AI lmao

Martin Johnsrud

July 17, 2019

Theory

A neural network is made up of l layers, where the ith layer, $i \in \{1, ..., l\}$, has L_i neurons. Each neuron j has a bias $b_j^{(i)}$ and an activation $a_j^{(i)}$. The activation of the neurons is a function of the activation of the neurons in layer $i-1$. A weighted sum

$$z_j^{(i)} = \sum_{k=0}^{L_i - 1} w_{jk}^{(i)} a_k^{(i-1)} + b_j$$

is passed through an activation function, in this case

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

The activation of a neurons then becomes

$$a_j^{(i)} = f\left(\sum_{k=0}^{L_i - 1} w_{jk}^{(i)} a_k^{(i-1)} + b_j \right),$$

This means layer i is associated with a matrix $w^{(i)} \in \mathbb{R}^{L_i \times L_{i-1}}$. In the special case of $i = 0$ is the activation of the neurons given by an input vector $x \in \mathbb{R}^{L_0}$, and there are no need for wights or biases.

With each input x is there a desired output vector $y \in \mathbb{R}^{L_l}$, wich is compared to the activation of the last layer $a^{(l)}$, by the cost function

$$C = \frac{1}{2} \sum_{j=0}^{L_l - 1} \left(y_j - a_j^{(l)} \right)^2.$$

The goal is to train the neural network, by using gradient decent to minimize C. C is a function of all the weights $w_{jk}^{(i)}$, the biases $b_j^{(i)}$ and the activations $a_j^{(i)}$ given an input x. To find all partial derivatives, back propagation is emploied.

Back propagation

To find representations of the partial derivatives, the cahin rule is used to linke the partial derivative of a layer to the weifghted sum in the next level.

The partial derivatives of the cost function with respect to the weights in the last layer is given by

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}.$$

We have

$$\frac{\partial C}{\partial a_j^{(l)}} = (a_j^{(l)} - y_j)$$

$$\frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = f'(z) = \frac{-\exp(z)}{(\exp(z)+1)^2},$$

and

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = a_k^{(l-1)},$$

giving us this parital derivative. For subsequent layers, the derivative with respect to the weight is

$$\frac{\partial C}{\partial w_{jk}^{(i)}} = \sum_{n=1}^{L_j} \frac{\partial C}{\partial z_n^{(i+1)}} \frac{\partial z_n^{(i+1)}}{\partial a_n^{(i)}} \frac{\partial a_n^{(i)}}{\partial z_j^{(i)}} \frac{\partial z_j^{(i)}}{\partial w_{jk}^{(i)}},$$

as the output depends on how the weight influences the activation in the next layer. Since the weight influences all the nodes in the next level, the partial derivative has to be summed over all the nodes in that layer. As before,

$$\frac{\partial a_n^{(i)}}{\partial z_n^{(i)}} = f'(z) = \frac{-\exp(x)}{(\exp(x)+1)^2},$$

and the other derivatives are

$$\frac{\partial z_j^{(i+1)}}{\partial a_n^{(i)}} = \frac{\partial}{a_n^{(i)}} \left(w_{jn}^{(i+1)} a_n^{(i)} + b_j^{(i+1)}\right) = w_n^{(i+1)}$$

and

$$\frac{\partial z_j^{(i)}}{\partial w_{jk}^{(i)}} = \frac{\partial}{w_{jk}^{(i)}} \left(w_{jk}^{(i)} a_k^{(i-1)} + b_j^{(i)}\right) = a_k^{(i-1)}.$$

To find the full gradient, we also need the partial derivatives with respect to the biases. These can now be easily found, as

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$$

and

$$\frac{\partial C}{\partial b_j^{(i)}} = \sum_{n=1}^{L_j} \frac{\partial C}{\partial z_n^{(i+1)}} \frac{\partial z_n^{(i+1)}}{\partial a_n^{(i)}} \frac{\partial a_n^{(i)}}{\partial z_j^{(i)}} \frac{\partial z_k^{(i)}}{\partial b_j^{(i)}} = \sum_{n=1}^{L_j} \frac{\partial C}{\partial z_n^{(i+1)}} \frac{\partial z_n^{(i+1)}}{\partial a_n^{(i)}} \frac{\partial a_n^{(i)}}{\partial z_j^{(i)}}.$$

We can see that the partial with respect to z gives a recursive reltionship backwards through the neural net, and that the derivatives with respect to the weights and biases can be calculated from it. This gives us a way to calculate the gradient. Substituring for all the known partial derivatives, we get

$$\frac{\partial C}{\partial z_j^{(l)}} = (a_j^{(l)} - y_j) f'(z)$$

$$\frac{\partial C}{\partial z_j^{(i)}} = \sum_{k=1}^{L_j} \frac{\partial C}{\partial z_k^{(i-1)}} f'(z) w_{jk}^{(i)},$$

$$\frac{\partial C}{\partial w_j^{(i)}} = \frac{\partial C}{\partial z_j^{(i)}} a_j^{(i-1)}$$

$$\frac{\partial C}{\partial b_j^{(i)}} = \frac{\partial C}{\partial z_j^{(i)}}$$

The backpropagating algorithm therefore consists of

1. Find $\partial C / \partial z_j^{(l)}$.
2. Loop backwards through the neural net to find $\partial C / \partial z_j^{(i)}$ for $i \in \{1 ... l-1\}$.
3. Use $\partial C / \partial z_j^{(i)}$ to find $\partial C / \partial w_{jk}^{(i)}$ and $\partial C / \partial b_j^{(i)}$.

This will result in the gradient of C with respect to the weights and biases,

$$\nabla C = \begin{bmatrix} \partial C / \partial w_{11}^{(1)} \\ \vdots \\ \partial C / \partial w_{L_l L_l}^{(l)} \\ \partial C / \partial b_j^{(1)} \\ \vdots \\ \partial C / \partial b_{L_l}^{(l)} \end{bmatrix}$$