Convolutional Neural Networks: An algorithmic explanation in pseudocode

MPCS 53111 - Machine Learning

May 2017

1 Convolution Operator

Consider the Mini-Batch $\mathcal{B}$ of B examples $\mathcal{B} = \{\mathcal{E}_1, .., \mathcal{E}_B\}$. Each example is represented by a **3D tensor** $\mathcal{E} \in \Re^{H \times W \times D}$ where W and H are the width and height of the representation matrix, and D is the depth (or number of chanels). For example, a color image has depth 3 or 3 channels of red, green and blue. Combined all the $\mathcal{B}$ images together, we have a **4D tensor** $\mathcal{L} \in \Re^{B \times H \times W \times D}$.

Given the set of filters $\mathcal{F} = \{f_1, f_2, ..\}$ where each filter $f_i \in \Re^{h \times w}$. The set of filters $\mathcal{F}$ maps the **4D tensor** $\mathcal{L}$ of D channels into another **4D tensor** $\mathcal{L}' \in \Re^{B \times H' \times W' \times D'}$ of D' channels. The set of filters can be represented by a **4D tensor** $\mathcal{F} \in \Re^{h \times w \times D \times D'}$.

Given the stride $s \in \mathbb{N}$ as the number of pixels we move a filter horizontally and vertically. The following pseudocode describes the Convolution of a CNN:

Remark: Here is **only** the pseudocode explaining the algorithm. In practice, one needs matrix operations (i.e., Numpy with Python) to execute it efficiently.

function Forward-Pass($\mathcal{L} \in \Re^{B \times H \times W \times D'}$, $\mathcal{F} \in \Re^{h \times w \times D \times D'}$, $s \in \mathbb{N}$):
    $W' \leftarrow (W - w)/s + 1$
    $H' \leftarrow (H - h)/s + 1$
    Initialize tensor $\mathcal{L}' \in \Re^{B \times H' \times W' \times D'}$
    for each $d = 1 \to D$:
        for each $d' = 1 \to D'$:
            Consider filter $f = \mathcal{F}[:, :, d, d'] \in \Re^{h \times w}$ mapping from chanel d to d'
            for each image $b = 1 \to B$:
                $\mathcal{L}'[b, :, :, d'] \leftarrow$ Convolve($\mathcal{L}[b, :, :, d]$, f, s)
            end for
        end for
    end for
    return $\mathcal{L}'$
end function

function Convolve($I \in \Re^{H \times W}$, $f \in \Re^{h \times w}$, $s \in \mathbb{N}$):
    Initialize $I' \in \Re^{H' \times W'}$
    for $x = 0 \rightarrow H' - 1$:
        for $y = 0 \rightarrow W' - 1$:
            $I'[x, y] \leftarrow$ InnerProduct($I[x * s : x * s + h, y * s : y * s + h]$, f)
        end for
    end for
    return $\mathcal{I}'$
end function

function InnerProduct($\mathcal{I}, f \in \Re^{h \times w}$)
    Let $d \leftarrow h \times w$
    Reshape $\mathcal{I}$ to $d \times 1$
    Reshape f to $d \times 1$
    return $\langle \mathcal{I}, f \rangle$ or $f^T \mathcal{I}$
end function

2 Convolution Gradient Computation

function Backward-Pass($\mathcal{L}'.\delta \in \Re^{B \times H' \times W' \times D'}$):
    for each $d' = 1 \rightarrow D'$:
        for each $d = 1 \rightarrow D$:
            $f \leftarrow \mathcal{F}[:, :, d, d']$
            for each $b = 1 \rightarrow B$:
                $I'.\delta \leftarrow \mathcal{L}'.\delta[b, :, :, d']$
                $I \leftarrow \mathcal{L}[b, :, :, d]$
                $\mathcal{L}.\delta[b, :, :, d] \leftarrow \mathcal{L}.\delta[b, :, :, d]+$ Convolve-Gradient-$\mathcal{L}(I'.\delta, f, s)$
                $\mathcal{F}.\delta[:, :, d, d'] \leftarrow \mathcal{F}.\delta[:, :, d, d']+$ Convolve-Gradient-$f(I'.\delta, I, s)$
            end for
        end for
    end for
end function

function Convolve-Gradient-$\mathcal{L}$($I'.\delta \in \Re^{H' \times W'}, f \in \Re^{h \times w}, s$):
    Padded_I' = Padding($I.\delta \in \Re^{H \times W}, 2 * h, 2 * w$)
    Transpose the filter: $f_{trans} = f^T$
    $I.\delta$ = Convolve(Padded_I', f_{trans})
    return $I.\delta$
end function

function Convolve-Gradient-f($I'.\delta \in \Re^{H' \times W'}, I \in \Re^{H \times W}, s$):
    $f.\delta$ = Convolve($I, I'.\delta$)
    return $f.\delta$
end function

