



# Artificial Intelligence in Practice: Elements and Tools

In-Ho Lee

*Korea Research Institute of Standards and Science, Daejeon 34113, Republic of Korea*

*ihlee@kriis.re.kr*

*incredible.egloos.com*

KIAS CAC, June 27 - July 1, 2022





# Goal

AI history and vision

Scikit-learn, python

Decision-tree, Random forest

Xgboost, lightGBM

Keras, tensorflow (sequential, functional)

Keras, tensorflow, pytorch (CNN)

Generative model, VAE, GAN

Genetic algorithms, Particle swarm optimization

artificial intelligence (AI)

The overall research goal of AI is to create technology that allows computers and machines to function in an intelligent manner.



# Survey, approach, and anaconda3

- Python {beginner, intermediate, expert}
- Matlab, Java, R, C++, C, FORTRAN, MPI,...
- Scikit-learn
- Keras
- Tensorflow
- Pytorch
- Git



```
    if (r > i &amp; r < n) {
        r = i + 1;
        a = a.length();
        a = a[0];
        if (a) {
            if (a) {
                for (j; a > a[j]; a)
                    if (r < a.replace(a[j], a[j], r == 1)) break;
            } else
                for (j; a > a[j]; a)
                    if (r < a.replace(a[j], a[j], r == 1)) break;
        } else if (a) {
            for (j; a > a[j]; a)
                if (r < a.replace(a[j], a[j], r == 1)) break;
        } else
            for (j; a > a[j]; a)
                if (r < a.replace(a[j], a[j], r == 1)) break;
        return a;
    }
    trim: b &amp; b.call("\ufe0f\ufe0f") > function(e) {
        return null == e ? "" : b.call(e);
    } : function(e) {
        return null == e ? "" : (e + "").replace(e, "");
    };
    unescape: function(e, t) {
        var n = t || [];
        return null != e &amp; (n.push(e)) &amp; n.escape("string" == typeof e ? [e] : e) : b.call(n, e);
    };
    integer: function(e, t, n) {
        var r = t || 10;
        if (e) {
            if (n) return b.call(t, e, n);
            for (r = t.length, n = e ? 0 : 0; r > n; r += 1, e += 1, n += 1)
                if (e > n &amp; t[r] == e) return e;
        }
    };
}
```

---

a positivist approach: Python + scikit-learn + keras + tensorflow

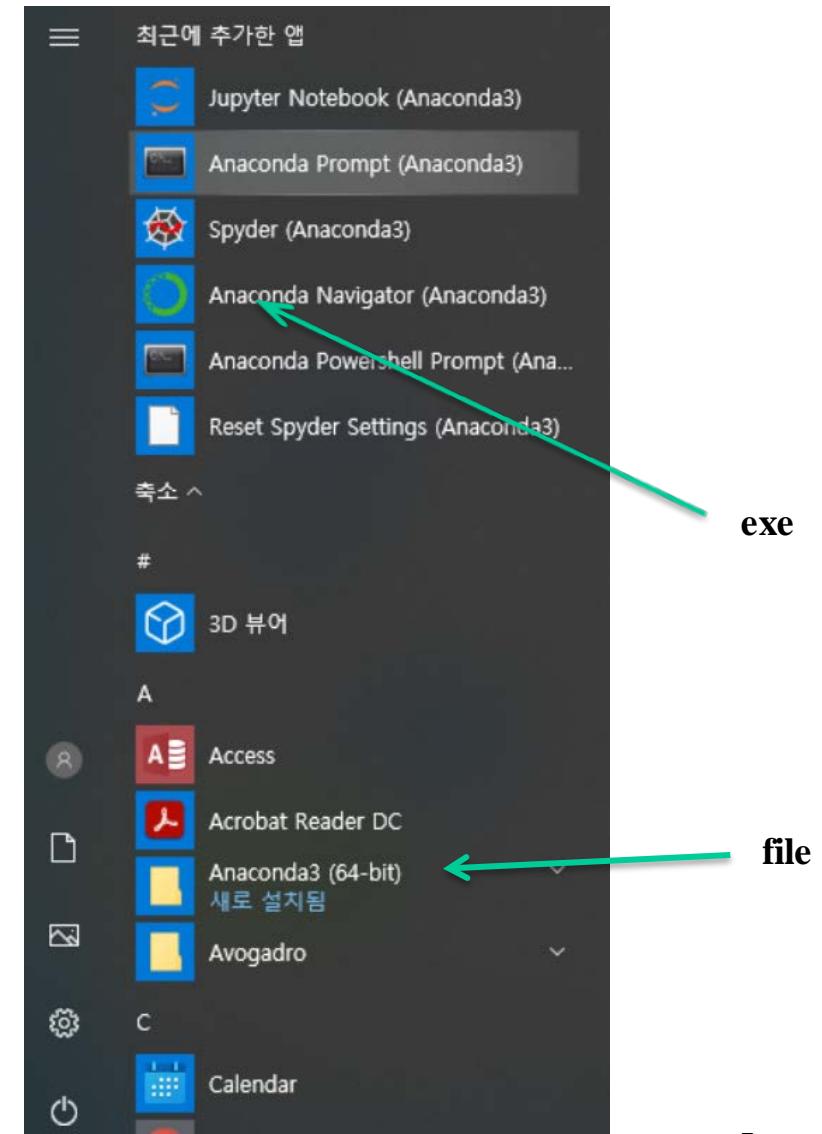
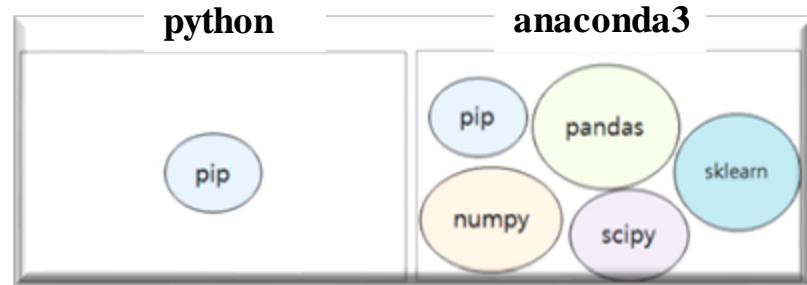
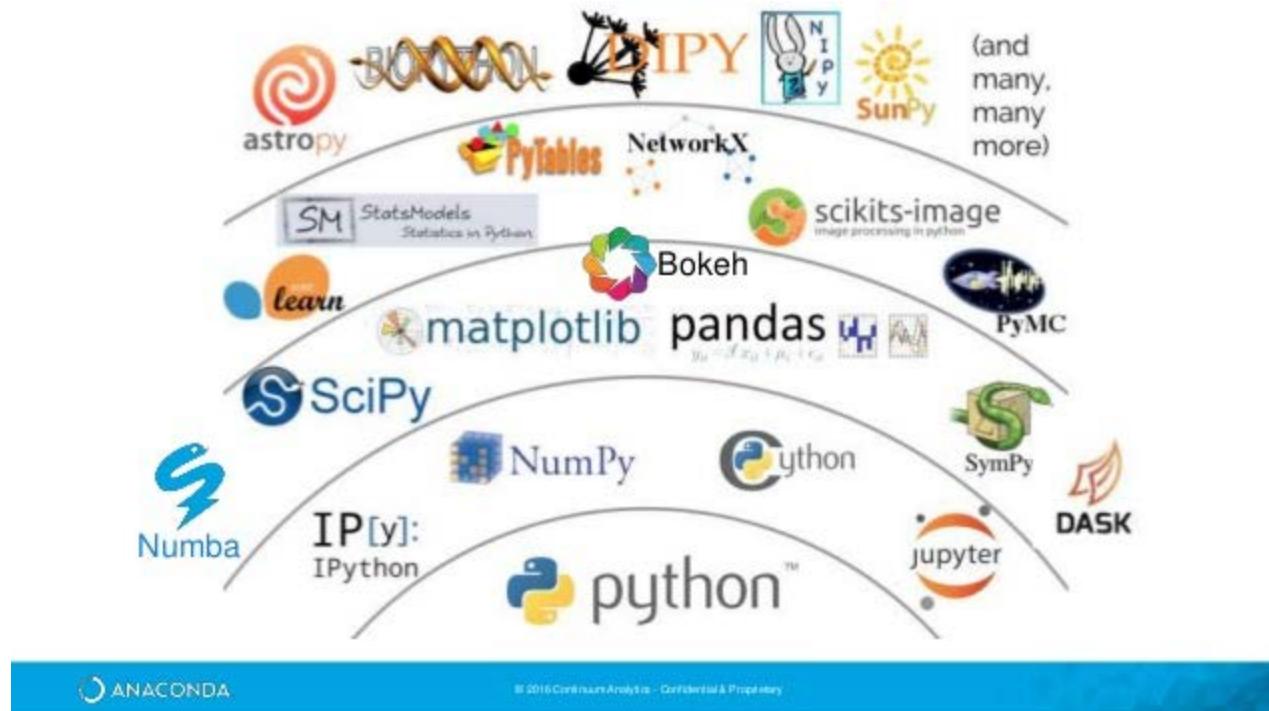
---

# Language is the house of the truth of Being.

-Martin Heidegger



# Anaconda3



# Contents

- \*Modern introduction (3+3)
- \*Scikit-learn and Keras tutorial (3+3+3+3+3)
- \*Projects/Applications (3)





# Frontiers in computation



# Market size?

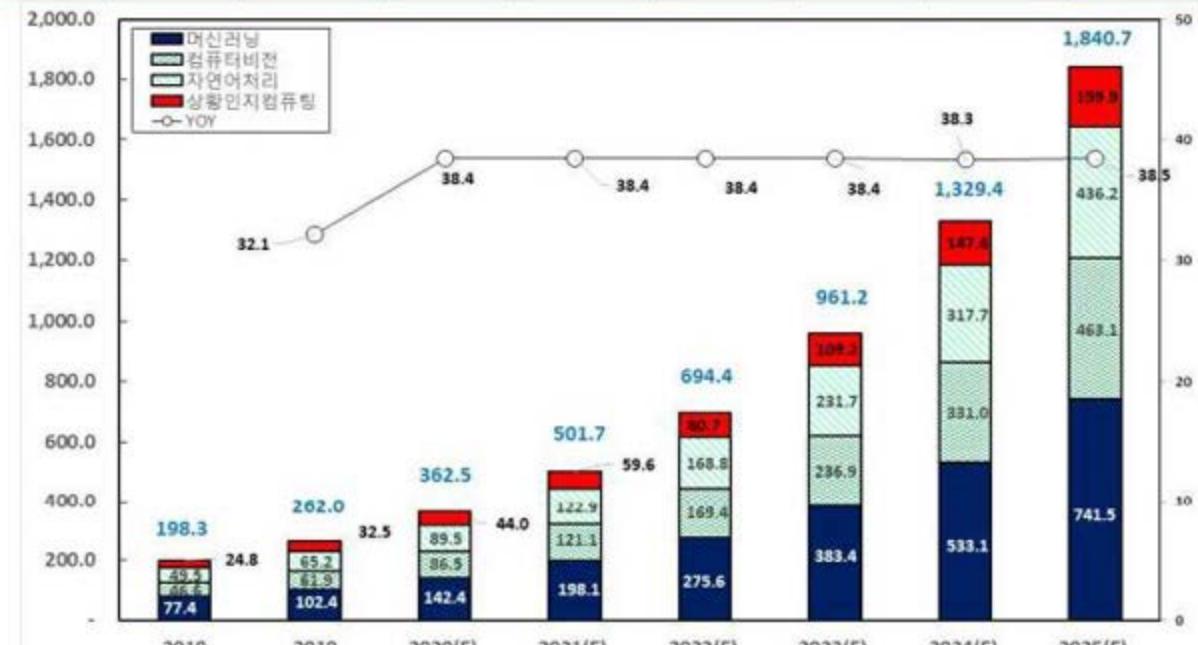


# Market size?

[세계 인공지능 시장규모]

(단위: 억 달러)  
CAGR(%)  
('19~'25)

구분	2018년	2019년	2020년(E)	2021년(E)	2022년(E)	2023년(E)	2024년(E)	2025년(E)	
시장규모	198.3	262.0	362.5	501.7	694.4	961.2	1329.4	1840.7	38.4%
머신러닝	77.4	102.4	142.4	198.1	275.6	383.4	533.1	741.5	39.1%
컴퓨터비전	46.6	61.9	86.5	121.1	169.4	236.9	331.0	463.1	39.9%
자연어처리	49.5	65.2	89.5	122.9	168.8	231.7	317.7	436.2	37.3%
상황인지컴퓨팅	24.8	32.5	44.1	59.6	80.6	109.2	147.6	199.9	35.4%
성장률(YoY)	-	32.1	38.4	38.4	38.4	38.4	38.3	38.5	-



주: 세계 인공지능 시장 규모에는 인공지능의 구현에 필요한 하드웨어, 소프트웨어, 인공지능을 이용한 서비스 시장을 모두 포함한 것으로, 2019년 기준 하드웨어 시장 43.8%, 소프트웨어 시장 27.9%, 서비스 시장 28.3%임.

자료: Global Artificial Intelligence(AI) Market, BCC(2020)

# AI 수준?

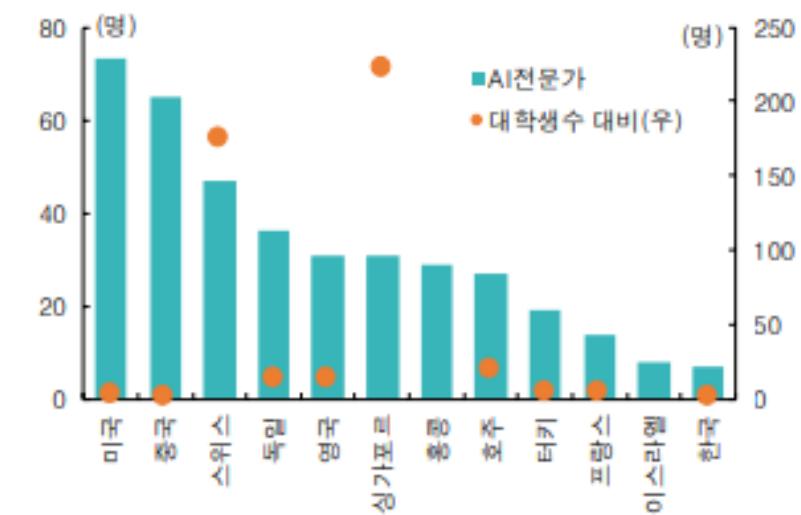
## 우리나라 인공지능 수준 비교 - 글로벌 지표 결과

출처: 한국정보화진흥원 '2019 NIA AI Index'

지표명	1위 국가/데이터 값	한국 데이터값	한국 순위	1위 국가 대비수준(%)
특허 등록(합계)	중국 / 1351건	497건	3위/7개국	36.8%
논문 등록 합계	중국 / 440건	37건	6위/7개국	8.4%
대학교·대학원 수	영국 / 55개	0개	5위/8개국	0.0%
Kaggle 상위 랭커	미국 / 27명	1명	8위/8개국	3.7%
시장규모	미국 / 7억6650만 달러	4760만 달러	5위/7개국	6.2%
인공지능 기업수	미국 / 2028개	26개	8위/8개국	1.3%
인공지능 스타트업 수	미국 / 1393개	465개	2위/8개국	33.4%
규제 샌드박스	영국 / 29건	0건	3위/4개국	0.0%

※ 데이터 값은 2018년을 기준으로 국가별 경제규모를 고려하지 않은 절대 수치

## ■ 국가별 AI 인력 현황



주1 : 2009~2018년 AI 논문 상위연구자 500명 기준

주2 : 고등교육(tertiary education) 1만명 기준 시 상대 규모

자료 : 소프트웨어정책연구소, OECD, 하나금융경영연구소



# 斯文亂賊(사문난적)

- 귀납 추론(歸納推論, induction)의 최고봉
- 연역적 추론(演繹的推論, deductive reasoning)

「유교(儒教)를 어지럽히는 도적(盜賊)」 이라는 뜻으로, 교리(教理)에 어긋나는 언동(言動)으로 유교(儒教)를 어지럽히는 사람을 이르는 말.

독서백편의자현(讀書百遍義自見) 또는 위편삼절(韋編三絕)

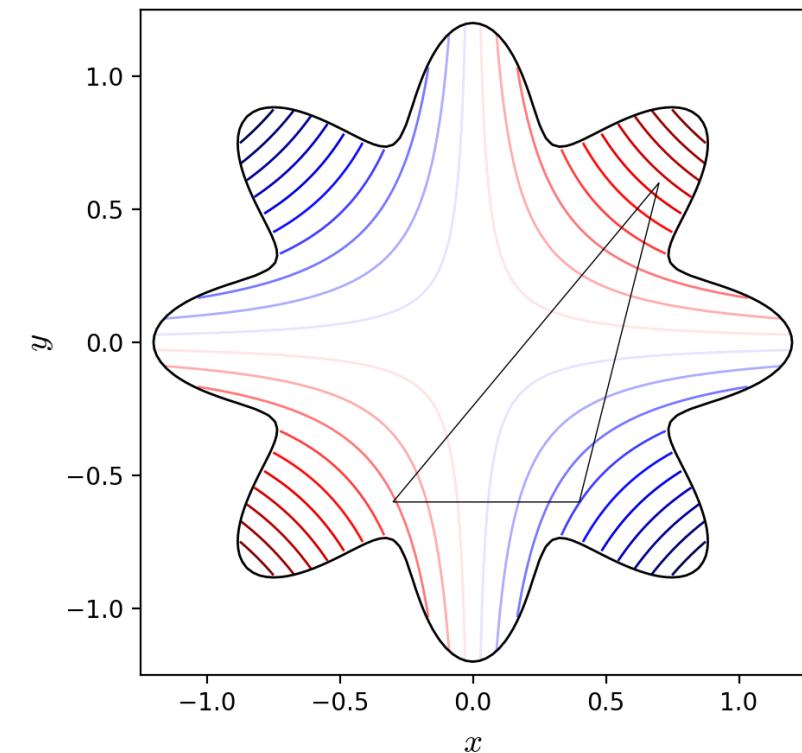
# AI

- 발견을 자동화
- 기존 정보에 지능을 추가, 축차합리성
- 점진적 학습, 데이터로부터 프로그램이 만들어질 수 있게 함
- 데이터를 보다 깊게 분석
- 이전에 불가능했던 작업을 새로운 계산 방식으로 수행
- 데이터의 활용도를 극대화 (비전, 이해력, 기억력 향상)



# 최적화를 통해서 다양한 목적을 달성한다.

- 목적함수 (손실함수) : 원하는 목적을 표현하는 함수
- 목적을 이루는 방법: 최적화 (슈퍼컴퓨팅, GPU)
- ‘문제 해결형’ 컴퓨터 프로그램 능력
- 상당한 컴퓨터 지식과 경험이 필요함





# References

**Machine Learning Mastery with Python, 2016**

**Machine Learning with Python/Scikit-Learn, 2015**

**Scikit-learn user guide, 2019**

**Learn Keras for Deep Neural Networks, 2019**

**Deep Learning with Keras, 2017**

**Deep Learning with Python, 2018**

**Hands-On Machine Learning with Scikit-Learn and TensorFlow, 2017**

**Introduction to Machine Learning with Python, 2017**

**Python data science handbook, 2017**

<https://github.com/jakevdp/PythonDataScienceHandbook/tree/master/notebooks>

<https://github.com/wesm/pydata-book/tree/3rd-edition>

[google.com](http://google.com)

[stackoverflow.com](http://stackoverflow.com)

[keras.io](http://keras.io)

[machinelearningmastery.com/start-here](http://machinelearningmastery.com/start-here)

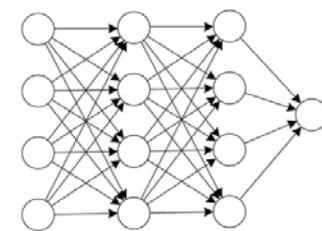
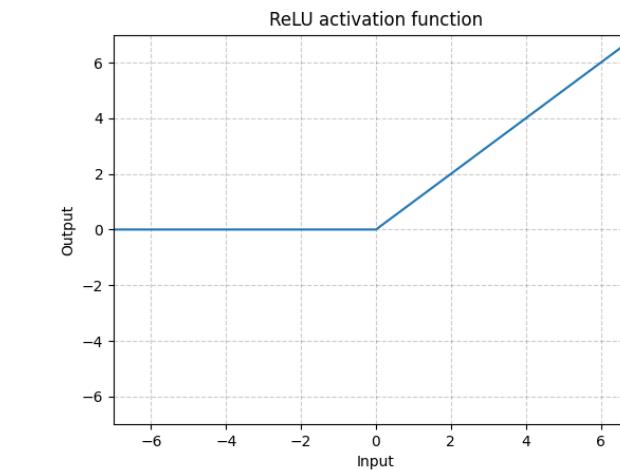
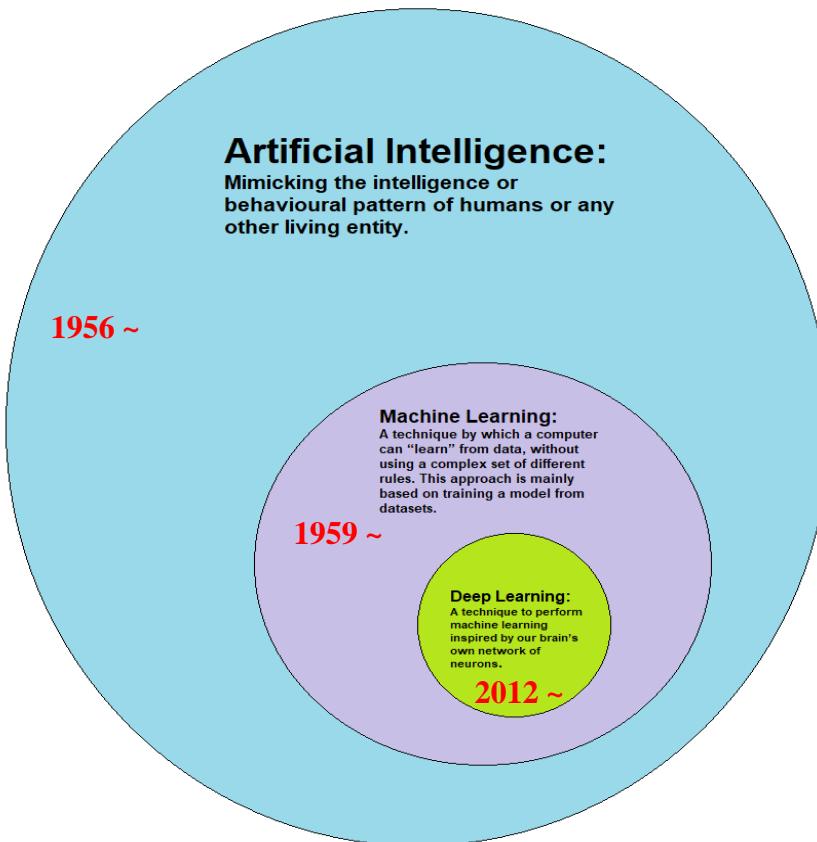
<https://machine-learning-for-physicists.org/>



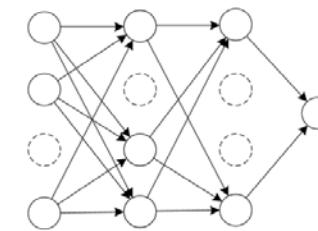


# A modern introduction

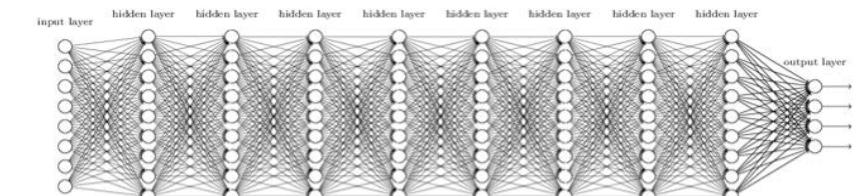
Deep learning is a class of **machine learning algorithms** that use multiple layers to progressively extract higher level features from raw input.



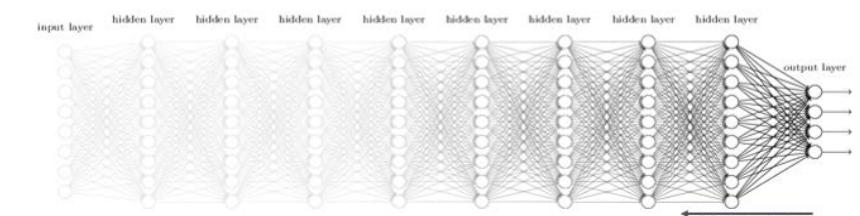
(a) Standard Neural Network



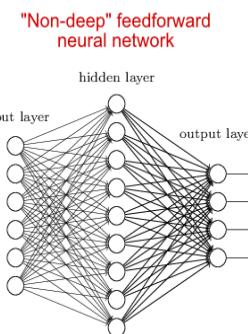
(b) Network after Dropout



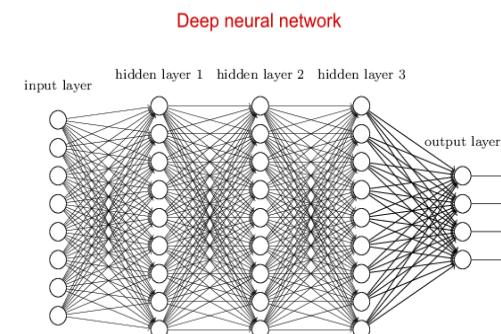
Deep Neural Network



Vanishing Gradient



"Non-deep" feedforward neural network



Deep neural network



# Machine-learning(ML)?

Computer science that gives computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.

- Bohr vs Schrödinger
- Bioinformatics
- Materials informatics?
- Metaheuristic
- Attribute oriented induction



# 얼마나 창의적인가? 새로운 수학공식들

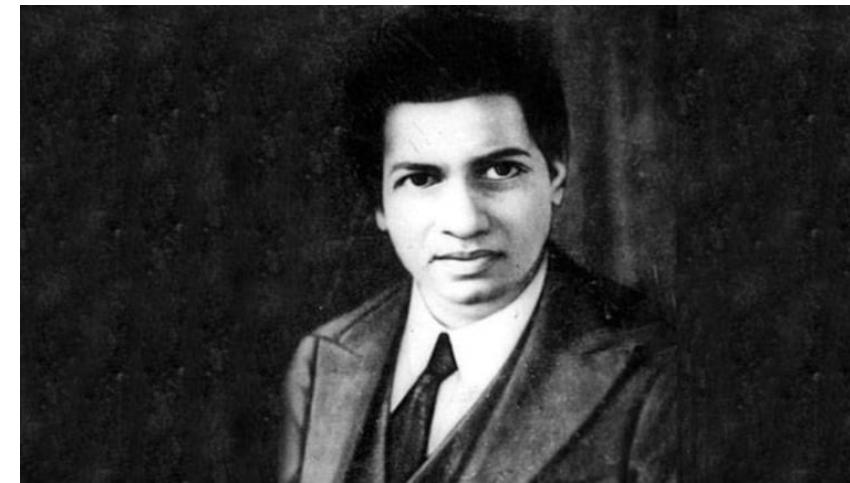
$$\pi = \sum_{n=0}^{\infty} \frac{1}{16^n} \left( \frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right).$$

$$\frac{8}{\pi^2} = 1 - \frac{2 \times 1^4 - 1^3}{7 - \frac{2 \times 2^4 - 2^3}{19 - \frac{2 \times 3^4 - 3^3}{37 - \frac{2 \times 4^4 - 4^3}{\dots}}}}$$

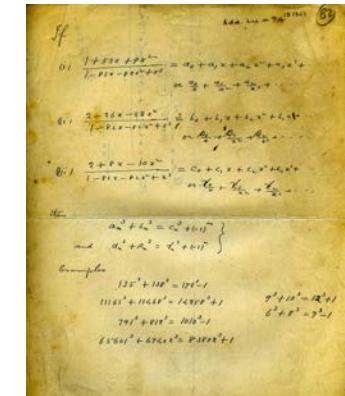
$$\frac{12}{7\zeta(3)} = 1 \times 2 - \frac{16 \times 1^6}{3 \times 12 - \frac{16 \times 2^6}{5 \times 32 - \frac{16 \times 3^6}{7 \times 62 - \frac{16 \times 4^6}{\dots}}}}$$

$$\frac{8}{7\zeta(3)} = 1 \times 1 - \frac{1^6}{3 \times 7 - \frac{2^6}{5 \times 19 - \frac{3^6}{7 \times 37 - \frac{4^6}{\dots}}}}$$

$$\frac{2}{-1+2G} = 3 + 0 \times 7 - \frac{6 \times 1^3}{3 + 1 \times 10 - \frac{8 \times 2^3}{3 + 2 \times 13 - \frac{10 \times 3^3}{\dots}}}$$
(4)



Srinivasa Ramanujan: The mathematical genius who credited his 3900 formulae to visions from Goddess Mahalakshmi

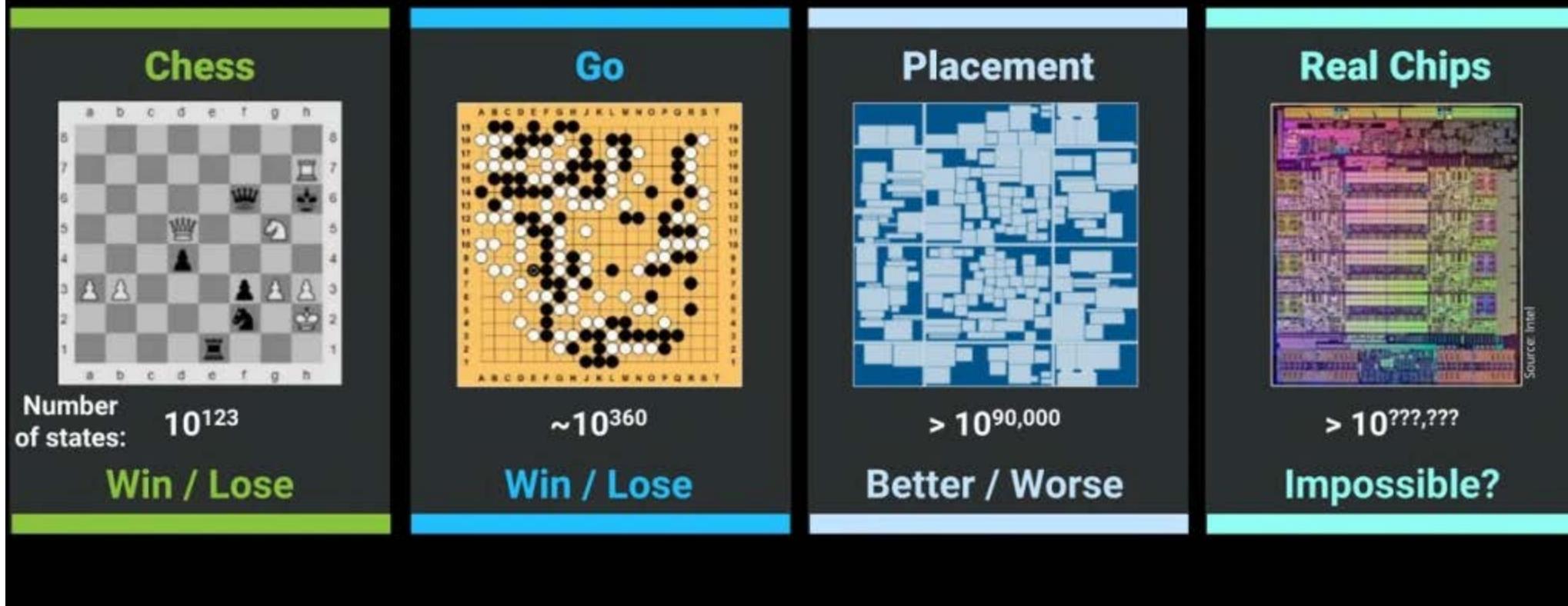




# Floor planning

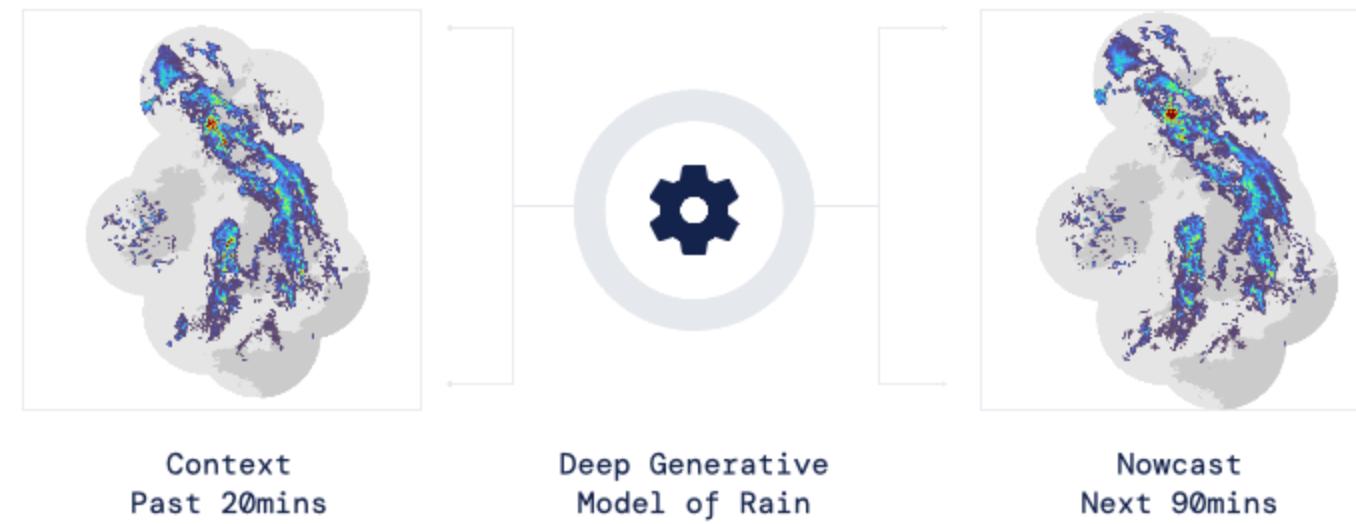
## Can We Design Personalized Chips?

*Systemic Complexity: 18-24 months and 100s of millions to bring a new chip to market*

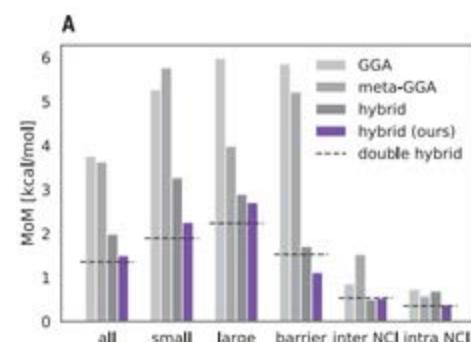
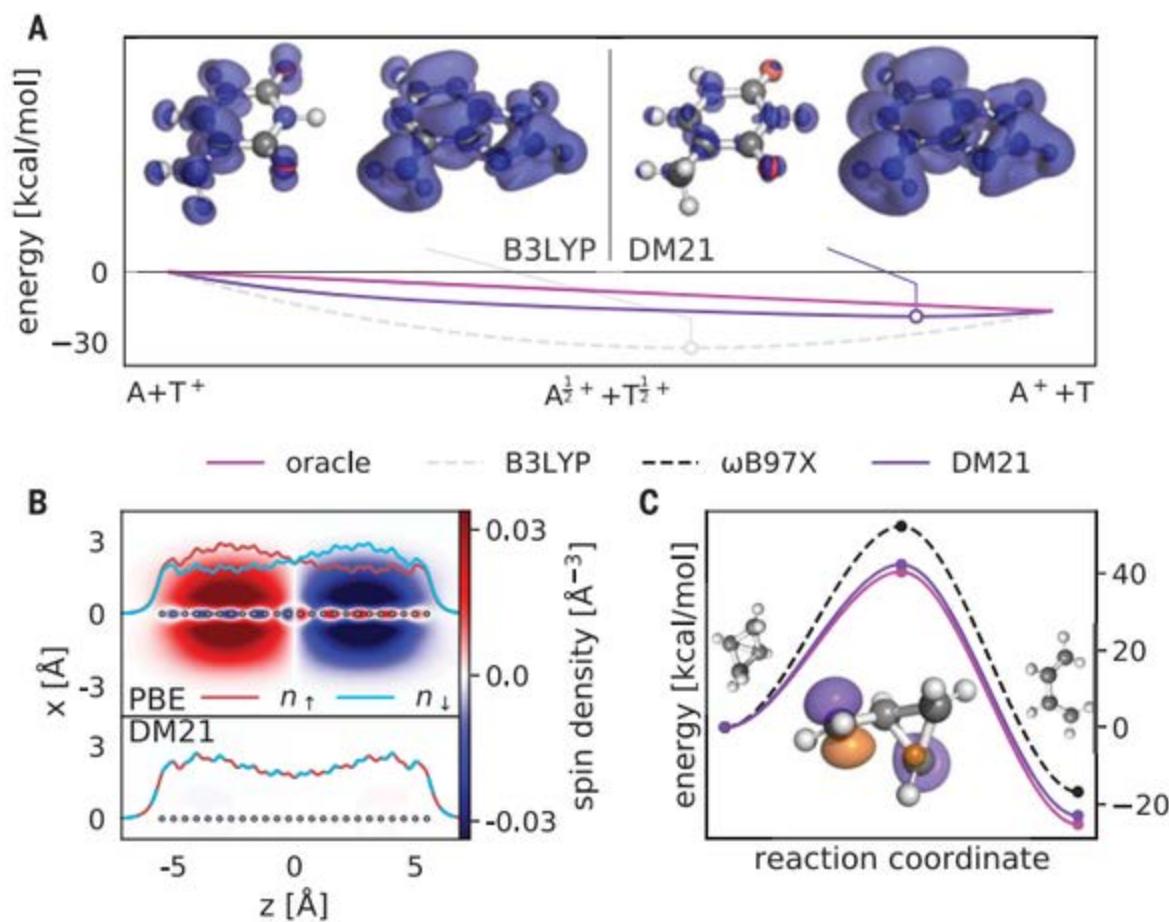




