Back-propagation Tutorial

Mingwen Dong

December 28, 2016

1 Backpropagation

Notations:

1. z_i^l: weighted linear (sum) input to i^{th} neuron in layer (l).
2. a_i^l: activation/output from i^{th} neuron in layer (l).

$$a_i^l = g(z_i^l)$$

   $g(\cdot)$ is the activation function. e.g., sigmoid function $g(x) = \frac{1}{1+e^{-x}}$, $tanh(\cdot)$, or ReLU.
   Note: activation function could be other form like Huber's function.

3. w_{ji}^l: connection strength/weight from i^{th} neuron in layer $(l-1)$ to j^{th} neuron in layer (l).

$$z_j^{l+1} = \sum_{i=0}^{n} w_{ji}^{l+1} a_i^l$$

$$\frac{\partial z_j^{l+1}}{\partial z_i^l} = \frac{\partial z_j^{l+1}}{\partial a_i^l}\frac{\partial a_i^l}{\partial z_i^l} = w_{ji}^{l+1} g'(z_i^l)$$

   Notice: the bias is included by adding a constant input "1" to every neuron in the network.

4. L: the output layer.
5. C: the cost. e.g., quadratic cost function, cross-entropy, or likelihood cost function.

$$\frac{1}{2}\sum_{i=0}^{n}(a_i^L - y_i)^2$$

   $i = 0, 1, 2, ..., n$, indicate different output units/neurons.
   In stochastic gradient descent, the cost is also summed over different input x.

6. δ_i^l: derivative of the cost C with respect to each neuron's linear input z_i^l

$$\delta_i^l = \frac{\partial C}{\partial z_i^L}$$

1.1 Feed_Forward

The weighted input to neurons in layer (l) is:

$$z_j^l = \sum_{i=0}^{n} a_i^{l-1} w_{ji}^l \quad \Rightarrow \quad \underset{\sim}{z}^l = \boldsymbol{W}^l \cdot \underset{\sim}{a}^{l-1}$$

where, $\boldsymbol{W}^l$ is the weight matrix for neurons in layer (l). "tilde" indicate a vector variable.
The corresponding activations from these neurons are:

$$\underset{\sim}{a}^l = g(\underset{\sim}{z}^l)$$

The derivative of a_i^l with respect to z_i^l can be calculated during this forward pass process:

$$\frac{\partial a_i^l}{\partial z_i^l} = g'(z_i^l) = \begin{cases} g(z_i^l)[1 - g(z_i^l)] & \text{sigmoid activation function} \\ 1 & \text{if } z_i^l > 0 \quad \text{otherwise } 0 \end{cases}$$

1.2 Errors at output layer

Define cost-function as cross-entropy:

$$C = \sum_{i=0}^{n} \left[y_i ln(a_i^L) + (1 - y_i) ln(1 - a_i^L) \right] \qquad i = 0, 1, ..., n \text{ indicate different neurons in the output layer}$$

Errors at the output layer (one could treat cost as another neuron where all output neurons converge to):

$$\delta_i^L \equiv \frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L} \frac{\partial a_i^L}{\partial z_i^L} = \left[\frac{y_i}{a_i^L} - \frac{1 - y_i}{1 - a_i^L} \right] \odot g'(z_i^L) \qquad \odot\text{: pointwise multiplication}$$

1.3 Back-propagate errors from output layer to hidden layers

Proof for the back-propagation algorithm using Dynamic Programming.
Using multivariate chain rule, we have:

$$\begin{aligned}
\delta_c^l &= \frac{\partial C}{\partial z_c^l} \quad \text{sum over all possible product sequences from layer } (l+1) \text{ to output layer } (L) \\
&= \sum_{i,j,k,m,...} \frac{\partial C}{\partial z_m^L} \frac{\partial z_m^L}{\partial z_k^{L-1}} \cdots \frac{\partial z_k^{l+3}}{\partial z_j^{l+2}} \frac{\partial z_j^{l+2}}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial z_c^l} \\
&= \sum_i \frac{\partial z_i^{l+1}}{\partial z_c^l} \cdot \sum_{j,k,m,...} \frac{\partial C}{\partial z_m^L} \frac{\partial z_m^L}{\partial z_k^{L-1}} \cdots \frac{\partial z_k^{l+3}}{\partial z_j^{l+2}} \frac{\partial z_j^{l+2}}{\partial z_i^{l+1}} \\
&= \sum_i \frac{\partial z_i^{l+1}}{\partial z_c^l} \cdot \delta_i^{l+1} \\
&= \frac{\partial a_c^l}{\partial z_c^l} \sum_i \frac{\partial z_i^{l+1}}{\partial a_c^l} \cdot \delta_i^{l+1} \\
&= g'(z_c^l) \sum_i w_{ic}^{l+1} \cdot \delta_i^{l+1} \\
&= g'(z_c^l) \left\{ \left[\underset{\sim}{w}_{\cdot c}^{l+1} \right]^T \cdot \delta_i^{l+1} \right\} \qquad \underset{\sim}{w}_{\cdot c}^{l+1}\text{: } c^{th} \text{ column of weight matrix } \boldsymbol{W}^{l+1}
\end{aligned}$$

