1 Convolutional Networks

This post is written for the AI Helsinki study group *Image and Video Statistics*. It is based on the Chapter 9 of the book *Deep Learning* by Ian Goodfellow, Yoshua Bengio and Aaron Courville, which is under preparation and available online at `http://www.deeplearningbook.org/`.

We will assume the reader to be familiar with basic concepts of artificial neural networks, such as weights and biases. Some mathematical preliminaries are introduced in the text.

Our goal is to introduce basic concepts related to convolutional neural networks, explain where they are used and what kind of benefits the convolutional structure offers.

1.1 Definition

We begin by introducing the mathematical notion of convolution.

Convolution in continuous case:

$$s(t) = \int x(a)w(t-a)da.$$

Denoted as

$$s(t) = (x * w)(t).$$

Convolution in discrete case:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

2-D case:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

Commutative:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

When defined without kernel-flipping:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

(also known as cross-correlation).

Definition: "Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers"

1.2 Tensors

We adopt the Deep Learning book's convention of defining tensors as multidimensional arrays of real numbers.

Namely, 0-D tensors are just real numbers, 1-D tensors are arrays

$$T = (T_1, \ldots, T_n)$$

of real numbers, 2-D tensors are arrays

$$T = ((T_{1,1}, \ldots, T_{1,n}), \ldots (T_{m,1}, \ldots, T_{m,n}))$$

of arrays of real numbers (with mutually equal lenght), and so on.

Basically our tensors can be viewed as D-dimensional grids of real numbers. Example in 2-D:

$$((1,2,3),(4,5,6),(7,8,9)) \quad \simeq \quad \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array}$$

In general case the sizes of the axes do not need to match.

1.3 Use of Convolutional Networks

Used to process data that has a grid-like structure (e.g. images, 1-D: sound?)

1.4 Benefits (submerge with previous section?)

*Main: Modelling of large inputs while being computationally efficient.
*Secondary: Spatial translations of inputs does not add problems.

1.5 Examples

(Pic of 2-D convolution, should be earlier?)

1.6 Sparse Connectivity

One benefit of using convolutional layers is that there are fewer connections between the nodes. Only as many input nodes as there are weights in the convolution kernel are connected to every single node in the output nodes and vice versa.

Pic: 1-D, connections from inputs to single node in the outputs

Pic: 1-D, connections from single input to nodes in the outputs

This is in drastic contrast to fully connected layers where there are connections from every input node to a given output node and vice versa.

1.7 Growing Receptive Fields

1.8 Parameter Sharing

Another benefit of using convolution is that there are fewer parameters needed. Namely the weights of the convolution kernel are shared between similar type of connection.

Pic: illustration of the parameter sharing

Sometimes even the biases are shared in similar manner, but this type of parameter sharing is not often used.

1.9 Zero Pading Enables Deeper Networks

One drawback of using convolutional layers is that the layers shrink, which opposes limit to the depth of the network.

Pic: 1-D network with shrinking layers

This however can be compansated by adding zeros to the boundary of each neuron layer. That way we can prevent the sizes of the layers shrinking and obtain deeper networks.

Pic: 1-D network with zero panding and same size layers

The method discribed above is known as zero padding. One way to do it is to add just enough zeros that the layer sizes stays the same. Other more extreme option is to add so many zeros that the convolution can visit every neuron k times, where k is the size of the convolution kernel.

Pic: 2-D, original layer and layer with minimal pading

Pic: 2-D, original layer and layer with maximal pading

One may notice that zero pading makes the outputs near the boundary to have less information from the inputs, especially if there is maximal amount of pading used. This may be compensated to some extend if the biases are not shared.

1.10 Max Pooling

Pictures: Without Shift, Shifted

1.11 Convolutional Network Components

1.12 Example of Learned Invariances

1.13 Examples of Architectures

1.14 Further Topics

* Pooling with Down Sampling (optional) * Convolution with Strides * Comparison of Local Connections, Convolution, and Full Connections (Optional) *

Partial Connectivity Between Channels * Tiled Convolution (optional) * Recurent Convolutional Networks

1.15 References/Links