Backpropagation

Markus Reynoso

Introduction

The goal of this paper is to demonstrate the computations behind the Backpropagation Algorithm for multilayer perceptrons (MLPs). This paper assumes prior knowledge of basic multivariable calculus, matrix multiplication, Haddamard products, and a general idea of the mechanism of MLPs.

Note that this paper is part of a personal project.

Gradient Descent

Definition 1 (descent direction)**.** *[Beck, 2014] Let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function over $\mathbb{R}^n$. A vector $\mathbf{0} \neq \boldsymbol{d} \in \mathbb{R}^n$ is called a descent direction of f at $\boldsymbol{x}$ if the directional derivative $f'(\boldsymbol{x}; \boldsymbol{d})$ is negative, meaning that*

$$f'(\boldsymbol{x}; \boldsymbol{d}) = \nabla f(\boldsymbol{x}) \cdot d < 0$$

The gradient method generally takes the following iterative form:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + t_k \boldsymbol{d}_k$$

where t is some step-size. The idea is that given an initial guess x_0, incrementing along the descent direction leads to the minimization of the objective function. This leads to the following questions:

1. How to choose a step-size?
2. How to choose a descent direction?

The simplest step-size, of course, is a constant one, which is what we will use. As for the descent direction, we choose the negative gradient of the function, which we can easily prove to be a valid choice.

Proof.

$$\begin{aligned} f'(\boldsymbol{x}; -\nabla f(\boldsymbol{x})) &= \nabla f(\boldsymbol{x}) \cdot -\nabla f(\boldsymbol{x}) \\ &= -||\nabla f(\boldsymbol{x})||^2 \\ &< 0 \end{aligned}$$

□

Altogether, these give us the core iterative formula for the gradient descent algorithm:

$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k - \beta \nabla f(\boldsymbol{x}) \tag{1}$$

for some constant β, which we generally want to be a small value. In component form, this takes on the more common expression:

$$\theta_{k+1} \leftarrow \theta_k - \beta \frac{\partial f}{\partial \theta} \tag{2}$$

where θ is an entry in the vector $\boldsymbol{x}$. This form is probably what the reader is familiar with.

The Backpropagation Algorithm

Computing δ_j^L and generalizing to δ^L

First, recall that the cost function is simply a function of the activations of the neurons in the output layer. By the chain rule for multivariable functions:

$$\begin{aligned} \frac{\partial C}{\partial z_j^L} &= \frac{\partial C}{\partial a_1^L}\frac{\partial a_1^L}{\partial z_j^L} + \frac{\partial C}{\partial a_2^L}\frac{\partial a_2^L}{\partial z_j^L} + \cdots + \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} + \cdots + \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} \\ \delta_j^L &= \frac{\partial C}{\partial a_j^L} a_j'^{(L)} \end{aligned}$$

Note that this computation gives us an expression for $\frac{\partial C}{\partial z_j^L}$; the partial derivative of the cost function with respect to the j-th neuron in the output layer. However, we would want to see how this computation is performed in the bigger picture. Converting this to matrix form is as simple as:

$$\delta^L = \begin{bmatrix} \frac{\partial C}{\partial a_1^L} a_1'^{(L)} \\ \frac{\partial C}{\partial a_2^L} a_2'^{(L)} \\ \vdots \\ \frac{\partial C}{\partial a_s^L} a_s'^{(L)} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial C}{\partial a_1^L} \\ \frac{\partial C}{\partial a_2^L} \\ \vdots \\ \frac{\partial C}{\partial a_s^L} \end{bmatrix} \odot \begin{bmatrix} a_1'^{(L)} \\ a_2'^{(L)} \\ \vdots \\ a_s'^{(L)} \end{bmatrix}$$

which finally leaves us with:

$$\delta^L = \nabla_{a^L} C \odot a'^{(L)} \tag{3}$$

The reader may be wondering why the gradient for the output layer is treated as a separate case. This question will be answered shortly.

Computing δ_j^l and generalizing to δ^l

$$\begin{aligned} \delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \frac{\partial C}{\partial z_1^{l+1}} \frac{\partial z_1^{l+1}}{\partial z_j^l} + \frac{\partial C}{\partial z_2^{l+1}} \frac{\partial z_2^{l+1}}{\partial z_j^l} + \cdots + \frac{\partial C}{\partial z_s^{l+1}} \frac{\partial z_s^{l+1}}{\partial z_j^l} \\ &= \delta_1^{l+1} \frac{\partial z_1^{l+1}}{\partial z_j^l} + \delta_2^{l+1} \frac{\partial z_2^{l+1}}{\partial z_j^l} + \cdots + \delta_s^{l+1} \frac{\partial z_s^{l+1}}{\partial z_j^l} \end{aligned}$$