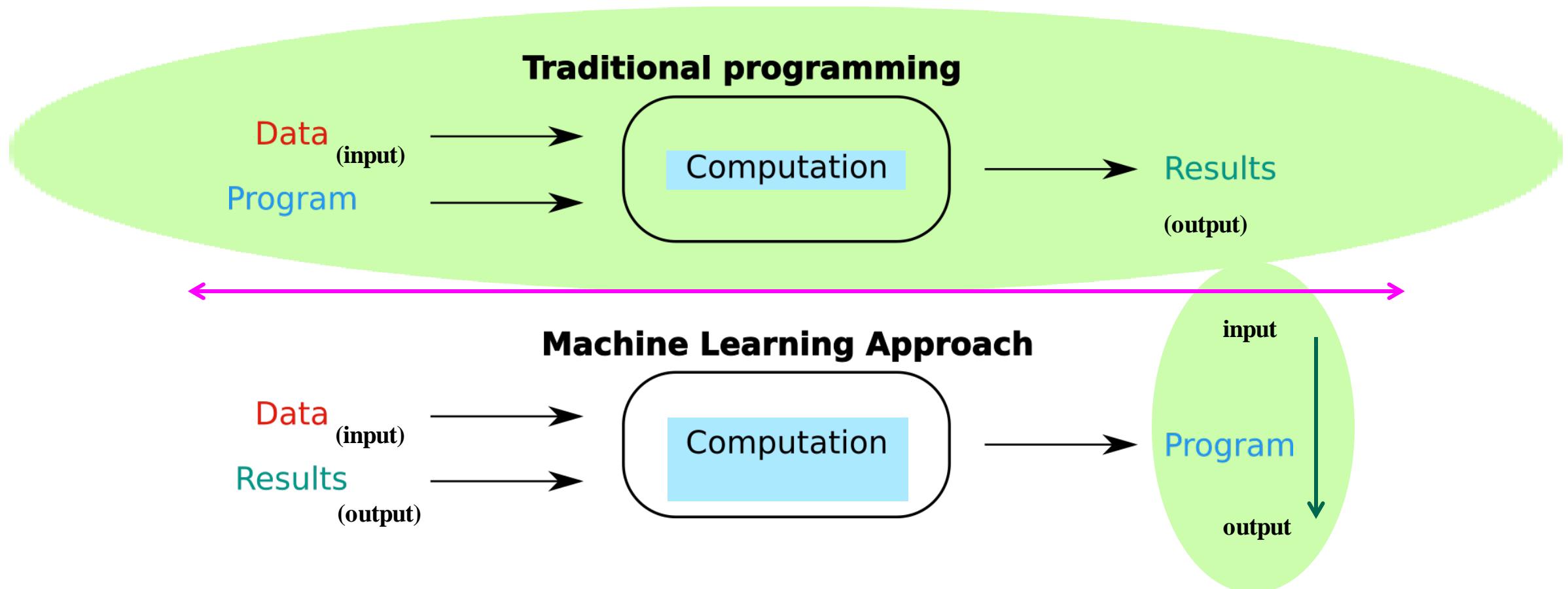
# Generative models for nowcasting



# Density functionals



# ML?



learning data **representations**, as opposed to task-specific algorithms



# Forms of Machine Learning

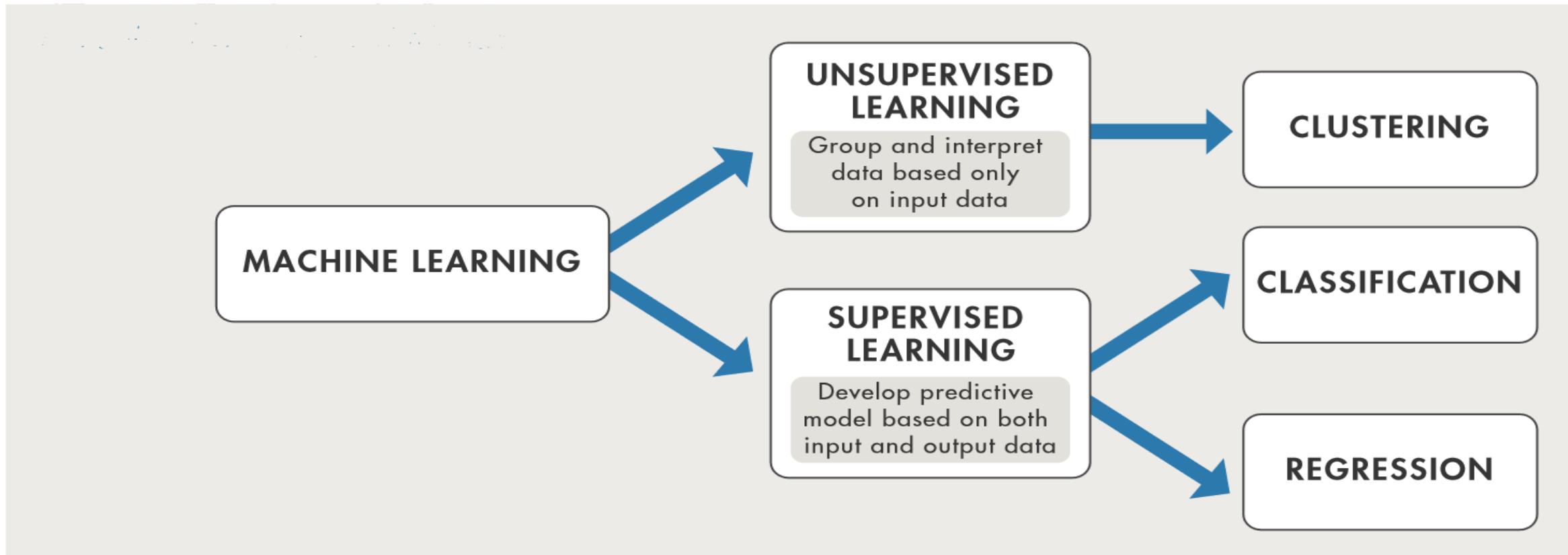
- \* **Supervised learning**
- \* **Unsupervised learning**
- \* **Reinforcement learning**
- \* **Evolutionary learning**

*Learning types*

*Three elements*

**Learning task = Representation + Evaluation + Optimization**

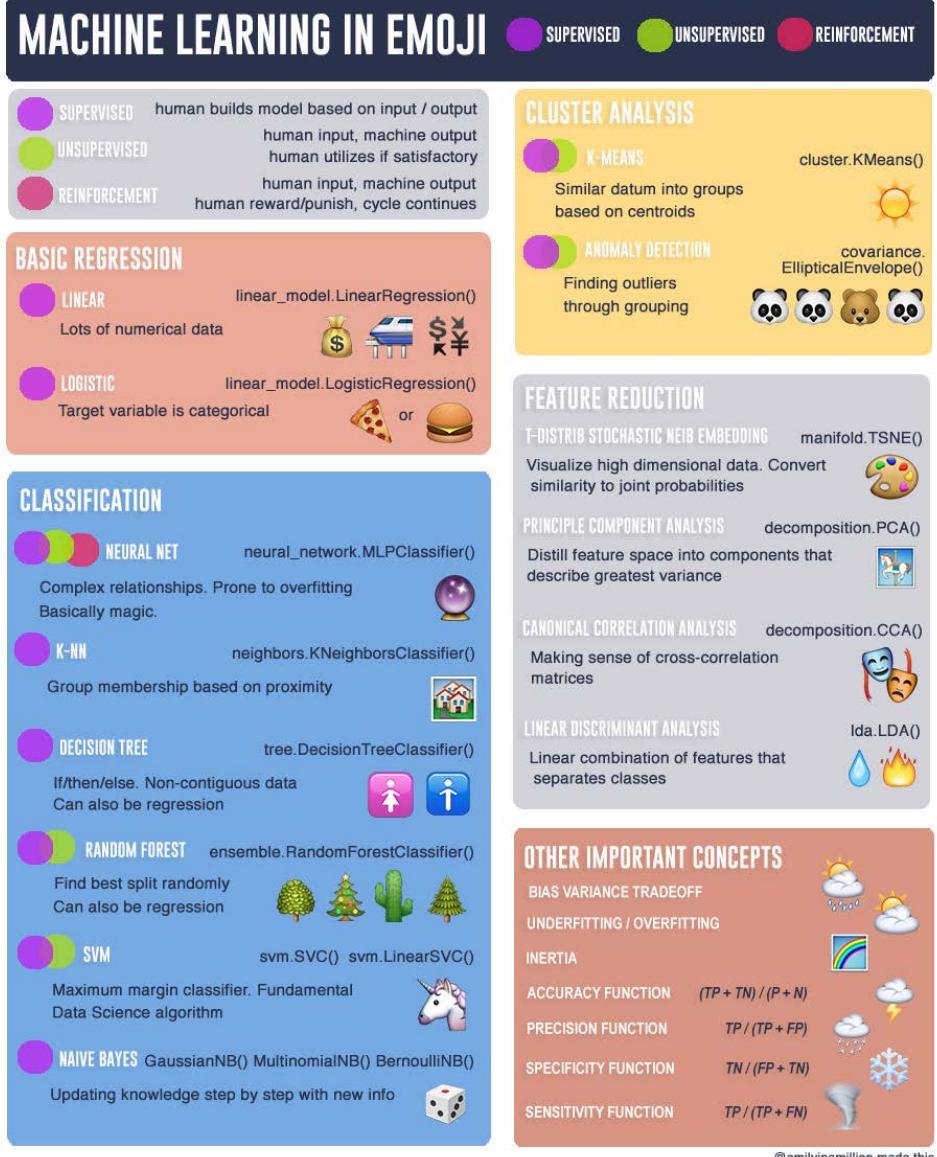
# ML?

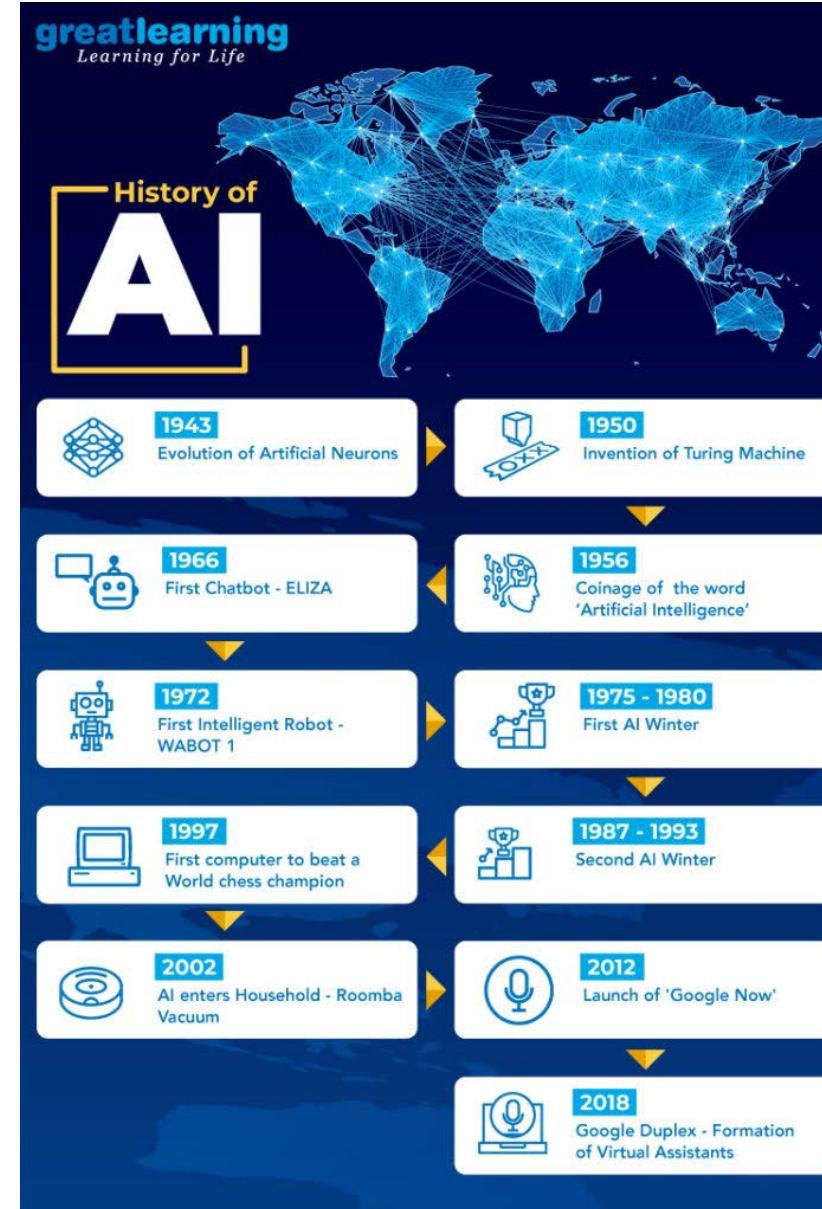


Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

# ML?

## MACHINE LEARNING IN EMOJI

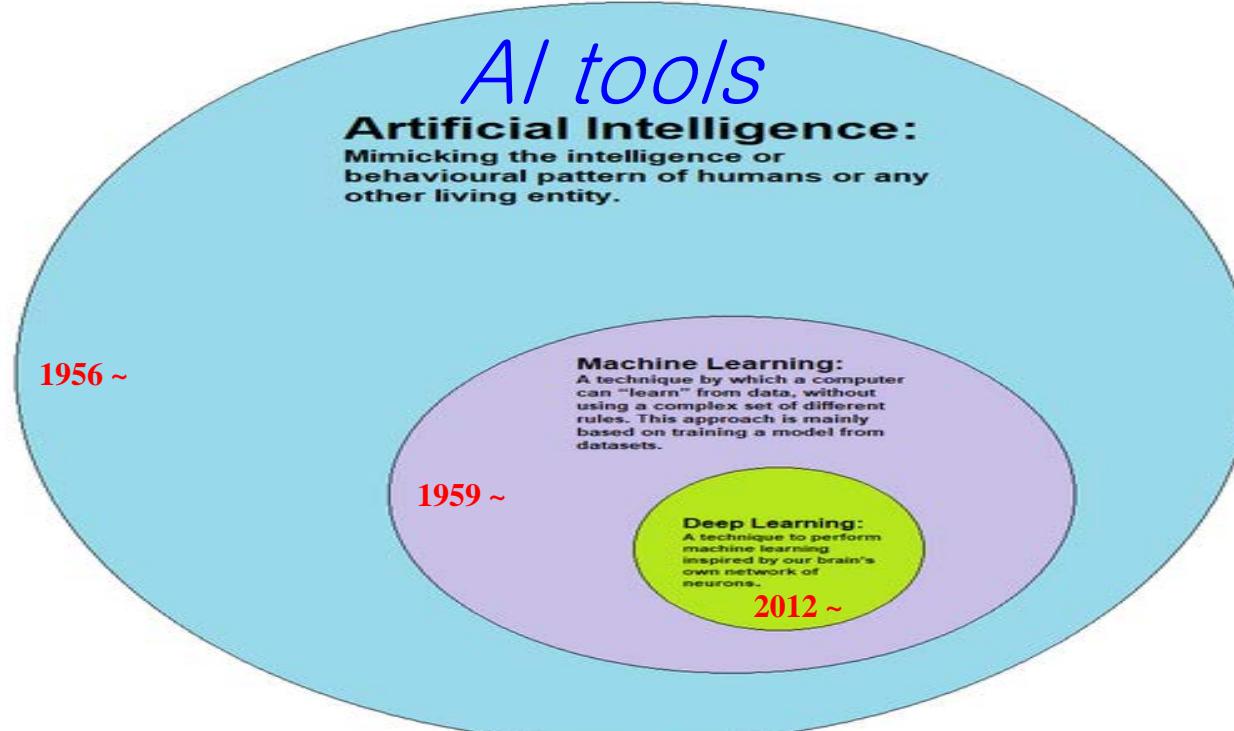




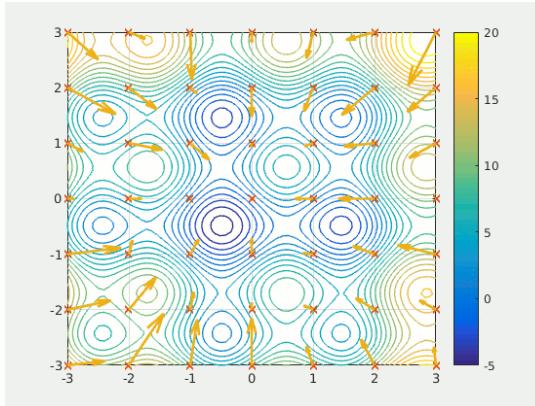
## Goals of Artificial Intelligence

**Problem-solving**  
**Knowledge representation**  
**Planning**  
**Learning**  
**Social Intelligence**

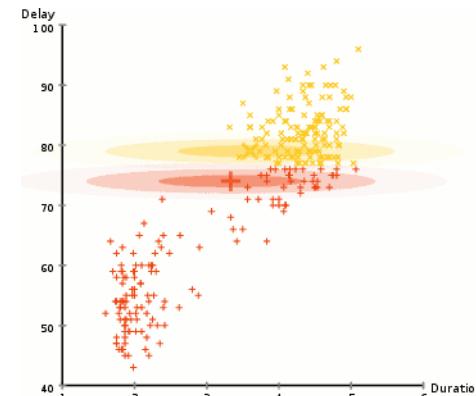
The overall research goal of artificial intelligence is to create technology that allows computers and machines to function in an intelligent manner.



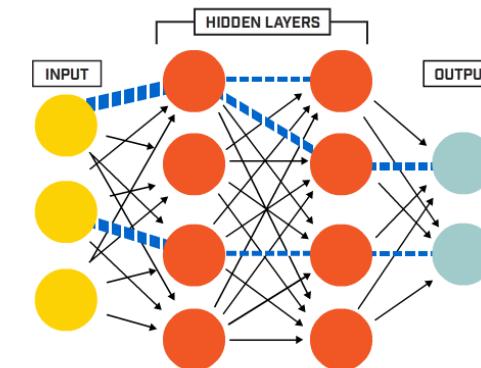
### Search and optimization



### Classifiers and statistical learning methods

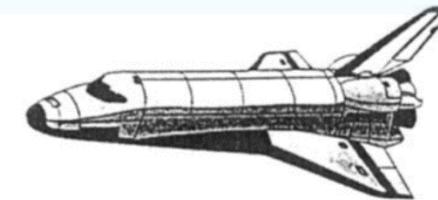
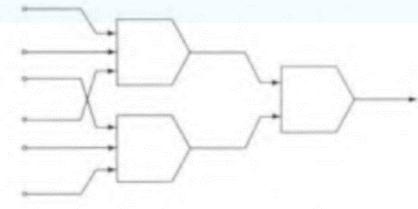
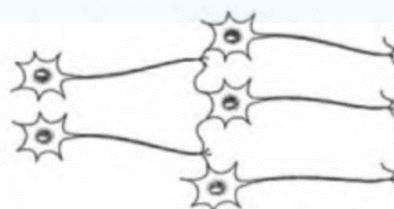
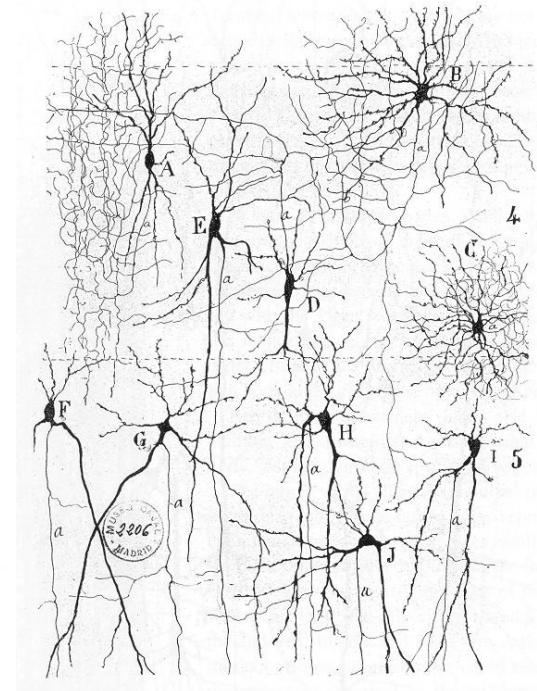
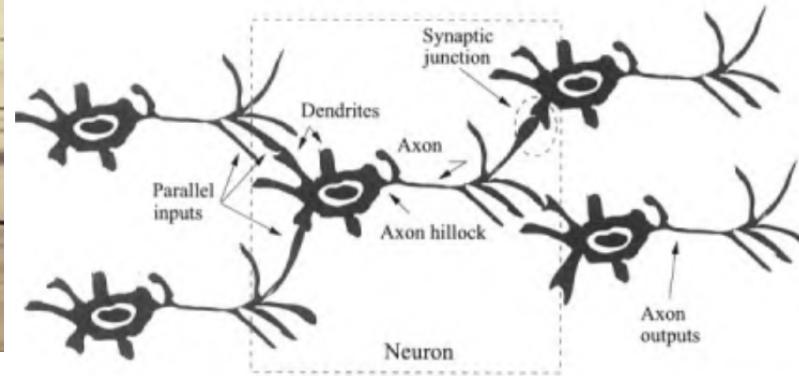


### Artificial neural networks



The overall research goal of artificial intelligence is to create technology that allows computers and machines to function in an intelligent manner.

# Biological neuron

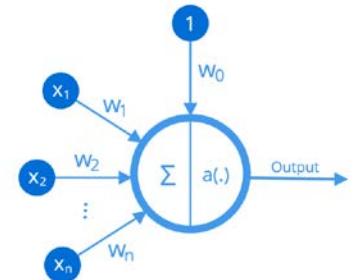
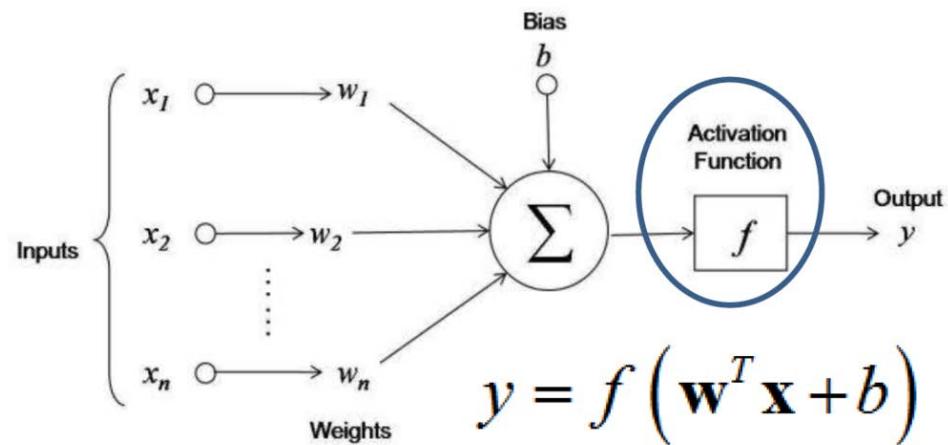


[https://en.wikipedia.org/wiki/Biological\\_neural\\_network](https://en.wikipedia.org/wiki/Biological_neural_network)

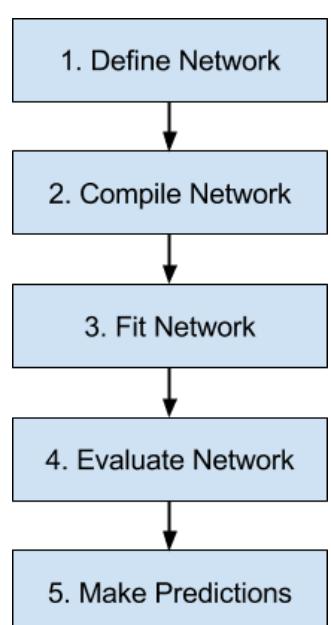
M. M. Gupta, L. Jin, N. Homma, *Static and Dynamic Neural Networks*, Wiley, 2003.

A. P. Engelbrecht, *Computational Intelligence*, 2<sup>nd</sup> ed., Wiley, 2007.

# An artificial neuron

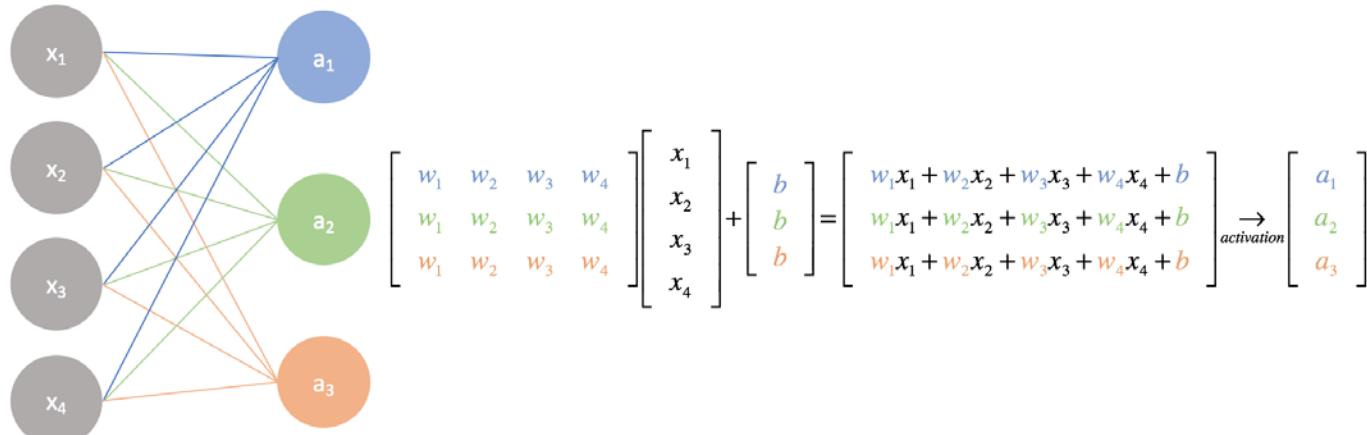


Nonlinear Function Implemented by a Neuron



Input layer      Output layer

A simple neural network





# History?

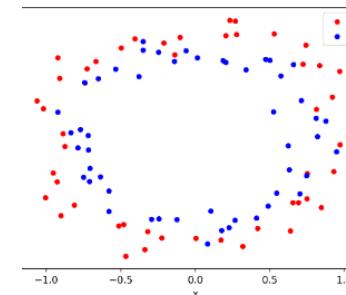
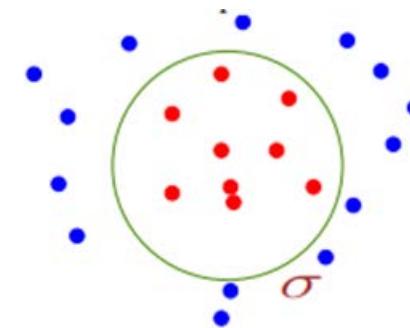
1957	Perceptron
1969	XOR argument (Minsky & Papert)
1982	Hopfield model
1985	Boltzmann machine
1986	Backpropagation algorithm, RBM
1989	Universal approximation theorem (Cybenko)
1999	Information bottleneck theory
2002	Contrastive divergence algorithm
2006	Deep belief network
2009	Recurrent temporal RBM
2016	AlphaGo (deep learning)



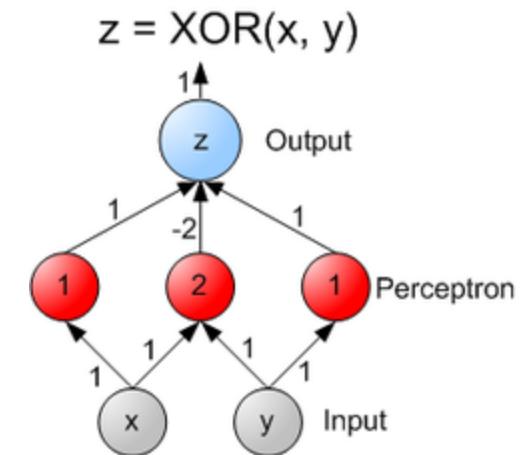
# Linearly separable?

- Two Boolean functions are examples of linearly separable functions.
- An example of a Boolean function that is not linearly separable is the **XOR**.

**Theorem:** Two sets of points **R** and **B** are separable by a circle in two dimensions, if and only if  $z(\mathbf{R})$  and  $z(\mathbf{B})$  are separable by a plane in three dimensions.

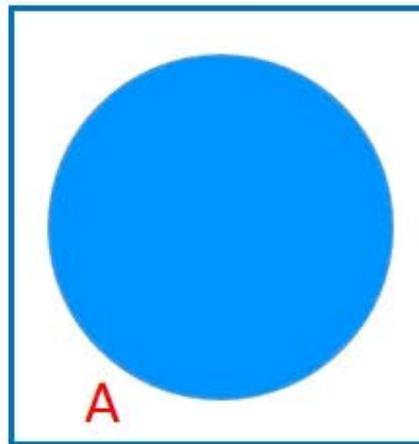


**2 dimensions : line,  
3 dimensions : plane,  
....**

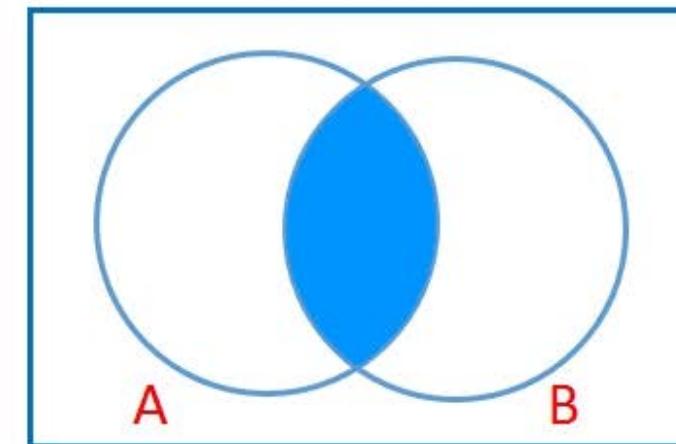




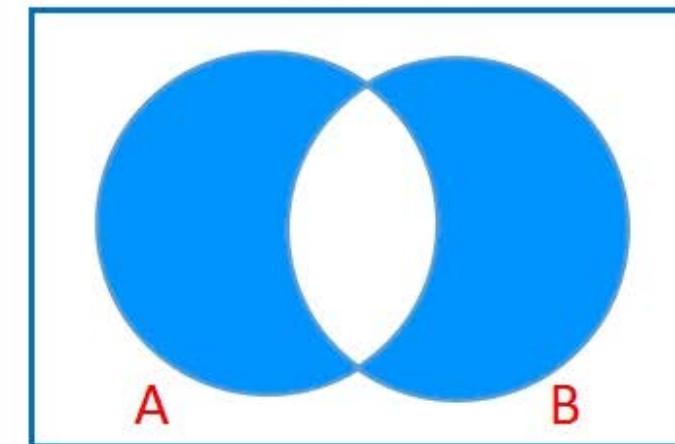
A



AND

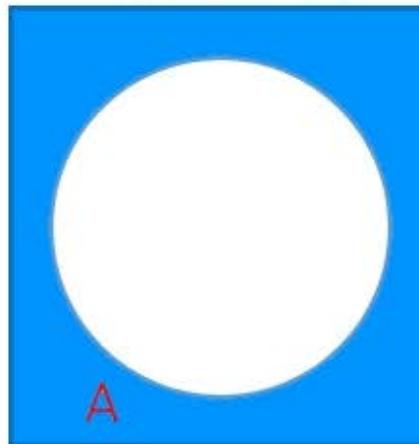


XOR

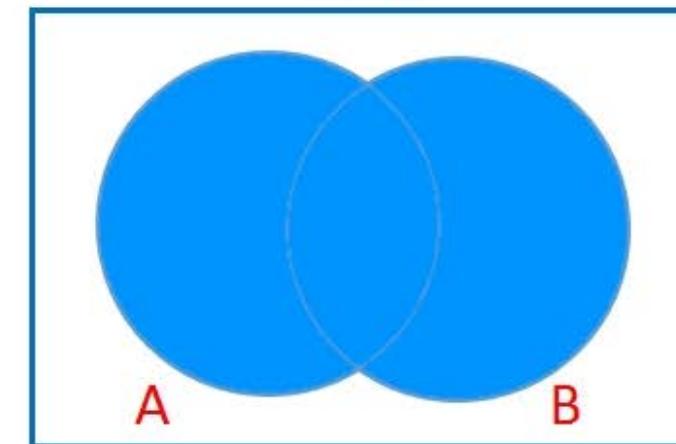


[www.electronics-micros.com](http://www.electronics-micros.com)

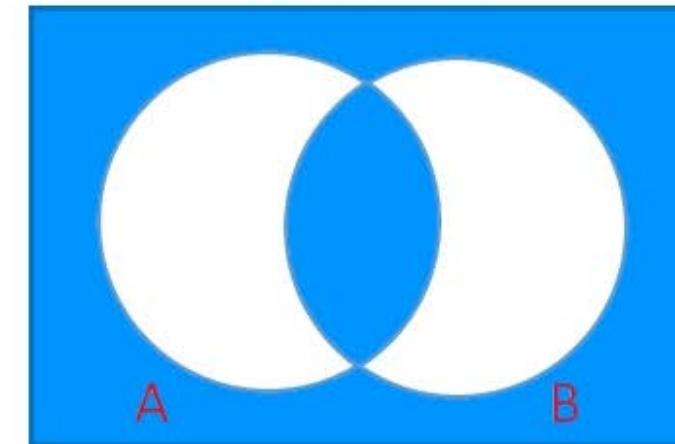
NOT A



OR



XNOR

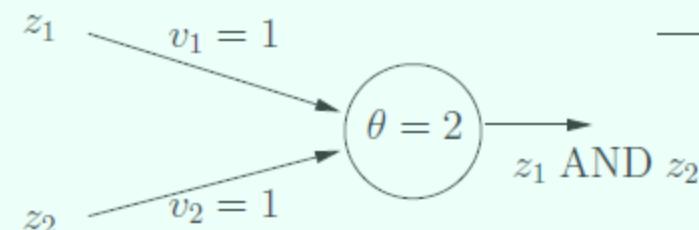




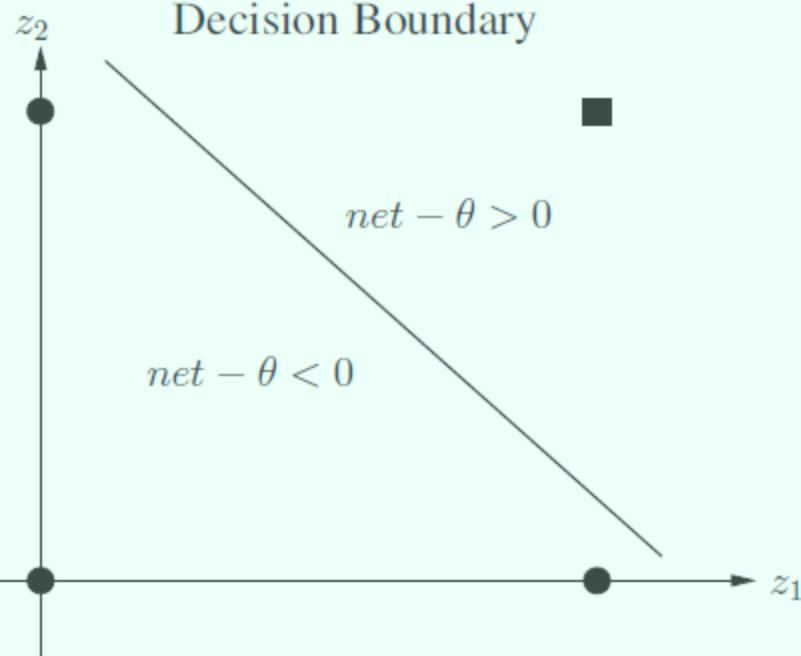
Truth Table

$z_1$	$z_2$	$z_1$ AND $z_2$
0	0	0
0	1	0
1	0	0
1	1	1

The Artificial Neuron



Decision Boundary



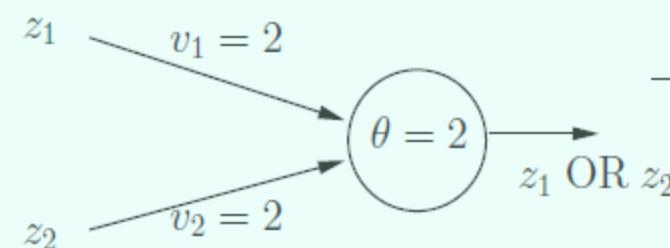
(a) AND Perceptron



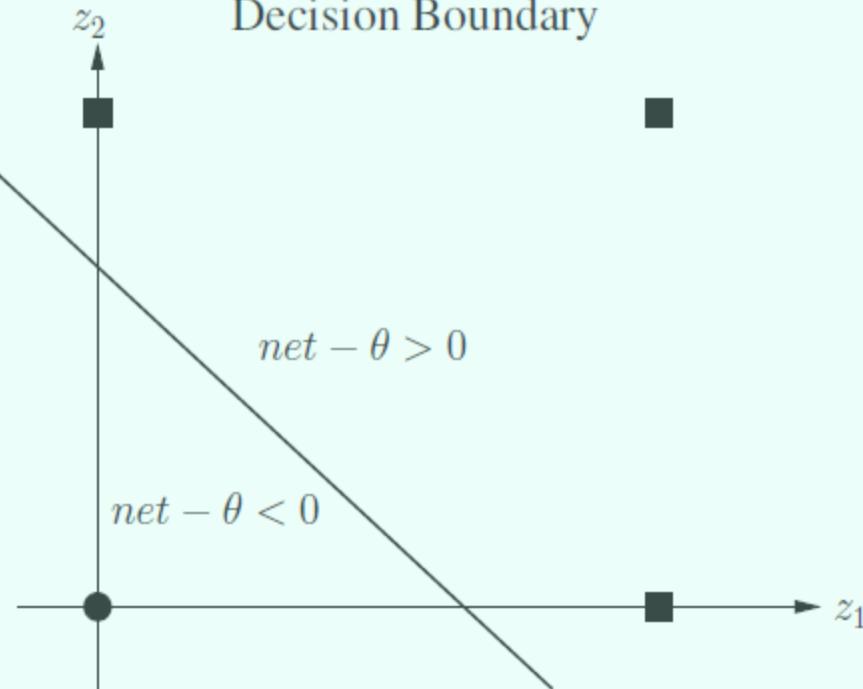
Truth Table

$z_1$	$z_2$	$z_1$ OR $z_2$
0	0	0
0	1	1
1	0	1
1	1	1

The Artificial Neuron



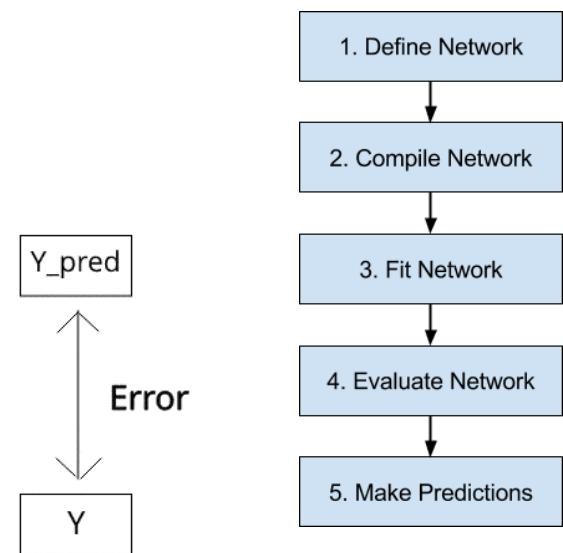
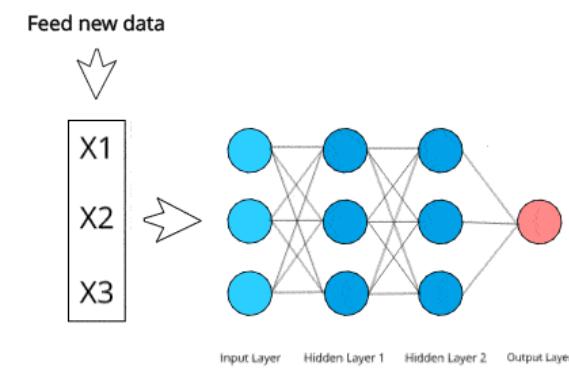
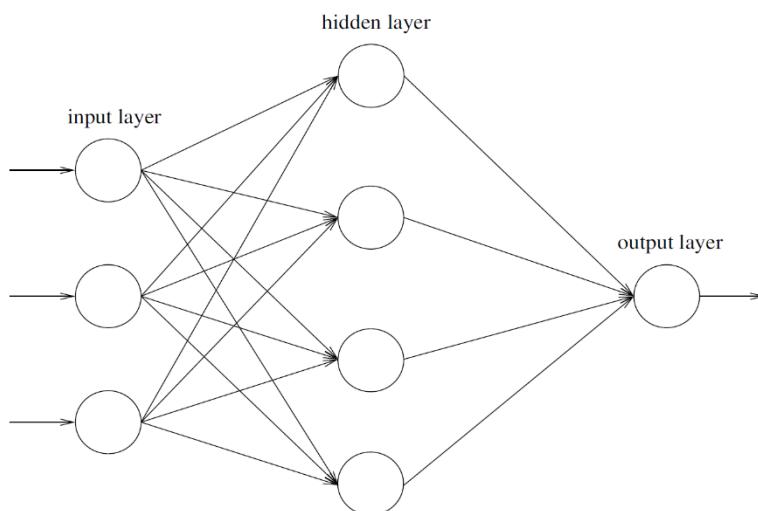
Decision Boundary



(b) OR Perceptron

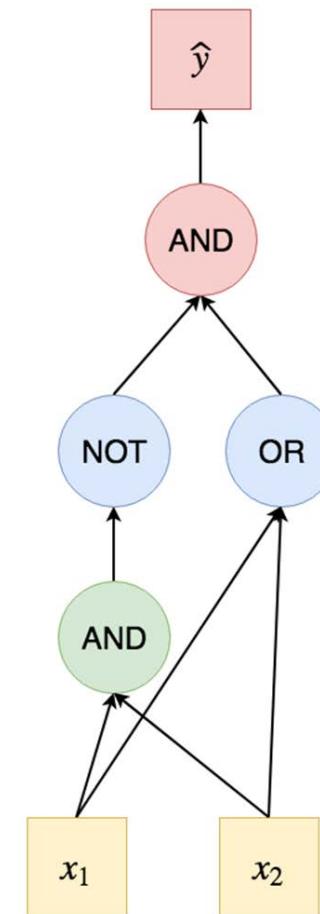
# Linearly separable?

- Since there is no straight line that can separate these patterns, we conclude that the XOR function is not a linearly separable function.
- The XOR requires two hidden layers.



