FROM LINEAR ALGEBRA TO MACHINE LEARNING

Omar Gutiérrez

September 13, 2018

@trinogz

Overview

MOTIVATION

Motivation

- **Linear algebra** is important to understand machine learning.
- As well as **calculus**, **probability theory**, and **statistics**.
- It is rewarding to take the **hard path** to learn machine learning (IMHO).

TENSORS

VECTORS - *RANK-1 TENSORS*

- A vector is a collection of numbers

$$\vec{a} = \boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$$

$$\begin{bmatrix} a_1 & a_2 & \vdots & a_n \end{bmatrix}$$

NumPy

```
# simple vector definition
a = np.array((5, 4, 3, 2, 1))
#  shape and size
a.shape
a.size
# zeros
np.zeros((5, 5))
# ones
np.ones((5, 5))
# matrix
np.matrix([[1, 0], [0, 1]])
```

Norms - L^p

$$||\boldsymbol{a}||_p = \left(\sum_{i=1}^{n} |a_i|^p \right)^{\frac{1}{p}}$$

NumPy

```
np.linalg.norm(a)
```

TensorFlow

```
tf.linalg.norm(a, ord=1)
tf.norm(a)
```

Length of a vector - L^2 norm

$$\|\mathbf{a}\|_2 = \left(\sum_{i=1}^{n} |a_i|^2 \right)^{\frac{1}{2}}$$

$$= \sqrt{\sum_{i=1}^{n} a_i^2}$$

NumPy

```
np.linalg.norm(a)
```

TensorFlow

```
tf.linalg.norm(a)
tf.norm(a)
```

DISTANCE BETWEEN VECTORS - L^2 NORM

$$d(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||$$
$$= \sqrt{\sum_{i=0}^{n} (a_i - b_i)^2}$$

NumPy

```
np.linalg.norm(a-b)
```

TensorFlow

```
# Manhattan
tf.norm(a-b, ord=1)
# Euclidean
tf.norm(a-b, ord="euclidean")
tf.norm(a-b, ord=2)
```

More distances

$$d(\boldsymbol{a}, \boldsymbol{b}) = ||\boldsymbol{a} - \boldsymbol{b}||$$
$$= \sqrt{\sum_{i=0}^{n} (a_i - b_i)^2}$$

NumPy

```
np.linalg.norm(a-b)
```

TensorFlow

```
# Manhattan
tf.norm(a-b, ord=1)
# Euclidean
tf.norm(a-b, ord="euclidean")
tf.norm(a-b, ord=2)
```

Dot product

$$a \cdot b = \sum_{i=0}^{n} a_i b_i$$

$$= a_0 b_0 + a_1 b_1 + \ldots + a_n b_n$$

NumPy

```
# do not confuse with np.multiply
np.dot(a, b)
# or with complex-conjugation
np.vdot(a, b)
# or
np.sum(np.multiply(a, b))
```

TensorFlow

```
tf.tensordot(a, b, 1)
# or
tf.matmul(tf.transpose(a), b)
# or
tf.matmul(a, b, transpose_a=True)
```

Dot product and norms

So,

$$\begin{aligned} \color{red}{a \cdot a} &= a_0 a_0 + a_1 a_1 + \ldots + a_n a_n \\ &= a_0^2 + a_1^2 + \ldots + a_n^2 \\ &= |\mathbf{a}|^2 \end{aligned}$$

```
np.linalg.norm(a) ** 2
# or
np.vdot(a, a)
```

Unit vectors

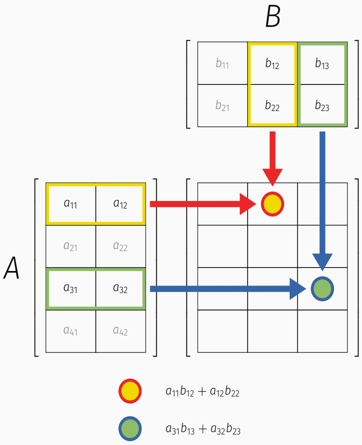
- ToDo.
- Normalization: $\frac{x}{||x||_2}$

Orthogonality

- ToDo.
- $\mathbf{x}^\top \mathbf{y} = 0$
- When two unit vectors are orthogonal are called **orthonormal**.

