Applied Math for Deep Learning

Prof. Kuan-Ting Lai

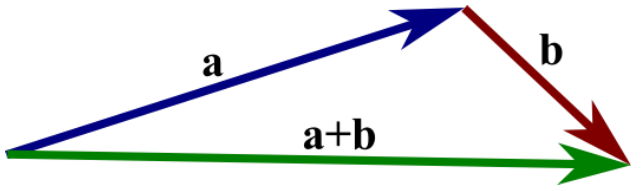
2021/3/8

Applied Math for Deep Learning

- Linear Algebra
- Probability
- Calculus
- Optimization

Linear Algebra

- Scalar
  - real numbers
- Vector (1D)
  - Has a magnitude & a direction



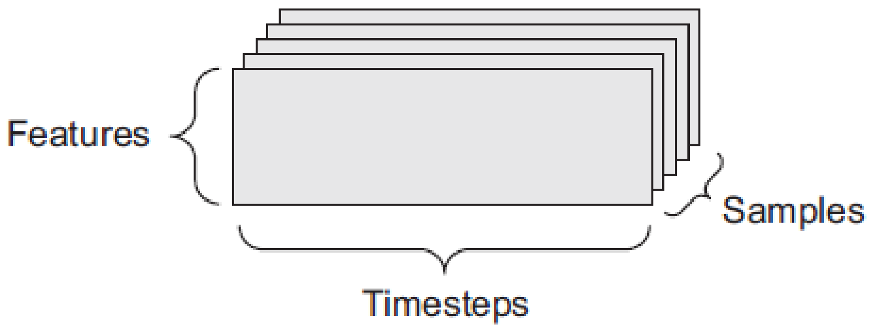
- Matrix (2D)
  - An array of numbers arranges in rows & columns

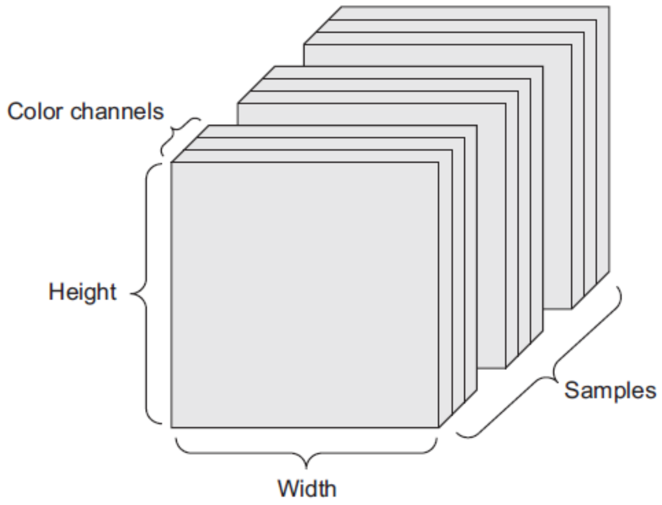
$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

- Tensor (>=3D)
  - Multi-dimensional arrays of numbers

Real-world examples of Data Tensors

- Timeseries Data – 3D (samples, timesteps, features)
- Images – 4D (samples, height, width, channels)
- Video – 5D (samples, frames, height, width, channels)





Vector Dimension vs. Tensor Dimension

- The number of data in a vector is also called “dimension”
- In deep learning , the dimension of Tensor is also called “rank”
- Matrix = 2d array = 2d tensor = rank 2 tensor
- Axis means the specific dimension of a Tensor

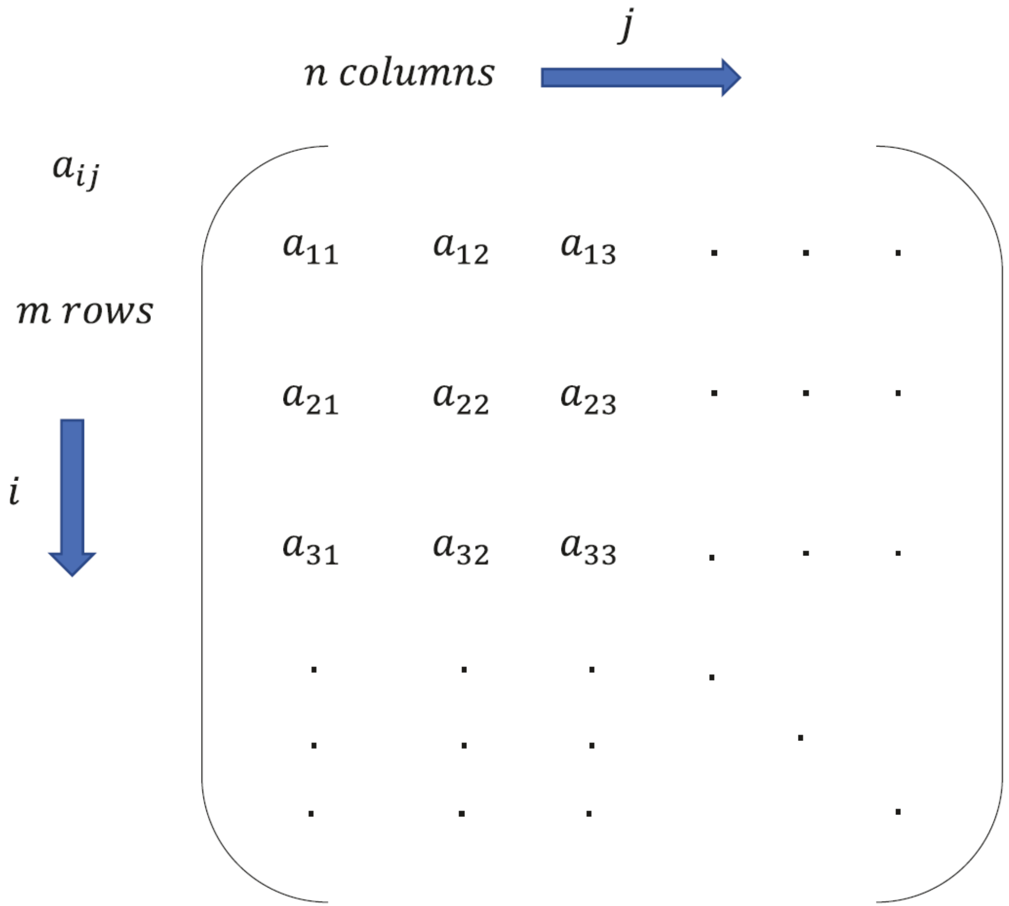
https://deeplizard.com/learn/video/AiyK0idr4uM

The Matrix

Matrix

- Define a matrix with m rows and n columns:

$$A_{m\times n} \in \mathbb{R}^{m\times n}$$



Santanu Pattanayak, ”Pro Deep Learning with TensorFlow,” Apress, 2017

Matrix Operations

- Addition and Subtraction

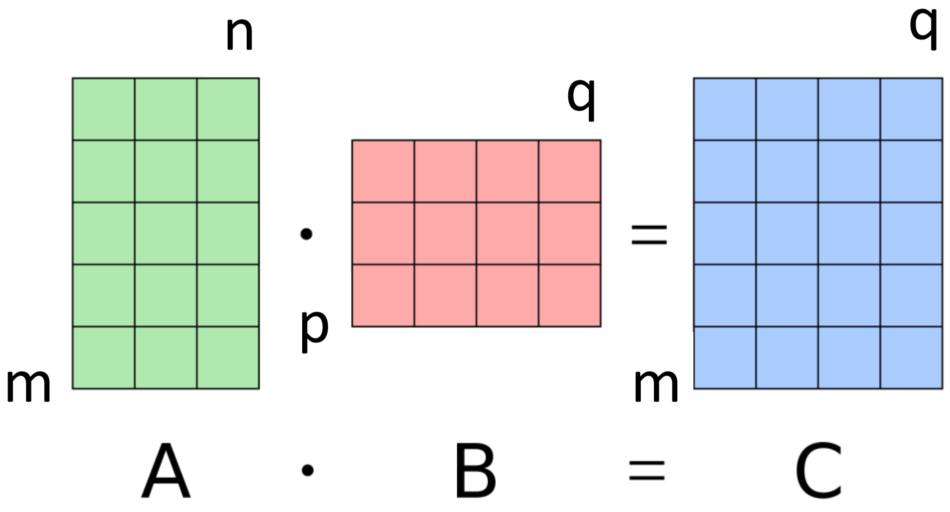
$$A=\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}\quad B=\begin{bmatrix}5 & 6\\7 & 8\end{bmatrix}$$

$$A+B=\begin{bmatrix}1+5 & 2+6\\3+7 & 4+8\end{bmatrix}=\begin{bmatrix}6 & 8\\10 & 12\end{bmatrix}$$

$$A-B=\begin{bmatrix}1-5 & 2-6\\3-7 & 4-8\end{bmatrix}=\begin{bmatrix}-4 & -4\\-4 & -4\end{bmatrix}$$

Matrix Multiplication

- Two matrices A and B, where $A \in \mathbb{R}^{m \times n}$ $B \in \mathbb{R}^{p \times q}$
- The columns of A must be equal to the rows of B, i.e. n == p
- A * B = C, where $C \in \mathbb{R}^{m \times q}$
- $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$



