CS4341 ARTIFICIAL INTELLIGENCE. C00
Prof. Carolina Ruiz
Department of Computer Science
Worcester Polytechnic Institute

Problem 2 - HW4
Training Neural Networks: Error Back Propagation

Problem Description

The purpose of this problem is to use the error back propagation procedure. Suppose you want a feed–forward neural network to learn a given collection of examples. What you need to do is to determine the weights of the different connections (synapses) in the net. The error back propagation procedure helps you adjust those weights. The steps of this procedure are enumerated below.

ERROR BACK PROPAGATION PROCEDURE (Adapted from [HKP91] and [Winston92])

I. Initialize the weights to small random values

II. Repeat

- For $e := 1, \ldots, n$ (i.e. for each example: $example_1, \ldots, example_n$) do
  1. Present the example to the input layer
  2. Compute the output of each node by propagating the signal forward through the net. Remember that the activation function of each node is given by the sigmoid function:

$$\sigma(y) = \frac{1}{1 + \exp^{-y}}, \quad \text{where:}$$

  y = the weighted sum of the inputs to that node.
  3. Compute the β's for the nodes in the output layer:

$$\beta_k = d_k - o_k$$

  by comparing the actual output o_k of node k with the desired output d_k for the example being used.
  4. Compute the β's for the nodes in the hidden layers by propagating the errors (β's) backwards until a β has been calculated for every node in the net:

$$\beta_j = \sum_k W_{j \to k} o_k (1 - o_k) \beta_k$$

  where k varies over all nodes in the layer to the right of node j.
  5. Compute the weight changes for all connections in the net:

$$\Delta_e W_{i \to j} = r o_i o_j (1 - o_j) \beta_j$$

  where r is a parameter denoting the learning rate.
- After processing all examples, update the weights:

$$W_{i \to j} := W_{i \to j} + \sum_{e=1}^{n} \Delta_e W_{i \to j}$$

Until you obtain satisfactory weights.

Intuition:

- The error back propagation procedure uses hill climbing over the space of weights.
  - Since this space is continuous (each weight is a real number) the direction to move to at each step of the hill climbing process is given by the gradient of the "terrain" (i.e. the objective function as a function of the weights). This gradient is computed using partial derivatives.
  - β's are the derivatives of the objective function w.r.t. the outputs.
  - $\Delta W_{i \to j}$'s are the derivatives of the objective function w.r.t. the weights.
- As with hill climbing, error back propagation can get lost in difficult terrains: foothills, plateaus, ridges, etc.

Problem Assignment

1. Complete Table 2 by following the error back propagation procedure outlined above. The goal is to train the neural net of Figure 1 using the training examples shown in Table 1. It is entirely up to you how you compute those values: you can do it by hand; or you can write a program that implements the error back propagation algorithm given above (in this case you can submit your program using turnin for extra credit); or you can use a spreadsheet to obtain the results.
   (a) Start with the weight values specified in Table 2 and learning rate $r = 1$.
   (b) Compute the updated weights after processing **all** training examples once.
   (c) Would the net with the updated weights produce the desired results? Or are more iterations of the error back propagation procedure needed?

References

[HKP91] Hertz, J., Krogh, A., and Palmer, R. *Introduction to the Theory of Neural Computation*, Addison–Wesley Publishing Company, 1991

[Winston92] Patrick H. Winston. "Artificial Intelligence" 3rd edition Addison Wesley.

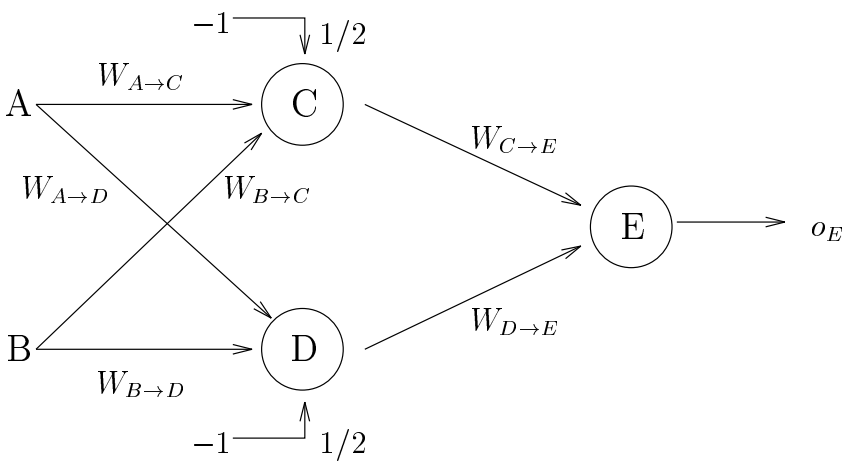

Figure 1: Neural Net with two inputs A and B

A	B	Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: Set of Examples to train the Neural Net

	Random Values	1st example	2nd example	3rd example	4th example	New weights
A		0	0	1	1	
B		0	1	0	1	
d_E		0	1	1	0	
$W_{A\to C}$	0.1					
$W_{A\to D}$	0.2					
$W_{B\to C}$	-0.1					
$W_{B\to D}$	0.05					
$W_{C\to E}$	-0.2					
$W_{D\to E}$	0.3					
o_C						
o_D						
o_E						
$\beta_E = d_E - o_E$						
$\beta_C = \sum_{k=E} W_{C\to k} o_k (1-o_k)\beta_k$						
$\beta_D = \sum_{k=E} W_{D\to k} o_k (1-o_k)\beta_k$						
$\Delta_e W_{A\to C} = r o_A o_C (1-o_C)\beta_C$						
$\Delta_e W_{A\to D} = r o_A o_D (1-o_D)\beta_D$						
$\Delta_e W_{B\to C} = r o_B o_C (1-o_C)\beta_C$						
$\Delta_e W_{B\to D} = r o_B o_D (1-o_D)\beta_D$						
$\Delta_e W_{C\to E} = r o_C o_E (1-o_E)\beta_E$						
$\Delta_e W_{D\to E} = r o_D o_E (1-o_E)\beta_E$						

Table 2: Running Back Propagation