3 Padding Operation

Suppose that the stride $s = 1$, that means we move only the filter one pixel at a time (for simplicity). The original image size is $H \times W$. The filter size is $h \times w$. The output image after convolution operator **shrinks** to $(H-h+1)\times(W-w+1)$. In practice, the size of filter is **always** odd. We want to keep the output size the same as the input size. We pad around the original image with a thick border of zeros: $\lfloor h/2 \rfloor$ on the left and on the right, $\lfloor w/2 \rfloor$ on the top and on the bottom. The idea is presented by the following pseudocode:

function Padding($\mathcal{L} \in \Re^{B\times H\times W\times D}$, $w \in \mathbb{N}_{odd}$, $h \in \mathbb{N}_{odd}$)
    Let $W' \leftarrow W + w$
    Let $H' \leftarrow H + h$
    Initialize tensor of zeros $\mathcal{L}' \in \Re^{B\times H'\times W'\times D}$
    $\mathcal{L}'[:, \lfloor h/2 \rfloor : H + \lfloor h/2 \rfloor, \lfloor w/2 \rfloor : W + \lfloor w/2 \rfloor, :] \leftarrow \mathcal{L}[:, :, :, :]$
    return $\mathcal{L}'$
end function

4 Max-Pooling Operation

Suppose we **max-pool** the image of size $H \times W$. The window size is $h \times w$. We take the maximum value in each window. The output image size is $(H/h) \times (W/w)$. We describe it by the following pseudocode:

function Max-Pooling($\mathcal{L} \in \Re^{B\times H\times W\times D}$
    $W' \leftarrow W/w$
    $H' \leftarrow H/h$
    Initialize tensor $\mathcal{L}' \in \Re^{B\times H'\times W'\times D}$
    for each $x = 0 \rightarrow H' - 1$:
        for each $y = 0 \rightarrow W' - 1$:
            $\mathcal{L}'[:, x, y, :] \leftarrow$ **max** $(\mathcal{L}[:, x*h : (x+1)*h, y*w, (y+1)*w, :])$
        end for
    end for
    return $\mathcal{L}'$
end function

5 Average-Pooling Operation

Similar to Max-Pooling operation, we have the Average-Pooling operation:

function Average-Pooling($\mathcal{L} \in \Re^{B\times H\times W\times D}$
    $W' \leftarrow W/w$
    $H' \leftarrow H/h$
    Initialize tensor $\mathcal{L}' \in \Re^{B\times H'\times W'\times D}$
    for each $x = 0 \rightarrow H' - 1$:

for each $y = 0 \to W' - 1$:
    $\mathcal{L}'[:, x, y, :] \leftarrow$ **average** $(\mathcal{L}[:, x*h : (x+1)*h, y*w, (y+1)*w, :])$
  end for
end for
return $\mathcal{L}'$
end function

6 Fully-Connected Layers

Suppose in the last **convolution** layer, we have the tensor $\mathcal{L} \in \Re^{B \times H \times W \times D}$. We will build a fully-connected network on top of it. For example, a 1 hidden layer Multi-Perceptron. First we reshape the tensor $\mathcal{L}$ into a 2D matrix $X \in \Re^{B \times (W*H*D)}$. Let $N = W * H * D$. Suppose the hidden layer has M neurons. And we have K classes for the task of classification. The output layer has K neurons.

Suppose the weight matrix from the input layer to hidden layer be $W^{(1)} \in \Re^{M \times N}$. The weight matrix from the hidden layer to the output layer $W^{(2)} \in \Re^{K \times M}$. With the nonlinearity σ (σ can be ReLU, Sigmoid or Tanh). The fully-connected network can be presented as:

$$\mathbb{P}(Y|X) = softmax\Bigg(\sigma\big(W^{(2)}\sigma(W^{(1)}X)\big)\Bigg)$$

where $X = [x^{(1)}, .., x^{(B)}]$ as the feature of the last convolution layer corresponding for each original input image and $Y = [y^{(1)}, .., y^{(B)}]$. The loss function is defined as the average log-loss:

$$\mathcal{L}(Y|X) = \frac{1}{B} \sum_{i=1}^{B} log\Bigg(\mathbb{P}(y^{(i)}|x^{(i)})\Bigg)$$

7 Running example

```
Input=
        [[ 1.,  0.,  1.,  0.,  1.],
         [ 0.,  1.,  0.,  1.,  0.],
         [ 1.,  1.,  1.,  1.,  1.],
         [ 0.,  0.,  0.,  0.,  0.],
         [ 1.,  1.,  1.,  0.,  0.]]

conv_filter in the first conv_layer:
         [[ 0.,  1.,  0.],
          [ 0.,  1.,  0.],
          [ 1.,  0.,  0.]]
```

```
Loss Function: (sum of the activations)^2

Activation after the conv_layer:
        [[ 2.,  2.,  2.],
         [ 2.,  1.,  2.],
         [ 2.,  2.,  2.]]

Total loss = (2 + 2 + 2 + 2 + 1 + 2 + 2 + 2 + 2)^2 = 17^2 = 289

gradient with respect to the output prediction node = 2 * 17 = 34

gradient with respect to the convolution activation =
        [[ 34.,  34.,  34.],
         [ 34.,  34.,  34.],
         [ 34.,  34.,  34.]]

gradient with respect to the input =
  [[   0.   34.   34.   34.    0.]
   [   0.   68.   68.   68.    0.]
   [  34.  102.  102.   68.    0.]
   [  34.   68.   68.   34.    0.]
   [  34.   34.   34.    0.    0.]]

gradient with respect to the weights of the convolution filter:
        [[ 204.,  204.,  204.],
         [ 136.,  170.,  136.],
         [ 204.,  170.,  136.]]
```