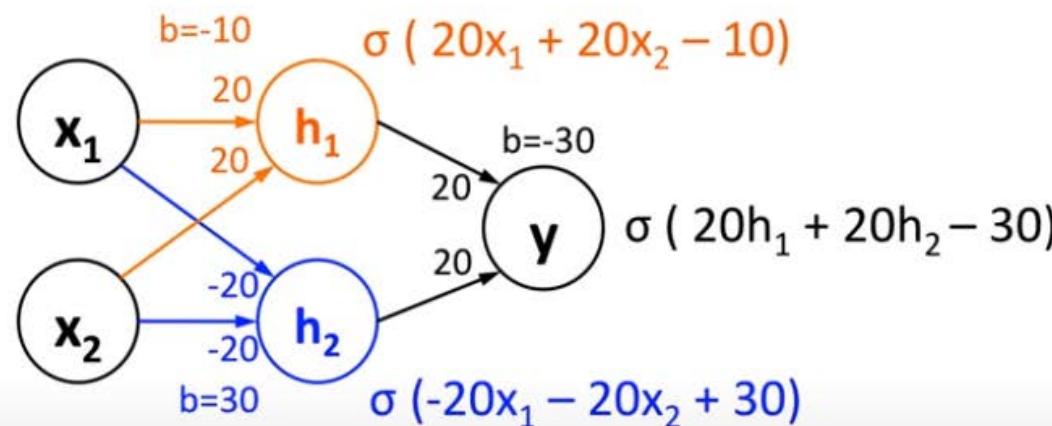
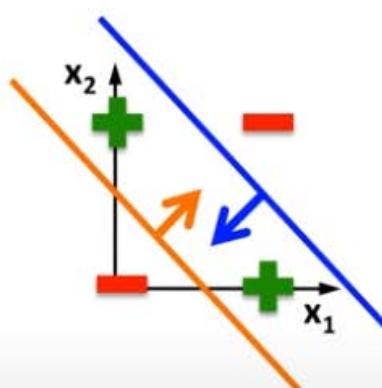
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

$$XOR(x_1, x_2) = AND(NOT(AND(x_1, x_2)), OR(x_1, x_2))$$



## Solving XOR with a Neural Net

Linear classifiers  
cannot solve this

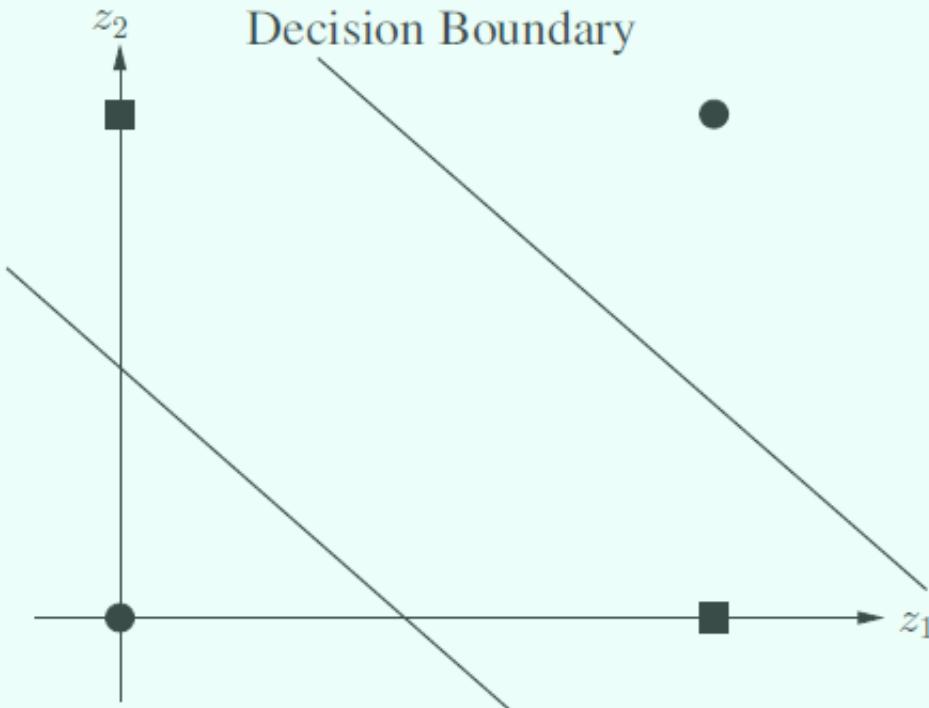


$$\begin{array}{lll}
 \sigma(20*0 + 20*0 - 10) \approx 0 & \sigma(-20*0 - 20*0 + 30) \approx 1 & \sigma(20*0 + 20*1 - 30) \approx 0 \\
 \sigma(20*1 + 20*1 - 10) \approx 1 & \sigma(-20*1 - 20*1 + 30) \approx 0 & \sigma(20*1 + 20*0 - 30) \approx 0 \\
 \sigma(20*0 + 20*1 - 10) \approx 1 & \sigma(-20*0 - 20*1 + 30) \approx 1 & \sigma(20*1 + 20*1 - 30) \approx 1 \\
 \sigma(20*1 + 20*0 - 10) \approx 1 & \sigma(-20*1 - 20*0 + 30) \approx 1 & \sigma(20*1 + 20*1 - 30) \approx 1
 \end{array}$$

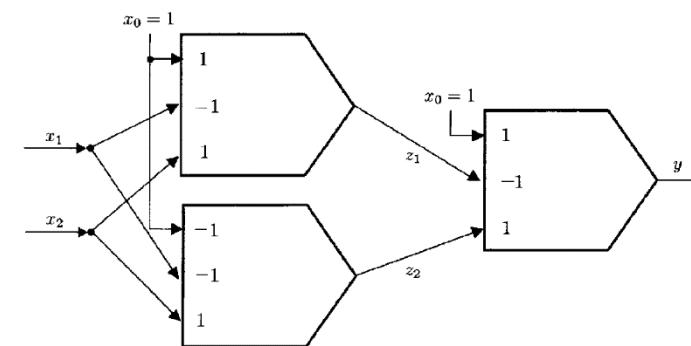


Truth Table

$z_1$	$z_2$	$z_1 \text{ XOR } z_2$
0	0	0
0	1	1
1	0	1
1	1	0

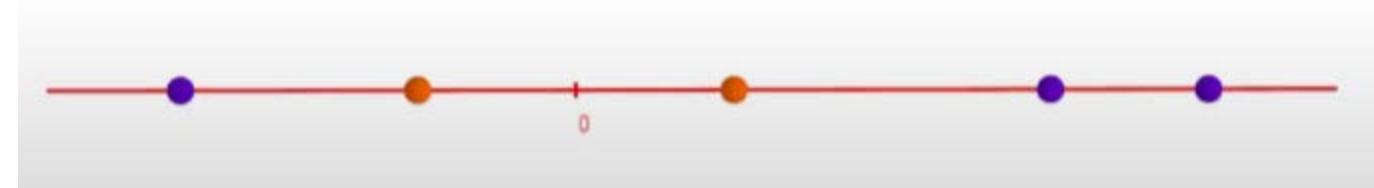


$$\begin{aligned}
 y = f(x_1, x_2) &= x_1 \oplus x_2 \\
 &= \text{sgn}(1 - z_1 + z_2) \\
 &= \text{sgn} \left( 1 - \text{sgn}(1 - x_1 + x_2) + \text{sgn}(-1 - x_1 + x_2) \right)
 \end{aligned}$$



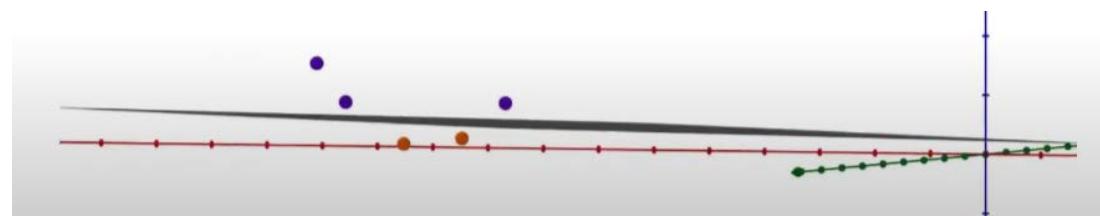
# Linearly separable in 3D/Kernel trick

$$X = \begin{bmatrix} -0.25 \\ -0.1 \\ 0.1 \\ 0.3 \\ 0.4 \end{bmatrix}, y = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$



$$\phi(x) = [1 \ \sqrt{2}x \ x^2]$$

$$X_{new} = \begin{bmatrix} 1 & -0.3535 & 0.0625 \\ 1 & -0.1414 & 0.01 \\ 1 & 0.1414 & 0.01 \\ 1 & 0.4242 & 0.09 \\ 1 & 0.5656 & 0.16 \end{bmatrix}$$



# Kernel trick

## 2.2 Mercer's Theorem

A symmetric function  $K(x, y)$  can be expressed as an inner product

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

for some  $\phi$  if and only if  $K(x, y)$  is positive semidefinite, i.e.

$$\int K(x, y)g(x)g(y)dxdy \geq 0 \quad \forall g$$

or, equivalently:

$$\begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ K(x_2, x_1) & \ddots & \\ \vdots & & \end{bmatrix} \text{ is psd for any collection } \{x_1 \dots x_n\}$$



# Universal approximation theorem

- The theorem states that simple neural networks can *represent* a wide variety of functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters.

$$F(x) = \sum_{i=1}^N v_i \varphi(w_i^T x + b_i)$$

**Rotation and shift**

***n*-dimensional input space,**  
**width-*n+4* networks with ReLU activation functions,**  
***n+1* networks with ReLu,**  
***if network depth is allowed to grow***

## Curse of Dimensionality

# Functional approximation capability of a feedforward neural network architecture

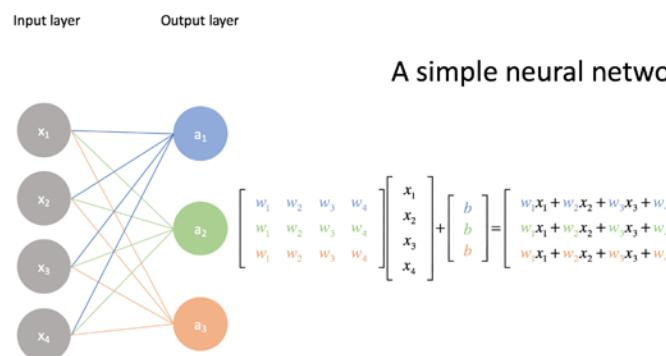
Feedforward neural networks are universal approximators.

Gaussian radial basis function networks are universal approximators.

Hidden layers

Learning algorithm

Sufficient information (learning signals)



$$\begin{aligned} y &= \sum_{i=1}^L w_i u_i(\vec{x}) \\ &= \sum_{i=1}^L w_i \exp \left[ -\frac{1}{2} \sum_{k=1}^K \left( \frac{x_k - c_{ki}}{\sigma_{ki}} \right)^2 \right]. \end{aligned}$$

parameters ( $\{c_{ki}\}$ ,  $\{\sigma_{ki}^2\}$ , and  $\{w_i\}$ )

$$\left\{ \frac{\partial y}{\partial w_i}, \frac{\partial y}{\partial c_{ki}}, \frac{\partial y}{\partial \sigma_{ki}} \right\}$$



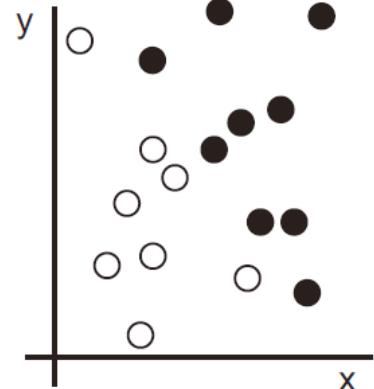
# Implications for neural networks

- (i) Identity function:  $f(x) = 1$ .**
- (ii) Separability of the function: [  $x_1 \neq x_2, f(x_1) \neq f(x_2)$  ]**
- (iii) Algebraic closure : nonlinear mappings, [  $fg, \alpha f + \beta g$  ]**

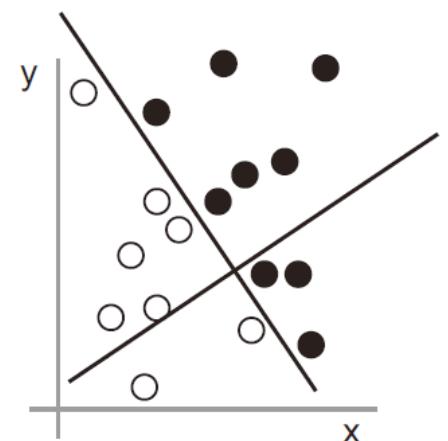


# a new representation

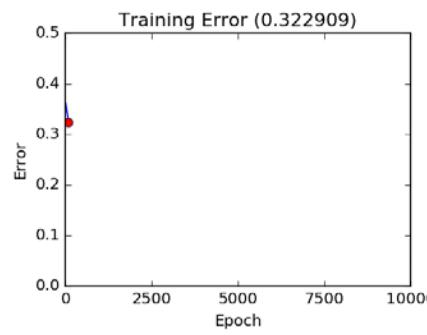
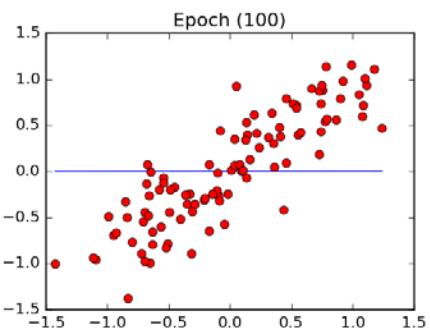
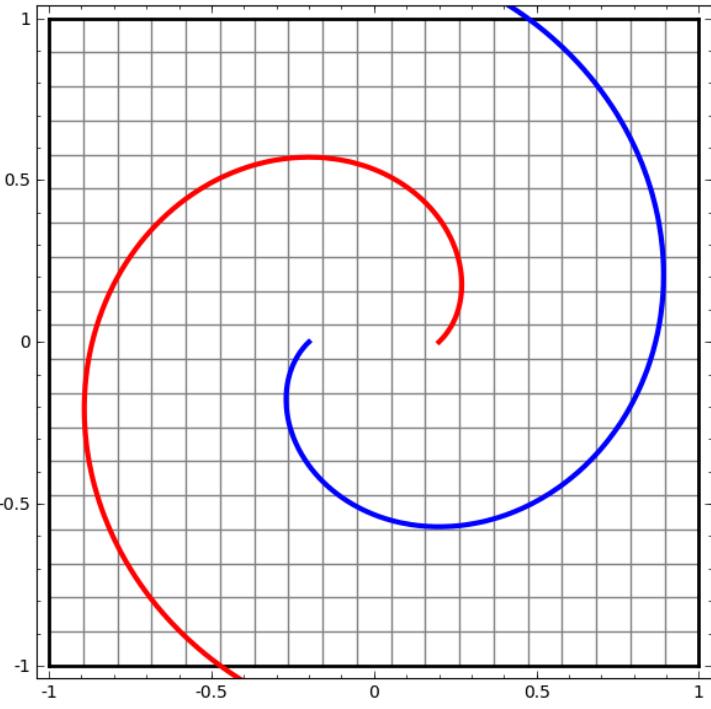
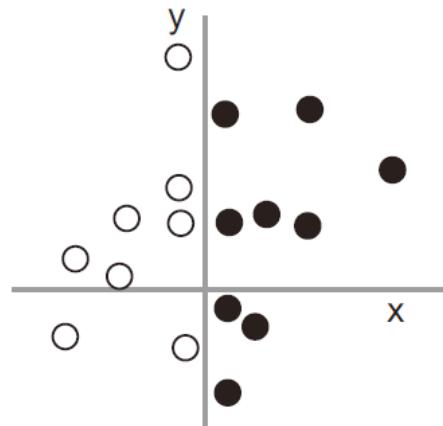
1: Raw data



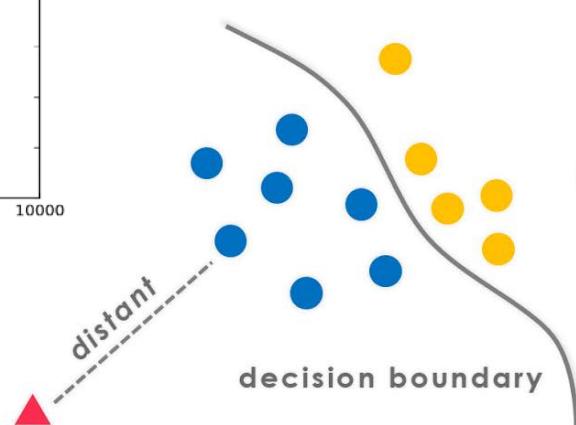
2: Coordinate change



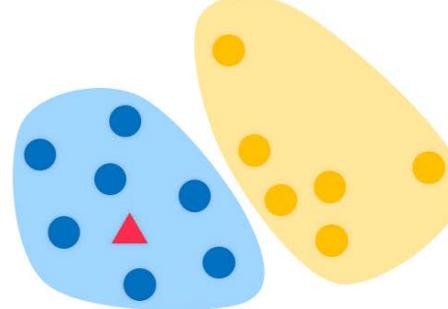
3: Better representation



Discriminative

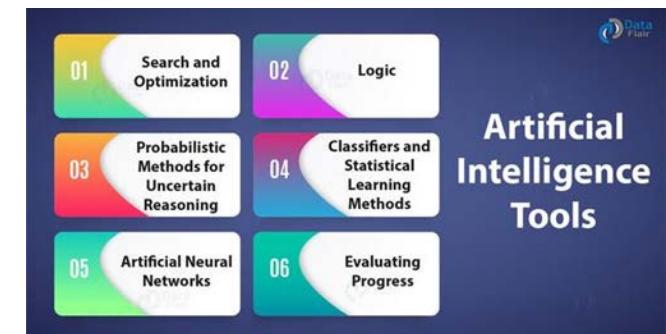
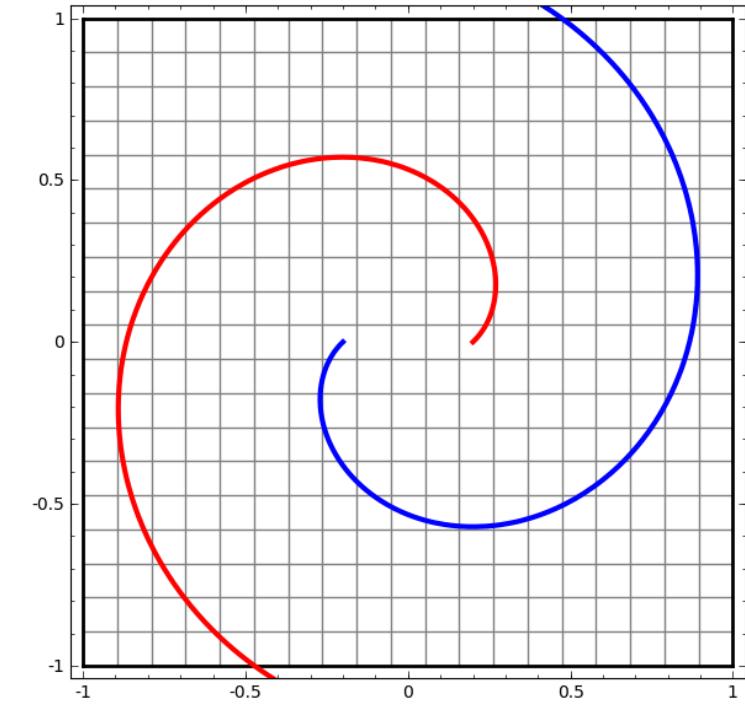
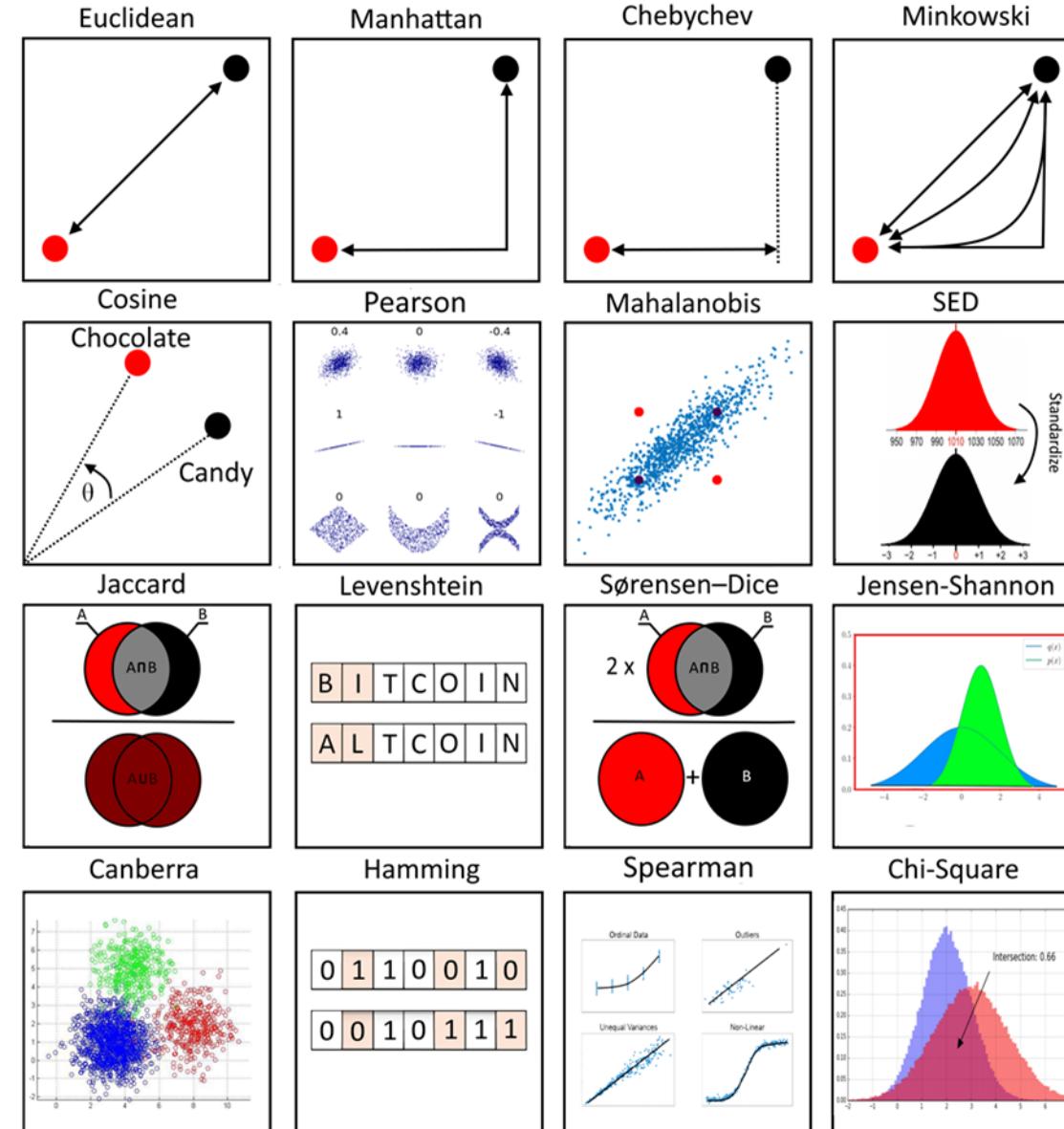


Generative





# 16 distance measures





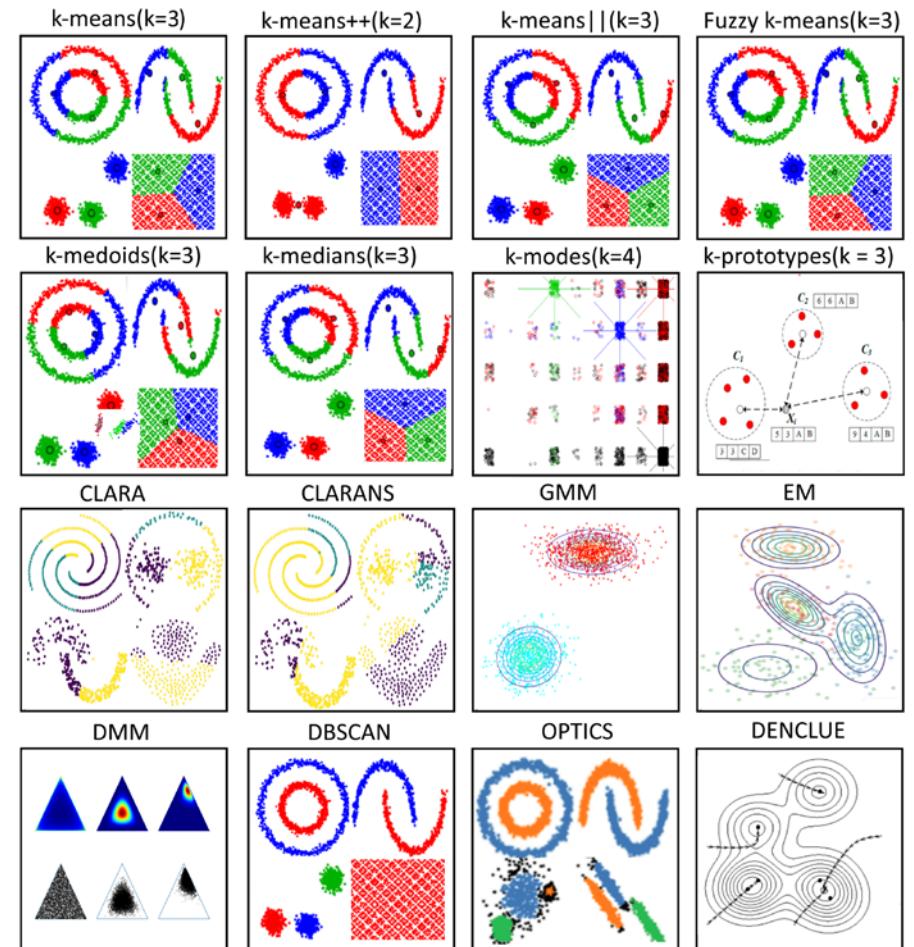
# Tow images

```
from skimage.measure import compare_ssim, compare_psnr, compare_mse
import cv2
img1 = cv2.imread(r'C:\Users\Wxr\Desktop\WATERLOO\train_low_high\29.jpg')
img2 = cv2.imread(r'C:\Users\Wxr\Desktop\WATERLOO\train_normal\29.jpg')
psnr = compare_psnr(img1, img2)
ssim = compare_ssim(img1, img2, multichannel=True) # For multichannel images (RGB, HSV etc. ) key
word multichannel To set up for True
mse = compare_mse(img1, img2)
print('PSNR : {},SSIM : {},MSE : {}'.format(psnr, ssim, mse))
```

# Clustering algorithms

## Clustering Algorithms

- Centroid-based clustering
  - k-means
  - k-means++
  - k-means||
  - Fuzzy C-means
  - k-medoids, PAM
  - k-Medians
  - k-Modes
  - k-prototypes
  - CLARA
  - CLARANS
- Distribution-based clustering
  - GMM
  - EM
  - DMM
- Density-based clustering
  - DBSCAN
  - ADBS CAN
  - DENCLUE
  - OPTICS

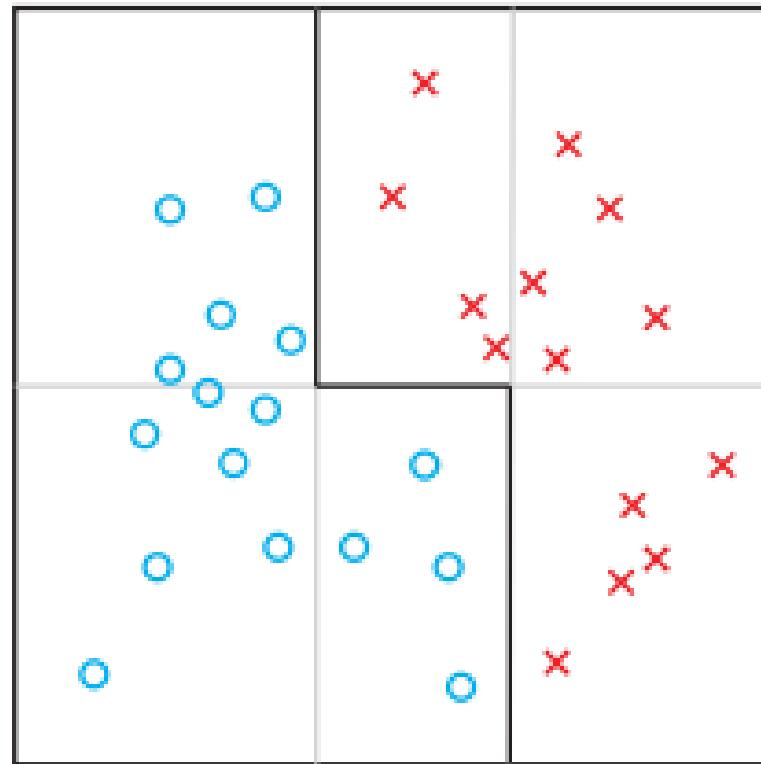




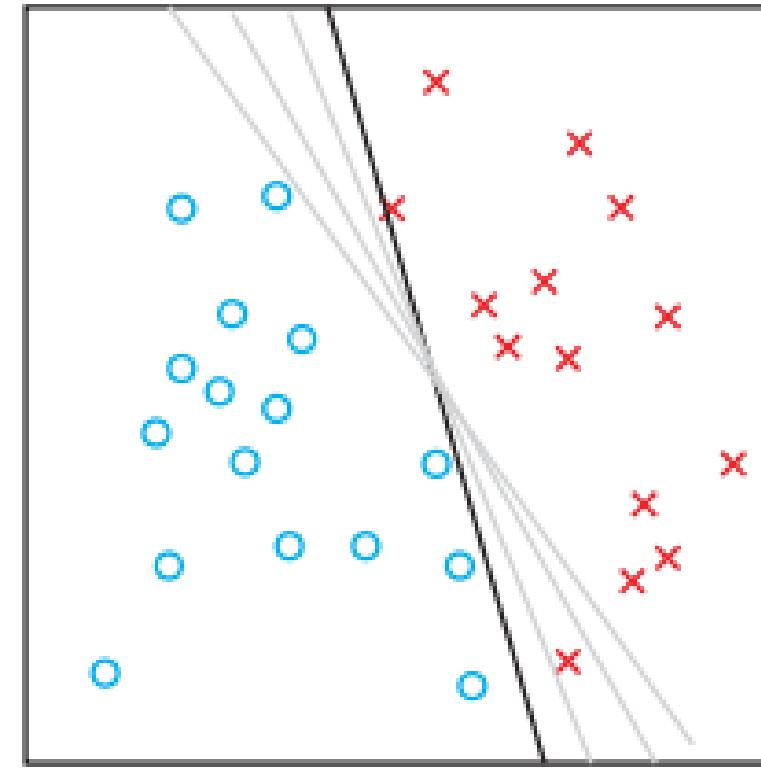
식품산업 바꾸는 인공지능(AI)

자료=각 사

	<b>미국 테이스트리</b>	AI로 수천 종 와인 분석해 소비자 기호에 맞는 와인 제안
	<b>미국 아로믹스</b>	AI로 인간 후각수용체와 향기 반응 분석해 차 품질 분석
	<b>미국 IBM</b>	액체 맛 평가하는 전자혀 하이퍼테이스트와 AI로 조미료 개발
	<b>덴마크 칼스버그</b>	AI가 효모와 발효 성분을 토대로 맥주 맛 예측
	<b>칠레 낫코</b>	AI로 식물성분 구조 분석해 육류 대체하는 성분 배합 개발
	<b>콜롬비아 데메트리아</b>	AI가 생두의 유기분자 분석 결과를 토대로 커피 맛과 향 예측



Aggregating different linear hypotheses



Linear perceptron hypothesis



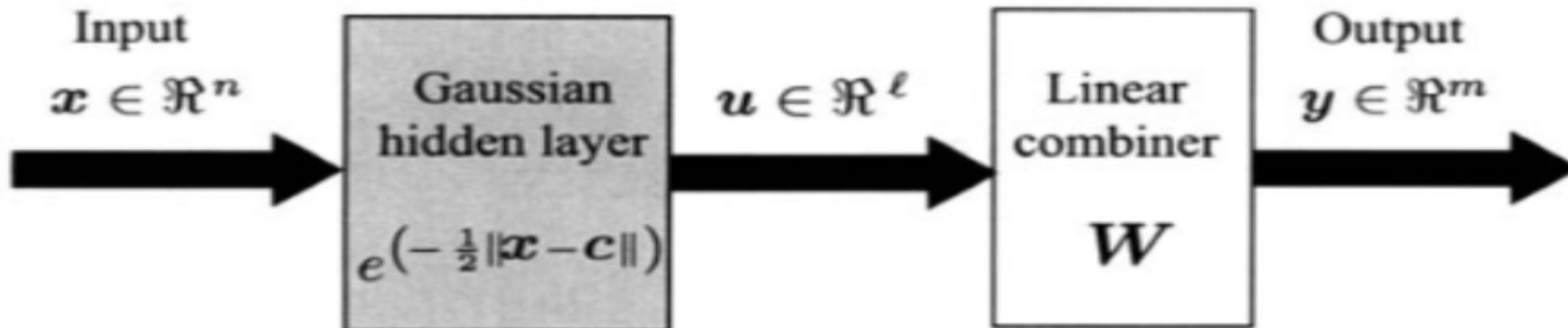
# Supervised (gradient descent) parameter learning in Gaussian networks

**Distance vs dot product**

$$\begin{aligned} y &= \sum_{i=1}^L w_i u_i(\vec{x}) \\ &= \sum_{i=1}^L w_i \exp \left[ -\frac{1}{2} \sum_{k=1}^K \left( \frac{x_k - c_{ki}}{\sigma_{ki}} \right)^2 \right]. \end{aligned}$$

parameters ( $\{c_{ki}\}$ ,  $\{\sigma_{ki}^2\}$ , and  $\{w_i\}$ )

$$\left\{ \frac{\partial y}{\partial w_i}, \frac{\partial y}{\partial c_{ki}}, \frac{\partial y}{\partial \sigma_{ki}} \right\}$$



$$\begin{aligned} y &= \sum_{i=1}^L w_i u_i(\vec{x}) \\ &= \sum_{i=1}^L w_i \exp \left[ -\frac{1}{2} \sum_{k=1}^K \left( \frac{x_k - c_{ki}}{\sigma_{ki}} \right)^2 \right]. \end{aligned}$$

parameters ( $\{c_{ki}\}$ ,  $\{\sigma_{ki}^2\}$ , and  $\{w_i\}$ )

**Distance vs dot product**

$$\left\{ \frac{\partial y}{\partial w_i}, \frac{\partial y}{\partial c_{ki}}, \frac{\partial y}{\partial \sigma_{ki}} \right\}$$

**Output of the  $i$ th hidden Gaussian neuron**

$$u_i = \exp \left( -\frac{1}{2} \sum_{k=1}^n \left[ \frac{x_k - c_{ik}}{\sigma_{ik}} \right]^2 \right), \quad 1 \leq i \leq \ell$$



$$\{x_j, d_j\}^{\{n, m\}}$$

$$E = \frac{1}{2} \sum_{j=1}^m (d_j - y_j)^2 = \frac{1}{2} \sum_{j=1}^m e_j^2$$

$$y = \sum_{i=1}^L w_i u_i(\vec{x})$$

$$e_j \triangleq d_j - y_j$$

$$= d_j - \sum_{i=1}^{\ell} w_{ij} \exp \left( -\frac{1}{2} \sum_{p=1}^n \left[ \frac{\mathbf{x}_p - c_{ip}}{\sigma_{ip}} \right]^2 \right)$$

$$= \sum_{i=1}^L w_i \exp \left[ -\frac{1}{2} \sum_{k=1}^K \left( \frac{x_k - c_{ki}}{\sigma_{ki}} \right)^2 \right].$$

parameters ( $\{c_{ki}\}$ ,  $\{\sigma_{ki}^2\}$ , and  $\{w_i\}$ )

$$u_i = \exp \left( -\frac{1}{2} \sum_{k=1}^n \left[ \frac{x_k - c_{ik}}{\sigma_{ik}} \right]^2 \right), \quad 1 \leq i \leq \ell$$

$$\left\{ \frac{\partial y}{\partial w_i}, \frac{\partial y}{\partial c_{ki}}, \frac{\partial y}{\partial \sigma_{ki}} \right\}$$

$$w_{ij}^{new} = w_{ij}^{old} + \eta_1 u_i e_j$$

$$c_{ip}^{new} = c_{ip}^{old} + \eta_2 \frac{(x_p - c_{ip})}{\sigma_{ip}^2} u_i e_j$$

$$\sigma_{ip}^{new} = \sigma_{ip}^{old} + \eta_3 \frac{(x_p - c_{ip})^2}{\sigma_{ip}^3} 2u_i e_j$$

$$1 \leq i \leq \ell; \quad 1 \leq j \leq m; \quad 1 \leq p \leq n$$

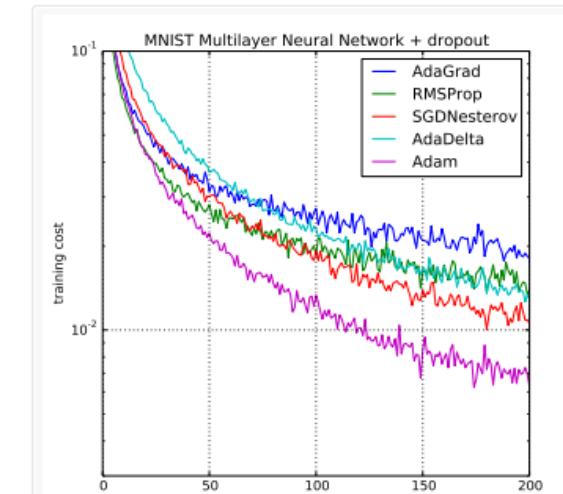
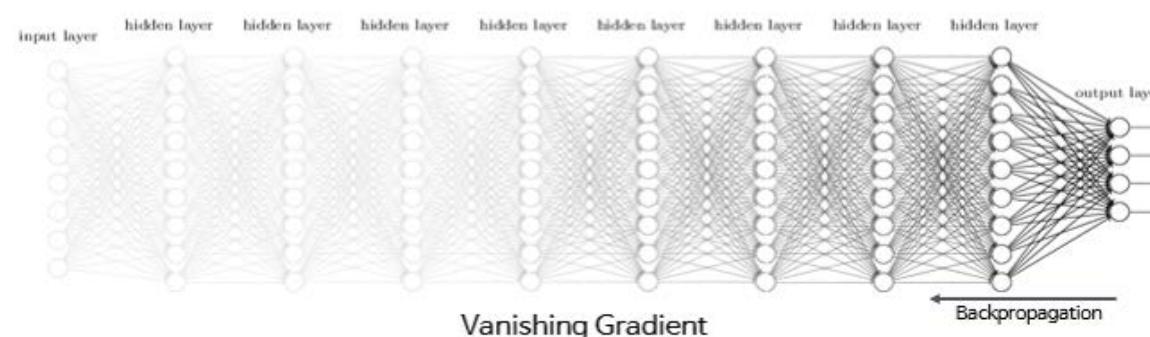
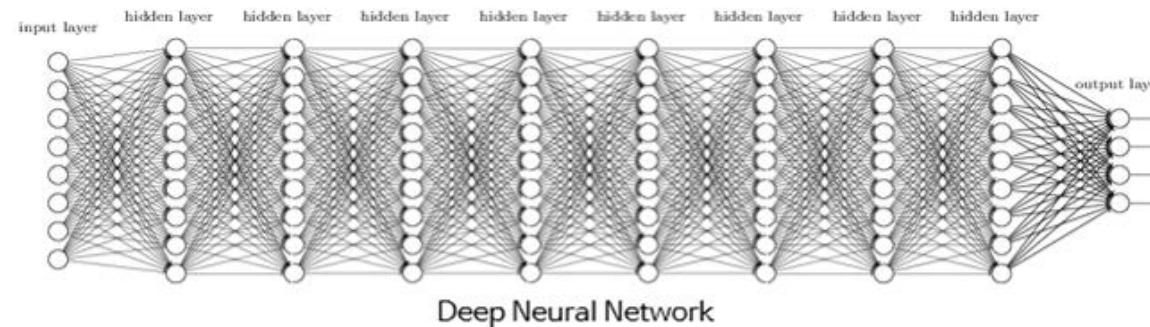
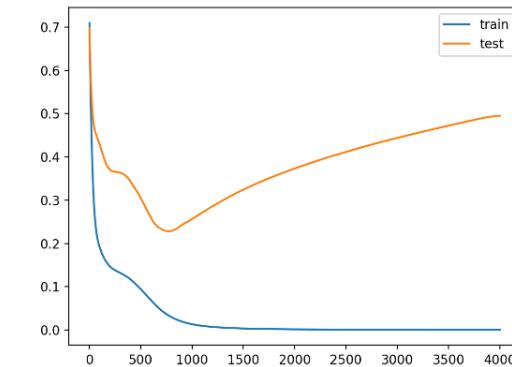
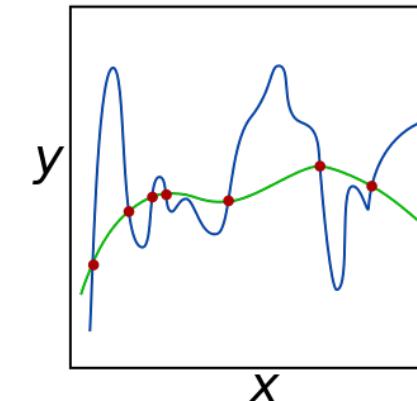
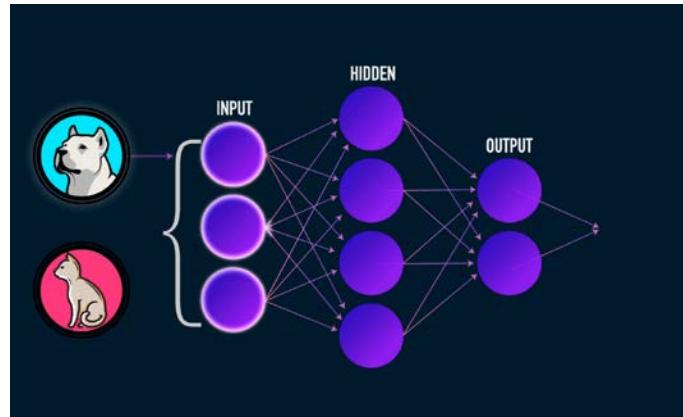


## DL(2012~) has achieved the following breakthroughs

- Near-human-level image classification
- Near-human-level speech recognition
- Near-human-level handwriting transcription
- Improved machine translation
- Improved text-to-speech conversion
- Digital assistants such as Google Now and Amazon Alexa
- Near-human-level autonomous driving
- Improved ad targeting, as used by Google, Baidu, and Bing
- Improved search results on the web
- Ability to answer natural-language questions
- Superhuman Go playing
- AlphaGo Zero, Alpha Zero
- AlphaFold, AlphaFold 2, MuZero



# Deep Learning (2012~)



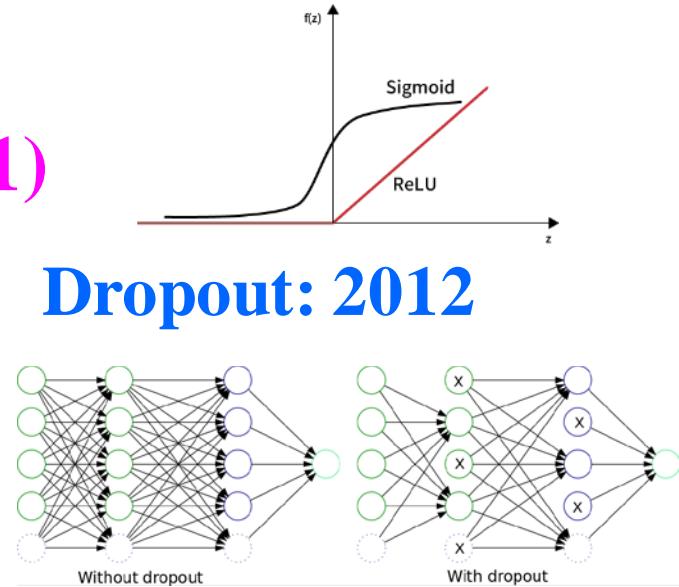
Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Taken from Adam: A Method for Stochastic Optimization, 2015.



