Lecture 6:Convolutional Neural Network (CNN), Convolution Rule

Convolutional Neural Network (CNN), Convolution Rule

- Agenda:
- Convolution
- Convolution Rule

Source: Section IV.2, IV.3 in Linear Algebra and Learning from Data (2019) by **Gilbert Strang**

Convolutional Neural Network (CNN), Convolution Rule

Why Convolution is so important? Take a reference paper

- When you go to convolution neural nets, CNN, because a convolutional net takes fewer weights and same weight as appearing along diagonals. It doesn't need a full matrix of weights, only one top row of weights
- Reference paper is one of the historical papers in the history of deep learning; in abstract, it said that it trained a large deep convolutional neural network. it ran for five days on two GPUs. So it was an enormous problem. So we trained a large deep network, CNN, to classify 1.2 million high-res images in ImageNet. We realize what's gone into it
- It has convolution layers, and it has some normal layers, and it has max pooling layers to cut the dimension down a little bit. It's a key problem is to reduce overfitting in the fully connected layers.
- Neural network has, say, 60 million parameters. With 650,000 neurons, five convolutional layers, and three fully connected layers.
- Convolution everybody needs to understand

Ref paper : ImageNet Classification with Deep Convolutional Neural Networks-Krizhevsky, Sutskever, Hinton

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Convolutional Neural Network (CNN), Convolution Rule

Convolution Rule:

Convolve two vectors c and d: $(c \circledast d)_k\,(x) = \sum_{i+j=k} c_i d_j\, x_i\, x_j$

$\rightarrow$ $(c_0 + c_1x + c_2x^2 + \ldots + c_Nx^N)\,(d_0 + d_1x + d_2x^2 + \ldots + d_Mx^M)$ $\rightarrow$ x^k coefficients are $c_0d_k + c_1d_k + \ldots$

$\rightarrow$ $\sum_{i+j=k} c_i d_j = \sum_{i=0} c_i d_{k-i}$

$\boxed{\text{Hence, } (c \times d)_k = \sum_{i=0} c_i d_{k-i}}$

Convolutional Neural Network (CNN), Convolution Rule

Convolution of functions: $\boxed{f(x) \circledast g(x) = (f \circledast g)(t) = \int f(x)\, g(t-x)\, dx}$

f⇔c, g⇔d, i⇔x and (k-i)⇔(t-x),
f corresponds to c. g corresponds to d. k corresponds to x. i corresponds to x. k-i corresponds to t-x. k corresponds to t. This is convolution of two functions

t is the amount of shift. g is shifted. we've reversed it. We've flipped it and shifting it by different amounts t. It's what you have in a filter. It's just also always present in signal processing.

Convolutional Neural Network (CNN), Convolution Rule

$$\boxed{f(x) \circledast g(x) = (f \circledast g)(x) = \int_{t=-\infty}^{t=\infty} f(t)\, g(x-t)\, dt}$$

if you want an x variable to come out, let us make an x variable come out by exchanging t and x

Convolutional Neural Network (CNN), Convolution Rule

Cyclic Convolution $(c\otimes d)_k\,(x)$ x is replaced by $w \rightarrow e^{2\pi i/N}$ and $w^N=1$

Property: w^n is 1 so that all vectors of length greater than n can be folded back using this rule to a vector of length n. So it is a cyclic component

Non periodic

$$T=\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & t_0 & t_1 & t_2 \\ 2 & t_{-1} & \ddots & \ddots \\ 3 & t_{-2} & \ddots & \ddots \end{array}$$

Toeplitz

Periodic

$$C=\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & C_0 & C_1 & C_2 \\ 2 & C_2 & C_0 & C_1 \\ 3 & C_1 & C_2 & C_0 \end{array}$$

Circulant NxN matrix
Its nxn period n

Length of non circulant case $(c\otimes d)_k\,(x)$

c has p components of polynomial degree p-1 and d has q components of polynomial degree q-1

