Figuring out Back-propagation

July 1, 2016

1 Conventions

- Weight from node j to node i in the l layer : $W_{ij}^{(l)}$
- Bias to node i in the l layer : $b_i^{(l)}$
- j-th input of layer l : $x_j^{(l)\ r,c}$
- Activation of node i in the layer l : $a_i^{(l)\ r,c}$
- Total weighted sum including the bias : $z_i^{(l)\ r,c}$
- Hypothesis or output i of the network : $h_i^{r,c}(x_i^{r,c}) = a_i^{(out)\ r,c}$
- Target i of the network : y_i
- "Error term" of node i in layer l : $\delta_i^{(l)\ r,c}$
- Activation function f : $a_i^{(l)\ r,c} = f(z_i^{(l)\ r,c}) = \tanh(z_i^{(l)\ r,c})$
- Derivative of the activation function : $f'(z_i^{(l)\ r,c}) = \text{sech}(z_i^{(l)\ r,c})^2$
- Activation function of the output layer $f^{(out)}$:
  $h_i^{r,c}(x_i^{r,c}) = f^{(out)}(z_i^{(out)\ r,c}) = z_i^{(l)\ r,c}$
- Derivative of the activation function : $f'(z_i^{(out)\ r,c}) = 1$
- Cost function, here we use the Sum Square Bias :
  $J(W,b) = \frac{1}{2}\sum_i \sum_c \left(\frac{\sum_r h_i^{r,c}}{\#r} - y_i\right)^2$
- r denotes the realization and c the case

2 Adapting back-propagation to our case

The goal is to change our weights so that we lower the value of the cost function. This can be done by gradient descent :

$$\Delta W_{ij}^{(l)} = -\eta \frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} \ , \tag{1}$$

where $\Delta W_{ij}^{(l)}$ is the amount by which we want to change the weight $W_{ij}^{(l)}$, and where η is an arbitrarily chosen parameter (ideally this won't be the case) and is generally $\sim 10^{-3}$. Back-propagation is a way of calculating $\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}}$.
Using the chain rule we have :

$$\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} = \sum_{r,c} \frac{\partial J(W,b)}{\partial z_i^{(l)\ r,c}} \frac{\partial z_i^{(l)\ r,c}}{\partial W_{ij}^{(l)}} = \sum_{r,c} \frac{\partial J(W,b)}{\partial z_i^{(l)\ r,c}} x_j^{(l)\ r,c} \ . \tag{2}$$

Let's define the "error terms" by : $\delta_i^{(l)\ r,c} = \frac{\partial J(W,b)}{\partial z_i^{(l)\ r,c}}$.
Thus we can rewrite eq.2 as follows :

$$\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}} = \sum_{r,c} \delta_i^{(l)\ r,c} x_j^{(l)\ r,c} \ . \tag{3}$$

Let's now find an easy way to calculate the "error terms", starting by the output layer :

$$\delta_i^{(out)\ r,c} = \frac{\partial J(W,b)}{\partial z_i^{(out)\ r,c}} = \frac{\partial J(W,b)}{\partial a_i^{(out)\ r,c}} \frac{\partial a_i^{(out)\ r,c}}{\partial z_i^{(out)\ r,c}} = \frac{\partial J(W,b)}{\partial a_i^{(out)\ r,c}} f'^{(out)}(z_i^{(out)\ r,c}) = \frac{\partial J(W,b)}{\partial h_i^{r,c}} \ , \tag{4}$$

because the activation function for the output layer is the identity function (derivative of 1) and $a_i^{(out)\ r,c} = h_i^{r,c}$ by definition.
Using the definition of the cost function we obtain :

$$\frac{\partial J(W,b)}{\partial h_i^{r,c}} = \frac{1}{\#r} \left(\frac{\sum_r h_i^{r,c}}{\#r} - y_i \right) \ . \tag{5}$$

Thus we can rewrite eq.4 as follows :

$$\delta_i^{(out)\ r,c} = \frac{1}{\#r} \left(\frac{\sum_r h_i^{r,c}}{\#r} - y_i \right) \ . \tag{6}$$

