Artificial Neural Network
Backpropagation

Diego Iturriaga - José Núñez - Roberto Ureta
December 21, 2018

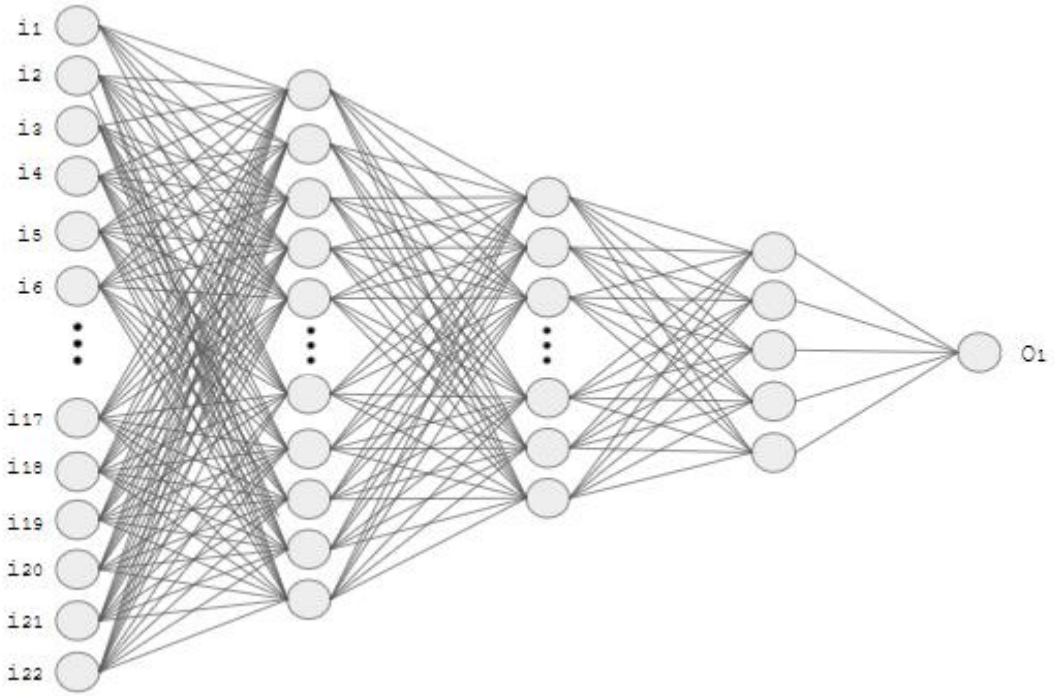
Resume

The objective of this project is analyze a serie of parameters about mushrooms that could to be poisoned or edibles. For do this, we going to train a neural network using backpropagation and a dataset obtained from the site UCI (Machine Learning Repository) that provide 8124 dates about differents mushrooms that are poisoned and not.

Development

Structure

In the first instance, we plan how would be the structure of the neural network, whiching a network that contains two external layers (input and output) and a hidden one that is in the middle of these. Each has 22, 15, 10, 5 and 1 nodes respectively, these are directly linked to each node of the layer directly adjacent. Graphically we have to:



Each of the connections that we can see in the graph have a weight that is set randomly, which will vary according to the training being carried out.

Training

To carry out the neural network training, we need a dataset with true data about the characteristics of a mushroom are needed, these data consist of 22 attributes that define whether a mushroom is poisonous or edible, where each attribute corresponds to different properties of the mushroom, which are represented by characters. We can known its meaning thanks to the description of the data of the UCI site. These are:

1) **Cap-shape:** bell = b, conical = c, convex = x, flat = f, knobbed = k, sunken = s
2) **cap-surface:** fibrous = f, grooves = g, scaly = y, smooth = s
3) **cap-color:** brown = n, buff = b, cinnamon = c, gray = g, green = r, pink = p, purple = u, red = e, white = w, yellow = y
4) **bruises?:** bruises = t, no = f
5) **odor:** almond = a, anise = l, creosote = c, fishy = y, foul = f, musty = m, none = n, pungent = p, spicy = s
6) **gill-attachment:** attached = a, descending = d, free = f, notched = n
7) **gill-spacing:** close = c, crowded = w, distant = d
8) **gill-size:** broad = b, narrow = n
9) **gill-color:** black = k, brown = n, buff = b, chocolate = h, gray = g, green = r, orange = o, pink = p, purple = u, red = e, white = w, yellow = y
10) **stalk-shape:** enlarging = e, tapering = t
11) **stalk-root:** bulbous = b,club = c, cup = u, equal = e, rhizomorphs = z, rooted = r, missing = ?
12) **stalk-surface-above-ring:** fibrous = f, scaly = y, silky = k, smooth = s
13) **stalk-surface-below-ring:** fibrous = f, scaly = y, silky = k, smooth = s
14) **stalk-color-above-ring:** brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y
15) **stalk-color-below-ring:** brown = n, buff = b, cinnamon = c, gray = g, orange = o, pink = p, red = e, white = w, yellow = y
16) **veil-type:** partial = p, universal = u
17) **veil-color:** brown = n, orange = o, white = w, yellow = y
18) **ring-number:** none = n, one = o, two = t
19) **ring-type:** cobwebby = c, evanescent = e, flaring = f, large = l, none = n, pendant = p, sheathing = s, zone = z
20) **spore-print-color:** black = k, brown = n, buff = b, chocolate = h, green = r, orange = o, purple = u, white = w, yellow = y
21) **population:** abundant = a, clustered = c, numerous = n, scattered = s, several = v, solitary = y
22) **habitat:** grasses = g, leaves = l, meadows = m, paths = p, urban = u, waste = w, woods = d

To make a better work with this data we decided to give each character a normalized value that is between 0 and 1 that is given sequentially, for example:

> **Cap-shape:** bell = 0.1, conical = 0.2, convex =0.3, flat = 0.4, knobbed = 0.5, sunken = 0.6

Implementation

For train, we must first have enough data to perform the tests and trainings, so we proceeded to randomly select 75% of the data for training, 15% test and 10% validation.

Then, the neural network was structured with 22 input neurons, who receive the attributes associated with a mushroom, so each input neuron receives an attribute which is previously transformed into a representative number. In the neural network, 3 internal layers of nodes and one output layer were structured.The first internal layer have 15 neurons, the second internal layer have 10 neurons, the third layer possesses 5 nodes and the output layer only have 1 neuron. Then the input layer connects to the first layer of hidden neurons and the latter connects to the second layer of hidden neurons too, so that the layers connect to each other until they reach the output neuron.

Once the weights are assigned, the entries of the internal layers are calculated from the random weights of the edges that connect them and the input values in the input neuron layer. Then, we proceed to multiply the value of the first input neuron with the value of the weight that connects it to the first node in the first hidden layer and the value is stored in this last node, and then we calculate the same value with the second input node multiplying his respective values with the weight that connects it to the first node of the internal layer and this value is added with the previous value product of the multiplication of the first neuron that connects to the first input node.This is done with all the nodes until you get the sum of all the weights multiplied with the corresponding input value, to store it in the internal layer node.

$$n1_{hiddenLayer1} = i1 * w1_{n1} + i2 * w2_{n1} + \cdots + i22 * w2_{n1}$$

$$n2_{hiddenLayer1} = i1 * w1_{n2} + i2 * w2_{n2} + \cdots + i22 * w2_{n2}$$

$$\vdots$$

$$n15_{hiddenLayer1} = i1 * w1_{n22} + i2 * w2_{n22} + \cdots + i22 * w2_{n22}$$

Multiplication for the first layer of internal neurons, were we use the input values that are used to calculate the values of each neuron.

