REDES NEURONALES 2024

Clase 18 parte 2

Jueves 23 de octubre 2024

FAMAF, Universidad Nacional de Córdoba

Instituto de Física Enrique Gaviola (UNC-CONICET)

J02/11/23 c20p1

RED NEURONAL FEED-FORWARD CON UNA CAPA OCULTA

Ya hemos analizado el potencial de un perceptrón simple o de un conjunto de *M* perceptrones simples formando una única capa de salida. Vimos tres casos:

- Perceptrón simple con función de activación binaria, apto para clasificar en dos categorías (presentamos el primer algoritmo de aprendizaje automático).
- Perceptrón simple con función de activación lineal, apto para hacer regresiones lineales y resolver problemas linealmente independientes (muy muy simples).
- Perceptrón simple con función de activación no lineal, apto para hacer regresiones nlogisticas y aprender a resolver problemas un poco más complicados, pero no mucho más complicadas. Aquí aprendimos a definir el *Error Cuadrático Medio* y a minimizarlo, como algoritmo de aprendizaje. Vimos que en el caso no lineal, el problema de encontrar un juego de parámetros (sinapsis y umbrales) que lleguen a resolver el problema definido por el conjunto de entrenamiento p con un pequeño error cuadrático medio, es muy complejo debido a la rugosidad de la función error. muy simplemente el caso lineal, a pesar de que ahora el algoritmo optimizador es mucho más complejo, dada la rugosidad de la función error que

Ya estamos listos para agregarle una cuota más de complejidad a nuestra red, con lo cual podrá aprender problemas más difíciles. Recordemos que estamos analizando el caso de *aprendizaje supervisado*, para lo cual necesitamos un conjunto de entrenamiento *C* que resume la experiencia previa:

