Implementation of Little Red Riding Hood Problem in Neural Networks

1.0 About

Little Red Riding Hood problem introduced by is an example using in neural networks used to show how the decisions are made according to a given set of input by using previous experiences. It is based on a well known Fairytale , 'The Little Red Riding Hood'. In the story,

"Riding Hood has to solve the problem of how to recognize the wolf, the grandmother, and the woodcutter and respond appropriately. To this end, she possesses six detectors which register the presence or absence (values of 0 or 1) of the following signals of a presented object: big ears, big eyes, big teeth, kindliness, wrinkles, handsomeness. If Riding Hood detects an object that is kindly, wrinkled, and has big eyes, which are supposed to be the features of grandmother, she responds by behaving in the following ways: approaching, kissing her cheeks, offering her food. With an object that has big ears, big eyes, and big teeth, and is probably the wolf, Riding Hood is supposed to run away, scream, and look for the woodcutter. Finally, if the object is kindly, handsome, and has big ears, like the woodcutter, Riding Hood is supposed to approach, offer food, and flirt " .[1]

2.0 Implementation

Here I have used a neural network with 3 layers seven input nodes, six output nodes and five hidden nodes.

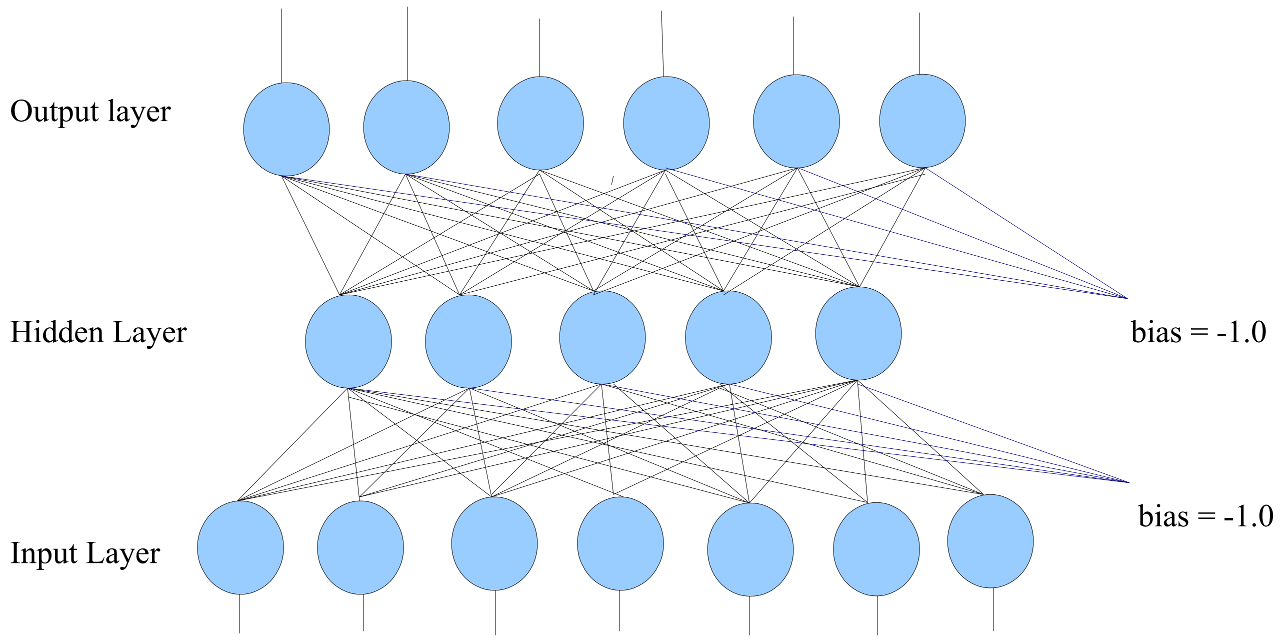

figure : 2.0 the structure of the neural network

Then the network is trained using back propagation. First the three patterns and outputs are feed through the network. Then the weight change needed is calculated. First for the output layer error gradient is calculated using the actual output and the desired output of the network.

$$\delta_k = y_k(1 - y_k)(d_k - y_k)$$
$$\textit{where } y_k \textit{ is the value at output neuron } k$$
$$\textit{and } d_k \textit{ is the desired value at output neuron } k$$

```
int tmp = num_layers - 1;
for(int j = 0; j < layers[tmp].num_nodes; j++){
        float dalta = (layers[tmp].chr[j]).output*
                                        (1.0 - (layers[tmp].chr[j]).output)*
                                        (dout[j] - (layers[tmp].chr[j]).output);

        for(int k = 0; k < (layers[tmp].chr[j]).num_inputs; k++){
                (layers[tmp].chr[j]).errors[k] = dalta + momen * (layers[tmp].chr[j]).errors[k];
                (layers[tmp].chr[j]).weights[k] +=
                                leaning_rate * (layers[tmp].chr[j]).inputs[k] *
                                (layers[tmp].chr[j]).errors[k];
        }
}
```

Then for the other layers the error gradient is calculated using the sum of the weighted errors of the previous layer of it.

$$\delta_j = y_j(1 - y_j)\sum_{k=1}^{n} w_{jk}\ \delta_k$$

```
for(int i = num_layers - 2; i >= 0; i--){
        for(int j = 0; j < layers[i].num_nodes; j++){

                float sum = get_weighted_error(i + 1, j);  // get the weighted error sum
                for(int k = 0; k < (layers[i].chr[j]).num_inputs; k++){

                        float dalta = layers[i].chr[j].output * (1 - layers[i].chr[j].output) * sum;
```

```
                    (layers[i].chr[j]).errors[k] = dalta + momen *
                                            (layers[i].chr[j]).errors[k];
                    (layers[i].chr[j]).weights[k] += leaning_rate *
                              (layers[i].chr[j]).inputs[k] * (layers[i].chr[j]).errors[k];
            }
        }
    }
```

Then the error gradients is used to get the weight change,

$$w_{ij} = w_{ij} + \Delta w_{ij} \; and \; w_{jk} = w_{jk} + \Delta w_{jk}$$

$$\text{where } \Delta w_{ij}(t) = \alpha . inputNeuron_i . \delta_j$$
$$and \; \Delta w_{jk}(t) = \alpha . hiddenNeuron_j . \delta_k$$

α = leaning rate

And after that the weights are adjusted using the that weight change. This will be done in number of iterations. After that the error for a patten is calculated. If the error is larger than the error tolerance the user is asked to do some more back propagation (leaning).

3.0 Reference

[1] Holk Cruse , *Neural Networks as Cybernetic Systems* – 2nd and revised edition , Bielefeld University , page 99.
[2] *Basic Neural Network Tutorial – Theory*, Taking Initiative,http://takinginitiative.net/2008/04/03/basic-neural-network-tutorial-theory/

[3] *Backpropagation*, wikipedia, http://en.wikipedia.org/wiki/Backpropagation

4.0 More Inf0

- tcg.galahena@gmail.com
- http://www.inf0warri0r.blogspot.com