$

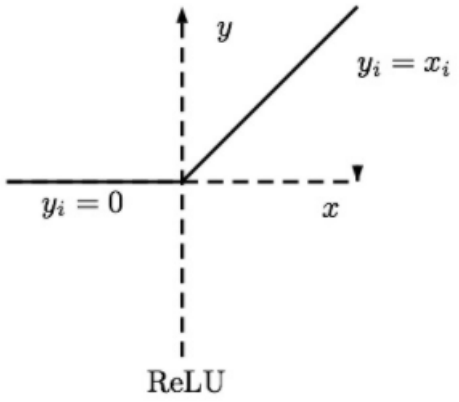

Figure 10: Función de rectificación lineal.

Observe que para $z = 0$, la derivada no está definida. Su indefinición no es de preocupación si tomamos en cuenta que el cálculo de $z = \vec{w}^T\vec{x} + b$ muy dificilmente arribará al resultado $z = 0$, pero en tal caso, se procede a hacer que $z = \epsilon$, con $\epsilon \to 0$.

La ventaja de la función maximo respecto a la función identidad $g(z) = z$ es que mantiene el gradiente con una magnitud alta cuando la unidad está activa, y cuando se desactiva, el gradiente se hace cero. Un gradiente alto cuando la unidad está activa, y un gradiente muy pequeño cuando la unidad está inactiva es en realidad lo más deseable, puesto que el gradiente nulo puede ocasionar un problema de **gradiente desvanecido.** Para ello se usa la generalización de la rectificación lineal, con la siguiente forma:

$$g(z) = \max\{0, z\} + \alpha_i \min\{0, z\}$$

Notese las siguientes particularizaciones de la formulación general anterior:

- Rectificación absoluta: $\alpha = -1$

3.1.2 Entrenamiento

Como observamos, cada funcional en la red convolucional, es una capa en sí misma. Esto permite, definir, por cada capa, su función gradiente por separado. Si la función en una capa específica, no tiene parámetros por ajustar, entonces no es necesario el cálculo del gradiente. En la red convolucional tratada hasta ahora, las siguientes son capas con pesos a ajustar, y por ende, gradiente a calcular:

- Capa convolucional
- Capa completamente conectada.

El resto de capas no tiene parámetros a ajustar, por lo que su función gradiente no se calcula:

- Capa de Reducción o ReLU.
- Capa de piscina.

3.2 Capa convolucional

Como se mencionó anteriormente, la capa convolucional recibe una matriz de entrada $X \in \mathbb{R}^{M_1 \times M_2}$ y está compuesta por uan serie de filtros $F_i \in \mathbb{R}^{U_1 \times U_2}$, cuyos coeficientes son *aprendidos* durante el proceso de entrenamiento. Los filtros tienen usualmente dimensiones pequeñas, del orden de 3×3, 5×5 o 7×7 (usualmente son cuadrados por lo que entonces $F_i \in \mathbb{R}^{U \times U}$,). Recordemos que el proceso de **deslizar el filtro y calcular el producto punto para cada pixel** en la entrada X resulta en el **mapa de activación**. Tal producto punto en cada pixel de la imagen de entrada, puede interpretarse como en fijar una conectividad *local* de las neuronas en la siguiente capa, respecto a la matriz de entrada X, como lo ilustra la imagen de la Figura 11. Intuitivamente, los filtros se entrenarán para encontrar características de bajo nivel en las capas más exteriores, como bordes, manchas de un color específico, etc (según estén definidas las etiquetas).

Observe que el tamaño de la ventana $U_1 \times U_2$ define la conectividad local de las neuronas en la siguiente capa, de modo que entre más grande, e incluso si llega a ser de las dimensiones de la imagen de entrada X, de modo que $U_1 = M_1$ y $U_2 = M_2$, tendremos definida una capa completamente conectada, tal cual sucede en el perceptrón multicapa estudiado anteriormente. Sin embargo, ello se torna impráctico computacionalmente, pero sobre todo, no explota la correlación local existente en las imágenes, donde las características que por ejemplo usamos para distinguir un rostro, tienen una marcada localidad. Al tamaño de la ventanta en los filtros $U_1 \times U_2$ se le llama el **campo receptivo,** y corresponde a un hiperparámetro de la red convolucional.

