PRÁCTICAS

DE

SISTEMAS

INTELIGENTES

PRÁCTICA 2:

REDES NEURONALES

DOCUMENTACIÓN

ALEJANDRO REYES ALBILLAR

45931406 - S

ara65@alu.ua.es

Curso 2015 / 2016

Grado en Ingeniería Informática

Universitat d'Alacant
Universidad de Alicante

1. INTRODUCCIÓN

En esta segunda y última práctica de la asignatura de Sistemas Inteligentes, nos invitan a estudiar qué son y cómo se comportan las redes neuronales.

Para ello se nos proporciona un escenario y unas herramientas con las que trabajar, las cuales son las siguientes:

- Se nos proporciona el código en .java de una red neuronal capaz de reconocer números manuscritos, más concretamente los números de la base de datos MNIST.
- A partir de un conjunto de entrenamiento se entrena la red neuronal.
- Una vez entrenada la red, se utiliza un conjunto de test para comprobar el error que da.

Esto nos lleva a experimentar a cerca de los diferentes tipos de elementos de una red neuronal, cuál es su funcionamiento y cómo de precisos pueden ser estos sistemas de aprendizaje automatizado encaminado al sector de la inteligencia artificial.

2. XOR

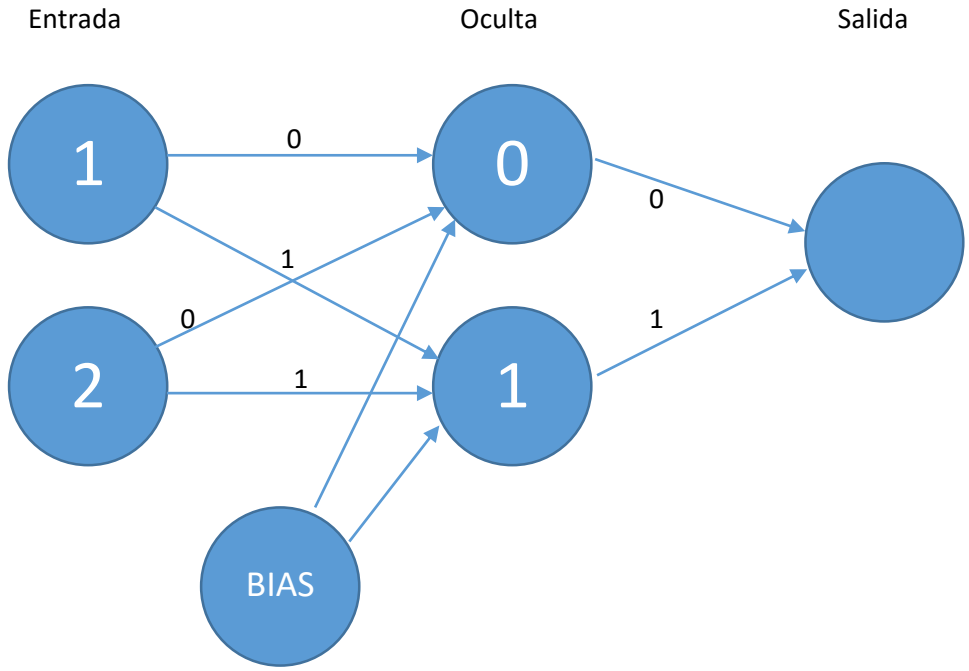
Se nos pide que describamos el algoritmo BackPropagation, que consiste en un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales. Este algoritmo está basado en el siguiente pseudocódigo:

Algoritmo BACKPROPAGATION($red\ ejemplos, \eta$) {
  $\{w_{ij}\} \leftarrow$ INICIALIZAR;
  Mientras $\neg$ CONVERGENCIA(red) **Hacer** {
    $e \leftarrow$ SELECCIONAREJEMPLO($ejemplos$);
    $\{y_k\} \leftarrow$ FORWARD(e);
    $\{d_k\} \leftarrow$ DESEADAS(e);
    Para cada $n_k \in$ CAPA(red, k) **Hacer** {
      $\delta_k = (d_k - y_k) f'(net_k)$;
    }
    Para $j = k - 1$ **hasta** 1 **Hacer** {
      Para $n_j \in$ CAPA(red, j) **Hacer** {
        $\delta_j = f'(net_j) \sum_{j+1} \delta_{k+1} w_{j(j+1)}$;
      }
    }
    Para $j = k$ **hasta** 1 **Hacer** {
      $w_{(j-1)j} = w_{(j-1)j} + \eta \delta_j y_{(j-1)}$;
    }
  $red \leftarrow$ ACTUALIZARRED($\{w_{ij}\}$);
  }
  Devolver red;
}

El algoritmo funciona del siguiente modo:

Dada una red y un conjunto de ejemplos dados, inicializamos la matriz de pesos. Una vez inicializada se seleccionará un ejemplo del conjunto de ejemplos hasta que el error baje de un umbral o llegue al número máximo de ejemplos especificados por el usuario. Se obtiene el error de la última capa y la salida de la red de cada uno de los ejemplos, así como los errores de una capa con respecto a la siguiente. Finalmente, se actualizan los pesos de todas las capas.

Para el ejemplo de la XOR, utilizando la estructura (entrada, oculta, salida)=(2,2,1) vemos lo siguiente:

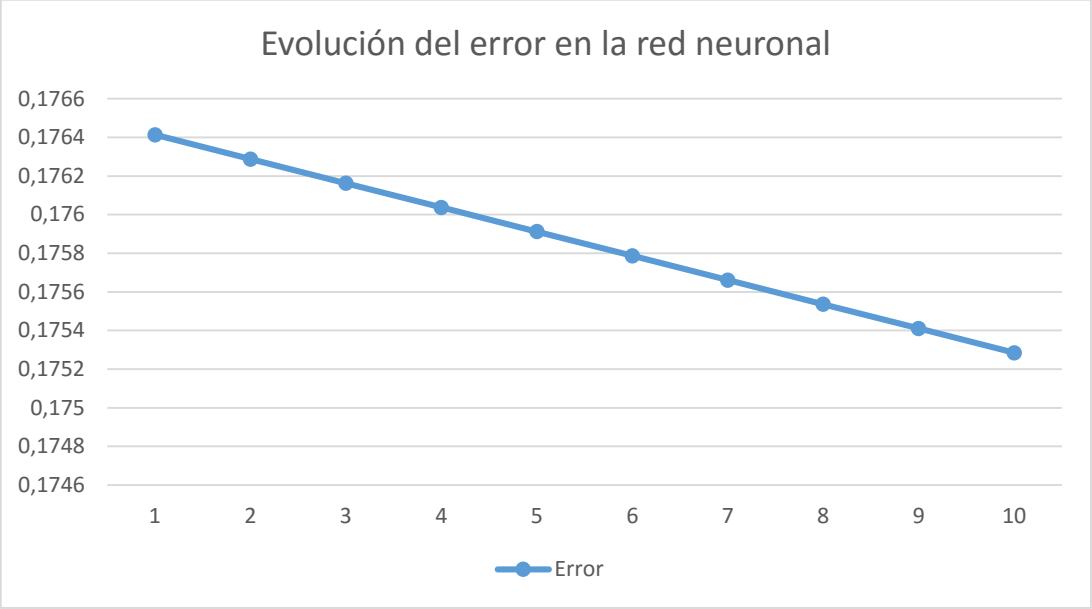


Sabemos que una XOR funciona de la siguiente manera:

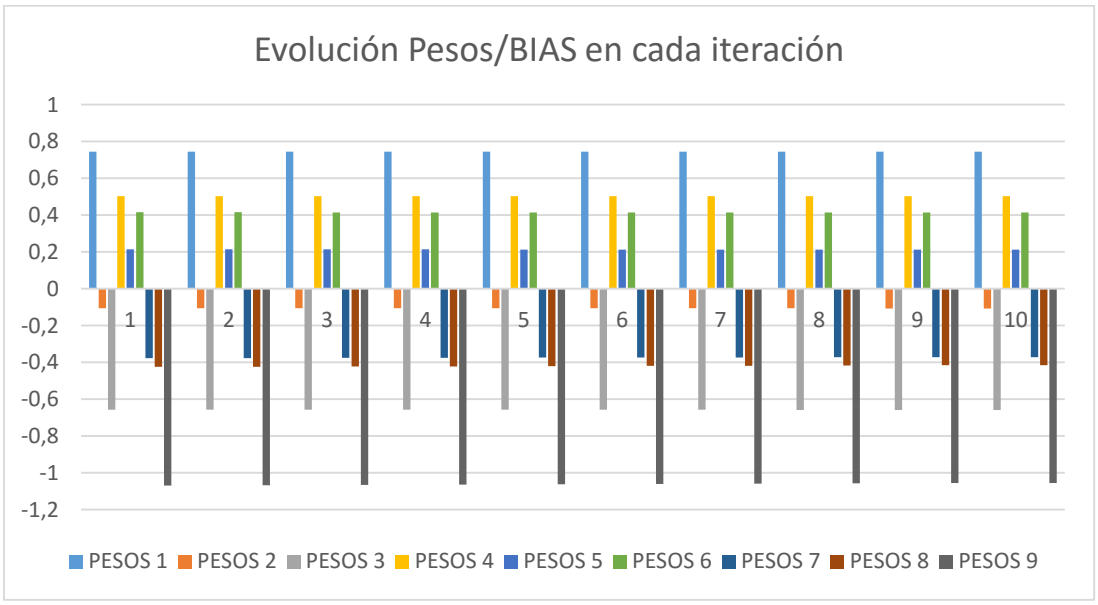
Entrada1	Entrada 2	Salida
0	0	0
0	1	1
1	0	1
1	1	0

A continuación mostramos la evolución de los pesos y bias de la XOR conforme va aprendiendo la red, además del error de la red neuronal:

Iteración	Error
1	0,17641318735357728
2	0,17628790937051383
3	0,1761626195871837
4	0,17603731918969703
5	0,1759120093672155
6	0,1757866913119091
7	0,17566136621891235
8	0,1755360352862799
9	0,17541069971494233
10	0,17528536070866088



Iteración	PESOS 1	PESOS 2	PESOS 3	PESOS 4	PESOS 5	PESOS 6	PESOS 7	PESOS 8	PESOS 9
1	0,7454	-0,1059	-0,6566	0,5034	0,2139	0,4154	-0,3770	-0,4248	-1,0689
2	0,7453	-0,1060	-0,6569	0,5033	0,2138	0,4152	-0,3764	-0,4238	-1,0673
3	0,7452	-0,1061	-0,6571	0,5031	0,2137	0,4150	-0,3757	-0,4227	-1,0657
4	0,7451	-0,1062	-0,6573	0,5030	0,2135	0,4148	-0,3750	-0,4216	-1,0641
5	0,7450	-0,1063	-0,6575	0,5029	0,2134	0,4146	-0,3744	-0,4205	-1,0626
6	0,7449	-0,1064	-0,6577	0,5028	0,2133	0,4144	-0,3737	-0,4194	-1,0610
7	0,7448	-0,1065	-0,6579	0,5027	0,2132	0,4141	-0,3731	-0,4184	-1,0594
8	0,7447	-0,1066	-0,6581	0,5026	0,2131	0,4139	-0,3724	-0,4173	-1,0578
9	0,7446	-0,1067	-0,6582	0,5025	0,2130	0,4137	-0,3717	-0,4162	-1,0562
10	0,7445	-0,1068	-0,6584	0,5024	0,2129	0,4135	-0,3711	-0,4151	-1,0547



