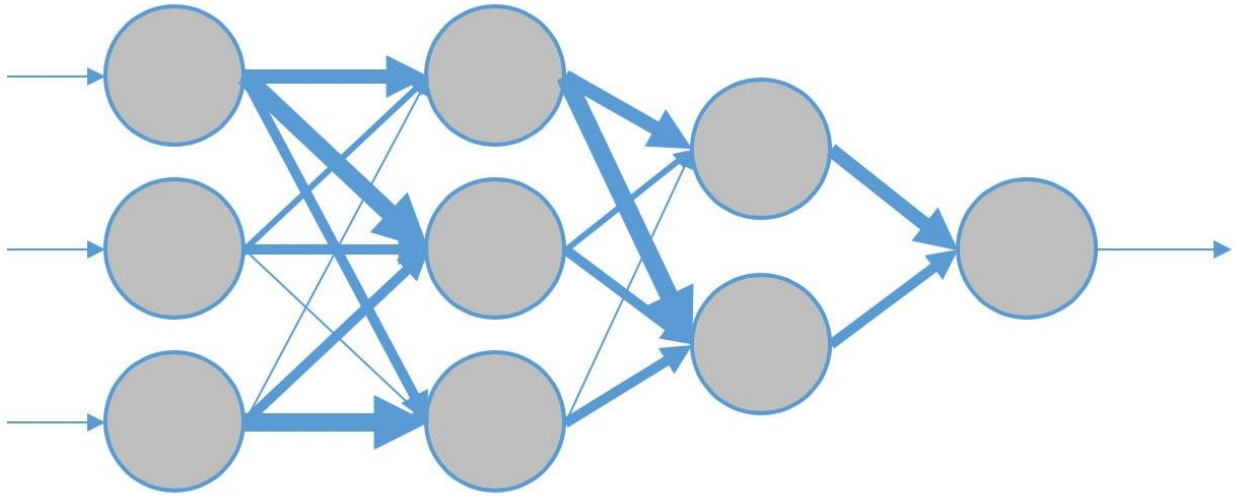
Redes Neuronales



Grupo A:
Adrián Racero Serrano
Santiago Ponce Arrocha
Carlos Velasco Hurtado
Juan Manuel Cardeñosa Borrego

Índice

Creación del perceptrón

```
int numInputs = 2;
int numOutputs = 1;
int numHiddenNeurons = 3;
double learningRate = 0.01;
double maxError = 0.01;
int maxIterations = 200;
```

```
// Create new simple perceptron network
MultiLayerPerceptron neuralNetwork =
        new MultiLayerPerceptron(TransferFunctionType.SIN,
                numInputs, numHiddenNeurons, numOutputs);

BackPropagation backpropagation = neuralNetwork.getLearningRule();
backpropagation.setLearningRate(learningRate);
backpropagation.setMaxError(maxError);
backpropagation.setMaxIterations(maxIterations);
```

El perceptrón tiene una capa oculta, con 3 neuronas

Establecemos el máximo de iteraciones de aprendizaje, error máximo y tasa de aprendizaje

Conjuntos de entrenamiento y validación

```
// Create training and validation set
DataSet trainingSet = new DataSet(2,1);
DataSet validationSet = new DataSet(2,1);

// Añadimos datos al conjunto de entrenamiento
trainingSet = createDataSet(1000);

// Añadimos datos al conjunto de validación
validationSet = createDataSet(1000);
```

```
private static DataSet createDataSet(int size) {
    DataSet dataSet = new DataSet(2, 1);
    Random random = new Random();

    for(int i = 0; i < size; i++) {
        double x = random.nextDouble()*Math.PI * sign(random);
        double y = random.nextDouble()*Math.PI * sign(random);
        double f = Math.sin(x) * Math.cos(y);
        dataSet.add(new DataSetRow(new double[] {x,y}, new double [] {f}));
    }


    return dataSet;
}
```

La función a estudiar tiene dos entradas y una salida

f(x,y) = sen(x) * cos(y)

