

Tarea Catorce: Redes Neuronales Alimentadas Hacia Adelante

Programación y Algoritmos I

Esteban Reyes Saldaña

27 de noviembre de 2020

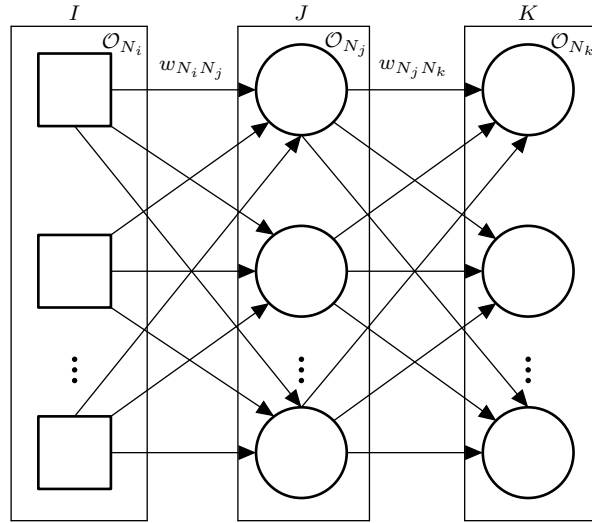
Enunciado

Retomando la tarea previa de la RNM embebida en la interfaz gráfica.

- (i) Implementar una interfaz gráfica que permita establecer los parámetros de la RNM.
- (ii) Implementar una gráfica 2D que muestre la razón de error de entrenamiento.
- (iii) Permitir la carga de datos mediante la interfaz gráfica.
- (iv) Mostrar la eficiencia de la RNM, así como los pesos asociados.

1 Red Neuronal Artificial

Consideremos una red neuronal como la siguiente



Donde

$$\begin{aligned}
 I &= \{N_1, N_2, \dots, N_n\} \\
 J &= \{N_{n+1}, N_{n+2}, \dots, N_{n+m}\} \\
 K &= \{N_{n+m+1}, \dots, N_{n+m+p}\}
 \end{aligned}$$




\mathcal{O}_l : salida de la neurona n_l , $1 \leq l \leq n + m + p$.

$w_{N_x N_y}$: nivel de influencia de N_x sobre N_y .

De tal manera que sólo hay conexiones hacia adelante entre neuronas de capas adyacentes.

1.1 Funciones de Activación

Nombre	Función	Gráfica
Lineal	$f(x) = x$	
Límite Duro	$f(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$	

Nombre	Función	Gráfica
Sigmoide	$f(x) = \frac{1}{1 + e^{-x}}$	
Tangente Hiperbólica	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
Lineal Positiva	$f(x) = \begin{cases} 0, & \text{si } x < 0 \\ x, & \text{si } x \geq 0 \end{cases}$	
Competitiva	$f(x) = \begin{cases} 1, & \text{neurona con el máximo} \\ 0 & \text{todas las demás.} \end{cases}$	

Observación 1. Como los pesos iniciales se dan de manera arbitraria, se requieren ajustar los pesos para que converjan a los valores esperados del conjunto de entrenamiento. Para el método de backpropagation, los pesos se ajustan midiendo el error entre el output obtenido y el valor esperado.

2 Backpropagation

Definición 2.1. Sea $f : R^n \rightarrow R$ una función derivable. El **gradiente** de f en p es

$$\nabla f(p) = \left(\frac{\partial f(p)}{\partial x_1}, \frac{\partial f(p)}{\partial x_2}, \dots, \frac{\partial f(p)}{\partial x_n} \right).$$

Para una función de n variables, el gradiente es el vector normal a la curva de nivel en un punto dado.

Teorema 1. Sea $f : R^n \rightarrow R$ una función derivable. Entonces la dirección donde f crece más rápido es la dirección de ∇f (el gradiente de f).

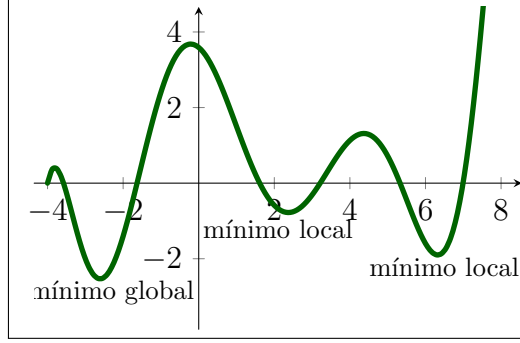
Corolario 1. Bajo las condiciones del teorema anterior, f decrece más rápido en la dirección de $-\nabla f$.