Mostramos a continuación el fragmento de código con el que hemos implementado la red neuronal.

```
package xor.practica.pkg2.si;

import java.io.IOException;
import java.util.Arrays;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.data.DataSetRow;
import org.neuroph.core.events.LearningEvent;
import org.neuroph.core.events.LearningEventListener;
import org.neuroph.nnet.MultiLayerPerceptron;
import org.neuroph.nnet.learning.BackPropagation;
import org.neuroph.samples.convolution.MNISTDataSet;
import org.neuroph.util.TransferFunctionType;

/**
 * @author ALEANDRO REYES ALBILLAR 45931406-S
 */
public class NeuralNet implements LearningEventListener {
    public NeuralNet(int inputSize, int hiddenLayerSize, int outputSize){
       _mlp = new MultiLayerPerceptron(TransferFunctionType.SIGMOID,
inputSize, hiddenLayerSize, outputSize);
       _bp = new BackPropagation();
       //Para recibir el evento en cada iteración de BP
       _bp.addListener(this);
    }

    public void randomize(){
        _mlp.randomizeWeights();
    }

    public void train(double learningRate, int maxIterations, DataSet
trainingSet) {

        _trainSet=trainingSet;

        System.out.println("Training network...");
        _bp.setMaxIterations(maxIterations);
        _bp.setLearningRate(learningRate);
        _mlp.learn(_trainSet,_bp);

    }

    @Override
    public void handleLearningEvent(LearningEvent le) {
        BackPropagation bp = (BackPropagation) le.getSource();
        System.out.println(bp.getCurrentIteration() + ") Network error: " +
bp.getTotalNetworkError());
        test(_trainSet);
    }


    public void test(DataSet ds){
```

```
       int i=1;
      // System.out.println("");
       _mlp.setInput(ds.getRowAt(i).getInput());//cogemos una imagen y la
metemos a _mlp
       _mlp.calculate();//Calculamos
       Double pesos[]= _mlp.getWeights();//Array de pesos

       System.out.print("[");
       for(int j=0;j<pesos.length;j++){
            if(j==0)
                System.out.print(pesos[j]);
            else
                System.out.print(","+pesos[j]);
        }
       System.out.println("]");
       i++;
   }

    public DataSet _trainSet;
    public MultiLayerPerceptron _mlp;
    public BackPropagation _bp;
}
```

Y el main que ejecuta los métodos del código anterior:

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package xor.practica.pkg2.si;

import org.neuroph.core.NeuralNetwork;
import org.neuroph.core.data.DataSet;
import org.neuroph.core.data.DataSetRow;
import org.neuroph.nnet.Perceptron;

/**
 *
 * @author ALEANDRO REYES ALBILLAR 45931406-S
 */
public class XORPractica2SI {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
    // create new perceptron network
        NeuralNet neuralNetwork = new NeuralNet(2,2,1);
    // create training set
        DataSet trainingSet = new DataSet(2, 1);
    // add training data to training set (logical OR function)
        trainingSet.addRow(new DataSetRow(new double[]{0, 0},new
double[]{0}));
```

```
        trainingSet.addRow(new DataSetRow(new double[]{0, 1},new
double[]{1}));
        trainingSet.addRow(new DataSetRow(new double[]{1, 0},new
double[]{1}));
        trainingSet.addRow(new DataSetRow(new double[]{1, 1},new
double[]{0}));
        // learn the training set

        //Entreno la red
        double learningRate = 0.005;
        int maxIterations = 10;
        neuralNetwork.train(learningRate,maxIterations, trainingSet);
    }

