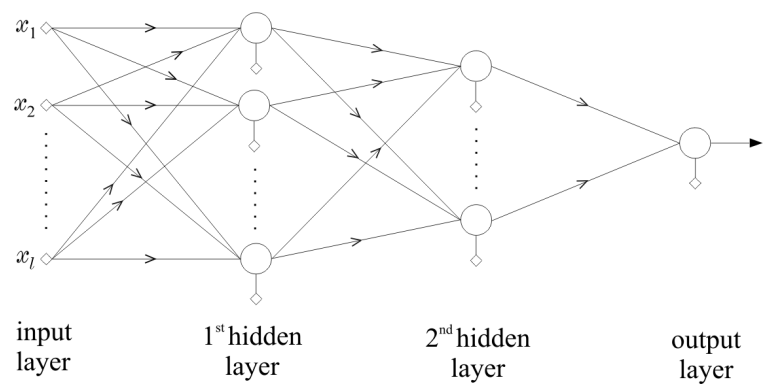


CSE 320 [Pattern Recognition Sessional]
Lab # 3

The Backpropagation Algorithm

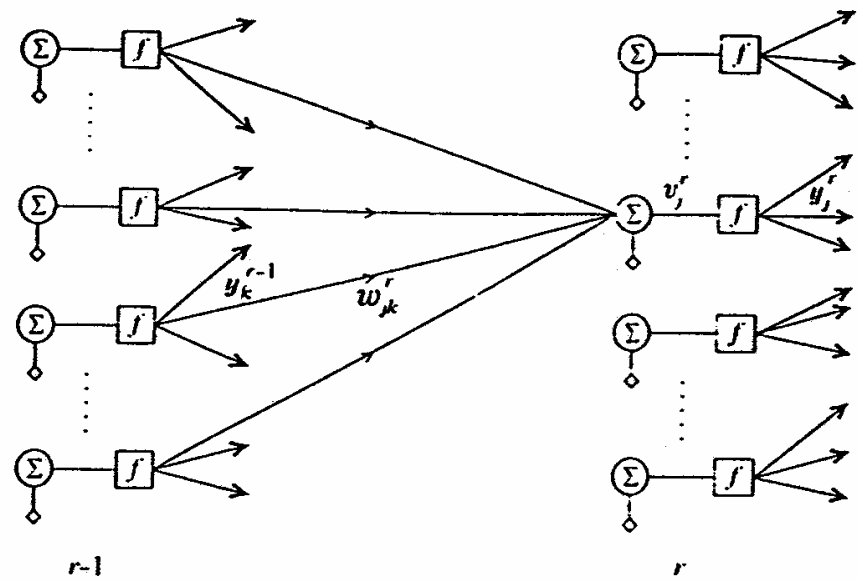
This assignment is on computing the weights of a multilayer perceptron (MLP) by using the famous Backpropagation Algorithm. You have already gathered the basic idea of the algorithm from my CSE 319 course (at least I hope so). Now here’s a brief recap:

The Architecture of an MLP (Three layer-perceptron)



The Backpropagation Algorithm (also known as the Error Backpropagation Algorithm or EBP) is an algorithmic procedure that computes the synaptic weights iteratively, so that an adopted cost function is minimized (optimized).

Notations of variables:

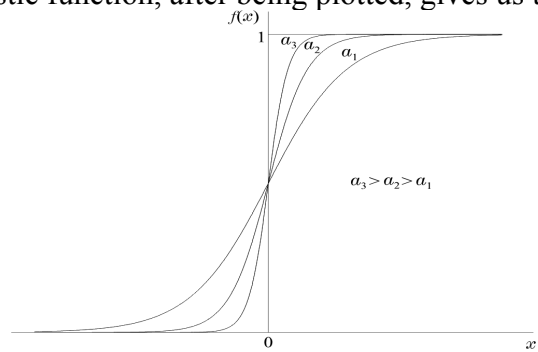


Activation Function:

Here computation of derivative of the activation function is required. This is the reason why we cannot use the step function as the activation function of the neurons. Instead we’ll use the logistic function:

$$f(x) = \frac{1}{1 + \exp(-ax)}$$

The logistic function, after being plotted, gives us the following curve:



The basic iteration step of the backpropagation algorithm is:

$$\underline{w}_j^r(\text{new}) = \underline{w}_j^r(\text{old}) + \Delta \underline{w}_j^r$$

$$\Delta \underline{w}_j^r = -\mu \frac{\partial J}{\partial \underline{w}_j^r}$$

So the goal is to compute $\Delta \underline{w}_j^r$ at each step of iteration. After some lengthy derivation we arrive at the following equation:

$$\Delta \underline{w}_j^r = -\mu \sum_{i=1}^N \delta_j^r(i) \underline{y}^{r-1}(i) \text{-----(1)}$$

where i is the index for training vectors and there are a total of N training vectors and

$$\delta_j^r(i) = \frac{\partial E(i)}{\partial v_j^r}$$

Here the error function

$$E(i) = 0.5 \sum_{m=1}^{k_L} e_m^2(i) = 0.5 \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2$$

$i = 1, 2, \dots, N$

And $y_m(i)$ is the desired output from the m-th neuron of the output layer for the i-th training vector, whereas $\hat{y}_m(i)$ is the output given by the MLP.

