MOTR JOURNAL

CHENG TAI

Convolutional Neural Networks

This describes the details of the back projection applied to convolutional neural networks. Some subtitles of implementation using MATLAB are also given.

The model of convolutional nets is roughly the following:

$$\begin{aligned} x^0 &:= \text{input} \\ x_j^{l+1} &:= f(\sum_{i \in M_j^{l+1}} x_k^l * a_{ij}^{l+1} + b_j^{l+1}), \quad l = 1, \cdots, L-1 \\ y^0 &:= \bigoplus u_j^L \\ y^{l+1} &:= \sigma(W^{l+1} y^l + c^{l+1}), \quad l = 1, \cdots, M-1 \\ E &:= \frac{1}{2} \|y^M - e\|_2^2 \end{aligned}$$

To apply back propagation, we need to compute the partial derivatives. For a mapping f from $\mathbb{R}^n$ to $\mathbb{R}^m$, the derivative is defined as:

$$(\frac{\partial f}{\partial x})_{ij} = \frac{\partial f_i}{\partial x_j}$$

With this definition, the derivative of a scalar function of n variables is a $1 \times n$ row vector instead of a $n \times 1$ column vector. Taking derivative with respect to y is easy:

$$\frac{\partial E}{\partial y^l} = \frac{\partial E}{\partial y^{l+1}} \cdot \sigma'(y^{l+1}) \cdot W^{l+1}$$

In MATLAB implementations, the derivative is usually represented as a column vector, so we take the transpose of the above expression in numerical implementations.

In this way, we have computed the derivatives with respect to all the ys, define

$$\delta_j^l := \frac{\partial E}{\partial x_j^l},$$

then

$$\begin{aligned} \frac{\partial E}{\partial x_i^{l+1}} &= \sum_j \frac{\partial E}{\partial x_j^{l+1}} \cdot \frac{\partial x_j^{l+1}}{x_i^l} \\ &= \delta_j^{l+1} \cdot \partial(x_i^l * a_{ij}^{l+1} + b_j{}^l + 1) x_i^l \end{aligned}$$

If we know how to take derivative of the convolution function, then the problem is solved. Now let us focus on the convolution. Let $f_a : x \mapsto x * a$ be the convolution subject to the boundary condition "valid". Because it is a linear operator, hence we can define:

$$W_a x := x * a, \text{'valid'}$$

Hence we have

$$\partial(x_i^l * a_{ij}^{l+1} + b_j{}^l + 1)x_i^l = W_{a_{ij}^{l+1}}$$

and

$$\frac{\partial E}{\partial x_i^l} = \delta_j^{l+1} \cdot W_{a_{ij}^{l+1}}$$

In MATLAB, taking the transpose, the above expression can be conveniently written as

$$\frac{\partial E}{\partial x_i^l} = \delta_j^{l+1} * a_{ij}^{l+1}(-\cdot), \text{'full'}$$

(That is, the transpose of a convolution with kernel a is still a convolution with the kernel flipped, and the boundary condition 'valid' becomes 'full').

The final step is to compute $\frac{\partial E}{\partial a_{ij}^l}$. Let us focus on this and with out loss of generality, we deal with the 1D case, extension to higher dimensions will be given in the end. Assume x is the signal of length n, a is the filter of length l, d is a vector of length $n - l + 1$, define

$$f := x * a, \text{'valid'}$$

That is

$$f(i) = \sum_{k=0}^{l-1} x(i+k)a(l-k) = \sum_{j=l}^{1} x(i+l-j)a(j)$$

We have

$$\left(\frac{\partial f}{\partial a}\right)_{ij} = x(i+l-j)$$

and

$$\begin{aligned} (d \cdot \frac{\partial f}{\partial a})_j &= \sum_i d_i \cdot (\frac{\partial f}{\partial a})_{ij} \\ &= \sum_{i=1}^{n-l+1} d(i)x(i+l-j) \end{aligned}$$

On the other hand,

$$\begin{aligned} (\bar{x} * d, \text{'valid'})_j &= \sum_{i=0}^{n-l} x(n+1-j-i)d(n-l+1-i) \\ &= \sum_{i=n-1+1}^{1} x(l-j+i)d(i) \\ &= (d \cdot \frac{\partial f}{\partial a})_j \end{aligned}$$

Therefore,

$$\frac{\partial E}{\partial a_{ij}^{l+1}} = \bar{x}_i^l * \delta_j^{l+1}, \text{'valid'}$$

For 2D signals, the derivation is similar. The transpose of a 2D convolution with 'valid' boundary condition is still a convolution with the kernel flipped in both dimensions with 'full' boundary condition.

In MATLAB implementations, we usually stack the batch of examples in the third dimension, in which case, we just compute the derivative example by example and sum the derivatives over the third dimensions. In MATLAB, this can be succinctly written in one line

$$\frac{\partial E}{\partial a_{ij}^{l+1}} = \text{convn(flipall}(x_i^l) \text{ , } \delta_j^{l+1}\text{,'valid')}$$

Code Design

Data must be created using shared_ptr. Then the layers can get input from this.

Think about what a layer and a convnet should be