Observación 2. Se considerará la función de error (Error Cuadrado) dada por

$$E = \frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2,$$

donde K es el conjunto de todos los índices de las neuronas de la capa de salida en una red neuronal, \mathcal{O}_k es la salida para una de las neuronas producida por la red en un momento τ y t_k es la salida esperada.

Con el descenso del gradiente, nos acercamos a un mínimo local. Esto quiere decir que este algoritmo podría converger a un mínimo que no reduzca el error lo suficiente.



2.1 Función de Activación Sigmoides

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

Se puede ver que

$$\begin{aligned} \sigma'(x) &= \sigma(x) - \sigma^2(x) \\ &= \sigma(x)(1 - \sigma(x)). \end{aligned}$$

Recordemos que para nuestra estructura de red neuronal, el argumento x de la función sigmoide es

$$x = h_l = \sum_{t \in T} O_t w_{tl} + \theta_l.$$

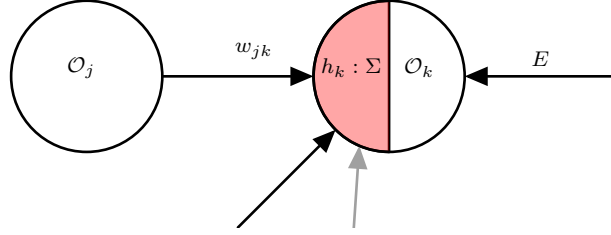
Así que para la derivada tenemos

$$\begin{aligned} \sigma'(x) &= \sigma(x)(1 - \sigma(x)) \\ &= \sigma(h_l)(1 - \sigma(h_l)) \\ &= \mathcal{O}_l(1 - \mathcal{O}_l). \end{aligned}$$

2.2 De la capa de salida a la capa oculta

La función $E = \frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2$ depende de los pesos. El ajuste de los pesos se hace mediante

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial \mathcal{O}_k} \frac{\partial \mathcal{O}_k}{\partial h_k} \frac{\partial h_k}{\partial w_{jk}} \quad (2)$$



Realizando los cálculos obtenemos

$$\begin{aligned}
 \boxed{\frac{\partial E}{\partial \mathcal{O}_k}} &= \frac{\partial}{\partial \mathcal{O}_k} \left(\frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2 \right) \\
 &= \frac{1}{2} \frac{\partial}{\partial \mathcal{O}_k} \left(\sum_{k \in K} (\mathcal{O}_k - t_k)^2 \right) \\
 &= \frac{1}{2} 2(\mathcal{O}_k - t_k) \frac{\partial}{\partial \mathcal{O}_k} (\mathcal{O}_k - t_k) \\
 &= \mathcal{O}_k - t_k.
 \end{aligned}$$

Nota. En el momento de aplicar la derivada a la función de error, el medio se considera para que no afecte a dicha derivada.

$$\begin{aligned}
 \boxed{\frac{\partial \mathcal{O}_k}{\partial h_k}} &= \frac{\partial}{\partial h_k} \sigma(h_k) \\
 &= \sigma(h_k)(1 - \sigma(h_k)) \\
 &= \mathcal{O}_k(1 - \mathcal{O}_k).
 \end{aligned}
 \quad
 \begin{aligned}
 \boxed{\frac{\partial h_k}{\partial w_{jk}}} &= \frac{\partial}{\partial w_{jk}} \left(\sum_{j \in J} w_{jk} \mathcal{O}_j + \theta_j \right) \\
 &= \mathcal{O}_j.
 \end{aligned}$$

De lo anterior se tiene que

$$\begin{aligned}
 \frac{\partial E}{\partial w_{jk}} &= \frac{\partial E}{\partial \mathcal{O}_k} \frac{\partial \mathcal{O}_k}{\partial h_k} \frac{\partial h_k}{\partial w_{jk}} \\
 &= (\mathcal{O}_k - t_k) \mathcal{O}_k (1 - \mathcal{O}_k) \mathcal{O}_j.
 \end{aligned}$$

Ya que esto depende de la capa K y por motivos de notación se nombrará

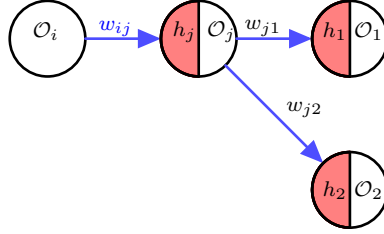
$$\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k).$$

