CS 189 Introduction to Machine Learning
Summer 2018 DIS13

1 Convolution and Backprop Revisited

In this problem, we will explore how image filtering can help us create useful high-level features that we can use instead of raw pixel values. We will walk through how discrete 2D convolution works and how we can use the backprop algorithm to compute derivatives through this operation.

(a) To start, let's consider convolution in one dimension. Convolution can be viewed as a function that takes a signal $f[t]$ and a filter $g[t]$, and the discrete convolution at point t of the signal with the filter is given as

$$(f * g)[t] = \sum_{k=-\infty}^{\infty} f[k]g[t-k]$$

If the filter $g[t]$ is nonzero in a finite range, then the summation can be reduce to just the range in which the filter is nonzero, which makes computing a convolution on a computer possible.

As an example, we can use convolution to compute a derivative approximation with finite differences. The derivative approximation of the signal is given by $f'[t] \approx (f[t+1] - f[t-1])/2$. Design a filter $g[t]$ such that $(f * g)[t] = f'[t]$.

Solution:

If we set $g[1] = -1/2$, $g[-1] = 1/2$, and $g[t] = 0$ everywhere else, we can use the equation for the discrete convolution to verify that $f * g$ gives us the derivative approximation.

(b) Convolution in two dimensions is similar to the one dimensional case except that we have an additional dimension to sum over. If we have some signal $I[x, y]$ and some filter $G[x, y]$, then the convolution at the point (x, y) is given by

$$(I * G)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I[m, n]G[x-m, y-n]$$

or equivalently,

$$(I * G)[x, y] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} G[m, n]I[x-m, y-n]$$

because convolution is commutative.

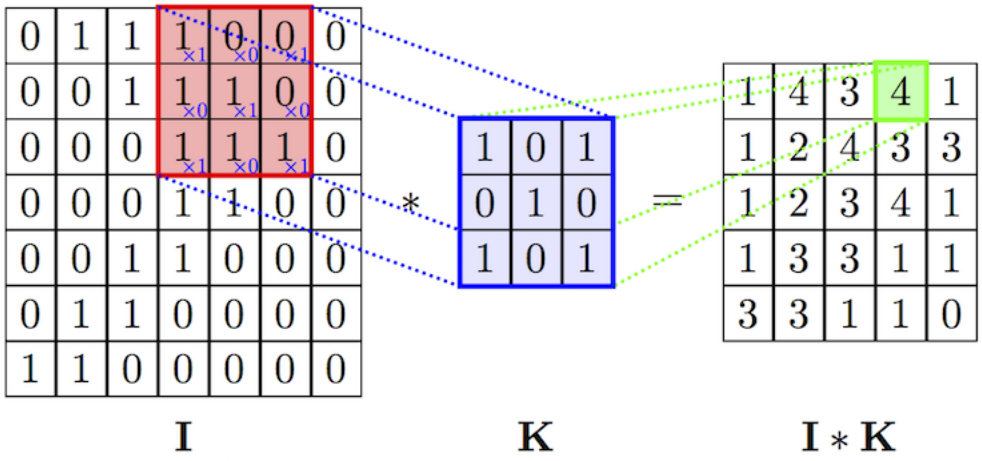

Figure 1: Figure showing an example of one convolution.

For implementation, we'll have an image I, which has three color channels I_r, I_g, I_b each of size $W \times H$, where W is the image width, H is the height. Each color channel represents the intensity of red, green, and blue for each pixel in the image. We also have the filter G with finite support. The filter also has three color channels, G_r, G_g, G_b and we represent these as a $w \times h$ matrix where w and h are the width and height of the filter. The output $(I * G)[x, y]$ at point (x, y) is given by

$$(I * G)[x, y] = \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{r,g,b\}} I_c[x + a, y + b] \cdot G_c[a, b]$$

In this case, the size of the output will be $(1 + W - w) \times (1 + H - h)$, and we only evaluate the convolution within the image I. We will not consider how to compute the convolution along the edge of the image. To reduce the dimension of the output, we can do a strided convolution in which we sample the convolution at every s point instead of every point. The resulting output will be $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (W - w)/s \rfloor$.

Write pseudocode to compute the convolution of an image I with a set of filters G and a stride of s.

Solution:

Note that the weights in a filter are shared across all the pixels in the input. For a convolutional net, we always use weight sharing because the same filters are applied across multiple positions of an input and because it reduces model complexity, which allows for fewer parameters than using a fully connected network.

```
for x in {0*s,1*s,2*s,...,W-w}
 for y in {0*s,1*s,2*s,...,H-H}
  total = 0
```

```
for c in {r,g,b}
 window = I_c[x:x+w,y:y+h]
 conv = window * G_c // * is element-wise multiplication
 total = total + summation(conv)
out_c[x/s,y/s] = total
```

The operator $*$ is element-wise multiplication of the two matrices, and summation() is the sum of all elements in the matrix.

(c) Filters can be used to identify different types of features in an image such as edges or corners. Design a filter G that outputs a large value for vertically oriented edges in image I.

Solution: An example vertical edge detector could look like

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ & \vdots & \\ -1 & 0 & 1 \end{bmatrix}$$

This detector would return a large positive value for edges that go from low color intensity to high color intensity and will return a large negative value for edges that go from high color intensity to low color intensity. The height of the detector determines the length of the edge that it can detect.

(d) Although hand-crafted filters can produce good results, we can learn filters specific to the problem that we are solving. To learn filters, we need to take derivatives of the filter parameters with respect to the error function. These derivatives can be computed efficiently using the backprop algorithm similar to how we computed derivates for model parameters in dense layers.