Example of Matrix Multiplication (3-1)

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

https://www.mathsisfun.com/algebra/matrix-multiplying.html

Example of Matrix Multiplication (3-2)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

https://www.mathsisfun.com/algebra/matrix-multiplying.html

Example of Matrix Multiplication (3-3)

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \checkmark$$

https://www.mathsisfun.com/algebra/matrix-multiplying.html

Matrix Transpose

$$A \in \mathbb{R}^{m \times n} \qquad A^{\mathrm{T}} \in \mathbb{R}^{n \times m}$$

$$a'_{ji} = a_{ij} \quad \forall\, i \in \{1,2,..m\}, \forall\, j \in \{1,2,..n\}$$

$$\mathbf{A} \qquad \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow \mathbf{A}^{\mathrm{T}} \qquad \boxed{\begin{matrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{matrix}}$$

https://en.wikipedia.org/wiki/Transpose

Dot Product

- Dot product of two vectors become a **scalar**
- Notation: $v_1 \cdot v_2$ or $v_1{}^T v_2$

$$v_1 = \begin{bmatrix} v_{11} \\ v_{12} \\ . \\ . \\ . \\ v_{1n} \end{bmatrix} v_2 = \begin{bmatrix} v_{21} \\ v_{22} \\ . \\ . \\ . \\ v_{2n} \end{bmatrix}$$

$$v_1 . v_2 = v_1{}^T v_2 = v_2{}^T v_1 = v_{11} v_{21} + v_{12} v_{22} + .. + v_{1n} v_{2n} = \sum_{k=1}^{n} v_{1k} v_{2k}$$

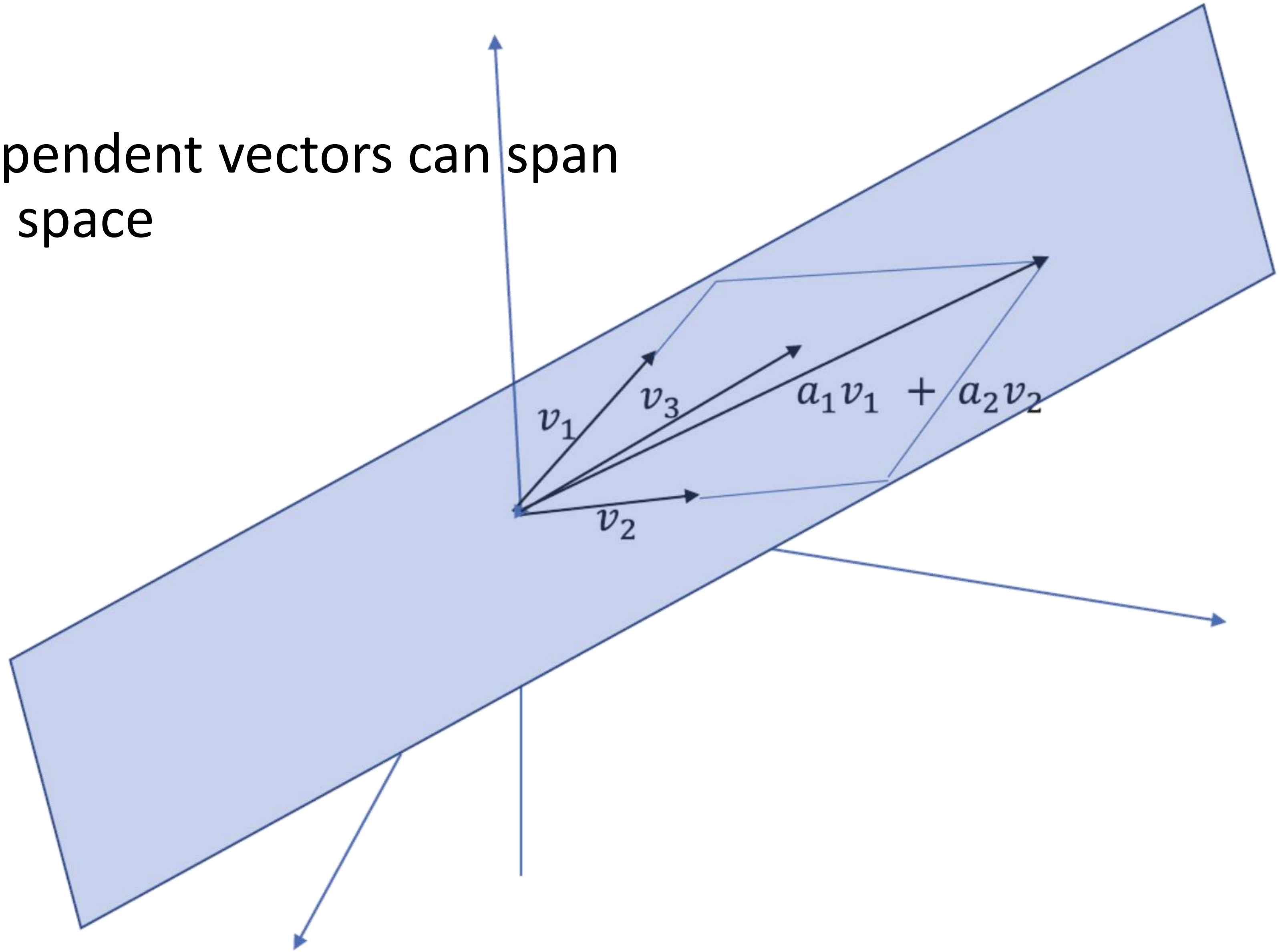
Linear Independence

- A vector is **linearly dependent** on other vectors if it can be expressed as the linear combination of other vectors
- A set of vectors $v_1, v_2, \cdots, v_n$ is **linearly independent** if $a_1 v_1 + a_2 v_2 + \cdots + a_n v_n = 0$ implies all $a_i = 0, \forall i \in \{1,2,\cdots n\}$

$$[v_1 v_2 \ldots . v_n] \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ a_n \end{bmatrix} = 0 \; where \;\; v_i \in \mathbb{R}^{m \times 1} \; \forall i \in \{1,2,\ldots,n\}, \begin{bmatrix} a_1 \\ a_2 \\ . \\ . \\ a_n \end{bmatrix} \in \mathbb{R}^{n \times 1}$$

Span the Vector Space

- *n* linearly independent vectors can span *n*-dimensional space



Rank of a Matrix

- Rank is:
  - The number of linearly independent row or column vectors
  - The dimension of the vector space generated by its columns
- Row rank = Column rank
- Example:

$$A = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} \xrightarrow{\text{Row-echelon form}} \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

https://en.wikipedia.org/wiki/Rank_(linear_algebra)

Identity Matrix I

- Any vector or matrix multiplied by I remains unchanged
- For a matrix $A_{m \times n}$, $AI_n = I_m A = A$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 3}$$

$$Iv = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

Inverse of a Matrix

- The product of a **square** matrix A and its inverse matrix A^{-1} produces the identity matrix I
- $AA^{-1} = A^{-1}A = I$
- Inverse matrix is square, but not all square matrices has inverses

Pseudo Inverse

- Non-square matrix and have left-inverse or right-inverse matrix
- Example:

$$Ax = b, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^n$$

  – Create a square matrix $A^T A$

$$A^T A x = A^T b$$

  – Multiplied both sides by inverse matrix $(A^T A)^{-1}$

$$x = (A^T A)^{-1} A^T b$$

  – $(A^T A)^{-1} A^T$ is the pseudo inverse function

Norm

- Norm is a measure of a vector's magnitude
- l_2 norm

$$\|x\|_2 = \left(|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2\right)^{1/2} = (x.x)^{1/2} = \left(x^T x\right)^{1/2}$$

- l_1 norm

$$\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$$

- l_p norm

$$\left(|x_1|^p + |x_2|^p + \ldots + |x_n|^p\right)^{1/p}$$

- l_∞ norm

$$\lim_{p\to\infty} \|x\|_p = \lim_{p\to\infty}\left(|x_1|^p + |x_2|^p + \ldots + |x_n|^p\right)^{1/p} = max\left(x_1, x_2, \ldots., x_n\right)$$

Eigen Vectors

- Eigenvector is a non-zero vector that changed by only a scalar factor λ when linear transform A is applied to:

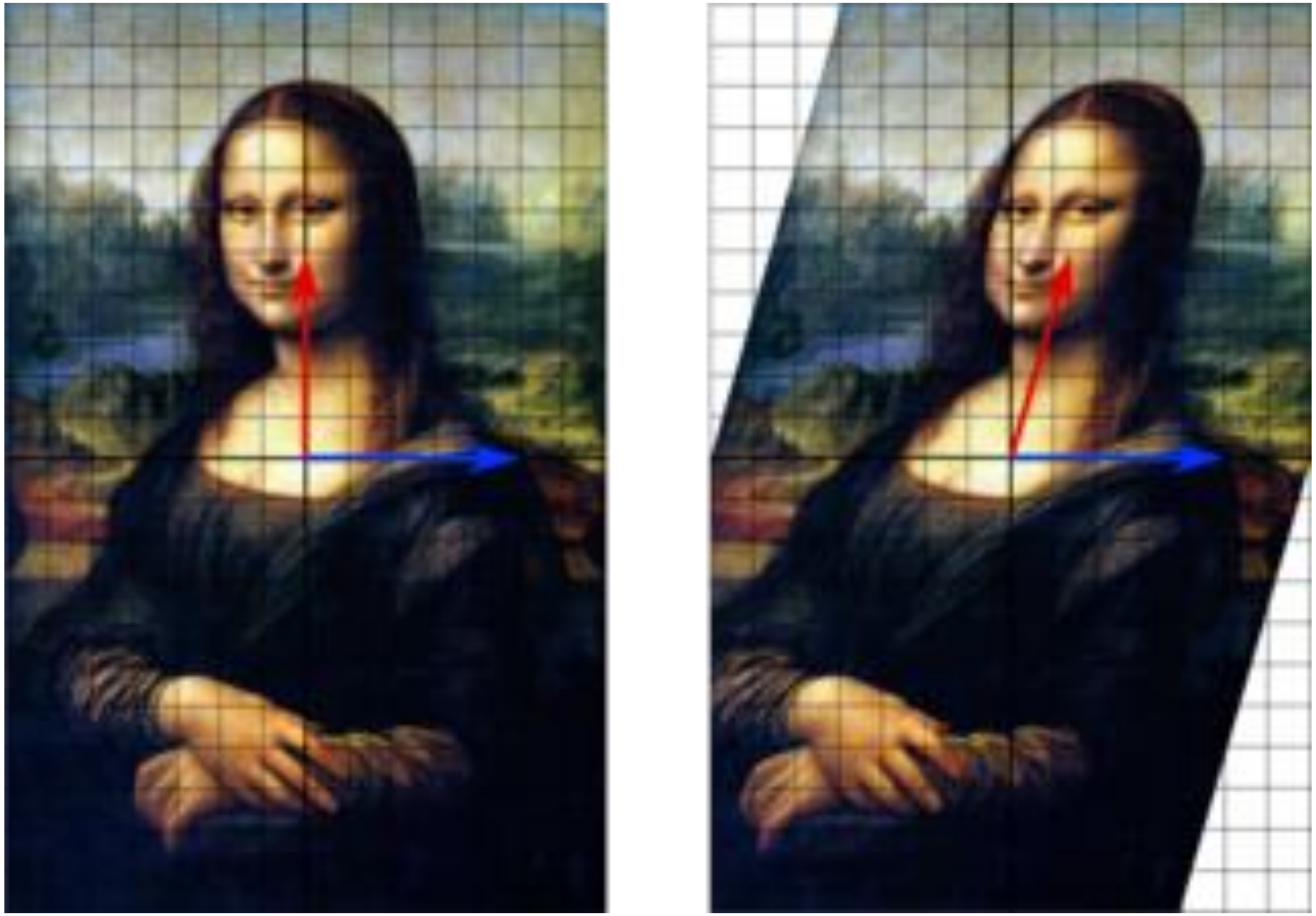
$$Ax = \lambda x, A \in \mathbb{R}^{n \times n}, x \in \mathbb{R}^n$$

- x are Eigenvectors and λ are Eigenvalues

- One of the most important concepts for machine learning, ex:
  - Principle Component Analysis (PCA)
  - Eigenvector centrality
  - PageRank
  - …

Example: Shear Mapping

- Horizontal axis is the Eigenvector



Power Iteration Method for Computing Eigenvector

1. Start with random vector v
2. Calculate iteratively: $v^{(k+1)} = A^k v$
3. After v^k converges, $v^{(k+1)} \cong v^k$
4. v^k will be the Eigenvector with largest Eigenvalue

NumPy for Linear Algebra

- NumPy is the fundamental package for scientific computing with Python. It contains among other things:
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities

Python & NumPy tutorial

- http://cs231n.github.io/python-numpy-tutorial/
- Stanford CS231n: Convolutional Neural Networks for Visual Recognition
  - http://cs231n.stanford.edu/

Table of contents:

Create Tensors

Scalars (0D tensors)

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

Vectors (1D tensors)

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

Matrices (2D tensors)

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

Create 3D Tensor

```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

Attributes of a Tensor

- Number of axes (dimensions)
  - x.ndim
- Shape
  - This is a tuple of integers showing how many data the tensor has along each axis
- Data type
  - uint8, float32 or float64

Manipulating Tensors in Numpy

```
>>> my_slice = train_images[10:100]
>>> print(my_slice.shape)
(90, 28, 28)
```

```
>>> my_slice = train_images[10:100, :, :]
>>> my_slice.shape
(90, 28, 28)
>>> my_slice = train_images[10:100, 0:28, 0:28]
>>> my_slice.shape
(90, 28, 28)
```

Equivalent to the previous example

Also equivalent to the previous example

```
my_slice = train_images[:, 7:-7, 7:-7]
```

Displaying the Fourth Digit

```
digit = train_images[4]

import matplotlib.pyplot as plt
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```

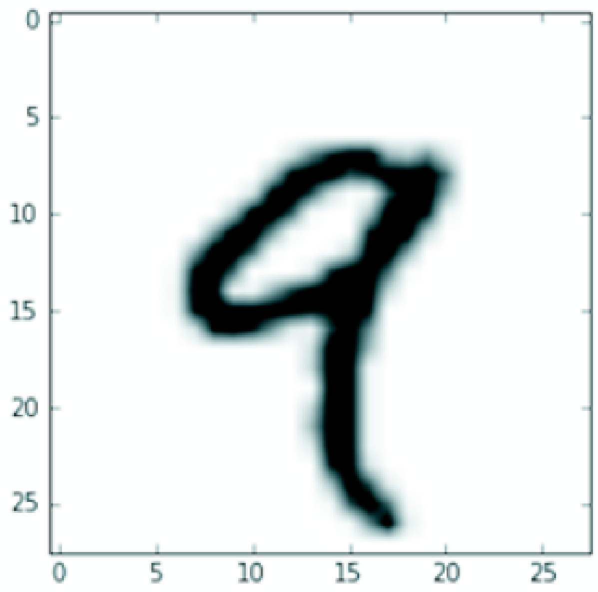


Figure 2.2 The fourth sample in our dataset

Numpy Multiplication

In [ ]:
```python
import numpy as np
```

In [4]:
```python
x = np.array([[1, 2, 3], [4, 5, 6]])
x
```

Out[4]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [5]:
```python
y = np.array([[7, 8], [9, 10], [11, 12]])
y
```

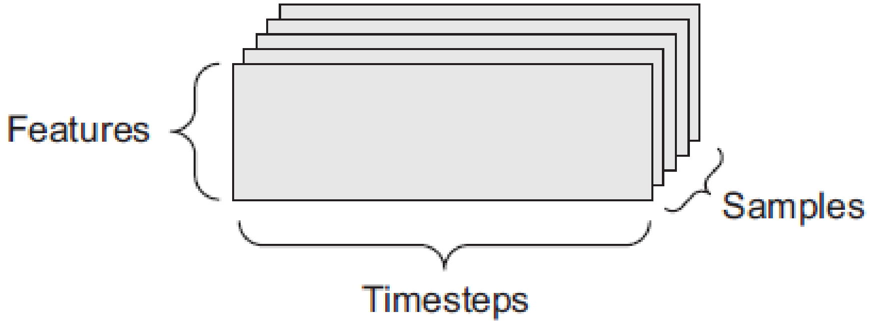
Out[5]:
```
array([[ 7,  8],
       [ 9, 10],
       [11, 12]])
```

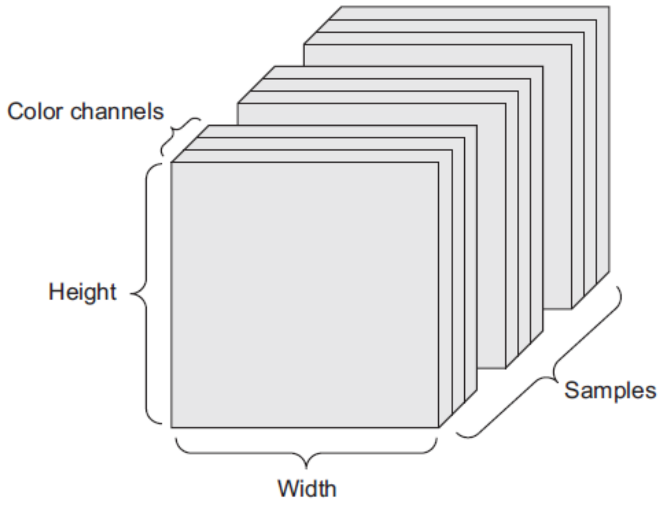
In [10]:
```python
np.matmul(x, y)
```

Out[10]:
```
array([[ 58,  64],
       [139, 154]])
```

Real-world examples of Data Tensors

- Vector data – 2D (samples, features)
- Timeseries Data – 3D (samples, timesteps, features)
- Images – 4D (samples, height, width, channels)
- Video – 5D (samples, frames, height, width, channels)





Batch size & Epochs

- A sample
  - A sample is a single row of data
- Batch size
  - Number of samples used for one iteration of gradient descent
  - Batch size = 1: stochastic gradient descent
  - 1 < Batch size < all: mini-batch gradient descent
  - Batch size = all: batch gradient descent
- Epoch
  - Number of times that the learning algorithm work through all training samples

Element-wise Operations for Matrix

- Operate on each element

```
def naive_add(x, y):
    assert len(x.shape) == 2
    assert x.shape == y.shape

    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[i, j]
    return x
```

x and y are 2D Numpy tensors.

