UiO : University of Oslo

institutt for informatikk

IN3050/IN4050 - Introduction to Artificial Intelligence and Machine Learning

Background A: Vectors and Matrices

Jan Tore Lønning

A.1 Vectors

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

In addition: Vectors, matrices, NumPy

- Efficient code: both writing and execution
  - A@B can replace three nested loops
  - GPUs – parallel processing
- NumPy:
  - Based on vectors and matrices
  - Used by Marsland
  - Libraries for ML, including Deep Learning
- Necessary for a deeper understanding
  - in particular, of complex neural networks
    - Tensor generalizes vectors and matrices

Vectors

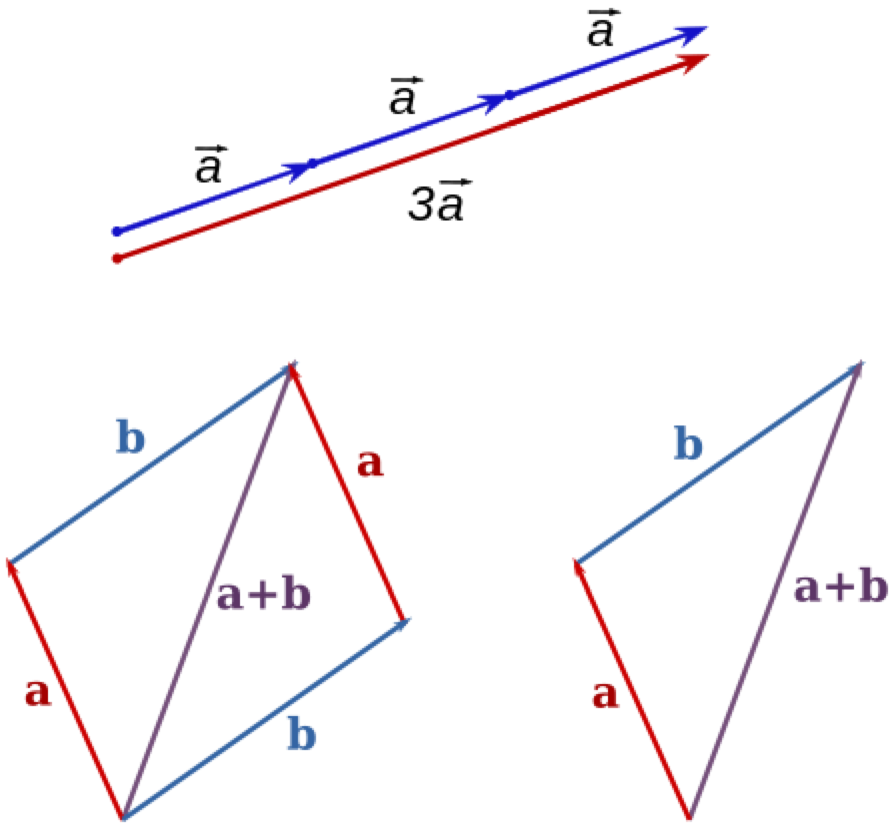
- An n-dimensional vector is an array of n scalars (real numbers)
  - $(x_1, x_2, \dots x_n)$

- Two operations on vectors
  - Scalar multiplication
  - $a(x_1, x_2, \dots x_n) = (ax_1, ax_2, \dots ax_n)$

  - Addition
  - $(x_1, x_2, \dots x_n) + (y_1, y_2, \dots y_n) = (x_1 + y_1, x_2 + y_2, \dots x_n + y_n)$

Euclidean vectors

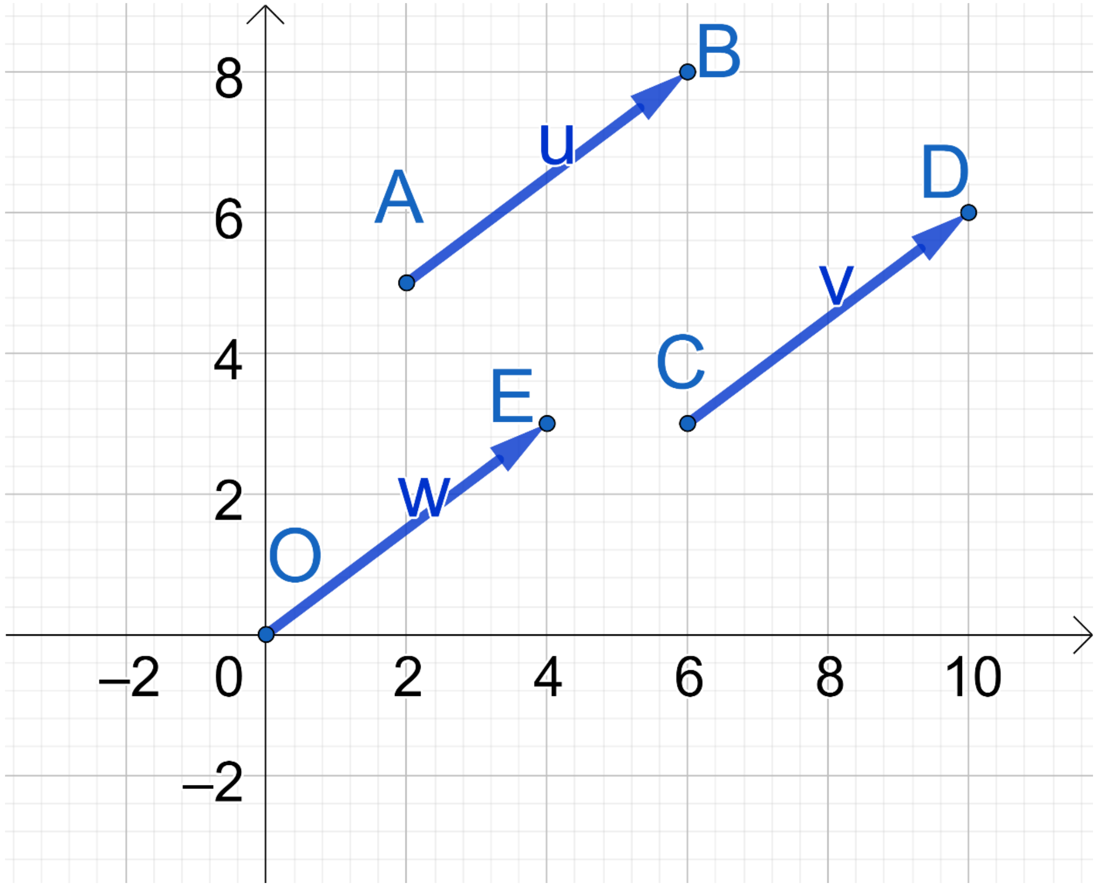
- Also called geometric or spatial vectors
- 2D or 3D
- Characterized by
  - length
  - direction
- Used in physics for e.g.
  - forces, speed, acceleration, etc.



