Back propagation algorithm

Jack Buckner

May 2025

1 Why back propagation?

Back propagation is a computationally efficient algorithm for calculating the effect of the weights of a neural network on the quality of its predictions. This algorithm allows deep neural networks to be trained efficiently using first-order optimization techniques like gradient descent.

2 Derivation

When solving complex problems, such as deriving an efficient method for calculating the derivative of complex objects like an artificial neural network, it can often be quite beneficial to write out clear notation and consider multiple formulations of the problem.

When deriving the backpropagation algorithm, it is very helpful to notice that the computations in feedforward networks can be expressed using both function composition and an interactive sequence of calculations.

If we let the output of the neural network by $\mathbf{y}$, the input $\mathbf{x}$, the weights in layer l be W_l and the activation function of layer l be f_l then the prediction of a neural network with L layers can be expressed as

$$\mathbf{y} = f_L(W_L f_{L-1}(W_{L=1}...f_1(W_1\mathbf{x})...)) \tag{1}$$
$$\mathbf{y} = f_L \circ W_L \cdot f_{L-1} \circ W_{L-1} \cdot f_{L-2} \circ ... \circ W_1 \cdot f_1(\mathbf{x}) \tag{2}$$
$$\tag{3}$$

A feed-forward network can also be written as an iterative process of applying the weights W_l to the activations in the prior layer $\mathbf{a}_l$ to get a set of inputs $\mathbf{z}_l$ for the current layer. These inputs are then used to compute the activations $\mathbf{a}_l$ by applying the activation function f_l

$$\mathbf{z}_1 = W_1\mathbf{x} \tag{4}$$
$$\mathbf{a}_1 = f_1(\mathbf{z}_1) \tag{5}$$
$$... \tag{6}$$
$$\mathbf{z}_l = W_l\mathbf{a_{l-1}} \tag{7}$$
$$\mathbf{a}_l = f_l(\mathbf{z}_l) \tag{8}$$
$$... \tag{9}$$
$$\mathbf{y} = f_L(\mathbf{z}_L) \tag{10}$$
$$\tag{11}$$

This notation will be very helpful in out derivation so it is useful to summarize here:

- z_l - Inputs to layer l
- a_l - Activations in layer l
- W_l - weights in layer l
- f_l - Activation function in layer l

The goal of the back propagation algorithm will be to calculate the derivative of the loss function C given a data point $\mathbf{t}$ with respect to the weights of the neural network. $\{W_1, W_2, ..., W_L\}$

The basic intuition for the algorithm is that we are going to calculate the effect of the weights W_l layer by layer, starting with the last layer of the network. To do this we will use the fact that the effect of weights in layer l on the loss C depends on the effect of the models prediction y on the loss, the impact of the inputs $\mathbf{z}_l$ to layer l on the prediction y and the effect of the weights W_l

$$\frac{\partial C}{\partial W_l} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial \mathbf{z}_l}\frac{\partial \mathbf{z}_l}{\partial W_l} \tag{12}$$

This leaves us with three terms we need to be able to calculate to get our desired quantity, the derivative of the loss with respect to the prediction $\frac{\partial C}{\partial W_l}$, the derivative of the prediction with respect to the input in layer $\frac{\partial y}{\partial \mathbf{z}_l}$ and the derivative of the input to layer l with respect to the weights $\frac{\partial \mathbf{z}_l}{\partial W_l}$.

2.1 Derivative of loss with respect to the output

The derivative of the cost with respect to the prediction will depend on the loss function chosen, but should be relatively straight forward. For example, regression models will often use the squared error loss function

$$C(t, y) = (t - y)^2 \tag{13}$$

$$\frac{\partial C}{\partial y} = -2(t - y) \tag{14}$$

2.2 Derivative the inputs with respect to the weights

The second derivative we need is the effect of the weights W_l in the inputs $\mathbf{z}_l$ to each layer. Notice there is a linear relationship between the weights and the inputs, so the derivative is just equal to the activations in the prior layer that we multiply the weights by

$$\mathbf{z}_l = W_l \mathbf{a}_{l-1} \tag{15}$$

$$\frac{\partial \mathbf{z}_l}{\partial W_l} = \mathbf{a}_{l-1}. \tag{16}$$

2.3 Derivative the output inputs with respect to the layer inputs

The difficult step of the algorithm is computing the derivative of the predictions with respect to the inputs of each layer. However, rather than trying to solve that problem directly it can help to solve a related problem which is calculating the derivative of the network with respect to the inputs $\partial \mathbf{y}/\partial \mathbf{x}\ \nabla_{\mathbf{x}} L$.

Remember that the structure of a neural network can be written as the composition of a sequence of functions.

$$\mathbf{y} = f_L(W_L f_{L-1}(W_{L=1}...f_1(W_1\mathbf{x})...)) \tag{17}$$

If we know the derivative of each function on its own we can apply the chain rule to evaluate the derivative of the composite function

$$f(x) = g(h(x)) \tag{18}$$

$$f'(x) = h'(x) \times g'(h(x)) \tag{19}$$

$$f(x) = g(h_1(h_2(x))) \tag{20}$$

$$f'(x) = h_2'(x) \times h_1'(h_2(x)) \times g'(h_1((h_2(x))) \tag{21}$$

$$f'(x) = h_2' \times h_1' \times g' \tag{22}$$

We can apply this to our neural network by multiplying the derivatives of the inputs $\mathbf{z}_l$ and activations $\mathbf{a}_l$ at each layer together until we reach the input $\mathbf{x}$

$$\frac{\partial y}{\partial x} = \cdot\frac{\partial y}{\partial \mathbf{z}_L} \cdot \frac{\partial \mathbf{z}_L}{\partial \mathbf{a}_{L-1}} \cdot \frac{\partial \mathbf{a}_{L-1}}{\partial \mathbf{z}_{L-1}} \cdot \frac{\partial \mathbf{z}_{L-1}}{\partial \mathbf{a}_{L-2}} \cdot ... \cdot \frac{\partial \mathbf{a}_1}{\partial \mathbf{z}_1} \cdot \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}} \tag{23}$$