Avoid overwriting the input tensor.

NumPy Operation for Matrix

- Leverage the Basic Linear Algebra subprograms (BLAS)
- BLAS is optimized using C or Fortran

```
import numpy as np

z = x + y                ← Element-wise addition

z = np.maximum(z, 0.)    ← Element-wise relu
```

Broadcasting

- Apply smaller tensor repeated to the extra axes of the larger tensor

```
def naive_add_matrix_and_vector(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 1
    assert x.shape[1] == y.shape[0]

    x = x.copy()
    for i in range(x.shape[0]):
        for j in range(x.shape[1]):
            x[i, j] += y[j]
    return x
```

x is a 2D Numpy tensor.

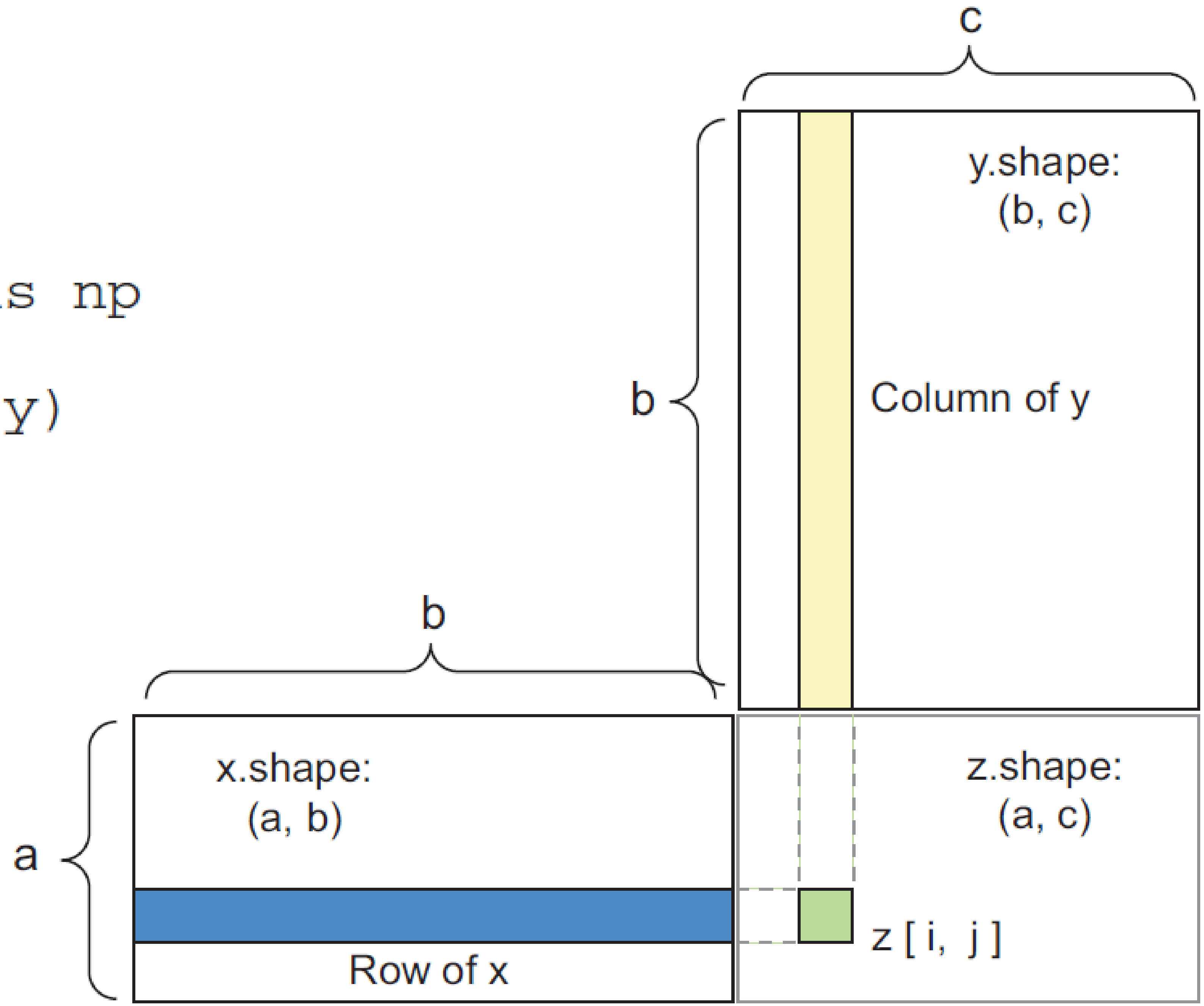
y is a Numpy vector.

Avoid overwriting the input tensor.

Tensor Dot

```
import numpy as np

z = np.dot(x, y)
```



Implementation of Dot Product

```
def naive_matrix_dot(x, y):
    assert len(x.shape) == 2
    assert len(y.shape) == 2
    assert x.shape[1] == y.shape[0]

    z = np.zeros((x.shape[0], y.shape[1]))
    for i in range(x.shape[0]):
        for j in range(y.shape[1]):
            row_x = x[i, :]
            column_y = y[:, j]
            z[i, j] = naive_vector_dot(row_x, column_y)
    return z
```

x and y are Numpy matrices.

The first dimension of x must be the same as the 0th dimension of y!

This operation returns a matrix of 0s with a specific shape.

Iterates over the rows of x …

… and over the columns of y.

