Basic Operations of Neural Networks

Sourya Dey

1 Notation

- Scalars are written as lower case letters.
- Vectors are written as lower case bold letters, such as $\boldsymbol{x}$, and can be either row (dimensions $1 \times n$) or column (dimensions $n \times 1$). Column vectors are the default choice, unless otherwise mentioned. Individual elements are indexed by subscripts, such as x_i ($i \in \{1, \cdots, n\}$).
- Matrices are written as upper case bold letters, such as $\boldsymbol{X}$, and have dimensions $m \times n$ corresponding to m rows and n columns. Individual elements are indexed by double subscripts for row and column, such as X_{ij} ($i \in \{1, \cdots, m\}$, $j \in \{1, \cdots, n\}$).
- Bracketed superscripts are used to denote layers, for example, $W_{ij}^{(l)}$ denotes the (i, j)th element of the weight matrix of layer l.

The **derivative** of f with respect to x is $\dfrac{\partial f}{\partial x}$. Both x and f can be a scalar, vector, or matrix. The **gradient** of f w.r.t x is $\nabla_x f = \left(\dfrac{\partial f}{\partial x}\right)^T$, i.e. **gradient is transpose of derivative**. The gradient at any point x_0 in the domain has a physical interpretation, its direction is the direction of maximum increase of the function f at the point x_0, and its magnitude is the rate of increase in that direction.

2 Neural Network Operations

A neural network has $(L+1)$ layers having $\left(N^{(0)}, N^{(1)}, \cdots, N^{(L)}\right)$ neurons respectively, i.e. there are $N^{(0)}$ input and $N^{(L)}$ output neurons. Any layer l ($l \neq 0$) has a bias vector $\boldsymbol{b}^{(l)}$, an activation vector $\boldsymbol{a}^{(l)}$, and a delta (error) vector $\boldsymbol{\delta}^{(l)}$, each of dimensions $N^{(l)} \times 1$, and a weight matrix $\boldsymbol{W}^{(l)}$ preceding it of dimensions $N^{(l)} \times N^{(l-1)}$.

2.1 Feedforward

The input layer is fed an input sample vector $\boldsymbol{a}^{(0)}$ of dimensions $N^{(0)} \times 1$. Then the feedforward operation for all layers $l \in \{1, \cdots, L\}$ proceeds as:

$$\boldsymbol{s}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{1a}$$

$$\boldsymbol{a}^{(l)} = \boldsymbol{h}\left(\boldsymbol{s}^{(l)}\right) \tag{1b}$$

where $\boldsymbol{h}(\cdot)$ is the activation function. Activation functions are discussed further in Sec. 3.

The final layer output activation vector $\boldsymbol{a}^{(L)}$ is compared with the ground truth output vector for that input sample, $\boldsymbol{y}^{(L)}$, to compute a scalar-valued cost C. A popular cost function is cross-entropy:

$$C = -\sum_{i=1}^{N^{(L)}} y_i^{(L)} \ln a_i^{(L)} \tag{2}$$

2.2 Backpropagation

The goal of backpropagation is to compute the gradients of the cost w.r.t all the network parameters, i.e. $\boldsymbol{\nabla}_{\boldsymbol{W}^{(l)}} C$ and $\boldsymbol{\nabla}_{\boldsymbol{b}^{(l)}} C$ for all $l \in \{1, \cdots, L\}$. These gradients are used for updating the parameter values to make the network learn.

Since cost is directly a function of $\boldsymbol{a}^{(L)}$, we start by computing $\dfrac{\partial C}{\partial \boldsymbol{a}^{(L)}}$:

$$\frac{\partial C}{\partial \boldsymbol{a}^{(L)}} = -\begin{bmatrix} \dfrac{y_1^{(L)}}{a_1^{(L)}} & \dfrac{y_2^{(L)}}{a_2^{(L)}} & \cdots & \dfrac{y_{N^{(L)}}^{(L)}}{a_{N^{(L)}}^{(L)}} \end{bmatrix} \tag{3}$$

Then we work backwards:

$$\frac{\partial C}{\partial \boldsymbol{s}^{(L)}} = \frac{\partial C}{\partial \boldsymbol{a}^{(L)}} \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{s}^{(L)}} = \frac{\partial C}{\partial \boldsymbol{a}^{(L)}} \boldsymbol{H'}^{(L)} \tag{4}$$

where $\boldsymbol{H'}^{(L)} = \dfrac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{s}^{(L)}}$ is the matrix of activation function derivatives (described further in Sec. 3).

