CSCE 970: Pattern Recognition	Spring Semester, 1999

Topic 5: The Backpropagation Algorithm and its Variations

Lecturer: Stephen Scott *Scribe: D. Khataniar*

1 Introduction

Backpropagation is the most widely used training algorithm for neural networks. The algorithm learns or adapts the synaptic weights based on a set of training data. It uses the technique of gradient descent in order to minimize a preselected cost function. The error values are propagated back from the output layer to the input layers, scaling with the weights. Learning progresses by alternately computing the "error term" for the output layer with the current estimate of weights and propagating backward the instantaneous errors and updating the weights. Variations on the backpropagation theme are used in order to overcome the disadvantage of the gradient descent approach: the slow convergence to the cost function minimum.

2 The Backpropagation Algorithm

For problems that are not linearly separable, multi-layer perceptrons are used in order to remap the feature vector space to one that is linearly separable. The backpropagation algorithm is used in order to do this. For this algorithm, the architecture is assumed to be fixed and complete, and the synaptic weights are computed so as to minimize a cost function of the output.

The multilayer perceptron architectures use continuous differentiable activation functions, e.g.

$$f(x) = \frac{1}{1+\exp(-ax)} \in [0,1] \tag{1}$$

$$f(x) = c\tanh(ax) \in [-c,c] \tag{2}$$

Let us assume that that the network consists of L layers of neurons, and that there are N pairs of input and desired output values available for training. Also let (Figure 1)

k_r = number of nodes in layer r, for r = 0, 1, 2, ... , L
w_{ji}^r = weight from neuron i in layer $r-1$ to neuron j in layer r
The argument of the activation function is

$$v_j^r = \sum_{k=1}^{k_{r-1}} w_{jk}^r y_k^{r-1} + w_{j0}^r \tag{3}$$

The output of neuron j in layer r is given by

$$y_j^r = f(v_j^r) \tag{4}$$

The cost function to be minimized is given by

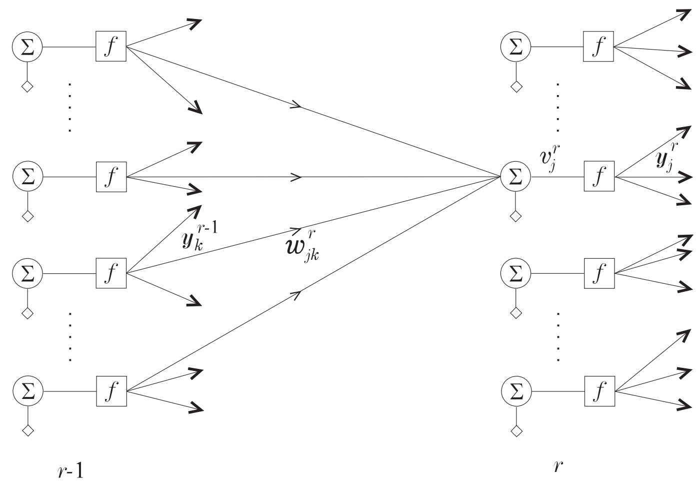

Figure 1: Definition of variables involved in the backpropagation algorithm

$$U(\mathrm{w}_j^L) = \sum_{k=1}^{k_{L-1}} \overbrace{\left(w_{jk}^L(t+1) - w_{jk}^L(t)\right)^2}^{squaredEuclideandistance(conservativeness)} + \eta \overbrace{\left(\overbrace{y_j(t)}^{correct} - \overbrace{f\left(\sum_{k=1}^{k_{L-1}} w_{kj}^L(t+1) y_k^{L-1}(t)\right)}^{pred=y_j^L(t) with \mathrm{w}(t+1)}\right)^2}^{squarederror(correctness)} \tag{5}$$

where η is the learning rate.

The Output Layer

Taking gradient of equation (5) with respect to $\mathrm{w}_j^L(t+1)$ and equating to $\mathbf{0}$, we have

$$0 = 2\left(w_{jk}^L(t+1) - w_{jk}^L(t)\right) - 2\eta\left(y_j(t) - f\left(\sum_{k=1}^{k_{L-1}} w_{kj}^L(t+1) y_k^{L-1}(t)\right)\right)$$
$$.f'\left(\sum_{k=1}^{k_{L-1}} w_{kj}^L(t+1) y_k^{L-1}(t)\right) y_k^{L-1}(t)$$

