Mathematics reminders for deep learning (and more)
Part 1: Linear Algebra

Georges Quénot

Univ. Grenoble Alpes, CNRS, Grenoble-INP, LIG, F-38000 Grenoble France

February, 2020

Algebra - Vector Space and Linear Applications

- A *Vector Space* (VS) V of dimension n over a *Field* F ($\mathbb{R}$ or $\mathbb{C}$) is isomorphic to F^n (via a bijective linear application)
- *Scalar* : element of F
- *Vector* : element of V (or of F^n)
- *Addition* of vectors $+ : V \times V \to V : (x, y) \to x + y$
- *Multiplication* of a vector by a scalar $. : F \times V \to V : (a, x) \to a.x$
  The dot may be omitted ($a.x \equiv ax$)
- *Linear Map* (VS $\to$ VS) or *Linear Form* (VS $\to F$) f:
  - $\forall (x, y) \in V \times V : f(x + x) = f(x) + f(y)$
  - $\forall (a, x) \in F \times V : f(a.x) = a.f(x)$
  - the source and target vector spaces may be different
- Also: vector spaces of infinite dimensions.

Algebra - Row and column vector and form representations

- *Representations* are relative to a *coordinate system* or *basis* in a “regular” vector space
- *Column* (“regular”) *vector representation*: $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}^T$
- *Row* (linear form) *covector representation*: $y = \begin{pmatrix} y_1 & y_2 & \dots & y_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}^T$
- The *transpose* operator (T is a superscript, not an exponent) swaps rows and columns
- Application of a linear form to a regular vector: $f_y(x) = \sum_{k=1}^{k=n} y_k x_k = y.x = yx = f_x^*(y)$
- Linear forms (covectors) and “regular” vectors belong to *dual* vector spaces

Algebra - Linear maps matrix representations

- *Matrix* as a *linear map representation* : $a = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}$
- in the absence of ambiguity, commas may be removed: $a_{i,j} \equiv a_{ij}$
- m : number of rows = dimension of the target vector space
- n : number of columns = dimension of the source vector space
- Particular cases (indexes fixed to 1 may be dropped):
  - $n = 1$: a vector is equivalent to a matrix that has a single column (column vector)
  - $m = 1$: a covector is equivalent to a matrix that has a single row (row vector)
  - $m = n = 1$: a scalar is equivalent to a matrix that has a single element
- The transpose of an $m \times n$ matrix is an $n \times m$ matrix: $(a_{ij})^T = (a_{ji})$
- The transpose of a row vector is a column vector and vice versa

Algebra - Matrix multiplication ("dot" or "inner" product)

- Matrices can be multiplied *if and only if* the number of columns in the left matrix is equal to the number of row in the right matrix (n here)
- The product of an $m \times n$ matrix a by an $n \times p$ matrix b is an $m \times p$ matrix c

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & \dots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{np} \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mp} \end{pmatrix}$$

with: $c_{ij} = \sum_{k=1}^{k=n} a_{ik} b_{kj} = \begin{pmatrix} a_{i1} & \dots & a_{in} \end{pmatrix} \begin{pmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{pmatrix} = a_{i.} b_{.j}$ for all $1 \leq i \leq m$ and $1 \leq j \leq p$

- The c_{ij} (scalar) element of c is the "dot" or "inner" product of the ith row $a_{i.}$ (covector) of a by the jth column $b_{.j}$ (vector) of b
- The matrix product is associative but *not* commutative

Algebra - Matrix multiplication: particular cases

- $p = 1$: product of a matrix with a column vector *on the right* $\rightarrow$ column vector:

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_m \end{pmatrix} \text{ with: } c_i = \sum_{k=1}^{k=n} a_{ik} b_k$$

- $m = 1$: product of a matrix with a row vector *on the left* $\rightarrow$ row vector:

$$\begin{pmatrix} a_1 & \dots & a_n \end{pmatrix} \begin{pmatrix} b_1 & \dots & b_p \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{np} \end{pmatrix} = \begin{pmatrix} c_1 & \dots & c_p \end{pmatrix} \text{ with: } c_j = \sum_{k=1}^{k=n} a_k b_{kj}$$

Algebra - Matrix multiplication: particular cases

- $p = m = 1$: product of a raw vector with a column vector $\rightarrow$ scalar:

$$\begin{pmatrix} a_1 & \dots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} c \end{pmatrix} \text{ with: } c = \sum_{k=1}^{k=n} a_k b_k \text{ (vector dimensions must be the same)}$$