Aside: Computing $\frac{\partial z_i^{l+1}}{\partial z_j^l}$

$$z_i^{l+1} = w_{i1} a_1^l + w_{i2} a_2^l + \cdots w_{is} a_s^l + b_i^{l+1}$$

$$\frac{\partial z_i^{l+1}}{\partial z_j^l} = w_{ij} a_j'^{(l)}$$

$$\begin{aligned} &= \delta_1^{l+1} w_{1j}^{l+1} a_j'^{(l)} + \delta_2^{l+1} w_{2j}^{l+1} a_j'^{(l)} + \cdots + \delta_s^{l+1} w_{sj}^{l+1} a_j'^{(l)} \\ \delta_j^l &= a_j'^{(l)} \sum_{\forall i} w_{ij}^{l+1} \delta_i^{l+1} \end{aligned}$$

Once again, converting this into matrix form:

$$
\begin{aligned}
\delta^l &= \begin{bmatrix} a_1'^{(l)} \\ a_2'^{(l)} \\ \vdots \\ a_s'^{(l)} \end{bmatrix} \odot \begin{bmatrix} \sum_{\forall i} w_{i1}^{l+1}\delta_i^{l+1} \\ \sum_{\forall i} w_{i2}^{l+1}\delta_i^{l+1} \\ \vdots \\ \sum_{\forall i} w_{is}^{l+1}\delta_i^{l+1} \end{bmatrix} \\
&= \begin{bmatrix} a_1'^{(l)} \\ a_2'^{(l)} \\ \vdots \\ a_s'^{(l)} \end{bmatrix} \odot \begin{bmatrix} w_{11}^{l+1} & w_{21}^{l+1} & w_{31}^{l+1} & \cdots & w_{s1}^{l+1} \\ w_{12}^{l+1} & w_{22}^{l+1} & w_{32}^{l+1} & \cdots & w_{s2}^{l+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{1s}^{l+1} & w_{2s}^{l+1} & w_{3s}^{l+1} & \dots & w_{ss}^{l+1} \end{bmatrix} \begin{bmatrix} \delta_1^{l+1} \\ \delta_2^{l+1} \\ \vdots \\ \delta_s^{l+1} \end{bmatrix} \\
&= \begin{bmatrix} a_1'^{(l)} \\ a_2'^{(l)} \\ \vdots \\ a_s'^{(l)} \end{bmatrix} \odot \begin{bmatrix} w_{11}^{l+1} & w_{12}^{l+1} & w_{13}^{l+1} & \cdots & w_{1s}^{l+1} \\ w_{21}^{l+1} & w_{22}^{l+1} & w_{23}^{l+1} & \cdots & w_{2s}^{l+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{s1}^{l+1} & w_{s2}^{l+1} & w_{s3}^{l+1} & \dots & w_{ss}^{l+1} \end{bmatrix}^T \begin{bmatrix} \delta_1^{l+1} \\ \delta_2^{l+1} \\ \vdots \\ \delta_s^{l+1} \end{bmatrix}
\end{aligned}
$$

which finally leaves us with:

$$\delta^l = a'^{(l)} \odot ((w^{l+1})^T \delta^{l+1}) \tag{4}$$

Notice that computing δ^l requires first knowing δ^{l+1}. This demonstrates the importance of establishing δ^L separately because it serves as the base case for the recursive form of the gradient.

Updating parameters

Now that we have expressions for δ^L and δ^l, we can now move on to updating the parameters of our neural network using (1).

Output layer weights

$$
\begin{aligned}
w^L_{jk} &\leftarrow w^L_{jk} - \beta \frac{\partial C}{\partial w^L_{jk}} \\
&= w^L_{jk} - \beta \frac{\partial C}{\partial z^L_j} \frac{\partial z^L_j}{\partial w^L_{jk}} \\
&= w^L_{jk} - \beta \frac{\partial C}{\partial z^L_j} \frac{\partial}{\partial w^L_{jk}} (w^L_{j1} a^{L-1}_1 + w^L_{j2} a^{L-1}_2 + \ldots + w^L_{jk} a^{L-1}_k + \ldots + w^L_{js} a^{L-1}_s + b^L j) \\
w^L_{jk} &\leftarrow w^L_{jk} - \beta \delta^L_j a^{L-1}_k
\end{aligned}
$$