$$n1_{hiddenLayer2} = n1_{hiddenLayer1} * w1_{n2} + n2_{hiddenLayer1} * w2_{n2} + \cdots + n15_{hiddenLayer1} * w2_{n2}$$

$$n2_{hiddenLayer2} = n1_{hiddenLayer1} * w1_{n2} + n2_{hiddenLayer1} * w2_{n2} + \cdots + n15_{hiddenLayer1} * w2_{n2}$$

$$\vdots$$

$$n10_{hiddenLayer2} = n1_{hiddenLayer1} * w1_{n2} + n2_{hiddenLayer1} * w2_{n2} + \cdots + n15_{hiddenLayer1} * w2_{n2}$$

Multiplication for the second layer of internal neurons, we use the values of the previous layer that are used to calculate the respective values neurons.
This process is repeated throughout the neural network. For this, 4 matrices were implemented that store the weights and represent the connections between the neurons of the network.

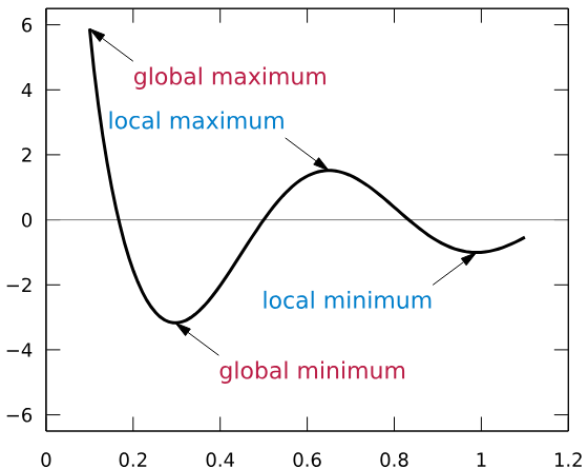
Sigmoid Function

To adjust the value of the weight corresponding to the association between each neuron with the adjacent (of the next layer), the Sigmoid function is used which provides numerical values between 0 to 1. We should to say that if the input is null also the function will present a stimulation.
This function is defined by the next formula, where the variable t corresponds to the weight to be adjusted:

$$P(t) = \frac{1}{1 + e^{-t}}$$

BackPropagation

This algorithm modify the different outputs of each layer through a network of neurons from back to front. This seeks to improve the output error using the method of stochastic gradient descent.



The optimal value to be obtained for each weight is where the error reaches a global minimum in the function.

During all the training, the weights are updated according to a learning rate (0.1) where each update tries to reach the global minimum. The following formulas are used for this update procedure:

- Stochastic gradient of descending output

$$gradient = (desiredValue - OutputNetwork) \cdot OutputNetwork \cdot (1 - OutputNetwork)$$

- Stochastic gradient of descending hidden

$$hiddenGradient = \left(\sum_i gradients_i \cdot previousWeights\right) \cdot OutputPrevious \cdot (1 - OutputPrevious)$$

The algorithm ends when it fulfill any of the next conditions:

- The last 3 accuracy values are equals
- The test success (done every 250 times) is greater than 80%
- Reach a maximum of 2000 times

Results

Having made several tests with different random data choices, it is possible to say that the neural network has an average success between 70 and 72 and with peaks of 83% for the cases tested. However, these results are strongly related to the disposition of the data itself, which influences notoriously in the percentages of success that the neural network can grant, this last because according to the order of the same we can have as positive peaks (83%) as negative (47%), which means that the most important thing when training a neural network is to be able to choose very well the data with which we are going to work and analyze the distribution of the data.

Execution	1	2	3	4	5	6	7	8	9	10	Average
Percentage of Successes in Training	67%	71%	58%	75%	60%	75%	69%	64%	76%	74%	69%
Percentage of Successes in Tests	62%	75%	62%	83%	67%	75%	65%	65%	77%	73%	71%
Percentage of Successes in Validation	63%	76%	61%	83%	67%	75%	60%	68%	78%	74%	71%
Average for each execution	64%	74%	60%	81%	65%	75%	64%	66%	77%	74%	70%