Feed Forward Neural Network -documentation-

Carmina Codre
Faculty of Mathematics and Computer Science
Babes-Bolyai University
Cluj-Napoca, Romania

1 Problem Definition

Classification is an unsupervised machine learning problem in which for each sample from a data set, a label must be assigned. The target outputs are known in advance and the datasets are labelled.

The aim of this project is the design and implementation of a model which could be used for both binary and multiclass classification.

Feed Forward Neural Networks represent a category of artificial neural networks which can be used for classification, as well as for other tasks and thus the chosen model for solving the problem is such a network.

2 Specification

3 Functional Requirements

The implementation of the feed-forward neural network should comprise the following characteristics:

- Should be formed by one input layer, one hidden layer and one output layer
- The possibility of choosing the activation function of the layers
- Implementation of the backpropagation algorithm together with the gradient descent and adjustment of the number of iterations performed
- Implementation of regularization
- Means of adjusting the learning rate
- Means of adjusting the regularization parameter
- Means of adjusting the size of each layer

4 Nonfunctional Requirements

The nonfunctional requirements of the project are:

- Scalability. We want to develop a system that can be further extended to incorporate other functionalities.
- Performance. The system should deal with big amounts of data in a reasonable amount of time
- Usability and portability. The resulting implementation should be easily used on other systems and should be comprehensible

5 Datasets

In order to test the model and take a look on how it performs the following datasets have been used.

5.1 Mock dataset composed of two classes

For generating a toy dataset , the Scikit learn library has been used. The chosen toy dataset is the one which scatters points in a two dimensional space in form of two interleaving half circles, each half circle representing thus a class. This is often used for clusterization and classification in order to visualize the results of the models. For testing the feed forward neural network, 200 samples have been generated from this sample space with a noise ratio of 0.20.

The documentation of this module can be found here:

http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html

The data can be visualized in the scatter plot from Figure.1.

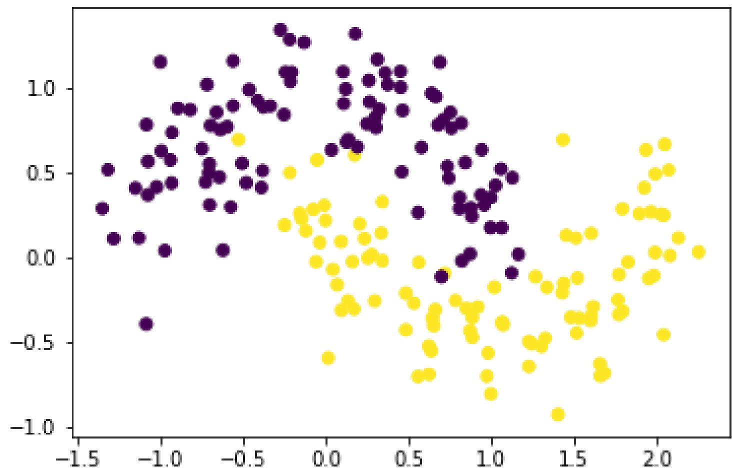

Fig. 1. Scatter plot of mock data

5.2 Iris dataset

For testing the model on the task of multiclass categorization, Iris dataset is used. The dataset can be found here:

https://archive.ics.uci.edu/ml/datasets/iris

The dataset is composed of 150 samples representing flowers.. Each sample contains 4 features, more exactly ’sepal length (cm)’, ’sepal width (cm)’, ’petal length (cm)’, ’petal width (cm)’. Each sample belongs to one of the following classes: ’setosa’ ’versicolor’ ’virginica’. The values of the attributes are real and there are no missing values.

The dataset is widely used for testing models for multiclass categorization and it does not require a lot of preprocessing.