Figures from Wikipedia

The connection

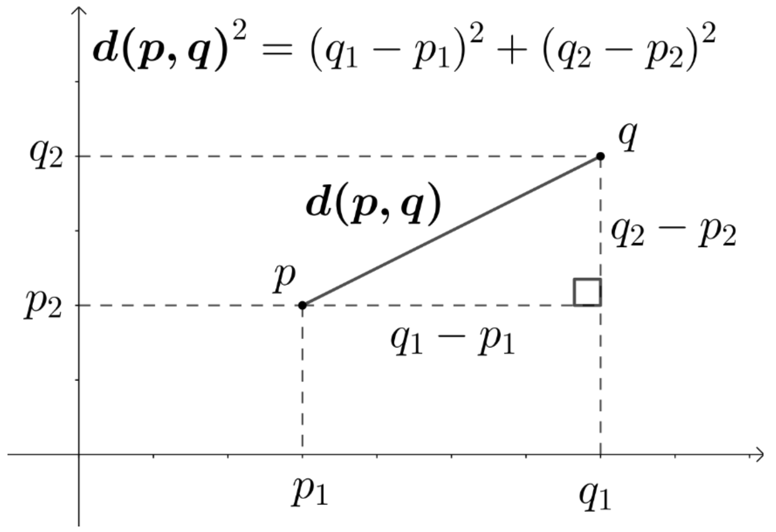
- Vectors with the same length and direction are considered equivalent
- A vector can be described by
  - start- and end-point
    - $\boldsymbol{u} = (A, B) = \big((2,5), (6,8)\big)$
    - $\boldsymbol{w} = \big((0,0), (4,3)\big)$
  - end-point
    - $\boldsymbol{w} = E = (4,3)$
    - the numeric form we use for addition and scalar multiplication



Norm of a vector

The norm (length) of a vector

- $\|(x_1, x_2, \dots x_n)\| = \sqrt{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}$
- This is called L2-norm

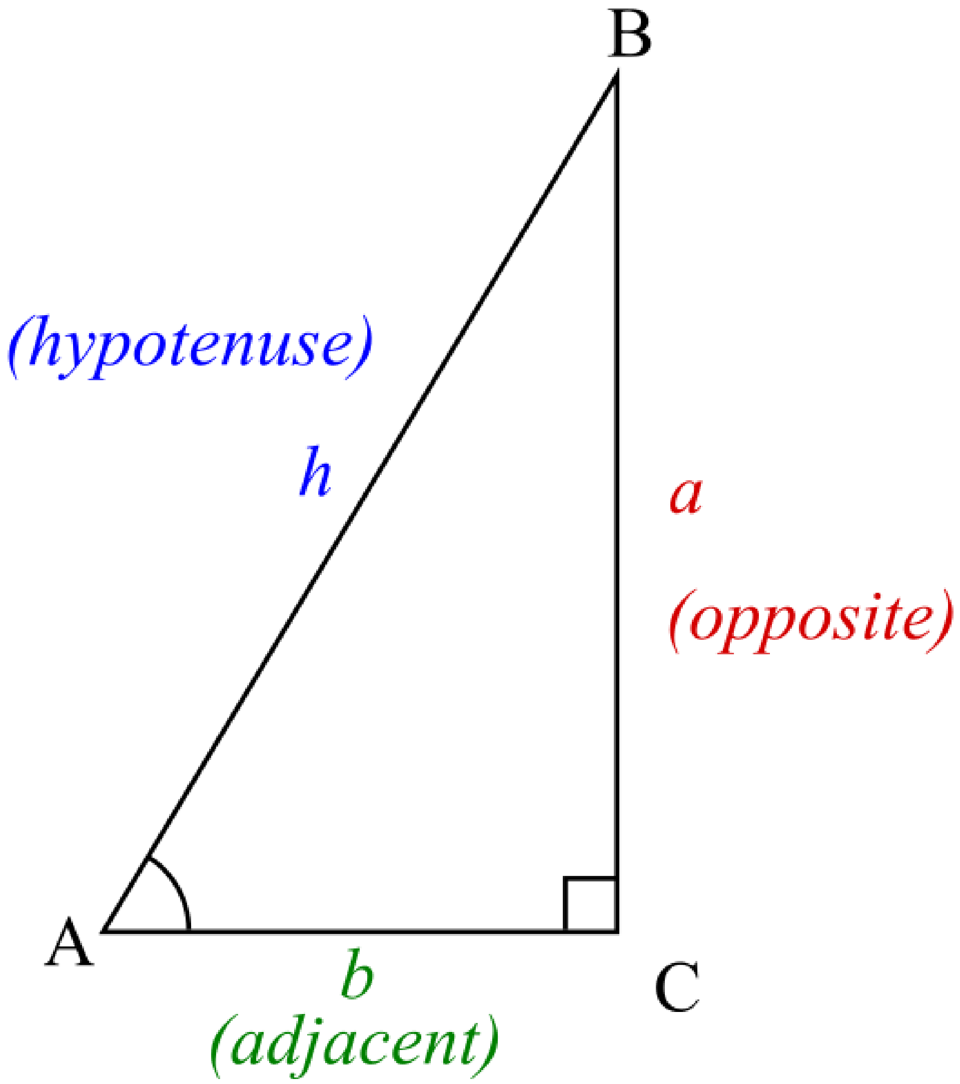


Possible to operate with other norms, e.g., L1-norm ("Manhattan")

- $\|(x_1, x_2, \dots x_n)\|_1 = |x_1| + |x_1|$+…+$|x_n|$
- used in machine learning e.g., for regularization