The delta vector of any layer is defined as $\boldsymbol{\delta}^{(l)}$ $= \boldsymbol{\nabla}_{\boldsymbol{s}^{(l)}} C$. Since gradient is transpose of the derivative, we get:

$$\boldsymbol{\delta}^{(\boldsymbol{L})} = \boldsymbol{\nabla}_{\boldsymbol{s}^{(L)}} C = \boldsymbol{H'}^{(L)^T} \boldsymbol{\nabla}_{\boldsymbol{a}^{(L)}} C \tag{5}$$

Continuing the chain rule, we get:

$$\frac{\partial C}{\partial \boldsymbol{b}^{(L)}} = \frac{\partial C}{\partial \boldsymbol{s}^{(L)}} \frac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{b}^{(L)}} \tag{6}$$

From (1a), $\dfrac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{b}^{(L)}} = I$, the identity matrix. Thus, $\dfrac{\partial C}{\partial \boldsymbol{b}^{(L)}} = \dfrac{\partial C}{\partial \boldsymbol{s}^{(L)}}$, transposing which we get our first important result:

$$\boldsymbol{\nabla}_{\boldsymbol{b}^{(L)}} C = \boldsymbol{\delta}^{(L)} \tag{7}$$

which is a vector of the same dimensions as $\boldsymbol{b}^{(L)}$, i.e. $\big(N^{(L)} \times 1\big)$.

Again from the chain rule, we get:

$$\frac{\partial C}{\partial \boldsymbol{W}^{(L)}} = \frac{\partial C}{\partial \boldsymbol{s}^{(L)}} \frac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{W}^{(L)}} \tag{8}$$

The quantity $\dfrac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{W}^{(L)}}$ is a 3rd order tensor of dimensions $\left(N^{(L)} \times N^{(L-1)} \times N^{(L)}\right)$, i.e. an outer matrix of dimensions $\left(N^{(L-1)} \times N^{(L)}\right)$ with each element being an $N^{(L)}$-dimensional column vector. Using (1a) for $l = L$, note that:

$$s_i^{(L)} = W_{i,1}^{(L)} a_1^{(L-1)} + W_{i,2}^{(L)} a_2^{(L-1)} + \cdots + W_{i,N^{(L-1)}}^{(L)} a_{N^{(L-1)}}^{(L-1)} \tag{9}$$

So the $N^{(L)}$-dimensional column vector at $\left(\dfrac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{W}^{(L)}}\right)_{jk}$ has zeroes everywhere except for location k having value $a_j^{(L-1)}$. This is visualized below for the simple case $N^{(L)} = 2$, $N^{(L-1)} = 3$:

$$\frac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{W}^{(L)}} = \begin{bmatrix} \begin{bmatrix} a_1^{(L-1)} \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ a_1^{(L-1)} \end{bmatrix} \\ \\ \begin{bmatrix} a_2^{(L-1)} \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ a_2^{(L-1)} \end{bmatrix} \\ \\ \begin{bmatrix} a_3^{(L-1)} \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ a_3^{(L-1)} \end{bmatrix} \end{bmatrix} \tag{10}$$