This leads to an recursion and in the output layer, we have:

$$\delta_i^L \equiv \frac{\partial C}{\partial z_i^L} = \left[\frac{y_i}{a_i^L} - \frac{1 - y_i}{1 - a_i^L} \right]$$

Write in vectorized computation:

$$\underset{\sim}{\delta}^l = g'(\underset{\sim}{z}^l) \odot \left\{ \left[\boldsymbol{W}^{l+1} \right]^T \cdot \underset{\sim}{\delta}^{l+1} \right\}$$

If we rewrite the recursion as a loop starting from the base case, it's the **back-propagation algorithm**. Intuitively, we could think the errors at the output layer as a new "input" ($\underset{\sim}{\delta}^L$), the errors at the hidden layer are obtained by backwards multiplying ($\underset{\sim}{\delta}^L$) with transposed weight matrix $\left[\underset{\sim}{w}^{l+1}_{\cdot c}\right]^T$ and then scaled with the neuron-specific derivatives. To some extent, this calculation is even simpler either than the feed-forward pass as the calculation only involves elementwise multiplication (assuming derivative is already calculated in the feed-forward process).

1.4 Weight update

From the computation above, we know:

$$\begin{aligned}\frac{\partial C}{\partial w^l_{ji}} &= \frac{\partial C}{\partial z^l_j} \cdot \frac{\partial z^l_j}{\partial w^l_{ji}} \\ &= \delta^l_j \cdot a^{l-1}_i\end{aligned}$$

Write in vectorized form:

$$\begin{aligned}\frac{\partial C}{\partial \underset{\sim}{w}^l_{j\cdot}} &= \delta^l_j \cdot \underset{\sim}{a}^{l-1} \\ \Rightarrow \quad \frac{\partial C}{\partial \boldsymbol{W}^l} &= \underset{\sim}{\delta}^l \cdot \left[\underset{\sim}{a}^{l-1}\right]^T\end{aligned}$$

If learning rate is η, the new $\boldsymbol{W}^l$ should be:

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \cdot \frac{\partial C}{\partial \boldsymbol{W}^l} = \boldsymbol{W}^l - \eta \cdot \underset{\sim}{\delta}^l \cdot \left[\underset{\sim}{a}^{l-1}\right]^T$$

1.5 Stochastic gradient descent

Forward pass:

1. $\boldsymbol{X}$: input matrix, dimension is $m \times n$ (n examples each with m features), each column indicates one example.
2. $\boldsymbol{W}$: weight matrix, dimension is $p \times m$ (receives from m inputs, output p linear sum).
3. $\boldsymbol{Z} = \boldsymbol{W} \cdot \boldsymbol{X}$: output matrix, dimension is $p \times n$.

Backward pass:

1. $\boldsymbol{\delta}^L$: error matrix at the output layer, dimension is $p \times n$, each column indicates output error from one example.
2. $\boldsymbol{W}$: weight matrix, dimension is $p \times m$, each row indicates the connection from one input neuron to p output neuron.
3. $\boldsymbol{\delta}^l$:, error matrix at the hidden layer, dimension is $m \times n$, each column indicates the errors at hidden layer (l) from one example.

2 Regularization

For L2 regularization (assume the regularization strength is λ), the cost function (cross-entropy) is:

$$C = \sum_{i=0}^{p} \left[y_i ln(a^L_i) + (1 - y_i) ln(1 - a^L_i) \right] + \frac{\lambda}{2} \sum_{i,j,l} \left[w^l_{ij}\right]^2 \quad \text{all weights in the network}$$

$$\frac{\partial}{\partial w^l_{ij}} \left\{ \frac{1}{2} \sum_{i,j,l} \left[w^l_{ij}\right]^2 \right\} = w^l_{ij}$$

The derivative from regularization term is directly related to each weight and doesn't need back-propagation. For stochastic gradient descent, the update rule becomes (the regularization term is scaled again by the size of training set n):

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \frac{\eta\lambda}{n}\boldsymbol{W}^l - \eta \cdot \frac{\partial C}{\partial \boldsymbol{W}^l} = \boldsymbol{W}^l - \frac{\eta\lambda}{n}\boldsymbol{W}^l - \eta \cdot \underset{\sim}{\delta}^l \cdot \left[\underset{\sim}{a}^{l-1}\right]^T$$