}
```

Por último mostramos a continuación una tabla con los errores mínimos obtenidos tras 10 ejecuciones del código con diferentes learning rate:

Learning Rate	Error de la red
0.5	0.13471827087689103
0.05	0.135818176283496
0.005	0.16693104129436537
0.0005	0.14008519663495472
0.00005	0.14681340044746524

De estos datos podemos concluir que cuanto mayor es el learning rate, menor es el error de la red.

3. RECONOCIMIENTO DE DÍGITOS

En la práctica se nos pide que, investigando el entorno proporcionado por el profesorado, contestemos las siguientes preguntas.

- ¿Cómo se puede acceder a una determinada imagen del conjunto de entrenamiento?

Se puede acceder a una imagen mediante la función GetRowAt(index) del DataSet, lo que te devuelve una imagen en concreto en formato DataSetRow, también puedes obtener el Array de objetos DataSetRow mediante la función GetRows() e ir iterando entre cada uno de los elementos del conjunto.

- ¿Cuál es su tamaño?

El tamaño de la imagen es de 28*28=784 píxeles.

- ¿Cómo se almacenan los píxeles de la imagen?

Los píxeles de la imagen se almacenan en forma de Array, leyendo la imagen por filas, de izquierda a derecha.

- ¿Cómo se podría visualizar una imagen del conjunto de entrenamiento?

Podemos acceder al Array de double del DataSetRow con la función getInput() y, con un doble bucle for de 28*28, imprimir un carácter cuando el double sea mayor que 0, es decir cuando exista un color en el píxel.

- ¿Qué tamaño tiene que tener la capa de entrada de la red?

La red debe tener un tamaño de entrada de 28*28, que es el tamaño de cada una de las imágenes que van a entrar a la red neuronal.

- ¿Cuántas neuronas de salida tiene que tener la red?

La red debe tener 10 neuronas de salida, numeradas del 0 al 9 y que indicarán, según cuál de ellas se active, el dígito que ha sido interpretado por la red neuronal, lo cual puede ser correcto o no.

Se ha diseñado una función que, a partir de una red entrenada y de una imagen que ha sido reconocida por la red entrenada, devuelve cuál es la neurona con máxima activación de entre las 10 neuronas de salida. El siguiente módulo recibe un Array de double con los valores de la imagen calculados por la red entrenada.

```
public int getMaxPerceptron(double salida[]){
  int perc=-1;
  double max=0;
  for(int i=0;i<salida.length;i++){
      if(salida[i]>max){
         // System.out.println(salida[i]);//Para ver la salida
          perc=i;
          max=salida[i];
      }
  }
  return perc;
}
```

Hemos creado también, una función que calcula el % de elementos del conjunto seleccionado que son correctos tras haber entrenado a la red.

```
  public void test(DataSet ds){
      int i=1;
      boolean[] aciertos= new boolean[ds.size()];

      for(DataSetRow datasr : ds.getRows()){
           //cogemos una imagen y la metemos a _mlp
          _mlp.setInput(datasr.getInput());
          _mlp.calculate();//Calculamos
          double teorica[]= datasr.getDesiredOutput();//Lo que debería ser
          double error=_mlp.getLearningRule().getTotalNetworkError();
          double salida[]= _mlp.getOutput();//La salida de la red
//        System.out.println(i+") Network error: "+error);
```

```
  //        System.out.println("Salida deseada: "+
getMaxPerceptron(teorica));//lo que debería salir
  //        System.out.println("Perceptrón máximo:" +
getMaxPerceptron(salida));//lo que sale