And now for the deltas of the other layers we have :

$$\begin{aligned} \delta_i^{(l)\ r,c} = \frac{\partial J(W,b)}{\partial z_i^{(l)\ r,c}} = \sum_k \frac{\partial J(W,b)}{\partial z_k^{(l+1)\ r,c}} \frac{\partial z_k^{(l+1)\ r,c}}{\partial z_i^{(l)\ r,c}} &= \frac{\partial J(W,b)}{\partial z_k^{(l+1)\ r,c}} \frac{\partial z_k^{(l+1)\ r,c}}{\partial a_i^{(l)\ r,c}} \frac{\partial a_i^{(l)\ r,c}}{\partial z_i^{(l)\ r,c}} \\ &= f'(z_i^{(l)\ r,c}) \sum_k \delta_k^{(l+1)\ r,c} W_{ki}^{(l+1)} \ , \end{aligned} \tag{7}$$

where k denote the indices of the nodes in layer $(l+1)$.

3 Implementing the BFGS algorithm

Th idea of the BFGS algorithm is to add the matrix $\boldsymbol{B_k}$ which is an approximation of the Hessian matrix of the cost function. The BFGS algorithm includes a smart way to calculate this matrix with no need for intermediate matrices to be stored. Usually if we don't have a better hypothesis $\boldsymbol{B_0}$ is defined as the identity matrix $\boldsymbol{I}$. We can define $\mathbf{W_k}$ as the vector containing all the weights at iteration k. So $\mathbf{W_0}$ is our initial guess for the weights.
First we want to find the direction of the update of the weights $\mathbf{p_k}$ by solving the following equation :

$$\boldsymbol{B_k}\mathbf{p_k} = -\nabla\mathbf{J}(\mathbf{W_k}) \ . \tag{8}$$

Then we perform a dichotomy in order to find an appropriate learning rate η_k such that the following update is sensible :

$$\mathbf{W_{k+1}} = \mathbf{W_k} + \eta_k\mathbf{p_k} \ . \tag{9}$$

To simplify the notation we can define : $\mathbf{s_k} = \eta_k\mathbf{p_k}$. We also define the vector $\mathbf{y_k}$ by : $\mathbf{y_k} = \nabla\mathbf{J}(\mathbf{W_{k+1}}) - \nabla\mathbf{J}(\mathbf{W_k})$. Finally update the matrix $\boldsymbol{B_k}$ as follows :

$$\boldsymbol{B_{k+1}} = \boldsymbol{B_k} + \frac{\mathbf{y_k}\mathbf{y_k^T}}{\mathbf{y_k^T}\mathbf{s_k}} - \frac{\boldsymbol{B_k}\mathbf{s_k}\mathbf{s_k^T}\boldsymbol{B_k}}{\mathbf{s_k^T}\boldsymbol{B_k}\mathbf{s_k}} \ . \tag{10}$$

Note that the gradients $\nabla\mathbf{J}(\mathbf{W_k})$ are obtained using the back-propagation algorithm specified above. And also note that for the implementation of this algorithm we only need to store the inverse of the matrix $\boldsymbol{B_k}$: $\boldsymbol{B_k^{-1}}$. So that trivially $\boldsymbol{B_0^{-1}}$ and applying the Sherman-Morrison formula to the equation regarding the update of $\boldsymbol{B_k}$ we find this update relation for its inverse :

$$\boldsymbol{B_{k+1}^{-1}} = \boldsymbol{B_k^{-1}} + \frac{(\mathbf{s_k^T}\mathbf{y_k} + \mathbf{y_k^T}\boldsymbol{B_k^{-1}}\mathbf{y_k})(\mathbf{s_k}\mathbf{s_k^T})}{(\mathbf{s_k^T}\mathbf{y_k})^2} - \frac{\boldsymbol{B_k^{-1}}\mathbf{y_k}\mathbf{s_k^T} - \mathbf{s_k}\mathbf{y_k^T}\boldsymbol{B_k^{-1}}}{\mathbf{s_k^T}\mathbf{y_k}} \ , \tag{11}$$

which can be computed quite easily.