$(c\otimes d) = p+q-1$ components

Convolutional Neural Network (CNN), Convolution Rule

Example: (cxd) $=(3+x)\,(1+2x)= 3+7x+2x^2$

C has (p=)2 components and d has has (q=)2 components. cxd has 2+2-1=3 components

If we had a two-diagonal matrix, Toeplitz matrix times Toeplitz matrix, that would give me a three diagonal . But if two times circulant matrices periodically, we would only have two diagonals

So that's a reminder of what convolution is. Cyclic and non-cyclic, vectors and functions
Then eigenvalues and eigenvectors are the next step, and then the convolution rule is the last step

Eigenvectors of the circulant. Only do for square matrices

Eigen vectors of Circulants=Columns of Fourier matrix

Fourier matrix

$$F=\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & i & i^2 & i^3 \\ 3 & 1 & i^2 & i^4 & i^6 \\ 4 & 1 & i^3 & i^6 & i^9 \end{array}$$

$C=C_0I+C_1P+C_2P^2+C_3P^3$ $\rightarrow$ Combinations of evectors $\rightarrow$ What are its evectors of colms of F ?

What are its evalues ?

Convolutional Neural Network (CNN), Convolution Rule

Multiply F times c, We get the eigenvalues of the matrix C.

$$FC = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & i & i^2 & i^3 \\ 3 & 1 & i^2 & i^4 & i^6 \\ 4 & 1 & i^3 & i^6 & i^9 \end{array} \begin{bmatrix} C_o \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \text{Four evalues of C} \rightarrow$$

there's only C_0, then only get C_0, C_0, C_0, C_0,.
It's four times repeated

Convolution Rule: $F(c \otimes d)$ Multiply by F times c and d, and we convolve them. Got another circulant matrix

Multiplying matrices CD, Topelitz matrices, multiplying those matrices with convolving the c's and make the connection. And that connection is the convolution rule. So, this would be the eigenvalues of CD

Back-up Slides

Convolutional Neural Network (CNN), Convolution Rule

Looking at the eigenvalues of CD- Because if we do that multiplication, we get another Toeplitz matrix, C times D. And the polynomial-- the coefficients associated on the diagonals of C times D are the coefficients of the convolution. So its diagonals come from convolving c with d cyclically

we can use cyclic multiplication and a circulant by a doubling trick: embed A into a circulant matrix C

$$
\boldsymbol{A} = \begin{matrix} a_0 & a_{-1} & \cdot & \cdot & a_{1-n} \\ a_1 & \ddots & & & \cdot \\ \cdot & & \ddots & & \cdot \\ \cdot & & & \ddots & a_{-1} \\ a_{n-1} & \cdot & \cdot & a_1 & a_0 \end{matrix}
\qquad
\boldsymbol{C} = \begin{bmatrix} \boldsymbol{A} & * \\ * & * \end{bmatrix} = \begin{matrix} a_0 & a_{-1} & \cdot & \cdot & a_{1-n} & a_{n-1} & \cdot & \cdot & a_1 \\ a_1 & \ddots & & & \cdot & \ddots & & & \cdot \\ \cdot & & \ddots & & \cdot & & \ddots & & \cdot \\ \cdot & & & \ddots & a_{-1} & & & \ddots & a_{n-1} \\ a_{n-1} & \cdot & \cdot & a_1 & a_0 & \ddots & & & a_{1-n} \\ a_{1-n} & \ddots & & & & \ddots & & & \cdot \\ \cdot & & \ddots & & & & \ddots & & \cdot \\ \cdot & & & \ddots & & & & \ddots & a_{-1} \\ a_{-1} & \cdot & \cdot & a_{1-n} & a_{n-1} & \cdot & \cdot & a_1 & a_0 \end{matrix}
$$

Source: Page 234 in Linear Algebra and Learning from Data (2019) by Gilbert Strang

Convolutional Neural Network (CNN), Convolution Rule

Evalues of CD= (eigen values of c) (eigen values of d)

F(c⊗d)= F(c) .✻ F(d) (.✻→ components by components multiplications)

If a vector with three components. So, n is 3 here. And I do point star or dot star. Component by component, a three component vector times a three-component vector, get three component vector