$$C = \{(\bar{\xi}^\mu, \bar{\zeta}^\mu)\} \qquad \mu = 1, 2, \ldots, p$$

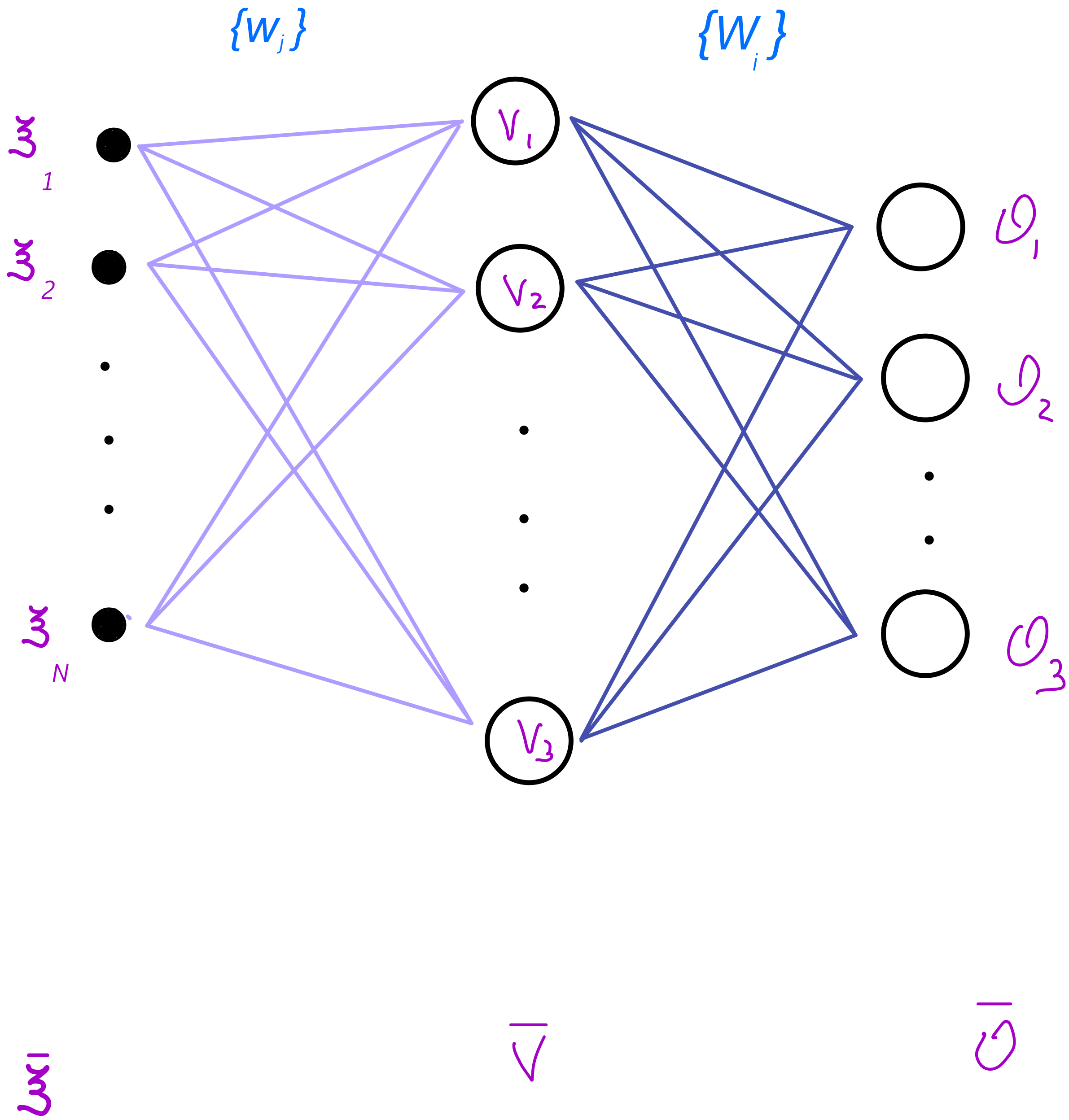
Entradas

Salidas etiquetadas

Nos limitaremos por ahora al caso de regresiones, o sea, de salidas reales. Todas las funciones de activación serán no lineales. Más adelante veremos cómo podemos usar estos regresores para atacar un problema complejo de clasificación, o sea, un problema *no linealmente separable.*

Nos limitaremos también a una arquitectura de red neuronal muy simple y eficiente, que permite tratar problemas sofisticados con mucho éxito. Se trata de las famosas *Redes Neuronales Feed-Forward*, o, en español, redes neuronales alimentadas hacia adelante. Se las llama así porque las neuronas se acomodan en capas y la información fluye secuencialmente de la entrada a la salida. Como el perceptrón simple, tienen una capa de N entradas, que no es en verdad una capa de neuronas (porque solo presentan la información pero no la procesan). Tienen también una capa de M neuronas de salida no-lineales. Entre la entrada y la capa de salida se acomodan capas *ocultas* de varias neuronas cada una. En definitiva tenemos un ordenamiento jerárquico de las neuronas en capas. Lo fundamental en las redes feed-forward es el hecho de que solo se permiten conexiones sinapticas entre las neuronas de una dada capa y las neuronas de la capa inmediatamente siguiente. O sea, no puede haber sinapsis hacia atrás, hacia los costados o incluso saltear capas hacia adelante.

Por razones didácticas, antes de analizar el caso más general de un número arbitrario de capas de neuronas, analizaremos en detalle el caso de una *red feed-forward con una única capa oculta*.



$\{w_j\}$
$\{W_i\}$
ξ_1
ξ_2
ξ_N
V_1
V_2
V_3
O_1
O_2
O_3
$\bar{\xi}$
$\bar{V}$
$\bar{O}$

Este es un diagrama esquemático de una red neuronal feed-forward con N entradas, M salidas y una única capa oculta de L neuronas.

- Usaremos la letra griega ξ_k para indicar la *k*-ésima entrada y reservamos la letra k para nombrarlas.

- Llamaremos V_j a la función de activación de las neuronas de la capa oculta y usaremos la letra j para nombrarlas.

- Llamaremos O_i a las neuronas de la capa de salida y usaremos la letra i para nombrarlas.

Pensamos en términos de vectores:

$$\bar{\xi} \in \mathbb{R}^N$$

$$\bar{V} \in \mathbb{R}^L$$

$$\bar{O} \in \mathbb{R}^M$$

$$\overline{W_i} \in \mathbb{R}^L$$

$$\bar{w}_j \in \mathbb{R}^N$$

función objetivo $\Big\}$ $f: \mathbb{R}^N \longrightarrow \mathbb{R}^M$ $\longleftarrow$ muy complicada y general

El proceso de entrenamiento es similar al visto en el caso de una sola capa de salida. Veamos como fluye la información por la red cuando le presentamos a la red el ejemplo μ del conjunto de entrenamiento.