This tells us how to update a single weight in the output layer. Once again, we want to see how this operation looks like for the entire matrix. The weight matrix of the output layer is given by the following:

$$
\begin{bmatrix}
w^L_{11} & w^L_{12} & w^L_{13} & \cdots & w^L_{1s} \\
w^L_{21} & w^L_{22} & w^L_{23} & \cdots & w^L_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
w^L_{s1} & w^L_{s2} & w^L_{s3} & \ldots & w^L_{ss}
\end{bmatrix}
$$

and we wish to update it to the following value:

$$
\begin{bmatrix}
w^L_{11} - \beta \delta^L_1 a^{L-1}_1 & w^L_{22} - \beta \delta^L_1 a^{L-1}_2 & w^L_{33} - \beta \delta^L_1 a^{L-1}_3 & \cdots & w^L_{1s} - \beta \delta^L_1 a^{L-1}_s \\
w^L_{21} - \beta \delta^L_2 a^{L-1}_1 & w^L_{22} - \beta \delta^L_2 a^{L-1}_2 & w^L_{23} - \beta \delta^L_2 a^{L-1}_3 & \cdots & w^L_{2s} - \beta \delta^L_2 a^{L-1}_s \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
w^L_{s1} - \beta \delta^L_s a^{L-1}_1 & w^L_{s2} - \beta \delta^L_s a^{L-1}_2 & w^L_{s3} - \beta \delta^L_s a^{L-1}_3 & \ldots & w^L_{ss} - \beta \delta^L_s a^{L-1}_s
\end{bmatrix}
$$

$$
\begin{aligned}
&= \begin{bmatrix}
w^L_{11} & w^L_{22} & w^L_{33} & \cdots & w^L_{1s} \\
w^L_{21} & w^L_{22} & w^L_{23} & \cdots & w^L_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
w^L_{s1} & w^L_{s2} & w^L_{s3} & \ldots & w^L_{ss}
\end{bmatrix} - \beta \begin{bmatrix} \delta^L_1 \\ \delta^L_2 \\ \vdots \\ \delta^L_s \end{bmatrix} \begin{bmatrix} a^{L-1}_1 & a^{L-1}_2 & \cdots & a^{L-1}_s \end{bmatrix} \\
&= w^L - \beta \delta^L (\beta^{L-1})^T
\end{aligned} \tag{5}
$$

Substituting our result from (3), we now get the following:

$$
w^L \leftarrow w^L - \beta (\nabla_{a^L} C \odot a'^{(L)}) (a^{L-1})^T
$$

Updating output layer biases

We do this same for the biases in the output layer:

$$
\begin{aligned}
b_j^L &\leftarrow b_j^L - \beta \frac{\partial C}{\partial b_j^L} \\
&= b_j^L - \beta \frac{\partial C}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_j^L} \\
&= b_j^L - \beta \frac{\partial C}{\partial z_j^L} \frac{\partial}{\partial b_j^L} (w_{j1}^L a_1^{L-1} + w_{j2}^L a_2^{L-1} + \ldots + w_{jk}^L a_k^{L-1} + \ldots + w_{js}^L a_s^{L-1} + b^L j) \\
b_j^L &\leftarrow b_j^L - \beta \delta_j^L
\end{aligned}
$$

Converting this into matrix form is much simpler than what we saw from the weights. We start off with the biases:

$$
\begin{bmatrix}
b_1^L \\
b_2^L \\
\vdots \\
b_s^L
\end{bmatrix}
$$

and update it to the following:

$$
\begin{bmatrix}
b_1^L - \beta \delta_1^L \\
b_2^L - \beta \delta_2^L \\
\vdots \\
b_s^L - \beta \delta_s^L
\end{bmatrix}
$$

which is simply just;

$$= b^L - \beta \delta^L$$

Substituting our computed value of δ^L from (3), we finally get our descent formula for b^L:

$$b^L \leftarrow b^L - \beta [\nabla_{a^L} C \odot a'^{(L)}] \tag{6}$$

Hidden layer weights

Updating hidden layer weights follows the same process as updating output layer weights (5), but we change the superscripts. The only difference is the expression of δ for these layers.

$$w^l - \beta \delta^l (a^{l-1})^T$$

Substituting the computed value of δ^l from (4):

$$w^l \leftarrow w^l - \beta [a'^{(l)} \odot ((w^{l+1})^T \delta^{l+1})](a^{l-1})^T$$

Hidden layer biases

Updating hidden layer biases follows the same process as updating output layer biases (6), but we change the superscripts and the delta value (4)

$$b^l \leftarrow b^l - \beta [a'^{(l)} \odot ((w^{l+1})^T \delta^{l+1})]$$

References

[Beck, 2014] Beck, A. (2014). *Introduction to Nonlinear Optimization*. Society for Industrial and Applied Mathematics and the Mathematical Optimization Society, United States of America.