Lo anterior se ilustra mejor al comparar la conectividad de una red perceptrón multicapa. Recapitulando el concepto de la convolución, la Figura 12 lo resume. Si tomamos la imagen de entrada como los valores en la capa de entrada:

$$\vec{x} = \begin{bmatrix} a & b & c & d & e & f & g & i & j & k \end{bmatrix}$$

de un perceptrón multicapa, el cual en su función de peso neto se realiza con la combinación lineal $s_i = \vec{w}^T\vec{x}$, la cual se desarrolla en la Figura 12. Como se observa en tal desarrollo, cada recuadro en la parte inferior corresponde al valor en la neurona de salida s_i. El vector de pesos en este caso estaría compuesto por los coeficientes del filtro o *kernel,* en este caso dado por*:*

$$\overrightarrow{w} = \begin{bmatrix} w & x & y & z \end{bmatrix}$$

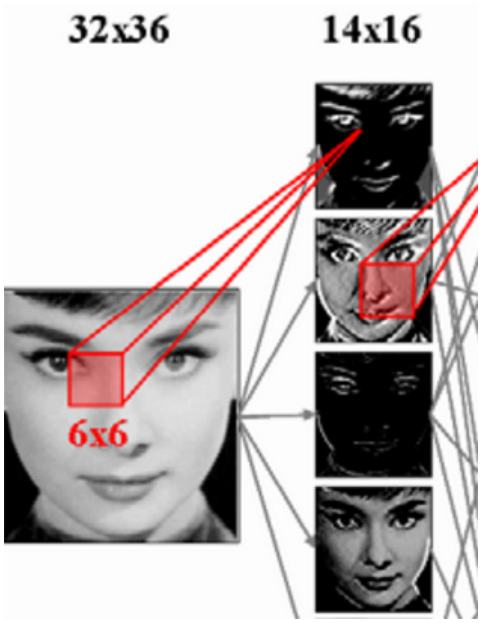

Figure 11: Conectividad local de las neuronas en la primer capa.

Supóngase un ejemplo más sencillo, en el el cual se convolucionan dos funciones $\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}$ y $\vec{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}$, con lo que $\overrightarrow{w} \in \mathbb{R}^3$. El vector de salidas está dado entonces por:

$$\overrightarrow{s} = \vec{x} * \vec{w} = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 & s_5 \end{bmatrix},$$

donde

$$\begin{aligned} s_1 &= w_2 x_1 + w_3 x_2 \\ s_2 &= w_1 x_1 + w_2 x_2 + w_3 x_3 \\ s_3 &= w_1 x_2 + w_2 x_3 + w_3 x_4 \\ s_4 &= w_1 x_3 + w_2 x_4 + w_3 x_5 \\ s_5 &= w_1 x_4 + w_2 x_5 \end{aligned}$$

La Figura 13 ilustra este simple ejemplo usando notación de grafos y la compara con la arquitectura de un perceptrón multicapa, también conocido como una **red completamente conectada**. Para el caso de tal red completamente conectada, cada arista corresponde a un peso w_i distinto, por lo que se tiene un total de $5 \times 5 = 25$ pesos distintos y por ende $\overrightarrow{w} \in \mathbb{R}^{25}$. Tales pesos deben ser definidos en la etapa de entrenamiento. La repetición de los pesos para el caso de una red convolucional ilustrada en el grafo de la Figura 13, y su consecuente menor dimensionalidad, se conoce como **el compartir parámetros** o *parameter sharing*. Tal como se mencionó anteriormente, la analogía con una ventana deslizante a realizarse en una capa convolucional otorga la capacidad de la red de *desacoplar* de su aprendizaje la posición de ciertas características

