Taller de Aprendizaje de Máquina: Redes Neuronales Artificiales

Julián D. Arias Londoño
Departamento de Ingeniería de Sistemas
Universidad de Antioquia, Medellín, Colombia
jdarias@udea.edu.co

April 16, 2017

1 Marco teórico

Una red neuronal artificial (RNA) es un modelo matemático inspirado en las redes neuronales biológicas, que consiste en un grupo interconectado de neuronas artificiales que permite modelar relaciones complejas entre entradas y salidas de un sistema. El tipo de red neuronal artificial se conoce como Perceptrón Multi-Capa en la cual la unidad fundamental o neurona es el perceptrón.

Un perceptrón toma un vector de valores de entrada y calcula una combinación lineal de dichas entradas. Posteriormente entrega una salida 1 si el resultado es mayor que algún umbral y -1 de lo contrario.
Más precisamente, dado un conjunto de entradas x_1 a x_d, la salida $O(x_1, x_2, ..., x_d)$ computada por el perceptrón es:

$$O(x_1, x_2, ..., x_d) = \begin{cases} 1 \text{ si } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d > 0 \\ -1 \text{ en otro caso} \end{cases}$$

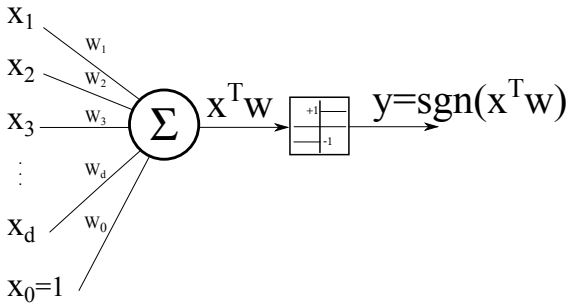

Figure 1: Esquema de un perceptrón.

donde w_i es una constante real o *peso* que determina la contribución de la entrada x_i a la salida del perceptrón [1].

Una forma alternativa de pensar el perceptrón, y que para nosotros es familiar, es asumir que existe una entrada adicional con valor constante 1, $x_0 = 1$, de tal forma que la inecuación anterior se puede escribir como:

$$\sum_{i=0}^{d} w_i x_i > 0$$

o en forma matricial, simplemente $\mathbf{w}^T\mathbf{x} > 0$. La determinación del valor a la salida del perceptrón se representa normalmente como la función signo (sgn), entonces la función de salida del perceptrón se puede reescribir como:

$$O(\mathbf{x}) = sgn(\mathbf{w}^T\mathbf{x})$$

donde

$$sgn(u) = \begin{cases} 1 \text{ si } u > 0 \\ -1 \text{ en otro caso} \end{cases}$$

Por su similitud con la regresión lineal, es simple establecer que el entrenamiento de un sólo perceptrón se puede llevar a cabo a través de un procedimiento de gradiente descendente, usando como criterio de entrenamiento la minimización del error cuadrático medio. Un aspecto importante a tener en cuenta es que la regla de actualización por gradiente descedente requiere que la función a optimizar sea derivable. En este caso la función de salida del perceptrón es discontinua y por lo tanto no tiene derivada. Sin embargo, se puede hacer la suposición que si la cantidad $\mathbf{w}^T\mathbf{x}$ tiende a 1 entonces la función $sgn(\mathbf{w}^T\mathbf{x})$ también lo hará. De igual manera se puede pensar en el caso negativo. Por esa razón el entrenamiento de un solo perceptrón puede ser llevado a cabo utilizando las mismas reglas de actualización que en el caso de la regresión lineal, teniendo en cuenta que la salida deseada toma valores 1 y -1.