Las neuronas de la capa oculta calculan sus estímulos:

$$h_j^\mu = \sum_{k=1}^{N} w_{jk}\,\xi_k^\mu = \bar{w}_{j\cdot}\,\bar{\xi}^\mu$$

Notemos que llamamos w_{jk} (minúscula) a la sinapsis entre la entrada k y la neurona j de la capa de entrada.

La salida de las neuronas de la capa oculta será entonces:

$$V_j^\mu = g_1(h_j^\mu) = g_1(\bar{W}_{j\cdot}\,\bar{\xi}^\mu)$$

Con los valores de las neuronas de la capa oculta podemos procesar la información en cada una de las M neuronas de la capa de salida:

$$h_i^\mu = \sum_{j=1}^{L} W_{ij} V_j^\mu = \overline{W}_i \cdot \overline{V}^\mu$$

$$O_i^\mu = g_2(h_i^\mu) = g_2\left(\overline{W}_i \cdot \overline{V}^\mu\right)$$

$$= g_2\left(\sum_{j=1}^{L} W_{ij}\, g_1\left(\sum_{k=1}^{N} w_{jk}\, \xi_k^\mu\right)\right)$$

Ahora que tengo las salidas de la red podemos comparar con el resultado correcto para el elemento μ del conjunto de entrenamiento.

$L \times N \quad w \to$

$L \times M \quad W \to$

$$E[\underbrace{\{\overline{w}\}}_{w\,W}] = \frac{1}{2} \sum_{\mu=1}^{P} \sum_{i=1}^{M} \left[\zeta_i^\mu - O_i^\mu\right]^2$$

$$= \frac{1}{2} \sum_{\mu=1}^{P} \sum_{i=1}^{M} \left[\zeta_i^\mu - g_2\left(\sum_{j=1}^{L} W_{ij}\, g_1\left(\sum_{k=1}^{N} w_{jk}\, \xi_k^\mu\right)\right)\right]^2$$

Una vez que tenemos la expresión funcional de la ECM en función de todas las componentes de los vectores $\overline{W}_i$ y $\overline{w}_j$, vamos a aplicar el descenso por el gradiente de una forma muy específica.

Primero corregimos los pesos sinápticos $\overline{W}_i$ (mayúscula) entre la capa oculta y la capa de salida sin importarnos por los $\overline{w}_j$ (minúscula).

$$W_{ij}^{nuevo} = W_{ij}^{anterior} + \Delta W_{ij}$$

donde:

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}}$$

$$= \eta \sum_{\mu=1}^{P} \frac{\cancel{2}}{\cancel{2}} \left[\zeta_i^\mu - O_i^\mu \right] g_2'(h_i^\mu)\, V_j^\mu$$

$$= \eta \sum_{\mu=1}^{P} \delta_i^\mu V_j^\mu$$

y

$$\delta_i^\mu = g_2'(h_i^\mu) \left[\zeta_i^\mu - O_i^\mu \right]$$

Una vez que hemos actualizado todos los parámetros sinápticos de la última capa, pasamos a hacer lo mismo con los parámetros que van de la entrada a la capa oculta, o sea, los $\overline{w}_j$ minúsculos. Para eso calculamos estas componentes del gradiente:

$$w_{jk}^{nuevo} = w_{jk}^{viejo} + \Delta w_{jk}$$

donde

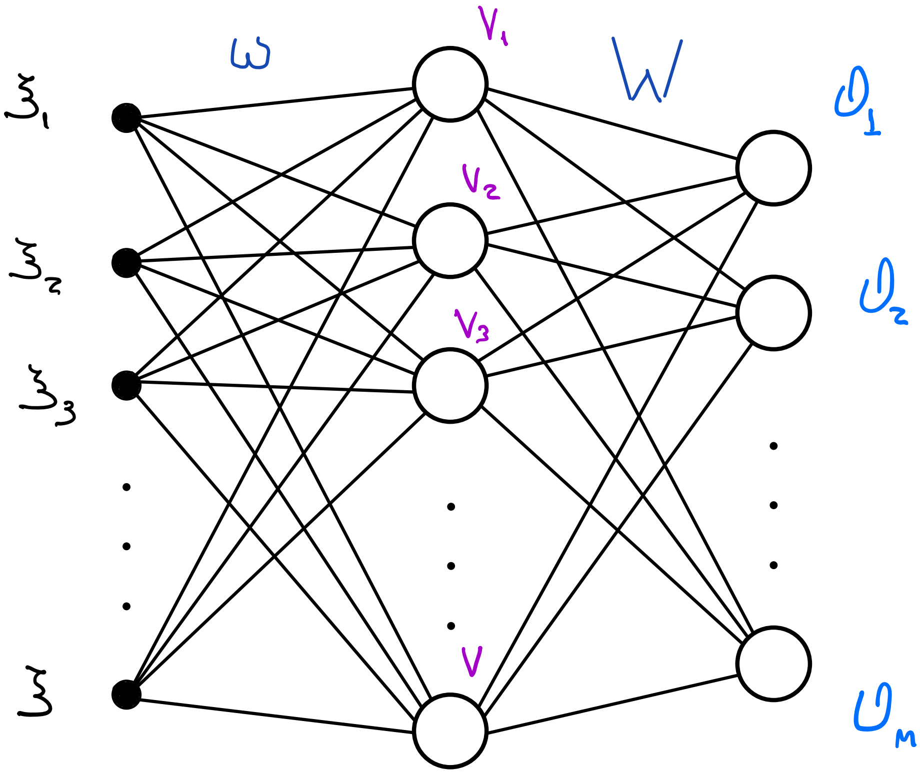
$$\Delta w_{jk} = - \eta \frac{\partial E}{\partial w_{jk}}$$

$$= \eta \sum_{\mu=1}^{P} \sum_{i} \left[\zeta_i^\mu - O_i^\mu \right] g_2'(h_i^\mu) W_{ij} \, g_1'(h_j^\mu) \, \xi_k^\mu$$

$$= \eta \sum_{\mu=1}^{P} \sum_{i} \delta_i^\mu W_{ij} \, g_1'(h_j^\mu) \, \xi_k^\mu$$

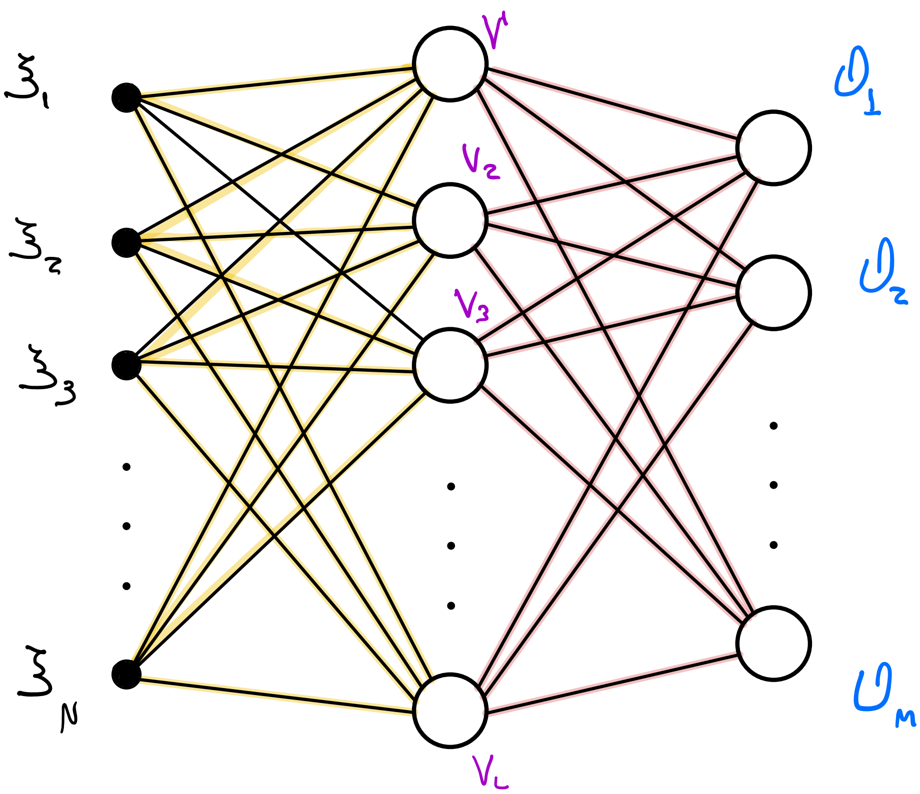
$$= \eta \sum_{\mu=1}^{P} \delta_j^\mu \, \xi_k^\mu$$

con $$\delta_j^\mu = g_1'(h_j^\mu) \sum_i W_{ij} \, \delta_i^\mu$$



$$\vec{\xi}^{\mu} \longrightarrow \vec{V}^{\mu} \longrightarrow \vec{O}^{\mu} \longrightarrow E$$

$$\xleftarrow[k=1,2,\dots,L]{\{\vec{\omega}_k\}} \vec{\delta}^{\mu} \xleftarrow[i=1,2,\dots,M]{\{\vec{W}_i\}} \vec{\delta}^{\mu} \longleftarrow E$$