# Deep Learning (2012~)

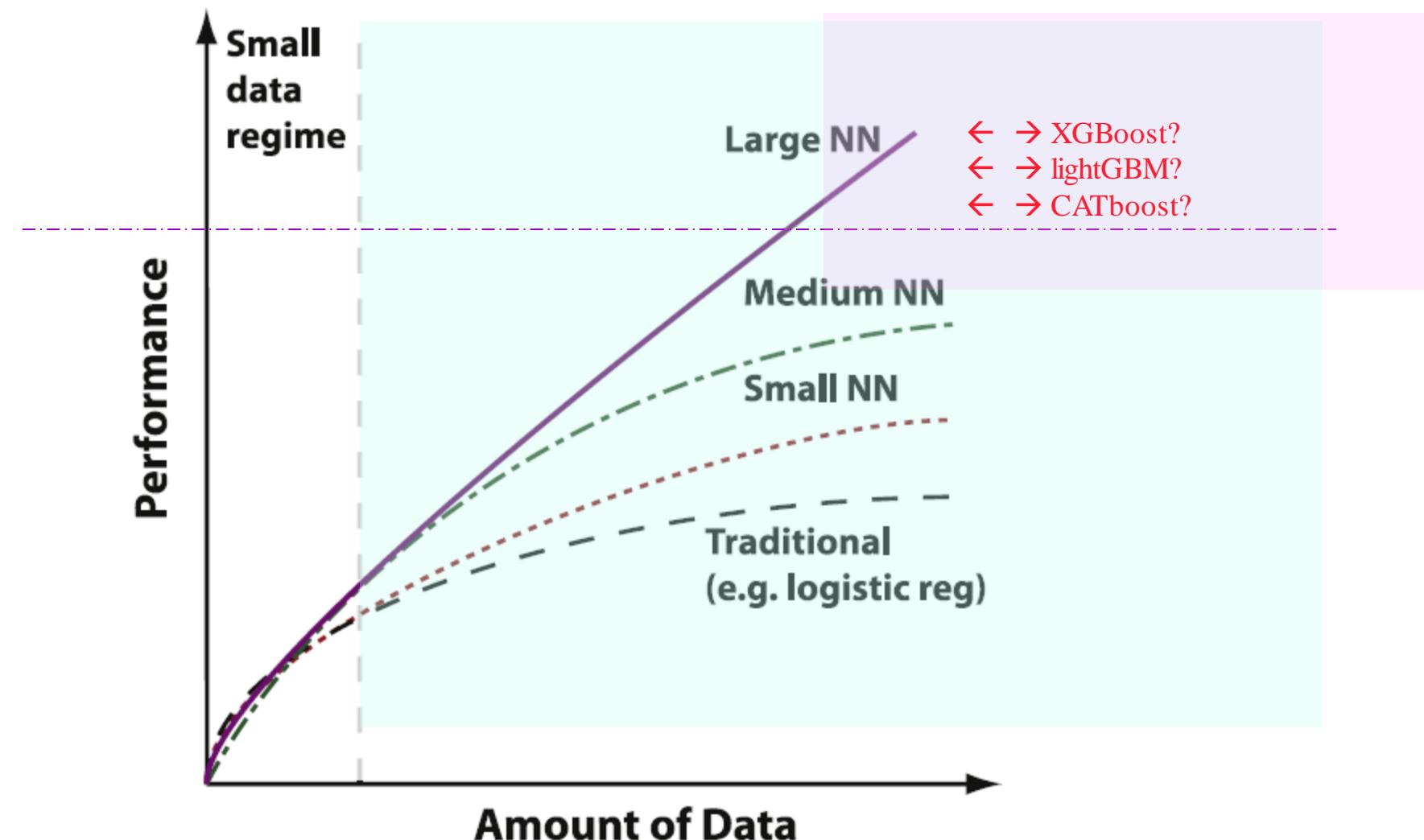
- \*Backpropagation (1986)
- \*Rectified linear unit activation function (2011)
- \*AlexNet wins ImageNet (2012) ; CNN
- \*GAN (2014)
- \*CNN beats human (2015)
- \*TensorFlow, Keras (2015)
- \*Gaussian type weights initialization (2015)
- \*AlphaGo (2016)
- \*AlphaGo Zero (2017)
- \*AlphaFold (2018)
- \*AlphaFold 2 (2020)
- \*GPU, TPU  $\leftrightarrow$  CPU, MPP, Big data



Overfitting and Computation time



# Deep Learning (2012~)

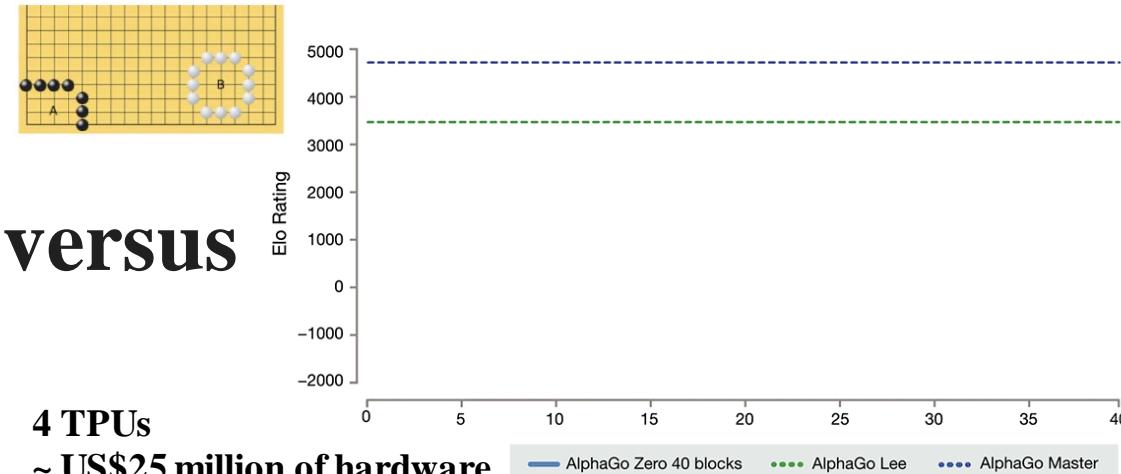




# Mastering the game of Go without human knowledge



versus



4 TPUs  
~ US\$25 million of hardware

*Tabula Rasa (blank state) → Discover new knowledge*

~~Learning to design materials from scratch~~  
Learning to design materials from first-principles

*Tabula Rasa* → Predict new crystal structures → (?) Discover new crystal structures

400 CPUs × 60 days ~ 65 years : CPU time

~ 60 days : Wall-clock time

**Self-play → superhuman level?**

Feeding back information on what worked to improve itself after each game

**3 days : complex tactics (human experts) → thousands of years of human knowledge**

**21 days**

**40 days : 30 million games → Discovering new knowledge**

**It simply asks the network to **predict** a winner.**

**4 TPUs ~ **US\$25 million** of hardware < 40 TPUs**

**days < months**

**Algorithms matter much more than either computing or data available.**



# What's next?

*As DeepMind co-founder, Demis Hassabis explains, Zero was not programmed to understand Go specifically, it could be reprogrammed to discover information in other fields: drug discovery, **protein folding**, quantum chemistry, particle physics, and **material design**.*

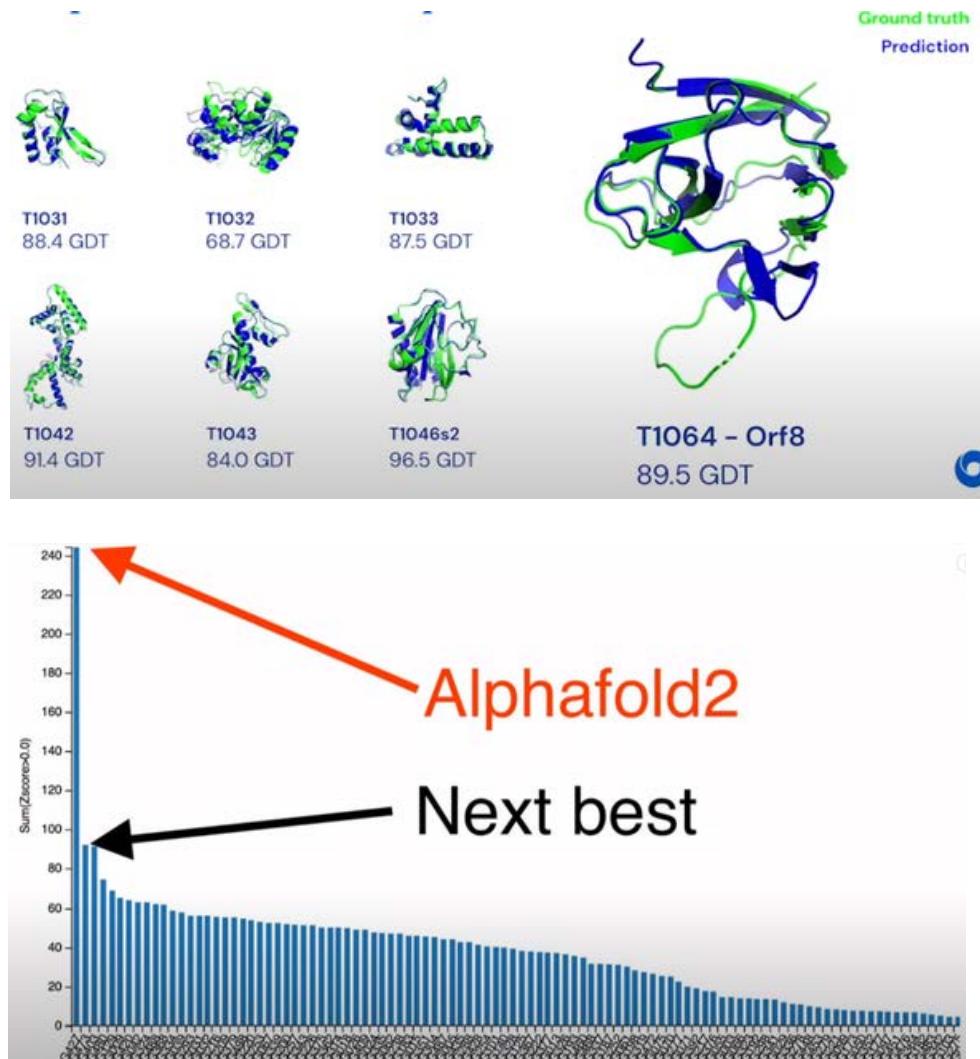


Demis Hassabis



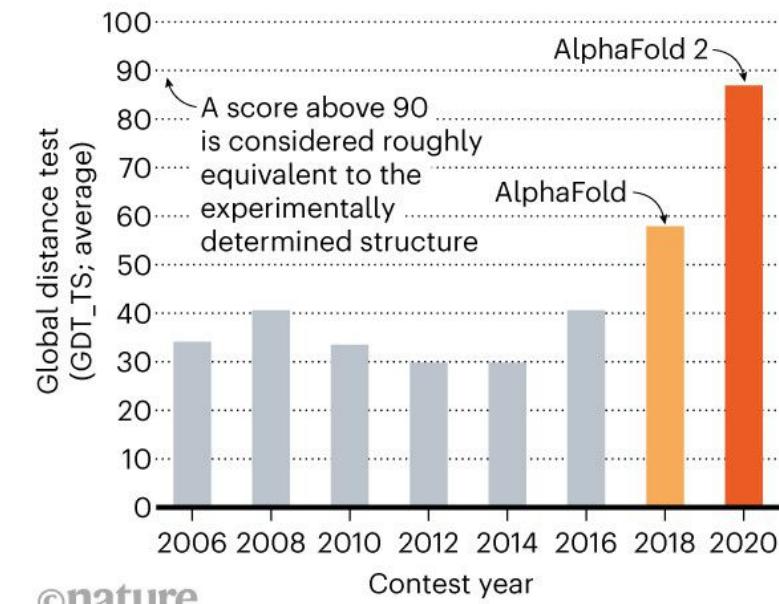


# Breakthrough!



## STRUCTURE SOLVER

DeepMind's AlphaFold 2 algorithm significantly outperformed other teams at the CASP14 protein-folding contest — and its previous version's performance at the last CASP.



허서비스 CEO는 딥마인드 블로그를 통해 “우리는 이것이 AI가 현재까지 과학 지식을 발전시키는 데 행한 가장 중요한 기여이자 AI가 인류에게 가져올 수 있는 이점의 훌륭한 예라고 믿는다”며 “이러한 통찰력은 생물학과 의학에 대한 우리의 이해에서 많은 흥미진진한 미래 발전을 뒷받침할 것”이라고 말했다.



# 2018 Turing Award

The 2018 Turing Award, known as the “Nobel Prize of computing,” has been given to a trio of researchers who laid the foundations for the current boom in artificial intelligence.

"Godfathers of AI"



From left to right: Yann LeCun | Photo: Facebook; Geoffrey Hinton | Photo: Google; Yoshua Bengio | Photo: Botler AI

**CNN,  
Backpropagation**

**Dropout,  
Backpropagation**

**ReLU,  
GAN**



# '인공지능계의 판타스틱 4'

구글 '구글 나우'

'딥페이스'

토론토 대학교, 제프리 힌턴 교수

페이스북, 뉴욕 대학교, 얀 르켕 박사



몬트리올 대학교, 요슈아 벤지오 교수

'왓슨' 삼성 word2vec

스탠포드 대학교, 앤드류 응(Andrew Ng) 교수

바이두



## Model representation (approach)

- Artificial neural network (connection)
- Genetic algorithm (evolution)
- Support vector machine (analogy)
- Bayes classifier (Bayesian)
- Decision tree (symbol)



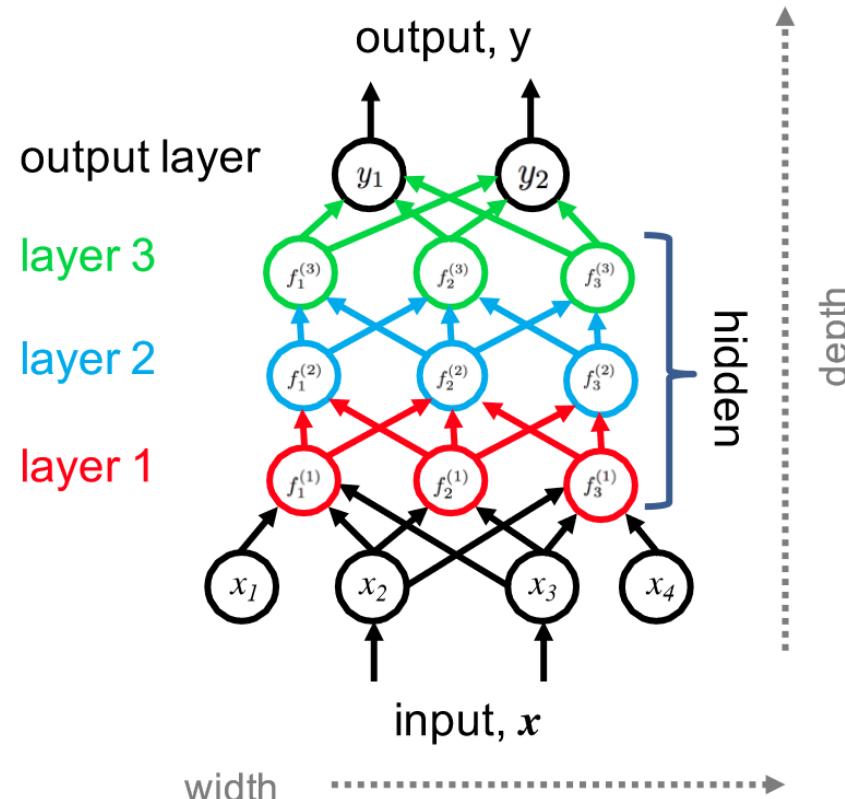
# Contemporary ML: a guide for practitioners in the physical sciences

$$y = f(x) = f^{(4)}(b^{(4)} + W^{(4)}f^{(3)}(b^{(3)} + W^{(3)}f^{(2)}(b^{(2)} + W^{(2)}f^{(1)}(b^{(1)} + W^{(1)}x)))$$

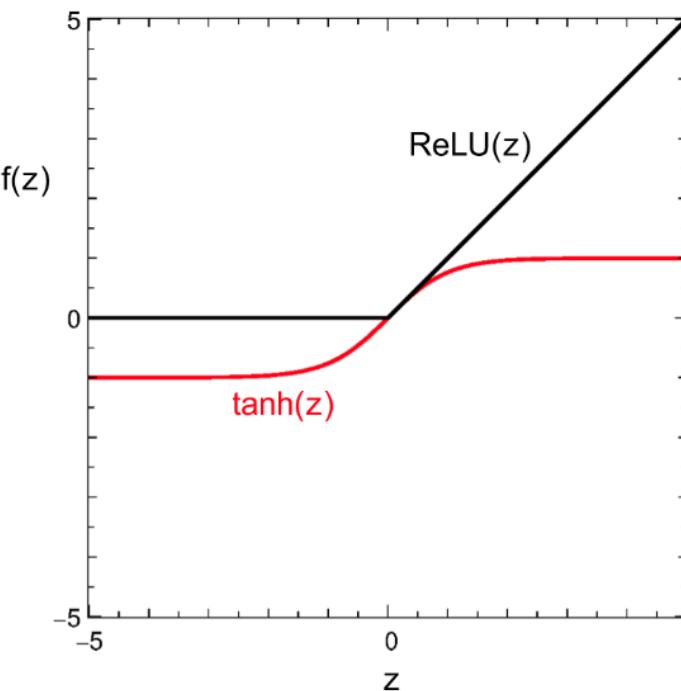
Rotation and shift

where the  $f(z) = \text{ReLU}(z) = \max\{0, z\}$

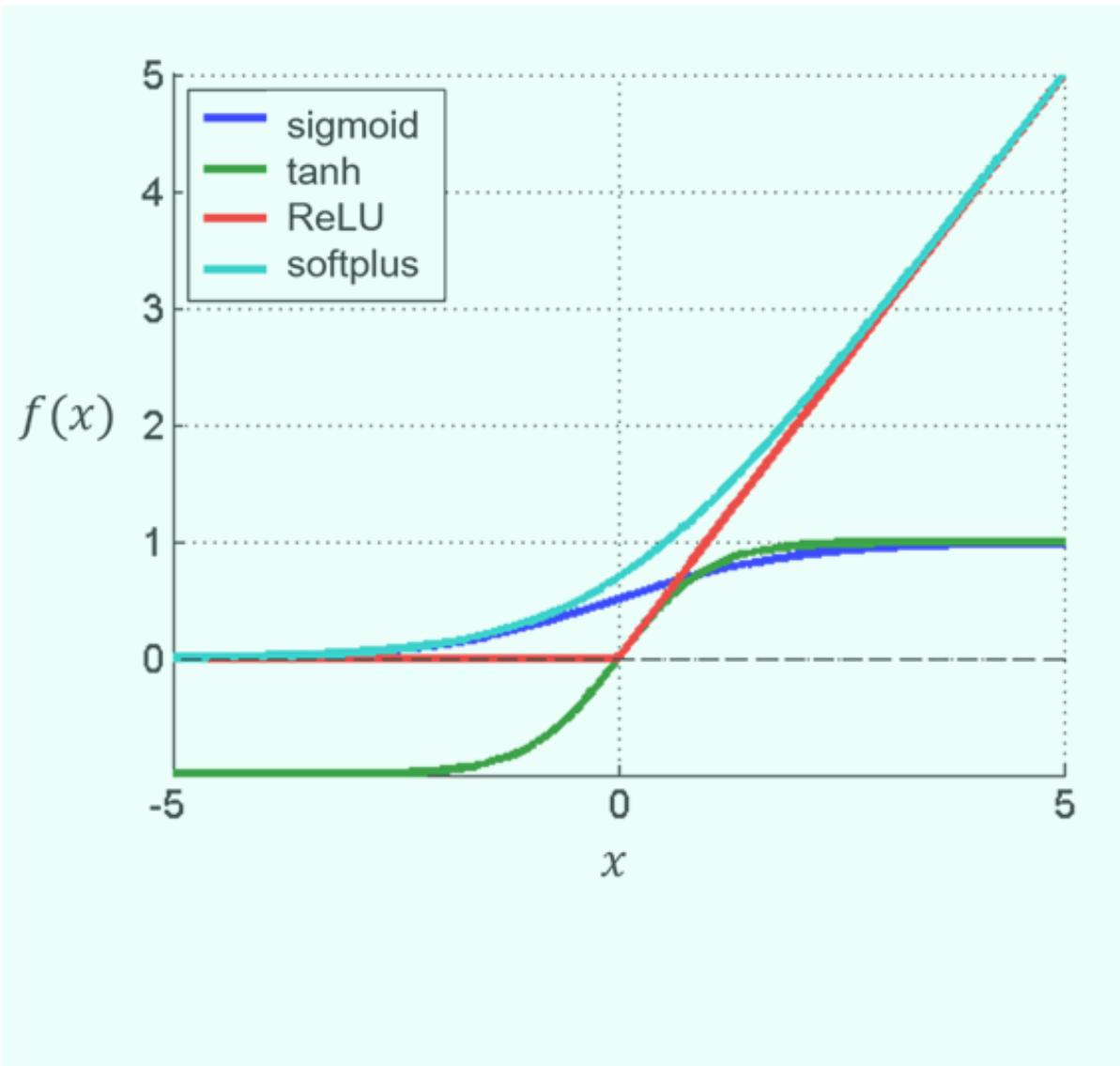
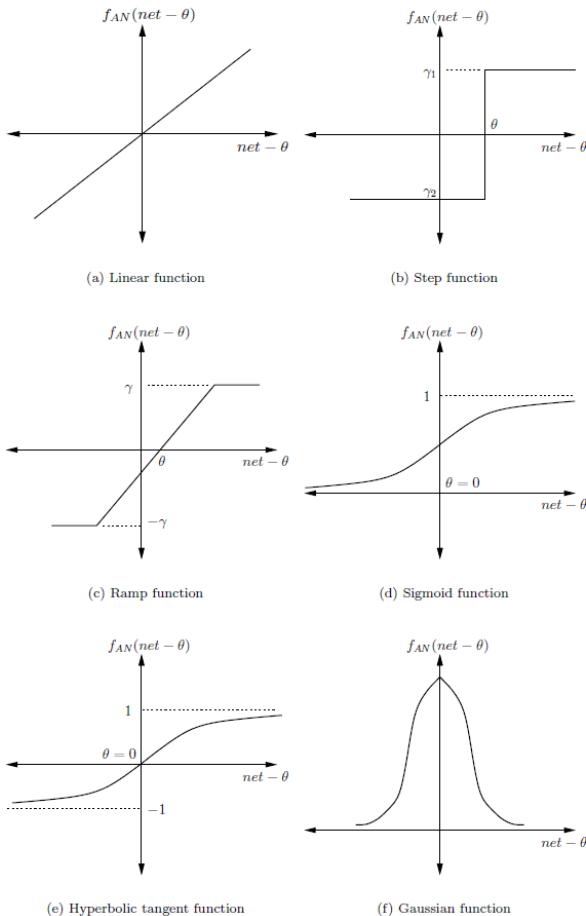
$$y = f(x) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x))) )$$



dot product vs distance



# Activation functions



cf. radial basis function, kernel function



# Vanishing gradient problem

```
from sklearn.datasets import make_circles
```

```
import matplotlib.pyplot as plt
```

```
# Make data: Two circles on x-y plane as a classification problem
```

```
X, y = make_circles(n_samples=1000, factor=0.5, noise=0.1)
```

```
plt.figure(figsize=(8,6))
```

```
plt.scatter(X[:,0], X[:,1], c=y)
```

```
plt.show()
```

```
from tensorflow.keras.layers import Dense, Input
```

```
from tensorflow.keras import Sequential
```

```
model = Sequential([ Input(shape=(2,)),Dense(5, "relu"),Dense(1, "sigmoid") ])
```

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
```

```
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
```

```
print(model.evaluate(X,y))
```

```
32/32 [=====] - 0s 2ms/step - loss: 0.3069 - acc: 0.9520[0.3068710267543793, 0.9520000219345093]
```

```
model = Sequential([ Input(shape=(2,)), Dense(5, "sigmoid"), Dense(5, "sigmoid"),Dense(5, "sigmoid"), Dense(1, "sigmoid")])
```

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
```

```
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
```

```
print(model.evaluate(X,y))
```

```
32/32 [=====] - 0s 4ms/step - loss: 0.6930 - acc: 0.4980[0.6930215954780579, 0.49799999594688416]
```

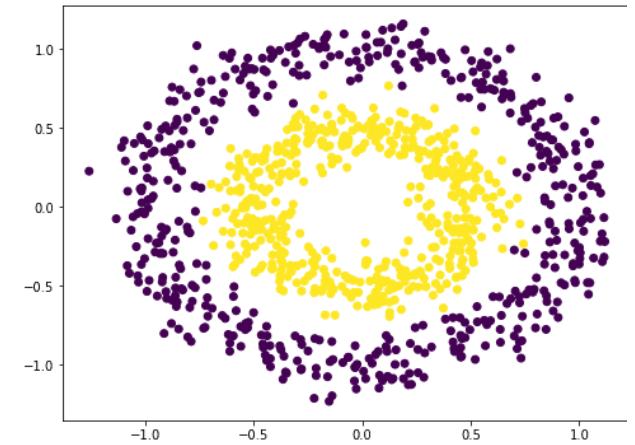
```
model = Sequential([ Input(shape=(2,)), Dense(5, "relu"), Dense(5, "relu"), Dense(5, "relu"), Dense(1, "sigmoid")])
```

```
model.compile(optimizer="adam", loss="binary_crossentropy", metrics=["acc"])
```

```
model.fit(X, y, batch_size=32, epochs=100, verbose=0)
```

```
print(model.evaluate(X,y))
```

```
32/32 [=====] - 0s 4ms/step - loss: 0.0182 - acc: 0.9970[0.01822291873395443, 0.996999979019165]
```





# Backpropagation (1974, 1986)

mean-squared-error (MSE) or quadratic cost

$$\mathcal{C}_{\text{quad}} = \frac{1}{n} \sum_{\mathbf{x}} |y_{\mathbf{x}} - y_{\mathbf{x}}^c|^2,$$

categorical cross entropy

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = - \sum_i^C t_i \log(f(s)_i)$$

confusion matrix  $C$  is a  $N$  by  $N$  matrix

$$C_{ij} = \frac{\text{Number of class } i \text{ samples classified as class } j}{\text{Number of class } i \text{ samples}}.$$

```
def CrossEntropy(yHat, y):
    if y == 1:
        return -log(yHat)
    else:
        return -log(1 - yHat)
```

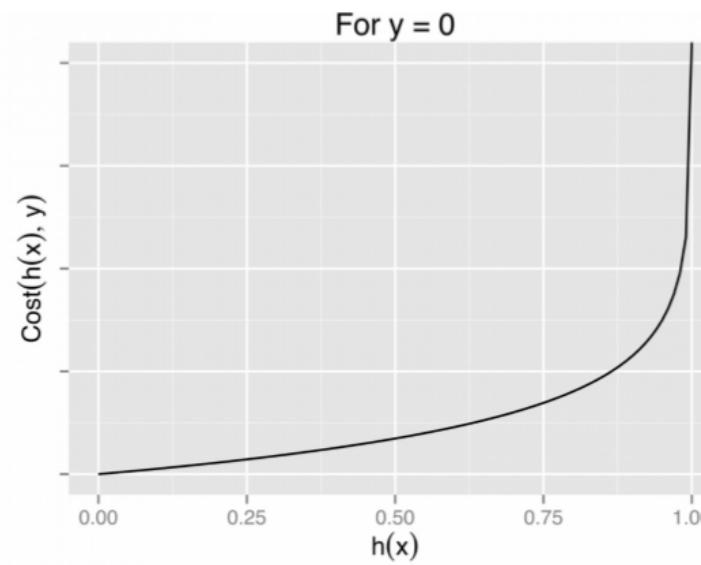
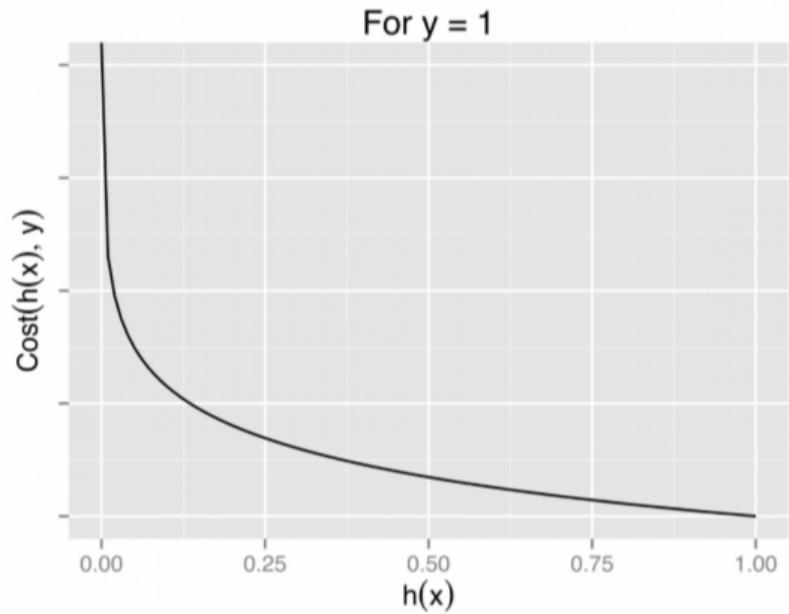
```
def KLDivergence(yHat, y):
    """
    :param yHat:
    :param y:
    :return: KLDiv(yHat // y)
    """
    return np.sum(yHat * np.log((yHat / y)))
```

output prediction  
 $\hat{y} = [0.2 \quad 0.7 \quad \hat{y}_i \quad 0.1]$

target  
 $y = [0 \quad 1 \quad y_i \quad 0]$

output size  
 $\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



$$J(\theta) = -\frac{1}{m} \sum \left[ y^{(i)} \log(h\theta(x(i))) + (1 - y^{(i)}) \log(1 - h\theta(x(i))) \right]$$



log-likelihood as a cost function  $J$  that can be minimized using gradient descent

$$J(\mathbf{w}) = \sum_i \frac{1}{2} (\phi(z^{(i)}) - y^{(i)})^2$$

$$L(\mathbf{w}) = P(\mathbf{y} | \mathbf{x}; \mathbf{w}) = \prod_{i=1}^n P(y^{(i)} | x^{(i)}; \mathbf{w}) = \prod_{i=1}^n (\phi(z^{(i)}))^{y^{(i)}} (1 - \phi(z^{(i)}))^{1-y^{(i)}}$$

$$l(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^n \left[ y^{(i)} \log(\phi(z^{(i)})) + (1 - y^{(i)}) \log(1 - \phi(z^{(i)})) \right]$$
regularization strength

$$J(\mathbf{w}) = \sum_{i=1}^n \left[ -y^{(i)} \log(\phi(z^{(i)})) - (1 - y^{(i)}) \log(1 - \phi(z^{(i)})) \right] + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

Target feature	Encoding
Encoding	
Categorical	
Tiger	→
Cat	→
Airplane	→
Cat	→

	Tiger	Cat	Airplane
→	↓	↓	↓
Tiger	1	0	0
Cat	0	1	0
Airplane	0	0	1
Cat	0	1	0

$$J(\phi(z), y; \mathbf{w}) = -y \log(\phi(z)) - (1 - y) \log(1 - \phi(z))$$

$$J(\phi(z), y; \mathbf{w}) = \begin{cases} -\log(\phi(z)) & \text{if } y = 1 \\ -\log(1 - \phi(z)) & \text{if } y = 0 \end{cases}$$

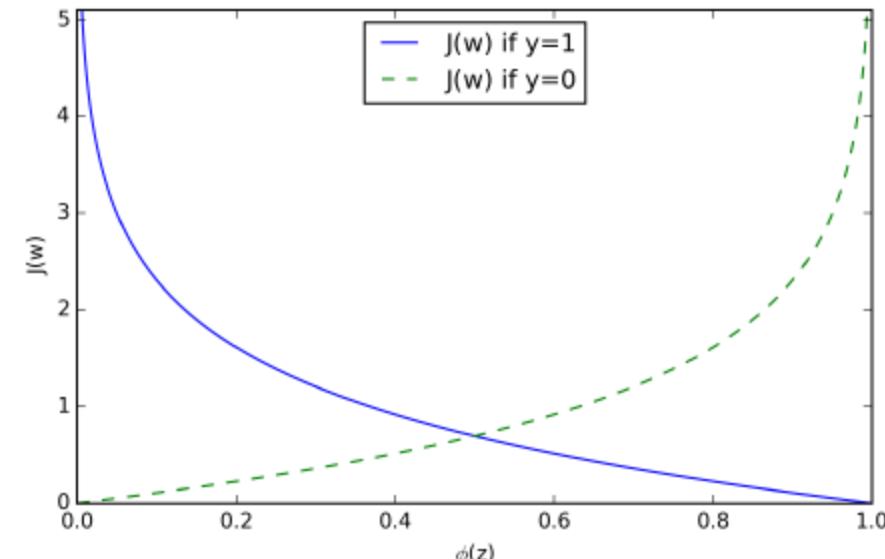
$$Loss = (Y)(-\log(Y_{pred})) + (1 - Y)(-\log(1 - Y_{pred}))$$

Remains when  $Y = 1$

Remains when  $Y = 0$

Removed when  $Y = 0$

Removed when  $Y = 1$



$$S = \begin{cases} - \int p(x) \cdot \log p(x) \cdot dx, & \text{if } x \text{ is continuous} \\ - \sum_x p(x) \cdot \log p(x), & \text{if } x \text{ is discrete} \end{cases}$$

$$L(X_i, Y_i) = - \sum_{j=1}^c y_{ij} * \log(p_{ij})$$

where  $Y_i$  is one-hot encoded target vector  $(y_{i1}, y_{i2}, \dots, y_{ic})$ ,

$$y_{ij} = \begin{cases} 1, & \text{if } i_{th} \text{ element is in class } j \\ 0, & \text{otherwise} \end{cases}$$

$p_{ij} = f(X_i)$  = Probability that  $i_{th}$  element is in class  $j$

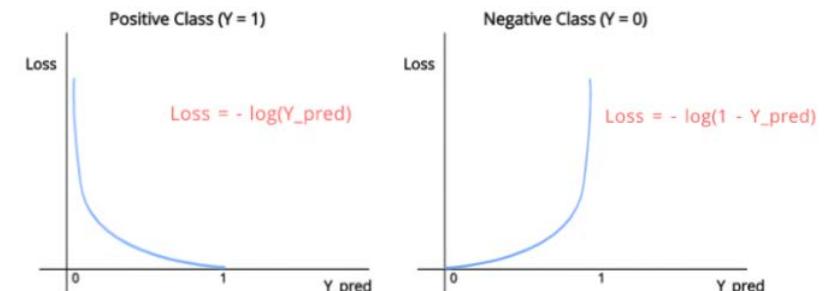
For probability distributions  $P$  and  $Q$ ,

KL-Divergence of  $P$  from  $Q$  is given by

$$D_{KL}(P||Q)$$

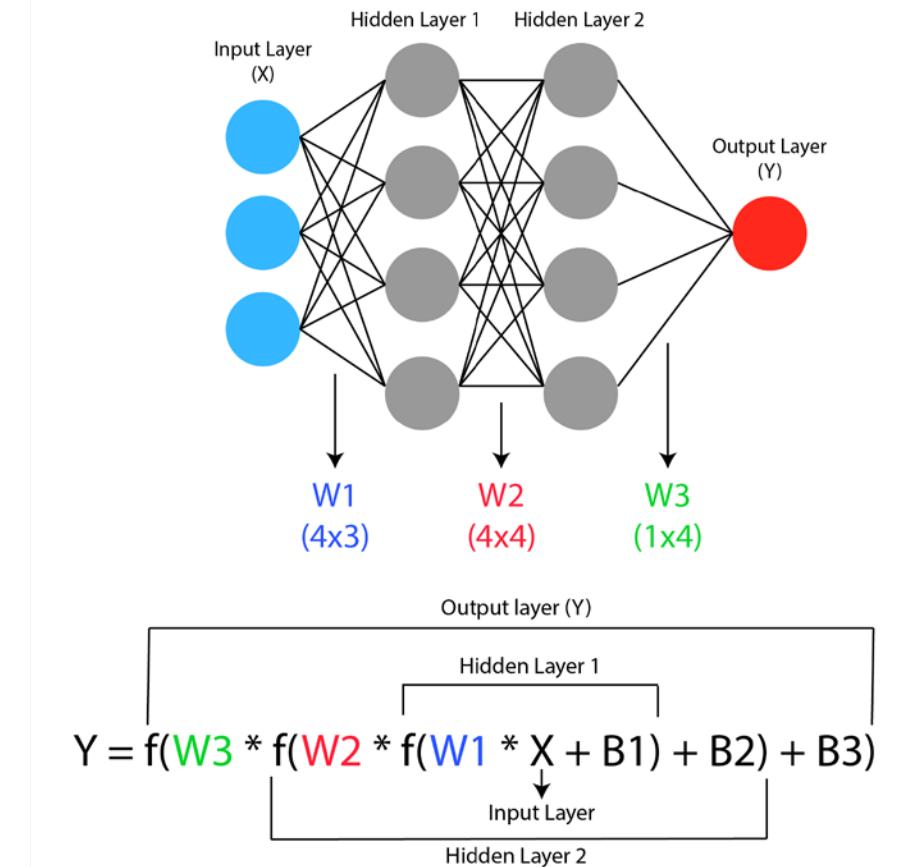
$$= \left\{ \begin{array}{l} - \sum_x P(x) \cdot \log \frac{Q(x)}{P(x)} = \sum_x P(x) \cdot \log \frac{P(x)}{Q(x)}, \quad \text{for discrete distributions} \\ - \int P(x) \cdot \log \frac{Q(x)}{P(x)} \cdot dx = \int P(x) \cdot \log \frac{P(x)}{Q(x)} \cdot dx, \quad \text{for continuous distributions} \end{array} \right.$$

= Expectation of logarithmic difference between  $P$  and  $Q$  with respect to  $P$





$$\begin{aligned}
 o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\
 &= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_{j,p}}) \right) \\
 &= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j} \left( \sum_{i=1}^{I+1} v_{ji} z_{i,p} \right) \right) \\
 o_{k,p} &= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j} \left( \sum_{l=1}^L v_{jl} h_l(\mathbf{z}_p) \right) \right)
 \end{aligned}$$



$$\begin{aligned}
o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\
&= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_{j,p}}) \right) \\
&= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j} \left( \sum_{i=1}^{I+1} v_{ji} z_{i,p} \right) \right)
\end{aligned}$$

$$\frac{\partial net_{o_k}}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \left( \sum_{j=1}^{J+1} w_{kj} y_j \right) = y_j$$

$$\mathcal{E}_p = \frac{1}{2} \left( \frac{\sum_{k=1}^K (t_{k,p} - o_{k,p})^2}{K} \right)$$

$$o_k = f_{o_k}(net_{o_k}) = \frac{1}{1 + e^{-net_{o_k}}}$$

$$\frac{\partial o_k}{\partial net_{o_k}} = \frac{\partial f_{o_k}}{\partial net_{o_k}} = (1 - o_k) o_k = f'_{o_k}$$

$$y_j = f_{y_j}(net_{y_j}) = \frac{1}{1 + e^{-net_{y_j}}}$$

$$\begin{aligned}
\frac{\partial o_k}{\partial w_{kj}} &= \frac{\partial o_k}{\partial net_{o_k}} \frac{\partial net_{o_k}}{\partial w_{kj}} \\
&= (1 - o_k) o_k y_j \\
&= f'_{o_k} y_j
\end{aligned}$$