Se generan valores aleatorios (muestras) de x e y entre –pi y pi

Se calcula la salida de la función y se añade al conjunto

Entrenamiento de la red

```java
// Entrenar la red neuronal
for (int epoch = 0; epoch < 30; epoch++) {
    neuralNetwork.learn(trainingSet);

    double trainingError = neuralNetwork.getLearningRule().getPreviousEpochError();
    double validationError = validate(neuralNetwork, validationSet);
    pw.println("Epoca #" + epoch + " - Error entrenamiento: " + trainingError + " - Error validacion: " + validationError);
    pwEntrenamiento.println(trainingError);
    pwValidacion.println(validationError);

    System.out.println("Epoca #" + epoch + " - Error entrenamiento: " + trainingError + " - Error validacion: " + validationError);
}
```

Entrenamos a la red neuronal durante 30 etapas.

La función validate calcula el error cuadrático medio.

Conjunto de prueba

```
DataSet testSet = new DataSet(2, 1); // 2 entradas, 1 salida

double xStep = 2 * Math.PI / 100;
double yStep = 2 * Math.PI / 100;

double x = -Math.PI;
double y = -Math.PI;

for (int i = 0; i < 10000; i++) {
    double[] input = {x, y};
    double[] output = {Math.sin(x) * Math.cos(y)};
    DataSetRow row = new DataSetRow(input, output);
    testSet.add(row);

    y += yStep;

    if ((i + 1) % 100 == 0) {
        y = -Math.PI;
        x += xStep;
    }
}
```

Generamos el conjunto de prueba

Todas las entradas están separadas uniformemente

Cálculo del error

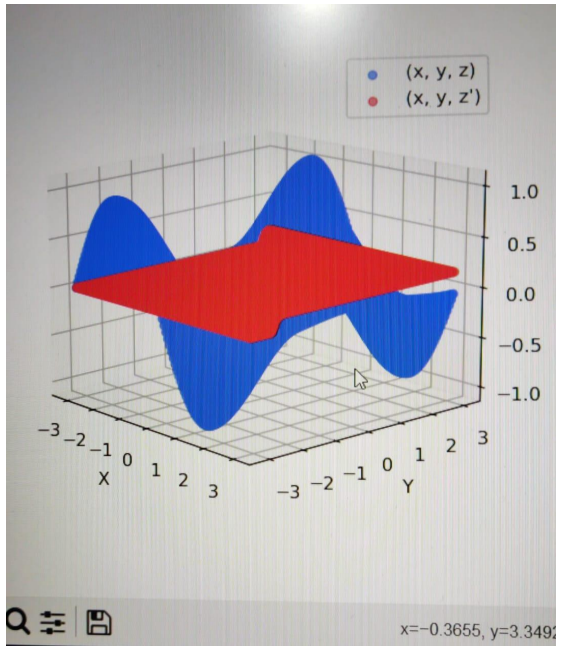
```
// Para generar el grafico 3D, guardamos x,y, y el resultado
// de la funcion que debería dar y el que da la red neuronal
for(DataSetRow row : testSet.getRows()) {
    neuralNetwork.setInput(row.getInput());
    pwX.println(row.getInput()[0]);
    pwY.println(row.getInput()[1]);
    f.println(row.getDesiredOutput()[0]);
    neuralNetwork.calculate();
    double[] output = neuralNetwork.getOutput();
    fRed.println(output[0]);
}

double ECM = validate(neuralNetwork, testSet);
```

Calculamos el error obtenido respecto al conjunto de prueba

Problemas surgidos

- Desconocimiento de la librería.
- Al principio utilizamos otra función de activación y el error era muy alto.
- Sobreajuste de la red al entrenarla en exceso.

