Backpropagation

Backpropagation is an algorithm for computing the gradient of a loss function for an ANN.

Chain Rule: For the composition of functions $f(x) = (f_1 \circ f_2 \circ \cdots \circ f_{n-1} \circ f_n)(x)$,

$$\frac{df}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3}\cdots\frac{df_{n-1}}{df_n}\frac{df_n}{dx}$$

(1) Let $f = 4f_1^2$, $f_1 = f_2 - 2$, $f_2 = 5x^3 - x + 1$. Use the chain rule to compute $\frac{df}{dx}$.

Recall the chain rule with partial derivatives:

Chain Rule with Partial Derivatives Let us assume $z = f(x, y)$ and that both x and y are functions of other variables u, v, i.e $x = x(u, v), y = y(u, v)$. Then we can define $\frac{\partial f}{\partial u}, \frac{\partial f}{\partial v}$ using chain rule,

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial u} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial u}, \qquad \frac{\partial f}{\partial v} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial v} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial v}$$

More generally, if $z = f(x_1, x_2, \ldots, x_n)$ and each x_i are a function of u, v, then

$$\frac{\partial f}{\partial u} = \sum_{k=1}^{n} \frac{\partial f}{\partial x_k}\frac{\partial x_k}{\partial u}$$

Similarly for $\frac{\partial f}{\partial v}$.

(2) Suppose $f(x, y) = 2x^3 + y$, $x(u, v) = \sin(u) + \cos(v)$, and $y(u, v) = e^{uv}$. Compute $\frac{\partial f}{\partial u}$ and $\frac{\partial f}{\partial v}$.

Remember that an FNN is a sequence of compositions of linear transformations and non-linear functions. The chain rule is the most important ingredient for computing the gradients of a given loss function for the weights of an FNN.

Notation: Define the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer as w_{jk}^l. Similarly, we denote by b_j^l the j^{th} bias in the l^{th} layer, a_j^l the j^{th} activated neuron in the l^{th} layer (after applying the activation function), and z_j^l the j^{th} pre-activated neuron (before applying the activation function).

(3) Write out the labels of each neuron in Figure 1 using the notation a_j^l.

Let σ be an activation function. With this notation,

$$a_j^l = \sigma\left(z_j^l\right), \qquad z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l, \tag{1}$$

Each layer of a FNN represents a vector. To get from one layer to the next, we compute $\sigma(W\mathbf{a} + \mathbf{b})$ where σ is an activation function, W is a matrix, $\mathbf{a}$ is a vector representing the

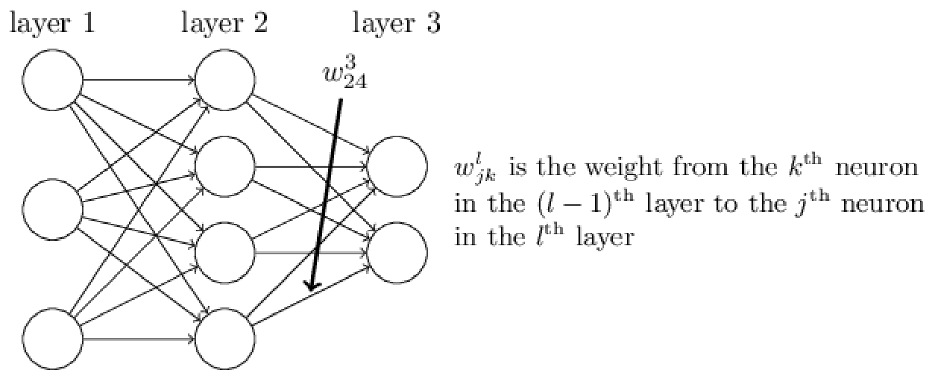

FIGURE 1. Image from Michael Nielsen's *Neural Networks and Deep Learning*.

previous layer, and $\mathbf{b}$ is a bias vector. For consistency of notation, use W^l and $\mathbf{b}^l$ to represent the matrix and biases that maps the $(l-1)^{\text{th}}$ layer $\mathbf{a}^{l-1}$ to the l^{th} layer $\mathbf{a}^l$. Hence,

$$\mathbf{a}^l = \sigma\left(\mathbf{z}^l\right), \qquad \mathbf{z}^l = W^l\mathbf{a}^{l-1} + \mathbf{b}^l \tag{2}$$

(4) Consider the FNN in Figure 1. Write out the matrices W^l and bias vectors $\mathbf{b}^l$ for $l = 2, 3$ using the notation for the weights w^l_{jk} and biases b^l_j.

(5) Briefly explain how the equations for a^l_j and z^l_j in Equation 1 are derived from Equation 2.