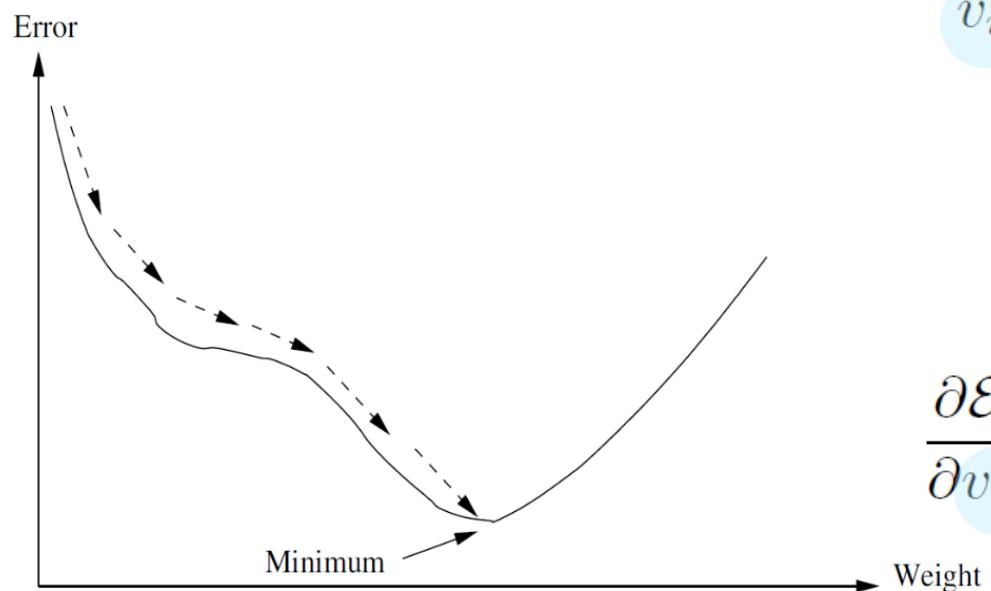
$$w_{kj}(t) + = \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1)$$

$$v_{ji}(t) + = \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1)$$

# Training a single neuron

$$\mathcal{E} = \sum_{p=1}^{P_T} (t_p - o_p)^2$$

---

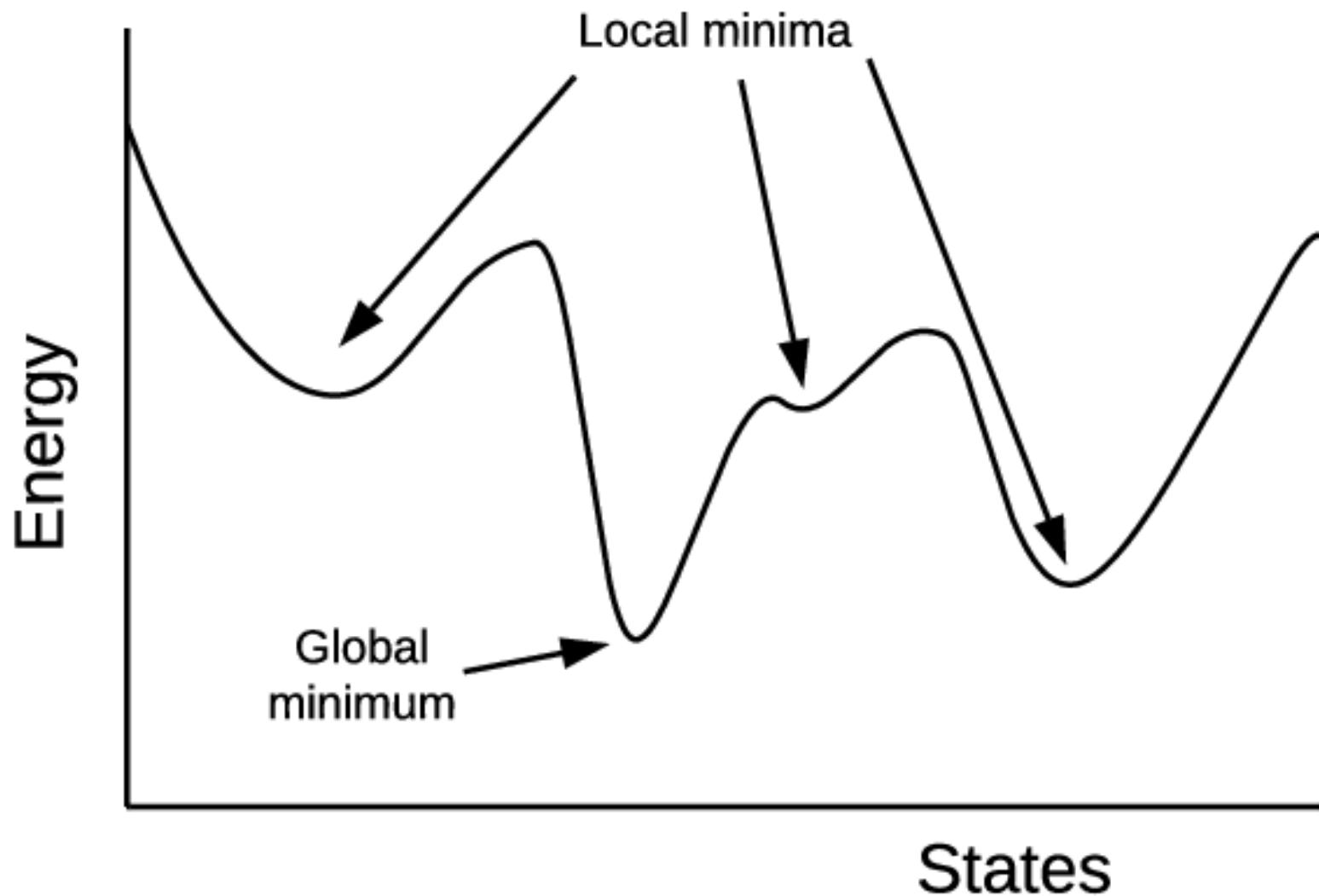


$$v_i(t) = v_i(t-1) + \Delta v_i(t)$$

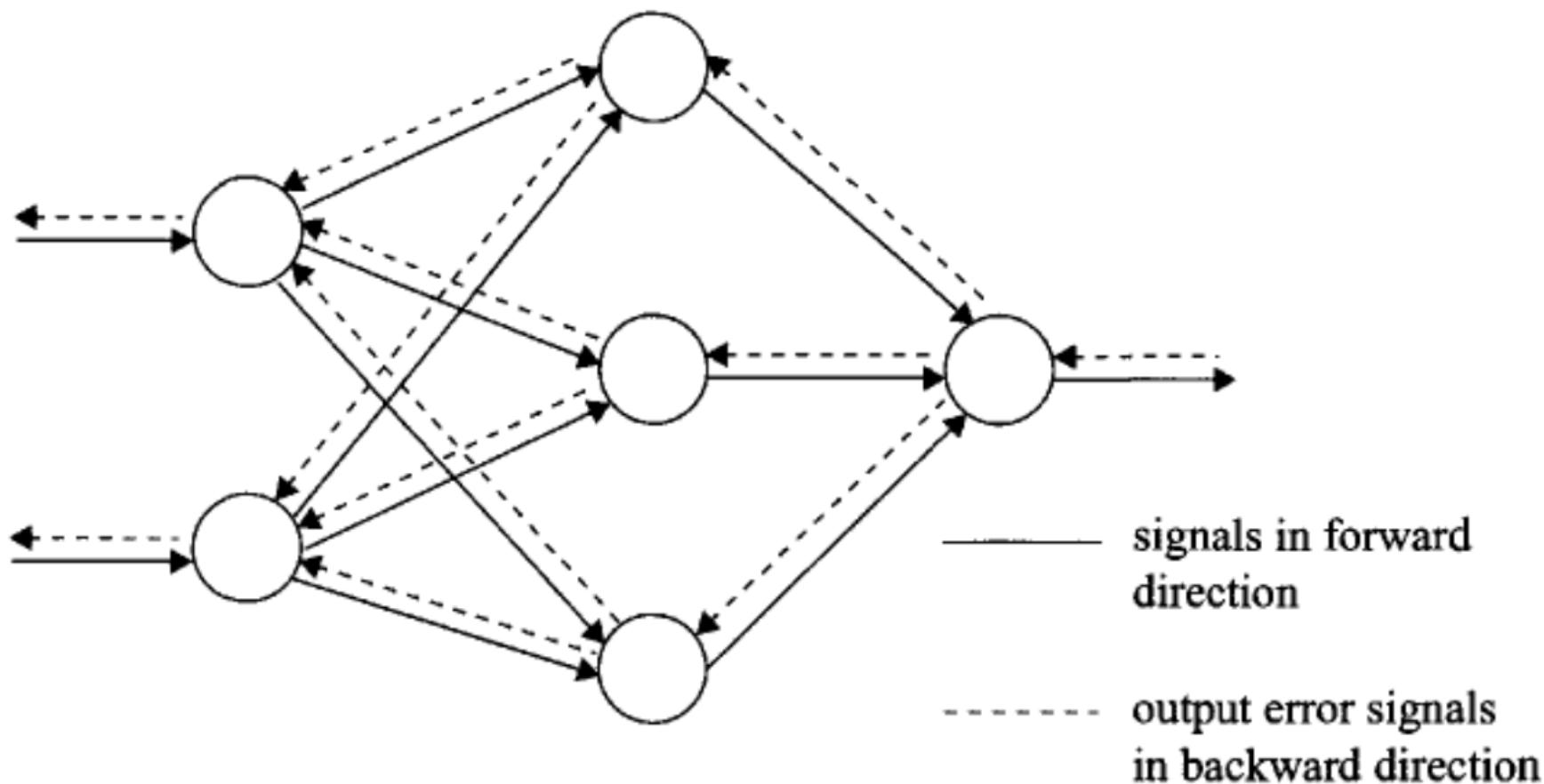
$$\Delta v_i(t) = \eta \left( -\frac{\partial \mathcal{E}}{\partial v_i} \right)$$

$$\frac{\partial \mathcal{E}}{\partial v_i} = -2(t_p - o_p) \frac{\partial f}{\partial \text{net}_p} z_{i,p}$$

$$\text{net} = \sum_{i=1}^I z_i v_i$$

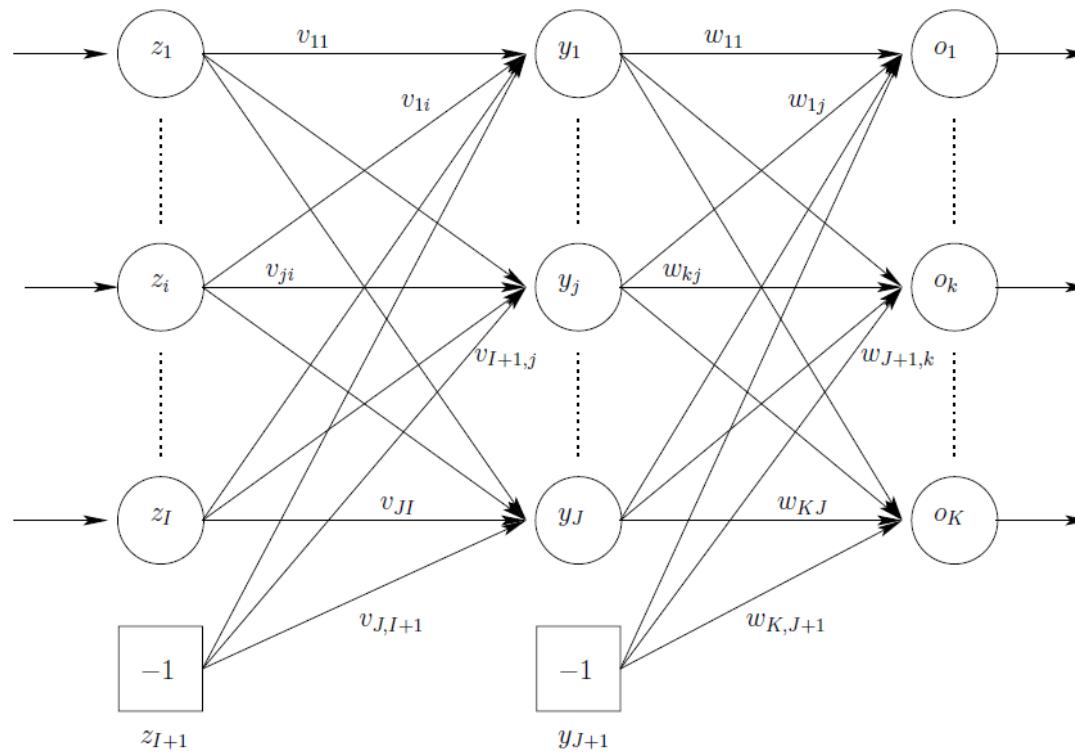


# Backward direction



# Feedforward Neural Networks

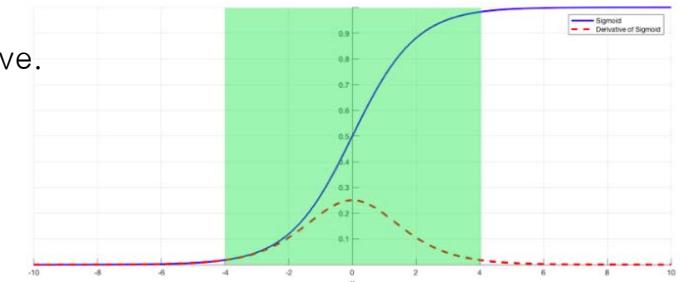
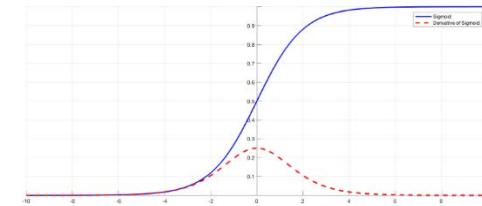
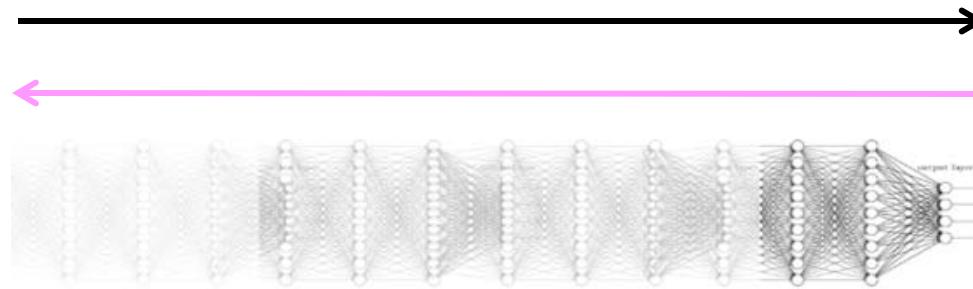
$$\begin{aligned}
 o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\
 &= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_{j,p}}) \right) \\
 \text{Input, hidden, and output layers} &= f_{o_k} \left( \sum_{j=1}^{J+1} w_{kj} f_{y_j} \left( \sum_{i=1}^{I+1} v_{ji} z_{i,p} \right) \right)
 \end{aligned}$$





# Backpropagation (1974, 1986)

## Vanishing gradient



The simplest solution is to use other activation functions, such as **ReLU**, which doesn't cause a small derivative.

**Residual networks** are another solution, as they provide residual connections straight to earlier layers.

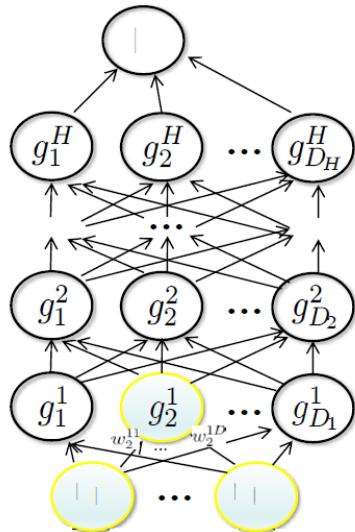
Finally, **batch normalization** layers can also resolve the issue.

Cf. Weight initialization

Error function, loss function, objective function

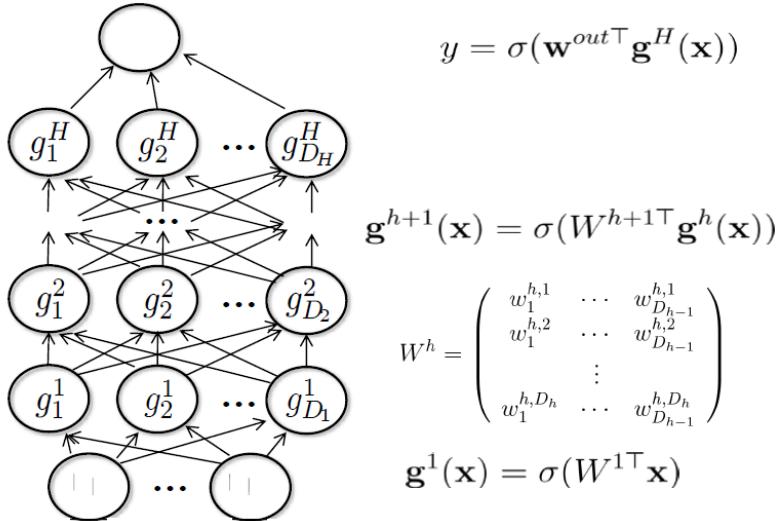
$$E = \frac{1}{2} \sum_{j=1}^m [d_j(k) - y_j(k)]^2$$

Given a desired output response vector  $\{d_j\}$



$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

$$g_2^1 = \sigma \left( \sum_{i=1}^D w_2^{1i} x_i \right)$$



$$y = \sigma(\mathbf{w}^{out\top} \mathbf{g}^H(\mathbf{x}))$$

...

$$\frac{dL}{dW^h} = \sum_{i=1}^N (y_i - \hat{y}_i) \left( -\frac{\partial \hat{y}_i}{\partial W^h} \right)$$

$$\frac{\partial \hat{y}}{\partial W^h} = \frac{\partial \hat{y}}{\partial \mathbf{g}^H} \frac{\partial \mathbf{g}^H}{\partial \mathbf{g}^{H-1}} \dots \frac{\partial \mathbf{g}^h}{\partial W^h}$$

$$\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1 - \sigma(a))$$

$$\frac{\partial \hat{y}}{\partial \mathbf{g}^H} = \frac{\partial \sigma(\mathbf{w}^{out\top} \mathbf{g}^H)}{\partial \mathbf{g}^H} = \sigma(\mathbf{w}^{out\top} \mathbf{g}^H)(1 - \sigma(\mathbf{w}^{out\top} \mathbf{g}^H)) \mathbf{w}^{out}$$

$$\frac{\partial \mathbf{g}^{h+1}}{\partial \mathbf{g}^h} = \frac{\partial \sigma([W^{h+1\top} \mathbf{g}^h]_i)}{\partial \mathbf{g}^h} = \sigma([W^{h+1\top} \mathbf{g}^h]_i)(1 - \sigma([W^{h+1\top} \mathbf{g}^h]_i)) W^{h+1}_{:i}$$

$$\frac{\partial \mathbf{g}^h}{\partial W^h} = \frac{\partial \sigma([W^{h\top} \mathbf{g}^{h-1}]_i)}{\partial W^h} = \sigma([W^{h\top} \mathbf{g}^{h-1}]_i)(1 - \sigma([W^{h\top} \mathbf{g}^{h-1}]_i)) \mathbf{g}^{h-1}$$

$$\frac{\partial \mathbf{g}^h}{\partial W^h} = 0, \quad i \neq j$$

$$\begin{aligned} W^h &\leftarrow W^h - \gamma \frac{dL}{dW^h} \\ &= W^h - \gamma \sum_{i=1}^N (y_i - \hat{y}_i) \left( -\frac{\partial \hat{y}_i}{\partial W^h} \right) \\ &= W^h + \gamma \sum_{i=1}^N (y_i - \hat{y}_i) \underbrace{\frac{\partial \hat{y}_i}{\partial \mathbf{g}^H} \frac{\partial \mathbf{g}^H}{\partial \mathbf{g}^{H-1}} \dots \frac{\partial \mathbf{g}^h}{\partial W^h}}_{\delta_i^h} \end{aligned}$$

$$\delta_i^h \equiv (y_i - \hat{y}_i) \frac{\partial \hat{y}_i}{\partial \mathbf{g}^H} \frac{\partial \mathbf{g}^H}{\partial \mathbf{g}^{H-1}} \dots \frac{\partial \mathbf{g}^{h+1}}{\partial \mathbf{g}^h}$$

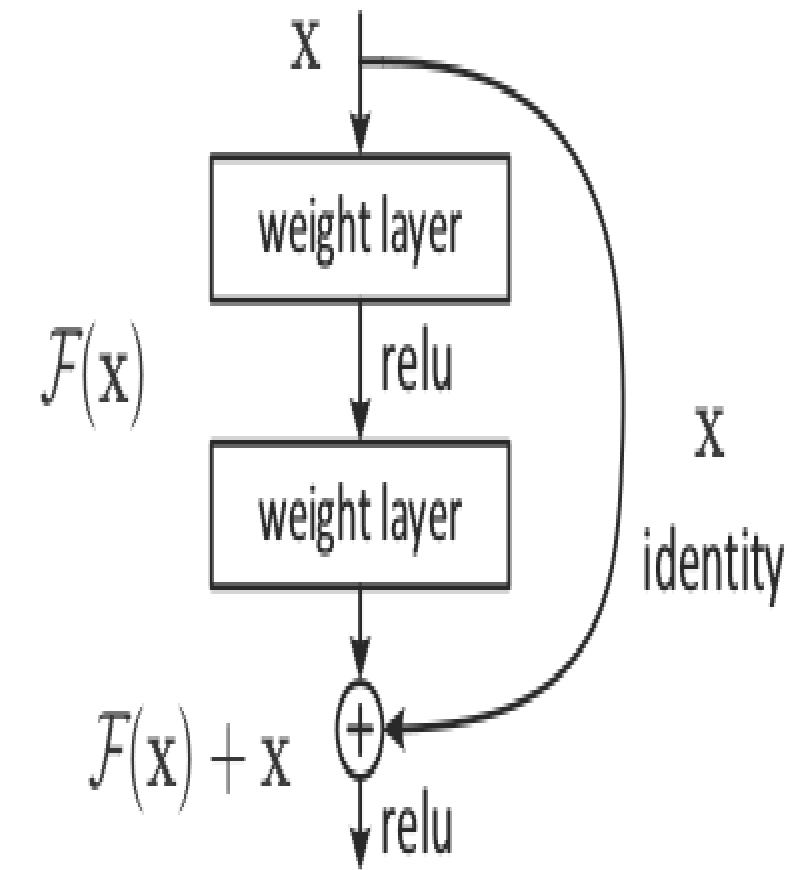
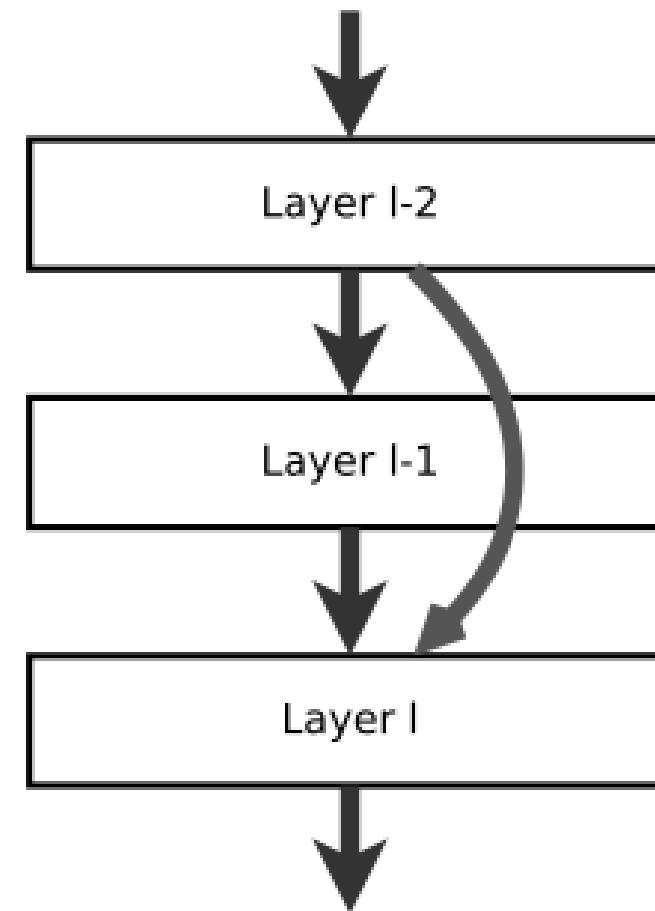
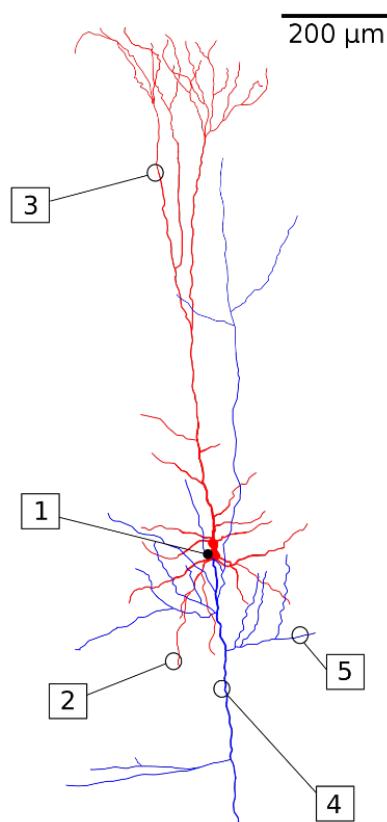
$$\delta_i^h = \delta_i^{h+1} \frac{\partial \mathbf{g}^{h+1}}{\partial \mathbf{g}^h}$$

Update of weights of the level  $h$  using backpropagated error

$$W^h \leftarrow W^h + \gamma \sum_{i=1}^N \delta_i^h \frac{\partial \mathbf{g}^h}{\partial W^h}$$



# Residual neural network





# Backpropagation (1974, 1986)

- *Vanishing gradient problem*
- ReLU(REcified Linear Unit)
- Weights initilization

‘uniform’, ‘normal’

*relu—relu—relu---relu--.....,--softmax*

*relu—relu—relu---relu--.....,--sigmoid*

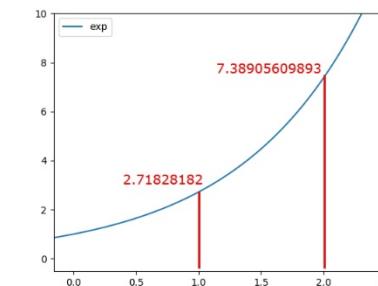
Error function, loss function, objective function

$$E = \frac{1}{2} \sum_{j=1}^m [d_j(k) - y_j(k)]^2$$

Given a desired output response vector  $\{d_j\}$

‘relu’, ‘linear’, ‘sigmoid’, ‘softmax’

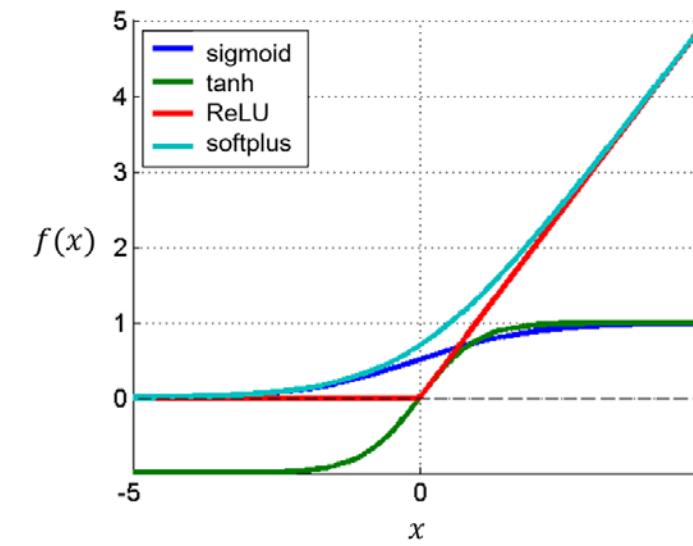
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$



5
0.3
2.1

Softmax  
→

0.94
0.01
0.05

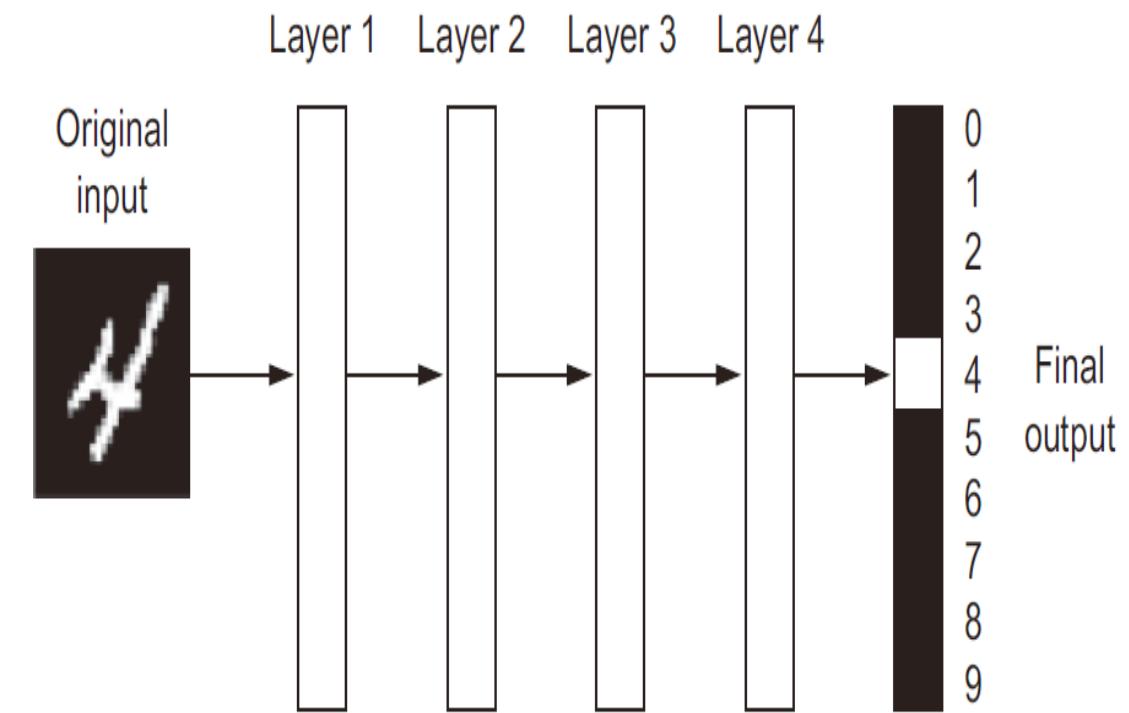
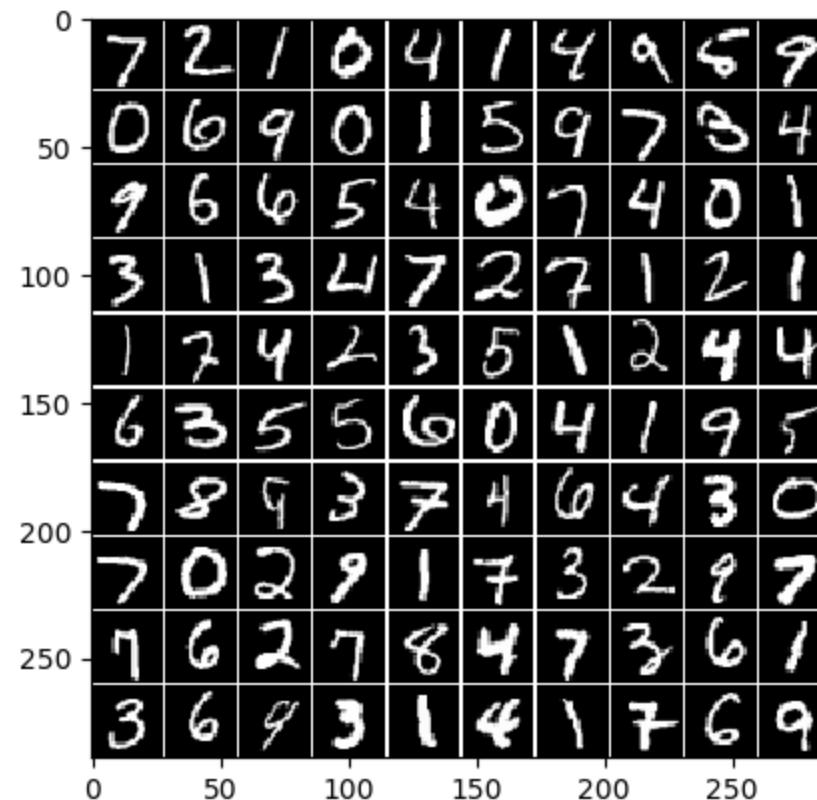




Let's examine how a network several layers deep transforms an image of a digit in order to recognize what digit it is.

MNIST (Modified National Institute of Standards and Technology database)

Data samples



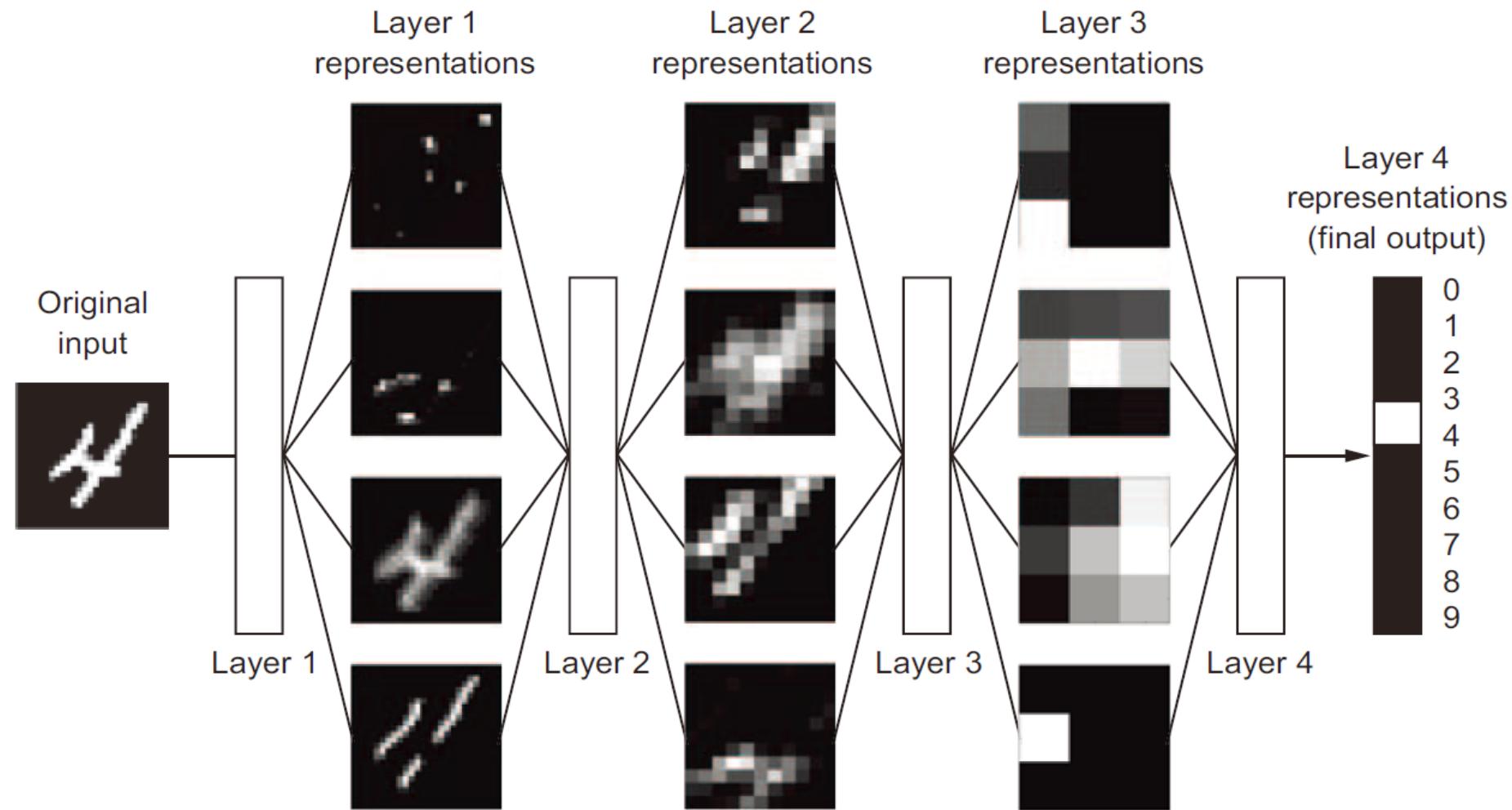


This is a table of some of the ML methods used on the database and their error rates, by type of classifier:

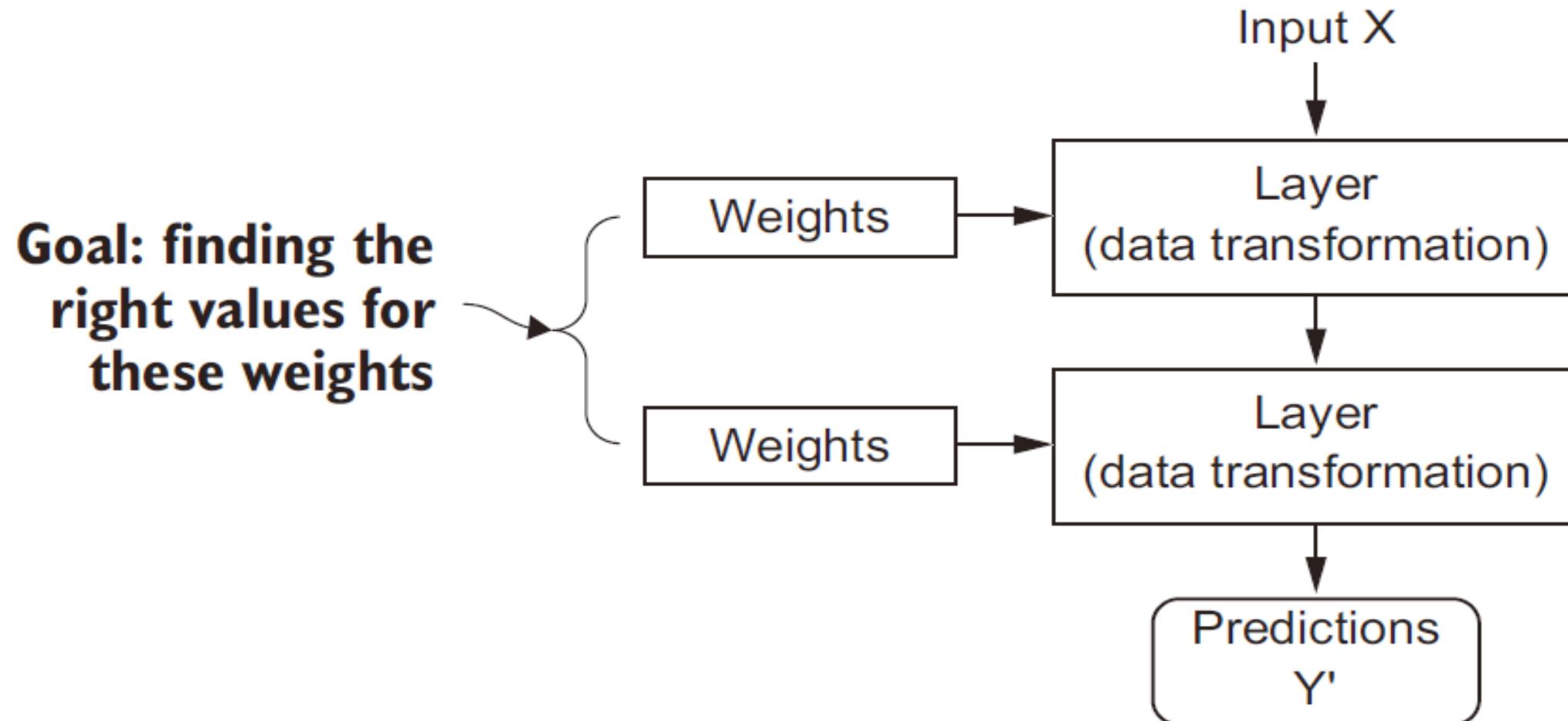
Type	Classifier	Distortion	Preprocessing	Error rate (%)
Linear classifier	Pairwise linear classifier	None	Deskewing	7.6 <sup>[9]</sup>
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	None	Shiftable edges	0.52 <sup>[19]</sup>
Boosted Stumps	Product of stumps on Haar features	None	Haar features	0.87 <sup>[20]</sup>
Non-linear classifier	40 PCA + quadratic classifier	None	None	3.3 <sup>[9]</sup>
Support-vector machine (SVM)	Virtual SVM, deg-9 poly, 2-pixel jittered	None	Deskewing	0.56 <sup>[21]</sup>
Deep neural network (DNN)	2-layer 784-800-10	None	None	1.6 <sup>[22]</sup>
Deep neural network	2-layer 784-800-10	Elastic distortions	None	0.7 <sup>[22]</sup>
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	Elastic distortions	None	0.35 <sup>[23]</sup>
Convolutional neural network (CNN)	6-layer 784-40-80-500-1000-2000-10	None	Expansion of the training data	0.31 <sup>[15]</sup>
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.27 <sup>[24]</sup>
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	Elastic distortions	Width normalizations	0.23 <sup>[8]</sup>
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	None	Expansion of the training data	0.21 <sup>[17][18]</sup>



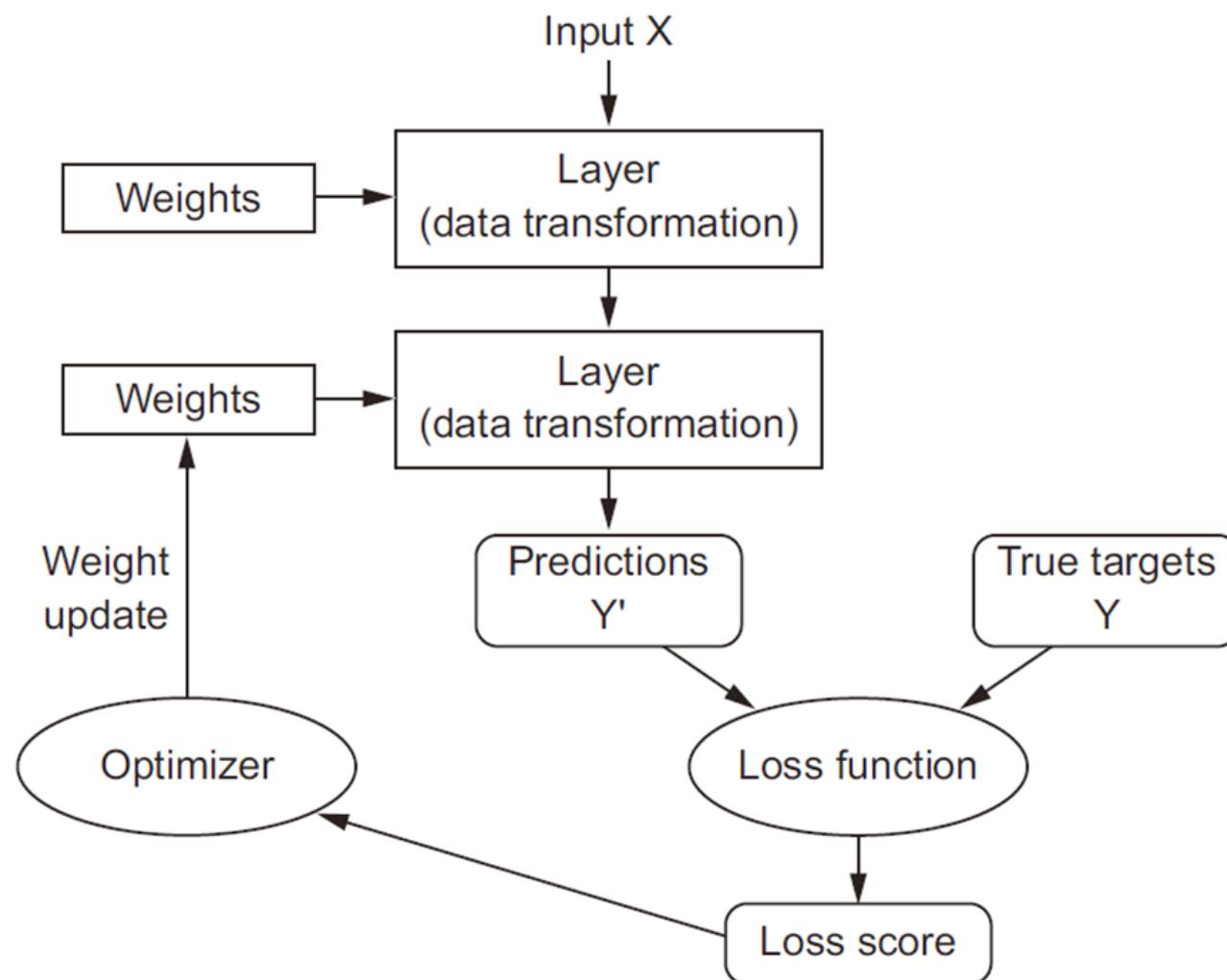
# Deep representations learned by a digit-classification model



A neural network is parameterized by its weights.

