CISC 452 Assignment #2

Alex Globus

October 17, 2016

1 Back-propagation Algorithm

Let the input layer consist of input vector $\mathbf{x} = \{\mathbf{x_1}, \ldots, \mathbf{x_K}\}$. The hidden layer has N neurons $\mathbf{h} = \{\mathbf{h_1}, \ldots, \mathbf{h_N}\}$ and the output layer is $\mathbf{y} = \{\mathbf{y_1}, \ldots, \mathbf{y_M}\}$. Every element in the input layer is connected to every neuron in the hidden layer with w_{ki} which represents the weight on the connection between the kth input element and the ith hidden neuron. Every output from the ith hidden layer to the jth output neuron is represented by w'_{ij}. Let w_{ki} be the (k, i) entry in $K \times N$ matrix $\mathbf{W}$ and w'_{ij} be the (i, j) entry in $N \times M$ matrix $\mathbf{W}'$.

The output of neuron h_i in the hidden layer is given as follows:

$$h_i = f(u_i) = f(\sum_{k=1}^{K} w_{ki} x_k)$$

Similarly for the output of neuron y_j in the output layer:

$$y_j = f(u'_j) = f(\sum_{i=1}^{N} w'_{ij} h_i)$$

Our error equation we wish to minimize:

$$E = \frac{1}{2} \sum_{j=1}^{M} (y_j - t_j)^2$$

1.1 Updating hidden to output nodes

We need to update both sets of weights (hidden-to-input w_{ki} and hidden-to-output w'_{ij}). To do this we compute the vector of partial derivatives of E with respect to both weights. First, we start with w'_{ij}.

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial w'_{ij}}$$

$$\frac{\partial E}{\partial y_j} = y_j - t_j$$

$$\frac{\partial y_j}{\partial u'_j} = y_j(1 - y_j) = f(u'_j)(1 - f(u'_j))$$

$$\frac{\partial u'_j}{\partial w'_{ij}} = \frac{\partial \sum_{i=1}^{N} w'_{ij} h_i}{\partial w'_{ij}} = h_i$$

$$\frac{\partial E}{\partial w'_{ij}} = (y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$$

$$w'_{ij,\text{new}} = w'_{ij,\text{old}} - \eta \cdot (y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$$

1.2 Updating input to hidden nodes

Now let's compute E with respect to the input-to-hidden weights w_{ki}. Remember that each hidden neuron that w_{ki} is connected to is itself connected to every output node. This is taken into account with the summation of the hidden-to-output gradients.

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M} \left(\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial h_i} \right) \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}}$$

$$\frac{\partial u'_j}{\partial h_i} = \frac{\partial \sum_{i=1}^{N} w'_{ij} h_i}{\partial h_i} = w'_{ij}$$

$$\frac{\partial h_i}{\partial u_i} = h_i(1 - h_i)$$

$$\frac{\partial u_i}{\partial w_{ki}} = \frac{\partial \sum_{k=1}^{K} w_{ki} x_k}{\partial w_{ki}} = x_k$$

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M} \left[(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij} \right] \cdot h_i(1 - h_i) \cdot x_k$$

$$w_{ki,\text{new}} = w_{ki,\text{old}} - \eta \cdot \sum_{j=1}^{M} \left[(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij} \right] \cdot h_i(1 - h_i) \cdot x_k$$

2 Implementation

We propagate the network forward with the following snippet of code. The code is responsible for generating the summation of a node's inputs and weights. The summation value is stored with the node for later access and also fed into the sigmoid activation function. The result of that is pushed onto an intermediary vector object.

$$y = f(u) = f(\sum_{x=1}^{X} w_x \cdot h_x)$$

```
for (size_t j = 0; j < l->num_neurons; j++) {
  Neuron_t *n = l->neurons[j];

  double net_input = 0;

  if (i > 0) {
    for (size_t k = 0; k < input_values.size(); k++) {
      net_input += n->weights[k] * input_values[k];
    }
  } else {
    net_input = working_set[j];
  }

  n->input_sum = net_input;
  output.push_back(sigmoid(n->input_sum));
}
```

