Proofs for the Four Fundamental Equations of the Backpropagation and Algorithms in Feedforward Neural Networks

Senior Project Research Paper

Hao Li

June 1, 2021

1. Introduction

In machine learning, **backpropagation** is a widely used algorithm for training feedforward neural networks [3]. The term *backpropagation* and its general use in neural networks was announced in Rumelhart, Hinton & Williams (1986a).[1] Generally speaking, *backpropagation* is an algorithm that quickly computes the gradient of the cost function with respect to any weights and biases in each layer of the network. Then with the help of stochastic gradient descent technique, the network will be able to find the best weights and biases that minimize the cost function which is a function measures the errors between the predictions and correct answers.

This paper will focus on proving the four fundamental equations of the backpropagation. Then I will show how to use this algorithm combined with the stochastic gradient descent technique to implement the network for recognizing the handwritten digits. Parts of the the proof are provided by the author Michael Nielsen in his online book *Neural Networks and Deep Learning* [2]. Meanwhile, this paper will provide more details of his proofs and some basic definitions of gradient.

- Key words: **Backpropagation**, **Machine Learning**, **Neural Networks**, **Gradient Descent Technique**, **Recognition of Handwritten Digits**.

2. Definitions

Notations: Let $w^l_{j,k}$ denotes the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. b^l_j denotes the j^{th} bias in the l^{th} layer. The activation a^l_j can be written as

$$a^l_j = \sigma\Big(\sum_k w^l_{j,k} \cdot a^{l-1}_k + b^l_j\Big)$$

[1]Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986a). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.

where σ is our activation function. Let C denotes the cost function that measures the error between prediction a^L and right answer, where the variables are the weights and biases.

Definition 1. *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a scalar field. The gradient of g at $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is denoted $\nabla g(\mathbf{x})$ and defined by*

$$\nabla g(\mathbf{x}) = \Big(\frac{\partial g}{\partial x_1}(\mathbf{x}), \frac{\partial g}{\partial x_2}(\mathbf{x}), \ldots, \frac{\partial g}{\partial x_n}(\mathbf{x}) \Big)$$

Definition 2. *Let $\mathbf{v}$ be a unit vector. The directional derivative of g at $\mathbf{x}$ in the direction of $\mathbf{v}$ is defined by*

$$D_v g(\mathbf{x}) = \mathbf{v} \cdot \nabla g(\mathbf{x})$$

Definition 3. *The error of the j^{th} neuron in the layer l is defined by:*

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

where $z_j^l = \sum_k w_{j,k}^l \cdot a_k^{l-1} + b_j^l$.

3. Theorems

Let the cost function C be the mean square error function that is

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \sum_{\mathbf{x}} ||y(\mathbf{x}) - \mathbf{a}||^2$$

Here, $\mathbf{w}, \mathbf{b}$ denotes the collection of all weights and biases, n is the total number of training inputs. $\mathbf{a}$ is the vector of network outputs (prediction) and $y(\mathbf{x})$ is the desired output (correct answer) when $\mathbf{x}$ is the inputs.

Theorem 4. *The partial derivative of the cost with respect to any weights and biases can be found through the following four equations:*

(BP1) $$\delta_j^L = \frac{\partial C}{\partial a_j^L}\ \sigma'(z_j^L)$$

(BP2) $$\delta_j^l = \sum_k w_{k,j}^{l+1}\ \delta_k^{l+1}\ \sigma'(z_j^l)$$

(BP3) $$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

(BP4) $$\frac{\partial C}{\partial w_{j,k}^l} = a_k^{l-1}\ \delta_j^l$$

Equation (BP1) tells us how to compute the error in the last layer of a feedforward network. Then equation (BP2) allows us to write error in the l^{th} layer in terms of error in the layer $l+1$. That means we can back-propagate the error from the last layer to the 2^{th} layer. Equation (BP3) and (BP4) tell us the partial derivative of the cost with respect to any biases is equal the error in that layer. The partial derivative of the cost with respect to any weights is equal to the multiplication of the error in the layer l with the activation in the layer $l-1$.

Proof.

- For equation (BP1), we first write the error in the last layer by the definition:

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

Since $a_j^L = \sigma(z_j^L)$, by the chain rule, we have:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}$$

Since $\partial a_j^L / \partial z_j^l = \sigma'(z_j^L)$, after substitution, the above equation becomes:

(BP1) $$\delta_j^L = \frac{\partial C}{\partial a_j^L}\ \sigma'(z_j^L)$$

- For equation (BP2), we are going to take the partial derivative of cost with respect to z_k in the layer $l+1$. By the chain rule, we have

$$\begin{aligned}\delta_j^l &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \cdot \delta_k^{l+1}\end{aligned}$$

Note that if the z_k^{l+1} in the denominator and numerator cancels out, that is just another way to express error δ_j^l. The last equality holds because of the definition of δ_k^{l+1}. Now by the definition of z_k^{l+1}, we have

$$z_k^{l+1} = \sum_j w_{k,j}^{l+1}\ a_j^l + b_k^{l+1} = \sum_j w_{k,j}^{l+1}\ \sigma(z_j^l) + b_k^{l+1}$$

Differentiating, we have

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{k,j}^{l+1}\ \sigma'(z_j^l)$$

Substituting back into the first equation, we get

(BP2) $$\delta_j^l = \sum_k w_{k,j}^{l+1}\ \delta_k^{l+1}\ \sigma'(z_j^l)$$

- For equation (BP3), first note that

$$z_j^l = \sum_k w_{j,k}^l\ a_k^{l-1} + b_j^l$$

Taking the derivative with respect to b_j^l, we have

$$\frac{\partial z_j^l}{\partial b_j^l} = 1$$

Now mutiply $\partial z_j^l / \partial b_j^l$ to both sides of the error formula, we have

$$\begin{aligned} \delta_j^l &= \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} \\ &= \frac{\partial C}{\partial b_j^l} \end{aligned} \tag{BP3}$$