- $n = 1$: product of a column vector with a raw vector $\rightarrow$ matrix:

- $$\begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \begin{pmatrix} b_1 & \dots & b_p \end{pmatrix} = \begin{pmatrix} c_{11} & \dots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mp} \end{pmatrix} \text{ with: } c_{ij} = a_i b_j \text{ (} m \text{ and } p \text{ may be different)}$$

Algebra - Multidimensional arrays

- A d-dimensional array is a set of numbers arranged on a regular d-dimensional grid
- The number d of *axes* of a multidimensional array must not be confused with the number of dimensions of a vector, e.g., e vector of n elements may be stored in a single axis (1-dimensional) array of n dimensions
- For instance an $n_1 \times n_2 \times n_3$ 3-dimensional (3D) array has three axes whose respective dimensionality are n_1, n_2 and n_3
- The various axes can be typed ("row", "column", "horizontal", "vertical", "feature")
- The order of the axes matters
- The transpose operation is a special case of axes permutation

Algebra - Tensors

- Tensors are d-dimensional arrays with two types of indexes (one index per axis):
  - *covariant* or "column" or "vector" or "regular" type
  - *contravariant* or "row" or "linear form" or "dual" type
- There are in principle different types of tensors with the same number of axes:
  - A column vector is a 1-dimensional tensor with one contravariant index (components in a column vectors correspond to different rows and vice-versa)
  - A row (co)vector is a 1-dimensional tensor with one covariant (column) index
  - A matrix is a 2-dimensional tensor with one contravariant index one covariant index
  - There may be 2-dimensional tensors with two contravariant indexes or with two covariant indexes
- We don't care much about that distinction, *except for dot product operations*
- In deep learning, we only consider tensors as d-dimensional arrays but *the order of axes, as well as how they are matched during product operations does matter*
- We also consider other types of indexes for axes not subject to dot product operations

Algebra - Tensors: outer (or dyadic) product

- Example: $\left(a_{ijk}\right) \otimes \left(b_{lm}\right) = \left(c_{ijklm}\right)$ with:
  - $c_{ijklm} = a_{ijk} b_{lm}$ for all valid i, j, k, l and m
  - $\left(a_{ijk}\right)$ denotes an $I \times J \times K$ 3D tensor
  - $\left(b_{lm}\right)$ denotes an $L \times M$ 2D tensor
  - $\left(c_{ijklm}\right)$ denotes an $I \times J \times K \times L \times M \times K$ 5D tensor
- The outer product of tensors can be done with tensors of any type and any axis lengths
- The number, order, type and length of axes are conserved during the outer product
- The outer product is associative but *not* commutative

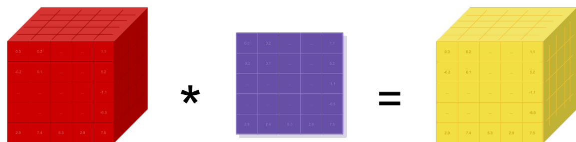
- The covariant / contravariant distinction is important only for axes subject to dot product operations (i.e. not for outer, Hadamard and convolution products)
- The dot product "contracts" dimensions of opposite types (removes two indexes)

Algebra - Tensors: product with (dot-type) contraction

- The left tensor a has d_a axes, one of which is of a row-type;
  it can be seen as a tensor a^* with $d_a - 1$ axes whose elements are row (co)vectors
- The right tensor b has d_b axes, one of which is of a column-type;
  it can be seen as a tensor b^* with $d_b - 1$ axes whose elements are column vectors
- the product tensor $c = ab$ is a tensor with $d_a + d_b - 2$ axes whose elements are the dot products of the row elements of a^* and the column elements of b^*
- $d_a = 2, d_b = 2$: standard matrix-matrix product
- $d_a = 2, d_b = 1$: standard matrix-vector product
- $d_a = 1, d_b = 2$: standard covector-matrix product
- $d_a = 1, d_b = 1$: standard covector-vector product

Algebra - Tensors: product with (dot-type) contraction

- $d_a = 3, d_b = 2$: the product of a "stack of matrices" by a standard matrix is a stack of matrices which is the stack of the matrix-matrix products



(Image from Pytorch tutorial)

- Any axis from the left tensor can be contracted with any axis of the right tensor as long as they have the same length and are of opposite types (if they are typed)
- Non-contracted axes are combined as in an outer product $(a^* \otimes b^*)$
- The product with contraction can be decomposed into an outer product followed by *trace* operation (sum over "diagonal") on axes with same lengths and opposite types

Implementation - numpy.array, torch.tensor, C arrays ...

