Neural Network Backpropagation

Mehmet

To compute the gradients, we'll work backwards:

1. Compute the error at the output layer:

$$\boldsymbol{\delta}_2 = (\mathbf{a}_2 - \mathbf{y}) \odot \sigma'(\mathbf{z}_2)$$

   where $\odot$ is element-wise multiplication and σ' is the derivative of the sigmoid function.

2. Compute gradients for $\mathbf{W}_2$ and $\mathbf{b}_2$:

$$\frac{\partial L}{\partial \mathbf{W}_2} = \mathbf{a}_1^T \boldsymbol{\delta}_2$$
$$\frac{\partial L}{\partial \mathbf{b}_2} = \sum \boldsymbol{\delta}_2 \quad \text{(sum over the batch dimension)}$$

3. Backpropagate the error to the hidden layer:

$$\boldsymbol{\delta}_1 = (\boldsymbol{\delta}_2 \mathbf{W}_2^T) \odot \sigma'(\mathbf{z}_1)$$

4. Compute gradients for $\mathbf{W}_1$ and $\mathbf{b}_1$:

$$\frac{\partial L}{\partial \mathbf{W}_1} = \mathbf{a}_0^T \boldsymbol{\delta}_1$$
$$\frac{\partial L}{\partial \mathbf{b}_1} = \sum \boldsymbol{\delta}_1 \quad \text{(sum over the batch dimension)}$$

The gradient of the sigmoid function is:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

These gradients can then be used to update the weights and biases using an optimization algorithm like gradient descent:

$$\mathbf{W} = \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$
$$\mathbf{b} = \mathbf{b} - \eta \frac{\partial L}{\partial \mathbf{b}}$$

where η is the learning rate.
Partial equation for a given weight then derived like this:

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_i} \frac{\partial a_i}{\partial b} \frac{\partial b}{\partial x} \frac{\partial x}{\partial w_1}$$

1 One Neuron Cost

A single neuron and input forwarding contains the multiplication of a weight and a input, the addition of bias, and the activation function used to get the output data.

$$z = w_1 x_1 + b \tag{1}$$

$$f(w_1, b) = \sigma(z) \tag{2}$$

where the activation function is a non-linear logistic function.

$$\sigma(x) = \frac{1}{1 + e^- x} \tag{3}$$

$$C(w_1) = \frac{1}{2} \sum \left(\sigma(z) - y_1\right)^2 \tag{4}$$