Cosine

- $\cos(A) = \frac{b}{h}$
- $\sin(A) = \frac{a}{h}$

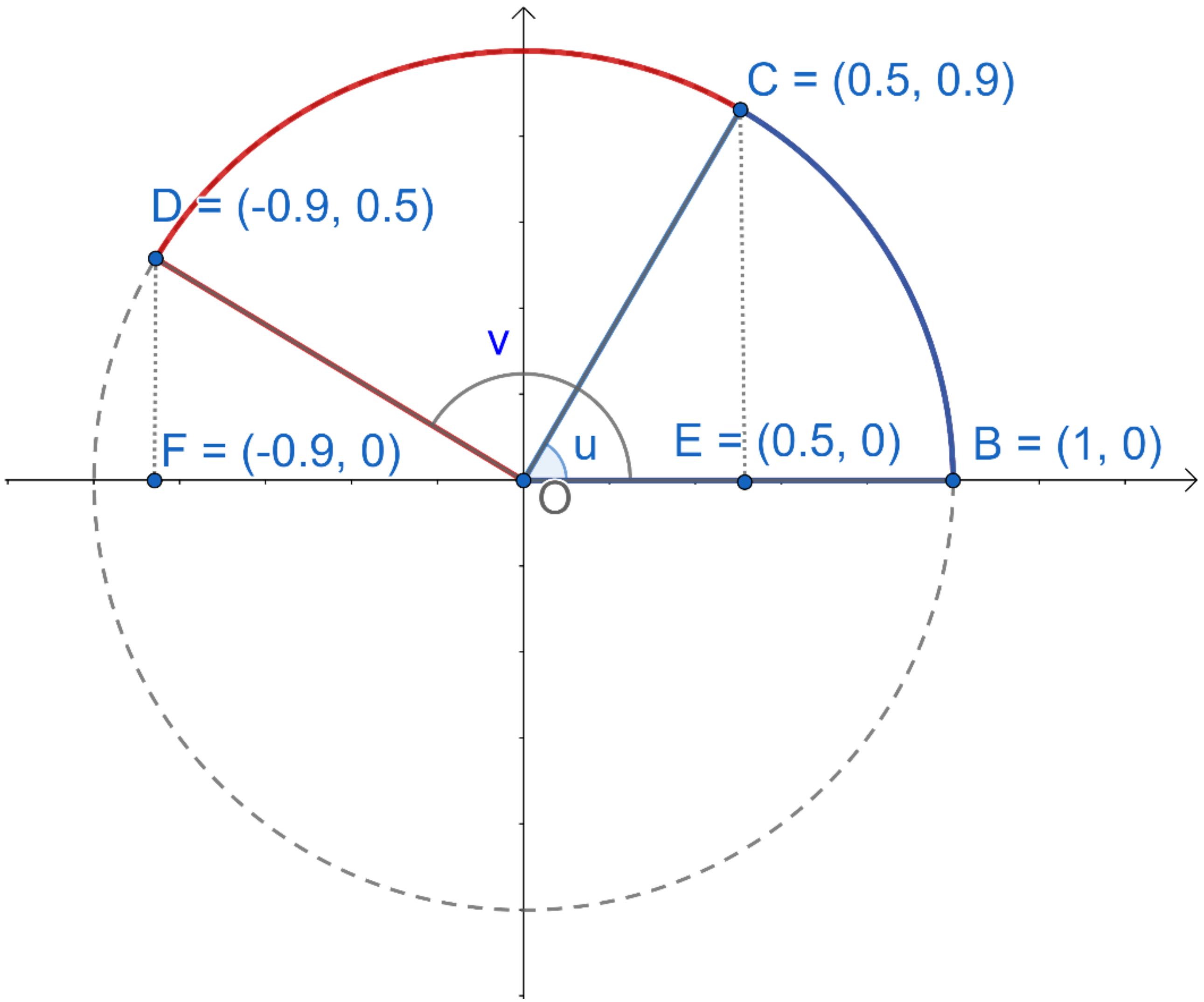


Cosine

Also defined for obtuse (non-acute) angles:

- $\cos(u) = C_1 = 0.5$
- $\cos(v) = D_1 =$

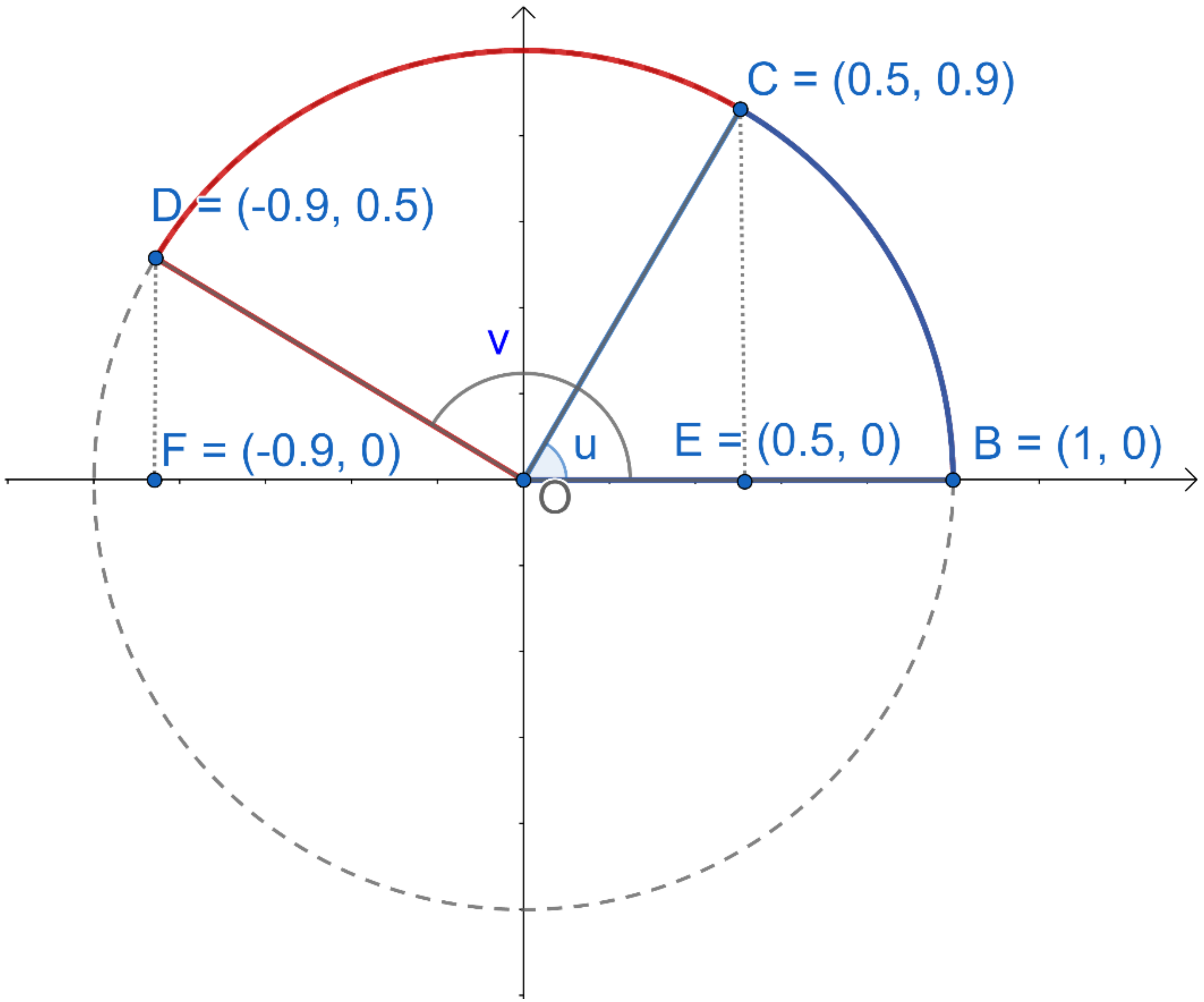
$$\sqrt{1 - 0.5^2} \approx -0.9$$



Cosine

Observations:

- $\cos(0) = 1$
- $\cos(u) = 0$ iff $u = \frac{\pi}{2} = 90^{\circ}$
- $0 < \cos(u) < 1$ iff $0 < u < \frac{\pi}{2}$
- $\cos(u) < 0$ iff $\frac{\pi}{2} < u \leq \pi$