Un solo perceptrón puede llevar a cabo tareas de decisión en las cuales una frontera lineal es suficiente. Cuando es necesario llevar a cabo decisiones más complejas (como representar una función lógica XOR), la solución pasa por interconectar varios perceptrones de tal manera que entre todos puedan llevar a cabo decisiones más complejas, obtieniéndose de esa manera una red de perceptrones de varias capas (milti-capa). Sin embargo la interconexión de los perceptrones implica asumir el problema de entrenamiento de la red sin poder utilizar la simplicación de la función a la salida, ya que la aplicación de dicha función es necesaria en pasos intermedios de la red. Por esa razón es necesario aproximar la función signo usada como salida en el perceptrón, por una función que presente un comportamiento aproximado y que a la vez sea continua (derivable), de tal manera que pueda ser utilizada en el entrenamiento usando una regla de gradiente descendente. Existen varias funciones que pueden ser usadas en este contexto, las más utilizadas son:

[1] Note que $-w_0$ es el umbral que la combinación sopesada de entradas $w_1x_1 + w_2x_2 + \cdots + w_dx_d$ debe sobrepasar para que la salida del perceptrón sea 1.

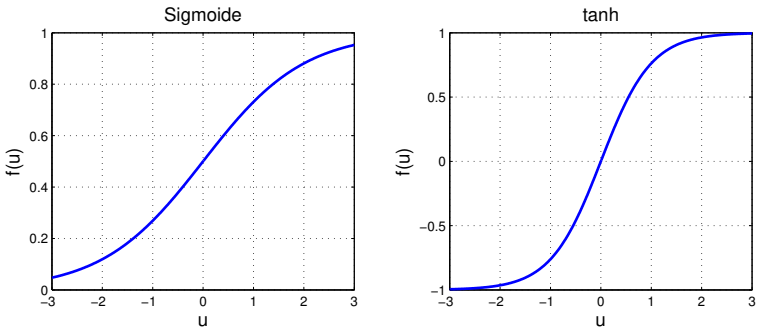

Figure 2: Funciones de activación típicas

- Función sigmoide:

$$f(u) = \frac{1}{1 + \exp(-u)}$$

  cuya derivada está dada por:

$$\dot{f}(u) = f(u)(1 - f(u))$$

- Función tangente hiperbólica:

$$f(u) = \frac{\exp(u) - \exp(-u)}{\exp(u) + \exp(-u)}$$

  cuya derivada está dada por:

$$\dot{f}(u) = 1 - (f(u))^2$$

A estas funciones se les conoce como funciones de activación. Adicional al hecho de que estas funciones son aproximaciones continúas de la función signo, también tienen la ventaja de que al ser no lineales permiten conseguir fronteras de decisión que no podrían ser obtenidas a partir de combinación de fronteras lineales.

La red interconectada de perceptrones se conoce entonces como Perceptrón Multicapa (En inglés *Multi-Layer Peceptron - MLP*, ver Fig. 3). Un MLP hace parte de las redes neuronales FeedForward o de propragación hacia adelante, debido a que el procedimiento normal de evaluación de la red consiste en colocar un conjunto de valores en la entrada de la red, utilizar dichos valores y los pesos de la primera capa para obtener la salida de los nodos de la segunda capa y seguir propagando esos valores hasta obtener los valores de salida. En este

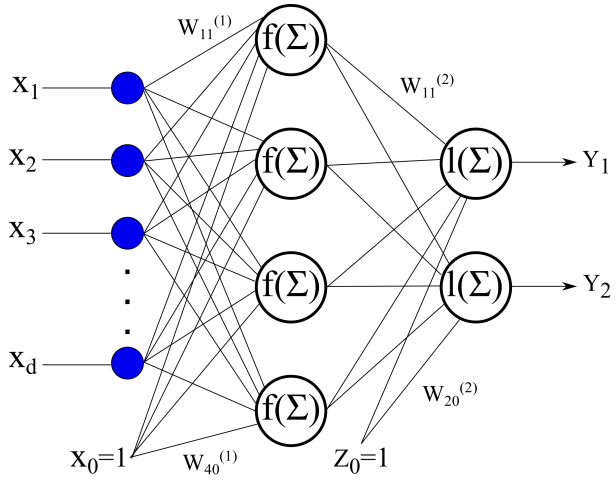

Figure 3: Esquema de un perceptrón multicapa

contexto un nodo corresponde a una entrada a la red o una salida de alguna unidad en la red (Por ejemplo en la Fig. 3 los círculos azules representan los nodos de entrada).

