Introducción a los modelos computacionales

Práctica 1. Implementación del perceptrón multicapa

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura "Introducción a los modelos computacionales"
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

20 de septiembre de 2018

Objetivos de la práctica

- Familiarizar al alumno con los modelos computacionales de redes neuronales, en concreto, con el perceptrón multicapa.
- Implementar el algoritmo de retropropagación básico para el perceptrón multicapa.
- Comprobar el efecto de distintos parámetros:
  - Arquitectura de la red.
  - Factor de momento.
  - Uso de parada rápida mediante conjunto de validación
  - etc.

Algoritmo de retropropagación

- Notación:
  - Patrones de entrenamiento:
    - Vector de entradas: $\mathbf{x} = (x_1, \ldots, x_k)$.
    - Vector de salidas deseadas: $\mathbf{d} = (d_1, \ldots, d_J)$.
  - Arquitectura de la red: $\{n_0 : n_1 : \ldots : n_H\}$
    - n_h es el número de neuronas de la capa h.
    - $n_0 = k$, $n_H = J$.
    - $H - 1$ capas ocultas.
  - Pesos de la red. Para cada capa h, sin contar la capa de entrada:
    - Matriz con un vector de pesos de entrada por cada neurona: $\mathbf{W}^h = (\mathbf{w}_1^h, \ldots, \mathbf{w}_{n_h}^h)$.
    - Vector de pesos de la neurona j de la capa h (incluye sesgo): $\mathbf{w}_j^h = (w_{j0}^h, w_{j1}^h, \ldots, w_{jn_{(h-1)}}^h)$.
  - Salida de la red: $\mathbf{o} = (o_1, \ldots, o_J)$.

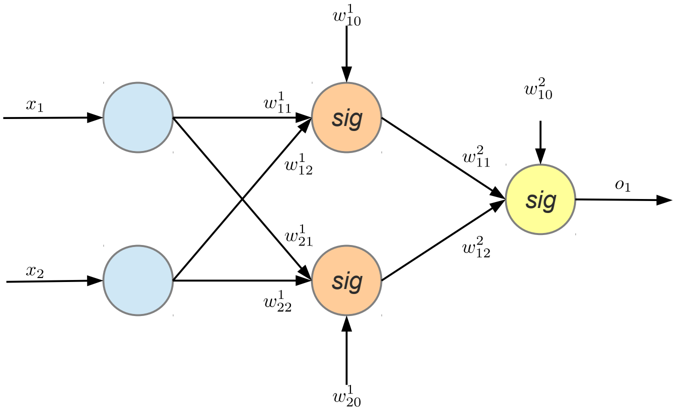
Algoritmo de retropropagación

- Vamos a considerar problemas de predicción (regresión) con una o más variables a predecir.
- Vamos a considerar que todas las neuronas, salvo las de la capa de entrada, serán de tipo sigmoide.

$$out_j^h = \frac{1}{1 + \exp(-w_{j0}^h - \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1})}$$

Algoritmo de retropropagación

- Vamos a calcular las derivadas para un ejemplo simple y luego veremos como se calculan de forma general.



Algoritmo de retropropagación

Fase 1: Propagación hacia delante.

- Llamamos out_j^h a la salida de la neurona j en la capa h.
- Dados dos valores x_1 y x_2 de entrada, calcular la salida de cada neurona.
    - Primera capa:

$$out_1^0 = x_1; out_2^0 = x_2$$

    - Segunda capa:

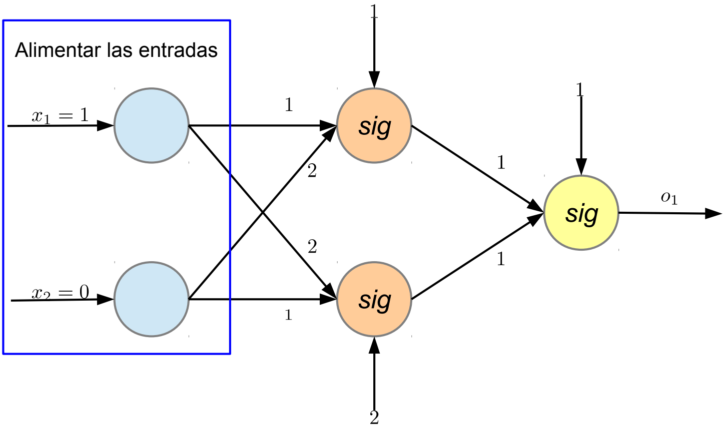
$$out_1^1 = \sigma(w_{10}^1 + w_{11}^1 out_1^0 + w_{12}^1 out_2^0) = \sigma(w_{10}^1 + w_{11}^1 x_1 + w_{12}^1 x_2);$$
$$out_2^1 = \sigma(w_{20}^1 + w_{21}^1 out_2^0 + w_{11}^1 out_2^0) = \sigma(w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2);$$

    - Tercera capa:

$$out_1^2 = o_1 = \sigma(w_{10}^2 + w_{11}^2 out_1^1 + w_{12}^2 out_2^1)$$

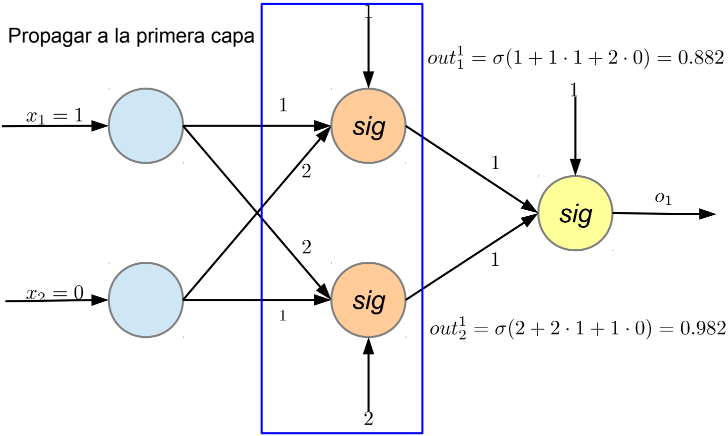
Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



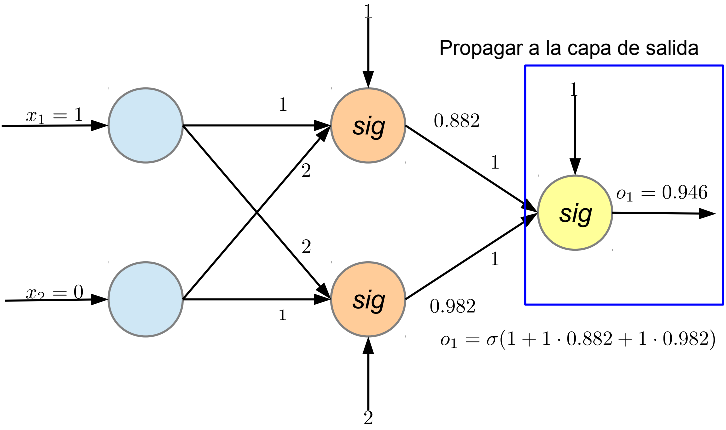
Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



Algoritmo de retropropagación

Fase 1: Propagación hacia delante.



Algoritmo de retropropagación

Fase 2: Cálculo del error y de las derivadas.