Approximating and solving for $w_{jk}^L(t+1)$, we have

$$w_{jk}^L(t+1) = w_{jk}^L(t) + \eta y_k^{L-1}(t).\left(y_j(t) - f\left(\sum_{k=1}^{k_{L-1}} w_{kj}^L(t) y_k^{L-1}(t)\right)\right)$$
$$.f'\left(\sum_{k=1}^{k_{L-1}} w_{kj}^L(t) y_k^{L-1}(t)\right) y_k^{L-1}(t)$$
$$= w_{jk}^L(t) + \eta y_k^{L-1}(t) \overbrace{\left(y_j(t) - f\left(v_j^L(t)\right)\right) f'\left(v_j^L(t)\right)}^{\delta_j^L(t)=\text{"}errorterm\text{"}} \tag{6}$$

For $f(x) = 1/(1+\exp(-ax))$,

$$\delta_j^L(t) = a.y_j^L(t).(y_j(t) - y_j^L(t))(1 - y_j^L(t)) \tag{7}$$

where $y_j(t)$ is the target and $y_j^L(t)$ is the output.

The Other Layers

The error values can now be propagated back from the output layer to the other layers of the network, scaling with the weights. Computing the gradient of the cost function with respect to $\mathrm{w}_j^r(t+1)$ via the Chain rule, we can show that

$$w_{jk}^r(t+1) = w_{jk}^r(t) + \eta y_k^{r-1}(t)\delta_j^r(t) \tag{8}$$

where

$$\delta_j^r(t) = f'(v_j^r(t)) \sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(t) w_{kj}^{r+1}(t) \tag{9}$$

2.1 A Simple Example

Let us consider a two-layer perceptron with one hidden node and one output node, i.e. $L = 2$. Also let $l = 2$.
$\mathbf{x}(1) = [1\ 0]^T$, target $\mathrm{y}(1) = 1$ (Trial 1)
$\mathbf{x}(2) = [0\ 1]^T$, target $\mathrm{y}(2) = 0$ (Trial 2)
$f(x) = 1/(1+\exp(-x))$.

- *Initialization*: $\eta = 0.3$
  $w_{11}^1(1) = 0.1; w_{12}^1(1) = 0.1; w_{10}^1(1) = 0.1; w_{11}^2(1) = 0.1; w_{10}^2(1) = 0.1$
  w_{10}^1 and w_{10}^2 are the thresholds. Therefore, for the computations, a 1 has to be appended to the inputs, i.e. $\mathbf{x}(1) = [1\ 0\ 1]^T$, and $\mathbf{x}(2) = [0\ 1\ 1]^T$

- *Forward computations (Trial 1)*: t=1. Using equations (3) and (4), compute v_j^r and y_j^r.
  $v_1^1(1) = 0.2000000$ $\quad$ $y_1^1(1) = 0.5498340$
  $v_1^2(1) = 0.1549834$ $\quad$ $y_1^2(1) = 0.5386685$
  Using equation (7), compute the error term for the output layer.
  $\delta_1^2(1) = 0.1146431$

- *Backward computations (Trial 1)*: t=1. Using equation (9), compute the error term for the hidden layer.
  $\delta_1^1(1) = 0.0028376$

- *Weight updates (Trial 1)*: Using equation (8), update the weights.
  $w_{11}^1(2) = 0.1008513$ $\quad$ $w_{12}^1(2) = 0.1000000$ $\quad$ $w_{10}^1(2) = 0.1008513$
  $w_{11}^2(2) = 0.1189104$ $\quad$ $w_{10}^2(2) = 0.1343929$
  These weights are now used for the second trial.

- *Forward computations (Trial 2)*: t=2. Using equations (3) and (4), compute v_j^r and y_j^r.
  $v_1^1(2) = 0.2008513$ $\quad$ $y_1^1(2) = 0.5500447$
  $v_1^2(2) = 0.1997990$ $\quad$ $y_1^2(2) = 0.5497842$
  Using equation (7), compute the error term for the output layer.
  $\delta_1^2(2) = -0.136083$

- *Backward computations (Trial 2)*: t=2. Using equation (9), compute the error term for the hidden layer.
  $\delta_1^1(2) = -0.004005$

- *Weight updates (Trial 2)*: Using equation (8), update the weights.
  $w_{11}^1(3) = 0.1008513$ $w_{12}^1(3) = 0.0987985$ $w_{10}^1(3) = 0.0996498$
  $w_{11}^2(3) = 0.0964548$ $w_{10}^2(3) = 0.0935679$

2.2 Issues

- The iterations through the training data are stopped when either the cost function becomes smaller than a certain threshold or the weights do not change much. Also, overtraining must be avoided. Figure 2 shows two output error curves as a function of iteration steps. One curve corresponds to the traininng set and the other corresponds to a test set of data. It can be seen that for the test set, the error decreases initially, but starts increasing later. This is because the weights, computed from the training set, adapt to the peculiarities of the specific training set, thus affecting the generalized performance of the network. The point where the two curves start departing, can be used as a termination point for the learning process iterations.