Once the network has iterated through the data, vector output holds the activated values at the output layer. Next we update the sets of weights from the hidden to the output layer according to the following rule.

$$w'_{ij,\text{new}} = w'_{ij,\text{old}} - \eta \cdot (y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$$

```
for (size_t i = 0; i < output_layer->num_neurons; i++) {
  Neuron *n = output_layer->neurons[i];

  double node_delta = -(desired[i] - output[i]) *
  sigmoid_derivative(n->input_sum);

  errors.push_back(node_delta);

  for (size_t j = 0; j < n->num_inputs; j++) {
    if (n->previous_weights[j] == 0)
      n->previous_weights[j] = n->weights[j];

      n->delta_weights[j] = n->weights[j] - n->previous_weights[j];
      n->previous_weights[j] = n->weights[j];

      double weight_change = sigmoid(hidden_layer->neurons[j]->input_sum) *
      node_delta + MOMENTUM_ * n->delta_weights[j];

      n->weights[j] -= ETA_ * weight_change;
  }
}
```

We first compute $(y_j - t_j) \cdot y_j(1 - y_j)$ since it is dependent on the node we are processing. Note that this value appears in our rule for updating input to hidden layers weights, so we save it in a vector for later use. Next, for each input to the hidden node, we calculate the weight change and set it in the node type. This is accomplished by multiplying our previously calculated value (the change in error with respect to the weights) with activated output of the hidden node that is connected to the output node input; $(y_j - t_j) \cdot y_j(1 - y_j) \cdot h_i$. We can add momentum to the equation by adding the product of the change in weights to some momentum constant. Finally, we update the weights and multiply it by our learning rate η.

Next, we update the weights from the input to the hidden layer using the following rule. We need to sum the product of our previous calculation with each weight from the weight between each hidden node and the output nodes. That result is multiplied with the sigmoid derivation of the hidden node multiplied with the activation function of the input neuron. The code is very similar to updating the hidden to output layer.

$$w_{ki,\text{new}} = w_{ki,\text{old}} - \eta \cdot \sum_{j=1}^{M} \left[(y_j - t_j) \cdot y_j(1 - y_j) \cdot w'_{ij} \right] \cdot h_i(1 - h_i) \cdot x_k$$

```
for (size_t i = 0; i < output_layer->num_neurons; i++) {
  sum_output += output_layer->neurons[i]->weights[j] * previous_error[i];
}
...
double weight_change = sigmoid(input_layer->neurons[k]->input_sum) *
sigmoid_derivative(hn->input_sum) * sum_output + MOMENTUM_ *
hn->delta_weights[k];

hn->weights[k] -= ETA_ * weight_change;
```

3 Results

The program will output the number of correctly classified samples and the average total error given the network parameters defined in the types header file. Below is a summary of various inputs given to the network and the corresponding outputs.

Input Nodes	Output Nodes	Momentum	Learning rate
1024	10	0.07	0.5

Training Samples	Testing Samples
1934	943

Table 1: Training Set

Hidden Nodes	Epochs	Correct	Total	Accuracy %	Avg Total Error
8	10	9309	19340	48.13%	0.3178
8	25	8347	48350	17.26%	0.4127
16	10	15726	19340	81.31%	0.1417
16	25	42281	48350	87.44%	0.1
16	100	177932	193400	92%	0.064
32	10	16498	19340	85.30%	0.119
32	25	43577	48350	90.13%	0.0832
32	100	180901	193400	93.53%	0.0552

Table 2: Testing set

Hidden Nodes	Epochs	Correct	Total	Accuracy %	Avg Total Error
8	10	504	943	53.44%	0.3471
8	25	175	943	18.56%	0.4034
16	10	768	943	81.44%	0.1286
16	25	836	943	88.65%	0.087
16	100	880	943	93.32%	0.057
32	10	855	943	90.67%	0.0844
32	25	836	943	88.65%	0.0941
32	100	884	943	93.74%	0.0519