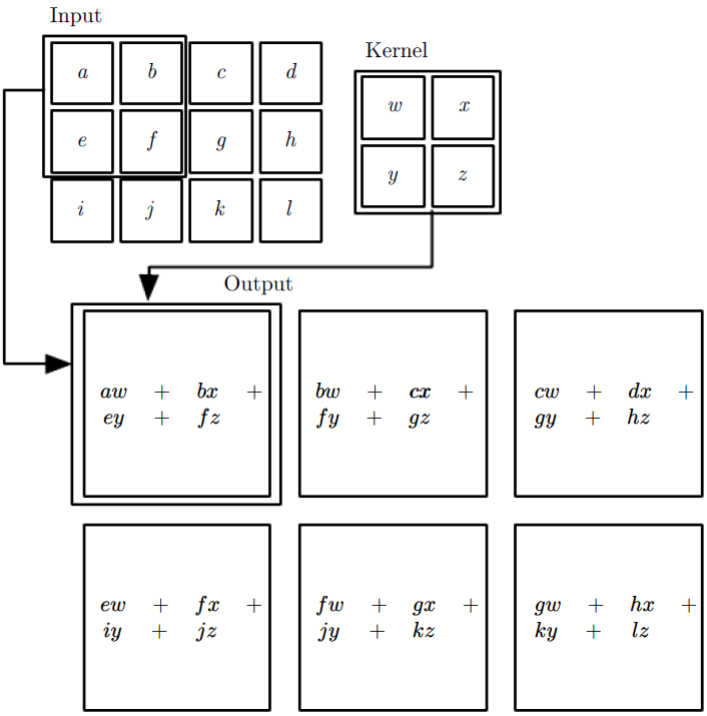

Figure 12: Convolución matricial. Tomado del libro *Deep Learning* de Yoshua Bengio.

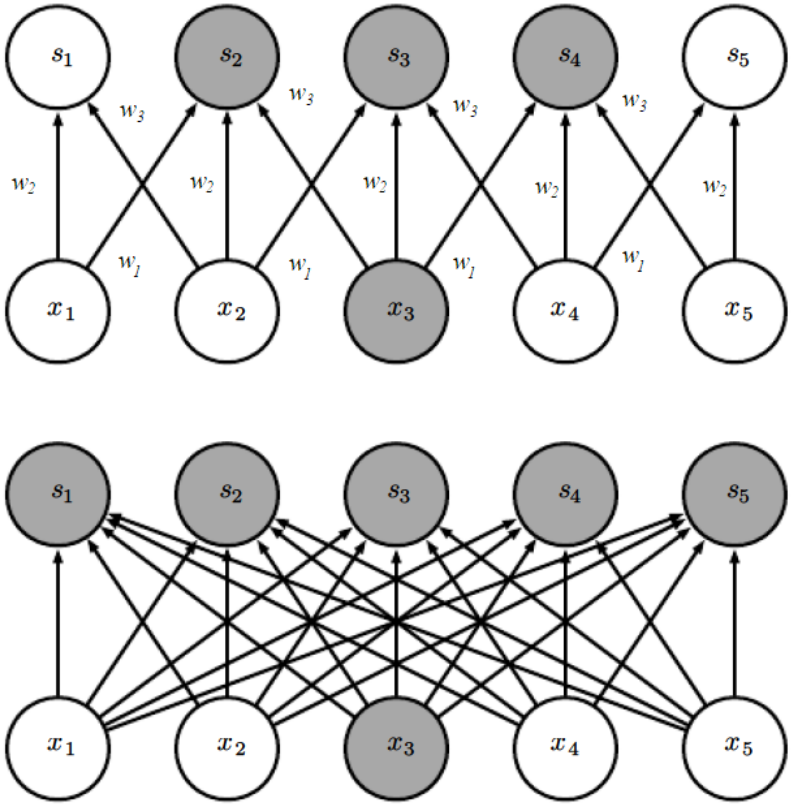

Figure 13: Arquitectura de una red completamente conectada comparada con una red convolucional, con un solo filtro.

que contribuyen a discernir la clase de una entrada específica, dándole **invariancia a la traslación** de las características.

Otros hiperparámetros en la capa convolucional son:

1. La cantidad de los n filtros $\mathbf{F} = \{F_1, F_2, \ldots, F_n\}$ a utilizar en la capa convolucional, los cuales definen la ***profundidad*** o cantidad de mapas de activación $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Observe que al finalizar el proceso de convolución, en un elemento (x, y) de cada una de las matrices A_i (correspondiente a la entrada $A_i(x, y)$), se habrá calculado la respuesta de el filtro F_i para la región circundante a tal elemento (x, y) en la matriz de entrada X, lo cual en una notación más compacta corresponde a $\mathbf{A}(x, y)$ y se le denomina **fibra**.