Tensor Reshaping

- Rearrange a tensor’s rows and columns to match a target shape

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)
```

```
>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.],
       [ 5.]])
```

```
>>> x = x.reshape((2, 3))
>>> x
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.]])
```

Matrix Transposition

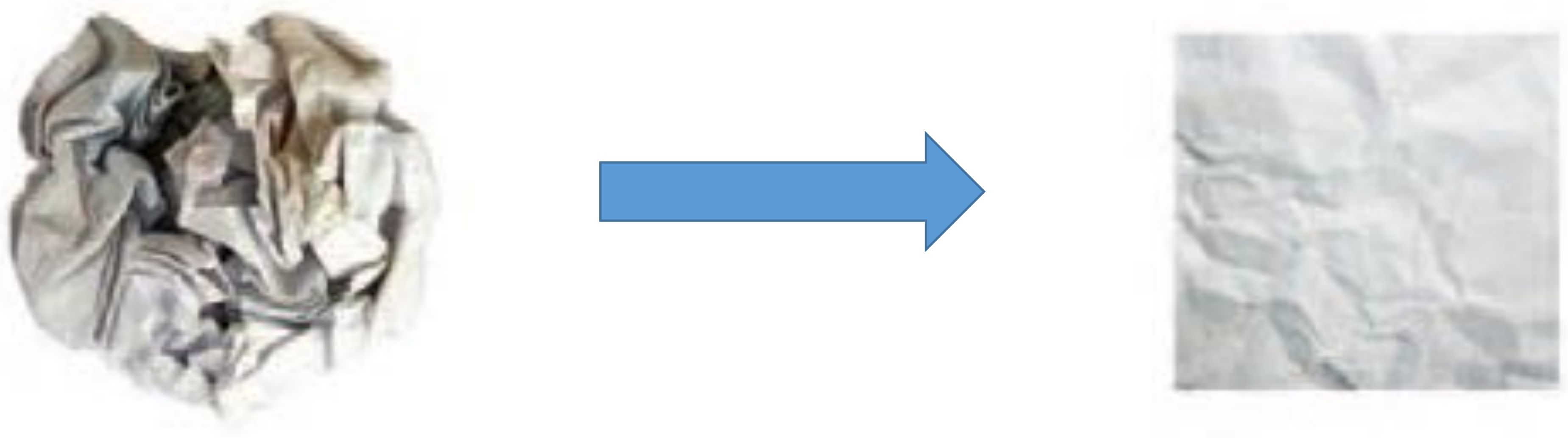
- Transposing a matrix means exchanging its rows and its columns

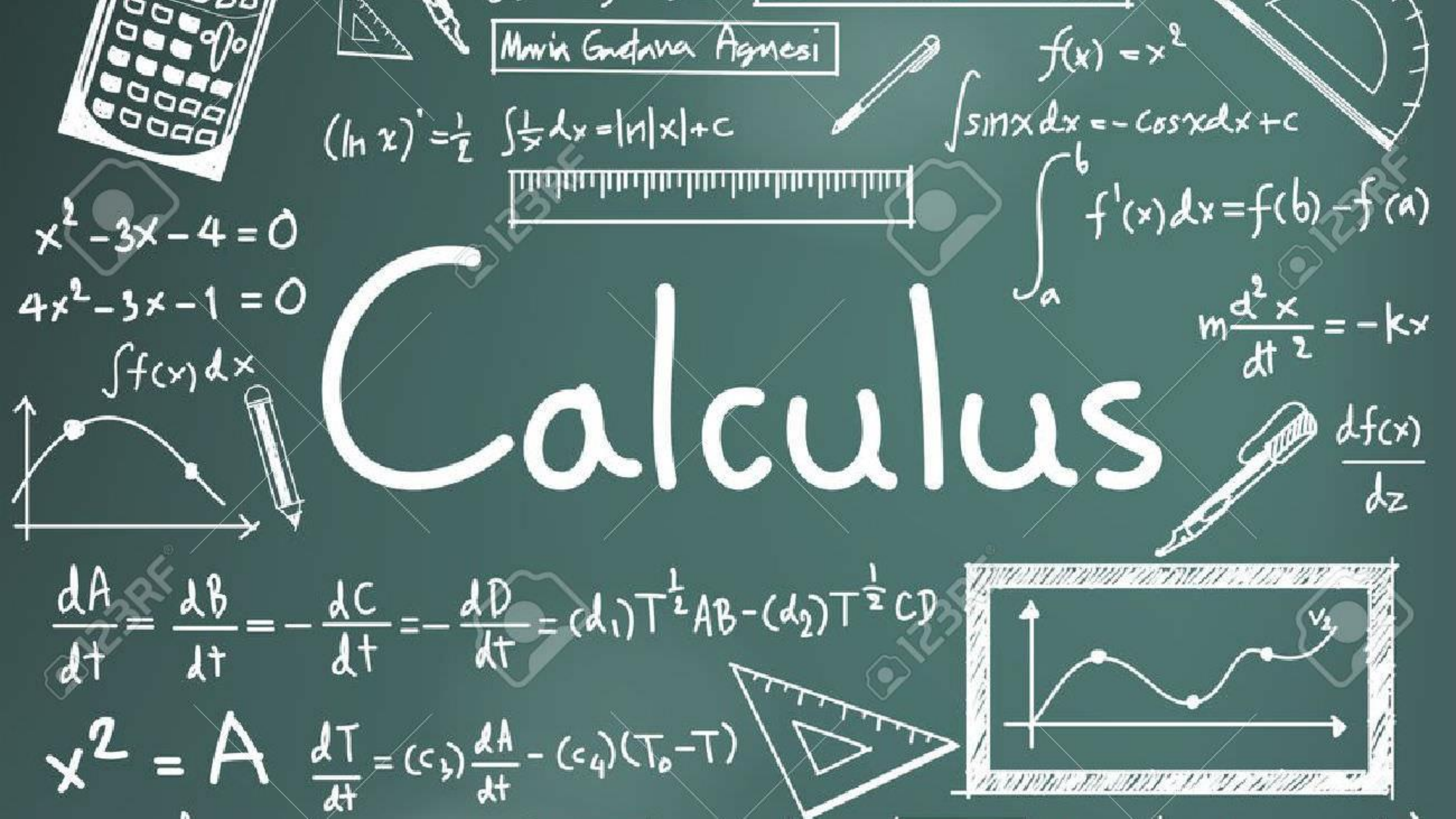
```
>>> x = np.zeros((300, 20))
>>> x = np.transpose(x)
>>> print(x.shape)
(20, 300)
```

Creates an all-zeros matrix of shape (300, 20)

Unfolding the Manifold

- Tensor operations are complex geometric transformation in high-dimensional space
  - Dimension reduction



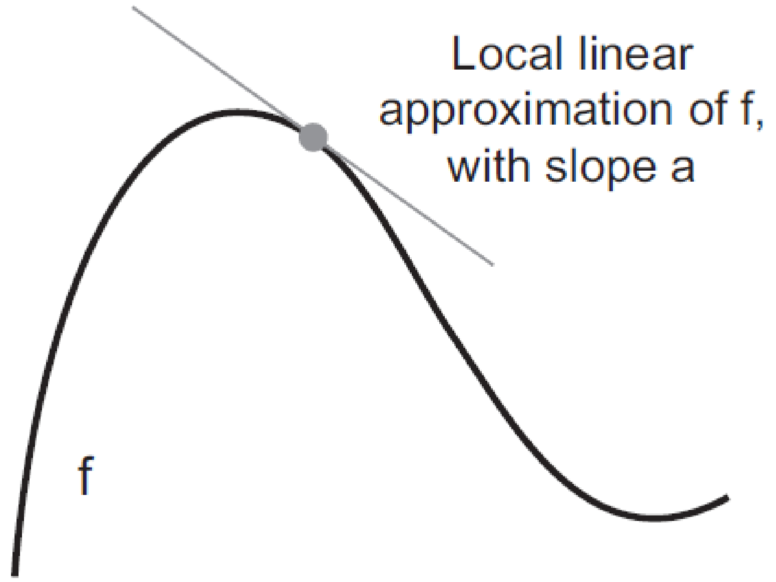

Maria Gaetana Agnesi
f(x) = x^2
(ln x)' = 1/x
∫ 1/x dx = ln|x| + c
∫ sinx dx = -cosx dx + c
∫_a^b f'(x) dx = f(b) - f(a)
x^2 - 3x - 4 = 0
4x^2 - 3x - 1 = 0
∫ f(x) dx
Calculus
m d^2x/dt^2 = -kx
df(x)/dz
dA/dt = dB/dt = -dC/dt = -dD/dt = (d1)T^1/2 AB - (d2)T^1/2 CD
v2
x^2 = A
dT/dt = (c3) dA/dt - (c4)(T0 - T)

Differentiation

$$\frac{df}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h}$$

OR

$$\frac{df}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t-h)}{2h}$$



Derivatives of Basic Function $\frac{dy}{dx}$

$y = x^n \rightarrow \frac{dy}{dx} = n x^{n-1}$ $\quad \frac{d}{dx}\left(\frac{1}{x}\right) \Rightarrow \frac{-1}{x^2}$

$y = e^x \rightarrow \frac{dy}{dx} = e^x$

