IN3050 Mathgroup, Matrices and Vectors

Tobias Opsahl

March 9th, 2023

UNIVERSITY
OF OSLO

Outline

1. Notation
2. Computations
3. How we do it (at blackboard)

Matrix and vector notation

Why?

Here are two general reasons that we use matrix and vector notation:

1. The mathematical notation becomes simpler

Matrix and vector notation

Why?

Here are two general reasons that we use matrix and vector notation:

1. The mathematical notation becomes simpler
2. We can get big computational speedups

Motivation 1

Ease of notation

Writing sums can be both hard to read and messy. Introducing new notation can fix these issues.

Motivation 1

Ease of notation

Writing sums can be both hard to read and messy. Introducing new notation can fix these issues.
Here are some examples:

Motivation 1

Ease of notation

Writing sums can be both hard to read and messy. Introducing new notation can fix these issues.
Here are some examples:

$$\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_1 + \ldots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

Motivation 1

Ease of notation

Writing sums can be both hard to read and messy. Introducing new notation can fix these issues.
Here are some examples:

$$\mathbf{w} \cdot \mathbf{x} = w_1 x_1 + w_2 x_1 + \ldots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

$$\mathbf{W}\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & \ldots & w_{1,m} \\ w_{2,1} & w_{2,2} & \ldots & w_{2,m} \\ \vdots & \ldots & \ddots & \ldots \\ w_{n,1} & w_{n,2} & \ldots & w_{n,m} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} w_{1,1}x_1 + w_{1,2}x_2 \ldots w_{1,m}x_m \\ w_{2,1}x_1 + w_{2,2}x_2 \ldots w_{2,m}x_m \\ \vdots \\ w_{n,1}x_1 + w_{n,2}x_2 \ldots w_{n,m}x_m \end{bmatrix}$$

Motivation 1

Ease of notation 2

As you see, we can save a lot of space by just using the left sides.

Motivation 1

Ease of notation 2

As you see, we can save a lot of space by just using the left sides.

These operations, *dot-product*, *matrix-vector multiplication*, in addition to *matrix multiplication*, are used very often. We see that defining the operations can make it easier to read.

Motivation 2

Computational efficiency 1

Python allows us to write code very flexibly and with minimal syntax, but can be slow. Having computationally expensive operations inside python-loops or looping over long arrays is not recommended (it is slow)!

Motivation 2

Computational efficiency 1

Python allows us to write code very flexibly and with minimal syntax, but can be slow. Having computationally expensive operations inside python-loops or looping over long arrays is not recommended (it is slow)!

Question: Then why is so much important code that require large computations (machine-learning etc) written in Python?

Motivation 2

Computational efficiency 1

Python allows us to write code very flexibly and with minimal syntax, but can be slow. Having computationally expensive operations inside python-loops or looping over long arrays is not recommended (it is slow)!

Question: Then why is so much important code that require large computations (machine-learning etc) written in Python?

Answer: They use libraries (such as NumPy, Pandas, PyTorch or TensorFlow) that do not actually perform the computations in python. Usually it is done in a compiled language like C++, with many optimizations

Motivation 2

Computational efficiency 2

Consider the example:

```
long_list = [i for i in range(1000000)]
quadratic_list = []
for i in range(len(long_list)):
    quadratic_list.append(long_list[i] ** 2)
```

Versus:

```
long_list = np.arange(1000000)
quadratic_list = np.square(long_list)
```

Motivation 2

Computational efficiency 2

Consider the example:

```
long_list = [i for i in range(1000000)]
quadratic_list = []
for i in range(len(long_list)):
    quadratic_list.append(long_list[i] ** 2)
```

Versus:

```
long_list = np.arange(1000000)
quadratic_list = np.square(long_list)
```

- NumPy will automaticly perform a lot of optimization (running in C++, parallell computations, etc) and make the code much more efficient.

Motivation 2

Computational efficiency 2

Consider the example:

```
long_list = [i for i in range(1000000)]
quadratic_list = []
for i in range(len(long_list)):
    quadratic_list.append(long_list[i] ** 2)
```

Versus:

```
long_list = np.arange(1000000)
quadratic_list = np.square(long_list)
```

- NumPy will automaticly perform a lot of optimization (running in C++, parallell computations, etc) and make the code much more efficient.
- It can be both easier or harder to read and write the code.

Motivation 2

Computational efficiency 3

- In order for NumPy to do this *efficiently*, it needs all (or much) information *at once*.

Motivation 2

Computational efficiency 3

- In order for NumPy to do this *efficiently*, it needs all (or much) information *at once*.
- A loop would only give us one multiplication or sum to do at a time. If we ask NumPy to multiply ararys and matrices, we can give it as many dimensions to sum over as we want.

Motivation 2

Computational efficiency 3

- In order for NumPy to do this *efficiently*, it needs all (or much) information *at once*.
- A loop would only give us one multiplication or sum to do at a time. If we ask NumPy to multiply ararys and matrices, we can give it as many dimensions to sum over as we want.
- This is further emphasized in many uses of *deep learning*, where often have 5-dimensional arrays, (row, col, channels, kernels, batch-size).

Vector and Matrix notation

How?

Let us actually do it!

I will use the blackboard to show you.

Let's use the perceptron as an example.