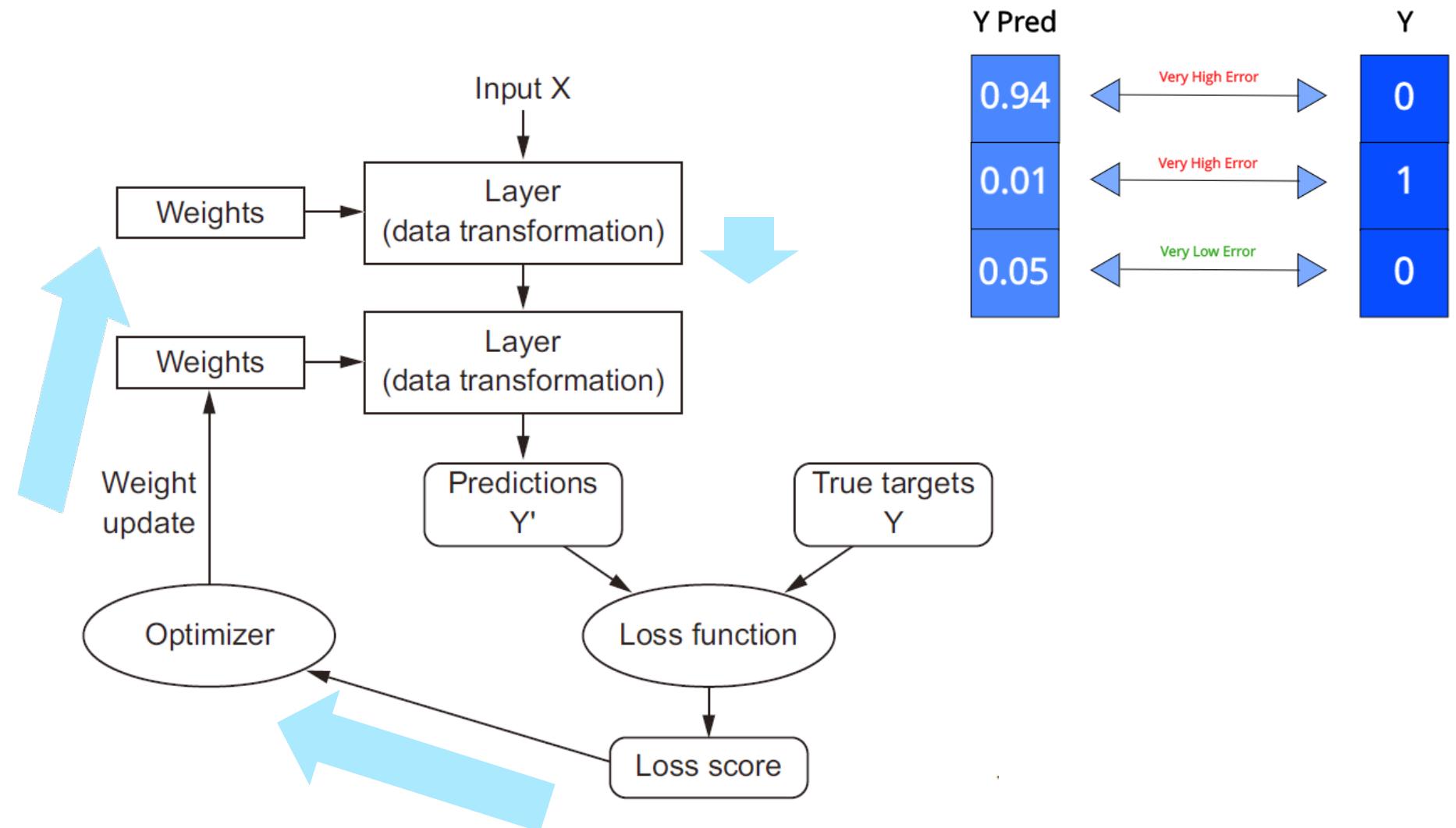


A loss function measures the quality of the network's output.





The loss score is used as a feedback signal to adjust the weights.



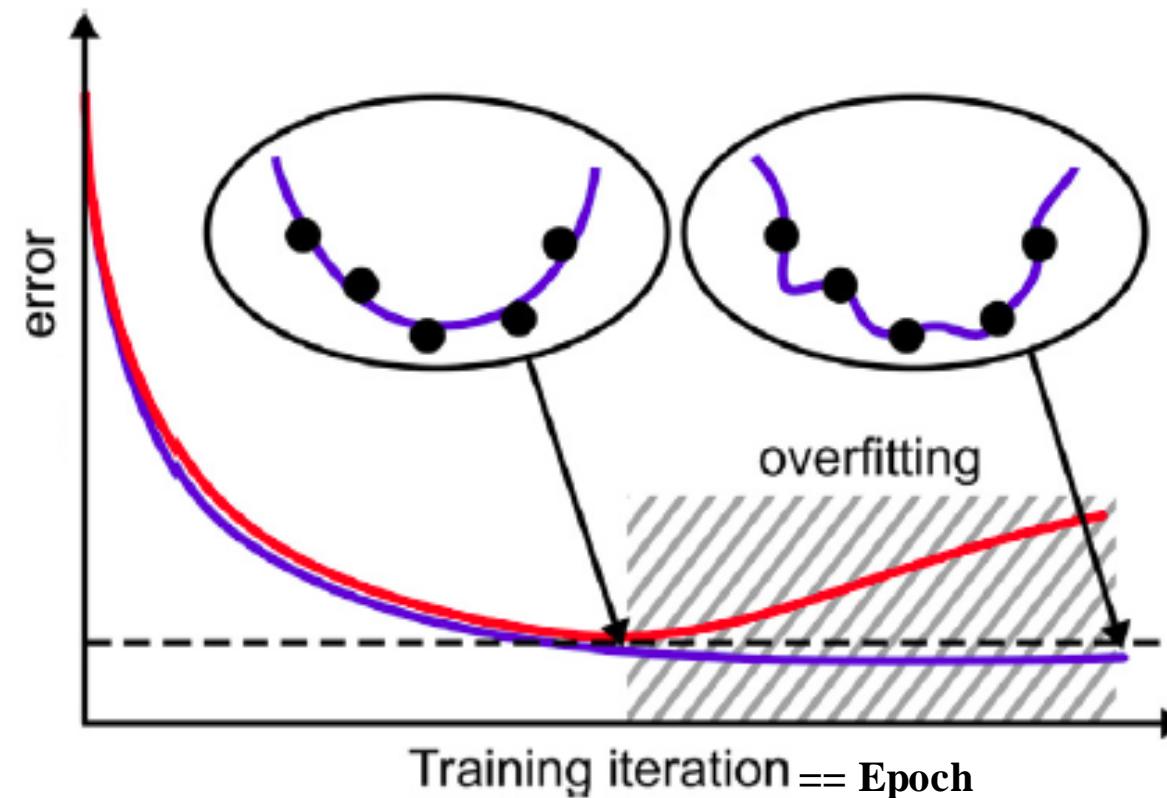


# Artificial neural networks

- Dataset: {input vector, output vector}
- Training set, test set, and **overfitting**

↔ DL

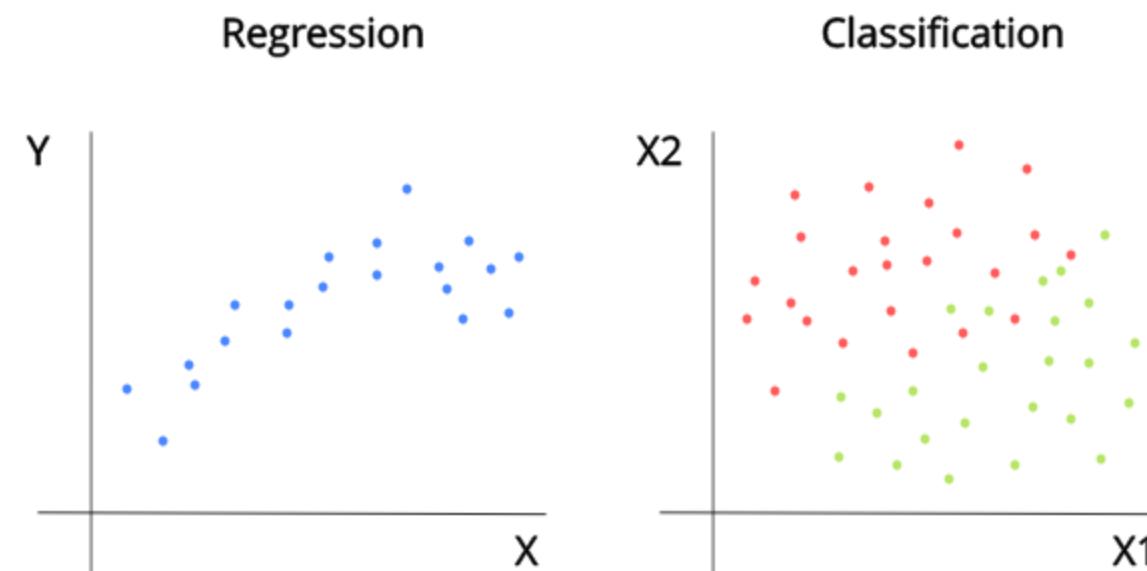
Dropout: 2012





# Regression analysis

- $Y \sim f(X, \{\beta\})$ ,  $E(Y|X) = f(X, \{\beta\})$
- $\{\beta\}$  : unknown parameters
- In statistical modeling, regression analysis is a set of statistical processes for **estimating** the relationships among variables.





# Terminologies

- In ML, a category in a classification problem is called a *class*. Data points are called *samples*. The class associated with a specific sample is called a *label*.
- Attributes (the variables being used to make predictions) are also known as the following:
  - \* Predictors
  - \* *Features*
  - \* Independent variables
  - \* Inputs



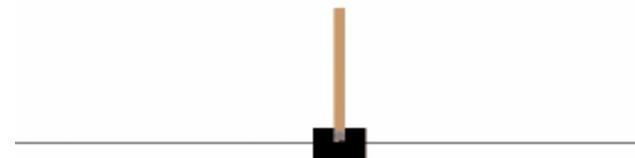
Labels are also known as the following:

- \* Outcomes
  - \* *Targets*
  - \* Dependent variables
  - \* Responses
- **Minibatch** : subset of the training data – for example, 50 samples at a time



# Reinforcement learning

## Cart-Pole game



## Atari Games



Pong, Breakout, Space Invaders, Seaquest, Beam Rider



2015

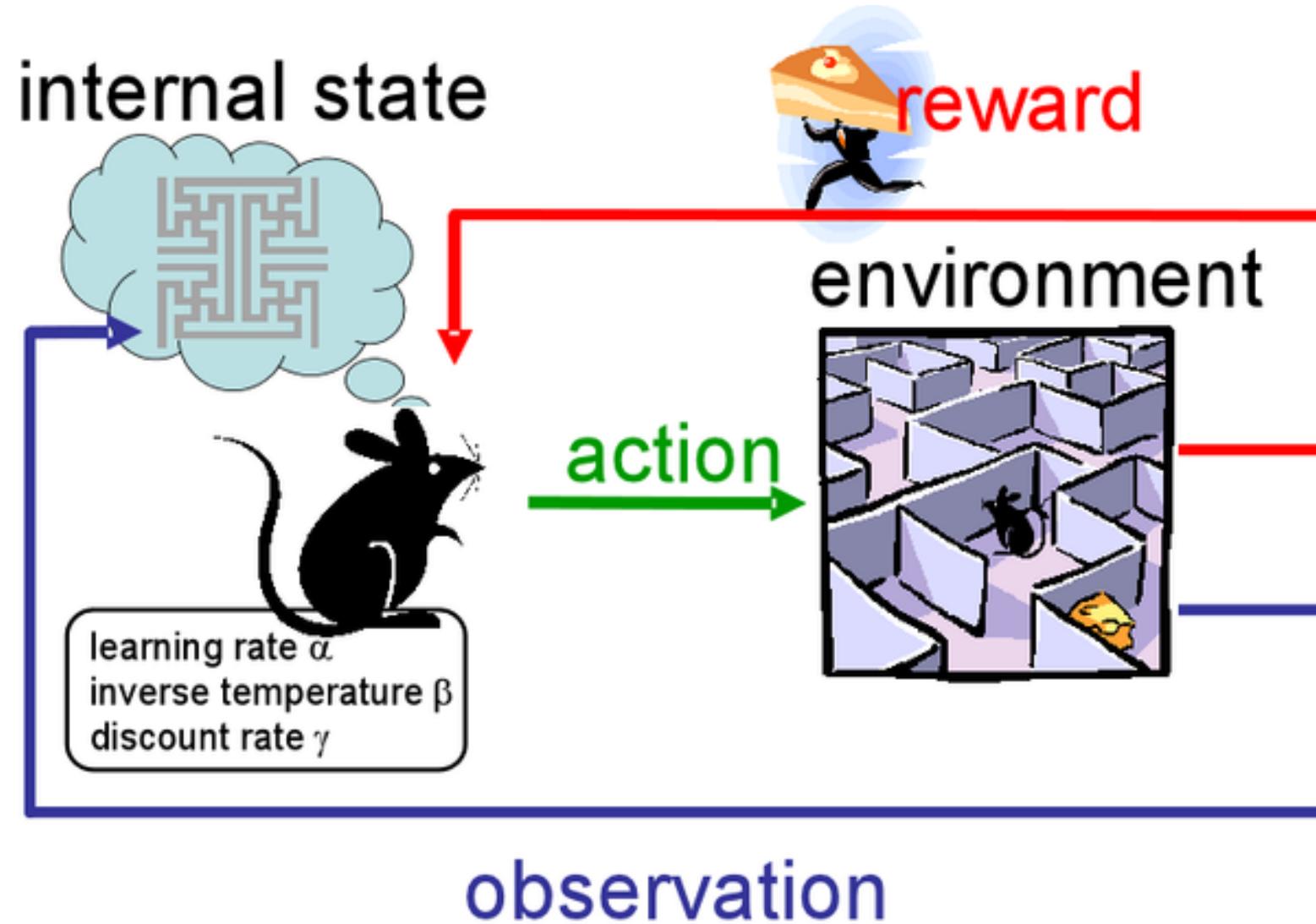
<https://www.youtube.com/watch?v=V1eYniJ0Rnk&vl=ko>  
<https://keon.io/deep-q-learning/>

88

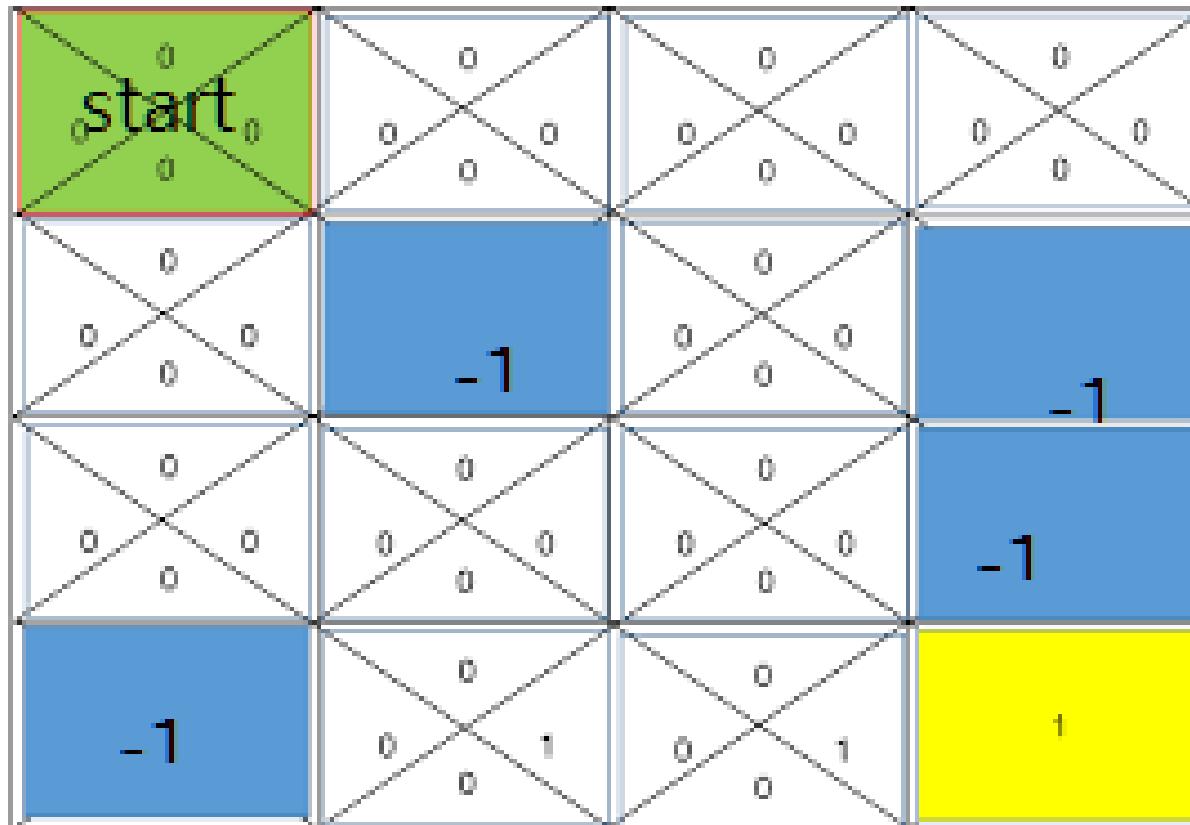
KRISS



# Reinforcement learning



# Reinforcement learning



```
initialize  $Q[\text{num\_states}, \text{num\_actions}]$  arbitrarily
observe initial state  $s$ 
repeat
    select and carry out an action  $a$ 
    observe reward  $r$  and new state  $s'$ 
     $Q[s, a] = Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$ 
     $s = s'$ 
until terminated
```

# Contents

- \*Modern introduction (3+3)
- \*Scikit-learn and Keras tutorial (3+3+3+3+3)
- \*Projects/Applications (3)





# Scikit-learn & Keras tutorial

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.

## Data mining and data analysis

*David Cournapeau*

Classification  
Regression  
Clustering  
Model selection  
Dimensionality reduction  
Preprocessing



Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is *François Chollet*, a Google engineer. Chollet also is the author of the Xception deep neural network model.

```
import keras
print(keras.__version__)
2.4.3
import sklearn
print(sklearn.__version__)
0.23.2
import tensorflow as tf
print(tf.__version__)
2.3.0
```





# Scikit-learn & Keras tutorial

SciPy plug-in or toolkit

[Python for Data Science Cheat Sheet: importing data](#)

[Python for Data Science Cheat Sheet: numpy](#)

[Python for Data Science Cheat Sheet: matplotlib](#)

[Python for Data Science Cheat Sheet: pandas](#)

[Python for Data Science Cheat Sheet: regular expressions](#)

[Python for Data Science Cheat Sheet: python basics](#)

[Beginner's Python Cheat Sheet](#)

[Python Cheat Sheet](#)

[Python Notes/Cheat Sheet](#)

.....

**Artificial neural network, github → python (No.1)**

[Python for Data Science Cheat Sheet: scikit-learn](#)

[Python for Data Science Cheat Sheet: keras](#)

In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy.

<https://github.com/wesm/pydata-book/tree/3rd-edition>

<https://github.com/jakevdp/PythonDataScienceHandbook/tree/master/notebooks>

[https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

<https://scikit-learn.org>

<https://keras.io>



# Machine Learning Mastery With Python

Jason Brownlee

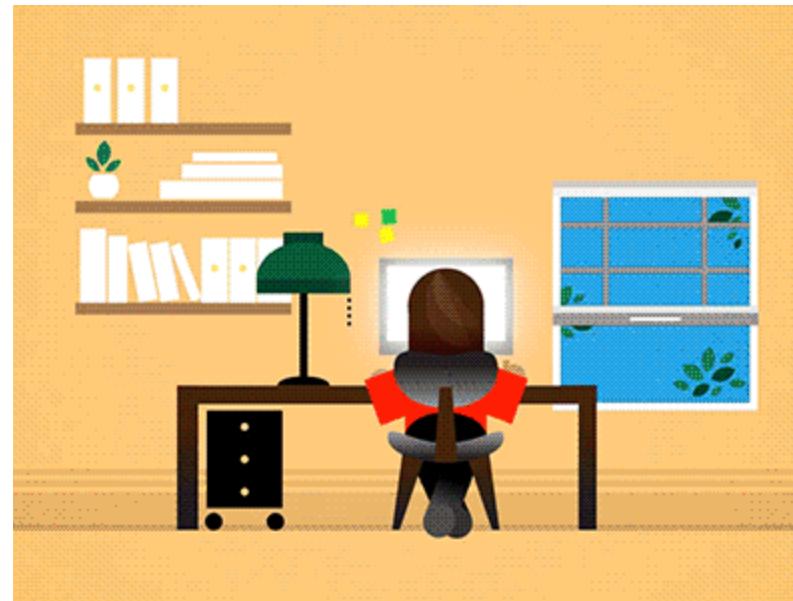
---

Machine Learning with Python/Scikit-Learn

– Application to the Estimation of Occupancy and Human Activities –

---

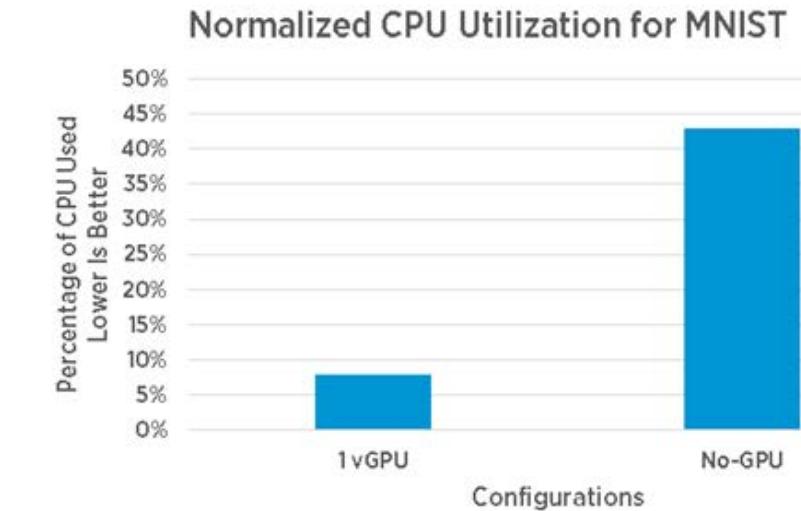
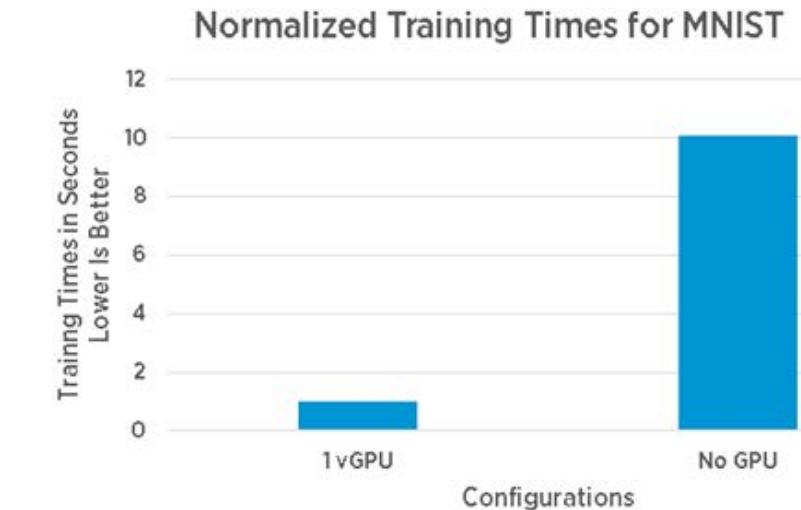
Introduction to Machine Learning with Python





# scikit-learn: Machine learning in Python

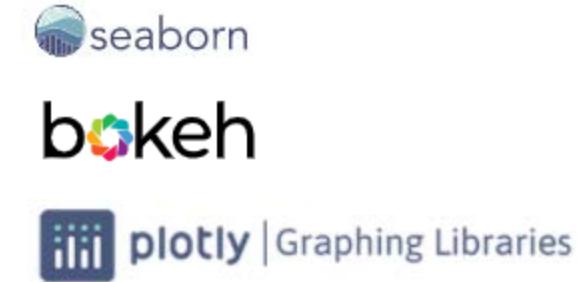
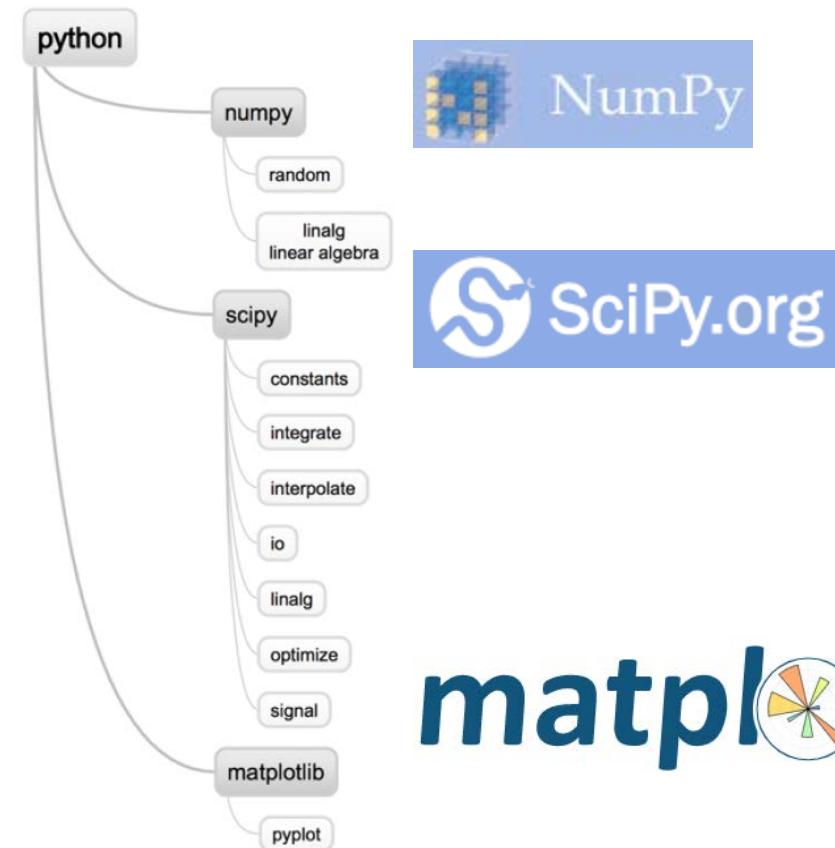
- ubuntu, python3, tensorflow, jupyter (CPU)
- nvidia, cuda, tensorflow-gpu (GPU)
- cuda toolkit, cuDNN, anaconda3, tensorflow, keras



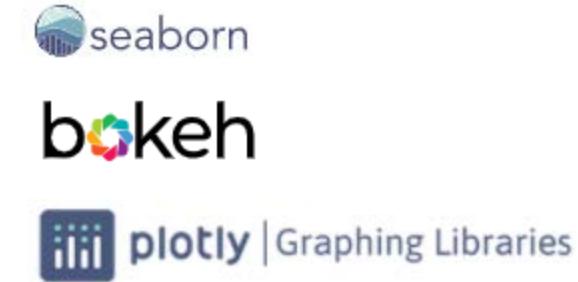


# scikit-learn: Machine learning in Python

- Simple and efficient tools for **data mining** and **data analysis**
- Accessible to everybody, and reusable in various contexts
- Built on **NumPy**, **SciPy**, and **matplotlib**
- Open source, commercially usable - BSD license

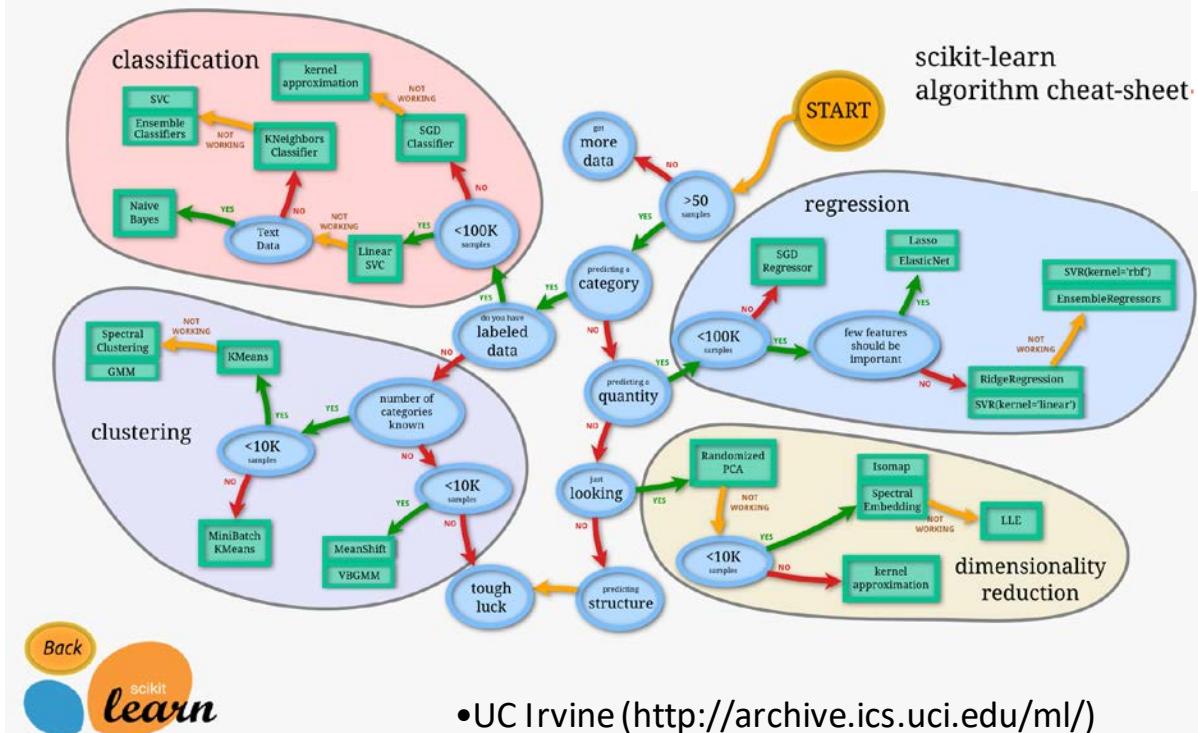


seaborn: statistical data visualization





# scikit-learn



- UCIrvine (<http://archive.ics.uci.edu/ml/>)
- Kaggle Dataset (<http://www.kaggle.com/datasets>)
- Amazon AWS Dataset (<http://aws.amazon.com/ko/datasets>)
- Wiki Dic. (<https://goo.gl/SJHN2k>)
- Quora.com (<http://goo.gl/zDR78y>)
- Dataset Subreddit(<http://www.reddit.com/r/datasets>)
- <http://lib.stat.cmu.edu/datasets/>

scikit-learn user guide, 2019  
Python for Data Science Cheat Sheet: Scikit-Learn

## Most Popular Data Sets (hits since 2007):

3730630:		<a href="#">Iris</a>
2024545:		<a href="#">Adult</a>
1565588:		<a href="#">Wine</a>
1399560:		<a href="#">Breast Cancer Wisconsin (Diagnostic)</a>
1397534:		<a href="#">Heart Disease</a>
1393620:		<a href="#">Wine Quality</a>
1358899:		<a href="#">Bank Marketing</a>
1296805:		<a href="#">Car Evaluation</a>
1070461:		<a href="#">Human Activity Recognition Using Smartphones</a>
1029519:		<a href="#">Abalone</a>
959220:		<a href="#">Forest Fires</a>
841517:		<a href="#">Student Performance</a>



# scikit-learn

## Classification

Identifying to which category an object belongs to.

**Applications:** Spam detection, Image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, ...

— Examples

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, ridge regression, Lasso, ...

— Examples

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, ...

— Examples

## Dimensionality reduction

Reducing the number of random variables to consider.

**Applications:** Visualization, Increased efficiency

**Algorithms:** PCA, feature selection, non-negative matrix factorization.

— Examples

## Model selection

Comparing, validating and choosing parameters and models.

**Goal:** Improved accuracy via parameter tuning

**Modules:** grid search, cross validation, metrics.

— Examples

## Preprocessing

Feature extraction and normalization.

**Application:** Transforming input data such as text for use with machine learning algorithms.

**Modules:** preprocessing, feature extraction.

— Examples



# scikit-learn

## Machine Learning in Python

[Getting Started](#)[Release Highlights for 0.24](#)[GitHub](#)

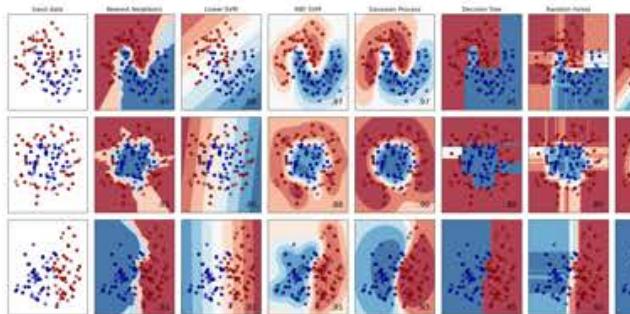
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

### Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

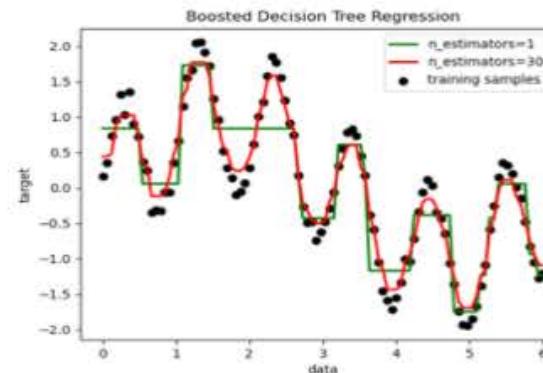
[Examples](#)

### Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...

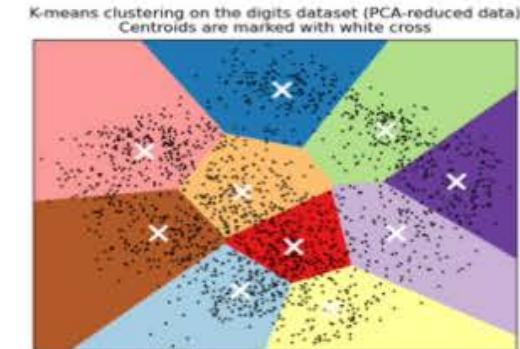
[Examples](#)

### Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...