- Obtenemos el valor de error cometido por la red:

$$E = (d_1 - o_1)^2$$

- Ahora derivamos ese error respecto a cada uno de los pesos:

$$\nabla E = \left\{ \frac{\partial E}{\partial w_{10}^1}, \frac{\partial E}{\partial w_{11}^1}, \frac{\partial E}{\partial w_{12}^1}, \frac{\partial E}{\partial w_{20}^1}, \frac{\partial E}{\partial w_{21}^1}, \frac{\partial E}{\partial w_{22}^1}, \frac{\partial E}{\partial w_{10}^2}, \frac{\partial E}{\partial w_{11}^2}, \frac{\partial E}{\partial w_{12}^2} \right\}$$

- De la expresión $(d_1 - o_1)^2$, los pesos solo influyen en o_1 (o_1 es una función de cada uno de los pesos). En estos casos, la regla de la cadena nos permite realizar estas derivadas de forma recursiva (lo siguiente se cumple para todos los pesos):

$$\frac{\partial E}{\partial w_{10}^2} = \frac{\partial E}{\partial o_1} \frac{\partial o_1}{\partial w_{10}^2} = -2(d_1 - o_1) \frac{\partial o_1}{\partial w_{10}^2}$$

Algoritmo de retropropagación

- Llamamos net_j^h a la suma ponderada de las entradas de la neurona j en la capa h, es decir, a la salida antes de aplicar la función de activación sigmoide:

$$out_j^h = \sigma(net_j^h)$$

Algoritmo de retropropagación

- Recordamos:

$$o_1 = \sigma(net_1^2)$$
$$\sigma(x)' = \sigma(x)(1 - \sigma(x))$$

- Continuamos derivando con respecto a o_1:

$$\frac{\partial E}{\partial w_{10}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{10}^2} = -2(d_1 - o_1) \cdot o_1(1 - o_1)\frac{\partial net_1^2}{\partial w_{10}^2}$$
$$\frac{\partial E}{\partial w_{11}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{11}^2} = -2(d_1 - o_1) \cdot o_1(1 - o_1)\frac{\partial net_1^2}{\partial w_{11}^2}$$
$$\frac{\partial E}{\partial w_{12}^2} = -2(d_1 - o_1)\frac{\partial o_1}{\partial w_{12}^2} = -2(d_1 - o_1) \cdot o_1(1 - o_1)\frac{\partial net_1^2}{\partial w_{12}^2}$$

Algoritmo de retropropagación

- Y ahora ya podemos escribir las derivadas completas para los pesos de la capa de salida (w_{1j}^2):

$$
\begin{aligned}
net_1^2 &= w_{10}^2 + w_{11}^2 out_1^1 + w_{12}^2 out_2^1 \\
\frac{\partial E}{\partial w_{10}^2} &= -2(d_1 - o_1)o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{10}^2} = \\
&= -2(d_1 - o_1)o_1(1-o_1)1 \\
\frac{\partial E}{\partial w_{11}^2} &= -2(d_1 - o_1)o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{11}^2} = \\
&= -2(d_1 - o_1)o_1(1-o_1)out_1^1 \\
\frac{\partial E}{\partial w_{12}^2} &= -2(d_1 - o_1)o_1(1-o_1)\frac{\partial net_1^2}{\partial w_{12}^2} = \\
&= -2(d_1 - o_1)o_1(1-o_1)out_2^1
\end{aligned}
$$

Algoritmo de retropropagación

- Pasamos a analizar las derivadas de los pesos de la capa oculta (w^1_{1i} y w^1_{2i}):
  - Los pesos w^1_{1i} influyen en out^1_1.
  - Los pesos w^1_{2i} influyen en out^1_2.
- Por tanto, su derivada será similar a las otras derivadas, pero cuando lleguemos a $\frac{\partial net^2_1}{\partial w^1_{ji}}$ tendremos que continuar derivando.
- Para el peso w^1_{10} (sesgo de la primera neurona en la primera capa):

$$
\begin{aligned}
net^2_1 &= \quad w^2_{10} + w^2_{11}\color{red}{out^1_1}\color{black} + w^2_{12}out^1_2 \\
\frac{\partial E}{\partial w^1_{10}} &= -2(d_1 - o_1)o_1(1-o_1)\frac{\partial net^2_1}{\partial w^2_{10}} = \\
&= -2(d_1 - o_1)o_1(1-o_1)\color{red}{w^2_{11}\frac{\partial out^1_1}{\partial w^2_{10}}}
\end{aligned}
$$

