Introducción a los modelos computacionales

Práctica 2. Perceptrón multicapa para problemas de clasificación

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)

8 de octubre de 2019

1. Contenidos
2. Introducción
3. Adaptación del perceptrón a problemas de clasificación
   - Representación de los valores de clase
   - Función softmax
   - Función de rendimiento y error
4. Resumen final

Objetivos de la práctica

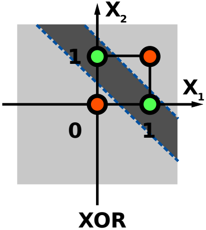
- Implementar la versión *off-line* del algoritmo de retropropagación básico para el perceptrón multicapa.
- Adaptar la formulación para problemas de clasificación mediante una interpretación probabilística de las salidas (función *softmax*).
- Utilizar una función de error probabilística para el entrenamiento de la red (entropía cruzada).
- Comprobar si estas modificaciones mejoran los resultados.

Clasificación

- Motivación: A menudo nos encontramos problemas del mundo real donde el objetivo es "*categorizar*" o "*clasificar*" según un conjunto de características.
- Necesitamos un modelo predictivo que, a partir de una base de datos, sea capaz de obtener el valor de una variable categórica o nominal.
- Por ejemplo:
    - Color de ojos: $\{\text{azul}, \text{verde}, \text{marron}\}$.
    - Éxito o fracaso de un tratamiento: $\{\text{si}, \text{no}\}$.
    - Presencia de cáncer en un órgano fotografiado: $\{\text{si}, \text{no}\}$.

Clasificación

- El problema del XOR también es un problema de clasificación:



- En realidad, la clasificación es una tarea intrínsecamente no lineal porque intentamos poner cosas que no son iguales en el mismo grupo, es decir, una diferencia es el vector de entradas no provoca una diferencia en la salida del modelo.

Representación de los valores de clase

- Representación de los valores de clase (color de ojos):

Tipo	Clase azul	Clase verde	Clase marrón
Valores enteros	1	2	3

- Utilizando esta representación, podríamos emplear el perceptrón multicapa de regresión, de forma que la clase predicha sería el entero más cercano al valor predicho por el modelo.
- Inconvenientes:
  - Se ha asumido que existe un orden entre las clases.
  - Se ha asumido una distancia entre cada una de las clases.

Representación de los valores de clase

- Representación 1-de-J, donde J es el número de clases.
  - Por cada patrón, tendremos un vector de J elementos, donde el elemento i-ésimo será igual a 1 si el patrón pertenece a esa clase y a 0 si no pertenece.
  - Es decir, el vector contendrá 0 en todas las posiciones menos en la posición que corresponde a la clase correcta (en la que habrá un 1).
- Representación de los valores de clase (color de ojos):

Tipo	Clase azul	Clase verde	Clase marrón
1-de-k	$\{1,0,0\}$	$\{0,1,0\}$	$\{0,0,1\}$

- Utilizando esta representación, el perceptrón multicapa modelará cada una de las J variables binarias por separado.
- 1 neurona de salida por clase (modelar tres variables a la vez).

Representación de los valores de clase

- Clase predicha: neurona con el máximo valor de salida.
- Si usamos una función sigmoide en la capa de salida, aseguramos que los valores predichos estarán entre 0 y 1.

d_1	d_2	d_3	o_1	o_2	o_3	Clase predicha
0	0	1	0,1	0,2	0,8	3
0	0	1	1,0	0,2	0,8	1
0	1	0	0,2	0,9	0,1	2

- Problema: inconsistencias, ya que las variables se están modelando de forma independiente.
- Solución: incorporar un significado probabilístico a las salidas.