[Examples](#)



# scikit-learn

Supervised learning → {classification, regression}, LR, SVM, NB, KNN

Unsupervised learning → {clustering, density estimation, dimensionality reduction}, PCA, Kmeans

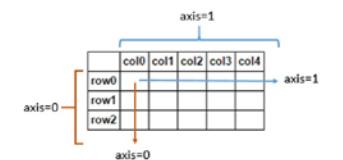
Classifying irises

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> data = iris.data
>>> data.shape
(150, 4)
```

It is made of **150 observations** of irises, each described by **4 features**: their sepal and petal length and width, as detailed in `iris.DESCR`.

**(n\_samples, n\_features) shape**

```
>>> a=[0, 1, 2, 3]
>>> a[-1]
3
>>> a[-2]
2
>>> a[-2:]
[2, 3]
>>> a[-2]
[0, 1]
>>> a[:1]
[1, 12, 11, 9]
>>> a[:1]
[2, 15, 1, 14]])
>>> np.array([[4, 3, 5, 7],
   [1, 12, 11, 9],
   [2, 15, 1, 14]])
>>> np.sort(a, axis=0)
array([[ 1,  3,  1,  7],
       [ 2, 12,  5,  9],
       [ 4, 15, 11, 14]])
>>> np.sort(a, axis=1)
array([[ 3,  4,  5,  7],
       [ 1,  9, 11, 12],
       [ 1, 12, 14, 15]])
```



```
np.sort(a) # axis=-1 or axis=1 , default
```



# Rescaling, Standardization, Normalization

```
# Rescale data (between 0 and 1)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.preprocessing import MinMaxScaler
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = MinMaxScaler(feature_range=(0, 1))
rescaledX = scaler.fit_transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```

```
# Standardize data (0 mean, 1 stdev)
from sklearn.preprocessing import StandardScaler
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = StandardScaler().fit(X)
rescaledX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(rescaledX[0:5,:])
```



# Rescaling, Standardization, Normalization

```
# Normalize data (length of 1)
from sklearn.preprocessing import Normalizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
scaler = Normalizer().fit(X)
normalizedX = scaler.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(normalizedX[0:5,:])
```

```
1 import csv
2
3 # f = open('sample.csv', 'a', encoding='utf-8', newline='')
4 # wr = csv.writer(f)
5 # # wr.writerow([1,2,3])
6 # wr.writerows([[1,2,3],[4,5,6],[7,8,9]])
7 # f.close()
8
9 f = open('sample.csv', 'r', encoding='utf-8')
10 rd = csv.reader(f)
11 # print(rd)
12 for i in rd:
13     print(i)
14     print(type(i))
```



# Binarize Data

```
# binarization
from sklearn.preprocessing import Binarizer
from pandas import read_csv
from numpy import set_printoptions
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
# separate array into input and output components
X = array[:,0:8]
Y = array[:,8]
binarizer = Binarizer(threshold=0.0).fit(X)
binaryX = binarizer.transform(X)
# summarize transformed data
set_printoptions(precision=3)
print(binaryX[0:5,:])
```



# Six classification algorithms

We are going to take a look at six classification algorithms that you can spot-check on your dataset. Starting with two linear machine learning algorithms:

- . Logistic Regression.
- . Linear Discriminant Analysis.

Then looking at four nonlinear machine learning algorithms:

- . k-Nearest Neighbors.
- . Naive Bayes.
- . Classification and Regression Trees.
- . Support Vector Machines.

K-Means Clustering  
Mean-Shift Clustering  
Density-Based Spatial Clustering of Applications with Noise (DBSCAN)  
Expectation-Maximization (EM) Clustering using Gaussian Mixture Models (GMM)  
Agglomerative Hierarchical Clustering

```
# Spot-Check Algorithms
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```



# Five classification algorithms

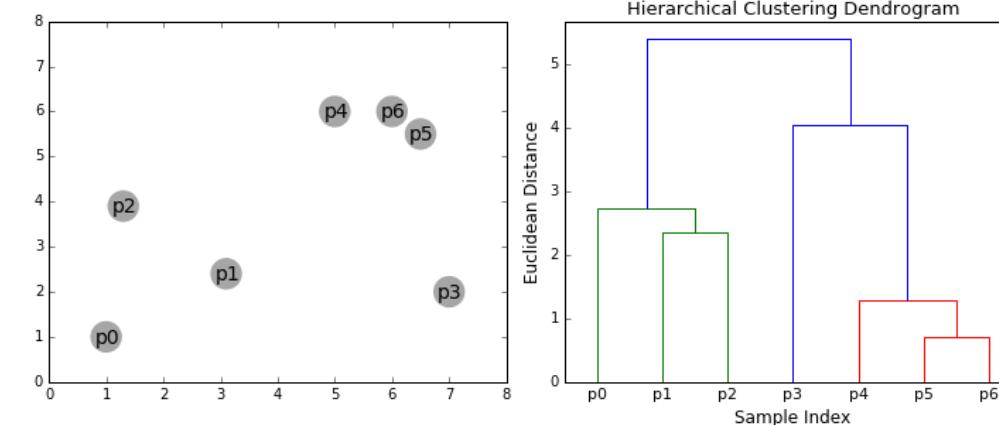
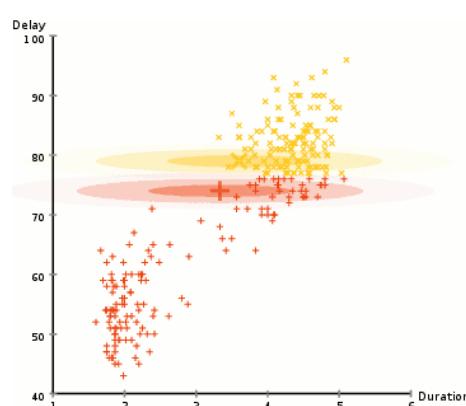
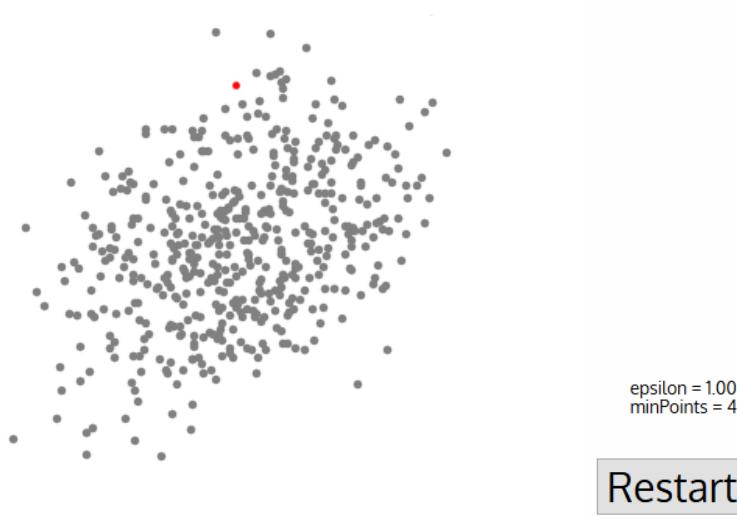
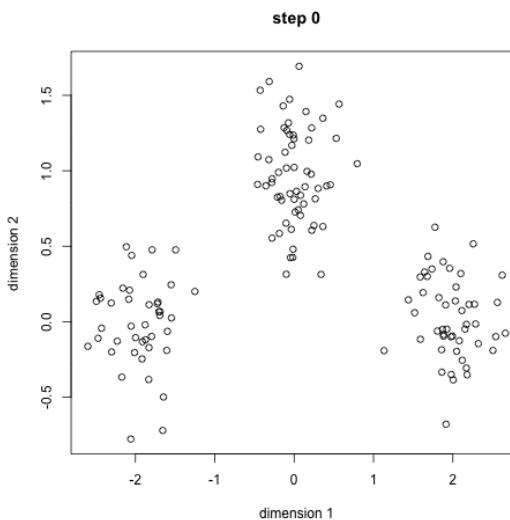
K-Means Clustering

Mean-Shift Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Expectation-Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

Agglomerative Hierarchical Clustering





# Seven regression algorithms

We are going to take a look at seven regression algorithms that you can spot-check on your dataset. Starting with four linear machine learning algorithms:

- . Linear Regression.
- . Ridge Regression.
- . LASSO Linear Regression.
- . Elastic Net Regression.

Then looking at three nonlinear machine learning algorithms:

- . k-Nearest Neighbors.
- . Classification and Regression Trees.
- . Support Vector Machines.



# Feature selection for ML (1/4)

```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)
from pandas import read_csv
from numpy import set_printoptions
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```



# Feature selection for ML (2/4)

```
# Feature Extraction with RFE
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression()
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d") % fit.n_features_
print("Selected Features: %s") % fit.support_
print("Feature Ranking: %s") % fit.ranking_
```



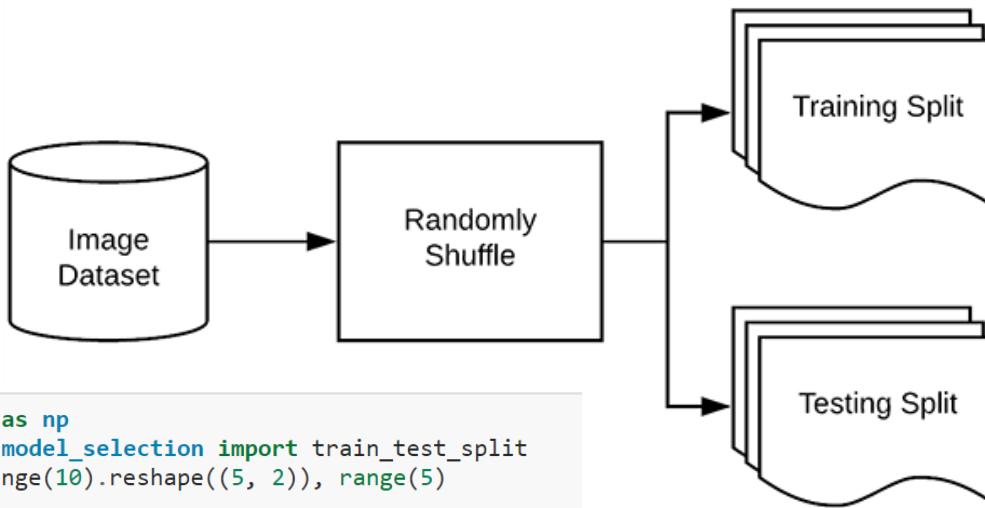
# Feature selection for ML (3/4)

```
# Feature Extraction with PCA
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```



# Feature selection for ML (4/4)

```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```



```

>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]

>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]

>>> train_test_split(y, shuffle=False)
[[0, 1, 2], [3, 4]]

```

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}|$$

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y})^2}{\sum (y_i - \bar{y})^2}$$

Where,

$\hat{y}$  – predicted value of  $y$   
 $\bar{y}$  – mean value of  $y$

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

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

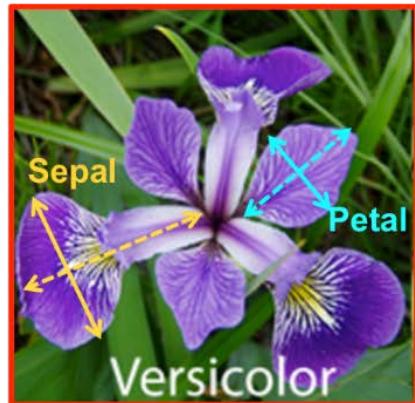
>>> x = np.array([[5, 78, 2, 34, 0],
   [6, 79, 3, 35, 1],
   [7, 80, 4, 36, 2]])
>>> x.ndim
2

>>> 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
```



# scikit-learn

## Classifying irises



3 different types of irises (Setosa, Versicolour, and Virginica)

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
>>> import numpy as np
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> iris_X = iris.data
>>> iris_y = iris.target
>>> np.unique(iris_y)
array([0, 1, 2])
```

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> data = iris.data
>>> data.shape
(150, 4)
```

Exploratory Data Analysis (EDA) is an approach to analyzing datasets to summarize their main characteristics.



# WebGraphviz is Graphviz in the Browser



← → ⌂ Not secure | [webgraphviz.com](http://webgraphviz.com)

WebGraphviz is Graphviz in the Browser

Enter your graphviz data into the Text Area:

(Your Graphviz data is private and never harvested)

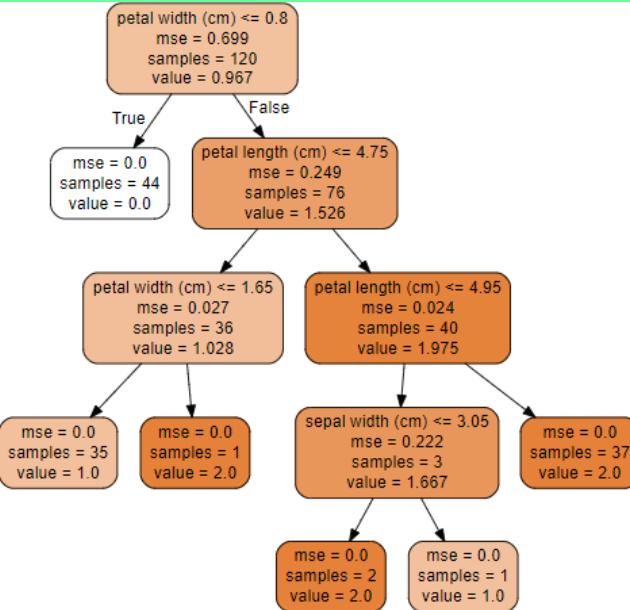
Sample 1 Sample 2 Sample 3 Sample 4 Sample 5

```

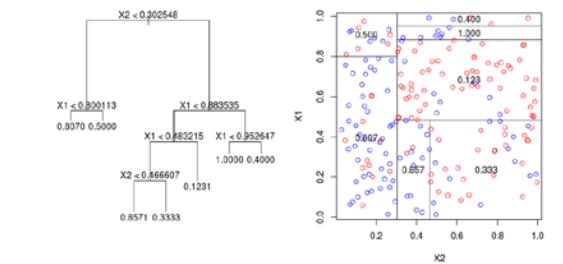
0-> 1 [label=instance=0, labelangle=42, headlabel=" "];
1-> 2 [label="petal length (cm) < 4.75\mse = 0.249\nsamples = 76\nvalue = 1.526", fillcolor="#f2c09c"];
0-> 2 [label=distance=2.5, labelangle=-45, headlabel="False"];
3-> 2 [label="petal width (cm) < 1.65\mse = 0.027\nsamples = 36\nvalue = 1.028", fillcolor="#f2c09c"];
2-> 3 ;
4-> 3 ;
5-> 5 ;
6-> 5 ;
7-> 6 ;
8-> 7 ;
9-> 8 ;
10-> 9 ;
11-> 10 ;

```

Generate Graph!

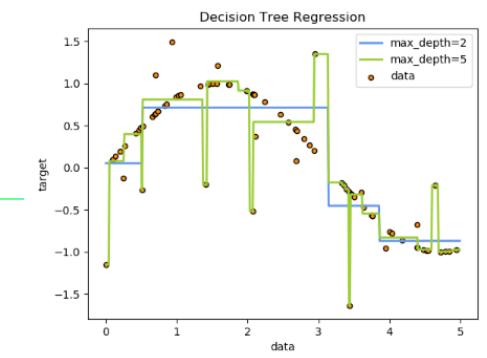


**iris\_model\_dot\_file\_output.ipynb**  
**iris\_tree.dot**



input

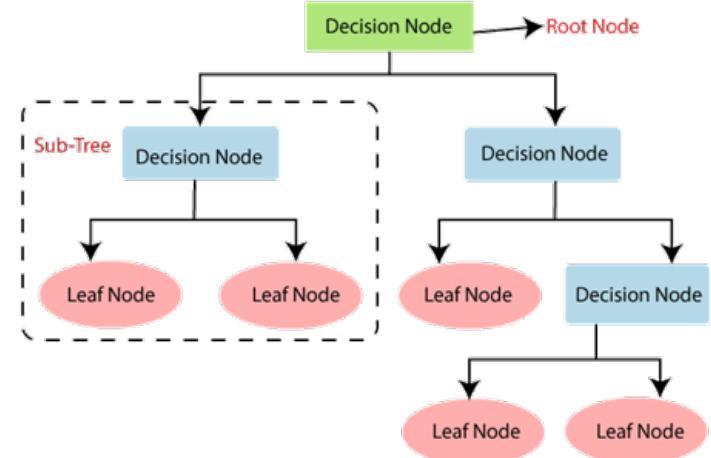
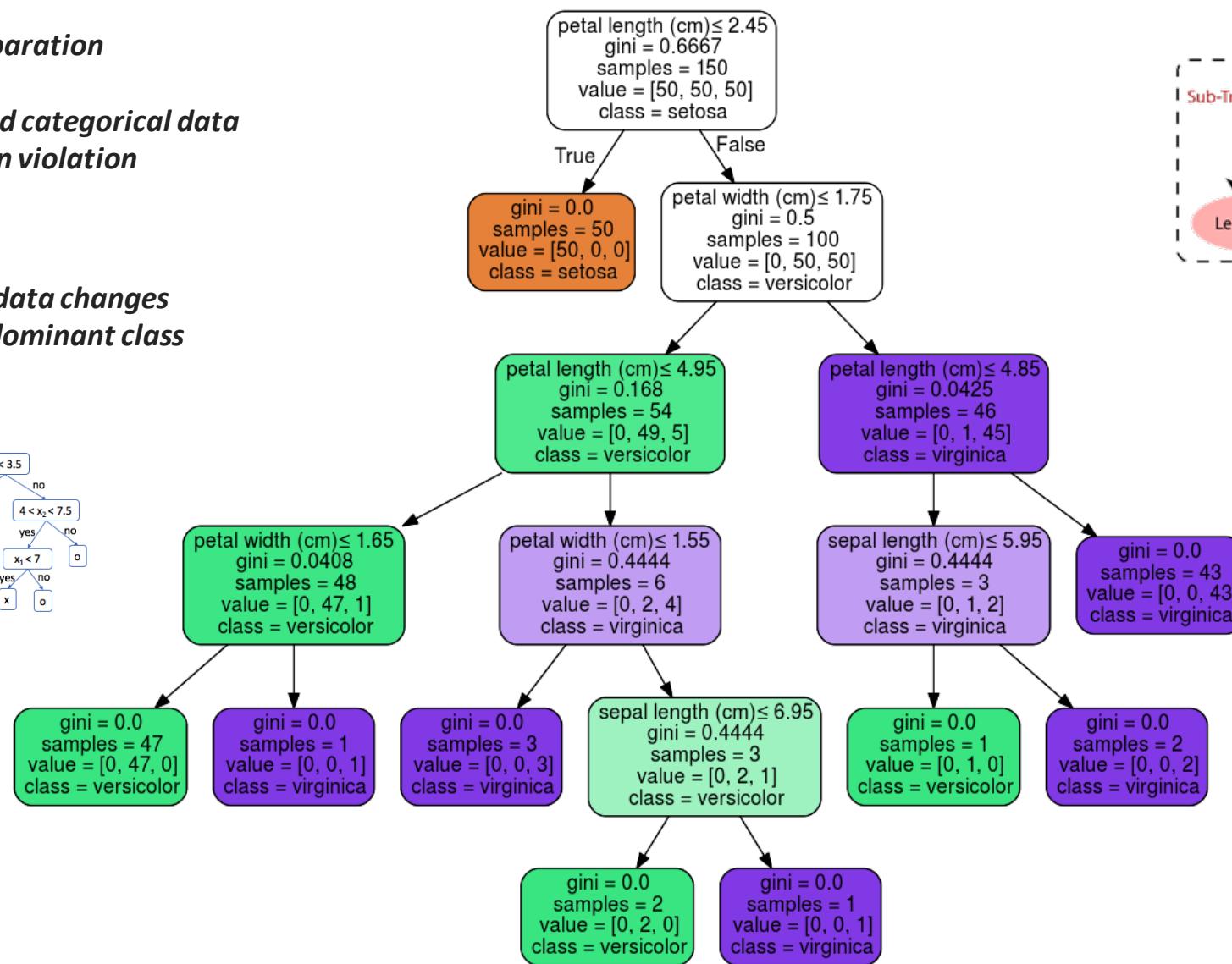
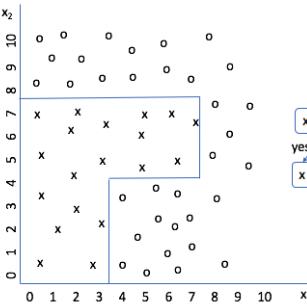
output



# Decision trees

1. **Interpretable**
2. **Little to no data preparation**
3. **Scale well**
4. **Handle numerical and categorical data**
5. **Robust to assumption violation**

1. **Overfitting**
2. **Non-robust to input data changes**
3. **Biased towards the dominant class**



A Decision tree is a flowchart like a tree structure, wherein each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



# k-means clustering

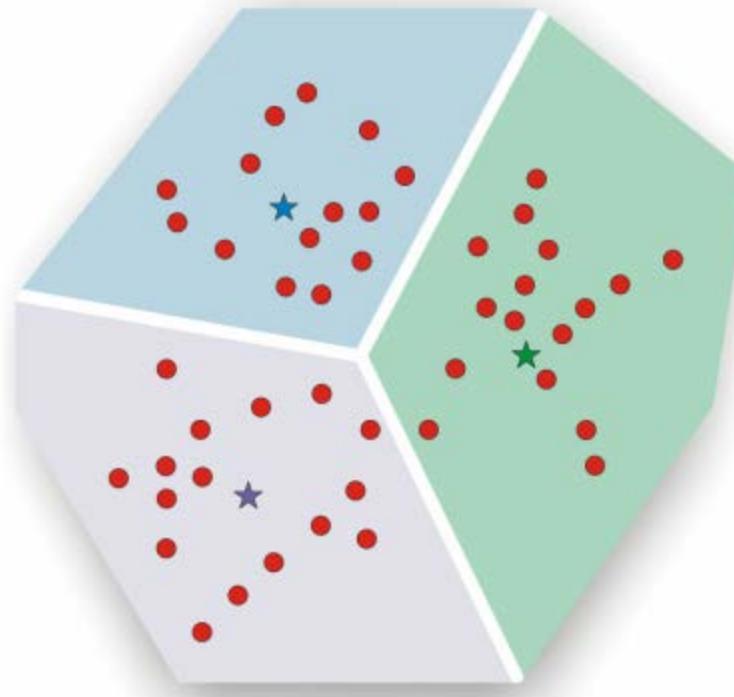
Unsupervised learning

Given a set of  $n$  observations  $(X_1, X_2, \dots, X_n)$ , where  $X_i \in \mathbb{R}^d$ , k-means clustering seeks to partition the  $n$  observations into  $k$  ( $k \leq n$ ) sets  $S = S_1, S_2, \dots, S_k$  so as to minimize the within-cluster sum of squares, which happens to be the variance.

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg \min_{\mathbf{S}} \sum_{i=1}^k |S_i| \text{Var } S_i,$$

where  $\boldsymbol{\mu}_i$  is the mean of the points in  $S_i$ .

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \frac{1}{2|S_i|} \sum_{\mathbf{x}, \mathbf{y} \in S_i} \|\mathbf{x} - \mathbf{y}\|^2.$$





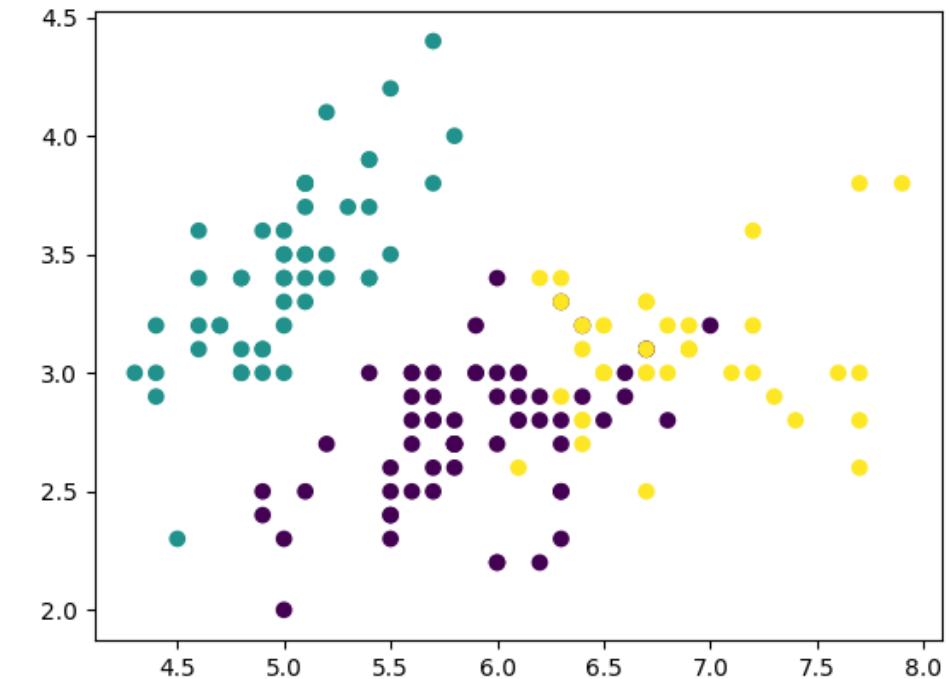
# K-means clustering

Classifying irises



3 different types of irises (Setosa, Versicolour, and Virginica)

```
from sklearn.cluster import KMeans
from sklearn import datasets
from pylab import *
iris = datasets.load_iris()
X, y = iris.data, iris.target
k_means = KMeans(n_clusters=3, random_state=0) # Fixing the RNG in kmeans
k_means.fit(X)
y_pred = k_means.predict(X)
scatter(X[:, 0], X[:, 1], c=y_pred);
show()
```





# KNN (k nearest neighbors) classification

```
>>> import numpy as np
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> iris_X = iris.data
>>> iris_y = iris.target
>>> np.unique(iris_y)
array([0, 1, 2])
```

```
>>> # Split iris data in train and test data
>>> # A random permutation, to split the data randomly
>>> np.random.seed(0)
>>> indices = np.random.permutation(len(iris_X))
>>> iris_X_train = iris_X[indices[:-10]]
>>> iris_y_train = iris_y[indices[:-10]]
>>> iris_X_test = iris_X[indices[-10:]]
>>> iris_y_test = iris_y[indices[-10:]]
>>> # Create and fit a nearest-neighbor classifier
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier()
>>> knn.fit(iris_X_train, iris_y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
>>> knn.predict(iris_X_test)
array([1, 2, 1, 0, 0, 0, 2, 1, 2, 0])
>>> iris_y_test
array([1, 1, 1, 0, 0, 0, 2, 1, 2, 0])
```

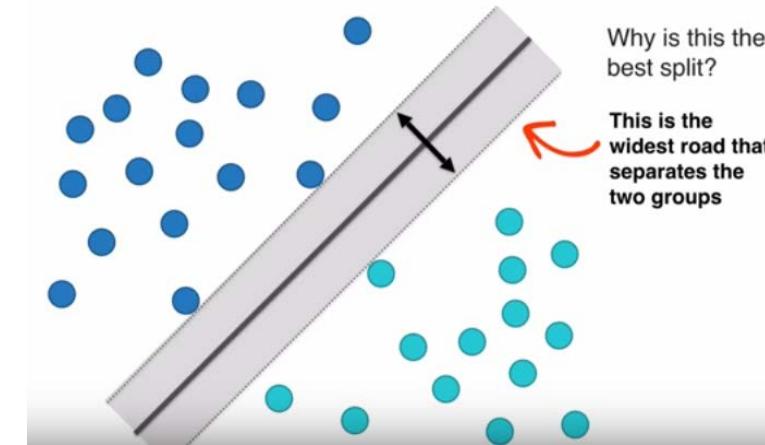
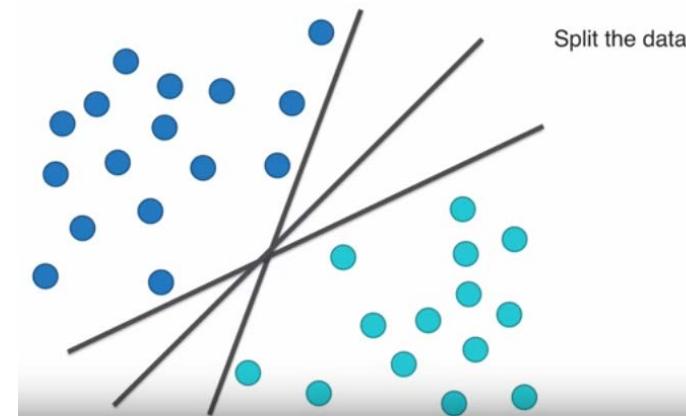
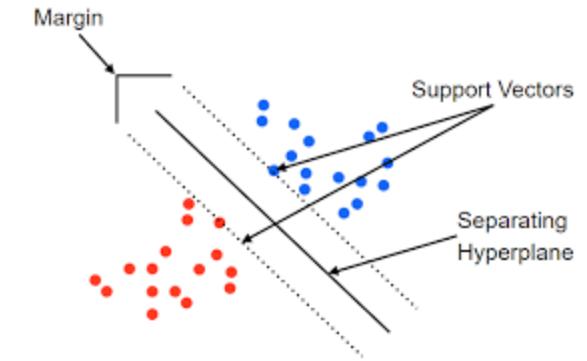
# Support vector machine

The SVM algorithm has been widely applied in the biological and other sciences.

Hand-written characters can be recognized using SVM.

Classification of images can also be performed using SVMs.

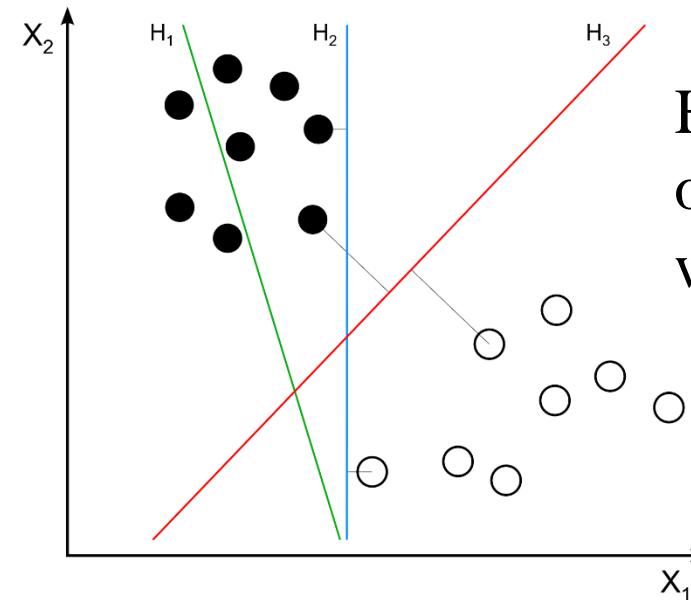
Not a probabilistic approach, no credible interval



Constrained optimization problem → Lagrange multipliers



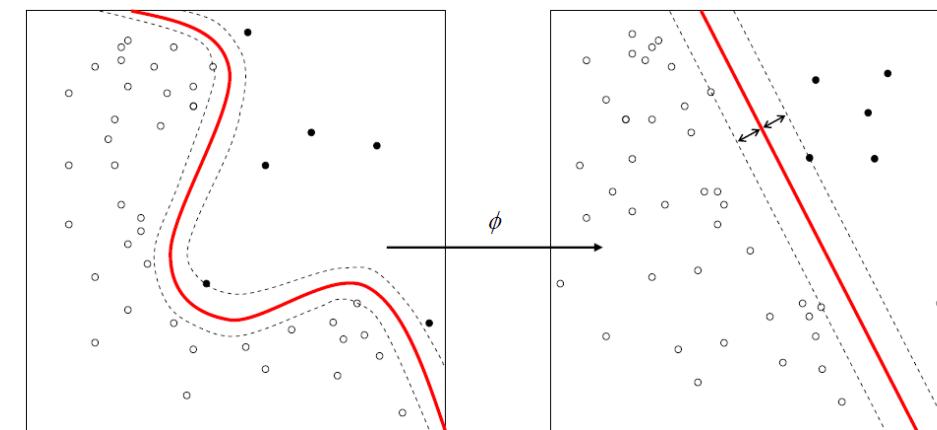
# Support vector machine



$H_1$  does not separate the classes.  $H_2$  does, but only with a small margin.  $H_3$  separates them with the maximum margin.

The vectors defining the hyperplanes can be chosen to be linear combinations with parameters of images of feature vectors that occur in the data base.

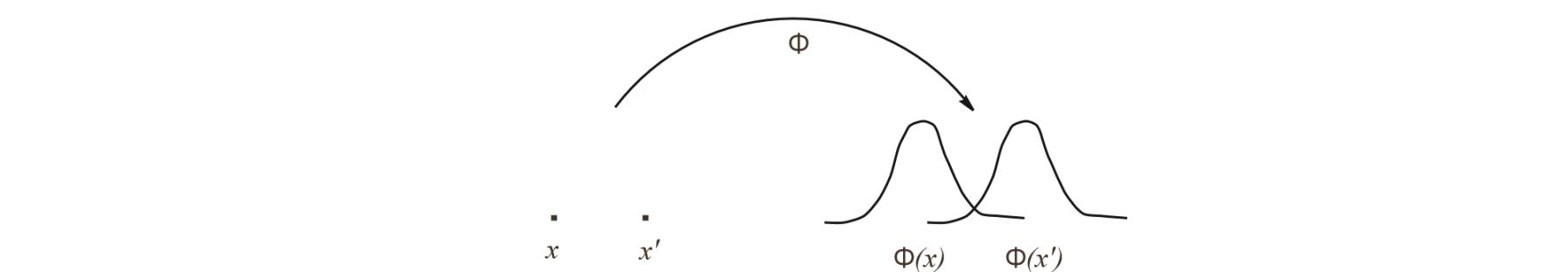
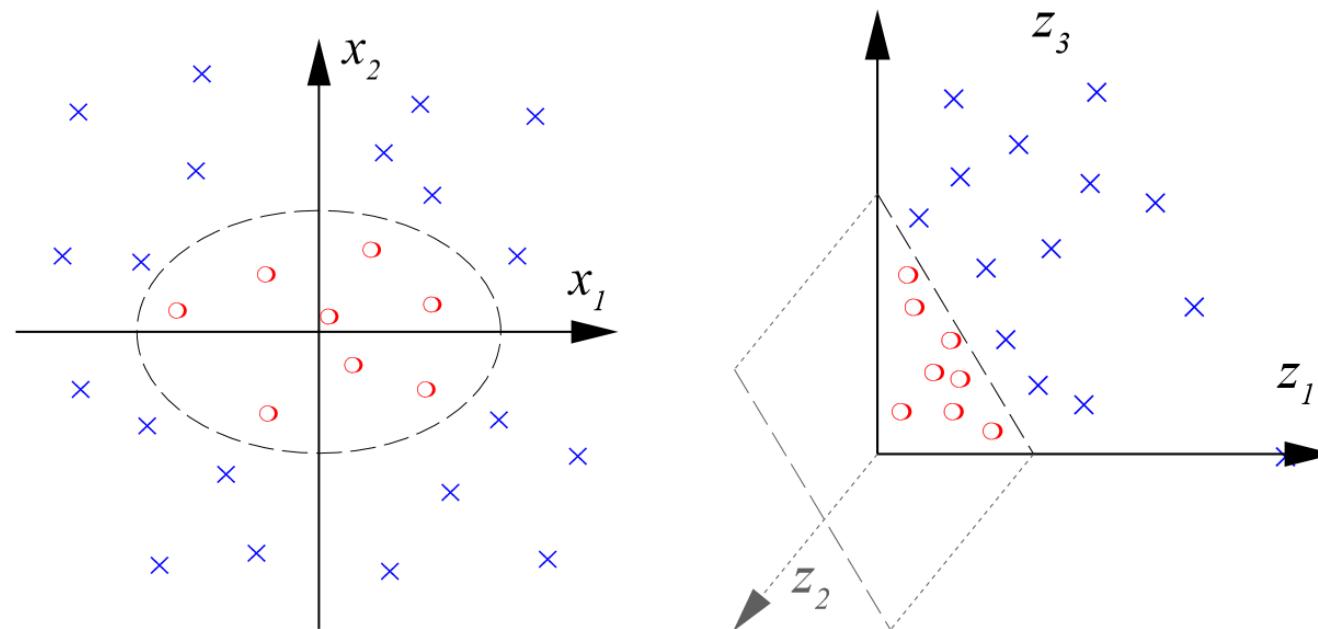
not linearly separable

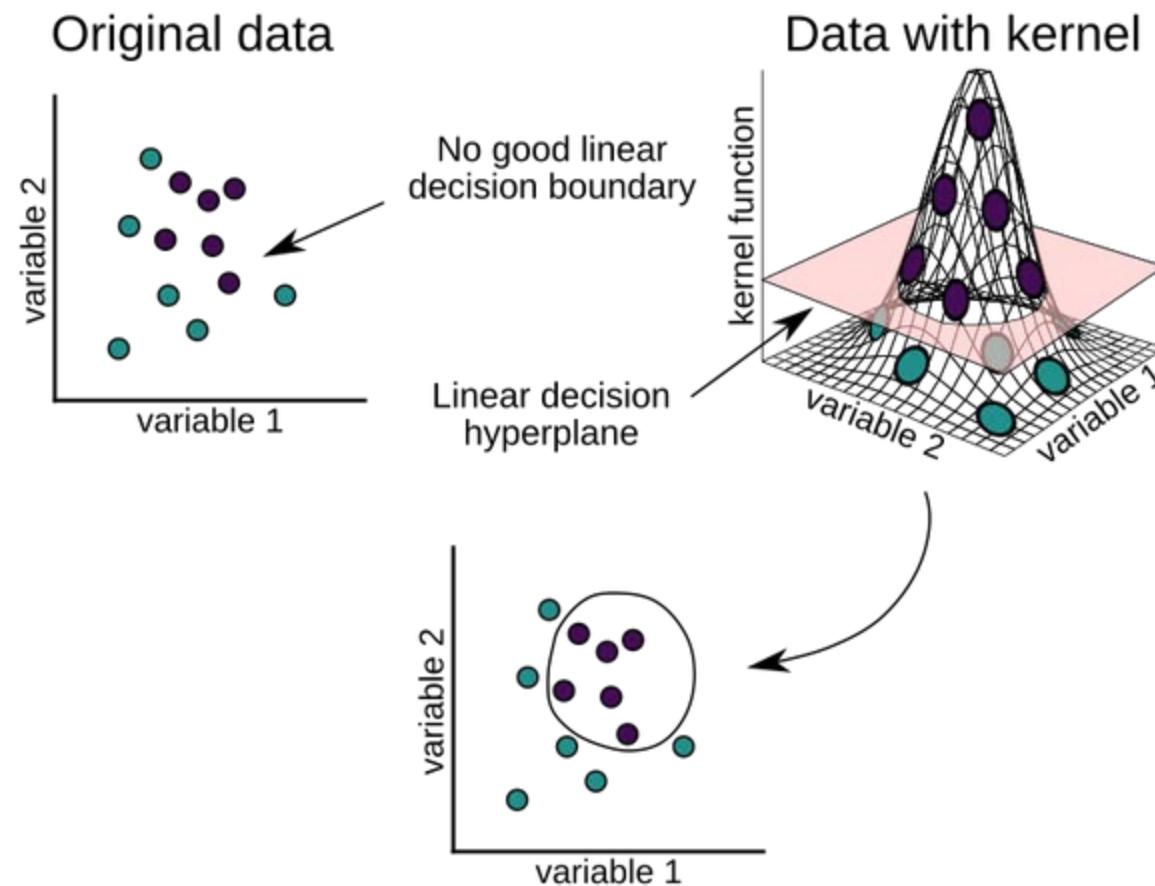




$$\Phi : R^2 \rightarrow R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt(2)x_1x_2, x_2^2)$$







```
import numpy as np
class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape
        y_ = np.where(y <= 0, -1, 1)
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]
    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx)
from sklearn import datasets
def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred)/len(y_true)
    return accuracy

X, y = datasets.make_blobs(n_samples=50, n_features=2, centers=2, cluster_std=1.05, random_state=40)
y = np.where(y == 0, -1, 1)
clf = SVM()
clf.fit(X, y)
print(clf.w, clf.b)
y_pred = clf.predict(X)
acc = accuracy(y, y_pred)
print("Training Accuracy: ", acc)
```



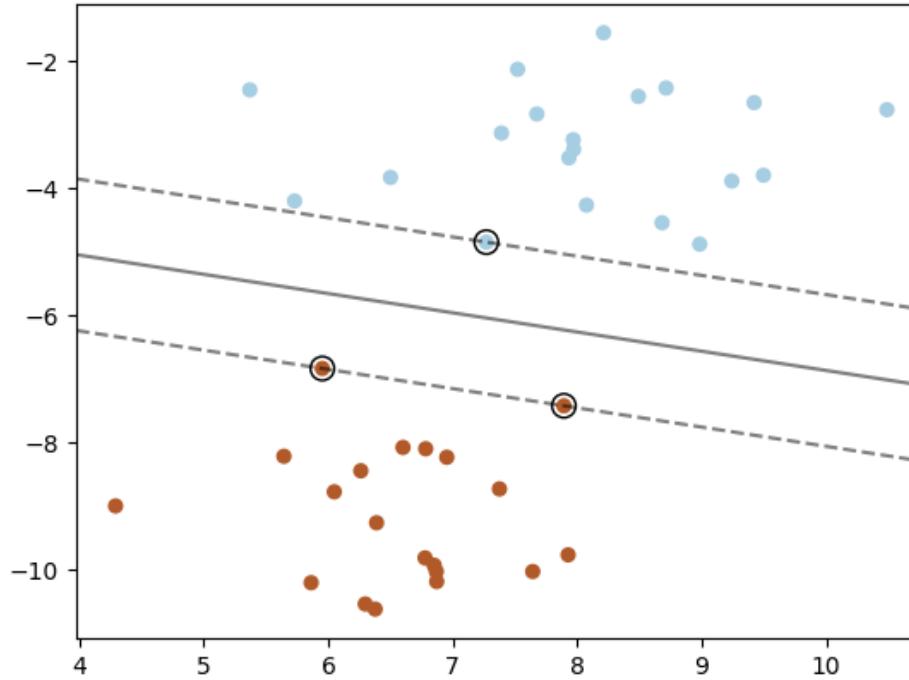
# Support vector machine

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import make_blobs
# we create 40 separable points
X, y = make_blobs(n_samples=40, centers=2, random_state=6)
# fit the model, don't regularize for illustration purposes
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X, y)

plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)

# plot the decision function
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5, linestyles='[-, -, --]')
# plot support vectors
ax.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')
plt.show()
```