El algoritmo básico de entrenamiento de las RNA se conoce como BACKPROPAGATION. Para poder describir dicho algoritmo es necesario hacer algunas aclaraciones en cuanto a la notación utilizada:

- La capa de entrada tendrá tantos nodos como variables (atributos o características).
- Cada nodo tendrá un sub-índice que indicará la posición del nodo en la capa y un super-índice que indicará la capa a la cual pertence el nodo. x_{ji} denota la entrada a partir del nodo i a la unidad j, y $w_{ji}^{(1)}$ denota el correspondiente peso.
- El número de neuronas o unidades en cada capa es diferente y se denotará por M_k, donde k hace referencia a la capa.

De acuerdo a la notación anterior, en la primera capa oculta se construyen M_1 combinaciones lineales de las variables de entrada $\mathbf{x} = \{x_1, x_2, ..., x_d\}$ de la forma:

$$a_j = \sum_{i=0}^{d} w_{ji}^{(1)} x_i \tag{1}$$

donde $j = 1, ..., M_1$ y el superíndice (1) indica que los parámetros corresponden a la primera "capa" oculta de la red. Las activaciones a_j luego se transforman usando una función de activación $f(\cdot)$ no lineal para dar:

$$z_j = h(a_j)$$

En este contexto los z_j se conocen como nodos ocultos.

Estos valores se combinan linealmente de nuevo para dar unidades de activación en otras capas ocultas o de salida

$$a_k = \sum_{j=0}^{M_1} w_{kj}^{(2)} z_j$$

donde $k = 1, ..., M_2$. Si la red sólo tiene una capa oculta, M_2 es el número total de salidas. En este punto es importante aclarar que la diferencia entre un modelo de red neuronal para un problema de regresión y para un problema de clasificación radica en la definicón de la función de activación:

-Regresión	$y_k = a_k$
-Clasificación	$y_k = \sigma(a_k)$

Combinando las dos etapas anteriormente descritas se obtiene

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^{M_1} w_{kj}^{(2)} f \left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

la razón por la cual se utilizan diferente notación para la función de activación de la capa oculta y la capa de salida, es para hacer claridad en que ambas funciones de activación no tienen porque ser iguales.

El algoritmo BACKPROPAGATION está basado en el algoritmo de gradiente descendente [1], para el cual la regla actualización de los pesos de la red esta dada por,

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

en donde la función de error es:

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

donde E_n es el error debido a la predicción obtenida por la red para la muestra n. La actualización de los pesos pasa entonces por obtener una formula cerrada para la estimación de la derivada $\nabla E_n(\mathbf{w})$.

Consideremos el modelo lineal simple en el que las salidas y_k son combinaciones lineales de las variables de entrada (problema de regresión):

$$y_k = \sum_i w_{kj} z_j$$

con una función de error

$$E_n = \frac{1}{2}\sum_k (y_{nk} - t_{nk})^2$$

donde z_j es la salida de la neurona j de la primera capa, w_{kj} es el peso asociado a la entrada j de la neurona de salida k y $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$ y t_{nk} es la salida deseada para la neurona de salida k [2]. El gradiente de esta función con respecto a w_{kj} está dado como:

$$\frac{\partial E_n}{\partial w_{kj}} = (y_{nk} - t_{nk})\, z_j$$

introduciendo la variable $\delta_k = (y_{nk} - t_{nk})$, obtenemos

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j$$

y la regla de actualización estará dada por:

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \sum_n \delta_k z_j$$

Es decir que cuando necesitamos actualizar los pesos de la última capa de la red, la regla de actualización puede obtenerse fácilmente. Sin embargo, la aplicación de la misma función de error cuadrático medio no puede ser llevada a cabo en la primera capa (o en cualquier capa oculta) debido a que no se cuentan con los valores deseados para la salida de dicha capa. Para poder realizar la actualizaión de los pesos de la primera capa debemos calcular la derivada del error con respecto a dichos pesos, por consiguiente si deseamos actualizar un peso w_{ji} ubicado en la primera capa, debemos hacer uso de la regla de la cadena y derivar las salidas de la red neuronal con respecto a la entrada j de la capa de salida y posteriormente derivar dicha salida con respecto al peso w_{ji}.

$$\frac{\partial E_n}{\partial w_{ji}} = \sum_k (y_{nk} - t_{nk}) \frac{\partial y_{nk}}{\partial w_{ji}}$$