Interpretación probabilística

- Si lo pensamos, la representación 1-de-J puede verse como una representación probabilística de una serie de eventos:
  - J clases: $\{C_1, C_2, \ldots, C_J\}$.
  - J eventos: el patrón pertenece a cada una de las clases del problema, es decir, "$\mathbf{x} \in C_1$", "$\mathbf{x} \in C_2$", ..., "$\mathbf{x} \in C_J$".
  - La salida deseada ($\mathbf{d}$) es predecir que el patrón pertenece a la clase correcta con la mayor probabilidad (salida 1-de-J):

$$d_j = \begin{cases} 1 & \text{Si } \mathbf{x} \in C_j \\ 0 & \text{Si } \mathbf{x} \notin C_j \end{cases} \qquad (1)$$

  - De esta forma, lo que modelamos y predecimos es la probabilidad de pertenecer a cada una de las clases:

$$o_j = \hat{P}(\mathbf{x} \in C_j | \mathbf{x}) \qquad (2)$$

La función *softmax*

- Ahora necesitamos que las salidas de la red neuronal ($\mathbf{o}$) sean "consistentes", desde un punto de vista probabilístico:

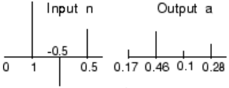
$$\sum_{j=1}^{J} o_j = 1, \qquad 0 \leq o_j \leq 1, \forall j \in \{1, \ldots, J\}. \tag{3}$$

- Para ello, podemos utilizar la función softmax que es una función de normalización que asegura que las salidas estarán entre 0 y 1 y que su suma será 1:

$$net_j^H = w_{j0} + \sum_{i=1}^{n_{H-1}} w_{ji} out_i^{H-1}, \tag{4}$$

$$out_j^H = \frac{\exp(net_j^H)}{\sum_{l=1}^{n_H} \exp(net_l^H)} \tag{5}$$

La función *softmax*



$$o_j = out_j^H = \frac{\exp(net_j^H)}{\sum_{l=1}^{n_H} \exp(net_l^H)} \tag{6}$$

- Es una aproximación plausible (desde el punto de vista biológico) y derivable a la función máximo.
- La función exponencial (exp) asegura que trataremos cantidades positivas y "exagera" mucho las salidas, para que el resultado se parezca a la función máximo (un 1 para el máximo y un 0 para el resto).

La función *softmax*

- El denominador, $\sum_{l=1}^{n_H} \exp(net_l^H)$, normaliza estas cantidades positivas, ya que es la suma de todas ellas. De esta forma, conseguimos que se cumplan las restricciones:

$$\sum_{j=1}^{J} o_j = 1, \qquad 0 \leq o_j \leq 1, \forall j \in \{1, \ldots, J\}. \tag{7}$$

- Por tanto, produce salidas correctas desde el punto de vista probabilístico.
- La clase predicha será el índice de la neurona de salida con valor más alto:

$$C(\mathbf{x}) = \arg \max_j o_j \tag{8}$$

La función *softmax*

net_1^H	net_2^H	net_3^H	$e^{net_1^H}$	$e^{net_2^H}$	$e^{net_3^H}$	$\sum_{l=1}^{n_H} e^{net_l^H}$
-1	0	3	0,368	1,000	20,086	21,454
1	4	-1	2,718	54,598	0,368	57,684
0,4	0	3	1,492	1,000	20,086	22,578

out_1^H	out_2^H	out_3^H	Clase predicha
0,017	0,047	0,936	3
0,047	0,947	0,006	2
0,066	0,044	0,890	3

La función *softmax*: derivadas

$$o_j = out_j^H = \frac{\exp(net_j^H)}{\sum_{l=1}^{n_H} \exp(net_l^H)} \tag{9}$$

- Para simplificar, tomemos $net_j = n_j$ y obviemos los límites del sumatorio:

$$o_j = \frac{e^{n_j}}{\sum_l e^{n_l}} \tag{10}$$

- Queremos calcular $\frac{\partial o_j}{\partial n_i}$.
- A la hora de calcular las derivadas, todos los n_l forman parte de la salida de cada neurona.