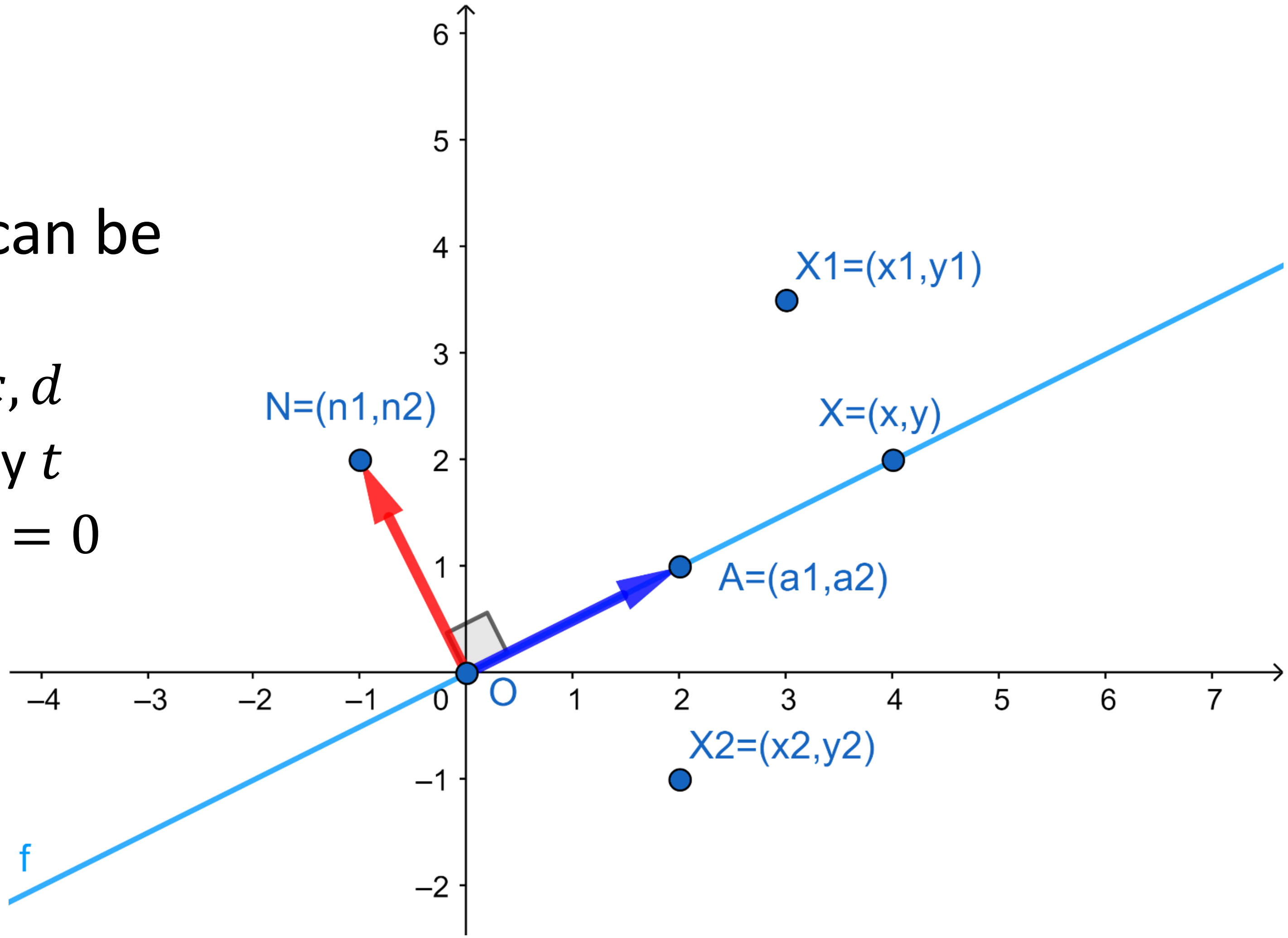


Dot product

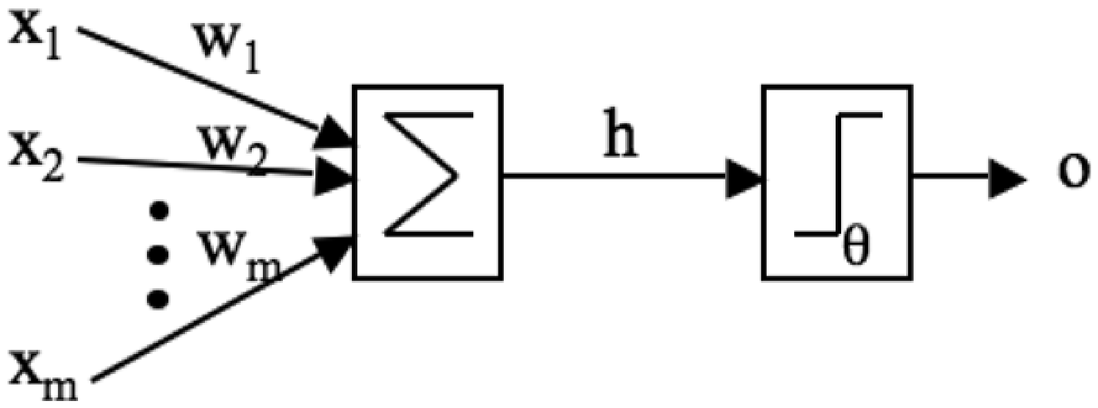
- $(x_1, x_2, \dots x_n) \cdot (y_1, y_2, \dots y_n) = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i$
- This is a scalar (real number) – not a vector
- $\boldsymbol{x} \cdot \boldsymbol{y} = \|\boldsymbol{x}\| \|\boldsymbol{y}\| \cos(u)$ where u is the angle between the two vectors
- $\cos(u) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|}$
- In 2D and 3D we can prove this
- In higher dimensions, we can use this to define cosine
  - and show that cosine gets the expected properties

Lines and vectors

- A line through the origin can be defined:
  1. $cx + dy = 0$, for some c, d
  2. $(x, y) = t(a_1, a_2)$ for any t
  3. $X \cdot N = (x, y) \cdot (n_1, n_2) = 0$
     - $n_1 = c, n_2 = d$
- Observe that
  - $X_1 \cdot N > 0$
  - $X_2 \cdot N < 0$