- For equation (BP 4), similarly, we take the derivative of z_j^l with respect to $w_{j,k}^l$, we get:

$$\frac{\partial z_j^l}{\partial w_{j,k}^l} = a_k^{l-1}$$

Now multiply this equation to both sides of the error formula, we have

$$\begin{aligned} \delta_j^l \; a_k^{l-1} &= \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{j,k}^l} \\ &= \frac{\partial C}{\partial w_{j,k}^l} \end{aligned} \tag{BP4}$$

□

Now we can quickly find gradients of the cost function with respect to any weights and biases. Let me introduce the gradient descent technique. With this technique, we will be able to find the best weights and biases that minimize the cost function.

4. Algorithms

Let $g : \mathbb{R}^n \to \mathbb{R}$ and let $\mathbf{x}^{(0)}$ denote the initial point. Let η be a constant greater than zero. Note that in our network, the $\mathbf{x}$ are the vector of weights and biases $\mathbf{w}, \mathbf{b}$.

Gradient Descent Algorithm:

(1) Evaluate g at initial point $\mathbf{x}^{(0)}$.

(2) Determine a direction from $\mathbf{x}^{(0)}$ that results in a fastest decease in the value of g.

(3) Move a certain amount in this direction and call the new value $\mathbf{x}^{(1)}$.

(4) Repeat steps 1 through 3 with $\mathbf{x}^{(0)}$ replaced by $\mathbf{x}^{(1)}$ [1].

The following equation generalizes the step (1) to (3).

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \eta \, \nabla g(\mathbf{x}^{(0)})$$

However, to apply the algorithm, we will do something different. This is because if we apply this algorithm for every weight and biase, the cost of computation is too high. The idea is to estimate the gradient ∇C by computing ∇C_x for small sample of randomly chosen training inputs.

We will label random training inputs $X_1, X_2, \ldots, X_m$ and refer to them as a *mini-batch*. If the sample size m is large enough, we expect that the average value of the ∇C_{X_j} will be about equal to the average over all ∇C_x, that is

$$\frac{\sum_{j=1}^{n} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

So, if we apply the **stochastic gradient descent technique**, the new weights and biases are given by:

$$\mathbf{w}_k^{(0)} \to \mathbf{w}_k^{(1)} = \mathbf{w}_k^{(0)} - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$\mathbf{b}_l^{(0)} \to \mathbf{b}_l^{(1)} = \mathbf{b}_l^{(0)} - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}$$

Now let me introduce the algorithm that combines the stochastic gradient descent technique and backpropagation to implement the network for recognizing the handwritten digits. First, given a mini-batch of m training examples:

The Backpropagation Algorithm:

(1) **Input a set of training example**

(2) **For each training example** $\mathbf{x}$:

- **Feedforward:** Compute $z^{x,l} = w^l \, a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$ from layer $2 \sim L$.

- **Output error:** Compute the error vector at the output layer: $\delta^{x,L} = \nabla_{a^L} C_x \circ \sigma'(z^{x,L})$.
- **Backpropagate the error:** Find the error vector at each layer by equation (BP2).

(3) **Update the weights and biases:**

$$w^l \to w^l - \frac{\eta}{m} \sum_x \delta^{x,l} \ (a^{x,l-1})^T$$

$$b^l \to b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

Note that inside the summation, we just replace the gradient with the equation (BP3) and (BP4).

So after each mini-batch, we will update the weights and biases and we will repeat this process for multiple iterations to check the accuracy.

5. Conclusions

We see that the output/prediction from a feedforward neural network is just a function of weights and biases. The weights and biases are very important because they give us how good this network can predicate. So the key idea of the machine learning is to learn a good weight and biases so that the machine can give us the prediction as close as possible to the correct answer. Therefore, the heart of this feedforward network is the gradient descent technique and backpropagation. They allow us to find the best weights and biases quickly and precisely and such weights and biases will minimize the errors between the predictions and correct answers.

Again, all the algorithms and some proofs of the theorem are provided by Michael Nielsen in his online book *Neural Networks and Deep Learning* [2]. The URL is given in the footnote. [2]

[2] http://neuralnetworksanddeeplearning.com/index.html

6. Appendix

In the appendix, I want to give a theorem and its proof to show why we should follow the gradient to minimize the cost.

Theorem 5. *∇f points in the direction of the maximum rate of increase of the scalar field f*

Proof. First, by the definition of dot product, we have

$$\begin{aligned} D_v f(\mathbf{x}) &= |\nabla f|\ |\mathbf{v}|\ \cos\theta \\ &= |\nabla f|\ \cos\theta \end{aligned}$$

where θ is the angle between ∇f and $\mathbf{v}$. Since $\cos\theta \in [-1, 1]$, then we have

$$D_v f(\mathbf{x})_{max} = |\nabla f| \quad \text{when } \theta = 0^\circ$$

$$D_v f(\mathbf{x})_{min} = |\nabla f| \quad \text{when } \theta = 180^\circ$$

□

Because the directional derivative tells us how fast the scalar field (in our case it is the cost) is changing in the directive of $\mathbf{v}$ at the point $\mathbf{x}$, the theorem says the quickest path to the minimum of the scalar field is to follow where the gradient is pointing.

References

[1] Richard L. Burden and J. Douglas Faires. *Numerical Analysis*. Thomson Brooks/Cole, 8 edition, 2005.

[2] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 1 edition, 2015.

[3] Wikipedia contributors. Backpropagation — Wikipedia, the free encyclopedia, 2021. [Online; accessed 1-June-2021].