La función *softmax*: derivadas

Distinguimos dos casos:

1. $i = j$, es decir, $\frac{\partial o_j}{\partial n_j}$.
2. $i \neq j$, es decir, $\frac{\partial o_i}{\partial n_j}$

Primer caso ($i = j$). Derivada de la salida (o_j) respecto a algo que está por detrás de la neurona j:

$$\begin{aligned}
\frac{\partial o_j}{\partial n_j} &= \frac{\partial}{\partial n_j}\frac{e^{n_j}}{\sum_l e^{n_l}} = \frac{\partial}{\partial n_j}\left(\sum_l e^{n_l}\right)^{-1} e^{n_j} = \\
&= \left((-1)\left(\sum_l e^{n_l}\right)^{-2} e^{n_j}\right) e^{n_j} + \left(\sum_l e^{n_l}\right)^{-1} e^{n_j} = \\
&= -\frac{\left(e^{n_j}\right)^2}{\left(\sum_l e^{n_l}\right)^2} + \frac{e^{n_j}}{\sum_l e^{n_l}} = -o_j^2 + o_j = o_j(1 - o_j)
\end{aligned}$$

La función *softmax*: derivadas

Segundo caso ($i \neq j$). Derivada de la salida (o_j) respecto a algo que está por detrás de cualquier neurona i que no sea j:

$$\begin{aligned}
\frac{\partial o_j}{\partial n_{i|i\neq j}} &= \frac{\partial}{\partial n_i}\frac{e^{n_j}}{\sum_l e^{n_l}} = e^{n_j}\frac{\partial}{\partial n_i}\left(\sum_l e^{n_l}\right)^{-1} = \\
&= e^{n_j}\left((-1)\left(\sum_l e^{n_l}\right)^{-2} e^{n_i}\right) = \\
&= -\frac{e^{n_j}}{\left(\sum_l e^{n_l}\right)} \cdot \frac{e^{n_i}}{\left(\sum_l e^{n_l}\right)} = -o_j o_i
\end{aligned}$$

Se puede resumir ambos del siguiente modo:

$$\frac{\partial o_j}{\partial n_i} = o_j(I(i = j) - o_i)$$

donde $I(cond)$ será 1 si *cond* es cierto y 0 en caso contrario.

La función *softmax*: derivadas

- Para una neurona de tipo *softmax*, el valor δ_j^h debe obtenerse sumando las derivadas con respecto a todos los net_j^h:

$$\delta_j^h = \sum_{i=1}^{n_h} out_j^h (I(i = j) - out_i^h) \quad (11)$$

donde out_j^h es la transformación *softmax*:

$$out_j^h = \frac{\exp(net_j^h)}{\sum_{l=1}^{n_h} \exp(net_l^h)} \quad (12)$$

Función de rendimiento: *CCR*

- En una tarea de clasificación, nuestro objetivo debería ser que el clasificador acertase casi siempre la clase.
- *Correctly Classified Ratio* o porcentaje de patrones bien clasificados:

$$CCR = 100 \times \frac{1}{N} \sum_{p=1}^{N} \left(I(y_p = y_p^*) \right) \quad (13)$$

- N: número de patrones.
- y_p: clase deseada para el patrón p, $y_p = \arg\max_o d_{po}$.
  - Índice del valor máximo del vector $\mathbf{d}_p$.
- y_p^* clase obtenida para el patrón p, $y_p^* = \arg\max_o o_{po}$.
  - Índice del valor máximo del vector $\mathbf{o}_p$ o la neurona de salida que obtiene la máxima probabilidad para el patrón p.

Función de rendimiento: *CCR*

- Podríamos entrenar el perceptrón intentando maximizar esta cantidad, pero tiene un problema:
  - Para obtener y_p e y_p^* hay que aplicar la función $\arg\max$ que es no derivable.
  - Además, el *CCR* mejora a saltos, lo cual no nos permitiría ir ajustando poco a poco los pesos $\Rightarrow$ Convergencia difícil.
- Se puede utilizar, de nuevo, el *MSE* como función de error (a minimizar) tomando la codificación 1-de-J para las salidas y las probabilidades predichas por la función *softmax*:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\left(\frac{1}{J}\sum_{o=1}^{J}(d_{po} - o_{po})^2\right) \tag{14}$$

