Linear Algebra in Deep Learning

(Vectorisation)

Tirtharaj Dash

Dept. of CS & IS and APPCAIR
BITS Pilani, Goa Campus

September 7, 2021

Linear Algebra[1]

[1] https://xkcd.com/1838/

What is this material about?

We look at how linear algebra allows us to write vectorised computer programs for machine- and deep learning, which exploit the power of SIMD and MIMD architectures such as CPUs and GPUs.

Scalars, Vectors, Matrices, Tensors I

Scalars A scalar is just a single number.

Example 1

$x = 3.5$. Here, x is a scalar; $x \in \mathbb{R}$.

Example 2

Similarly, Iris dataset has $d = 4$ features. Here, $d \in \mathbb{N}$ is a scalar.

Sometimes, we write a scalar as $(a)_{1\times 1}$.

Scalars, Vectors, Matrices, Tensors II

Vectors A vector is an array of number.

Example

$\mathbf{x} = [1.3, -4.0, 11.7, 4]^\mathsf{T}$ is a vector; $\mathbf{x} \in \mathbb{R}^4$.

A d-dimensional vector is written as $\mathbf{x} \in \mathbb{R}^d$ and in matrix form as: $(\cdots)_{d\times 1}$.

In a programming convention, we refer $(\cdots)_{d\times 1}$ as an 1-D array with d-elements.

Scalars, Vectors, Matrices, Tensors III

Matrices A matrix is a 2-D array of numbers.

Example

$\mathbf{A} = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 4 & 6 \\ 3 & 6 & 8 \end{bmatrix}$ is a 3×3 matrix.

We write a matrix as $\mathbf{A} \in \mathbb{R}^{m \times n}$.

We read this as: $\mathbf{A}$ is a real-valued matrix of order $m \times n$, where m is number of rows, n is number of columns.

An element of a matrix is referenced as $a_{i,j} \in \mathbf{A}$, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Scalars, Vectors, Matrices, Tensors IV

Tensors In some cases we will need an array with more than two axes (dimensions). A tensor is an array of numbers arranged on a regular grid with a variable number of axes.

Example

A photo that you clicked is stored as a tensor: (3-depth; RGB) or (4-depth; CMYK).

In an image, the first two axes are *height* and *width* of the image and the third axis refers to *depth* (no. of color channels).

Norms I

- Norm measures the size of a vector.
- Formally, L^p norm is given as

$$||\mathbf{x}||_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

  for $p \in \mathbb{R}$, $p \geq 1$.
- Norm is a function that maps a vector to a non-negative value.
- Intuitively, norm measures the distance of a vector $\mathbf{x}$ from origin.

Norms II

Norm is any function f that satisfies the following properties:

- $f(\mathbf{x}) = 0 \Rightarrow \mathbf{x} = \mathbf{0}$
- $f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ (*triangle inequality*)
- $\forall a \in \mathbb{R}, f(a\mathbf{x}) = |a| f(\mathbf{x})$

Norms III

Euclidean norm Most frequently, we will be using L^2 norm in ML. It is enoted as $||\mathbf{x}||$.

$$||\mathbf{x}|| = \mathbf{x}^{\mathsf{T}}\mathbf{x}$$

Norms IV

L^1 **norm** There will be situations in ML where difference between a zero and a non-zero element is very important. In such cases, we will use L^1 norm.

$$||\mathbf{x}||_1 = \sum_i |x_i|$$

i.e. every time an element x_i moves ϵ-away from 0, the norm increases by ϵ.

Norms V

Max norm It is called L^∞ norm, which is calculated as

$$||\mathbf{x}||_\infty = \max_i |x_i|$$

Norms VI

Frobenius norm It measures the size of a matrix.

This norm is used most frequently in deep learning.

$$||\mathbf{A}||_F = \left(\sum_{i,j} a_{i,j}^2 \right)^{\frac{1}{2}}$$

Norms VII

Relationship between dot product and norm The dot product of two vectors **x** and **y** can be written as

$$\mathbf{x}^\mathsf{T}\mathbf{y} = ||\mathbf{x}||_2||\mathbf{y}||_2 \cos\theta$$

where, θ is angle between the two vectors.

Parallelism I

(A sudden topic change from Linear Algebra to this. Surprised?)

- There are 4 different computing architectures (Flynn's taxonomy):
  1. Single Instruction, Single Data (SISD)
  2. Single Instruction, Multiple Data (SIMD)
  3. Multiple Instructions, Single Data (MISD)
  4. Multiple Instructions, Multiple Data (MIMD)

Parallelism II

- A multi-core processor is MIMD. A Graphics Processing Unit (GPU) is SIMD.
- Deep Learning is a problem for which SIMD is well-suited.
- Example: You want to compute non-linear activation of a matrix:
  1. Either you can call the `transform()` operation for each element. Total $m \times n$ calls.
  2. Or, call the `transform()` operation once for whole matrix.

Parallelism III

Given a matrix $\mathbf{Z}$, compute its non-linear activation $\sigma(\mathbf{Z})$.

1. Using explicit for loop:

```
A = np.zeros(Z.shape)
for i in range(0, Z.shape[0]):
for j in range(0, Z.shape[1]):
A[i][j] = 1 / (1 + np.exp(-Z[i][j]))
```

2. Using **vectorisation**:

```
A = 1 / (1 + np.exp(-Z))
```

Vectorisation I

Let $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{w} \in \mathbb{R}^d$, $b \in R$. We want to compute $z = \mathbf{w}^\mathsf{T}\mathbf{x} + b$.

1. Without vectorisation

```
z = 0.
for i in range(d):
z += W[i] * X[i]
z += b
```

2. With vectorisation

```
z = np.dot(W, X) + b
```

Vectorisation II

Computation time (64GB RAM, 16-core Intel Xeon CPU, 3.10GHz):

d	non-vec(s)	vec(s)
10^3	6.1×10^{-4}	2.6×10^{-5}
10^6	0.622	0.003
10^7	6.099	0.008
10^8	55.191	0.081
10^9	–	0.487

– : couldn't wait!

Vectorisation III

Vectorised matrix multiplication ($A, B \in \mathbb{R}^{d \times d}$). The same machine with 8GB GPGPU. Comparing CPU and GPU:

d	CPU(s)	GPU(s)
10^4	5.600	0.001

*Both results are obtained using PyTorch.