$y = \ln x \rightarrow y' = \frac{1}{x}$

Gradient of a Function

- Gradient is a multi-variable generalization of the derivative
- Apply partial derivatives

$$\nabla f = \left[\frac{\partial f}{\partial x_1} \; \frac{\partial f}{\partial x_2} \cdots\cdots \frac{\partial f}{\partial x_n} \right]^T$$

- Example

$$f(x, y, z) = x + y^2 + z^3$$

$$\nabla f = 1 \times 2y \times 3z^2$$

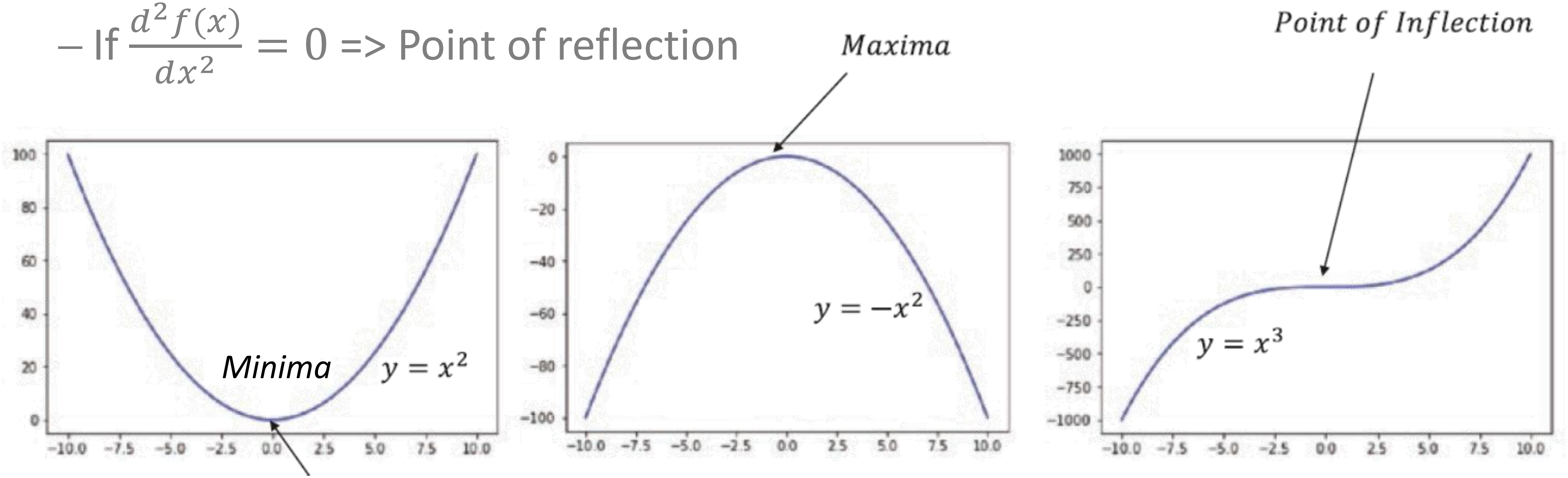
Hessian Matrix

- Second-order partial derivatives

$$Hf = \begin{bmatrix} \frac{\delta^2 f}{\delta x^2} & \frac{\delta^2 f}{\delta x \delta y} & \frac{\delta^2 f}{\delta x \delta z} \\ \frac{\delta^2 f}{\delta y \delta x} & \frac{\delta^2 f}{\delta y^2} & \frac{\delta^2 f}{\delta y \delta z} \\ \frac{\delta^2 f}{\delta z \delta x} & \frac{\delta^2 f}{\delta z \delta y} & \frac{\delta^2 f}{\delta z^2} \end{bmatrix}$$

Maxima and Minima for Univariate Function

- If $\frac{df(x)}{dx} = 0$, it's a minima or a maxima point, then we study the second derivative:
  - If $\frac{d^2f(x)}{dx^2} < 0$ => Maxima
  - If $\frac{d^2f(x)}{dx^2} > 0$ => Minima
  - If $\frac{d^2f(x)}{dx^2} = 0$ => Point of reflection



Maxima and Minima for Multivariate Function

- Computing the gradient and setting it to zero vector would give us the list of stationary points.
- For a stationary point $x_0 \in \mathbb{R}^n$
  - If the Hessian matrix of the function at x_0 has both positive and negative eigen values, then x_0 is a saddle point