2. El ancho de cada filtro o núcleo (kernel) k. A esto también se le conoce como el espacio receptivo, y define la sensibilidad a características de distintos tamaños.

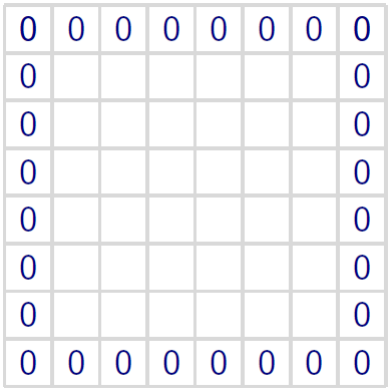

Figure 14: Agregado de ceros en los bordes de la imagen.

3. El **paso** p a utilizarse en la convolución, con lo que por ejemplo, si el paso es unitario, el filtro se desliza de pixel en pixel, y si el paso es de 2, el filtro se deslizará cada par de pixeles, y así sucesivamente.

4. La **cantidad de ceros** c añadidos a los bordes de la imagen. El añadir ceros en los bordes de la imagen permite preservar las dimensiones de la entrada $X \in \mathbb{R}^{M_1 \times M_2}$ en los mapas de activación $A_i \in \mathbb{R}^{V_1 \times V_2}$, de modo que $V_1 = M_1$ y $V_2 = M_2$, si se hace que

$$c = \frac{k-1}{2}.$$

   En la Figura 14 se ilustra el caso en el que $k = 3 \Rightarrow c = 1$.

5. **Coeficientes para el aprendizaje**: Estos son heredados de las ecuaciones de actualización de los pesos en un perceptrón multi-capa:

   (a) **Coeficiente de aprendizaje** α, el cual define el *paso* con el cual se varían los pesos en la superficie de error:

$$W_i(\tau + 1) = W_i(\tau) - \alpha \triangle W_i(\tau),$$

   el cual se define idéntico para toda capa i.

   (b) **El coeficiente de moméntum** μ: en los algoritmos más comunes de entrenamiento estocástico, se incluye la influencia de los pesos anteriores, para mejorar la exploración de regiones locales en la superficie de error, modelando la *inercia* de un cuerpo:

$$W_i(\tau + 1) = W_i(\tau) - \alpha \triangle W_i(\tau) + \mu \triangle W_i(\tau - 1)$$

   (c) El **tamaño del lote para el entrenamiento** o la cantidad de muestras por mini lote Q.

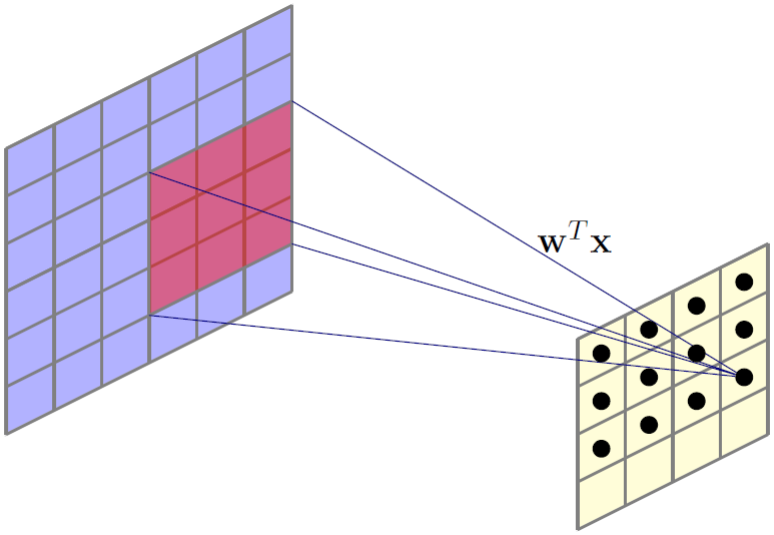

Figure 15: Capa convolucional con $n = 1$ filtros e igual número de mapas de activación resultantes.