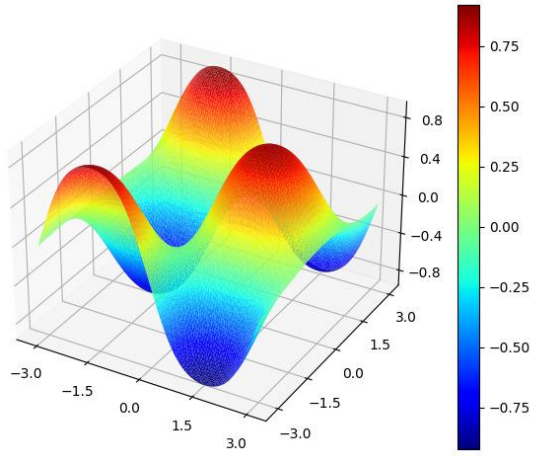


Conclusión

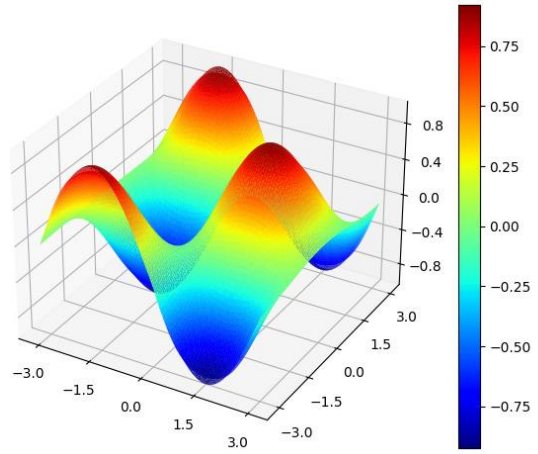
Ha sido una práctica muy entretenida a pesar de no tener mucha información acerca de la librería neuroph. Aclara los contenidos vistos en clase.