Por lo que

$$\frac{\partial E}{\partial \mathcal{O}_k} = \mathcal{O}_j \delta_k.$$

2.3 De la Capa Oculta a la Capa de Entrada

El error para w_{ij} depende del error de las conexiones de la capa anterior.



$$\begin{aligned}
 \frac{\partial E}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \frac{1}{2} \sum_{k \in K} (\mathcal{O}_k - t_k)^2 \\
 &= \frac{1}{2} \cdot 2 \sum_{k \in K} (\mathcal{O}_k - t_k) \frac{\partial}{\partial w_{ij}} (\mathcal{O}_k - t_k) \\
 &= \frac{1}{2} \cdot 2 \sum_{k \in K} (\mathcal{O}_k - t_k) \left(\frac{\partial \mathcal{O}_k}{\partial w_{ij}} - \frac{\partial t_k}{\partial w_{ij}} \right) \\
 &= \frac{1}{2} \cdot 2 \sum_{k \in K} (\mathcal{O}_k - t_k) \frac{\partial \mathcal{O}_k}{\partial w_{ij}} \\
 &= \sum_{k \in K} (\mathcal{O}_k - t_k) \frac{\partial \mathcal{O}_k}{\partial h_k} \frac{\partial h_k}{\partial \mathcal{O}_j} \frac{\partial \mathcal{O}_j}{\partial w_{ij}} \\
 &= \frac{\partial \mathcal{O}_j}{\partial w_{ij}} \sum_{k \in K} (\mathcal{O}_k - t_k) \frac{\partial \mathcal{O}_k}{\partial h_k} \frac{\partial h_k}{\partial \mathcal{O}_j}.
 \end{aligned}$$

Realizando las cuentas llegamos a

$$\begin{aligned}
 \boxed{\frac{\partial \mathcal{O}_j}{\partial w_{ij}}} &= \frac{\partial \mathcal{O}_j}{\partial h_j} \frac{\partial h_j}{\partial w_{ij}} \\
 &= \frac{\partial}{\partial h_j} \sigma(h_j) \frac{\partial}{\partial w_{ij}} \sum_{j \in J} (w_{ij} \mathcal{O}_i + \theta_j) \\
 &= \mathcal{O}_j (1 - \mathcal{O}_j) \mathcal{O}_i.
 \end{aligned}$$

Nota. El término $\frac{\partial \mathcal{O}_k}{\partial h_k}$ se calculó previamente (es la derivada de la función sigmoide).

$$\begin{aligned} \boxed{\frac{\partial h_k}{\partial \mathcal{O}_j}} &= \frac{\partial}{\partial \mathcal{O}_j} \sum_{k \in K} (w_{jk} \mathcal{O}_j + \theta_k) \\ &= w_{jk}. \end{aligned}$$

Así que para la capa oculta se obtiene

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial \mathcal{O}_j}{\partial w_{ij}} \sum_{k \in K} (\mathcal{O}_k - t_k) \frac{\partial \mathcal{O}_k}{\partial h_k} \frac{\partial h_k}{\partial \mathcal{O}_j} \\ &= \mathcal{O}_j (1 - \mathcal{O}_j) \mathcal{O}_i \sum_{k \in K} \boxed{(\mathcal{O}_k - t_k) \mathcal{O}_k (1 - \mathcal{O}_k)} w_{jk} \\ &= \mathcal{O}_j (1 - \mathcal{O}_j) \mathcal{O}_i \sum_{k \in K} \boxed{\delta_k} w_{jk} \\ &= \mathcal{O}_i \mathcal{O}_j (1 - \mathcal{O}_j) \sum_{k \in K} \delta_k w_{jk}. \end{aligned}$$

Si se define

$$\delta_j = \mathcal{O}_j (1 - \mathcal{O}_j) \sum_{k \in K} \delta_k w_{jk},$$

entonces

$$\frac{\partial E}{\partial w_{ij}} = \mathcal{O}_i \delta_j.$$

Para las neuronas que tienen sesgo (θ_l), se tiene que

$$\begin{aligned} \frac{\partial E}{\partial \theta_l} &= \boxed{\frac{\partial E}{\partial \mathcal{O}_l} \frac{\partial \mathcal{O}_l}{\partial h_l}} \frac{\partial h_l}{\partial \theta_l} \\ &= \boxed{\delta_l} \frac{\partial h_l}{\partial \theta_l} \\ &= \delta_l \frac{\partial}{\partial \theta_l} \sum_{l \in L} (w_{l-1,l} \mathcal{O}_{l-1} + \theta_l) \\ &= \delta_l. \end{aligned}$$

