**Prediction of Fertility using Neural Networks**

**Introduction**: In this project, we demonstrate how neural networks can be used in the medical field to predict fertility in adult males. The dataset was obtained from the UCI Machine Learning Repository. It consists of data procured by analysis of samples provided by 100 volunteers in accordance with the WHO 2010 criteria.

This fertility prediction scheme is based on various factors such as socio-demographic data, environmental factors, health status and life habits.

**Problem Statement:** The goal is to train the neural network to predict whether a patient is fertile or not, based on the aforementioned parameters, expressed in the form of nine inputs as stated below. The input values are as indicated in the brackets.

1. Season in which the analysis was performed.

i. Winter (-1)

ii. Spring (-0.33)

iii. Summer (0.33)

iv. Fall (1)

2. Age at the time of analysis

i. Between 18-36 (1)

ii. 0 otherwise

3. Diseases in childhood (chicken pox, measles, mumps, polio)

i. Yes (1)

ii. No (0)

4. Accident or serious trauma

i. Yes (1)

ii. No (0)

5. Surgical Intervention

i. Yes (1)

ii. No (0)

6. High fevers in the last year

i. less than 3 months ago (-1)

ii. more than 3 months ago (0)

iii. No (1)

7. Frequency of alcohol consumption

i. High (1)

ii. Low (0)

8. Smoking tendencies

i. Never (-1)

ii. Occasional (0)

iii. Daily (1)

9. Number of hours spent sitting in a day

i. more than 10 (1)

ii. less than 10 (0)

The output, either a 1 or a 0, indicates a normal or altered diagnosis.

**The Neural Network**: For this problem, we have used an Error Back Propagation Feed Forward Neural Network, owing to the availability of a large dataset, and the fact that this is a supervised learning network which can be trained to achieve a balance between the ability to respond correctly to the input patterns that are used for training and the ability to give reasonable responses to input that is similar, but not identical to that used in training.

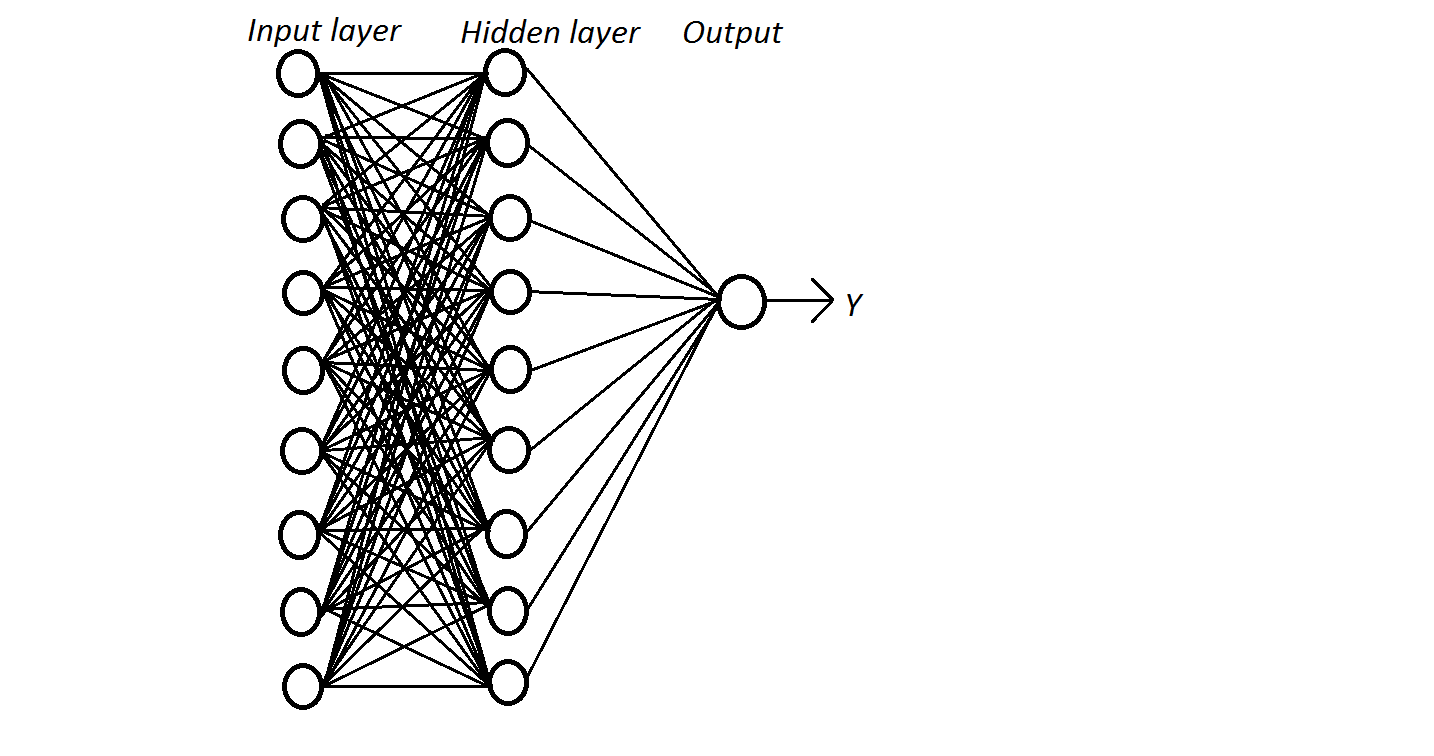
The training of this network by backpropagation involves three stages -