Algoritmo de retropropagación

- Recordamos:

$$out_1^1 = \sigma(net_1^1)$$
$$net_1^1 = w_{10}^1 + w_{11}^1 x_1 + w_{12}^1 x_2$$
$$\sigma(x)' = \sigma(x)(1 - \sigma(x))$$

- Continuamos derivando con respecto a out_1^1:

$$\frac{\partial E}{\partial w_{10}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 \frac{\partial out_1^1}{\partial w_{10}^2} =$$
$$= -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)\frac{\partial net_1^1}{\partial w_{10}^1} =$$
$$= -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)1$$

Algoritmo de retropropagación

- Repetimos este proceso para todos los pesos de capa oculta:

$$\frac{\partial E}{\partial w_{10}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)1$$
$$\frac{\partial E}{\partial w_{11}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)x_1$$
$$\frac{\partial E}{\partial w_{12}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{11}^2 out_1^1(1 - out_1^1)x_2$$
$$\frac{\partial E}{\partial w_{20}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)1$$
$$\frac{\partial E}{\partial w_{21}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)x_1$$
$$\frac{\partial E}{\partial w_{22}^1} = -2(d_1 - o_1)o_1(1 - o_1)w_{12}^2 out_2^1(1 - out_2^1)x_2$$

Algoritmo de retropropagación

- Recapitulando:
  - Capa de salida:

$$\frac{\partial E}{\partial w_{ji}^{2}} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)out_i^{1}, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^{1}} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^{2}out_j^{1}(1 - out_j^{1})1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^{2}out_j^{1}(1 - out_j^{1})x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

- Ojo, hay partes comunes:
  - Capa de salida:

$$\frac{\partial E}{\partial w_{ji}^2} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)out_i^1, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^1} = \begin{cases} -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)1, & \text{si } i = 0, \\ -2(d_1 - o_1)o_1(1 - o_1)w_{1j}^2 out_j^1(1 - out_j^1)x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

- Muchas partes son comunes, el cálculo de derivadas se puede realizar de manera recursiva.
- Llamamos δ_j^h a la derivada de la neurona j de la capa oculta h con respecto al error ("cuánta culpa tiene sobre el error esa neurona").

$$\begin{aligned}
\delta_1^2 &= -2(d_1 - o_1)o_1(1 - o_1) \\
\delta_1^1 &= w_{11}^2 \delta_1^2 out_1^1 (1 - out_1^1) \\
\delta_2^1 &= w_{12}^2 \delta_1^2 out_2^1 (1 - out_2^1)
\end{aligned}$$

- A efectos de la actualización de pesos, la constante (2) puede obviarse.

Algoritmo de retropropagación

- Redefinimos las derivadas en función de estos valores δ_j^h:
  - Capa de salida:

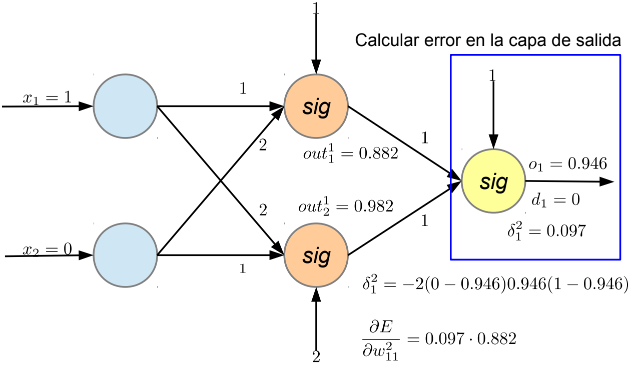
$$\frac{\partial E}{\partial w_{ji}^2} = \begin{cases} \delta_j^2 1, & \text{si } i = 0, \\ \delta_j^2 out_i^1, & \text{si } i \neq 0. \end{cases}$$

