Convolution as a Moving Window

These pages are about networks with different architecture. An m by n matrix still connects a layer with n neurons to the next layer with m neurons. Up to now, the layers were *fully connected: A had mn independent weights*. Now we might have only $E = 3$ or $E^2 = 9$ independent weights in A.

The fully connected "dense net" will be extremely inefficient for image recognition. First, the weight matrices A will be huge. If one image has 200 by 300 pixels, then its input layer has $60,000$ components. The weight matrix A_1 for the first hidden layer has $60,000$ columns. The problem is: We are looking for connections between faraway pixels.

Almost always, the important connections in an image are **local**.

Music has a 1D local structure

Images have a 2D local structure (3 copies for red-green-blue)

Video has a 3D local structure: Images in a time series

More than this, the search for structure is essentially *the same everywhere in the image*. There is normally no reason to process one part of a text or image or video differently from its other parts. We can use the same weights in all parts: *Share the weights*. The neural net of local connections between pixels is **shift-invariant**: the same everywhere.

The result is a big reduction in the number of independent weights. Suppose each neuron is connected to only E neurons on the next layer, and those connections are the same for all neurons. Then the matrix A between those layers has only E independent weights x. The optimization of those weights becomes enormously faster. In reality we have time to create several different channels with their own E or E^2 weights. They can look for edges in different directions (horizontal, vertical, and diagonal).

In one dimension, a banded shift-invariant matrix is a **Toeplitz matrix** or a **filter**. Multiplication by that matrix A is a **convolution** $\boldsymbol{x} * \boldsymbol{v}$. The network of connections between the layers is a **Convolutional Neural Net** (**CNN** or **ConvNet**). Here $E = 3$.

$$A = \begin{bmatrix} x_1 & x_0 & x_{-1} & 0 & 0 & 0 \\ 0 & x_1 & x_0 & x_{-1} & 0 & 0 \\ 0 & 0 & x_1 & x_0 & x_{-1} & 0 \\ 0 & 0 & 0 & x_1 & x_0 & x_{-1} \end{bmatrix} \qquad \begin{array}{l} \boldsymbol{v} = (v_0, v_1, v_2, v_3, v_4, v_5) \\ \boldsymbol{y} = A\boldsymbol{v} = \quad (y_1, y_2, y_3, y_4) \\ N+2 \text{ inputs } \boldsymbol{v} \text{ and } N \text{ outputs } \boldsymbol{y} \end{array}$$

It is valuable to see A as *a combination of* **shift matrices $\boldsymbol{L}, \boldsymbol{C}, \boldsymbol{R}$**: Left, Center, Right.

Each shift has a diagonal of 1's $\qquad \boldsymbol{A = x_1 L + x_0 C + x_{-1} R}$

Then the derivatives of $\boldsymbol{y} = A\boldsymbol{v} = \boldsymbol{x_1 L v} + \boldsymbol{x_0 C v} + \boldsymbol{x_{-1} R v}$ are exceptionally simple:

$$\frac{\partial(\text{output})}{\partial(\text{weight})} \qquad \boxed{\frac{\partial \boldsymbol{y}}{\partial x_1} = L\boldsymbol{v} \qquad \frac{\partial \boldsymbol{y}}{\partial x_0} = C\boldsymbol{v} \qquad \frac{\partial \boldsymbol{y}}{\partial x_{-1}} = R\boldsymbol{v}} \tag{1}$$

Convolution of Vectors $\boldsymbol{x} * \boldsymbol{v}$

The convolution of two vectors is written $\boldsymbol{x} * \boldsymbol{v} = (2,1,2) * (3,3,1)$. Computing the result $\boldsymbol{x} * \boldsymbol{v} = (6,9,11,7,2)$ is like multiplying the numbers 212 and 331, without carrying.

$$\begin{array}{rrrrrl}
 & & 3 & 3 & 1 & \text{Notice that we leave} \\
 & & 2 & 1 & 2 & 6+3+2=11 \text{ as is (no carrying)} \\
\cline{3-5}
 & & 6 & 6 & 2 & \text{Same steps for multiplying} \\
 & 3 & 3 & 1 & & 2x^2+x+2 \text{ times } 3x^2+3x+1 \\
6 & 6 & 2 & & & \text{That answer would be} \\
\hline
\mathbf{6} & \mathbf{9} & \mathbf{11} & \mathbf{7} & \mathbf{2} & \mathbf{6}x^4+\mathbf{9}x^3+\mathbf{11}x^2+\mathbf{7}x+\mathbf{2}
\end{array}$$

The previous page just put the numbers $(x_1, x_0, x_{-1}) = (2,1,2)$ on three diagonals of A. Then ordinary multiplication 212 times 331 converts to matrix-vector multiplication $A\boldsymbol{v}$. When $\boldsymbol{x}$ has length $j+1$ and $\boldsymbol{v}$ has length $k+1$, convolution $\boldsymbol{x} * \boldsymbol{v}$ has length $j+k+1$.