  - If the eigen values of the Hessian matrix are all positive then the stationary point is a local minima
  - If the eigen values are all negative then the stationary point is a local maxima

Chain Rule

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

$$\frac{d^2y}{dx^2} = \frac{d^2y}{du^2}\left(\frac{du}{dx}\right)^2 + \frac{dy}{du}\frac{d^2u}{dx^2}$$

$$\frac{d^3y}{dx^3} = \frac{d^3y}{du^3}\left(\frac{du}{dx}\right)^3 + 3\,\frac{d^2y}{du^2}\frac{du}{dx}\frac{d^2u}{dx^2} + \frac{dy}{du}\frac{d^3u}{dx^3}$$

$$\frac{d^4y}{dx^4} = \frac{d^4y}{du^4}\left(\frac{du}{dx}\right)^4 + 6\,\frac{d^3y}{du^3}\left(\frac{du}{dx}\right)^2\frac{d^2u}{dx^2} + \frac{d^2y}{du^2}\left(4\,\frac{du}{dx}\frac{d^3u}{dx^3} + 3\left(\frac{d^2u}{dx^2}\right)^2\right) + \frac{dy}{du}\frac{d^4u}{dx^4}.$$

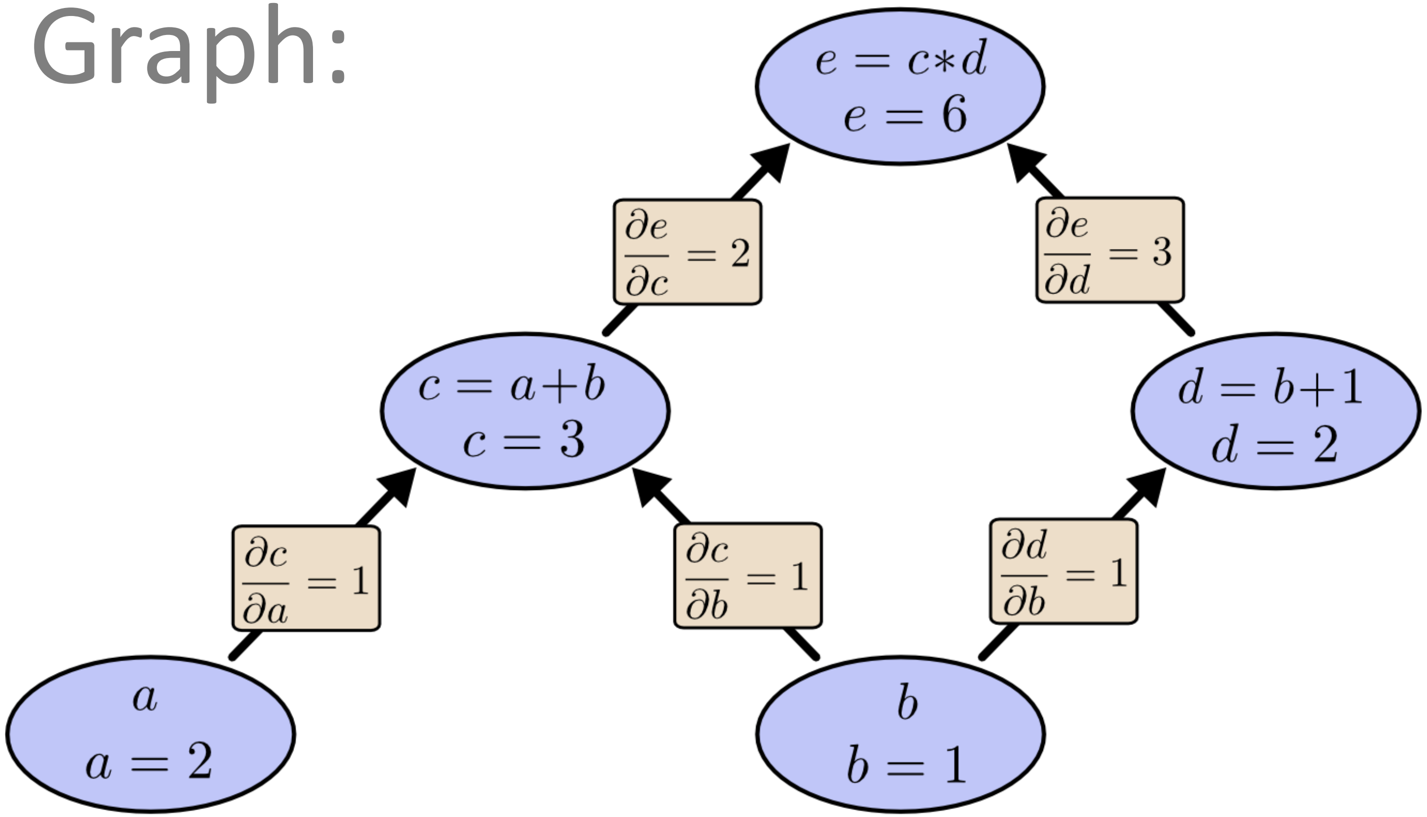
Symbolic Differentiation

Computation Graph:

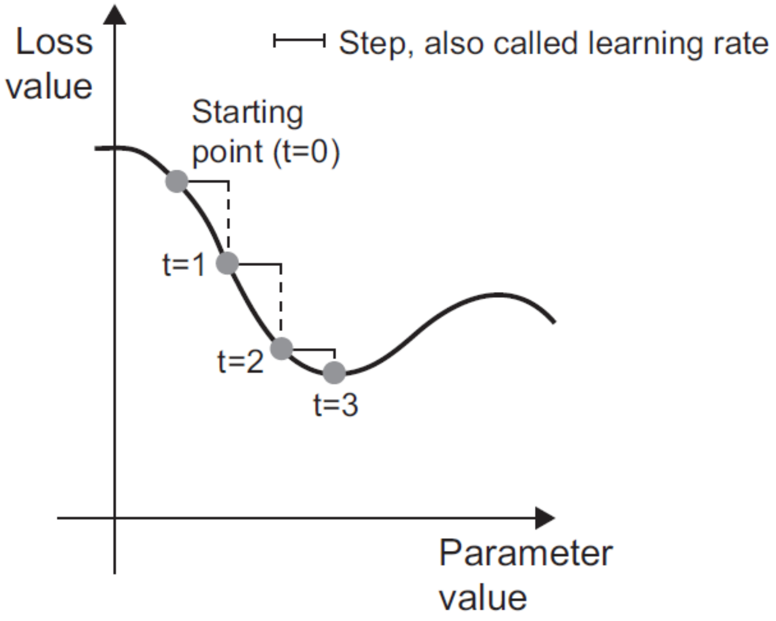
c = a + b

d = b + 1

e = c*d

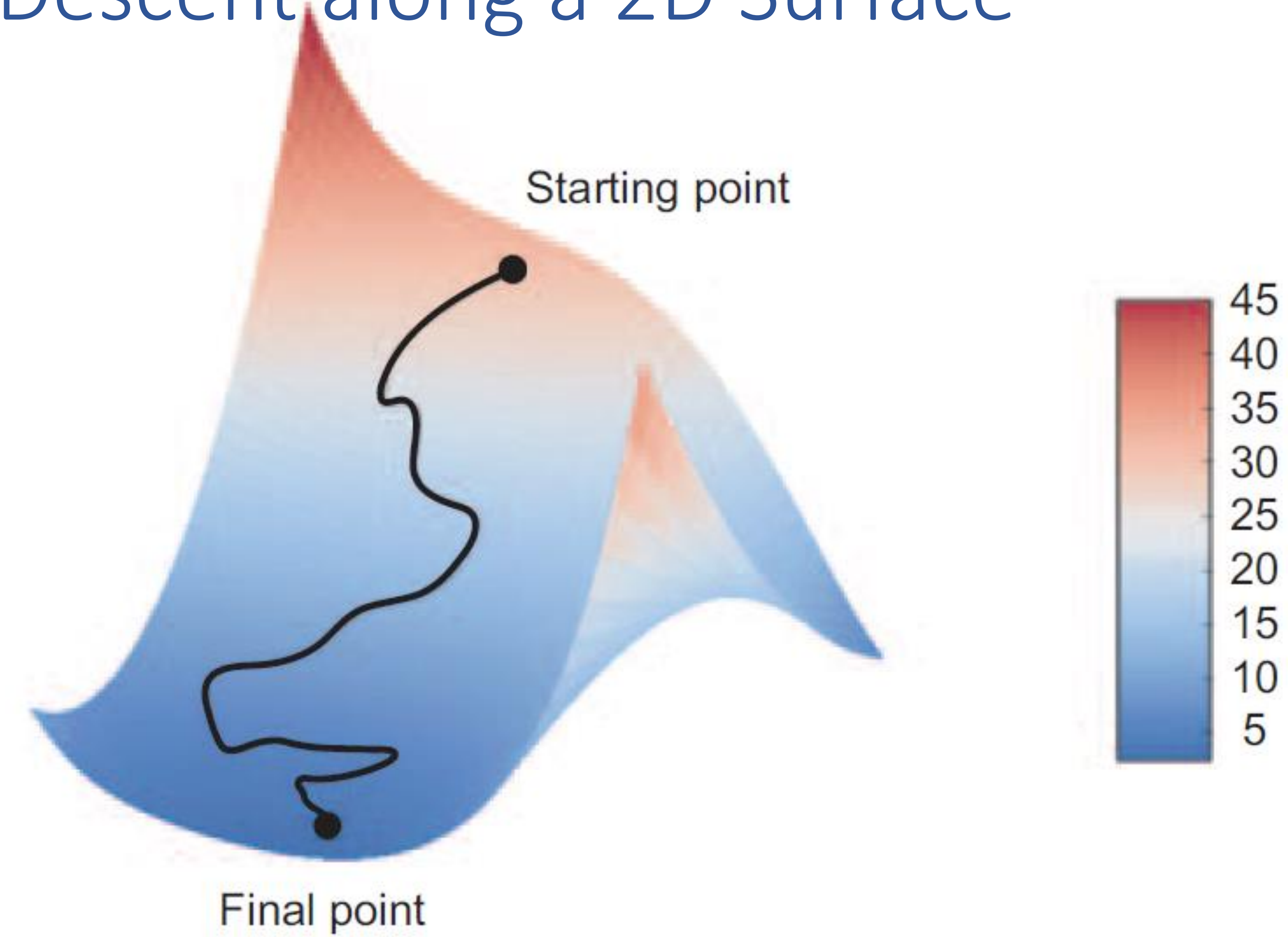


Stochastic Gradient Descent



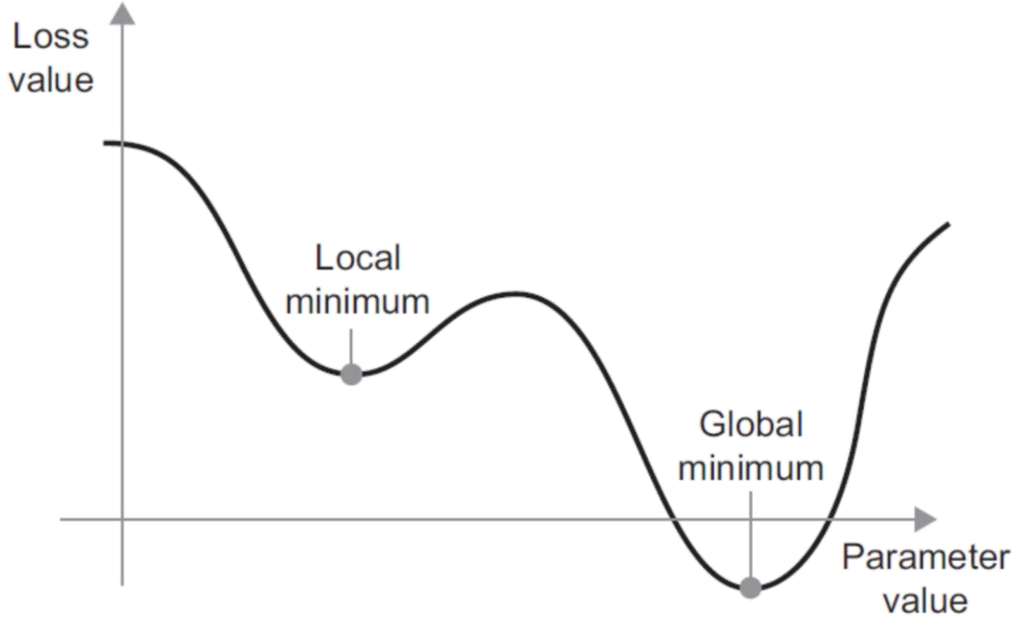
1. Draw a batch of training samples x and corresponding targets y
2. Run the network on x to obtain predictions y_pred
3. Compute the loss of the network on the batch, a measure of the mismatch between y_pred and y
4. Compute the gradient of the loss with regard to the network's parameters (a backward pass).
5. Move the parameters a little in the opposite direction from the gradient: W -= step * gradient

Gradient Descent along a 2D Surface



Avoid Local Minimum using Momentum