La definición de los primeros 4 hiperparámetros influye en las dimensiones de los n mapas de activación resultantes al aplicar una capa de convolución con n filtros de dimensiones $k \times k$, con un paso p, con lo que cada mapa de activación tendrá de alto y ancho:

$$\frac{n-k}{p} + 1$$

Con una entrada de $m \times m$ pixeles. Tómese por ejemplo la aplicación de una capa convolucional a una imagen con $m = 6$ pixeles de alto y ancho, usando un filtro de ancho y alto $k = 3$, y un paso $p = 1$, como se ilustra en la Figura 15. El ancho y alto del mapa de activación viene entonces dado por:

$$\frac{6-3}{1} + 1 = 4$$

como también se ilustra en la Figura 15. Esto en el caso en el que no se concatenen ceros en los extremos de la imagen.

4 Herramientas y ejemplos

4.1 Redes convolucionales con Keras

Keras es un API de tensor-flow, CNTK, o Theano, que permite en generar redes neuronales y convolucionales de forma muy sencilla. El siguiente es un ejem-

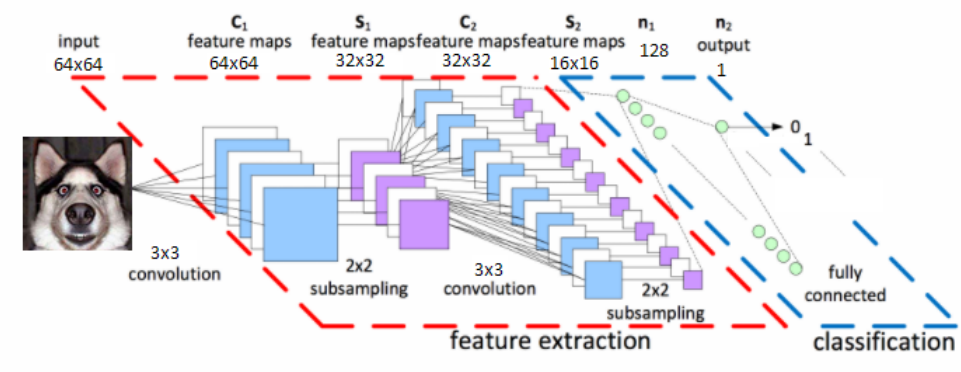

Figure 16: Ilustración de red convolucional simple para la clasificación de imágenes con perros y gatos, en dos clases.

plo sencillo de una red diseñada para clasificar imágenes digitales de perros y gatos. La Figura 16 muestra los detalles de la arquitectura.

El siguiente código de Keras implementa la arquitectura propuesta en la Figura 16, y detalla los parámetros de cada función.

```
# Convolutional Neural Network
# Installing Theano # pip install --upgrade --no-deps git+git://github.
    com/Theano/Theano.git
# Installing Tensorflow # pip install tensorflow
# Installing Keras # pip install --upgrade keras
# Part 1 - Building the CNN
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
# Initialising the CNN classifier = Sequential()
# Step 1 - Convolution, input image has 64x64 pixels #32 filters of 3x3
    , by default keras does not do zero padding, changing dimensions
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation
    = 'relu'))
# Step 2 - Pooling classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Adding a second convolutional layer
#RELU activation function
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))
# Step 3 - Flattening (vectorization)
classifier.add(Flatten())
# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))
# Compiling the CNN
#loss function: error function to be minimized, usually is the MSE, but
    the crossentropy converges faster
#for weight optimization ADAM is used, a mod. of stochastic gradient
```

```
    descent
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
# Part 2 - Data augmentation and model training
from keras.preprocessing.image import ImageDataGenerator
# Data augmentation: performs pixel wise normalization (0-1), shear,
    size and flipping
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,  zoom_range =
     0.2, horizontal_flip = True)
#Test images are also normalized
test_datagen = ImageDataGenerator(rescale = 1./255)
#We specify the training data set path, a folder per class
#Batch size: how many samples are randomly taken before doing the back
    propagation (32 samples are used to calculate the error and
    recalculate the weights)
training_set = train_datagen.flow_from_directory('dataset/training_set'
    , target_size = (64, 64), batch_size = 32, class_mode = 'binary')

                                   #We specify the test data set
    path, random samples are taken from this folder
test_set = test_datagen.flow_from_directory('dataset/test_set',
    target_size = (64, 64), batch_size = 32, class_mode = 'binary')
#We train the model
#steps per epoch: number of samples taken per epoch, if the number is
    higher than the samples available, data augmentation is performed
#epochs: one epoch consists in a complete iteration over the whole data
     set
classifier.fit_generator(training_set, steps_per_epoch = 8000,   epochs
    = 25,   validation_data = test_set, validation_steps = 2000)
```