Linear classification



Last week

1. An adder (including bias) :

$$h = \sum_{i=0}^{m} w_i x_i$$
$$= w_0 x_0 + w_2 x_2 + \cdots + w_m x_m$$

1. An activation function,

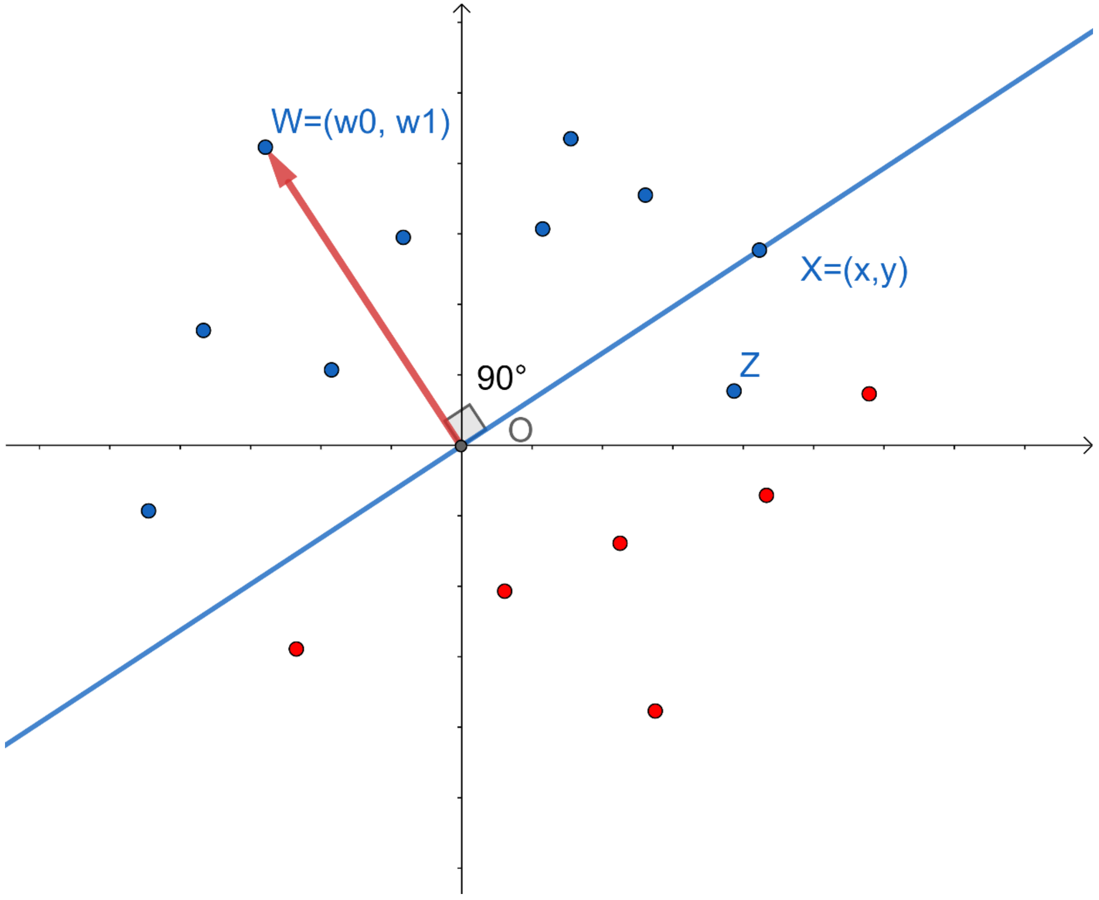
   Predict

$$o = g(h) = \begin{cases} 1\ if\ h > 0 \\ 0\ if\ h\ \leq 0 \end{cases}$$

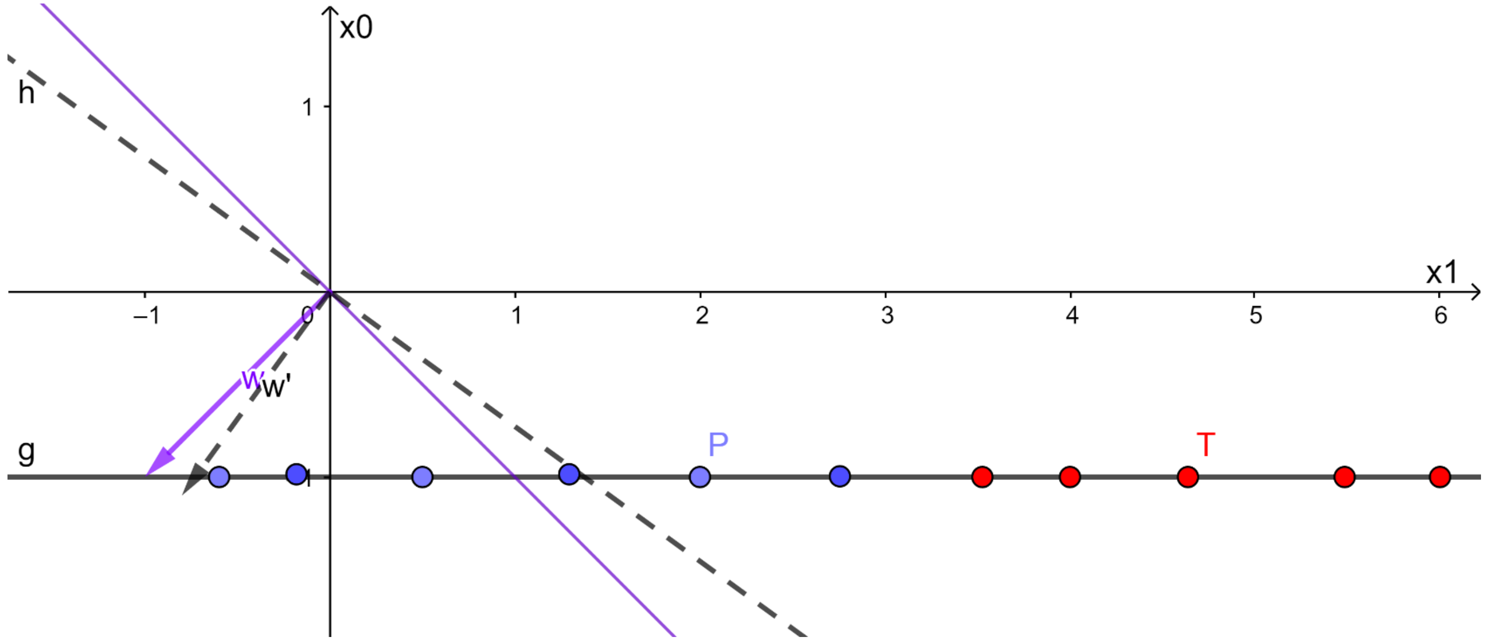
- The weights can be considered a vector $\boldsymbol{w} = (w_0, , \ldots , w_m)$
- Adding as dot product $h = \sum_{i=0}^{m} w_i x_i = \boldsymbol{w} \cdot \boldsymbol{x}$
- Predict
  - 1 iff $0 < \angle(\boldsymbol{w}, \boldsymbol{x}) < \frac{\pi}{2}$
  - Otherwise: zero

Perceptron update

- Point Z gets wrong class
- When updating for Z, we add a small vector pointing in the direction of Z to W
- Hence, we tilt the decision boundary line towards Z



Example



- The example from the perceptron algorithm
- Positive class $g(h) = 1$ iff
- $w_1x_1 + w_0x_0 = (w_0, w_1) \cdot (x_0, x_1) > 0$
- Initial vector: $\mathbf{w} = (w_1, w_0)$=(-1,-1)
- Updated vector: $\mathbf{w}' = (w_1, w_0) = (-0.8, -1.1)$

Vectors in NumPy

- Vectors
  - In [1]: import NumPy as np
  - In [2]: a = np.array([1,2,3])
  - In [3]: a
  - Out[3]: array([1, 2, 3])
- Scalar multiplication
  - In [7]: c = 5.0
  - In [8]: c*a
  - Out[8]: array([5., 10., 15.])
- Vector addition:
  - In [4]: b = np.array((4.5, 6, 7))
  - In [5]: b
  - Out[5]: array([4.5, 6. , 7.])
  - In [6]: a+b
  - Out[6]: array([5.5, 8. , 10.])

Dot-product in NumPy

- Three ways:
  - `np.dot(a,b)`
  - `a.dot(b)`
  - `a @ b`
- @ is most readable for complex expressions

Implementing the forward step

Pure python implementation

- *x* and *weights* as lists (or tuples)
-
```
forward = sum([self.weights[i]*x[i]
              for i in range(self.dim)])
```

NumPy-implementation

- *x* and *weights* as NumPy-arrays
- `forward = self.weights @ x`

The perceptron update step

Pure python implementation

- ```
  for i in range(dim):
      weights[i] += eta * (t - y) * x[i]
  ```

