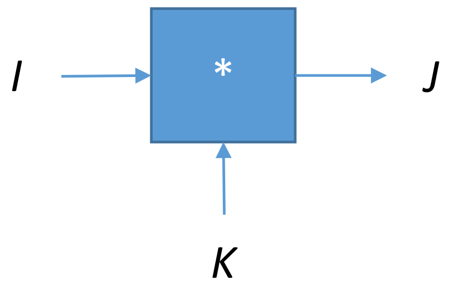
Training Convnet with Backprop

Nilanjan Ray

Backpropagation (BP) for a conv layer



$$J = I * K$$

Input to convolution layer: *I*, a *H*-by-*W* matrix

Parameter of the layer: *K*, a *h*-by-*w* matrix

Output of the layer: *J*, a (*H*-*h*+1)-by-(*W*-*w*+1) matrix

Assume: *H* >= *h* and *W* >= *w*

Given the gradient of loss function δJ with respect to *J*, BP tries to find answers to the following:

(1) What is the gradient of the loss function with respect to *K*? Denote this gradient by δK.

(2) What is the gradient of the loss function with respect to *I*? Denote this gradient by δI.

Why do we need δK? Because, we want to adjust the parameter *K* by gradient descent: *K* = *K* – (learning rate)δK

Why do we need δI? Because, we want to apply BP to the layer that precedes this conv layer.

Derivation of δK

$$J(i,j) = \sum_{l=1}^{h}\sum_{m=1}^{w} I(i+l-1, j+m-1)K(l,m) \quad \Rightarrow \quad \frac{\partial J(i,j)}{\partial K(p,q)} = I(i+p-1, j+q-1)$$

Using chain rule of derivative:

$$\delta K(p,q) = \sum_{i=1}^{H-h+1}\sum_{j=1}^{W-w+1} \frac{\partial J(i,j)}{\partial K(p,q)} \delta J(i,j) = \sum_{i=1}^{H-h+1}\sum_{j=1}^{W-w+1} I(i+p-1, j+q-1)\delta J(i,j)$$

Thus, $\boxed{\delta K = I * \delta J}$

Derivation of δI

$$J(i,j) = \sum_{l=1}^{h} \sum_{m=1}^{w} I(i+l-1, j+m-1) K(l,m) \qquad \Longrightarrow$$

$$\frac{\partial J(i,j)}{\partial I(p,q)} = \begin{cases} K(p-i+1, q-j+1), & \text{if } 0 \le p-i \le h-1 \text{ and } 0 \le q-j \le w-1, \\ 0, & \text{otherwise.} \end{cases}$$

Using chain rule of derivative:

$$\delta I(p,q) = \sum_{i=1}^{H-h+1} \sum_{j=1}^{W-w+1} \frac{\partial J(i,j)}{\partial I(p,q)} \delta J(i,j) = \sum_{i=\max(1,p-h+1)}^{\min(p,H-h+1)} \sum_{j=\max(1,q-w+1)}^{\min(q,W-w+1)} K(p-i+1, q-j+1) \delta J(i,j)$$

$$= \sum_{l=\max(1,p+h-H)}^{\min(p,h)} \sum_{\max(1,q+w-W)}^{\min(q,w)} K(l,m) \delta J(p-l+1, q-m+1)$$

Thus, $\boxed{\delta I = \text{pad}(\delta J) * \text{flip}(K)}$

"pad" function adds (*h*-1) 0 rows at the top and bottom and also adds (*w*-1) 0 columns at the left and at the right of a matrix.

Size of pad(δJ) is (*H*+*h*-1)-by-(*W*+*w*-1).

flip(*K*) is best understood by an example:

$$K = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \qquad \text{flip}(K) = \begin{bmatrix} 6 & 4 & 2 \\ 5 & 3 & 1 \end{bmatrix}$$

BP for a max pooling layer

$$\begin{tikzcd}
\{x_i\}_{i=1}^n \arrow[r] & \boxed{\text{Max pool}} \arrow[r] & y = \max(\{x_i\}_{i=1}^n)
\end{tikzcd}$$

Note that in case of a tie, only a single index is chosen for the following operation:

$$i = \text{argmax}_k \{x_k\}_{k=1}^n$$

By chain rule: $\delta x_i = \dfrac{\partial y}{\partial x_i} \delta y = \begin{cases} \delta y, & \text{if} \quad i = \text{argmax}_k \{x_k\}_{k=1}^n, \\ 0, & \text{otherwise.} \end{cases}$

Example of a 2-by-2, stride 2 max pooling:

$$\begin{tikzcd}
x = \begin{bmatrix} 2 & -3 & 6 & 5 \\ 4 & 5 & 6 & -7 \\ 0 & -1 & 3 & -4 \\ -2 & -6 & 6 & 8 \end{bmatrix} \arrow[r] & \boxed{\text{Max pool}} \arrow[r] & y = \begin{bmatrix} 5 & 6 \\ 0 & 8 \end{bmatrix}
\end{tikzcd}$$

Suppose, $\delta y = \begin{bmatrix} \delta y_1 & \delta y_3 \\ \delta y_2 & \delta y_4 \end{bmatrix}$,

then, $\delta x = \begin{bmatrix} 0 & 0 & \delta y_3 & 0 \\ 0 & \delta y_1 & 0 & 0 \\ \delta y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \delta y_4 \end{bmatrix}$.