2.4 El algoritmo

1. Crear una red neuronal y cargar los datos de entrenamiento.
2. Crear pesos y bias aleatorios para cada conexión. (El bias se puede incorporar como una neurona con valor 1 en la capa de entrada).
3. Para cada nodo en la capa de salida calcular

$$\delta_k = \mathcal{O}_k(1 - \mathcal{O}_k)(\mathcal{O}_k - t_k).$$

4. Para cada nodo en la capa oculta calcular

$$\delta_j = \mathcal{O}_j(1 - \mathcal{O}_j) \sum_{k \in K} \delta_k w_{jk}.$$

5. Actualizar los pesos y los bias como sigue

$$\begin{aligned} \Delta w &= -\eta \delta_l \mathcal{O}_{l-1} \\ \Delta \theta &= -\eta \delta_l \end{aligned}$$

para un $0 < \eta \leq 1$. Luego aplicar

$$\begin{aligned} w + \Delta w &\rightarrow w \\ \theta + \Delta \theta &\rightarrow \theta. \end{aligned}$$

3 Implementación

Se retomó la implementación de la RNM de la tarea anterior y se utilizó *qt5* usando *qt creator* para implementar la interfaz gráfica. Se utilizó ubuntu 20.10, los comandos de instalación son

```
sudo apt-get install qtcreator
sudo apt-get install qtdeclarative5-dev
```

qt creator crea la clase *MainWindow* por defecto para mostrar una pantalla.

A esta clase se le declararon las variables privadas *tolerance*, *rate*, *iterations*, *filename* para recibir desde la ventana la tolerancia, el rango de aprendizaje, el número máximo de iteraciones y el nombre del archivo.

Para leer la arquitectura de la red se usaron las variables *hidden_layers* y

nhidden_{size}. La primera variable es de tipo entero y recibe el número de capas ocultas y la segunda es de tipo QString que recibe un string del número correspondiente de neuronas en cada capa oculta separadas por espacio. El nombre del archivo se recibe mediante el botón *openfile* que abre el explorador de archivos para seleccionar el dataset. Por defecto se abre en el home path.

Para realizar el gráfico de la curva del error acumulado se utilizó el vector de errores previamente generado en un entrenamiento de la red neuronal. Se copiaron estos errores en un QVector declarado como variable privada de la clase *MainWindow*. Para graficarla se utilizó un widget customPlot utilizando la librería *qcustomplot* versión 2.0.1 tomada de la siguiente dirección

<https://www.qcustomplot.com/index.php/download>

La gráfica de la red neuronal se utilizó la librería *QGraphicsScene*

<https://doc.qt.io/qt-5/qgraphicsscene.html>

4 Resultados

Para correr el programa se tienen dos opciones

- (i) Desde Qt Creator abra la carpeta */NNInterface* y seleccione el archivo *NNInterface.pro*, luego corra el programa.
- (ii) Desde terminal, vaya a la carpeta */build – NNInterface – Desktop – Debug* y ejecute make. Luego corra el programa como *./NNInterface*.

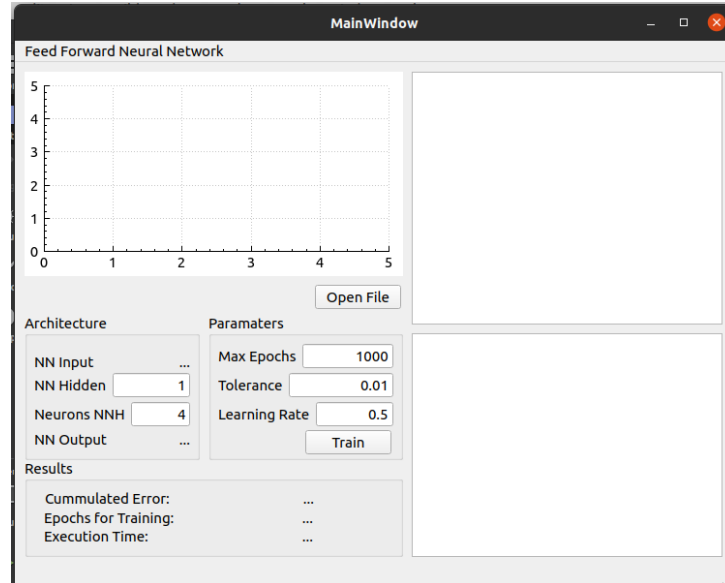
Por defecto, al abrir el programa se tienen los parámetros
Arquitectura

NNHidden : 1
NeuronsNNH : 4.