4.2 BoNET simple

Para el siguiente tutorial corto, se implementará una simplificación de la red propuesta en el artículo *Deep learning for automated skeletal bone age assessment in X-ray images* , diseñada para estimar la edad en meses de un sujeto, a partir de la imagen digital de rayos-x de su mano.

La arquitectura BoNET simple implementa los siguientes parámetros:

- **Entrada de la imagen** con 220×220 pixeles.
- **Múltiples capas**: la red implementa 2 capas, donde cada capa está formada por una capa de convolución y otra de *max pooling*. La cantidad de filtros implementada en la capa convolucional
  - La primera capa implementa filtros de tamaño de 7×7 y en la segunda el núcleo es de 5×5 pixeles.
  - La primera capa, la cual funciona como extractor de características de más bajo nivel, debe tener sus pesos congelados, para evitar el problema del gradiente que se desvanece.
  - Por capa, la cantidad de filtros o *mapas* de características es: $96, y\, 1024$.
  - No se cargarán los pesos de otras redes (no habrá *transfer learning*).

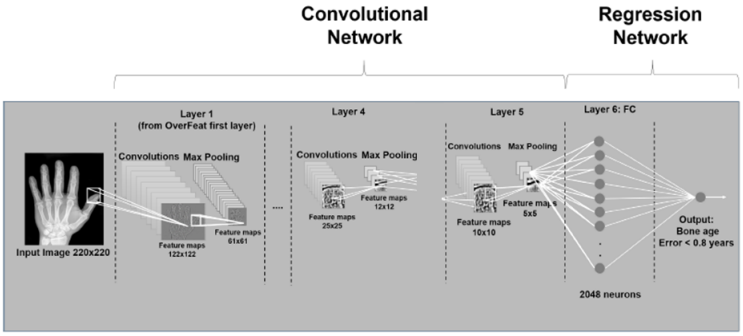

Figure 17: Red BoNET.

Figure 18: Vgg16

  - La capa de max pooling toma el valor máximo cada dos pixeles, reduciendo la dimensionalidad a la mitad.
  - Según lo reportado en artículo, el MAE promedio para los distintos conjuntos es de 1.1 años.

- **Capa completamente conectada:** O también conocida como la *fully connected layer.* Esta red está diseñada para hacer regresión, por lo que a la salida se fija una sola neurona, con una función de activación lineal en su defecto la función identidad. La cantidad de neuronas en la capa oculta es de 2048, según la experiencia de los autores del artículo.

- La métrica para el error es la *Mean Absolute Distance* o *Mean Absolute Error,* correspondiente a $|y - \tilde{y}|$.

- El coeficiente de aprendizaje α está fijado inicialmente con $\alpha = 0.002$ con un decaimiento de 0.0002.

- El factor de momentum es 0.9.

- La cantidad de iteraciones o *epochs* es 150.

4.3 BoNET completa

La arquitectura de BoNET implementa los siguientes parámetros:

- **Entrada de la imagen** con 220×220 pixeles.

- **Múltiples capas**: la red implementa 5 capas, donde cada capa está formada por una capa de convolución y otra de *max pooling.* La cantidad de filtros implementada en la capa convolucional

  - La primera capa implementa filtros de tamaño de 7×7, la segunda el núcleo es de 5×5 pixeles y en las siguientes de 3×3 pixeles.
  - La primera capa, la cual funciona como extractor de características de más bajo nivel, debe tener sus pesos congelados, para evitar el problema del gradiente que se desvanece.
  - Por capa, la cantidad de filtros o *mapas* de características es: $96, 2048, 1024, 1024\, y\, 1024$.
  - La primera capa corresponde a la capa de la red *OverFeat*.
  - La capa de max pooling toma el valor máximo cada dos pixeles, reduciendo la dimensionalidad a la mitad.