Convolution as a Moving Window

Suppose we average each number with the next number in $\boldsymbol{v} = (1,3,5)$. The result is $\boldsymbol{y} = (2,4)$. This is a typical convolution of $\boldsymbol{v}$ with the averaging vector $\boldsymbol{x} = \left(\frac{1}{2}, \frac{1}{2}\right)$:

$$\boldsymbol{y} = A\boldsymbol{v} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Notice the decision not to pad $\boldsymbol{v}$ with a zero at each end (and extend A to be 4 by 5). That would lead to 4 outputs $\boldsymbol{y}$ instead of 2. It would be consistent with multiplying numbers : 11 times 135 is 1485 and dividing by 2 gives $\frac{1}{2}, 2, 4, \frac{5}{2}$.

Python and MATLAB offer both versions of convolution, padded or not (and a third option with three outputs). We will choose not to pad the input with zeros. Each row of A is a perfect shift of the previous row, as above.

Often the convolution process $A\boldsymbol{v}$ is seen as a moving window. The window starts at $1\ 3$ and moves to $3\ 5$. Averaging produces 2 in the first window and 4 in the second window. The whole point of "shift invariance" is that *a convolution does the same thing in each window*.

Windows in Two Dimensions

This approach is helpful in two dimensions where the window is a square or a rectangle. It is easy to see 2 by 2 overlapping windows filling an n by n square. There would be $(n-1)^2$ windows and an average over each window. The matrix A has $(n-1)^2$ outputs from n^2 inputs. Each row of A has $\left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$: four nonzeros to average over a 2 by 2 window.

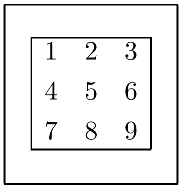
To produce that two-dimensional averaging convolution, first average neighboring pairs *in every row*. Then average every pair of neighbors *in every column*. The combination will average over every square. It works because the 2D averaging matrix has rank 1 :

$$\text{average over a 2 by 2 window} = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} = \text{column averages of row averages}$$

Experiments have pointed to $E = 3$ as a good choice for convolutions in deep learning. Deep learning could look for an E by E matrix like this of rank 1 — and apply the two averages along rows and then down columns. The rank 1 matrix for 2D convolution from an E by $E = 3$ by 3 matrix would have $2E - 1 = 5$ instead of $E^2 = 9$ unknown weights.

For clarity here are *nine* 3 *by* 3 *windows that fill a* 5 *by* 5 *square*.

3 × 3 windows in a 5 × 5 square



Move this window left / right / up / down / left up / right up / left down / right down to produce 9 windows centered in these nine positions

2D Convolution by One Large Matrix

When the input $\boldsymbol{v}$ is an image, convolution becomes two-dimensional. $E = 3$ numbers x_{-1}, x_0, x_1 change to $E^2 = 3^2$ independent weights. The inputs v_{ij} have two indices and $\boldsymbol{v}$ represents $(N+2)^2$ pixels. The outputs have only N^2 pixels unless we pad $\boldsymbol{v}$ with zeros at the boundary. The 2D convolution $A\boldsymbol{v}$ is a linear combination of **9 shifts of $\boldsymbol{v}$**.

	x_{11}	x_{01}	x_{-11}	**Input image**	v_{ij}	i, j from $(0,0)$ to $(N+1, N+1)$
Weights	x_{10}	x_{00}	x_{-10}	**Output image**	y_{ij}	i, j from $(1,1)$ to (N, N)
	x_{1-1}	x_{0-1}	x_{-1-1}	**Shifts L, C, R, U, D**		= **L**eft, **C**enter, **R**ight, **U**p, **D**own

$$\boxed{A = x_{11}LU + x_{01}CU + x_{-11}RU + x_{10}L + x_{00}C + x_{-10}R + x_{1-1}LD + x_{0-1}CD + x_{-1-1}RD}$$

This expresses the convolution matrix A as a combination of 9 shifts. The derivatives of the output $\boldsymbol{y} = A\boldsymbol{v}$ are again exceptionally simple. We use the nine derivatives in (2) to create the gradients $\boldsymbol{\nabla F}$ and $\boldsymbol{\nabla L}$ (learning function, loss function) that are needed in gradient descent to improve the weights $\boldsymbol{x}_k$. The next iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - s\boldsymbol{\nabla L}_k$ has weights that better match the correct outputs from the training data.