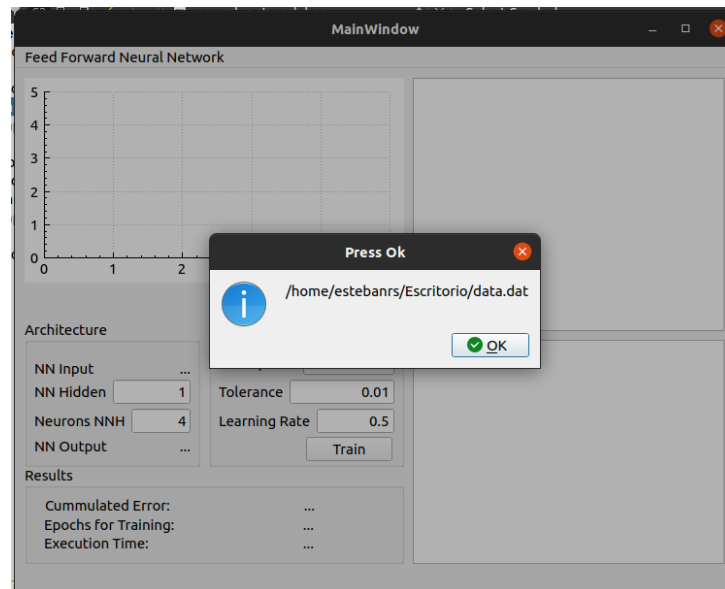
Parámetros

MaxEpochs : 1000
Tolerance : 0.01
LearningRate : 0.5

La interfaz creada es



Se carga el archivo para crear un dataset (se utilizó el mismo dataset de la tarea pasada).

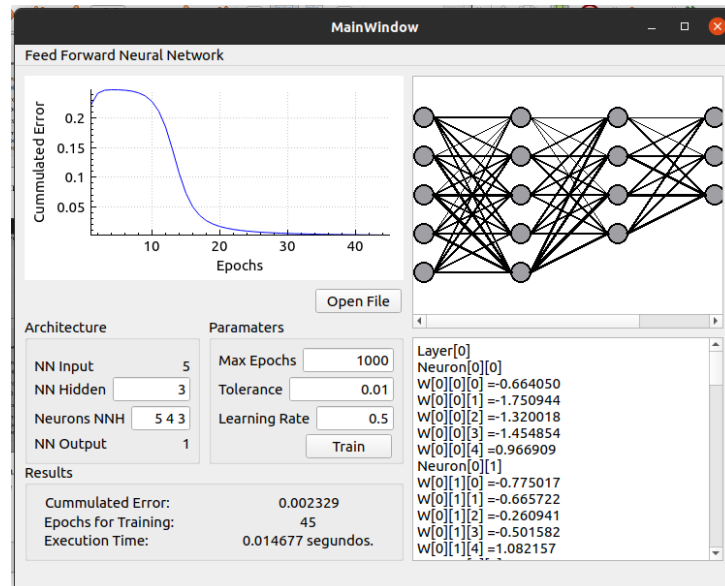


Tome en cuenta que si no selecciona el archivo o alguno de los parámetros no es válido, se mostrará un mensaje para que revise los datos. Una vez establecidos, se pulsa el botón *train*. A continuación se muestran tres corridas

con distintos parámetros.

Nota. En la gráfica de la red neuronal, el grosor de las conexiones depende de la significancia de los pesos correspondientes.





5 Conclusiones

Para esta base de datos la red neuronal aprendió en menos de 50 iteraciones para los parámetros establecidos y distintas arquitecturas. Esto se debe a que son pocos datos de entrenamiento y la primer columna es binaria.

La visualización de la curva de error permite visualizar cómo decae el error a través de las épocas y el gráfico permite ver qué pesos tienen más significancia en la red.

Se observa además que para una tolerancia de 0.0001 la red no converge respecto al error, esto podría deberse a que la función de error se estancó en un mínimo local.