Condensed Summaries

Contents

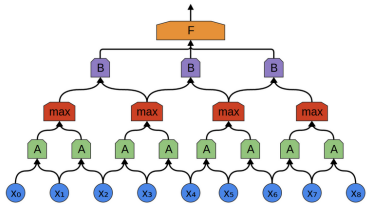
Condensed Summaries **Fall 2016**

Conv Nets: A Modular Perspective: December 21

Table of Contents Local *Written by Brandon McKinzie*

From this post on Colah's Blog.

The title is inspired by the following figure. Colah mentions how groups of neurons, like A, that appear in multiple places are sometimes called **modules**, and networks that use them are sometimes called modular neural networks. You can feed the output of one convolutional layer into another. With each layer, the network can detect higher-level, more abstract features.



⟶ Function of the A neurons: compute certain *features*.

⟶ Max pooling layers: kind of "zoom out". They allow later convolutional layers to work on larger sections of the data. They also make us invariant to some very small transformations of the data.

Understanding Convolutions: December 21

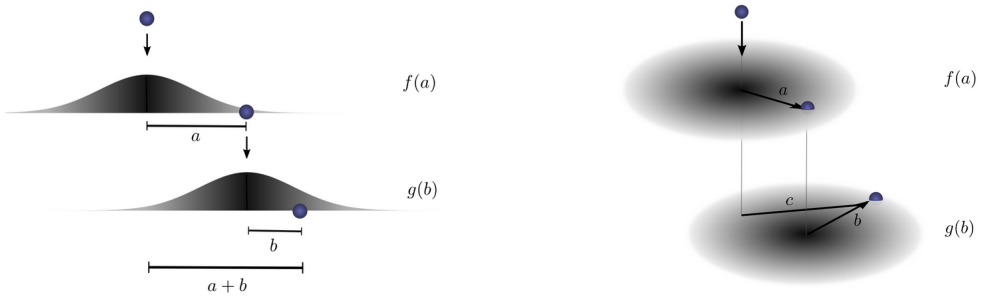
Table of Contents Local

Written by Brandon McKinzie

From Colah's Blog.

Ball-Dropping Example. The posed problem:

> Imagine we drop a ball from some height onto the ground, where it only has one dimension of motion. How likely is it that a ball will go a distance c if you drop it and then drop it again from above the point at which it landed?



From basic probability, we know the result is a sum over possible outcomes, constrained by $a + b = c$. It turns out this is actually the definition of the convolution of f and g.

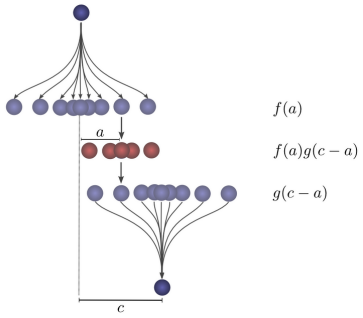
$$\Pr(\mathrm{a} + \mathrm{b} = \mathrm{c}) = \sum_{a+b=c} f(a) \cdot g(b) \tag{1}$$

$$(f * g)(c) = \sum_{a+b=c} f(a) \cdot g(b) \tag{2}$$

$$= \sum_{a} f(a) \cdot g(c - a) \tag{3}$$

Visualizing Convolutions. Keeping the same example in the back of our heads, consider a few interesting facts.

- **Flipping directions**. If $f(x)$ yields the probability of landing a distance x away from where it was dropped, what about the probability that it was dropped a distance x from where it *landed*? Apparently[1] it is $f(-x)$.



- Above is a visualization of one term in the summation of $(f * g)(c)$. It is meant to show how we can move the bottom around to think about evaluating the convolution for different c values.

We can relate these ideas to image recognition. Below are two common kernels used to convolve images with.

0	0	0	0	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

On the left is a kernel for *blurring* images, accomplished by taking simple averages. On the right is a kernel for *edge detection*, accomplished by taking the difference between two pixels, which will be largest at edges, and essentially zero for similar pixels.

[1] Not entirely sold on the generalization of this, or even how true it is here.