  - Capa oculta:

$$\frac{\partial E}{\partial w_{ji}^1} = \begin{cases} \delta_j^1 1, & \text{si } i = 0, \\ \delta_j^1 x_i, & \text{si } i \neq 0. \end{cases}$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.



$$\delta_1^2 = -2(0 - 0.946)0.946(1 - 0.946)$$

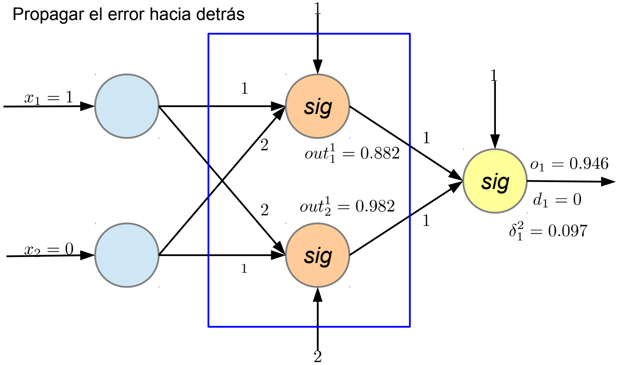
$$\frac{\partial E}{\partial w_{11}^2} = 0.097 \cdot 0.882$$

$$\frac{\partial E}{\partial w_{12}^2} = 0.097 \cdot 0.982$$

$$\frac{\partial E}{\partial w_{10}^2} = 0.097 \cdot 1$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.



$$\delta^1_1 = 1 \cdot 0.097 \cdot 0.882 \cdot (1 - 0.882) = 0.0101$$
$$\delta^1_2 = 1 \cdot 0.097 \cdot 0.982 \cdot (1 - 0.982) = 0.0017$$

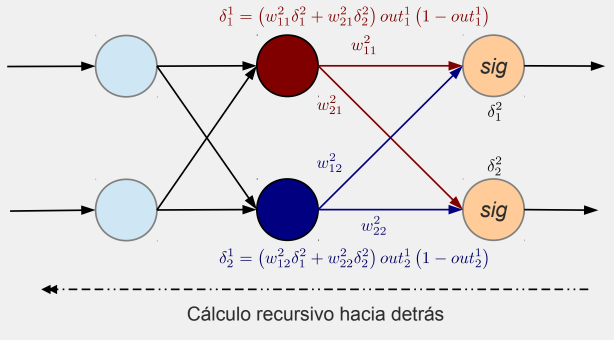
$$\frac{\partial E}{\partial w^1_{10}} = 0.0101 \cdot 1 \quad \frac{\partial E}{\partial w^1_{11}} = 0.0101 \cdot 1 \quad \frac{\partial E}{\partial w^1_{12}} = 0.0101 \cdot 0$$
$$\frac{\partial E}{\partial w^1_{20}} = 0.0017 \cdot 1 \quad \frac{\partial E}{\partial w^1_{21}} = 0.0017 \cdot 1 \quad \frac{\partial E}{\partial w^1_{22}} = 0.0017 \cdot 0$$

Algoritmo de retropropagación

Fase 2: Propagación hacia detrás.

Si hay varias neuronas en la siguiente capa, cada δ_j^h recibe su valor a partir de las neuronas con las que esté conectado:

$$\delta_j^h \leftarrow \left(\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1} \delta_i^{h+1} \right) \cdot out_j^h \cdot (1 - out_j^h)$$



Algoritmo de retropropagación

Fase 3: Actualización de pesos.