Matrix - rank-2 tensors

- Matrix multiplication is not commutative. $AB \neq BA$ in general.



NumPy

```
# matmul vs dot
np.matmul(a, b)
```

TensorFlow

```
tf.matmul(a, b)
```

More special matrices

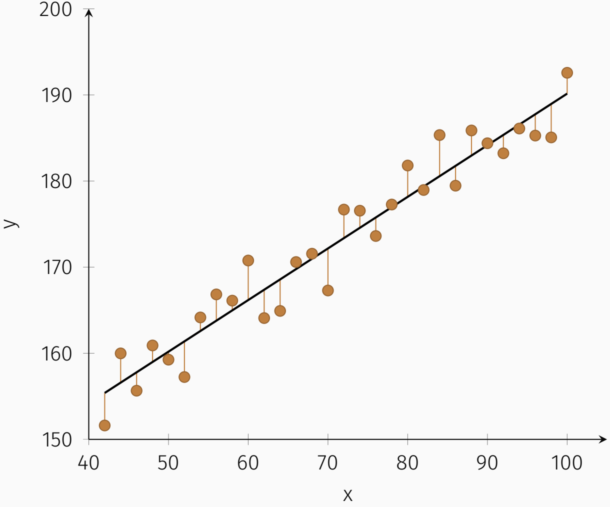
- ToDo
- Transpose
- Diagonal matrices
- Identity matrices
- Symmetric matrices
- Inverse matrix

Vector transformations

- ToDo
- Matrices are sometimes used to represent **vector transformations**.

EXAMPLES

Linear Regression



Linear Regression

- Linear regression is the Swiss Army Knife of Data Science.

$$\arg\min_{a,b} \sum_i (y_i - (ax_i + b))^2 = \arg\min_{\boldsymbol{w}} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||^2$$

- We want to calculate the w coefficients

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \dots \\ 1 & x_n \end{pmatrix} \bullet \begin{pmatrix} w_0 \\ w_1 \end{pmatrix}$$

Linear Regression

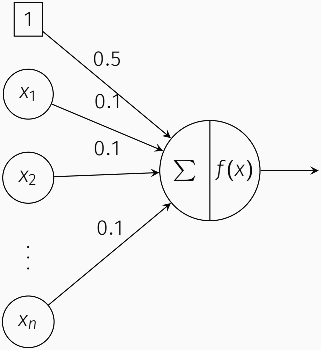
- In the simplest case we want to calculate the intercept a and the slope b.

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \dots \\ 1 & x_n \end{pmatrix} \bullet \begin{pmatrix} a \\ b \end{pmatrix}$$

- The solution to this optimization problem is:

$$w^* = (X^T X)^{-1} X^T y \, .$$

Simple perceptron



x_0	x_1	$\sum$	$f(x)$
1	1	$1 \times 0.5 + 1 \times -1 = -0.5$	0
1	0	$1 \times 0.5 + 0 \times -1 = 0.5$	1

Neural Networks

- ToDo.

Eigendecomposition

- This **matrix decomposition** technique is the base of **dimensionality reduction**.

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

- $\boldsymbol{A}$ (with dimension $N \times N$) is a square matrix
- $\boldsymbol{v}$ (with dimension N) is the eigenvector
- λ is a scalar value

Neural Networks

- ToDo.

CONCLUSIONS

More topics we should check

- Gradient descent is a beautiful optimization algorithm, basically, we multiply matrices to many times.
- Be aware that numerical instabilities can happen, and avoid these ones.

References

- Mathematics for Machine Learning: Linear Algebra by Coursera.
- The Math of Intelligence by Siraj Raval.
- Deep Learning Book by Bengio and Goodfellow, has a chapter summarizing which linear algebra topics you need to learn neural networks.

Thank you.
Questions?
Comments?