- d-dimensional arrays:
  - $d = 0$: scalar, single real number (unused as an array)
  - $d = 1$: vector or covector, one-dimensional array of real numbers
  - $d = 2$: matrix, two-dimensional array of real numbers
  - $d > 2$: tensor, d-dimensional array of real numbers
- numpy arrays and torch tensors may have any number of axes, some of which may be of length 1, for example: 3×3 (2D), $2 \times 3 \times 4$ (3D), $4 \times 4 \times 4 \times 4 \times 4$ (5D), 2 (1D), 2×1 (2D with one of length 1), 1×2 (2D with one of length 1) ...
- *No difference* between column vectors and row (co)vectors; both are usually coded in 1D arrays, though these can also be explicitly coded in $1 \times n$ and $n \times 1$ arrays
- Operation between d-dimensional arrays are specified with *the type of operation* (inner, outer, Hadamard, convolution ...) and with *the dimension axes* (indexes) on which they apply and/or with *the order of the operands* (see numpy/scipy and torch documentation)

1D convolution

- A finite vector $a = (a_k)_{(1 \leq k \leq n)}$ is a function $f : [1, n] \rightarrow \mathbb{R} : k \rightarrow f(k) = a_k$
- An infinite vector $a = (a_k)_{(k \in \mathbb{Z})}$ is a function $f : \mathbb{Z} \rightarrow \mathbb{R} : k \rightarrow f(k) = a_k$
- An infinite vector $a = (a_k)_{(k \in \mathbb{Z})}$ is square-bounded if $\sum\limits_{k \in \mathbb{Z}} a_k^2$ converges
- A convolution $a * b$ is defined between square-bounded infinite vectors a and b (or $f * g$ between square-bounded functions from $\mathbb{Z}$ to $\mathbb{R}$ f and g) as:
  $(a * b)_i = \sum\limits_{k \in \mathbb{Z}} a_{i-k} b_k$ or: $(f * g)(i) = \sum\limits_{k \in \mathbb{Z}} f(i - k) g(k)$
- The convolution of square-bounded infinite vectors (or functions) is square-bounded
- The convolution operation is actually symmetric and commutative

1D convolution

- The convolution of a signal a by a kernel (or a filter) b at index i is a linear combination of the neighbors of a_i (possibly including a_i itself) weighted by the elements of b
- As these do not depend upon i, the result is invariant by translation as i changes ("sliding window" constant linear combination)
- Examples : FIR filter, local averaging filter Gaussian filter, derivatives ...
- The signal/kernel distinction is arbitrary, the convolution operation is symmetric.
- In practice, finite-support (where they are non-zero) vectors are used as kernels
- The sliding window operation is also generally applied on finite vectors
  $\rightarrow$ side (border) effects, dealt with by padding or cropping

2D convolution

- An infinite 2D array $a = (a_{kl})_{((k,l)\in\mathbb{Z}^2)}$ is a function $f : \mathbb{Z}^2 \to \mathbb{R} : (k,l) \to f(k,l) = a_{kl}$
- An infinite 2D array $a = (a_{kl})_{(k\in\mathbb{Z})}$ is square-bounded if $\sum_{(k,l)\in\mathbb{Z}^2} a_{kl}^2$ converges
- A convolution $a * b$ is defined between square-bounded infinite 2D arrays a and b (or $f * g$ between square-bounded functions from $\mathbb{Z}^2$ to $\mathbb{R}$ f and g) as:
  $(a * b)_{ij} = \sum_{(k,l)\in\mathbb{Z}^2} a_{i-k,j-l} b_{kl}$ or: $(f * g)(i,j) = \sum_{(k,l)\in\mathbb{Z}^2} f(i-k, j-l) g(k,l)$
- The convolution of square-bounded infinite vectors (or functions) is square-bounded
- The 2D convolution operation is actually symmetric and commutative
- a and b can respectively (and arbitrarily) be considered as a 2D signal and a 2D kernel

Summary: *4 types* of vector products (all generalizable to tensors)