- Una vez hemos obtenido el vector gradiente, debemos actualizar los pesos.
- Se utiliza el propio valor de la derivada (sobre todo por su signo), multiplicado por una constante *eta* (η) que controla que los pasos que se van dando no sean muy pequeños o muy grandes (tasa de aprendizaje).
- Fórmula general:

$$w_{ji}^h = w_{ji}^h - \eta \Delta w_{ji}^h$$

$$\Delta w_{ji}^h = \frac{\partial E}{\partial w_{ji}^h} = \begin{cases} \delta_j^h \cdot 1, & \text{si } i = 0 \\ \delta_j^h \cdot out_i^{h-1}, & \text{si } i \neq 0 \end{cases}$$

Algoritmo de retropropagación

Fase 3: Actualización de pesos.

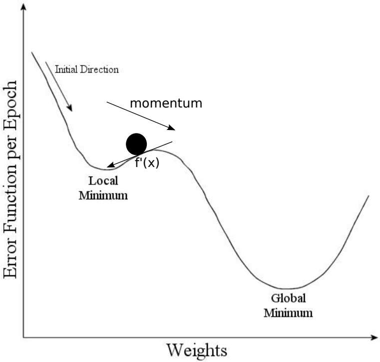
- Ajustar el valor de η es difícil:
  - Si es demasiado grande, podemos provocar oscilaciones.
  - Si es demasiado pequeño, necesitaremos muchas iteraciones.
- Utilizaremos el concepto de momento, para mejorar la convergencia:
  - Los cambios anteriores deberían influir en la dirección de movimiento actual:

$$w_{ji}^{h} = w_{ji}^{h} - \eta \Delta w_{ji}^{h} - \mu \left(\eta \Delta w_{ji}^{h}(t-1) \right)$$

  - Así, los pesos que empiezan a moverse en una determinada dirección, tienden a moverse en ese dirección.
  - La constante *mu* (μ) controla el efecto del momento.

Algoritmo de retropropagación

- La idea es que en un ejemplo como el siguiente, el momento haga que se pueda escapar del óptimo local.
- Pese a que la derivada te diga que vayas a la izquierda, la "*inercia*" puede llevarte a seguir para la derecha:



Algoritmo de retropropagación

- Hasta ahora, hemos visto como ajustar los pesos según el error cometido en un solo patrón.
- Existen diferentes formas de adaptar la red de acuerdo con la forma en que se use la información de entrenamiento (patrones).
    - Aprendizaje Off-line (*batch*):
        - Cada vez que actualizamos los parámetros, consideramos todos los patrones de entrenamiento.
        - Usualmente llamado época (*epoch*).
        - Costoso si hay muchos datos
    - Aprendizaje On-line:
        - Por cada patrón de entrenamiento, realizamos una actualización de pesos.
        - Problema de "olvido" de patrones "viejos".
    - Métodos intermedios $\Rightarrow$ adaptación cada p patrones (*mini batches*).

Algoritmo de retropropagación

Algoritmo de retropropagación *on-line*

Inicio

1. $w_{ji}^{h} \leftarrow U[-1,1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
    1. **Para** cada patrón con entradas **x**, y salidas **d**
        1. $\Delta w_{ji}^{h} \leftarrow 0$ *// Se aplicarán cambios por cada patrón*
        2. $out_{j}^{0} \leftarrow x_j$ *// Alimentar entradas*
        3. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
        4. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
        5. acumularCambio() *// Calcular ajuste de pesos*
        6. ajustarPesos() *// Aplicar el ajuste calculado*

    Fin Para

    Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Algoritmo de retropropagación

Algoritmo de retropropagación *off-line*

Inicio

1. $w_{ji}^{h} \leftarrow U[-1,1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
    1. $\Delta w_{ji}^{h} \leftarrow 0$ *// Se aplicarán cambios al final*
    2. **Para** cada patrón con entradas **x**, y salidas **d**
        1. $out_j^0 \leftarrow x_j$ *// Alimentar entradas*
        2. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
        3. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
        4. acumularCambio() *// Acumular ajuste de pesos*

       Fin Para
    3. ajustarPesos() *// Aplicar el ajuste calculado*

   Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Pseudocódigo del algoritmo a implementar en la práctica:

Algoritmo de retropropagación *on-line*

Inicio

1. $w_{ji}^{h} \leftarrow U[-1, 1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
    1. **Para** cada patrón con entradas **x**, y salidas **d**
        1. $\Delta w_{ji}^{h} \leftarrow 0$ *// Se aplicarán cambios por cada patrón*
        2. $out_{j}^{0} \leftarrow x_j$ *// Alimentar entradas*
        3. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
        4. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
        5. acumularCambio() *// Calcular ajuste de pesos*
        6. ajustarPesos() *// Aplicar el ajuste calculado*

    Fin Para

   Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Pseudocódigo de la versión *off-line* (no se pide):

Algoritmo de retropropagación *off-line*

Inicio

1. $w_{ji}^{h} \leftarrow U[-1,1]$ *// Aleatorios entre -1 y $+1$*
2. **Repetir**
    1. $\Delta w_{ji}^{h} \leftarrow 0$ *// Se aplicarán cambios al final*
    2. **Para** cada patrón con entradas **x**, y salidas **d**
        1. $out_j^0 \leftarrow x_j$ *// Alimentar entradas*
        2. propagarEntradas() *// Propagar las entradas ($\Rightarrow\Rightarrow$)*
        3. retropropagarError() *// Retropropagar el error ($\Leftarrow\Leftarrow$)*
        4. acumularCambio() *// Acumular ajuste de pesos*

       Fin Para
    3. ajustarPesos() *// Aplicar el ajuste calculado*

   Hasta (CondicionParada)
3. **Devolver** matrices de pesos.

Fin

Algoritmo de retropropagación: funciones

propagarEntradas()

Inicio

1. **Para** h de 1 a H *// Para cada capa (⇒⇒)*
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. $net_j^h \leftarrow w_{j0}^h + \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1}$
      2. $out_j^h \leftarrow \sigma\left(net_j^h\right)$

      Fin Para

   Fin Para

Fin

Algoritmo de retropropagación: funciones

retropropagarError()

Inicio

1. **Para** j de 1 a n_H *// Para cada neurona de salida*
   1. $\delta_j^H \leftarrow -\left(d_j - out_j^H\right) \cdot out_j^H \cdot \left(1 - out_j^H\right)$ *// Hemos obviado la constante (2), ya que el resultado será el mismo*

   Fin Para
2. **Para** h de $H-1$ a 1 *// Para cada capa ($\Leftarrow\Leftarrow$)*
   1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
      1. $\delta_j^h \leftarrow \left(\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1}\delta_i^{h+1}\right) \cdot out_j^h \cdot \left(1 - out_j^h\right)$ *// Pasa por todas las neuronas de la capa $h+1$ conectadas con j*

      Fin Para

   Fin Para

Fin

Algoritmo de retropropagación: funciones

acumularCambio()

Inicio

1. **Para** h de 1 a H *// Para cada capa ($\Rightarrow\Rightarrow$)*
    1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
        1. **Para** i de 1 a n_{h-1} *// Para cada neurona de la capa $h-1$*
        $\Delta w_{ji}^h \leftarrow \Delta w_{ji}^h + \delta_j^h \cdot out_i^{h-1}$
        Fin Para
        2. $\Delta w_{j0}^h \leftarrow \Delta w_{j0}^h + \delta_j^h \cdot 1$ *// Sesgo*

    Fin Para

Fin Para

Fin

Algoritmo de retropropagación: funciones

<u>ajustarPesos()</u>

Inicio

1. **Para** h de 1 a H *// Para cada capa ($\Rightarrow\Rightarrow$)*
    1. **Para** j de 1 a n_h *// Para cada neurona de la capa h*
        1. **Para** i de 1 a n_{h-1} *// Para cada neurona de la capa $h-1$*
           $w_{ji}^h \leftarrow w_{ji}^h - \eta\Delta w_{ji}^h - \mu\left(\eta\Delta w_{ji}^h(t-1)\right)$
           Fin Para
        2. $w_{j0}^h \leftarrow w_{j0}^h - \eta\Delta w_{j0}^h - \mu\left(\eta\Delta w_{j0}^h(t-1)\right)$ *// Sesgo*