This expression has two basic terms: the derivative of the activations with respect to the inputs $\frac{\partial \mathbf{a}_l}{\partial \mathbf{z}_l}$ and the derivative of the inputs with respect to the activations $\frac{\partial \mathbf{z}_l}{\partial \mathbf{a}_{l-1}}$.The derivative of the activation a with respect to the input of the activation function is the derivative of the activation function f_l evaluated at the input z_l

$$\frac{\partial \mathbf{a}_l}{\partial \mathbf{z}_l} = f_l'(\mathbf{z}_l) \tag{24}$$

The derivative of the inputs z_l with respect to the activations of the last layer a_{l-1} is equal to the weight matrix W_l

$$\frac{\partial \mathbf{z}_l}{\partial \mathbf{a}_{l-1}} = W_l \tag{25}$$

This allows us to write out the derivative in terms of the weight matrices W_l and the derivative of the activation functions f_l

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \mathbf{y}} \cdot f'_L(\mathbf{z}_L) \cdot W_L \cdot f'_{L-1}(\mathbf{z}_{L-1}) W_{L-1} \cdot ... \cdot f'_1(\mathbf{z}_1) \cdot W_1 \mathbf{x} \tag{26}$$

Notice that we can evaluate the product iteratively in much the same way that we evaluated the network iteratively in the forward direction. the effect of the input to the last layer $\mathbf{z}_L$ in the output is just the derivative of the activation function f'_L. We will call this value δ_L.

We can then calculate the effect of the input in the next layer back $\mathbf{z}_{L-1}$ by multiplying δ_L by the weights W_L to get the effect of the activation $\mathbf{a}_{L-1}$ in the last layer and the derivative of the activation function f'_{L-1} to get the effect of the inputs $\mathbf{z}_l$. This process can be repeated, iterating backwards through the network until the first layer is reached.

$$\lambda_L = \frac{\partial y}{\partial \mathbf{z}_L} = f'_L(\mathbf{z}_L) \tag{27}$$
$$\lambda_{L-1} = \lambda_L \cdot W_L \cdot f'_{L-1}(\mathbf{z}_{L-1}) \tag{28}$$
$$... \tag{29}$$
$$\lambda_{l-1} = \lambda_l \cdot W_l \cdot f'_{l-1}(\mathbf{z}_{l-1}) \qquad ... \tag{30}$$
$$\lambda_1 = \lambda_2 \cdot W_1 \tag{31}$$

You might have noticed that in making this calculation, we have generated a factor δ_l; this is actually equal to the quantity we need! The derivative of the output with respect to the inputs to each layer

$$\frac{\partial \mathbf{y}}{\partial \mathbf{z}_l} = \lambda_l \tag{32}$$

This quantity allows us to calculate the final gradient we want

$$\frac{\partial C}{\partial W_l} = \frac{\partial C}{\partial y} \lambda_l \frac{\partial \mathbf{z}_l}{\partial W_l} = \frac{\partial C}{\partial y} \lambda_l \mathbf{a}_{l-1}. \tag{33}$$

As the last step, notice that calculating the gradients this way would require that we multiply the factor $_l$ by the derivative of the loss function $\frac{\partial C}{\partial y}$ at each layer. We can eliminate these repeated calculations by multiplying the first step in the accumulation procedure by $\frac{\partial C}{\partial y}$. This creates a new factor δ_l

$$\delta_L = \frac{\partial C}{\partial \mathbf{z}_L} = \frac{\partial C}{\partial \mathbf{y}} f'_L(\mathbf{z}_L) \tag{34}$$
$$\delta_{L-1} = \delta_L \cdot W_L \cdot f'_{L-1}(\mathbf{z}_{L-1}) \tag{35}$$
$$... \tag{36}$$
$$\delta_{l-1} = \delta_l \cdot W_l \cdot f'_{l-1}(\mathbf{z}_{l-1}) \qquad ... \tag{37}$$
$$\delta_1 = \delta_2 \cdot W_1 \tag{38}$$

Our final gradient is then equal to

$$\frac{\partial C}{\partial W_l} = \delta_l \mathbf{a}_{l-1}. \tag{39}$$

2.4 Algorithm scaling

The reason this algorithm is so powerful is that it scales very efficiently in the number of neural network parameters. When we evaluate a neural network in the forward direction to calculate a prediction, the primary computational cost is multiplying the weight matrices by the activations in the previous layer.

On the backward pass, we require only about twice this number of calculations to compute the gradient. The backward pass multiplies the weights W_l by the derivative of the activation function f_l at each layer to get the factor δ and the factor δ is multiplied by the activations $\mathbf{a}_l$ to calculate the effect of the weights on the loss.

You might also notice that in order to calculate the gradient, we need to evaluate the neural network and save the inputs and activations in each layer, so we know where to evaluate the derivative of each function. However, these quantities are all vectors while the weights are all matrices, so we only need about the square root of the number of weights of additional memory.