1. The feedforward of the input training pattern

2. The calculation and back propagation of the associated error

3. Adjustment of weights

After training, the application of the net involves only the computations of the feed forward phase. Even if training is slow, a trained net can produce its output very rapidly.



Structure of the Neural Network

**The Algorithm**:

Each training set can be represented as **(x,t)**,where x is the input vector and t is the target (desired) output. Weight from unit j to unit i is indicated as wij.

* **Definitions:**

|  |  |  |
| --- | --- | --- |
| * the **error** signal for unit j: |  |  |
| * the (negative) **gradient** for weight wij: |  | Delta w_{ij} = -partial E/partial w_ij |
| * the set of nodes **anterior** to unit i: |  |  |
| * the set of nodes **posterior** to unit j: |  | P_j = {i : exists w_ij} |

* **The gradient.** .

To compute this gradient, we thus need to know the activity and the error for all relevant nodes in the network.

* **Forward activation.** The activity of the input units is determined by the network's external input **x**. For all other units, the activity is propagated forward:

y_i = f_i(sum_{j in A_i} w_ij y_j)

Note that before the activity of unit i can be calculated, the activity of all its anterior nodes (forming the set Ai) must be known. Since feedforward networks do not contain cycles, there is an ordering of nodes from input to output that respects this condition.

* **Calculating output error.** Assuming that we are using the sum-squared loss



the error for output unit o is simply



* **Error backpropagation.** For hidden units, we must propagate the error back from the output nodes (hence the name of the algorithm). Again using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:



Of the three factors inside the sum, the first is just the error of node i. The second is

while the third is the derivative of node j's activation function:

partial y_j/partial net_j =
 partial f_j(net_j)/partial net_j = f'_j(net_j)

Putting all the pieces together we get



Note that in order to calculate the error for unit j, we must first know the error of all its posterior nodes (forming the set Pj). Again, as long as there are no cycles in the network, there is an ordering of nodes from the output back to the input that respects this condition. For example, we can simply use the reverse of the order in which activity was propagated forward.

**Results, Conclusion and Limitations**: The network was tested with inputs from the dataset, as well as inputs similar to the dataset, and worked with 90% efficiency.

Some limitations of EBP NN encountered while working are-

1. 100% efficiency could not be achieved, as the network cannot evolve in a direction which is not pre-defined.

2. The network doesn't give precise numeric outputs always.

3. If the dataset is not big enough, using EBP NN is not advisable.