Teniendo en cuenta que la función de activación de la capa oculta es $h(\cdot)$

$$\frac{\partial y_{nk}}{\partial w_{ji}} = w_{kj} \frac{\partial z_j}{\partial w_{ji}}$$

considerando la Eq. (1),

$$\frac{\partial z_j}{\partial w_{ji}} = \dot{h}(a_j) x_i$$

reordenando,

[2] Note que como la red neuronal tiene K salidas, el error cuadrático medio obtenido para una muestra n, corresponde a la suma de los errores para cada una de las salidas de la red dada la muestra n

$$\frac{\partial E_n}{\partial w_{ji}} = \sum_k \left(y_{nk} - t_{nk}\right) w_{kj} \dot{h}(a_j) x_i$$

sacando de la sumatoria los términos que no dependen de k y utilizando la variable δ_k previamente definida obtenemos

$$\frac{\partial E_n}{\partial w_{ji}} = \dot{h}(a_j) \sum_k \delta_k w_{kj} x_i$$

introduciendo ahora la variable $\delta_j = \dot{h}(a_j) \sum_k \delta_k w_{kj}$ obtenemos

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

finalmente la regla de actualización estará dada por:

$$w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} - \eta \sum_n \delta_j x_i$$

Si se observa con detenimiento podemos ver que para calcular el valor de δ_j unicamente necesitamos el valor de los δ_k, es decir el "error" de la capa siguiente propagado hacia atrás. Este mismo procedimiento puede ser utilizado para el entrenamiento de varias capas ocultas, sólo es necesario tener en cuenta que los δ's de la capa de salida se calculan como $\delta_k = (y_{nk} - t_{nk})$ si es un problema de regresión, o $\delta_k = \dot{\sigma}(a_k)\left(y_{nk} - t_{nk}\right)$ si es un problema de clasificación; mientras que los δ's de las capas ocultas se calculan como $\delta_j = \dot{h}(a_j) \sum_k \delta_k w_{kj}$, donde δ_k serán los δ's de la capa inmediatamente siguiente.

El procedimiento de propagación hacia atrás se puede resumir como:

1. Aplicar un vector de entrada $\mathbf{x}_n$ a la red y propagarlo hacia adelante.
2. Evaluar todos los δ_k para las unidades de salida
3. Propagar los δ's del paso anterior para obtener los δ_j de cada nodo oculto en la red.
4. Usar $\partial E_n / w_{ji} = \delta_j z_i$ para evaluar las derivadas requeridas.
5. Realizar la sumatoria de derivadas del error para todas las muestras n del conjunto de entrenamiento.

2 Instrucciones para el trabajo con el toolbox de Matlab

1. La librería (toolbox) que utilizaremos es propietaria de Matlab así que está cargada por defecto. Dependiendo de la versión de Matlab, pueden existir diferentes funciones asociadas a la creación de redes neuronales tipo MLP:

`newff, feedforwardnet, fitnet y patternnet` las cuales reciben varios parámetros de entrada y retornan una estructura de red (`net`). Revise cuáles funciones están disponibles en la versión de Matlab que está usando. Revise los parámetros de entrada para la función que desea usar, para el caso de `newff`, los parámetros de entrada más importantes son (en las demás funciones los parámetros son más fácilmente identificables.):

- P: Es una matriz de $(d \times N)$ que contiene las muestras de entrenamiento, donde N es el número de muestras y d es la dimensión del espacio de características (número de características). **Note** que es la traspuesta de la matriz que usualmente hemos usado en los talleres anteriores.
- T: Es una matriz de ($K \times N$ que contiene los valores deseados t (objetivo), para cada una de las K salidas de la red neuronal por cada una de las N muestras de entrenamiento).
- S: Es un vector que contiene el número de unidades o neuronas en cada una de las capas ocultas. La capa de entrada y de salida ya están determinadas por el número de columnas de P y el número de columnas de T respectivamente.
- TF: Es una celda que contiene las funciones de activación de cada capa oculta y de la capa de salida. Las opciones a escoger son: `'tansig', 'logsig' y 'purelin'`. La primera corresponde a una función de activación tangente hiperbólica, la segunda a una función sigmoidal y la tercera implica no imponer ninguna función no linea, sino dejar la salida lineal correspondiente a la suma de las entradas multiplicadas por sus correspondientes pesos.
- BTF: Es un string que permite escoger entre diferentes algoritmos de entrenamiento. El algoritmo por defecto es `'trainlm'` el cuál tiene una convergencia más rápida que el algoritmo Backpropagation pero demanda mucha memoria RAM y es posible obtener errores de "OUT OF MEMORY" que significa que el equipo se ha quedado sin memoria para llevar a cabo el entrenamiento de la RNA. Una buena opción en ese caso es utilizar `'trainbfg'` que alcanza buenos resultados con tamaños menores de memoria RAM. Finalmente el algoritmo Backpropagation estándar se puede encontrar en la opción `'traingd'`.