¿Cuántos parámetros tenemos?

$$(N \times L) + (L \times M) = L \times (N + M)$$

Este método se llama *back-propagation* o retro-propagación. Noten que le mostramos el elemento μ del conjunto de entrenamiento a la red, la información viaja hacia adelante, en dirección a la salida. Con los resultados de las M salidas calculamos el error. Con el error primero calculamos las correcciones a los pesos sinápticos $\overline{W}_i$ entre la capa oculta y la capa de salida. Una vez actualizados estos parámetros sinápticos pasamos a corregir los pesos sinápticos $\overline{w}_j$ entre la entrada y la capa oculta.

El algoritmo **ÉPOCA**

A. Sea $\mu = 1$

B. Mientras $(\mu \leq p)$ repetimos

1. Con $\bar{\xi}^{\mu}$ calculamos $\bar{\boldsymbol{V}}^{\mu}$

2. Con $\bar{\boldsymbol{V}}^{\mu}$ calculamos $\bar{\boldsymbol{O}}^{\mu}$

3. Con $\bar{\boldsymbol{O}}^{\mu}$ y $\bar{\xi}^{\mu}$ calculamos E

4. Con E calculamos $\bar{\delta}_i^{\mu}$

5. Con $\bar{\delta}_i^{\mu}$ calculamos los $\Delta \bar{\boldsymbol{W}}$ y los acumulamos

$$\Delta \overline{W} = \Delta \overline{W} + \Delta \overline{W}^{\mu}$$

6. Con $\bar{\delta}_i^{\mu}$ calculamos los $\bar{\delta}_j$

7. Con $\bar{\delta}_j$ actualizamos los $\Delta \bar{\boldsymbol{w}}$ y los acumulamos

$$\Delta \bar{w} = \Delta \bar{w} + \Delta \bar{w}^{\mu}$$

8. Pasamos al próximo ejemplo $\mu = \mu + 1$

9. Si $\mu = p$ actualizamos los $\bar{\boldsymbol{W}}_i$ y $\bar{\boldsymbol{w}}_j$ y volvemos a **B**

Leemos los datos:

- Conjunto de entrenamiento $(\bar{\xi}^{\mu}, \bar{\zeta}^{\mu})$
- Tolerancia *tol*
- Razón de aprendizaje η

Sea t=1

Llamamos a la rutina ***ÉPOCA*** que devuelve E_{trrain}, $\overline{\boldsymbol{W}}$ y $\overline{\boldsymbol{w}}$

Con $\overline{\boldsymbol{W}}$ y $\overline{\boldsymbol{w}}$ calculamos E_{test}

Si $E_{test} < tol$ paramos si no hacemos $t = t+1$

Hasta acá supusimos que actualizamos los acoplamientos sinápticos $\overline{\boldsymbol{W}}$ y $\overline{\boldsymbol{w}}$ después de mostrarle a la red todos los ejemplos del conjunto de entrenamiento, o sea después de una época. Esta forma de actualizar se conoce como *actualización en lote (batch)*.

$$\Delta W_{ij} = \overbrace{\Delta W_{ij}^{(1)} + \Delta W_{ij}^{(2)} + \cdots\cdots + \Delta W_{ij}^{(p)}}^{\mu}$$

$$\Delta w_{jk} = \overbrace{\Delta w_{jk}^{(1)} + \Delta w_{jk}^{(2)} + \cdots\cdots + \Delta w_{jk}^{(p)}}^{\mu}$$

Otra posibilidad que no hemos analizado es actualizar todos los pesos sinápticos $\overline{\boldsymbol{W}}_i$ y $\overline{\boldsymbol{w}}_j$ después de presentarle cada ejemplo del conjunto de entrenamiento a la red. Este método se denomina *actualización en línea (on line)*. Es una actualización más fina y precisa pero requiere mucho más cálculo numérico.

Veremos pronto que la forma más adecuada es una actualización en *mini-lotes (mini-batch)*, o sea, algo intermedio entre los métodos en lote y en línea.