Here we know that

$$\underline{y}^{r-1}(i) = \begin{bmatrix} +1 \\ y_1^{r-1}(i) \\ \vdots \\ y_{k_{r-1}}^{r-1}(i) \end{bmatrix}$$

So the only term that we'll have to determine in equation (1) is δ_j^r . There are two cases:

Case I. $r = L$

For this case:

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i)) \text{-----(2)}$$

Case II. $r < L$

For this case: we compute the δ values backward. So we have already computed $\delta_j^L(i)$ by case I above. Now we can compute $\delta_j^{r-1}(i)$ from $\delta_j^r(i)$ by the following equation:

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \right] \cdot f'(v_j^{r-1}(i)) \text{-----(3)}$$

Now that the theory part is done, it's time to see the algorithm.

The Backpropagation Algorithm:

1. Initialization: Initialize all the weights with small random real values, preferably from -1 to +1.

2. Forward Computation: For each of the training vectors $\underline{x}(i)$, $i = 1, 2, \dots, N$, compute all the $v_j^r(i)$, $y_j^r(i) = f(v_j^r(i))$, $j = 1, 2, \dots, k_r$ and $r = 1, 2, \dots, L$, by using the equation:

$$v_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1}(i) + w_{j0}^r$$

Compute the cost function J by using the following equations:

$$J = \sum_{i=1}^N E(i) \quad E(i) = 0.5 \sum_{m=1}^{k_L} e_m^2(i) = 0.5 \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2$$

If the value of J is smaller than some predetermined value (like 0.0001) then stop the training is done. Now is the time to evaluate performance- go to **performance evaluation** (Step 5).

3. Backward Computation:

For each $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, k_L$ compute $\delta_j^L(i)$ by the equation (2), rewritten here:

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i))$$

Now go backward and compute $\delta_j^{r-1}(i)$ from $\delta_j^r(i)$, $r = L, L-1, \dots, 2$ and $j = 1, 2, \dots, k_L$, by using equation (3), restated here:

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \right] \cdot f'(v_j^{r-1}(i))$$

4. Weight Update:

Using equation (1), For $r = 1, 3, \dots, L$ and $j = 1, 2, \dots, k_L$ we can update the weights as follows:

$$\Delta w_j^r = -\mu \sum_{i=1}^N \delta_j^r(i) y_i^{r-1} \quad [\text{This is Equation (1)}]$$

$$w_j^r(\text{new}) = w_j^r(\text{old}) + \Delta w_j^r$$

$$\Delta w_j^r = -\mu \frac{\partial J}{\partial w_j^r}$$

After weights are being updated **go to step 2.**

5. Performance Evaluation:

The weights are computed by using the training vectors given to us. Now to evaluate performance of the MLP we'll use another set called the validation set.

Validation set:

This is similar to the training set. The job of the training set is to help compute the weights. But the job of the validation set is to test the performance of the MLP. Suppose there are 100 training vectors in the validation set (i.e. the desired output of each of these vectors is known). Then we apply these 100 vectors of the validation set to the MLP and we see that for 90 of these vectors we get the correct output, but for the rest 10 we get wrong output. So the accuracy of the MLP is 90%. **If the accuracy is below some limit, say 90%, then retrain the whole network again.**

Test Set:

There's another set called the **test set**. We use this set only after validation step is done. This is also like the training set. For the feature vectors in this set, the desired output is known. By using the test set, we can measure the accuracy of the MLP. But we do not go for retraining after using the test set. Test set is solely for measuring performance.

Implementation issues:

1. What's the value of μ ?

μ should be a small real number. If it is big then the algorithm may not converge. But if it is too small then the algorithm takes a lot of time to converge. You can take 0.5 or so for μ .

2. How many layers and how many neurons in each layer should I take?

That's left to your choice. But there should be at least 3 layers.

of neurons in the output layer = # of outputs.

of neurons in the hidden layers = > Your choice.

3. Description of the input file:

The format of the input file is:

Input_dimension (i.e. # of features in each feature vector)

Output_dimension (i.e. # of outputs)

N(= # of training vectors)

Vn(= # of validation vectors)

Tn(= # of test vectors)

N lines, in each line there is a training vector (input and output)

Vn lines, in each line there is a validation vector (input and output)

Tn lines, in each line there is a test vector (input and output)

Example:

```
8
2
8
2
2
0.352941 0.83 0.606557 0 0 0.396423 0.0964987 0.75 0 1
0.352941 0.77 0.606557 0.32 0.228132 0.436662 0.324936 0.3 0 1
0.117647 0.46 0.622951 0.2 0 0.360656 0.691716 0.116667 0 1
0.117647 0.555 0.491803 0 0 0.390462 0.113151 0.0333333 0 1
0.470588 0.325 0.590164 0.23 0 0.4769 0.222886 0.35 0 1
0.0588235 0.53 0.57377 0.28 0.159574 0.509687 0.0273271 0.0166667 0 1
0 0.49 0.672131 0.15 0.0992908 0.375559 0.0943638 0.0166667 0 1
0.235294 0.475 0.57377 0.32 0 0.47839 0.22801 0.05 0 1
0.411765 0.405 0.639344 0.4 0.0567376 0.695976 0.0781383 0.35 0 1
0.0588235 0.485 0.557377 0.21 0 0.405365 0.434244 0.0166667 0 1
0.411765 0.75 0.639344 0.29 0.148936 0.52459 0.262169 0.55 1 0
0.764706 0.725 0.672131 0.19 0.130024 0.330849 0.0713066 0.6 0 1
```

A number of input files are given in a shared directory:

4. Description of Output:

- Training time of the network: stdout
- Accuracy of the network: stdout
- The weights: to a disk file

5. Is copying allowed?

NO. Copying is strictly prohibited. An incomplete program will get more marks than a copied program.

Prepared by
 Mohammad Tanvir Irfan
 Lecturer, CSE Dept.
 BUET