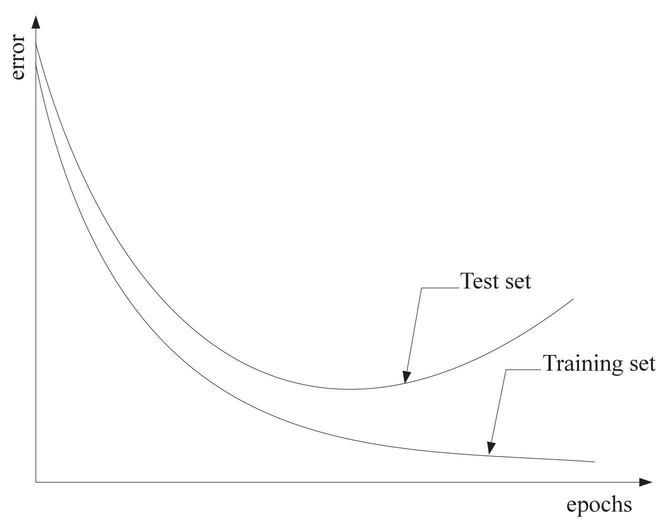

Figure 2: Output error versus number of iterations showing overtraining of the training set

- The convergence speed of the backpropagation algorithm depends on the learning rate η. The learning rate should be small enough to avoid overshooting the minimum, but not so small that the convergence speed becomes very slow. η can also be adapted over time.

- The backpropagation algorithm runs the risk of getting trapped in a local minimum. If the local minimum is not good enough, the algorithm should be run again with a new set of initial weights.

- The algorithm can update the weights for each of the training pairs as we have presented. This is known as the on-line mode. This mode might avoid local minima. The other mode is the batch mode, which updates the weights once *all* the training pairs have appeared in the network. This mode averages gradients giving better estimates and smoother convergence. The weight update then becomes

$$\mathrm{w}_j^r(\text{new}) = \mathrm{w}_j^r(\text{old}) + \eta \sum_{t=1}^{N} \delta_j^r(t)\mathrm{y}^{r-1}(t)$$

3 Variations on the Backpropagation Theme

Both the on-line and batch versions of the backpropagation scheme have the inherent disadvantage of gradient descent algorithms: slow convergence to the cost function minimum. Therefore, variations on the backpropagation theme are used:

- A momentum term $\alpha < 1$ is used with the update equations in order to smooth the oscillations of the weight vector.

$$\Delta\mathrm{w}_j^r(t+1) = \alpha\Delta\mathrm{w}_j^r(t) + \eta y_k^{r-1}(t)\delta_j^r(t)$$

$$\mathrm{w}_j^r(t+1) = \mathrm{w}_j^r(t) + \Delta\mathrm{w}_j^r(t+1)$$

  The momentum term tends to keep the weight vector moving in the same direction as previous trials.

- A recurrent network, where the output of a layer, say r is fedback to some earlier layer $r' < r$, can also be used. This allows predictions to be influenced by past predictions. Such networks are used in voice recognition, natuaral language understanding, stock market prediction, etc.

- Backpropagation can also be used with the exponentiated gradient algorithm.

- Some other nonlinear optimizations schemes such as conjugate gradient, Newton's method, genetic algorithms and simulated annealing, may also be used.

- Depending on the specific problem, other cost functions may be used to give better results. An example is the *cross-entropy* cost function

$$U = -\sum_{t=1}^{N}\sum_{k=1}^{k_L}\left(\overbrace{y_k(t)}^{label}\ln(\overbrace{y_k^L(t)}^{predicted} + (1-y_k(t))\ln(1-y_k^L(t))\right)$$

  This funcion becomes infinitely large if $y_k(t) \approx 1$ and $y_k^L(t) \approx 0$ or vice versa. Thus, if one of the outputs converges to the wrong extreme, the cost function diverges causing the gradient descent algorithm to react fast.