NumPy-implementation

- `weights += eta * (t - y) * x`
    - *x* and *weights* as NumPy-arrays
    - *eta, t, y* as scalars (floats)
  ```

For more

- See
- *Geometry and linear algebra for IN3050/IN4050*
- Next: Matrices

A.2 Matrices

IN3050/IN4050 Introduction to Artificial Intelligence and Machine Learning

MATRIX

Matrix

- A rectangular array of numbers
  - *m* rows
  - *n* columns
  - A $m \times n$ –matrix ("*m by n*")

$$
\begin{array}{c|cccc|}
 & 1 & 2 & \dots & n \\
\hline
1 & a_{11} & a_{12} & \dots & a_{1n} \\
2 & a_{21} & a_{22} & \dots & a_{2n} \\
3 & a_{31} & a_{32} & \dots & a_{3n} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
m & a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}
$$

(In programming, e.g., Python and NumPy, we typically count from 0 to n-1)

Matrix operations

- Addition: $\begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$+$\begin{bmatrix} 11 & 22 & 33 \\ 21 & 22 & 23 \end{bmatrix}$=$\begin{bmatrix} 22 & 34 & 46 \\ 42 & 44 & 46 \end{bmatrix}$
- Multiplication by scalars $5B = 5 \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} = \begin{bmatrix} 55 & 60 & 65 \\ 105 & 110 & 115 \end{bmatrix}$

Transposed

- If $B = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$,

the transposed of B is

- $B^T = \begin{bmatrix} 11 & 21 \\ 12 & 22 \\ 13 & 23 \end{bmatrix}$

- Interchanges rows and columns

Notation

- Alternative notation for the element (a scalar) in row *i* and column *j* of matrix A:
  - $a_{i,j}$
  - $A_{i,j}$
  - $A[i,j]$
- The last two are useful for multiplication:
  - $(AB)_{i,j}$
  - $(AB)[i,j]$

$$\begin{array}{cc} & \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

https://en.wikipedia.org/wiki/Matrix_(mathematics)

Notation 2

- We can use $A[i,:]$ for the vector consisting of the elements in row i:
  - $A[i,:] = (a_{i,1}, a_{i,2}, \dots, a_{i,n})$
- $A[:,j]$ for the vector consisting of the elements in column j:
  - $A[:,j] = (a_{1,j}, a_{2,j}, \dots, a_{m,j})$

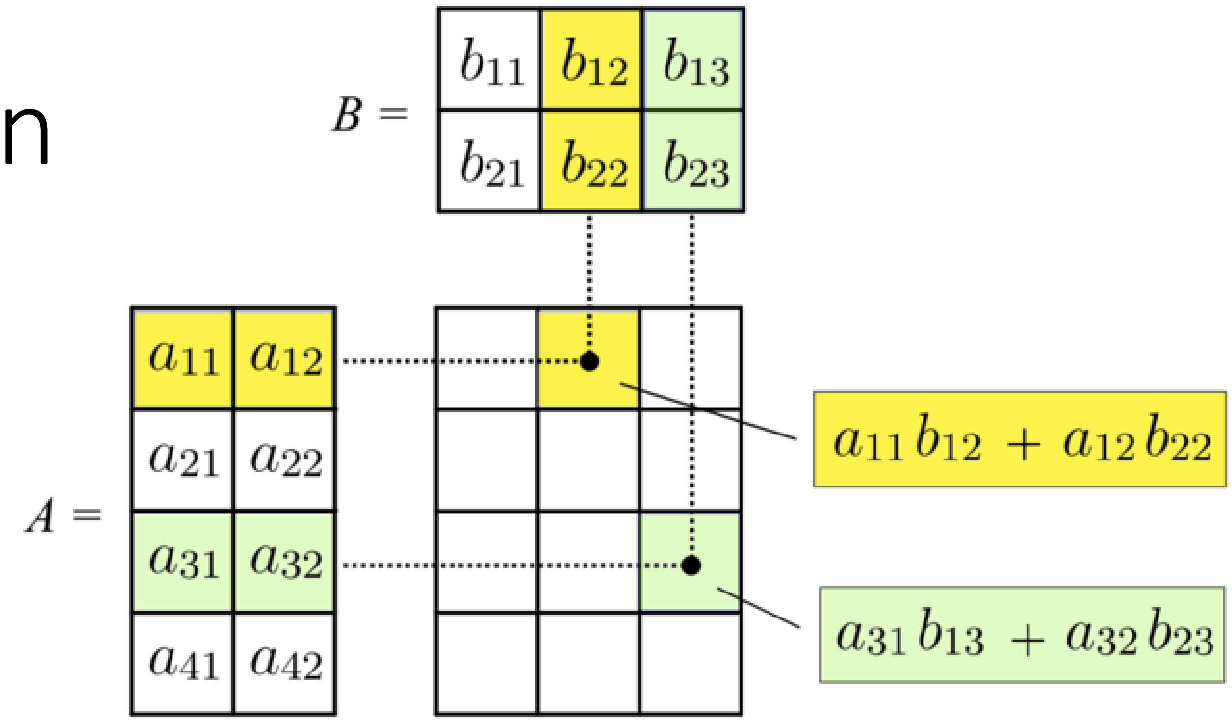
$$
\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & \ldots & n \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} &
\begin{bmatrix}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
a_{31} & a_{32} & \ldots & a_{3n} \\
\vdots & \vdots & \vdots & \vdots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{bmatrix}
\end{array}
$$

https://en.wikipedia.org/wiki/Matrix_(mathematics)

Matrix multiplication

- If
  - A is a $m \times n$ matrix
  - B is a $n \times p$ matrix
- Define the product $C = AB$
  - A $m \times p$ matrix, where
  - $c_{i,j} = \sum_{r=1}^{n} a_{i,r} b_{r,j}$

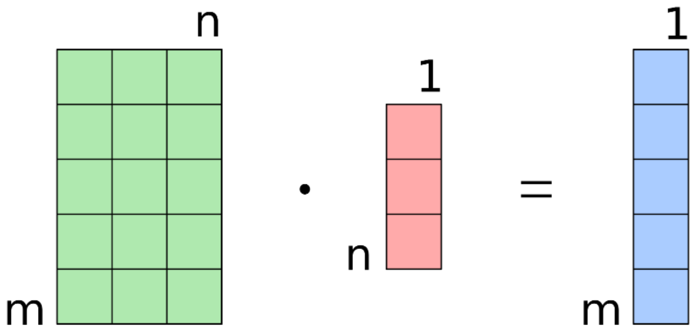
    $= A[i,:] \cdot B[:,j]$

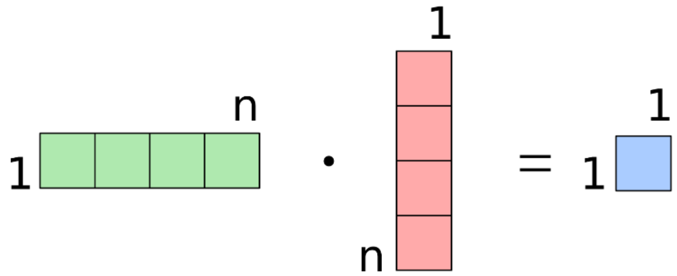


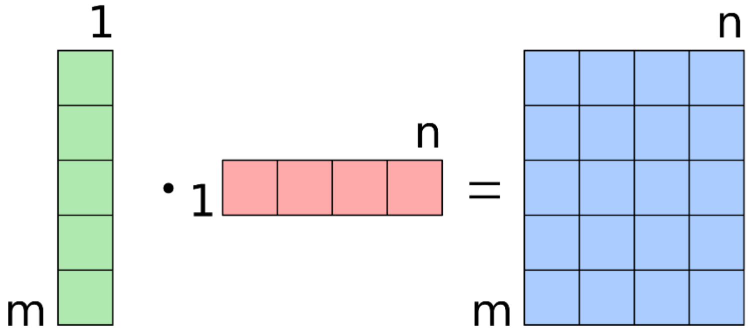