```
past_velocity = 0.
momentum = 0.1            <- Constant momentum factor
while loss > 0.01:        <- Optimization loop
    w, loss, gradient = get_current_parameters()
    velocity = past_velocity * momentum + learning_rate * gradient
    w = w + momentum * velocity - learning_rate * gradient
    past_velocity = velocity
    update_parameter(w)
```



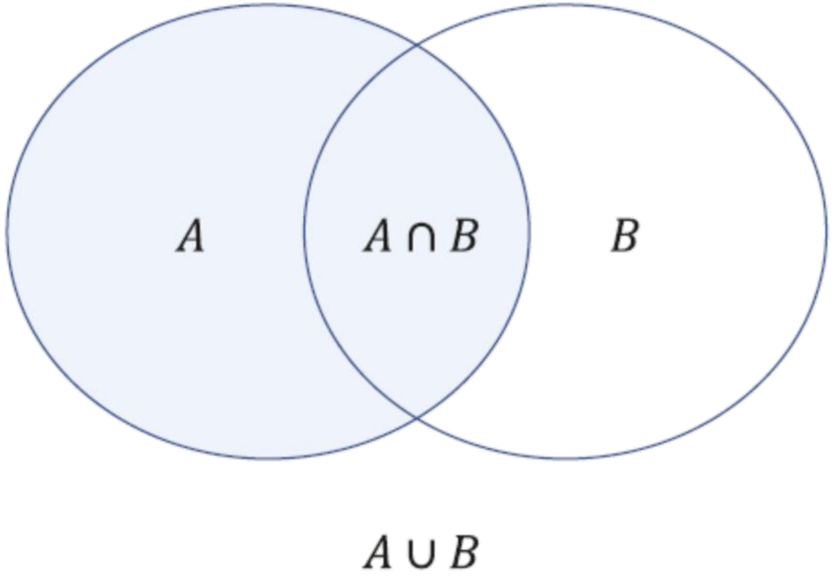
Basics of Probability

Three Axioms of Probability

- Given an Event E in a sample space S, $S = \bigcup_{i=1}^{N} E_i$
- First axiom
  - $P(E) \in \mathbb{R}, 0 \leq P(E) \leq 1$
- Second axiom
  - $P(S) = 1$
- Third axiom
  - Additivity, any countable sequence of mutually exclusive events E_i
  - $P(\bigcup_{i=1}^{n} E_i) = P(E_1) + P(E_2) + \cdots + P(E_n) = \sum_{i=1}^{n} P(E_i)$

Union, Intersection, and Conditional Probability

- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- $P(A \cap B)$ is simplified as $P(AB)$
- Conditional Probability $P(A|B)$, the probability of event A given B has occurred
  - $P(A|B) = P\left(\frac{AB}{B}\right)$
  - $P(AB) = P(A|B)P(B) = P(B|A)P(A)$



Chain Rule of Probability

- The joint probability can be expressed as chain rule

$$P(A_1A_2A_3\ldots A_n)=P(A_1)P(A_2/A_1)P(A_3/A_1A_2)\ldots\ldots P\left(A_n/A_1A_2\ldots A_{(n-1)}\right)$$

$$=P(A_1)\prod_{i=2}^{n}P\left(A_i/A_1A_2A_3\ldots A_{(n-1)}\right)$$

Mutually Exclusive

- $P(AB) = 0$
- $P(A \cup B) = P(A) + P(B)$

Independence of Events

- Two events A and B are said to be independent if the probability of their intersection is equal to the product of their individual probabilities
  - $P(AB) = P(A)P(B)$
  - $P(A|B) = P(A)$

Bayes Rule

(Training Data)

feature class(label)

likelihood

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

prior (class)

class

features (Image)

Evidence

Proof:

Remember $P(A|B) = P\left(\frac{AB}{B}\right)$

So $P(AB) = P(A|B)P(B) = P(B|A)P(A)$

Then Bayes $P(A|B) = P(B|A)P(A)/P(B)$

Naïve Bayes Classifier

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

$$p(C_k \mid x_1, \dots, x_n)$$

$$\begin{aligned}
p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2, \dots, x_n, C_k) \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k)\, p(x_3, \dots, x_n, C_k) \\
&= \cdots \\
&= p(x_1 \mid x_2, \dots, x_n, C_k)\, p(x_2 \mid x_3, \dots, x_n, C_k) \cdots p(x_{n-1} \mid x_n, C_k)\, p(x_n \mid C_k)\, p(C_k)
\end{aligned}$$

Naïve = Assume All Features Independent

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &= p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\ &= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

Probability Mass Function and Dense Function

- Probability mass function (PMF)
  - Function that gives the probability that a discrete random variable is exactly equal to some value

$$P(X = i) = \frac{1}{6}, i \in \{1,2,3,4,5,6\}$$

- Probability dense function (PDF)
  - Specify the probability of the random variable falling within a particular range of values

$$\int_D P(x)dx = 1$$

Expectation of a Random Variable

- Expectation of a discrete random variable

$$E[X] = x_1p_1 + x_2p_2 + \cdots + x_np_n = \sum_{i=1}^{n} x_ip_i$$

- Expectation of a continuous random variable

$$E[X] = \int_D xP(x)dx$$

Variance of a Random Variable

- Expectation of a discrete random variable

$$Var[X] = E[(X - \mu)^2], \text{where } \mu = E[X]$$

- Expectation of a continuous random variable

$$Var[X] = \int_D (x - \mu)^2 P(x) dx$$

- Standard deviation σ is the square root of variance

Covariance and Correlation Coefficient

- Expectation of a discrete random variable

$$Cov(X,Y) = E\left[(X-\mu_x)\left(Y-\mu_y\right)\right],$$
$$\text{where } \mu_x = E[X], \mu_y = E[Y]$$

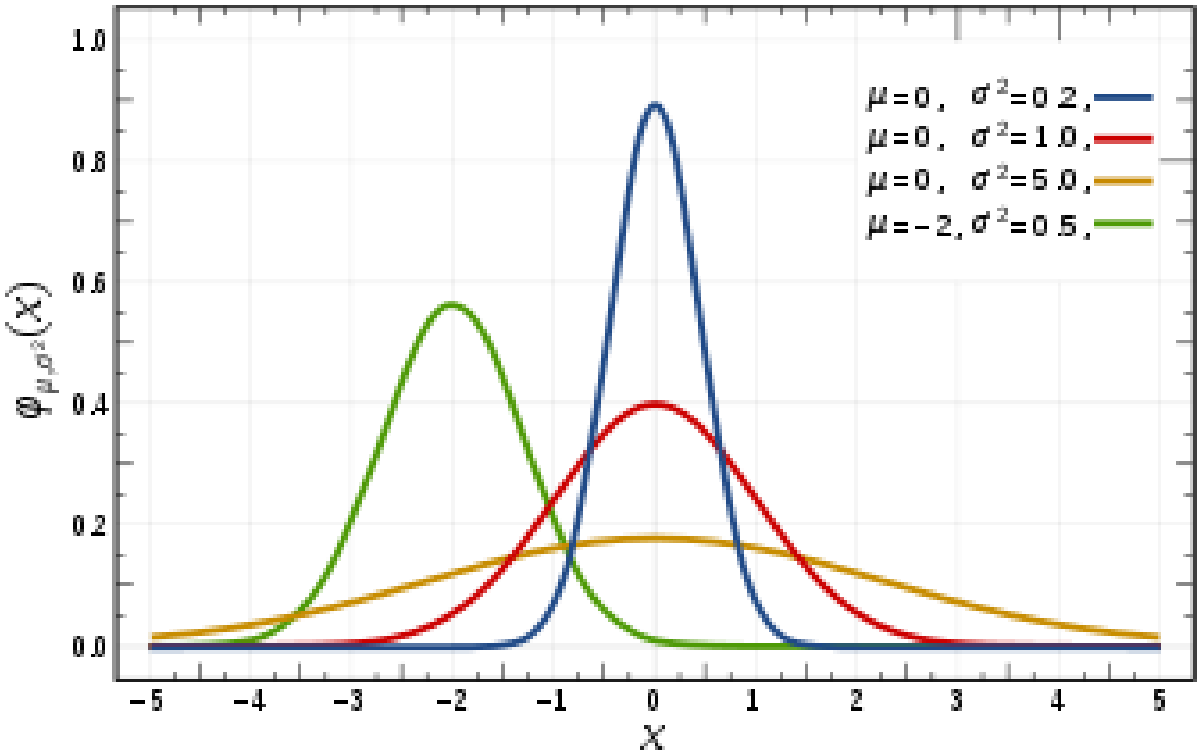
- Correlation coefficient

$$\rho = \frac{cov(X,Y)}{\sigma_x \sigma_y}$$

Normal (Gaussian) Distribution

- One of the most important distributions
- Central limit theorem
  - Averages of samples of observations of random variables independently drawn from independent distributions converge to the normal distribution

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



Optimization

The *standard form* of a continuous optimization problem is[1]

$$\begin{aligned} &\underset{x}{\operatorname{minimize}} && f(x) \\ &\operatorname{subject\ to} && g_i(x) \leq 0, \quad i = 1, \dots, m \\ &&& h_j(x) = 0, \quad j = 1, \dots, p \end{aligned}$$

where

- $f : \mathbb{R}^n \to \mathbb{R}$ is the **objective function** to be minimized over the *n*-variable vector x,
- $g_i(x) \leq 0$ are called **inequality constraints**
- $h_j(x) = 0$ are called **equality constraints**, and
- $m \geq 0 \; and \; p \geq 0.$

https://en.wikipedia.org/wiki/Optimization_problem

Formulate Your Problem

- Linear model: $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + \mathrm{b}$
- Least-squared Error: $(f(\boldsymbol{x}) - \boldsymbol{y})^2$
- Regularization: $||\boldsymbol{w}||$
- Objective function:

$$\min_{w}.\ \left(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{y}\right)^2 + \lambda||\boldsymbol{w}||$$

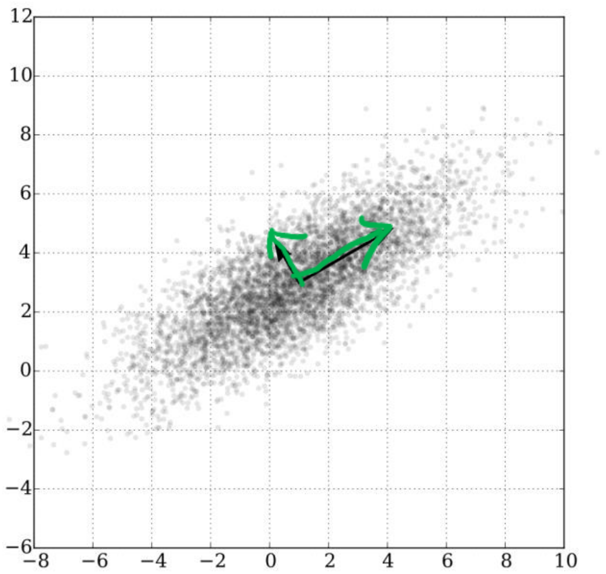
Principle Component Analysis (PCA)

$y = W^T x$

- Assumptions
  - *Linearity*
  - *Mean and Variance are sufficient statistics*
  - *The principal components are orthogonal*

$\max \; Cov(y, y)$

s.b.t. $W^T W = I$



Principle Component Analysis (PCA)

$$\max_{w}. \quad \mathrm{cov}(\mathbf{Y}, \mathbf{Y})$$
$$s.b.t \quad \mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{I}$$

$$f = \mathrm{cov}(Y, Y) + \lambda(W^TW - I)$$

$Y = W^Tx$

$$\mathrm{Cov}(Y,Y) = \frac{1}{N-1}(Y-\mu_y)^T(Y-\mu_y) = \frac{1}{N-1}(W^Tx - W^T\mu_x)^T(W^Tx - W^T\mu_x) = W\,\Sigma_x\,W^T$$

$\Sigma_x = \mathrm{Cov}(X,X)$

$$\frac{df}{dw} = 0 \Rightarrow \frac{d}{dw}\left(W\Sigma_x W^T + \lambda(W^TW - I)\right)$$

$$\Rightarrow 2\Sigma_x W + 2\lambda W = 0 \Rightarrow \Sigma_x W = \lambda W$$

References

- Francois Chollet, “Deep Learning with Python,” Chapter 2 “Mathematical Building Blocks of Neural Networks”
- Santanu Pattanayak, ”Pro Deep Learning with TensorFlow,” Apress, 2017
- Machine Learning Cheat Sheet
- https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
- https://www.quora.com/What-is-the-difference-between-L1-and-L2-regularization-How-does-it-solve-the-problem-of-overfitting-Which-regularizer-to-use-and-when
- Wikipedia