- **Capa completamente conectada:** O también conocida como la *fully connected layer.* Esta red está diseñada para hacer regresión, por lo que a la salida se fija una sola neurona, con una función de activación lineal en su defecto la función identidad. La cantidad de neuronas en la capa oculta es de 2048, según la experiencia de los autores del artículo.

- La arquitectura *fuera-del-paquete* que mejor funcionó fue GoogLeNet.

- La métrica para el error es la *Mean Absolute Distance* o *Mean Absolute Error,* correspondiente a $|y - \tilde{y}|$.
- El coeficiente de aprendizaje α está fijado inicialmente con $\alpha = 0.002$ con un decaimiento de 0.0002.
- El factor de momentum es 0.9.
- La cantidad de iteraciones o *epochs* es 150.

4.3.1 Código

Los siguientes son los paquetes a importar.

```
from keras.models import Sequential, Model, load_model
from keras import applications
from keras import optimizers
from keras.layers import Dropout, Flatten, Dense, Convolution2D,
    MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

Generación de semilla y carga de los datos

```
qsub -l nodes=tule-00.cnca vgg16.pbs
```

Para matar un proceso:

```
qdel 16672
```

5 Arquitecturas de redes convolucionales avanzadas

5.1 La Arquitectura LeNet 5

Una de las primeras arquitecturas de redes convolucionales profundas, puede encontrarse en el artículo *Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998* donde se definió la red **LeNet 5**, cuya arquitectura se ilustra en la Figura 19.

Las características básicas de la arquitectura propuesta por Y. LeCun et. al. son las siguientes:

- Sub-muestreo de promediado o *average pooling* en ventanas de 2×2.
- Funciones de activación sigmoidales o tangentes hiperbólicas.
- Capa completamente conectada antes de la salida del modelo.
- Un paso $p = 1$.
- Entrenada y utilizada para la base de datos muestral MNIST de digitos escritos a mano.

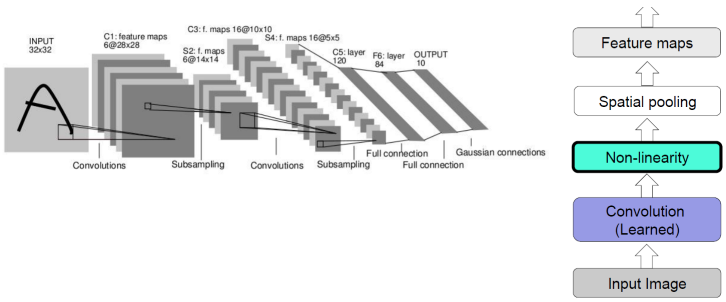

Figure 19: Arquitectura LeNet 5.

- La cantidad de parámetros por capa está dada por:
  - **Capa 1 (C1)**: Compuesta por 6 filtros de 5×5 dimensiones o con tal tamaño de **campo receptivo**, con $(5 \times 5 + 1) \times 6 = 156$ parámetros por aprender. Si hubiese sido completamente conectada, se hubiesen necesitado definir $(32 \times 32 + 1)\,(28 \times 28) \times 6 = 4821600$. Al realizarse la convolución, no se efectúa el *zero padding* de la imagen original, por lo que las dimensiones de los mapas de características o *feature maps* se reducen a 28×28.
    - **Capa 2, S2**: Posterior al aplicar la función no lineal de activación (función sigmoidal o hiperbólica), se realiza un sub-muestreo, aplicando el promediado local en una ventana de 2×2, en celdas disjuntas (no ventanas), lo que resulta en mapas de características de 14×14.
  - **Capa 3 (C3):** Genera 16 mapas de activación con dimensiones de 10×10, con conexiones arbitrarias entre los mapas de características de S2 y la capa C3 . Presenta 1516 parámetros entrenables.