Las demás opciones permiten entre otras cosas variar la función de error utilizada como criterio de entrenamiento, pero que no son de interés en este taller. Si desean tener una idea visual de la estructura de la red que acaban de crear pueden utilizar la función `'view(net)'`.

2. Una vez creada la red pueden modificar el campo `'net.trainParam.epochs = 50'`, en la cual podrán definir el número de épocas que desean usar durante el entrenamiento. En el contexto de las redes neuronales una

época equivale a una iteración completa del algoritmo BACKPROPAGATION. También pueden modificar el error de entrenamiento deseado `'net.trainParam.goal = 0.01'`, aunque en el entrenamiento de las RNA no suele ser de mucha utilidad dicho valor, porque suelen presentar problemas de sobre ajuste y no sólo es importante alcanzar una tasa de error pequeña sino también terner una buena capacidad de generalización. A este respecto trabajaremos en próximas sesiones del curso.

3. El entrenamiento de la RNA se puede llevar a cabo usando la función `'train'`, es muy simple, pueden verificar los parámetros de enreda y de salida. Lleve a cabo el entrenamiento de la red para las variables de salida $Y1$ y $Y2$ de la base de datos indicada en los puntos 1 y 2.

4. Finalmente la validación de la RNA se puede llevar a cabo con la función `'Yest= sim(net,Xtest)'`, la cual recibe como parámetro la estructura de la RNA entrenada y una matrix con los datos que se desean validar. La variable de salida será una matriz de K filas e igual número de columnas que la matriz de entrada. En este caso, al igual que el entrenamiento, cada columna de la matriz de datos corresponde a una muestra y las filas corresponden a cada una de las salidas de la RNA. Realice la validación de la RNA y estime el error de validación. Repita el procedimiento para diferente número de neuronas en la capa oculta.

3 Ejercicios

1. Descargue el conjunto de datos sobre la eficiencia en el consumo de energía para la refrigeración y la calefacción de edificios con diferentes formas, que encontrarán en el Machine Learning Repository:
`http://archive.ics.uci.edu/ml/datasets/Energy+efficiency`.

   Importe o cargue los datos en MatLab;

2. El conjunto de datos a analizar es el mismo utilizado en el Taller 2. El objetivo en este punto es crear la estructura de una red neuronal artificial que permita hacer la predicción de las variables $Y1$ y $Y2$ al mismo tiempo.

3. Lleve a cabo un proceso de validación para determinar el mejor conjunto de parámetros para el problema de regresión de múltiples salidas, a partir de una RNA. Deben variar el número de épocas usando los valores $[100, 400, 800, 1000]$. Use una sola capa oculta y varíe el número de neuronas por capa. Deben decidir la malla de valores a evaluar para el número de neuronas (mínimo 5 valores) y explicar la razón por la cual se decidieron dichos valores.

4. Construir una tabla para presentar los resultados de la evaluación. ¿Qué medidas de desempeño van usar y porqué? Escriba la formula matemática de cada una.

5. Tabla con los resultados.

6. *Haga una gráfica de las salidas reales vs las predicciones hechas por la RNA, para evaluar visualmente el desempeño de la red.

Nota. El procedimiento debe llevarse a cabo utilizando un esquema de validación similar al de los laboratorios anteriores (Validación cruzada con 10 folds). Recuerde que la normalización de variables sigue siendo un paso imprescindible teniendo en cuenta que el algoritmo BACKPROPAGATION es en esencia un algoritmo de gradiente descendente.

References

[1] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. New Jersey, NY, USA: Wiley-Interscience, 2nd ed., 2000.