A visualization of the features is provided in the scatter plots from Figure.2

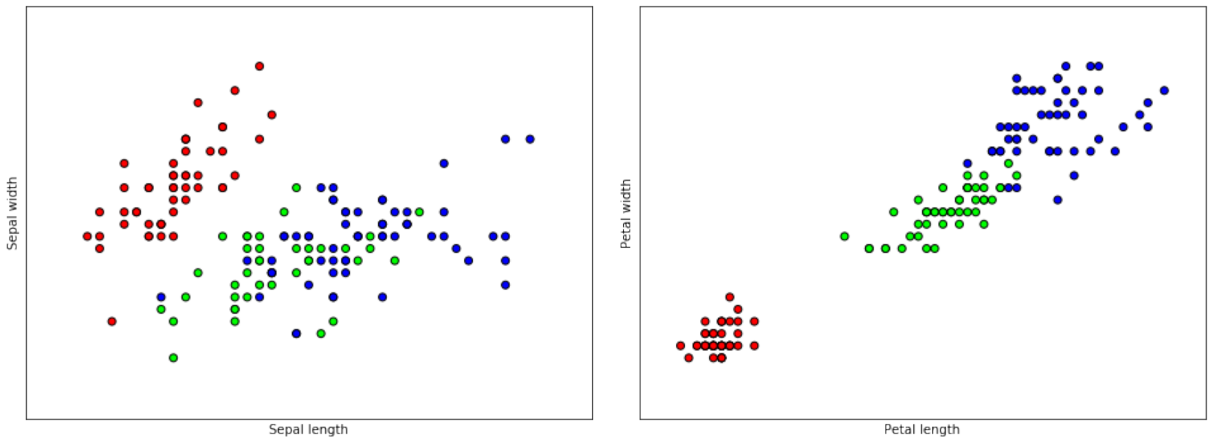

Fig. 2. Scatter plots of iris data

6 Analysis and Design

6.1 Preprocessing

Since the dataset is already cleaned and prepared the only preprocessing step tried is the normalization of the features.

6.2 Model

6.2.1 Architecture

The model is designed as in Figure.3.

The **input layer** is not trained, but has the purpose of feeding the input to the higher layers.

The **hidden layer** contains multiple units. Each of the input units is connected with each of the units from the hidden layer. The size of this can be adjusted in order to give more representational power to the model.

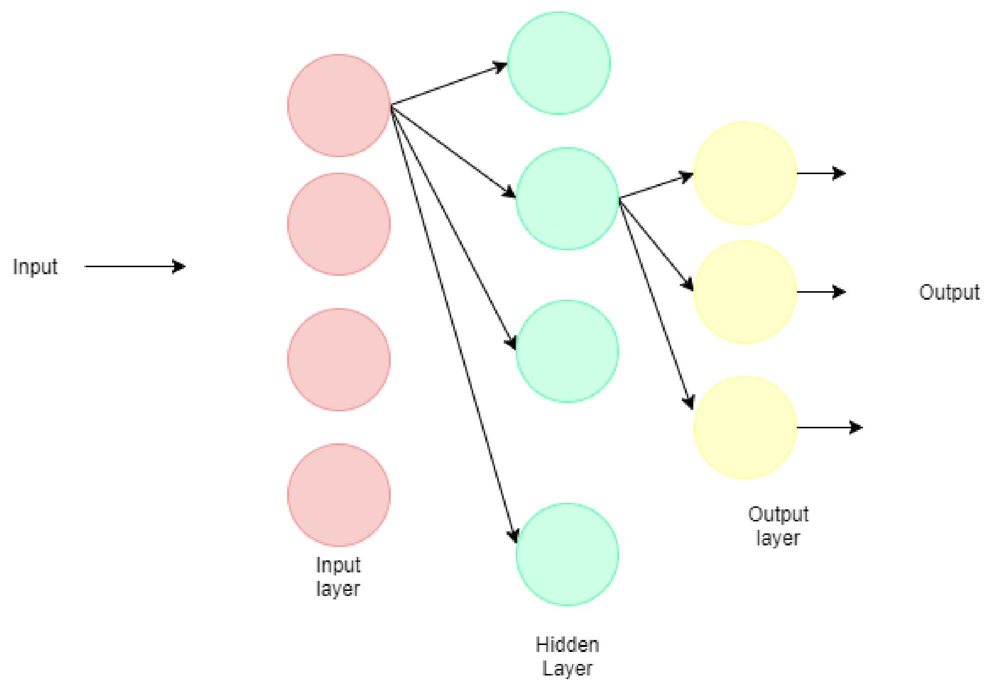

Fig. 3. Feed Forward Neural network

The **output layer** provides the output of the network. Usually its size must be equal to the number of classes such that the output can be treated as a vector of probabilities yielding the conditional probability of a class given a certain sample $Pr(c|sample)$.