//Me guarda en un array de booleans si se ha detectado bien o no
            if(getMaxPerceptron(teorica)==getMaxPerceptron(salida)){
                aciertos[i-1]=true;
            }
            else{
                aciertos[i-1]=false;
            }
        double count =0;
        for(int j=0;j<aciertos.length;j++){
            if(aciertos[j]){
                count++;
            }
        }
        double rate= (count*100)/aciertos.length;
        System.out.println("% de acierto: "+rate +" ("+ count +" ok; " +
(aciertos.length-count)+" error)");
    }
```

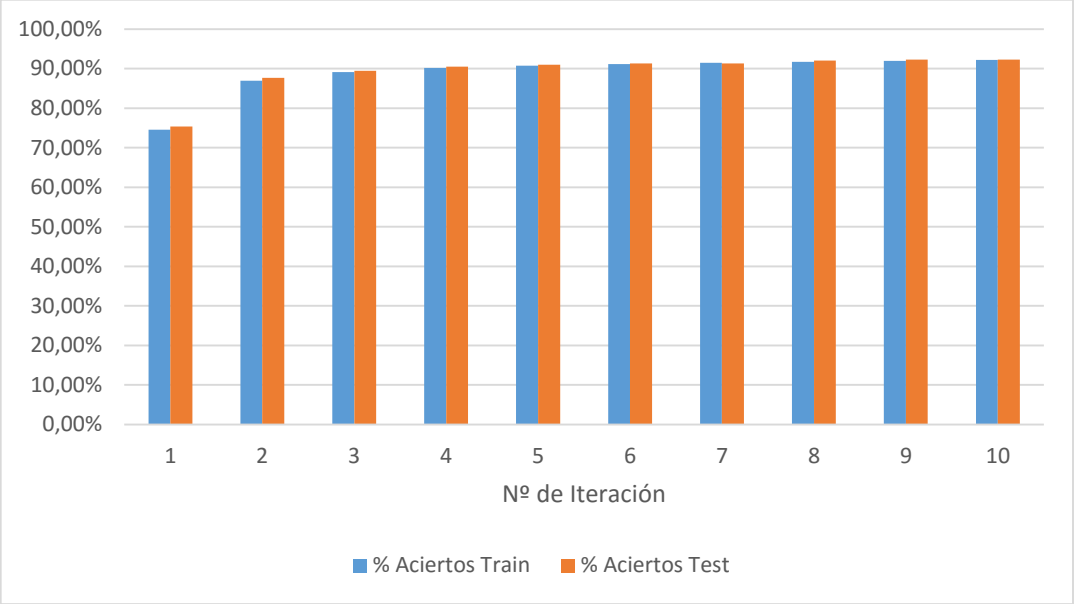
Las dos funciones mostradas anteriormente se han utilizado para experimentar con los posibles valores que pueden tener los elementos de la red neuronal para contestar las preguntas que nos han propuesto en el enunciado de la práctica.

Ejecutando el módulo test tras cada iteración del entrenamiento, podemos visualizar la mejora proporcionada por cada nueva iteración.

A continuación, mostramos las tablas y gráficas que muestran los valores obtenidos de error, % de aciertos para el conjunto de train y el % de aciertos para el conjunto de test. Todos estos ejemplos se realizan con 60.000 ejemplos en el conjunto train y 10.000 ejemplos en el conjunto Test con tasa de aprendizaje al 0.005.

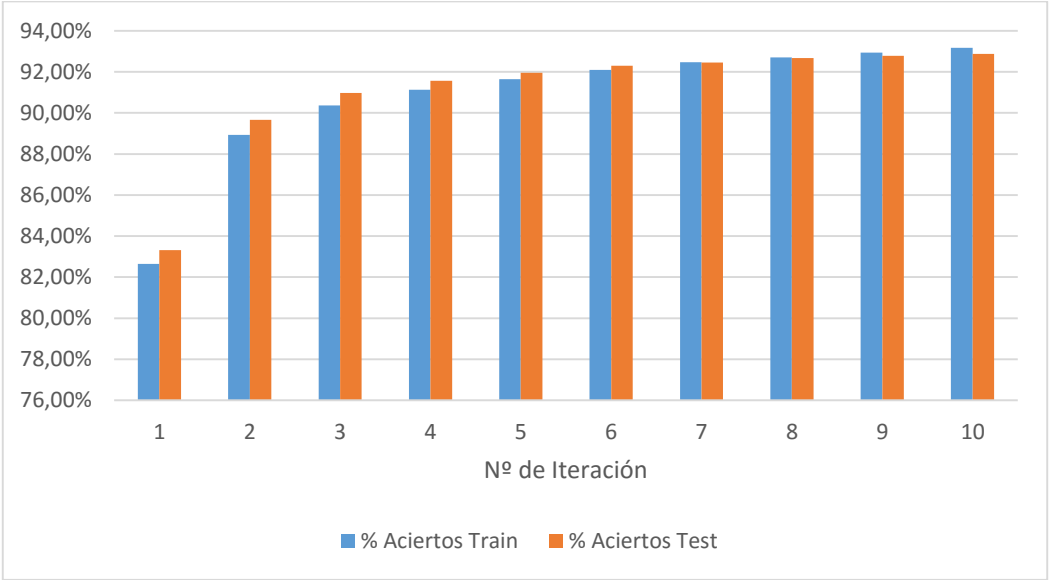
<u>15 nodos en la Capa Oculta</u>

Iteración	% Aciertos Train	% Aciertos Test
1	74,59%	75,38%
2	86,93%	87,71%
3	89,10%	89,47%
4	90,17%	90,49%
5	90,71%	91,02%
6	91,12%	91,35%
7	91,49%	91,35%
8	91,75%	92,04%
9	91,97%	92,25%
10	92,19%	92,32%



20 nodos en la Capa Oculta

Iteración	% Aciertos Train	% Aciertos Test
1	82,65%	83,31%
2	88,93%	89,67%
3	90,37%	90,97%
4	91,13%	91,57%
5	91,64%	91,95%
6	92,09%	92,30%
7	92,47%	92,46%
8	92,70%	92,67%
9	92,94%	92,78%
10	93,17%	92,88%



Hemos probado a aumentar más todavía los nodos en la capa oculta pero conseguíamos peores resultados.