F(c⊗d)= F(c) .✻ F(d) → convolution rule

This term by term "Hadamard product" is denoted in MATLAB by .✻

$$\begin{bmatrix} \lambda_0(CD) \\ \vdots \\ \lambda_{N-1}(CD) \end{bmatrix} = \begin{bmatrix} \lambda_0(C)\lambda_0(D) \\ \vdots \\ \lambda_{N-1}(C)\lambda_{N-1}(D) \end{bmatrix} = \begin{bmatrix} \lambda_0(C) \\ \vdots \\ \lambda_{N-1}(C) \end{bmatrix} .\!* \begin{bmatrix} \lambda_0(D) \\ \vdots \\ \lambda_{N-1}(D) \end{bmatrix} = \boldsymbol{Fc} \ .\!* \ \boldsymbol{Fd}$$

Source: Page 218 in Linear Algebra and Learning from Data (2019) by Gilbert Strang

Convolutional Neural Network (CNN), Convolution Rule

C -rule : Convolve then transform by F. Or transform separately by F. And then multiply point one. Element by element. Component by component

Why is it so important?
$F(c \otimes d) \rightarrow$ Convolve c and d first, then transform by F

$F(c) .* F(d) \rightarrow$ Transform by F first, then multiply component by component. It is extremely fast by the FFT

Presence of the FFT gives us really two different ways to do it

Complexity: $F(c \otimes d) \rightarrow N^2$ for c and d multiplications and NlogN for Fourier transform, i.e., $N^2 + N\log N$

$F(c) .* F(d) \rightarrow$ 2NlogN for two multiplications and N for component by component multiplications, i.e., $2N\log N + N$ which is faster

Convolutional Neural Network (CNN), Convolution Rule

Consider 2D functions of x and y i.e., $f(x,y)=\int_t \int_u f(t,i)\, g(x-t, y-u)\, dt\, du$ two-dimensional convolution integral

Two dimensional matrices, $A_{(NxN)}$ and $B_{(NxN)}$ and their products are (N^2xN^2). Because if we have two dimensional signals, then the components fit into a matrix. And to operate in both dimensions

This construction uses the Kronecker products $F \otimes F$, We call Kron (F, F) [MATLAB Command also Kron (F, F)]

$$K= \text{Kron}\,(A,B)= A \otimes B= \begin{pmatrix} a_{11}B & a_{12}B & & a_{1N}B \\ a_{21}B & a_{22}B & & a_{2N}B \\ & & & \\ a_{N1}B & a_{N2}B & & a_{NN}B \end{pmatrix}$$

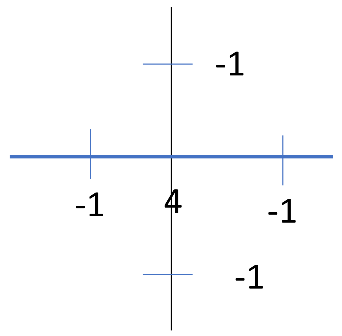
Example:

$$A= \begin{pmatrix} 2 & -1 & \\ -1 & \ddots & \ddots \\ & \ddots & 2 \end{pmatrix} \simeq -(\partial^2 u/\partial x^2)$$

$$\text{Similarly, } B= \begin{pmatrix} 2 & -1 & \\ -1 & \ddots & \ddots \\ & \ddots & 2 \end{pmatrix} \simeq -(\partial^2 v/\partial y^2)$$

So, $K= -(\partial^2 u/\partial x^2)-(\partial^2 v/\partial y^2)$

Convolutional Neural Network (CNN), Convolution Rule



From two matrices A, B, Five weights at each point.
It takes four of the-- on the diagonal and -1 on the four neighbors

Kron sum= Kron (A,I)+Kron (I,B)

Kron of A times the identity. That gives me the two dimensional thing for x-axis direction (horizontal direction)

Kron of I B for the vertical derivative, the derivatives in the y direction

This would be another Kronecker product, and the total is called the Kronecker sum

Because really, Fourier transforming is such a central operation and especially in signal processing.