Given the output of a convolution L, the derivative of the error function with respect to the output ∂L, the convolution filters G, and the input image I, write an expression for the derivative of the filter parameter $G_c[i, j]$ for $c \in \{r, g, b\}$ and the derivative of $I_c[x, y]$ from the input image.

Solution:

Let's denote the error function as f, and we are given I, G, $(I * G) = L$, and $\frac{\partial f}{\partial L}$. Without knowing anything about the error function after L, we can compute the derivatives with respect to elements in I and G using the chain rule. Specifically, we have

$$\frac{\partial f}{\partial G_c[x, y]} = \frac{\partial f}{\partial L} \frac{\partial L}{\partial G_c[x, y]}$$

and

$$\frac{\partial f}{\partial I_c[x, y]} = \frac{\partial f}{\partial L} \frac{\partial L}{\partial I_c[x, y]}$$

where $G_c[x, y]$ denotes the entry in the filter for color c at position (x, y) and similarly for $I_c[x, y]$.

From the equation for discrete convolution, we can compute the derivatives for each entry (i, j) in L as

$$\begin{aligned}\frac{\partial L[i,j]}{\partial G_c[x,y]} &= \frac{\partial}{\partial G_c[x,y]} \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{r,g,b\}} I_c[i+a, j+b] \cdot G_c[a,b] \\ &= I_c[i+x, j+y]\end{aligned}$$

For the input image, we similarly compute the derivative as

$$\begin{aligned}\frac{\partial L[i,j]}{\partial I_c[x,y]} &= \frac{\partial}{\partial I_c[x,y]} \sum_{a=0}^{w-1} \sum_{b=0}^{h-1} \sum_{c \in \{r,g,b\}} I_c[i+a, j+b] \cdot G_c[a,b] \\ &= G_c[x-i, y-j]\end{aligned}$$

where we have $i + a = x$ and $j + b = y$. When $x - i$ or $y - j$ go outside the boundary of the filter, we can treat the derivative as zero.

We can collect the derivatives of the filter parameter for all $L[i, j]$ into a vector and multiply it by the derivatives we computed to get

$$\frac{\partial f}{\partial L}\frac{\partial L}{\partial G_c[x,y]} = \sum_{i,j} \frac{\partial f}{\partial L[i,j]} I_c[i+x, j+y] \tag{1}$$

$$= \frac{\partial f}{\partial G_c[x,y]} \tag{2}$$

and for the image

$$\begin{aligned}\frac{\partial f}{\partial L}\frac{\partial L}{\partial I_c[x,y]} &= \sum_{i,j} \frac{\partial f}{\partial L[i,j]} G_c[x-i, y-j] \\ &= \frac{\partial f}{\partial I_c[x,y]}\end{aligned}$$

Equation 1 shows the derivative of the loss with respect to one element of your weights at a particular layer. If you have multiple convolutional layers with different filter sizes, Equation 1 for different layers will have different number of terms, which may warrant adjusting the step size for normalization.

(e) Sometimes, the output of a convolution can be large, and we might want to reduce the dimension of this. A common method to reduce the dimension of an image is called max pooling. This method works similar to convolution in that we have a filter that moves around the image, but instead of multiplying the filter with a subsection of the image, we take the maximum value in the subimage. Max pooling can also be thought of as downsampling the image but keeping the largest activations for each channel from the original input. Given a filter size of $w \times h$, and a stride s, the output will be $\lfloor 1 + (W - w)/s \rfloor \times \lfloor 1 + (H - h)/s \rfloor$ for an input image of size $W \times H$.

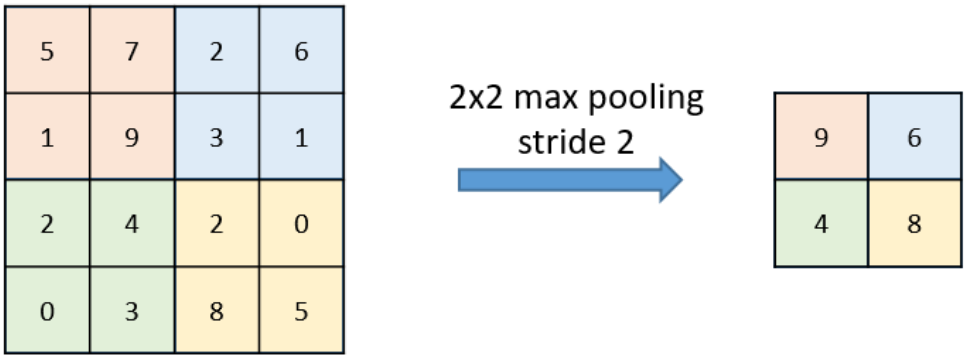

Figure 2: Figure showing an example of one maxpooling.

Let the output of a max pooling operation be L, write an expression for element $L[i, j]$ of the output.

Solution:

$$\text{maxpool}(I_c)[x, y] = \max_{a=\{0,\ldots,w-1\}} \max_{b=\{0,\ldots,h-1\}} I_c[x + a, y + b]$$

(f) Explain how we can use the backprop algorithm to compute derivates through the max pooling operation.

Solution: Similar to how we computed the derivatives through a convolution layer, we'll be given the derivative with respect to the output of the maxpool layer.

The gradient from the next layer is passed back only to the neuron which achieved the max. All other neurons get zero gradient.

Because maxpooling doesn't have any trainable parameters, we won't need to worry about calculating any derivatives for weights.

Once we have the derivative with respect to the input, the backprop algorithm can continue on to the layer before the maxpool by using this derivative.