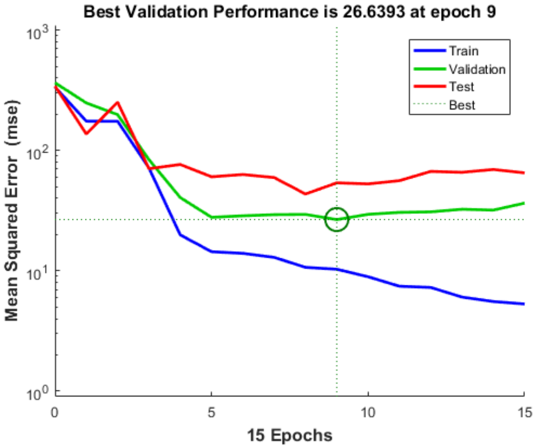
       Fin Para

   Fin Para

Fin

Condición de parada

Early stopping: mecanismo para detectar cuando se sobre-entrena.



Condición de parada

Early stopping: mecanismo para detectar cuando se sobre-entrena.

1. Dividimos los datos en tres partes: entrenamiento (p.ej. 60 %), validación (p.ej. 20 %) y test (p.ej. 20 %).
2. Ajustamos los pesos con el conjunto de entrenamiento.
3. Evaluamos la red en el conjunto de validación.
4. Si el error de validación baja en un valor mayor que t (tolerancia), volvemos al paso 2. En caso contrario, paramos el entrenamiento.
5. Evaluar el modelo final sobre el conjunto de test.

Condición de parada

- Versión estándar, el algoritmo para si:
  - El error de entrenamiento no baja más de 0,00001 o sube, durante 50 iteraciones (bucle externo).
- Versión con validación, el algoritmo para si:
  - El error de entrenamiento no baja más de 0,00001 o sube, durante 50 iteraciones (bucle externo).
  - El error de validación no baja más de 0,00001 o sube, durante 50 iteraciones (bucle externo).

Decremento de tasa de aprendizaje por capas

- Es interesante incorporar una tasa de aprendizaje distinta para cada capa.
- Por ser más sensibles, la tasa de aprendizaje puede ser más pequeña conforme la capa es más cercana a la capa de entrada.
- Se puede hacer con la siguiente ecuación:

$$\eta_h = F^{-(H-h)}\eta, h \in \{1, \ldots, H\}$$

siendo F un factor de decremento establecido por el usuario y η la tasa de aprendizaje original.

Decremento de tasa de aprendizaje por capas

- Para $H = 2$ y $F = 2$:

$$\eta_1 = 2^{-(2-1)}\eta = 2^{(-1)}\eta = \frac{\eta}{2}$$
$$\eta_2 = 2^{-(2-2)}\eta = 2^{(0)}\eta = \eta$$

- Para $H = 3$ y $F = 2$:

$$\eta_1 = 2^{-(3-1)}\eta = 2^{(-2)}\eta = \frac{\eta}{4}$$
$$\eta_2 = 2^{-(3-2)}\eta = 2^{(-1)}\eta = \frac{\eta}{2}$$
$$\eta_3 = 2^{-(3-3)}\eta = 2^{(0)}\eta = \eta$$

Introducción a los modelos computacionales

Práctica 1. Implementación del perceptrón multicapa

Pedro Antonio Gutiérrez
pagutierrez@uco.es

Asignatura "Introducción a los modelos computacionales"
4º Curso Grado en Ingeniería Informática
Especialidad Computación
Escuela Politécnica Superior
(Universidad de Córdoba)

20 de septiembre de 2018