Función de error: entropía cruzada

- El error cuadrático medio (MSE) no es la función natural de error cuando tenemos salidas probabilísticas, ya que trata por igual cualquier diferencia de error.
- Para problemas de clasificación, deberíamos penalizar más los errores cometidos para la clase correcta ($d_j = 1$) que para la incorrecta ($d_j = 0$).
- La entropía cruzada ($-\ln$ verosimilitud) es más adecuada para problemas de clasificación ya que compara las dos distribuciones de probabilidad:

$$L = -\frac{1}{N \cdot J} \sum_{p=1}^{N} \left(\sum_{o=1}^{J} d_{po} \ln(o_{po}) \right) \tag{15}$$

Función de error: entropía cruzada

- Dado el conjunto de entrenamiento, entrenar un algoritmo de clasificación minimizando esta función de error supone estimar los parámetros que maximizan la verosimilitud de mis parámetros (*Maximum likelihood estimation*).
- La derivada se obtiene de forma similar (para un solo patrón y obviando la constante $\frac{1}{J}$):

$$L = -\sum_{l=1}^{J} d_l \ln(o_l)$$

$$(\ln(x))' = \frac{1}{x}$$

$$\frac{\partial L}{\partial w_{ji}^h} = -\sum_{l=1}^{J} \frac{\partial L}{\partial o_l} \frac{\partial o_l}{\partial w_{ji}^h} = -\sum_{l=1}^{J} \left(\frac{d_l}{o_l}\right) \frac{\partial o_l}{\partial w_{ji}^h}$$

Resumen final

- Debemos hacer que el programa saque información del *CCR*.
- Debemos incorporar la función *softmax* en la capa de salida, es decir, cambiar la forma en que se propagan las entradas (según la definición de la *softmax*) y la forma en que se retropropaga el error (según la nueva expresión de δ_j^h).
- Debemos incorporar la función de error L (entropía cruzada), haciendo que se calcule en las funciones que tienen que calcular un error y que se modifique la forma en que se retropropaga el error en los δ_j^H (de la capa de salida).
- Debemos incorporar la versión *off-line* del algoritmo (práctica anterior).

Resumen final: cálculo de δ_j^h

- Derivadas para neuronas de tipo sigmoide:
  - Capa de salida:
    - Error *MSE*:
      $\delta_j^H \leftarrow -\left(d_j - out_j^H\right) \cdot out_j^H \cdot \left(1 - out_j^H\right)$
    - Entropía cruzada:
      $\delta_j^H \leftarrow -\left(d_j / out_j^H\right) \cdot out_j^H \cdot \left(1 - out_j^H\right)$
  - Capas ocultas:
    $\delta_j^h \leftarrow \left(\sum_{i=1}^{n_{h+1}} w_{ij}^{h+1} \delta_i^{h+1}\right) \cdot out_j^h \cdot \left(1 - out_j^h\right)$
- Derivadas para neuronas de tipo *softmax*:
  - Capa de salida:
    - Error *MSE*:
      $\delta_j^H \leftarrow -\sum_{i=1}^{n_H} \left(\left(d_i - out_i^H\right) \cdot out_j^H (I(i=j) - out_i^H)\right)$
    - Entropía cruzada:
      $\delta_j^H \leftarrow -\sum_{i=1}^{n_H} \left(\left(d_i / out_i^H\right) \cdot out_j^H (I(i=j) - out_i^H)\right)$

Resumen final: ajuste de las derivadas para el modo *off-line*

- Cuando trabajamos en modo *off-line*, las derivadas se van acumulando para todos los patrones, haciendo que su magnitud pueda ser muy alta.
- Como el error usado es un error medio, deberíamos dividir el cambio realizado por el número de patrones de entrenamiento (N).

Resumen final: ajuste de las derivadas para el modo *off-line*

ajustarPesosOffLine()

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* $h-1$

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

           Fin Para
        2. $w_{j0}^h \leftarrow w_{j0}^h - \frac{\eta \Delta w_{j0}^h}{N} - \frac{\mu\left(\eta \Delta w_{j0}^h(t-1)\right)}{N}$ *// Sesgo*

       Fin Para

   Fin Para

Fin

Introducción a los modelos computacionales

Práctica 2. Perceptrón multicapa para problemas de clasificación

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)

8 de octubre de 2019