Returning to (8), the (j,k)th element of $\dfrac{\partial C}{\partial \boldsymbol{W}^{(L)}}$ is the inner product of the row vector $\dfrac{\partial C}{\partial \boldsymbol{s}^{(L)}}$ with the column vector at $\left(\dfrac{\partial \boldsymbol{s}^{(L)}}{\partial \boldsymbol{W}^{(L)}}\right)_{jk}$. This inner product is $a_j^{(L-1)} \delta_k^{(L)}$, so $\dfrac{\partial C}{\partial \boldsymbol{W}^{(L)}} = \boldsymbol{a}^{(L-1)} \boldsymbol{\delta}^{(L)^T}$. Transposing this gives our second important result:

$$\boldsymbol{\nabla}_{\boldsymbol{W}^{(L)}} C = \boldsymbol{\delta}^{(L)} \boldsymbol{a}^{(L-1)^T} \tag{11}$$

which is a matrix of the same dimensions as $\boldsymbol{W}^{(L)}$, i.e. $\left(N^{(L)} \times N^{(L-1)}\right)$.

Thus, we have obtained the gradients of cost w.r.t $\boldsymbol{W}$ and $\boldsymbol{b}$ of the output layer from its $\boldsymbol{\delta}$ vector. The only remaining task is to obtain $\boldsymbol{\delta}^{(l)}$ from $\boldsymbol{\delta}^{(l+1)}$, i.e. get the delta vector for each layer from that of its next. Using the wonderful chain rule again, we get:

$$\frac{\partial C}{\partial \boldsymbol{s}^{(l)}} = \frac{\partial C}{\partial \boldsymbol{s}^{(l+1)}} \frac{\partial \boldsymbol{s}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{s}^{(l)}} \tag{12}$$

From (1a), $\dfrac{\partial \boldsymbol{s}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} = \boldsymbol{W}^{(l+1)}$, and we have already established that $\dfrac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{s}^{(l)}} = \boldsymbol{H'}^{(l)}$, i.e. the matrix of derivatives of the activation function in layer l (see (17)). Combining all this and transposing to get gradient, we get our 3rd and final important result:

$$\boldsymbol{\delta^{(l)}} = \boldsymbol{H'}^{(l)^T} \boldsymbol{W}^{(l+1)^T} \boldsymbol{\delta^{(l+1)}} \tag{13}$$

That's it!

In summary, backpropagation proceeds by computing the gradient of cost w.r.t the following:

1. Output layer: $\boldsymbol{\delta^{(L)}} = \boldsymbol{H'}^{(L)^T} \boldsymbol{\nabla}_{\boldsymbol{a}^{(L)}} C$
2. Intermediate layers: $\boldsymbol{\delta^{(l)}} = \boldsymbol{H'}^{(l)^T} \boldsymbol{W}^{(l+1)^T} \boldsymbol{\delta^{(l+1)}} \qquad \forall l \in \{1, \cdots, L-1\}$
3. All biases: $\boldsymbol{\nabla}_{\boldsymbol{b}^{(l)}} C = \boldsymbol{\delta}^{(l)} \qquad \forall l \in \{1, \cdots, L\}$
4. All weights: $\boldsymbol{\nabla}_{\boldsymbol{W}^{(l)}} C = \boldsymbol{\delta}^{(L)} \boldsymbol{a}^{(L-1)^T} \qquad \forall l \in \{1, \cdots, L\}$

If you do not want to treat the biases separately, you can augment the weight matrix $\boldsymbol{W}^{(l)}$ with a column for $\boldsymbol{b}^{(l)}$ and augment the activation vector $\boldsymbol{a}^{(l-1)}$ with a single 1. Then backprop step 4 takes care of step 3 automatically. Think about it.

2.3 Update

Note that the gradient of cost w.r.t any parameter $\boldsymbol{p}$ is of the same dimensions as $\boldsymbol{p}$ ($\boldsymbol{p}$ could be any weight matrix, bias vector, or any other parameter which the network needs to learn). Now we can apply the gradient descent update rule:

$$\boldsymbol{p} \leftarrow \boldsymbol{p} - \eta \boldsymbol{\nabla}_{\boldsymbol{p}} C \tag{14}$$

where η is the learning rate. This takes $\boldsymbol{p}$ in the direction opposite to the gradient, i.e. in the direction of maximum decrease of C.