Las redes neuronales utilizan el método de aprendizaje supervisado, ya que utilizan un conjunto de datos proporcionado para aprender acerca de los elementos que van a tener que reconocer en un futuro. De este modo, el usuario puede regular cuanto puede aprender una red neuronal aportando más o menos datos a la misma.

Gracias a la experimentación realizada podemos contestar a las siguientes preguntas:

- ¿Cuántas neuronas tiene que tener en la capa oculta?

En la capa oculta debe tener 20 neuronas, ya que con menos el rendimiento de la red neuronal es menor y con más, el tiempo de cálculo aumenta demasiado.

- ¿Cuál es valor óptimo de aprendizaje para este problema?

Habiendo probado varios valores desde el 0.0001 y valorando cómo se comportaba la red al aumentar dicho valor el valor de tasa de aprendizaje óptimo conseguido es 0.01 ya que es el valor mínimo con el que se obtiene una tasa de aciertos mayor sin sacrificar el tiempo de ejecución del algoritmo.

- ¿Qué ocurre si utilizamos una tasa de aprendizaje menor? ¿Y una mayor?

Si utilizamos una tasa de aprendizaje mayor podríamos dar a la posibilidad de un sobre entrenamiento de la red neuronal, lo que daría a lugar a errores a la hora de reconocer nuevos dígitos. En caso de utilizar una tasa menor, no conseguiríamos obtener la precisión suficiente para que la red neuronal reconozca la mayor cantidad de números posibles sin errores.

- ¿Influye el número de datos de entrenamiento en dicha tasa?

No, ya que la tasa de aprendizaje únicamente indica qué error consideramos correcto a la hora de entrenar la red neuronal. A menor tasa habrá mayor precisión, pero si se aumenta mucho el número de datos puede ocurrir que nunca llegue a alcanzarse dicha tasa, al producirse sobre entrenamiento en la red.

4. BIBLIOGRAFÍA

Materiales de la asignatura Sistemas Inteligentes

Introducción a Neuroph y BackPropagation
http://neuroph.sourceforge.net/Getting%20Started%20with%20Neuroph%202.7.pdf

Librería MNIST
http://yann.lecun.com/exdb/mnist/

Javadoc del proyecto Neuroph
http://neuroph.sourceforge.net/javadoc/index.html