RNNLIB: Softmax Layer

2015-02-05 21:08:17 +0800

- list element with functor item {:toc}

link name

I used to think that, in order to get the proper gradient, we have to take derivative of log of softmax with respect to weights. However, the RNNLIB shows that we can actually factorize the network into single layers. In this post, we look into the Softmax Layer.

Fundamentals

List of Symbols

Symbol	Meaning
J	cost function
y_k	activation of a neon
u_k	input of a neon
$S_i(\mathbf{u})$	softmax function, ith value for a vector $\mathbf{u}$

Formulas

Softmax of a vector $\mathbf{u}$ is defined as,

$$S_i(\mathbf{u}) = \frac{e^{u_i}}{\sum_k e^{u_k}} = y_i \tag{1}$$

the derivative of softmax is,

$$\frac{\partial S_i(\mathbf{u})}{\partial u_j} = \frac{\partial y_i}{\partial u_j} = \begin{cases} y_i(1-y_i) & i = j \\ -y_i y_j & i \neq j \end{cases} \tag{2}$$

Layers in RNNLIB

Every layer in RNNLIB consists of input and output sides, both sides contain activations and errors. Their relations with terms in math are shown in following table,

Variable	Term
inputActivations	u_k
outputActivations	y_k
inputErrors	$\frac{\partial J}{\partial u_k}$
outputErrors	$\frac{\partial J}{\partial y_k}$

Forward Pass

Forward pass computes y_k from u_k using equation (1). There is a trick in the code, we can call it the *safe* softmax.

To understand it, consider dividing both numerator and denominator by e^c in equation (1),

$$S_i(\mathbf{u}) = \frac{\frac{e^{u_i}}{e^c}}{\frac{\sum_k e^{u_k}}{e^c}} = \frac{e^{u_i - c}}{\sum_k e^{u_k - c}} = S_i(\hat{\mathbf{u}}) \tag{3}$$

thus, in order to avoid overflow when calculating exponentials[1], we can replace u_k with $\hat{u}_k = u_k - c$. Typically, c is set to u_{max}.

In RNNLIB,

$$c = \frac{u_{max} + u_{min}}{2}$$

.

Backpropagating

Backpropagation computes $\frac{\partial J}{\partial u_k}$ from $\frac{\partial J}{\partial y_k}$.

In RNNLIB, the result is

$$\frac{\partial J}{\partial u_j} = y_j(\frac{\partial J}{\partial y_j} - \langle \mathbf{y}, \frac{\partial J}{\partial \mathbf{y}} \rangle) \tag{4}$$

[1]Strictly speaking, this converts overflow into underflow. Underflow is no problem, because that rounds off to zero, which is a well-behaved floating point number. otherwise, it will be Infinity or NaN. see this article for details.

where, $\langle \cdot , \cdot \rangle$ denotes inner product.

To get the above equation, we first notice that variations in u_j give rise to variations in the error function J through variations in all y_ks. Thus, according to the Multivariable Chain Rules, we can write,

$$\frac{\partial J}{\partial u_j} = \sum_k \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial u_j} \tag{5}$$

Using equation (2) to replace $\frac{\partial y_k}{\partial u_j}$, we get,

$$\begin{aligned} \frac{\partial J}{\partial u_j} &= y_j(1 - y_j)\frac{\partial J}{\partial y_j} + \sum_{k:k\neq j} -y_k y_j \frac{\partial J}{\partial y_k} \\ &= y_j(\frac{\partial J}{\partial y_j} - y_j \frac{\partial J}{\partial y_j} + \sum_{k:k\neq j} -y_k \frac{\partial J}{\partial y_k}) \\ &= y_j(\frac{\partial J}{\partial y_j} - \sum_k y_k \frac{\partial J}{\partial y_k}) \\ &= y_j(\frac{\partial J}{\partial y_j} - \langle \mathbf{y}, \frac{\partial J}{\partial \mathbf{y}} \rangle) \end{aligned} \tag{6}$$

Finally, we reach equation (4).

In this way, softmax operation can be implemented to be a standalone layer.