- Vectors of same size:
  - inner or scalar or dot product: $(a_k).(b_k) = \sum_{k=1}^{k=n} a_k b_k = (c)$, scalar result
  - element-wise or Hadamard product: $(a_k) \circ (b_k) = (a_k b_k) = (c_k)$, vector result
- Vectors of (possibly) different sizes:
  - outer or dyadic product: $(a_i) \otimes (b_j) = (a_i b_j) = (c_{ij})$, matrix result
  - convolution product: $(a_i) * (b_k) = \sum_{b_k \neq 0} a_{i-k} b_k = (c_i)$, vector result

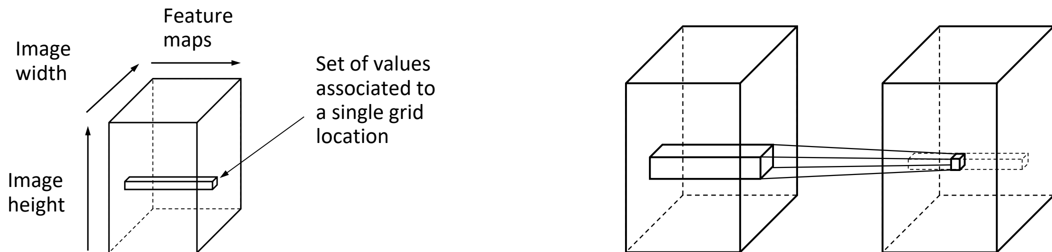
    (b_k) can be seen as a kernel, possible "side effects" to consider
- Generalization by adding axes (examples above)
- Generalization by repeating and/or combining the operations (examples to follow)

CNN layer convolution product: translation invariant linear operator

- CNN layers input and outputs: 3D tensors as stacks of "feature maps"
  - Two dimensions following to the image topology (i and j indices)
  - A third "feature" dimension (k and l indices respectively for input and output)
  - Sizes of $I_f \times I_w \times I_h$ and $O_f \times O_w \times O_h$ for the input and output tensors respectively
- CNN kernel: 4D tensor as stack of stacks of 2D convolution kernels
  - Size of $O_f \times I_f \times (2K_w + 1) \times (2K_h + 1)$ for the kernel tensor
- CNN layer product ($*_{NN}$): generalized combination (not just a sequence) of:
  - A convolution within the image plane (m and n indices, translation invariant)
  - A matrix-vector operation in the feature dimension (k and l indices, "all to all")
- $$(O_{l,i,j}) = (K_{l,k,m,n}) *_{NN} (I_{k,i,j}) = \sum_{k=1}^{k=I_f} \sum_{m=-K_w}^{m=+K_w} \sum_{n=-K_h}^{n=+K_h} K_{l,k,m,n} \, I_{k,i-m,j-n}$$

  for all $1 \leq l \leq O_f$, $1 \leq i \leq O_w$ and $1 \leq j \leq O_h$ (possible side effects in the image plane)

CNN layer convolution product: translation invariant linear operator



- Each map point in the output is connected to all maps points of a fixed size neighborhood in the input
- Kernel weights between maps are the same at all grid locations so that the computations are invariant by translation in the image plane
- One stack (over input feature maps) of independent 2D convolution kernels per output feature map $\rightarrow$ one 3D kernel per output feature map $\rightarrow$ one 4D global kernel

CNN layer convolution product, batch processing

- CNN layers input and outputs: 4D tensors as sequences of 3D tensors
  - A fourth "batch" dimension (b index for both input and output)
  - Sizes of $B \times I_f \times I_w \times I_h$ and $B \times O_f \times O_w \times O_h$ for the input and output tensors respectively
  - Independent and identical "per image" processing
- CNN kernel: 4D tensor as stack of stacks of 2D convolution kernels (no change)
  - Size of $O_f \times I_f \times (2K_w + 1) \times (2K_h + 1)$ for the kernel tensor
- $$(O_{b,l,i,j}) = (K_{l,k,m,n}) *_{NN} (I_{b,k,i,j}) = \sum_{k=1}^{k=I_f} \sum_{m=-K_w}^{m=+K_w} \sum_{n=-K_h}^{n=+K_h} K_{l,k,m,n} \, I_{b,k,i-m,j-n}$$

  for all $1 \leq b \leq B$, $1 \leq l \leq O_f$, $1 \leq i \leq O_w$ and $1 \leq j \leq O_h$
- The use of 4D data tensors enables very efficient parallel implementations on GPUs
- 2D signal considered here but generalizable to nD signal processing (1D, 3D, 4D ...)