BP for a ReLU layer

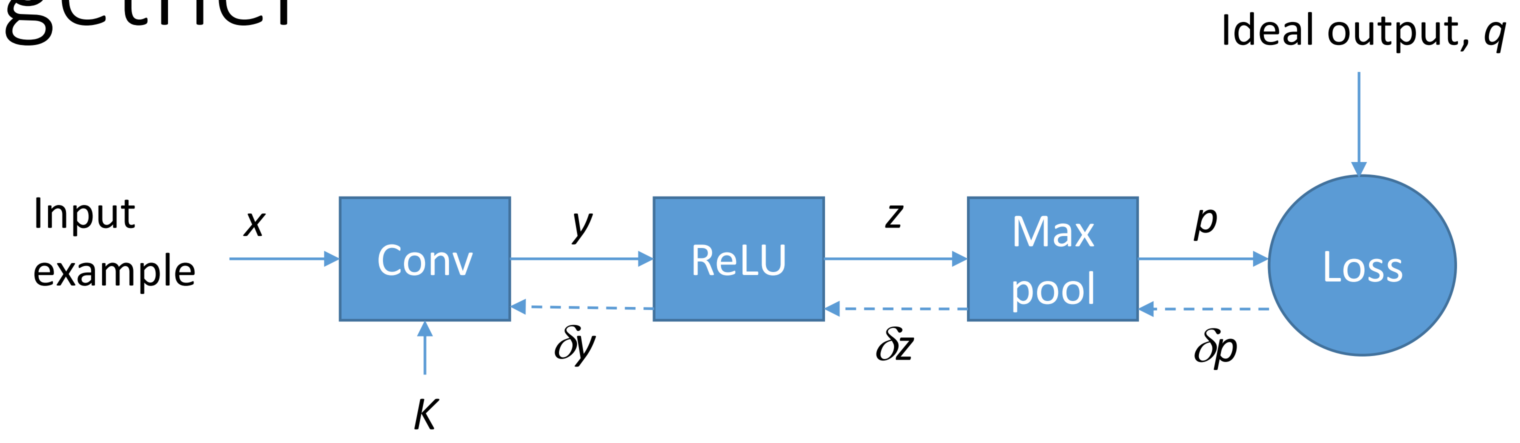
$$\begin{tikzcd} {} \arrow[r, "x"] & \boxed{\text{ReLU}} \arrow[r, "y = \max(0, x)"] & {} \end{tikzcd}$$

By chain rule: $\delta x = \dfrac{\partial y}{\partial x}\delta y = \begin{cases} \delta y, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases}$

Example: $$\begin{tikzcd} x = \begin{bmatrix} -5 & 4 \\ 0 & 2 \end{bmatrix} \arrow[r] & \boxed{\text{ReLU}} \arrow[r] & y = \begin{bmatrix} 0 & 4 \\ 0 & 2 \end{bmatrix} \end{tikzcd}$$

Suppose, $\delta y = \begin{bmatrix} \delta y_1 & \delta y_3 \\ \delta y_2 & \delta y_4 \end{bmatrix}$, then $\delta x = \begin{bmatrix} 0 & \delta y_3 \\ 0 & \delta y_4 \end{bmatrix}$.

Putting it all together



An example network with 3 layers and a loss function

<u>Convnet training algorithm:</u>

```
Initialize parameter $K$.
Iterate:
    Step 1: (Forward pass)
        Step 1a: Randomly choose a training example $x$ and its corresponding ideal output $q$.
        Step 1b: Pass $x$ through “Conv” to get $y$; pass $y$ though ReLU to get $z$; pass $z$ through Max pool to get $p$.
    Step 2: Compute “Loss” function for diagnostic purposes. /* Loss function measures deviation of $p$ from $q$. */
    Step 3: (Backward pass aka backpropagation)
        Step 3a: Compute gradient of Loss function with respect to $p$. Denote this gradient by $\delta p$.
        Step 3b: Compute $\delta z$ given $\delta p$.        /* Look at “BP for Max pooling.” */
        Step 3c: Compute $\delta y$ given $\delta z$.        /* Look at “BP for ReLU.” */
        Step 4c: Compute $\delta K$ given $\delta y$.        /* Look at “BP for Conv.” */
    Step 4: (Update parameter $K$ by gradient descent) $K = K –$ (learning rate) $\delta K$.
```

Note: We don’t have to compute δx, because there is no preceding layer before conv in the example above.