https://en.wikipedia.org/wiki/Matrix_(mathematics)

Don't use · for matrix multiplication
Write AB
Not $A \cdot B$

Product dimensions (but don’t use the dot)







Column vectors

- A column vector is a nx1 matrix, e.g., $C = \begin{bmatrix} -1 \\ 2 \\ 4 \end{bmatrix}$
- It is not a vector
- It can sometimes be convenient to use the column vector to represent the vector
  - $C[:,1] = (-1,2,4)$
  - This can simplify operations, reducing them to matrix multiplication
  - Some books just take vectors to be column vectors
  - But when we program e.g., in Python, we should distinguish between the $n \times 1$ matrix C and the n-dimensional vector it represents $C[:,1]$

Marsland's representation

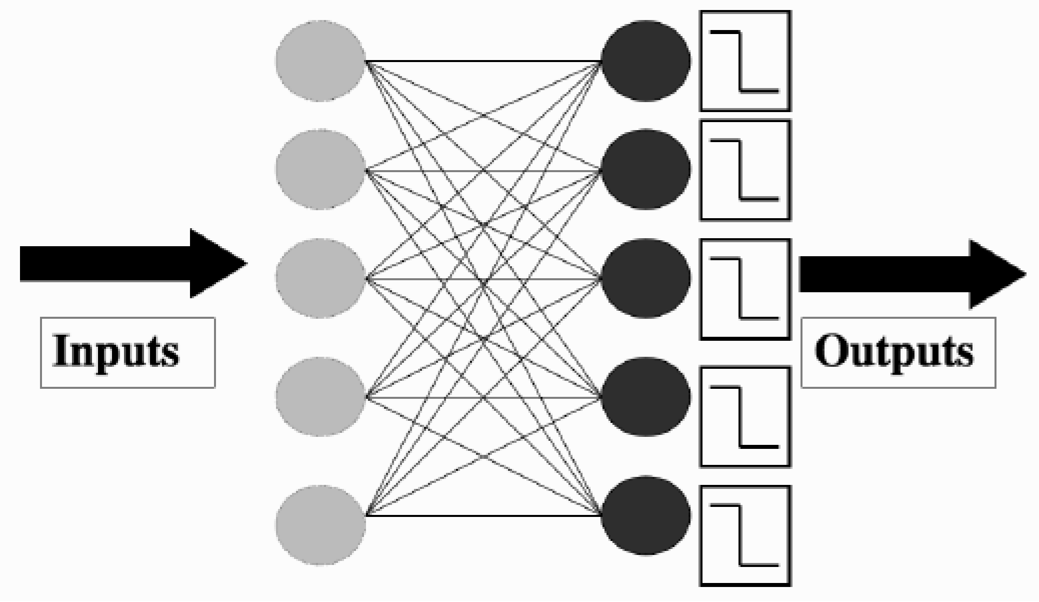
$$\overset{\color{red}{X}}{\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix}} \overset{\color{red}{W}}{\begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{m,1} \end{bmatrix}} = \overset{\color{red}{Y}}{\begin{bmatrix} y_{1,1} \\ y_{2,1} \\ \vdots \\ y_{N,1} \end{bmatrix}} \sim \overset{\color{red}{T}}{\begin{bmatrix} t_{1,1} \\ t_{2,1} \\ \vdots \\ t_{N,1} \end{bmatrix}}$$

- Each row represent the vector of one data point
- $X[i,:] = \boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,m})$
- Each datapoint has m many features
- There are N many datapoints
  - (input vectors)

- The weight vector $\boldsymbol{w}$ represented by a column vector, W:
- $W[:,1] = \boldsymbol{w} = (w_{1,1}, w_{2,1}, \dots, w_{m,1})$
- Use matrix multiplication to calculate forward for all datapoints in one go.
- $Y[i,1] = y_{i,1} = \boldsymbol{x}_i \cdot \boldsymbol{w}$

Vector output

- Sometimes the target value to an input vector $(x_1, x_2, \dots, x_m)$ is a vector $(y_1, y_2, \dots, y_n)$
- Then the weights can be represented by matrix $m \times n$



$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N,1} & y_{N,2} & \cdots & y_{N,n} \end{bmatrix}$$

Matrices in NumPy

```
In [3]: a =
np.array([[11,12,13,
[21,22,23]])

In [4]: a

Out[4]:

array([[11, 12, 13],

       [21, 22, 23]])


In [5]: a.shape

Out[5]: (2, 3)
```

```
In [6]: a.T

Out[6]:

array([[11, 21],

       [12, 22],

       [13, 23]])
```

```
In [8]: c

Out[8]: array(
[0, 1, 2, 3, 4, 5, 6,
7, 8, 9, 10, 11])


In [9]:
d=c.reshape(3,4)

In [10]: d

Out[10]:

array([[ 0, 1, 2, 3],

       [ 4, 5, 6, 7],

       [ 8,9,10,11]])
```

Matrix multiplication in NumPy

```
In [4]: a
Out[4]:
array([[11, 12, 13],
       [21, 22, 23]])

In [10]: d
Out[10]:
array([[ 0, 1, 2, 3],
        [ 4, 5, 6, 7],
        [ 8,9,10,11]])
```

- `In [12]: a @ d`
- `Out[12]:`
- `array([[152, 188, 224, 260],`
- `       [272, 338, 404, 470]])`

For more

- See *Geometry and linear algebra for IN3050/IN4050*
- Practice using NumPy