Parte voluntaria

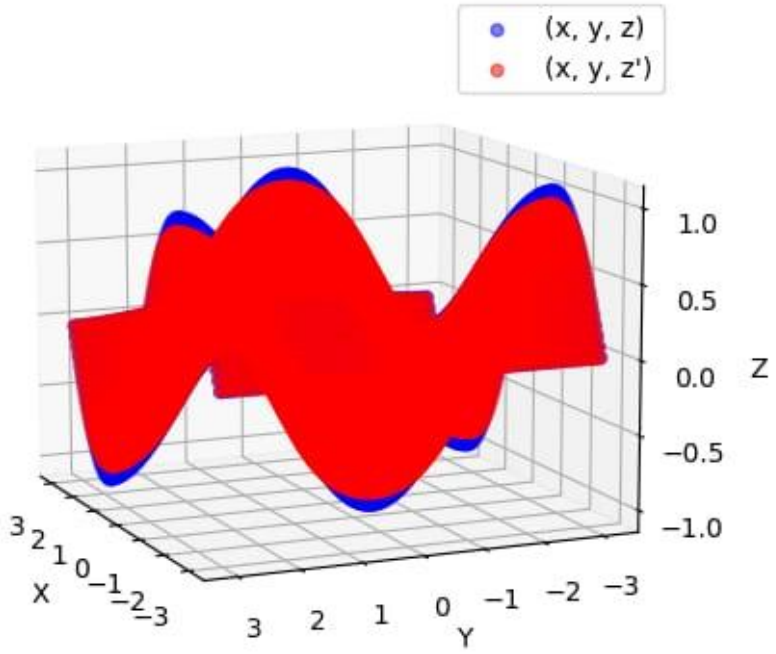
Gráfica esperada

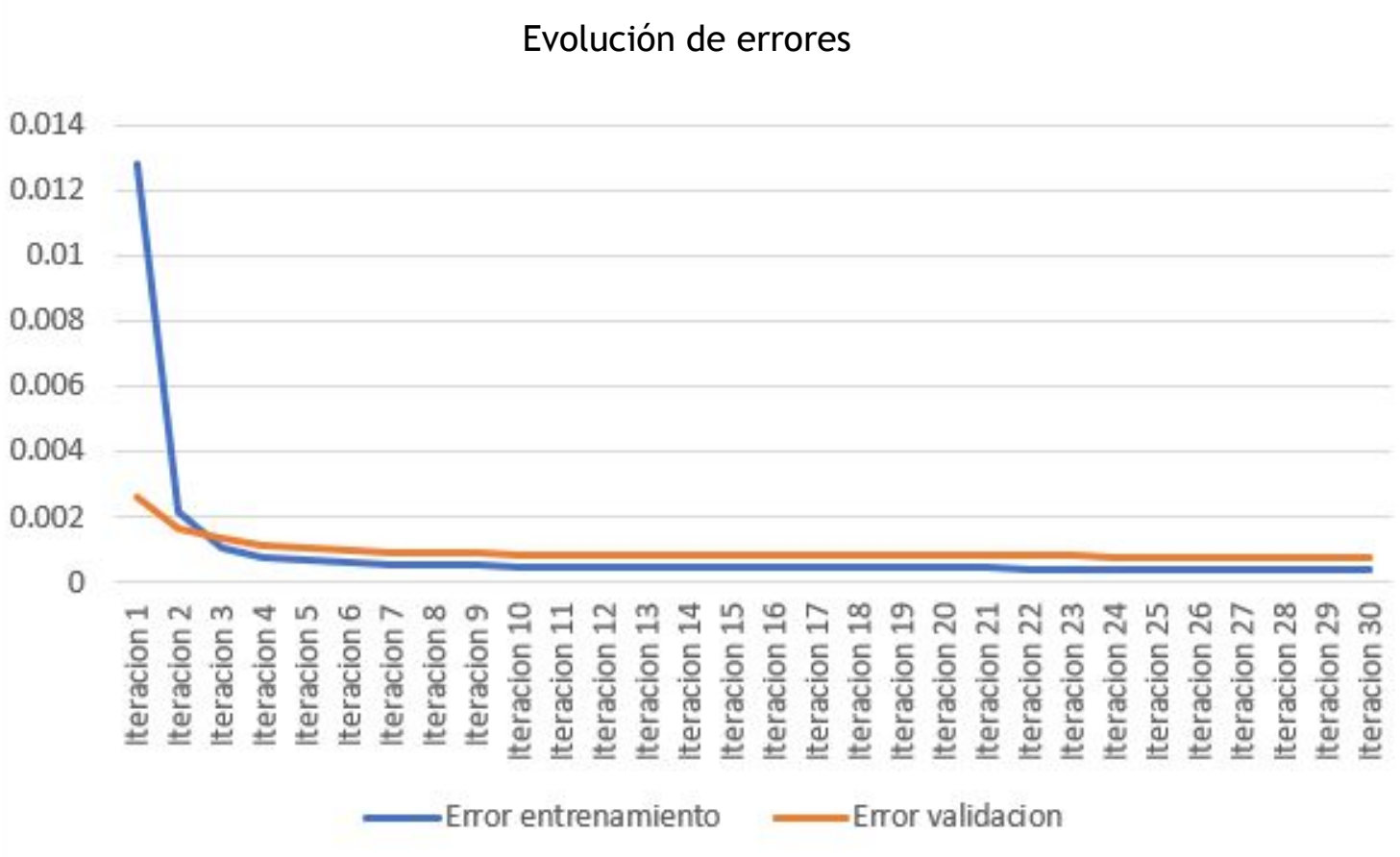


Gráfica obtenida



Azul: Gráfica esperada
Rojo: Gráfica de la red neuronal




Evolución de errores
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
Iteracion 1
Iteracion 2
Iteracion 3
Iteracion 4
Iteracion 5
Iteracion 6
Iteracion 7
Iteracion 8
Iteracion 9
Iteracion 10
Iteracion 11
Iteracion 12
Iteracion 13
Iteracion 14
Iteracion 15
Iteracion 16
Iteracion 17
Iteracion 18
Iteracion 19
Iteracion 20
Iteracion 21
Iteracion 22
Iteracion 23
Iteracion 24
Iteracion 25
Iteracion 26
Iteracion 27
Iteracion 28
Iteracion 29
Iteracion 30
Error entrenamiento
Error validacion