A network with only one hidden layer has been used due to the fact that this is able to represent any continuous function on real numbers with its output in a real interval.

6.2.2 Learning Algorithm

The learning algorithm used is batch gradient descent. This has been chosen in order to simplify the learning.

Algorithm 1 Batch Gradient Descent

```
Require: Learning rate $\epsilon$
Require: Initial weights $\Theta$
  for number of interations do
    Take the whole training set $x_1, x_2, ..., x_m$ with corresponding labels $y_1, y_2, ..., y_m$
    Compute the loss $L$
    Compute the derivatives of the loss with respect to all parameters $\Theta$: $\frac{\partial L}{\partial W2}$ $\frac{\partial L}{\partial b2}$ $\frac{\partial L}{\partial W1}$ $\frac{\partial L}{\partial b1}$
    Apply update
    $W2 \leftarrow W2 - \epsilon\frac{\partial L}{\partial W2}$
    $b2 \leftarrow b2 - \epsilon\frac{\partial L}{\partial b2}$
    $W1 \leftarrow W1 - \epsilon\frac{\partial L}{\partial W1}$
    $b1 \leftarrow b1 - \epsilon\frac{\partial L}{\partial b1}$
  end for
```

6.2.3 Activations

Activation functions are used upon the weighted summation of the inputs at each unit. These functions provide the output of a neuron.

Their derivatives have also been used for propagating the error through the network and updating the weights.

The activations that have been implemented are the following:

Hiberbolic Tangent

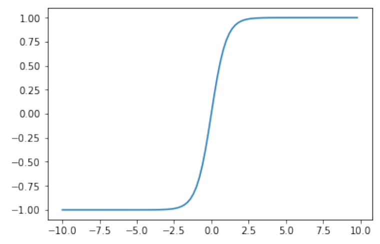

Fig. 4. Tanh

In Figure.4 can be seen the hiperbolic tangent plotted on the interval $[-10, 10]$.

Sigmoid

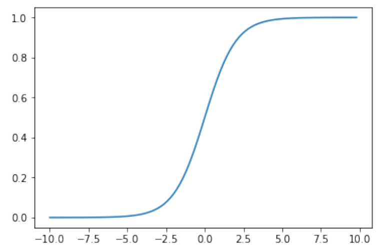

Fig. 5. Sigmoid

It can be easily used for binary classification at the output layer with good results. Its output is in the range $[0, 1]$ as it is shown in Figure.5, where the function is plotted on the interval $[-10, 10]$. This function is given by the following formula:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Rectified Linear Unit

Its output belongs to the range $[0, \infty]$ ReLu activation function is given by:

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Figure.6 shows the ReLu function on the interval $[-10, 10]$.

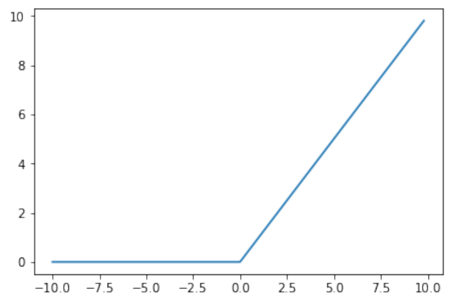

Fig. 6. Rectified Linear Unit (ReLU)

