Deep Learning for Computer Vision

Backpropagation in CNNs

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Exercises

Given a $32 \times 32 \times 3$ image and 6 filters of size $5 \times 5 \times 3$, what will be the dimension of the output volume when a stride of 1 and a padding of 0 is considered?

Recall: $W_2 = \frac{W_1 - F + 2P}{S} + 1$; $H_2 = \frac{H_1 - F + 2P}{S} + 1$. Hence, dimension of output volume $=$ $28 \times 28 \times 6$

Acknowledgements

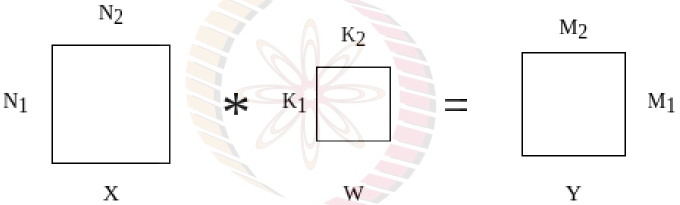
- This lecture's content is largely based on a similar lecture by Dhruv Batra at Georgia Tech (with a few adaptations)

Backpropagation in Convolutional Layers: Assumptions

- For simplicity, we consider a grayscale image i.e., number of input channels $C = 1$.
- Also, we consider the number of convolutional filters (which is also the number of output channels) to be 1.

Convolution Operation

- Consider a single convolutional filter $W^{K_1 \times K_2}$ applied to an image $X^{N_1 \times N_2}$ resulting in an output $Y^{M_1 \times M_2}$.



- Let an element of output $Y[i,j]$ be written as (note that we are not centering the convolution at a pixel here, but placing the filter at a corner of the window rather - this is only for convenience and simplicity of notation):

$$Y[i,j] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i-a, j-b] W[a,b]$$

Backpropagation in Convolutional Layer

- Given a loss function L used to train the CNN, for our convolutional layer, we need to calculate two gradients:
  1. $\partial L/\partial W$ with respect to the weights, for weight update
  2. $\partial L/\partial X$ with respect to the input, for further backprop to previous layers

Backpropagation in Convolutional Layer

Let's start with $\partial L / \partial W$, gradient w.r.t weights:

- Consider the gradient of the loss function with respect to a single weight in the convolutional filter (this can be generalized to other weights): $\partial L / \partial W[a', b']$
- How many pixels in the output (in the next layer, Y) does this weight affect?

Backpropagation in Convolutional Layer

Let's start with $\partial L/\partial W$, gradient w.r.t weights:

- Consider the gradient of the loss function with respect to a single weight in the convolutional filter (this can be generalized to other weights): $\partial L/\partial W[a', b']$
- How many pixels in the output (in the next layer, Y) does this weight affect?
- It affects every pixel in Y because:
  - Each pixel in the output corresponds to one position of the filter overlapping the input
  - Every pixel in the output is a weighted sum of a part of the input image

Backpropagation in Convolutional Layer

- We assume $\partial L/\partial Y$ is known since we compute gradients backward from the last layer
- Hence, $\partial L/\partial W[a', b']$ can be written as (summing all gradients coming from each pixel in the output):

$$\frac{\partial L}{\partial W[a', b']} = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} \underbrace{\frac{\partial L}{\partial Y[i, j]}}_{\text{known}} \underbrace{\frac{\partial Y[i, j]}{\partial W[a', b']}}_{\text{not known yet}}$$

- To expand this expression, let's compute $\frac{\partial Y[i,j]}{\partial W[a',b']}$.

Backpropagation in Convolutional Layer

Computing $\partial Y[i,j]/\partial W[a',b']$:

- We have (by definition of convolution):

$$Y[i,j] = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} X[i-a, j-b]W[a,b]$$

- So, we can compute $\partial Y[i,j]/\partial W[a',b']$ as:

$$\frac{\partial Y[i,j]}{\partial W[a',b']} = \frac{\partial\left(\sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} X[i-a,j-b]W[a,b]\right)}{\partial W[a',b']}$$
$$= \frac{\partial(W[a',b']X[i-a',j-b'])}{\partial W[a',b']} = X[i-a', y-b']$$

Backpropagation in Convolutional Layer

- We can hence write the gradient of loss function w.r.t weights as:

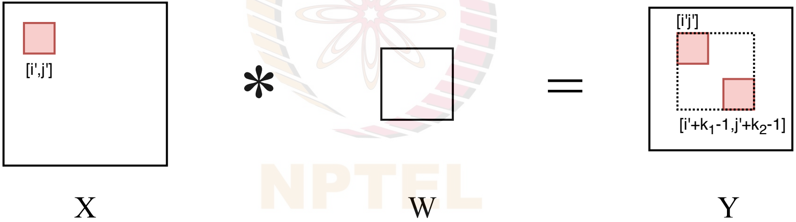
$$\frac{\partial L}{\partial W[a', b']} = \sum_{i=0}^{M_1-1} \sum_{j=0}^{M_2-1} \frac{\partial L}{\partial Y[i,j]} X[i-a', y-b']$$
$$= X * \frac{\partial L}{\partial Y}$$

- This is a convolutional operation, which is nice!

Backpropagation in Convolutional Layer

Let's now compute $\partial L/\partial X$, gradient w.r.t input:

- We consider a single input pixel $X[i', j']$. Which output pixels does it affect?
- That depends on the size of the convolutional filter:



- The dotted region in the output represents the output pixels affected by $X[i', j']$. Let's call the region P.

Backpropagation in Convolutional Layer

- Applying chain rule, we have:

$$\frac{\partial L}{\partial X[i',j']} = \sum_{p \in P} \underbrace{\frac{\partial L}{\partial Y[p]}}_{\text{known}} \underbrace{\frac{\partial Y[p]}{\partial X[i',j']}}_{\text{not known yet}}$$

- From the figure in previous slide, we can mathematically define the region P:

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} \underbrace{\frac{\partial L}{\partial Y[i'+a,j'+b]}}_{\text{known}} \underbrace{\frac{\partial Y[i'+a,j'+b]}{\partial X[i',j']}}_{\text{not known yet}}$$

- In the next slides, we calculate $\partial Y[i'+a,j'+b]/\partial X[i',j']$.

Backpropagation in Convolutional Layer

- We already have:

$$Y[i', j'] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i' - a, j' - b] W[a, b]$$

- We can rewrite it as:

$$Y[i' + a, j' + b] = \sum_{a=0}^{K_1-1} \sum_{b=0}^{K_2-1} X[i', j'] W[a, b]$$

- So the derivative can be calculated as:

$$\frac{\partial Y[i' + a, j' + b]}{\partial X[i', j']} = W[a, b]$$

Backpropagation in Convolutional Layer

- The final expression for gradient of the loss function w.r.t an input pixel can be written as:

$$\frac{\partial L}{\partial X[i',j']} = \sum_{a=0}^{K_1-1}\sum_{b=0}^{K_2-1} \frac{\partial L}{\partial Y[i'+a,j'+b]} W[a,b]$$
$$= \frac{\partial L}{\partial Y} * flip_{180^0}(W)$$

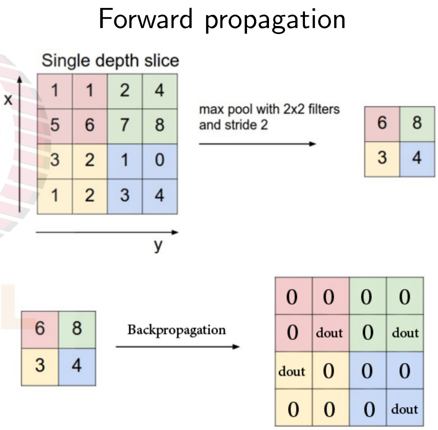
- Thus, the final expression is a convolution operation with a flipped version of the filter.

Backpropagation in Pooling Layers

- There are no weights to learn, only have to propagate gradients through

Backpropagation in Pooling Layers

- There are no weights to learn, only have to propagate gradients through
- In max-pooling, backpropagated gradient is assigned only to **the winning pixel** i.e., the one which had maximum value in the pooling block; this can be kept track of in the forward pass
- In average pooling, the backpropagated gradient is **divided by the area of the pooling block** $(K \times K)$ and **equally assigned to all pixels** in the block



Homework

Readings

- Lecture 5 Notes, Dhruv Batra, ECE 6504: Deep Learning for Perception,
- Jefkine, Backpropagation in CNNs