Goal: Compute the gradient vector of a loss function L for any weights and biases. That is, we must compute

$$\frac{\partial L}{\partial w^l_{jk}} \qquad \text{and} \qquad \frac{\partial L}{\partial b^l_j}$$

for all l, j, k.

Before we do this, let's do a couple more problems and define an operation that will be useful for programming backpropagation.

(6) Compute $\dfrac{\partial z^l_j}{w^l_{jk}}$ and $\dfrac{\partial z^l_j}{b^l_j}$.

Definition. The **Hadamard product** denoted by $\odot$ is the element-wise product of two vectors of the same size. That is,

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \odot \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 \\ x_2y_2 \\ \vdots \\ x_ny_n \end{bmatrix}$$

(7) Compute $\begin{bmatrix} 0 \\ 1 \\ 5 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 2 \\ 1 \\ 9 \end{bmatrix}$

Pause to derive the following equations with the whole class for backpropagation:

$$\mathbf{d}^F = \nabla_a L \odot \sigma'\left(\mathbf{z}^F\right), \qquad d_j^F = \frac{\partial L}{\partial a_j^F}\sigma'(z_j^F)$$

$$\mathbf{d}^l = \left(\left(W^{l+1}\right)^T \mathbf{d}^{l+1}\right) \odot \sigma'\left(\mathbf{z}^l\right), \qquad d_j^l = \sum_k w_{kj}^{l+1} d_k^{l+1} \sigma'\left(z^l\right)$$

$$\frac{\partial L}{\partial w_{jk}^l} = a_k^{l-1} d_j^l$$

$$\frac{\partial L}{\partial b_j^l} = d_j^l$$

where F is the number of layers (i.e., layer F is the final layer), $d_j^l = \dfrac{\partial L}{\partial z_j^l}$, $\mathbf{d}^l = \left[d_1^l, d_2^l, \ldots, d_n^l\right]^T$, and n is the number of nodes in the l^{th} layer.

Backpropagation Algorithm

1. **Input** x**:** Set the corresponding activation $\mathbf{a}^1$ for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, F$ compute $\mathbf{z}^l = W^l \mathbf{a}^{l-1} + \mathbf{b}^l$ and $\mathbf{a}^l = \sigma\left(\mathbf{z}^l\right)$.
3. **Output** $\mathbf{d}^F$**:** Compute $\mathbf{d}^F = \nabla_a L \odot \sigma'\left(\mathbf{z}^F\right)$.
4. **Backpropagate:** For each $l = L-1, L-2, \ldots, 2$ compute $\mathbf{d}^l = \left(\left(W^{l+1}\right)^T \mathbf{d}^{l+1}\right) \odot \sigma'\left(\mathbf{z}^l\right)$.
5. **Output**: The gradient of the cost function is given by $\dfrac{\partial L}{\partial w_{jk}^l} = a_k^{l-1} d_j^l$ and $\dfrac{\partial L}{\partial b_j^l} = d_j^l$.

Total Loss: At this point, everything we've done to compute the partials $\dfrac{\partial L}{\partial w_{jk}^l}$ and $\dfrac{\partial L}{\partial b_j^l}$ for a single input $\mathbf{a}^1$. Suppose we have n training samples. Then the total loss function is given by

$$L_T = \frac{1}{n}\sum_{i=1}^n L_i = \frac{1}{n}\sum_{i=1}^n L\left(\mathbf{y}_i, \tilde{\mathbf{y}}_i\right) = \frac{1}{n}\sum_{i=1}^n L\left(\mathbf{y}_i, \mathbf{a}_i^F\right),$$

where $\mathbf{y}_i$ is the true value and $\tilde{\mathbf{y}}_i = \mathbf{a}_i^F$ is the predicted value given by the final layer for each input.

(8) Compute $\dfrac{\partial L_T}{\partial w_{jk}^l}$ and $\dfrac{\partial L_T}{\partial b_j^l}$ with respect to $\dfrac{\partial L_i}{\partial w_{jk}^l}$ and $\dfrac{\partial L_i}{\partial b_j^l}$.

Note: To perform gradient descent, we use $\dfrac{\partial L_T}{\partial w_{jk}^l}$ and $\dfrac{\partial L_T}{\partial b_j^l}$ to update the model weights and biases. To perform stochastic gradient descent, we change the total loss to the batch loss L_B which is defined similarly to total loss but for a specific batch of our training dataset instead of all of it.

(9) Let σ be the sigmoid activation function. Compute σ'.

(10) Let σ be the ReLU activation function. Compute σ'. *Hint:* Your answer should be a piecewise function.

(11) Let L be the mean squared error loss function. Compute $\nabla_a L$. That is, compute $\dfrac{\partial L}{\partial a_j^F}$ for all nodes j in the final layer.