Neural Network Implementation Draft

October 2019

1 Matrix Calculus

1.1 Chain Rule for Matrix Calculus

The chain rule for a vectors is similar to the chain rule for scalars. Except the order is important. For $\mathbf{z} = f(\mathbf{y})$ and $\mathbf{y} = g(\mathbf{x})$ the chain rule is:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \tag{1}$$

2 Example: 3 Layer Fully Connected Neural Network

For the input x the neural network which is described by its weights W, its biases b and the activation functions $g(t)$. The network has L_1 neurons in the first layer, L_2 neurons in the second layer and L_3 neurons in the final layer.

y	$\frac{\partial}{\partial x} y$
Ax	A^T
$x^T A$	A
$x^T x$	$2x$
$x^T A x$	$Ax + A^T x$

Table 1: Useful derivatives equations

Layer	Weights	Bias
1	[L1 nx]	[L1 1]
2	[L2 L1]	[L2 1]
3	[ny L2]	[ny 1]

Table 2: Dimensions of the weight and bias matrices

$$\begin{aligned}
z_1 &= W_1 x + b_1 \\
a_1 &= f(z_1) \\
z_2 &= W_2 a_1 + b_2 \\
a_2 &= f(z_2) \\
z_3 &= W_3 a_2 + b_3 \\
h &= z_3
\end{aligned}$$

$$J = \frac{1}{N} \sum_{i}^{N} (h(x_i; W, b) - y_i)^2 \tag{2}$$

2.1 Backpropagation

The update rule for gradient descent is

$$p_{i+1} = p_i + \mu \frac{\partial J}{\partial p_i} \quad \forall p \in \{W, b\} \tag{3}$$

The main difficulty here is to calculate the gradient for each parameter in the network, which can easily be several thousands or even million parameters. Here back-propagation is used to efficiently calculate those derivatives. The first step is to differentiate the cost function with respect to an parameter p which can describe an weight or a bias

$$\frac{\partial J}{\partial p_i} = \frac{2}{N} \sum_{i}^{N} (h(x_i; W, b) - y_i) \frac{\partial h}{\partial p_i} \tag{4}$$

The total list of needed derivatives are:

The order is not correct yet

$$
\begin{aligned}
\frac{\partial h}{\partial W_3} &= \frac{\partial h}{\partial z_3}\frac{\partial z_3}{\partial W_3} = a_2 \\
\frac{\partial h}{\partial W_2} &= \frac{\partial h}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial W_2} = W_3^T f'(z_2) a_2 \\
\frac{\partial h}{\partial W_1} &= \frac{\partial h}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial W_1} = W_2^T W_3^T f'(z_2) f'(z_1) x \\
\frac{\partial h}{\partial b_3} &= \frac{\partial h}{\partial z_3}\frac{\partial z_3}{\partial b_3} = 1 \\
\frac{\partial h}{\partial b_2} &= \frac{\partial h}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial b_2} = W_3 f'(z_2) \\
\frac{\partial h}{\partial b_1} &= \frac{\partial h}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial a_1}{\partial b_1} = W_2^T W_3^T f'(z_2) f'(z_1)
\end{aligned}
$$

Note: Here the equations are used as scalars. Because those are vectors all equations need to be transposed.

Calculations of the delta terms is as follows

$$\delta^{(4)} = h - y \tag{5}$$

then the next delta value is computed using the update equation

$$\delta^{(l-1)} = W_i^T \delta^{(l)} . * f'(z) \tag{6}$$

and then the total update term is calculated:

$$\Delta^{(l)} = \delta^{(l)} a^{(l)} \tag{7}$$

It can be seen that the same derivatives are used more often

Here something to note here: $f'(z_i)$ is the derivative of the activation function with respect to z_i. As an example, the equation of the ReLU activation function is:

$$f(t) = \begin{cases} t & \text{if} \quad t > 0 \\ 0 & \text{else} \end{cases} \tag{8}$$

The derivative $f'(t)$ is

$$f'(t) = \begin{cases} 1 & \text{if} \quad t > 0 \\ 0 & \text{else} \end{cases} \tag{9}$$

Derivative	Result
$\partial z_1 / \partial W_1$	x
$\partial z_1 / \partial b_1$	1
$\partial a_1 / \partial z_1$	$f'(z_1)$
$\partial z_2 / \partial a_1$	W_2
$\partial z_2 / \partial W_2$	a_1
$\partial z_2 / \partial b_2$	1
$\partial a_2 / \partial z_2$	$f'(z_2)$
$\partial z_3 / \partial a_2$	W_3
$\partial z_3 / \partial W_3$	a_2
$\partial z_3 / \partial b_3$	1

Table 3: Calculations of all derivatives of the network

3 Example: Convolutional Neural Network

The two dimensional convolution is defined as:

$$z(i,j) = (f * g)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m,n)g(m-i,n-j) \tag{10}$$

Compared to the previous example the input data is now not one dimensional but two-dimensional.

Also because the problem tackled here is a classification one, the loss function used is the cross-entropy function:

$$J = -y. * \log(h) + (1-y). * \log(1-h) \tag{11}$$

where $.*$ is used as the element-wise multiplication. This can be interpreted as:

$$J = \begin{cases} -y_i * \log(h_i) & \text{if} \quad y_i = 1 \\ (1-y_i) * \log(h_i) & \text{if} \quad y_i = 0 \end{cases} \tag{12}$$

Because each image is exclusively in one class, the vector

$$y = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^T \tag{13}$$

consists of all zeros except for a single 1.