In practice, we do not update after every input, but perform feedforward and backpropagation on several inputs before doing a single averaged update. The number of inputs between two updates is the *minibatch size* M, i.e.:

$$\boldsymbol{p} \leftarrow \boldsymbol{p} - \frac{\eta}{M} \sum_{i=1}^{M} \left(\boldsymbol{\nabla}_{\boldsymbol{p}} C\right)^{[i]} \tag{15}$$

where $\left(\boldsymbol{\nabla}_{\boldsymbol{p}} C\right)^{[i]}$ is the gradient for input sample i.

3 Different Activation Functions

For simplicity, we discard the layer superscript (l) in this section. Recall that the activation operation in any layer with N neurons is $\boldsymbol{a} = \boldsymbol{h}(\boldsymbol{s})$, where $\boldsymbol{a}$ and $\boldsymbol{s}$ are N-dimensional column vectors.

3.1 Hidden layers

For most hidden layers, $\boldsymbol{h}$ will be a vectorized scalar function applied element-wise to $\boldsymbol{s}$. This is denoted as $\underline{h}$, i.e.:

$$\boldsymbol{a} = \underline{h}\left(\begin{bmatrix} s_1 \\ \vdots \\ s_N \end{bmatrix}\right) = \begin{bmatrix} h\left(s_1\right) \\ \vdots \\ h\left(s_N\right) \end{bmatrix} \tag{16}$$

In this case, the derivative $\boldsymbol{H'} = \dfrac{\partial \boldsymbol{a}}{\partial \boldsymbol{s}}$ will be a $N \times N$ diagonal matrix:

$$\boldsymbol{H'} = \begin{bmatrix} h'\left(s_1\right) & & \text{\Large 0} \\ & \ddots & \\ \text{\Large 0} & & h'\left(s_N\right) \end{bmatrix} \tag{17}$$

3.2 Output layer

For the output layer, we generally use **softmax activation**, where every element in the output vector is influenced by all elements of the input vector:

$$\boldsymbol{a} = \begin{bmatrix} \frac{e^{s_1}}{\sum_{i=1}^{N} e^{s_i}} \\ \vdots \\ \frac{e^{s_N}}{\sum_{i=1}^{N} e^{s_i}} \end{bmatrix} = \begin{bmatrix} \frac{k_1}{k} \\ \vdots \\ \frac{k_N}{k} \end{bmatrix} \tag{18}$$

where $k_i = e^{s_i}$ and $k = \sum_{i=1}^{N} e^{s_i}$.

Using simple calculus gives the diagonal entries of $\boldsymbol{H'}$ as $\frac{\partial a_i}{\partial s_i} = \frac{k_i\,(k - k_i)}{k^2}$, and the off-diagonal entries as $\frac{\partial a_i}{\partial s_j} = \frac{\partial a_j}{\partial s_i} = -\frac{k_i k_j}{k^2}$. Thus, $\boldsymbol{H'}$ is a symmetric matrix.

As shown in (3), using cross-entropy cost results in $\frac{\partial C}{\partial \boldsymbol{a}} = -\begin{bmatrix} \frac{y_1}{a_1} & \cdots & \frac{y_N}{a_N} \end{bmatrix}$. Then the delta vector of the output layer becomes:

$$\boldsymbol{\delta} = \boldsymbol{H'} \nabla_{\boldsymbol{a}} C = \begin{bmatrix} \left(a_1 \sum_{i=1}^{N} y_i\right) - y_1 \\ \vdots \\ \left(a_N \sum_{i=1}^{N} y_i\right) - y_N \end{bmatrix} \tag{19}$$

This has a particularly simple interpretation when the outputs are one-hot. This means that if the correct class is n, then $y_n = 1$ and $y_m = 0 \quad \forall m \neq n$. In this case, $\delta_n = a_n - 1$ and $\delta_m = a_m \quad \forall m \neq n$. This means that $\boldsymbol{\delta} = \boldsymbol{a} - \boldsymbol{y}$, which makes perfect sense as the ‘error vector’.