FROM LINEAR ALGEBRA TO MACHINE LEARNING

Omar Gutiérrez

September 4, 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

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

Length of a vector

$$|\mathbf{a}| = \sqrt{\sum_{i=0}^{n} a_i^2}$$

NumPy

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

TensorFlow

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

Distance between vectors

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

NumPy

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

TensorFlow

```
tf.norm(a-b, ord="euclidean")
```

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

So,

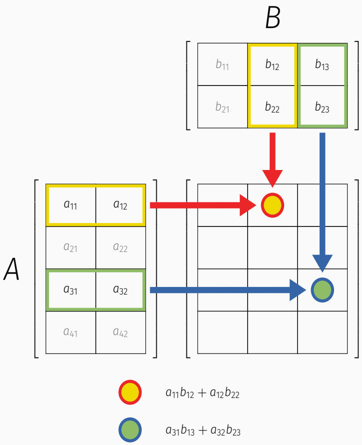
$$\begin{aligned} a \cdot a &= a_0 a_0 + a_1 a_1 + \ldots + a_n a_n \\ &= a_0^2 + a_1^2 + \ldots + a_n^2 \\ &= |\mathbf{a}|^2 \end{aligned}$$

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

Matrix - *rank-2 tensors*

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

Matrix - *rank-2 tensors*



NumPy

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

TensorFlow

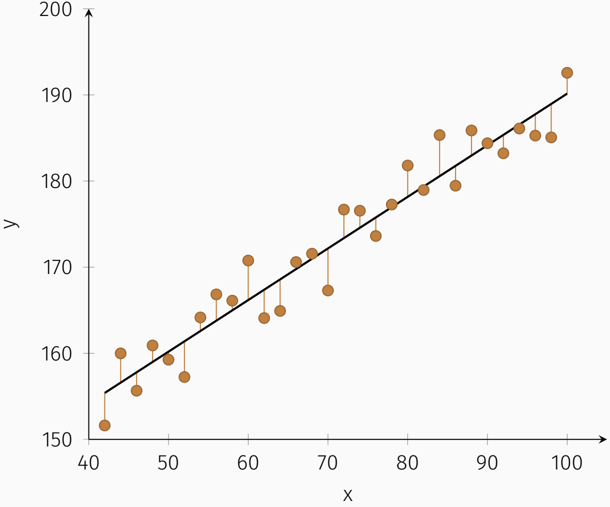
```
tf.matmul(a, b)
```

Matrices

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

EXAMPLES

Linear Regression



Linear Regression

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

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

Linear Regression

- 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*.

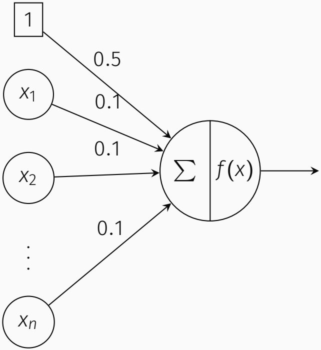
$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \dots \\ 1 & x_n \end{pmatrix} \bullet \begin{pmatrix} a \\ b \end{pmatrix}$$

Linear Regression

- 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.

CONCLUSIONS

More topics we should check

- Gradient descent is a beautiful optimization algorithm, basically, we multiply matrices to many times.
- Eigenvectors and eigenvalues; some dimensionality reduction techniques are based on eigendecomposition.
- 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?