    - **Capa 4, S4**: Después de aplicar la función no lineal de activación se realiza un sub-muestreo, aplicando el promediado local en una ventana de 2×2, en celdas disjuntas (no ventanas), lo que resulta en 16 mapas de características de 5×5.
  - **Capa 5 (C5)**: Esta capa realiza un *aplanado* de los datos de forma *pesada* o ponderada, con una convolución usando 120 mapas de activación de 1×1 dimensiones. Esos mapas de activación están conectados a todos los pixeles de los 16 mapas de 5×5 en la capa anterior. De esta forma, en esta capa se deben definir $120 \times (16 \times (5 \times 5) + 1) = 48120$ pesos.
  - **Capa 6 (F6):** Capa completamente conectada, con 84 unidades que conecta las 120 unidades de la capa anterior, lo que hace necesario que se necesiten definir $84 \times (120 + 1) = 10164$ parámetros.

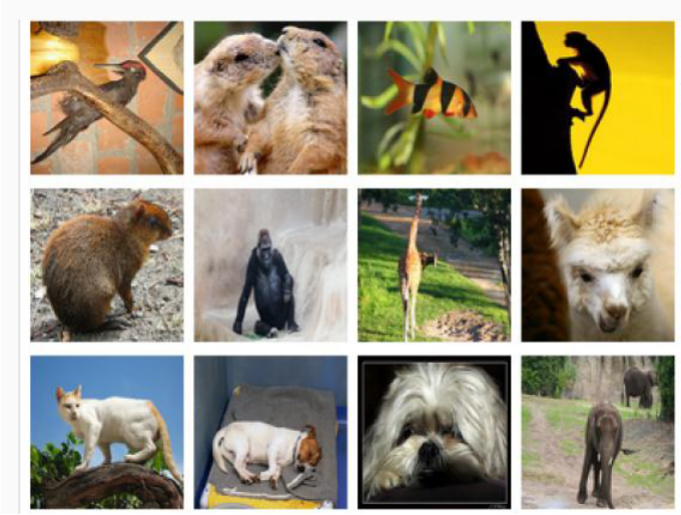

Figure 20: Muestras de ejemplo de ImageNet.

5.2 ImageNet: Competencia de clasificación de imágenes

ImageNet es un repositorio de imágenes con 15 millones de imágenes RGB de resolución variable, 22 mil categorías, recolectadas de la red, y anotadas manualmente con la herramienta de *crowd-sourcing Mechanical Turk*. El repositorio creadas para la competencia anual ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2010 fue la primer edición). En tal competición se utilizan 1000 categorías y 1.2 millones de imágenes de entrenamiento, con 50 000 imágenes de validación y 150 000 imágenes de prueba. Existen dos modos de validación en la competencia:

- El error de cima 1: realizar una estimación de la etiqueta.
- El error de cima 5: Realizar 5 estimaciones de la etiqueta de una imagen.

La Figura 20 presenta algunas muestras de este conjunto de datos.

5.3 La Arquitectura AlexNet

La arquitectura AlexNet fue propuesta en el artículo *ImageNet Classification with Deep Convolutional Neural Networks, por Alex Krizhevsky et. al.* y se ilustra en la Figura 21.

AlexNet presenta las siguientes características:

- Introdujo el uso de las funciones de activación ReLU, las cuales simplificaron el proceso de entrenamiento .
- Se basó en el uso intensivo de aumentado de datos, con rotaciones de imágenes, achicamientos, etc.

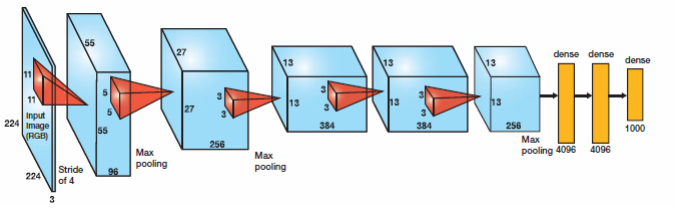

Figure 21: Arquitectura de AlexNet.

- Introdujo el uso del *dropout* en los pesos de la red para mejorar su generalización. El *dropout* como se estudió anteriormente, consiste en *apagar* aleatoriamente pesos en la red, para evitar la sobre-especialización de un campo receptivo a una característica.
- Las capas se definieron como:
  - **Capa 1**: 96 filtros con $k = 11$ de alto y ancho, con un paso $p = 4$, lo que resulta en dimensiones del mapa de activación de $\frac{227-11}{4} + 1 = 55$. La red fue diseñada para procesar imágenes RGB (3 capas), por lo que entonces la cantidad total de parámetros a entrenar en esta capa es de