Backpropagation finds these 9 derivatives of $\boldsymbol{y} = A\boldsymbol{v}$ with respect to 9 weights :

$$\frac{\partial \boldsymbol{y}}{\partial x_{11}} = LU\boldsymbol{v} \quad \frac{\partial \boldsymbol{y}}{\partial x_{01}} = CU\boldsymbol{v} \quad \frac{\partial \boldsymbol{y}}{\partial x_{-11}} = RU\boldsymbol{v} \quad \ldots \quad \frac{\partial \boldsymbol{y}}{\partial x_{-1-1}} = RD\boldsymbol{v} \qquad (2)$$

CNN's can readily afford to have $\boldsymbol{B}$ **parallel channels** (and that number B can vary as we go deeper into the net). The count of weights in $\boldsymbol{x}$ is so much reduced by weight sharing and weight locality, that we don't need and we can't expect one set of $E^2 = 9$ weights to do all the work of a convolutional net. *B convolutions give the next layer.*

A convolution is a combination of shift matrices (producing a filter or Toeplitz matrix)

In deep learning, the coefficients in the combination will be the "weights" to be learned.

Several convolutions in parallel will extract more information from the image.

Two-dimensional Convolutional Nets

Now we come to the real success of CNN's: **Image recognition**. ConvNets and deep learning have produced a small revolution in computer vision. The applications are to self-driving cars, drones, medical imaging, security, robotics—there is nowhere to stop. Our interest is in the algebra and geometry and intuition that makes all this possible.

In two dimensions (for images) the matrix A is **block Toeplitz**. Each small block is E by E. This is a familiar structure in computational engineering. The count E^2 of independent weights to be optimized is far smaller than for a fully connected network. The same weights are used around all pixels (*shift-invariance*). The matrix A produces a 2D convolution $\boldsymbol{x} * \boldsymbol{v}$. Frequently A is called a **filter**.

To understand an image, look to see where it changes. *Find the edges.* Our eyes look for sharp cutoffs and steep gradients. The computer can do the same by creating a filter. The difficulty with two or more dimensions is that edges can have many directions. We will need horizontal and vertical and diagonal filters for the test images. And filters have many purposes, including smoothing and gradient detection and edge detection.

Smoothing For functions, one smoother is *convolution with a Gaussian* $e^{-x^2/2\sigma^2}$. For vectors, we could convolve $v * G$ with $G = \frac{1}{17}(1, 4, 7, 4, 1)$.

Gradient detection Image processing (as distinct from learning by a CNN) needs filters that detect the gradient. They contain specially chosen weights. We mention some simple filters just to indicate how they can find first derivatives of $\boldsymbol{f}$.

One dimension $E = 3$ $\quad (x_1, x_0, x_{-1}) = \left(-\frac{\mathbf{1}}{\mathbf{2}}, \mathbf{0}, \frac{\mathbf{1}}{\mathbf{2}}\right)$ Then $(A\boldsymbol{v})_i = \frac{1}{2}v_{i+1} - \frac{1}{2}v_{i-1}$.

Two dimensions These 3×3 *Sobel operators* approximate $\partial/\partial x$ and $\partial/\partial y$:

$$E = 3 \qquad \frac{\boldsymbol{\partial}}{\boldsymbol{\partial x}} \approx \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \frac{\boldsymbol{\partial}}{\boldsymbol{\partial y}} \approx \frac{1}{2}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3}$$

Edge detection Those weights were created for image processing, to locate the most important features of a typical image: *its edges*. These would be candidates for E by E filters inside a 2D convolutional matrix A. But remember that in deep learning, weights like $\frac{1}{2}$ and $-\frac{1}{2}$ are not chosen by the user. They are created from the training data.