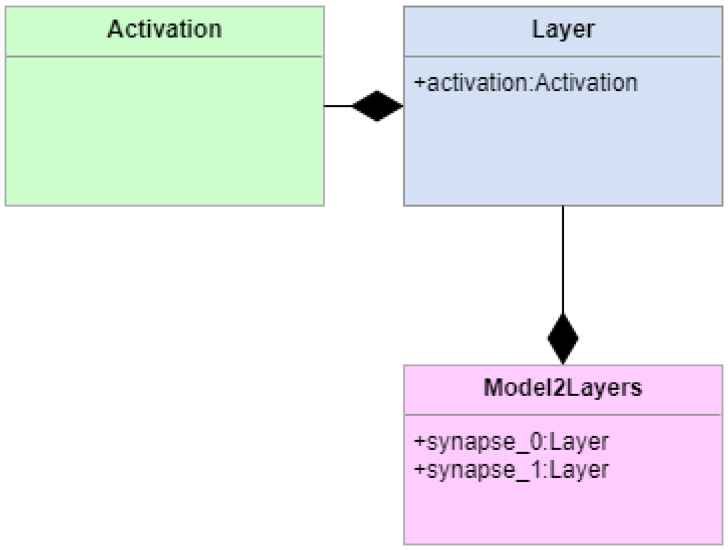

Fig. 7. Class Diagram

6.3 Class Diagram

There are three main classes used as it can be seen from Figure.7.

The Activation class has the responsibility of keeping track of which function has to be used and connects it to its derivative.

The Layer class has the responsibility of keeping track of which activation the layer uses and the parameters of the layer.

The Model2Layers class implements the learning algorithm together with means of computing the loss, the accuracy and predictions.

7 Implementation

7.1 Environment and libraries used

The environment used for developing the project is Python version 2.7 The implementation is entirely done in Jupyter notebooks. These can also be visualized in other environments for Python. Jupyter notebook can be installed together with Anaconda which provides an open source distribution of Python. This can be found at:

https://anaconda.org/anaconda/python

The libraries and modules used are the following:

- numpy
- sklearn.datasets
- matplotlib.pyplot

The jupyter notebooks can be started by writing the following command in the command line:

```
jupyter notebook
```

The libraries listed can be installed using:

```
conda install name_of_library
```

7.2 Learning algorithm

```
# Feed forward through layers 0, 1, and 2
output = self.feed_forward(X)

#compute error
error_2 = output - y
layer_2_delta = error_2*self.synapse_1.activation.get_derivative()
                                        (self.synapse_1.output)

#backprop the error to layer 1
error_1 = layer_2_delta.dot(self.synapse_1.weights.T)
layer_1_delta = error_1*self.synapse_0.activation.get_derivative()
                                        (self.synapse_0.output)

dW2 = self.synapse_1.input.T.dot(layer_2_delta)
dW1 = self.synapse_0.input.T.dot(layer_1_delta)

# Add regularization terms
dW2 += self.reg_lambda * self.synapse_1.weights
dW1 += self.reg_lambda * self.synapse_0.weights

#update the weights
self.synapse_1.weights -= self.learning_rate * dW2
self.synapse_1.biases -= self.learning_rate *
(np.ones((1,self.synapse_1.input.shape[0])).dot(layer_2_delta))
self.synapse_0.weights -= self.learning_rate * dW1
self.synapse_0.biases -= self.learning_rate *
(np.ones((1,self.synapse_0.input.shape[0])).dot(layer_1_delta))
```

8 Testing and Results Obtained

8.1 Mock Dataset

On the mock dataset, the network reaches fast the accuracy 1 and is able to distinguish between the two different classes. By using a learning rate of 0.8, 5 hidden units, a regularization factor of l2 of 0.0005 the model prints the following results while training:

```
Error    after 0 iterations:0.336960885152
Accuracy after 0 iterations:1.0
Error    after 10000 iterations:0.0160790132269
Accuracy after 10000 iterations:1.0
Error    after 20000 iterations:0.0113440622282
Accuracy after 20000 iterations:1.0
Error    after 30000 iterations:0.0137852590084
Accuracy after 30000 iterations:1.0
Error    after 40000 iterations:0.0121776696565
Accuracy after 40000 iterations:1.0
Error    after 50000 iterations:0.0127486359326
Accuracy after 50000 iterations:1.0
```

8.2 Iris Dataset

The learning rate used is 0.001, the number of hidden units is 5 and the regularization factor of l2 is 0.0005

8.2.1 10-Fold Cross Validation on Initial Data

```
    Average accuracy obtained by 10-fold cross-validation
    0.9733331
```

8.2.2 10-Fold Cross Validation on Normalized Data

```
    Average accuracy obtained by 10-fold cross-validation
    on normalized data 0.9800001
```