**evaluation**

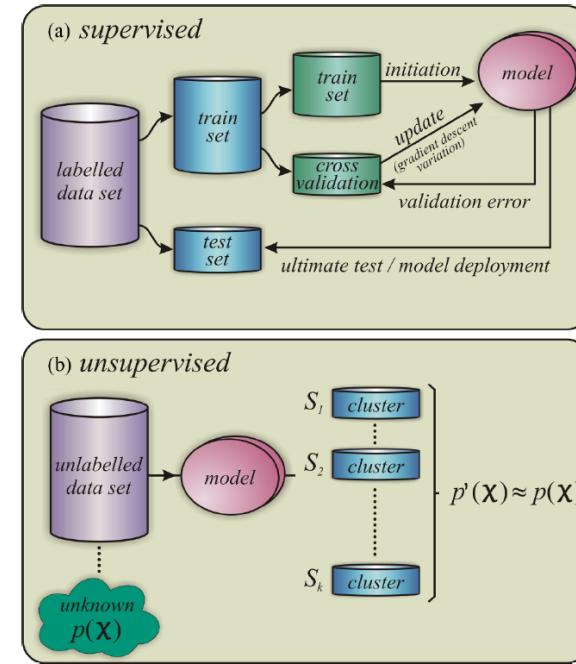
**predict**



# DBSCAN

```
from sklearn.cluster import DBSCAN
import numpy as np
X=np.array([[1, 2], [2, 2], [2, 3], [8, 7], [8, 8], [25, 80]])
clustering=DBSCAN(eps=3, min_samples=2).fit(X)
print(clustering.labels_)
print(clustering)
```

```
[ 0 0 0 1 1 -1]
DBSCAN(algorithm='auto', eps=3, leaf_size=30, metric='euclidean',
 metric_params=None, min_samples=2, n_jobs=None, p=None)
```



---

K-Means Clustering

Mean-Shift Clustering

Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

Expectation-Maximization (EM) Clustering using Gaussian Mixture Models (GMM)

Agglomerative Hierarchical Clustering



# Pipeline

Pipelines allow you to create a single object that includes all steps from data preprocessing and classification.

```
# Create a pipeline that standardizes the data then creates a model
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# create pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(estimators)
# evaluate pipeline
seed = 7
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

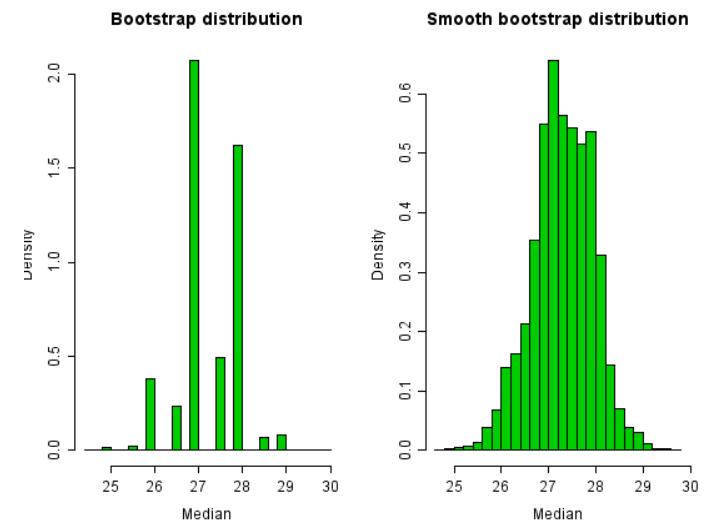
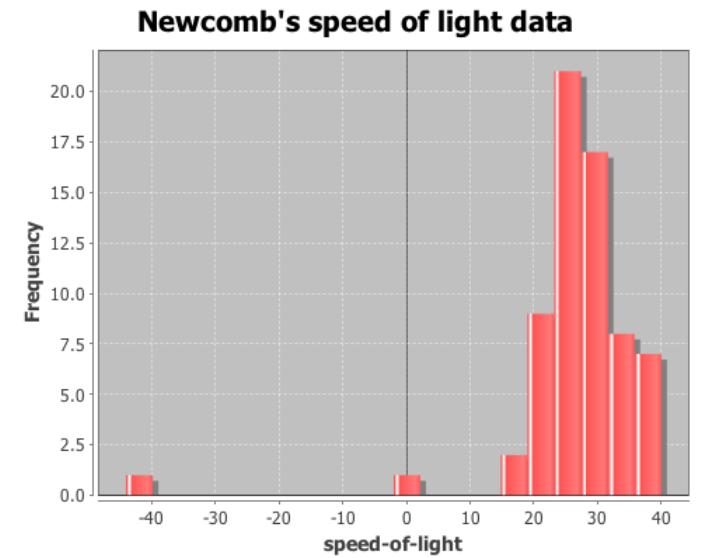
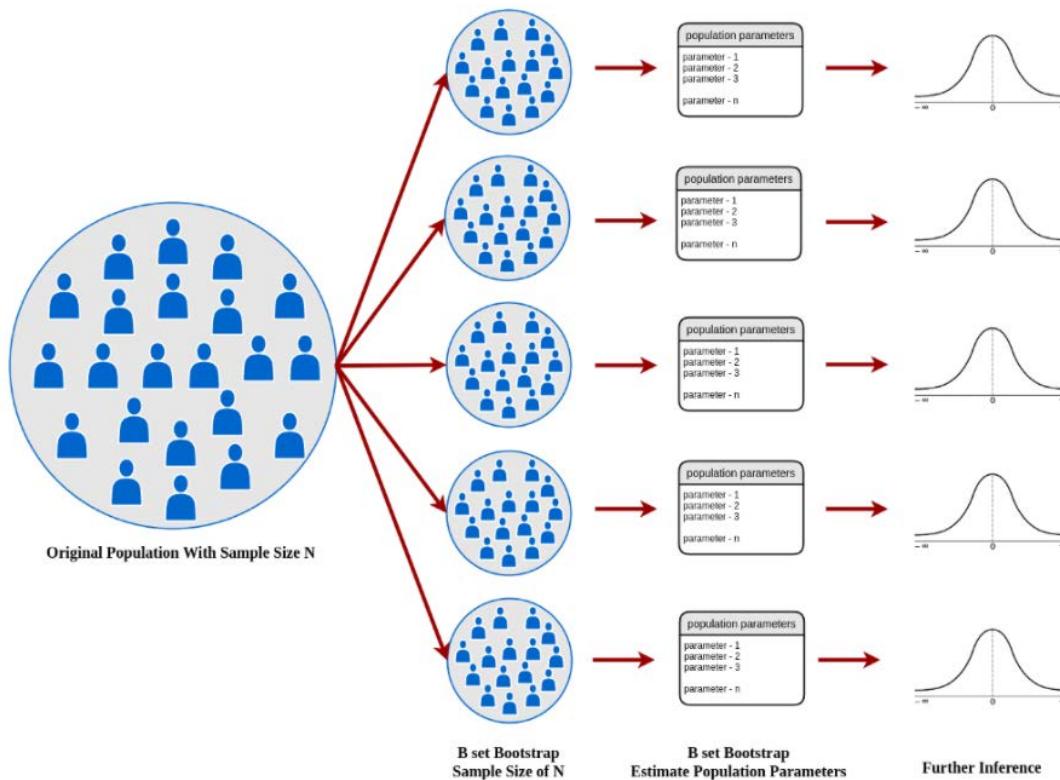


# Pipeline

```
# Create a pipeline that extracts features from the data then creates a model
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest
# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# create feature union
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=6)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('logistic', LogisticRegression()))
model = Pipeline(estimators)
# evaluate pipeline
seed = 7
kfold = KFold(n_splits=10, random_state=seed)
results = cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```



# Bootstrapping



[Newcomb's\\_speed\\_of\\_light\\_outliers.ipynb](#)

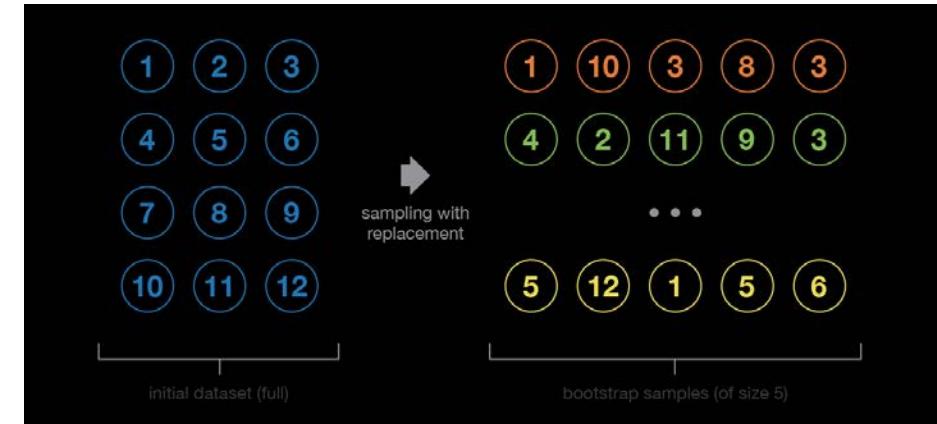
[Risk Engineering: Analyzing speed of light measurements](#) (risk-engineering.org)

[Bootstrapping \(statistics\) - Wikipedia](#)



# Ensemble ML algorithms for improving the performance of models

- . **Bagging Ensembles (bootstrap aggregating)** including Bagged Decision Trees, Random Forest, and Extra Trees.
- . **Boosting Ensembles** including AdaBoost and Stochastic Gradient Boosting.
- . **Voting Ensembles** for averaging the predictions for any arbitrary models.



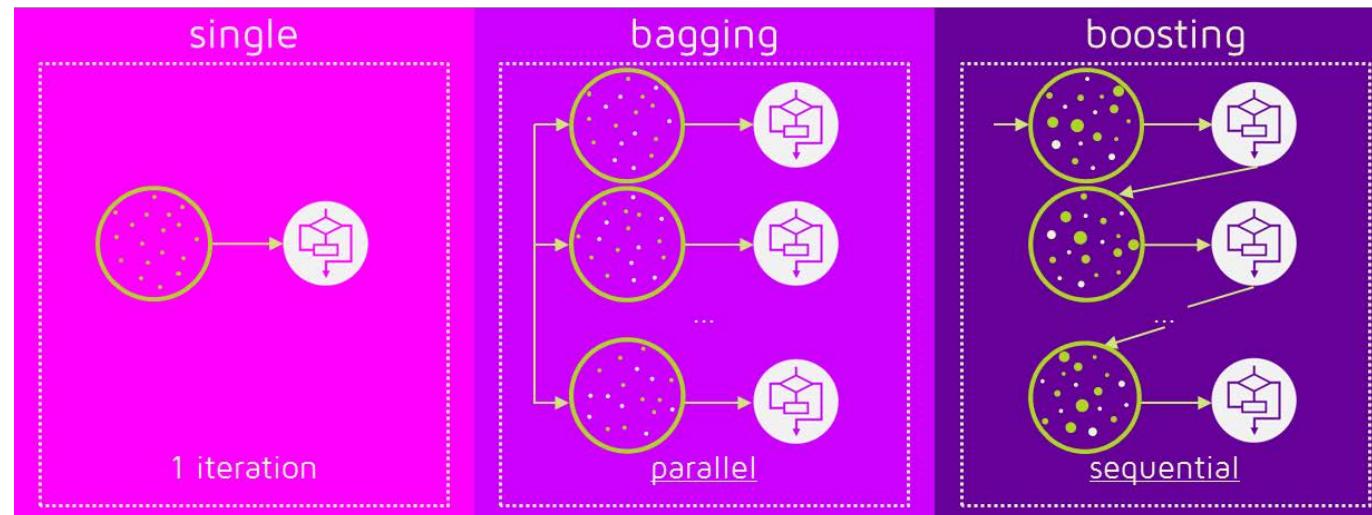
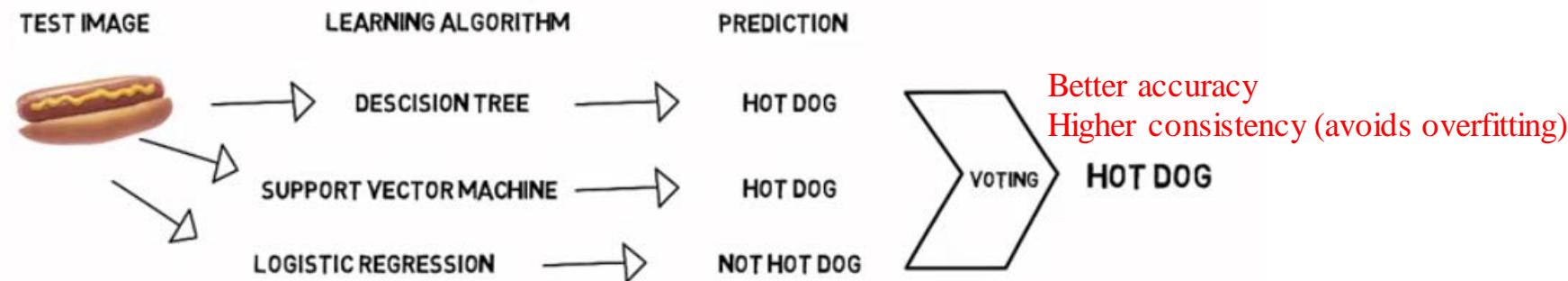
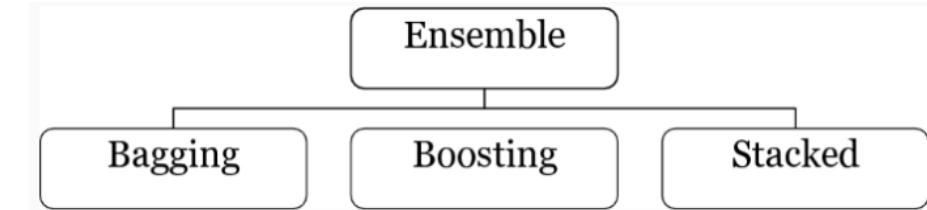
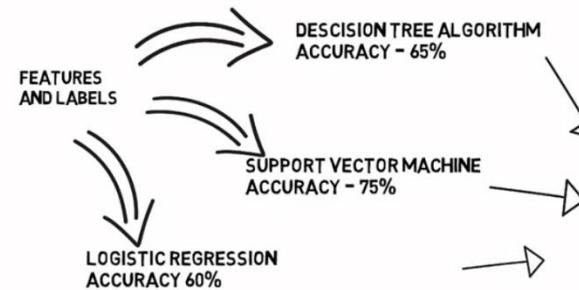
```
bg=BaggingClassifier( DecisionTreeClassifier(), max_samples=0.5, max_features=1.0, n_estimators=20)
bg.fit(x_train,y_train)
bg.score(x_test,y_test)
bg.score(x_train,y_train)
```

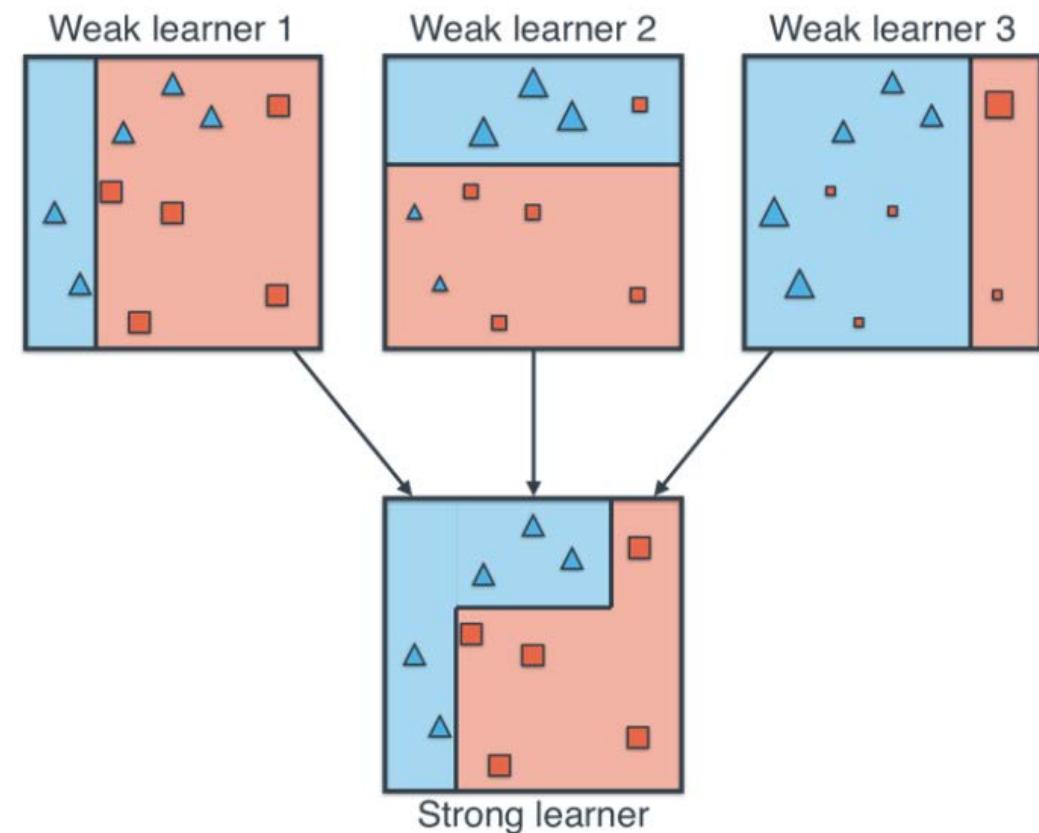
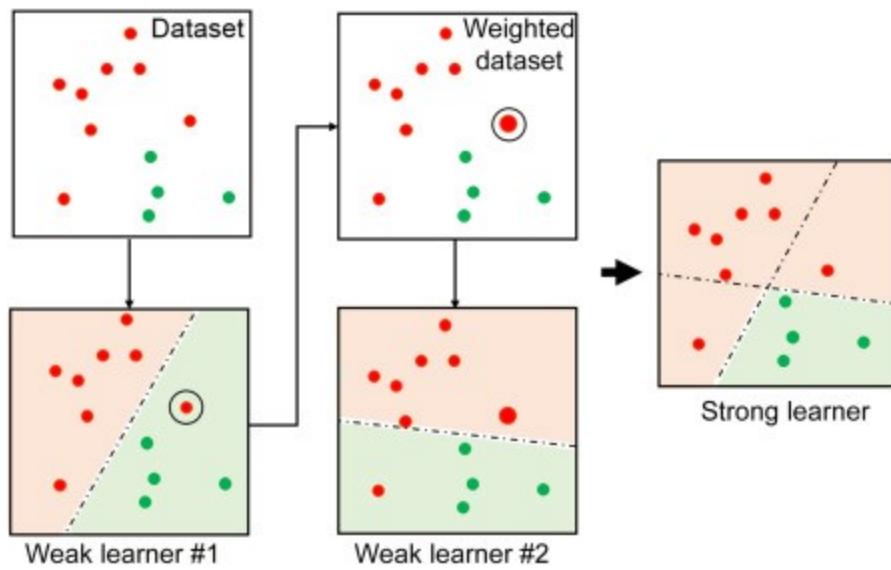
~1?, overfitting?

```
adb=AdaBoostClassifier( DecisionTreeClassifier(), n_estimators=10, learning_rate=1)
adb.fit(x_train,y_train)
adb.score(x_test,y_test)
adb.score(x_train,y_train)
```

~1?, overfitting?

# Ensemble learning (Bagging, Boosting, Stacked)







# RandomForestClassifier Ensemble method

## Classifying irises

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.utils import shuffle
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
np.random.seed(123)
#rn.seed(1234)
#tf.set_random_seed(210)
iris = datasets.load_iris()
X = iris.data[:,4]
y = iris.target
X, y = shuffle(X,y,random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)
#rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
y_pred_rf = rnd_clf.predict(X_test)
```

sepal length (cm) 0.09711140999352112  
sepal width (cm) 0.013922805429994443  
petal length (cm) 0.49479568987578054  
petal width (cm) 0.3941700947007034

3 different types of irises (Setosa, Versicolour, and Virginica)

```
rf=RandomForestClassifier(n_estimators=20)
rf.fit(x_train,y_train)
rf.score(x_test,y_test)
pred=rf.predict(x_test)
```

**from sklearn.ensemble import RandomForestRegressor**

```
rf = RandomForestRegressor(n_estimators = 80, max_features = 'auto')
rf.fit(xtrain, ytrain)
print('Training Done using Random Forest')
```

*#Random Forest - Ensemble of Descision Trees*

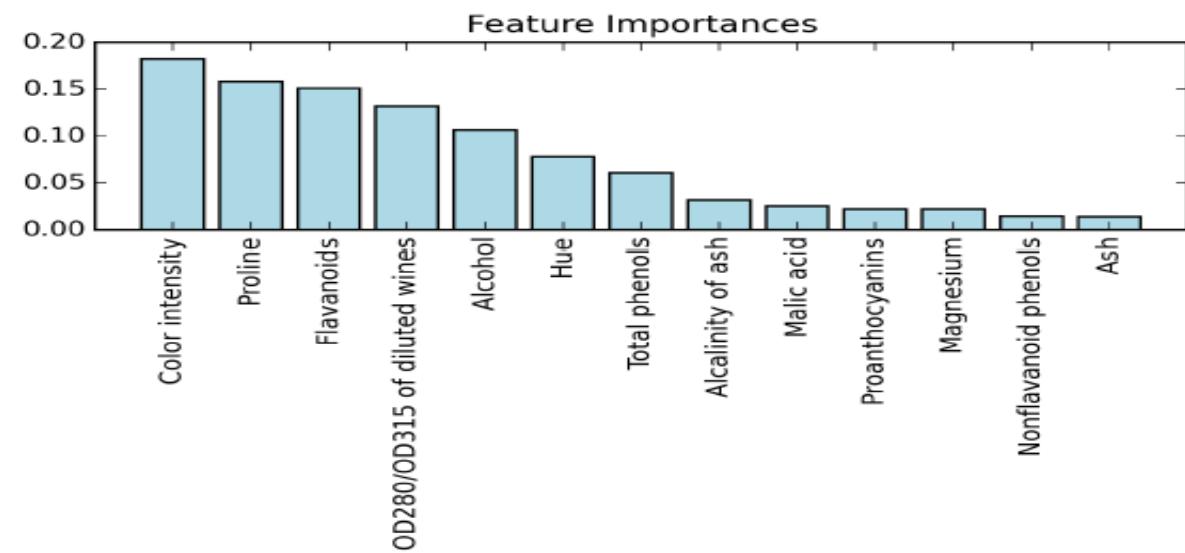
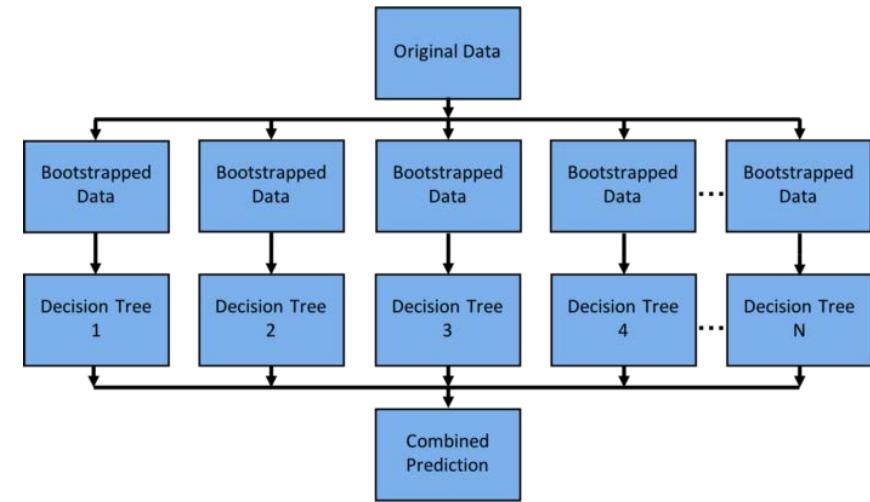
```
rf = RandomForestClassifier(n_estimators=20)
rf.fit(x_train,y_train)
```



# RandomForestClassifier

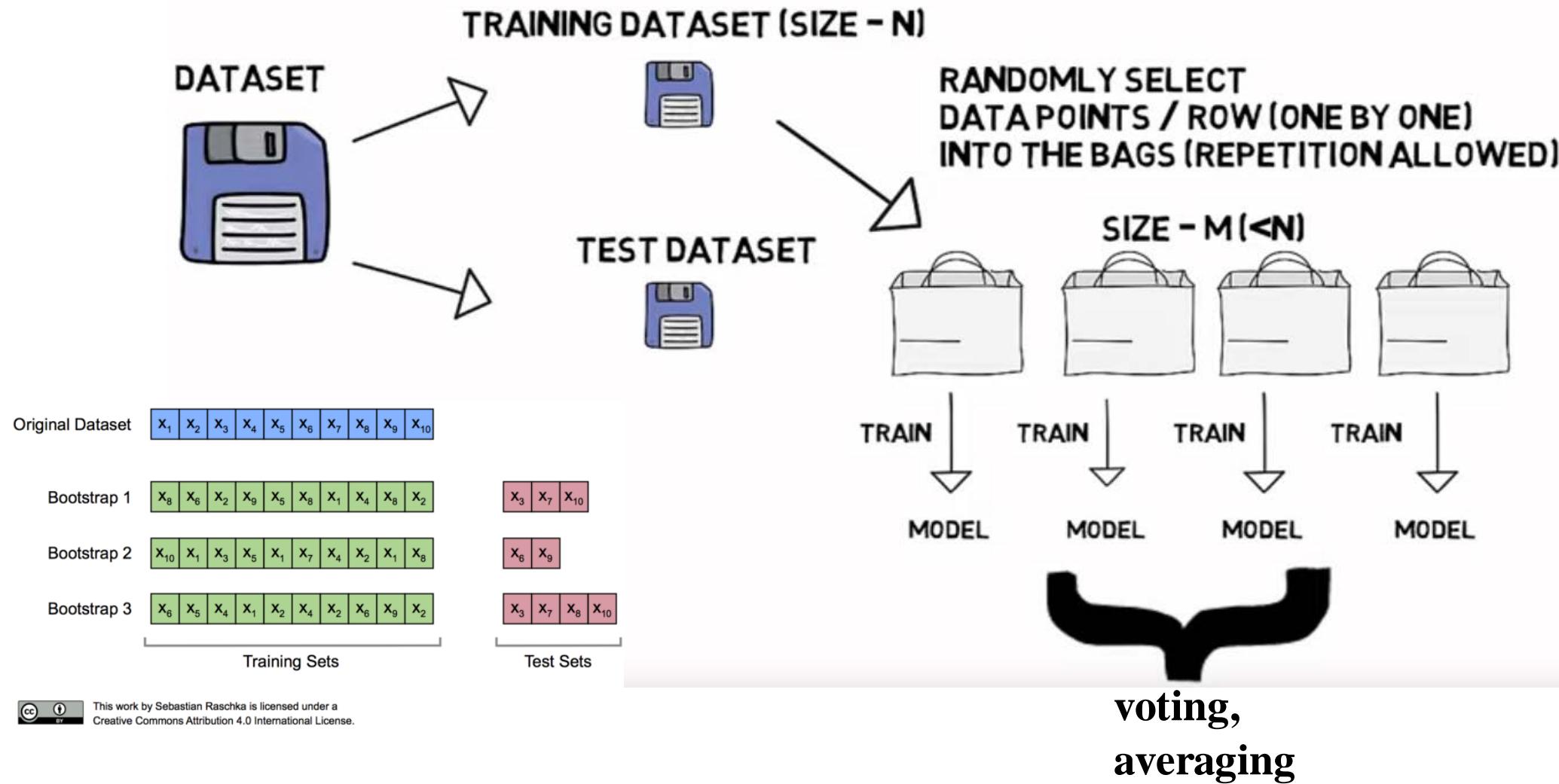
```
>>> from sklearn.ensemble import RandomForestClassifier
>>> feat_labels = df_wine.columns[1:]
>>> forest = RandomForestClassifier(n_estimators=10000,
...                                 random_state=0,
...                                 n_jobs=-1)
>>> forest.fit(X_train, y_train)
>>> importances = forest.feature_importances_
>>> indices = np.argsort(importances)[::-1]
>>> for f in range(X_train.shape[1]):
...     print("%2d %-*s %f" % (f + 1, 30,
...                            feat_labels[indices[f]],
...                            importances[indices[f]]))

>>> plt.title('Feature Importances')
>>> plt.bar(range(X_train.shape[1]),
...          importances[indices],
...          color='lightblue',
...          align='center')
>>> plt.xticks(range(X_train.shape[1]),
...             feat_labels[indices], rotation=90)
>>> plt.xlim([-1, X_train.shape[1]])
>>> plt.tight_layout()
>>> plt.show()
```





# Bootstrap Aggregating (=Bagging) reducing over-fitting



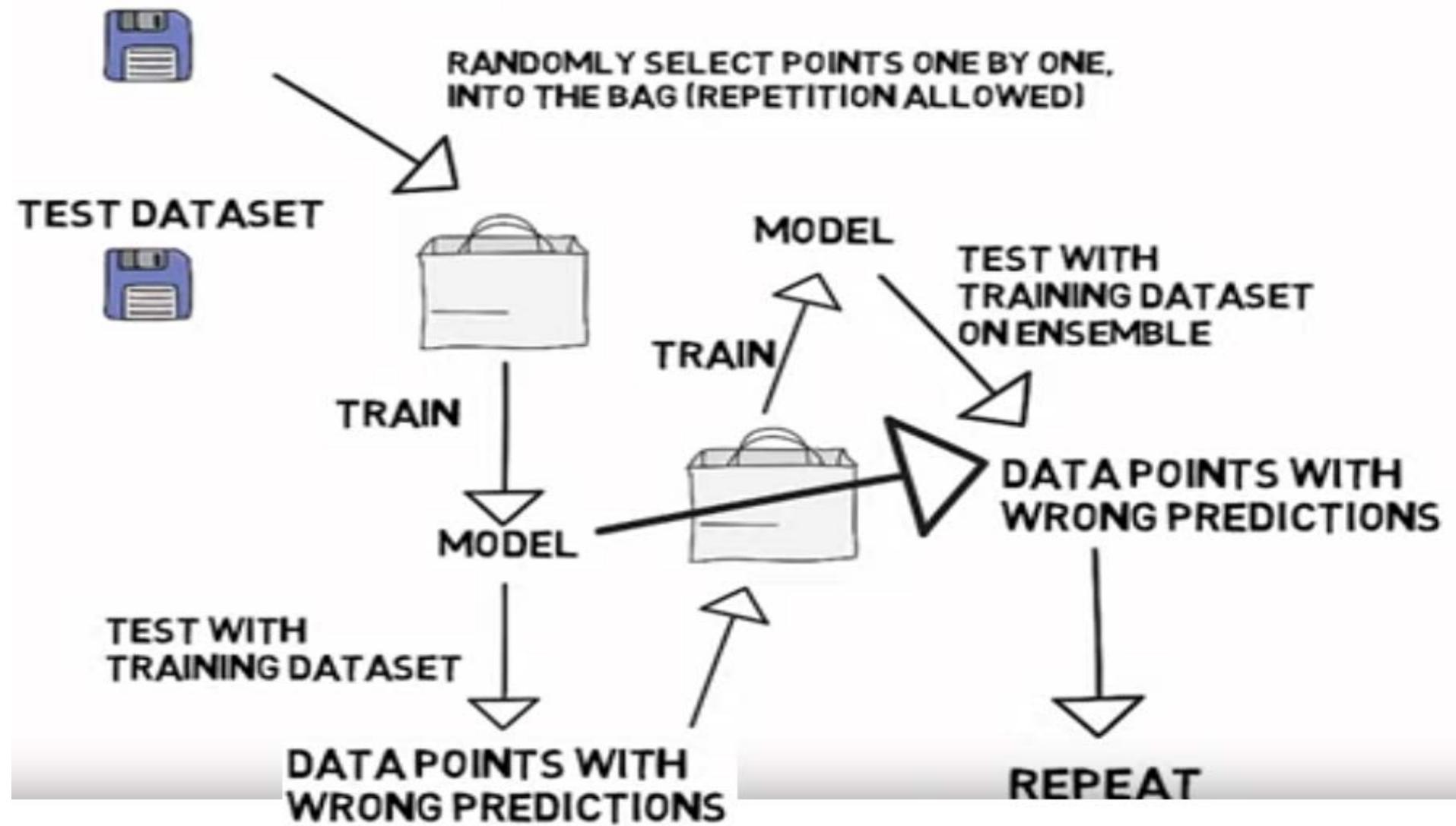
This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

```
bg=BaggingClassifier( DecisionTreeClassifier(), max_samples=0.5, max_features=1.0, n_estimators=20)
bg.fit(x_train,y_train)
bg.score(x_test,y_test)
bg.score(x_train,y_train)
```

<https://www.youtube.com/watch?v=m-S9Hojj1as&pbjreload=10>

# Boosting

Can a set of weak learners create a single strong learner?

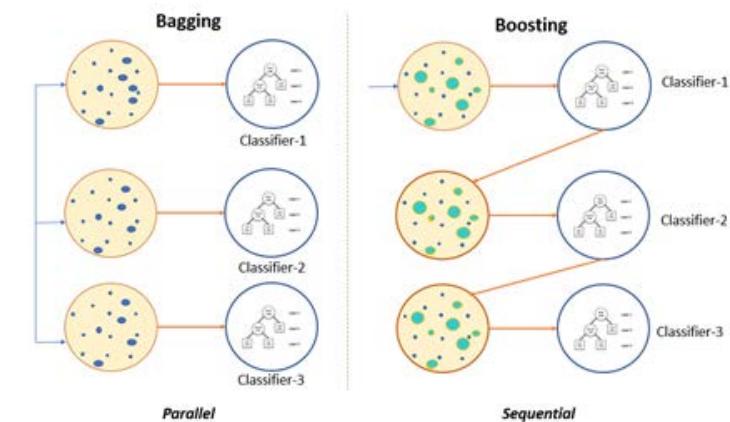


```
adb=AdaBoostClassifier( DecisionTreeClassifier(), n_estimators=10, learning_rate=1)
adb.fit(x_train,y_train)
adb.score(x_test,y_test)
adb.score(x_train,y_train)
```

# Bagging vs Boosting

Bagging and Boosting decrease the variance of your single estimate as they combine several estimates from different models. So the result may be a model with **higher stability**.

Bagging	Boosting
<p>Both the ensemble methods get N learners from 1 learner. But..</p>	
..follows parallel ensemble techniques, i.e. base learners are formed independently.	..follows Sequential ensemble technique, i.e. base learners are dependent on the previous weak base learner.
<p>Random sampling with replacement.</p>	
<p>Both gives out the final prediction by taking average of N learners. But..</p>	
..equal weights is given to all model. (equally weighted average)	..more weight is given to the model with better performance. (weighted average)
<p>Both are good at providing high model scalability. But..</p>	
..it reduces variance and solves the problem of overfitting.	..it reduces the bias but is more prone to overfitting. Overfitting can be avoided by tuning the parameters.





# XGBoost Battle-tested

It has gained much popularity and attention recently as the algorithm of choice for many winning teams of machine learning competitions.



```
# First XGBoost model for Pima Indians dataset
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

**XGBClassifier** and **XGBRegressor**

**import xgboost as xgb**

```
xgb=xgb.XGBRegressor(max_depth=3,learning_rate=0.1,n_estimators=1000,
reg_alpha=0.001,reg_lambda=0.000001,n_jobs=-1,min_child_weight=3)
xgb.fit(xtrain,ytrain)
```

<https://machinelearningmastery.com/xgboost-python-mini-course/>

<https://github.com/dmlc/xgboost/tree/master/demo>

<https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/>

<https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/>

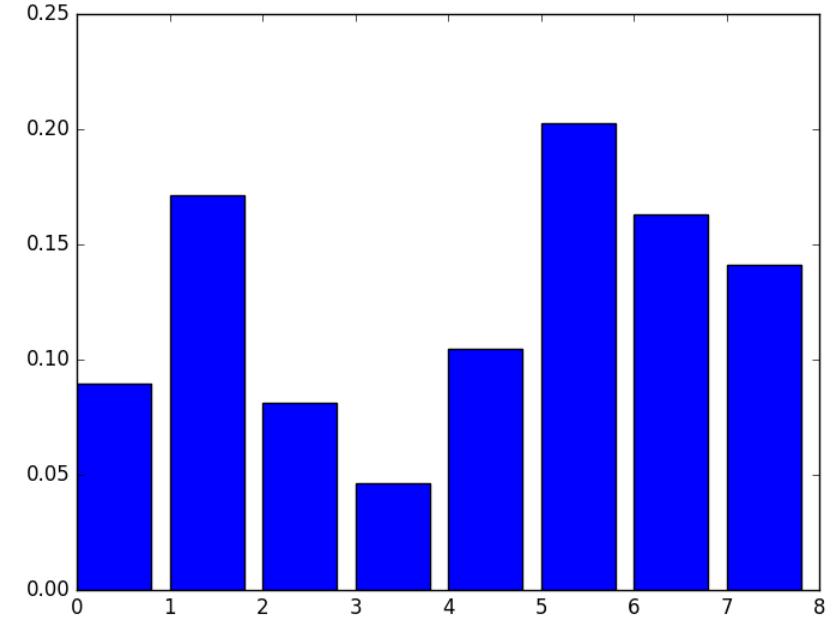
[https://www.ibm.com/developerworks/community/blogs/jfp/entry/Installing\\_XGBoost\\_For\\_Anaconda\\_on\\_Windows?lang=en](https://www.ibm.com/developerworks/community/blogs/jfp/entry/Installing_XGBoost_For_Anaconda_on_Windows?lang=en)

<https://en.wikipedia.org/wiki/XGBoost>



# Feature importance

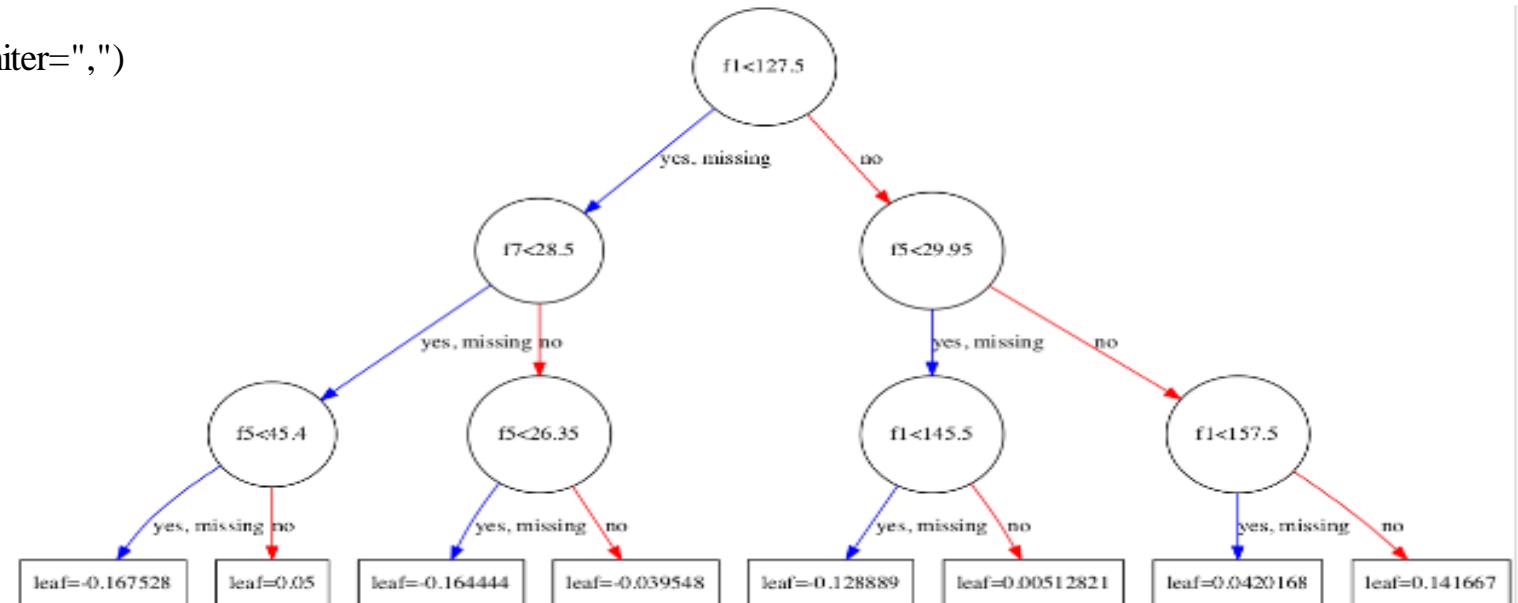
```
# plot feature importance manually
from numpy import loadtxt
from xgboost import XGBClassifier
from matplotlib import pyplot
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split data into X and y
X = dataset[:,0:8]
y = dataset[:,8]
# fit model no training data
model = XGBClassifier()
model.fit(X, y)
# feature importance
print(model.feature_importances_)
# plot
pyplot.bar(range(len(model.feature_importances_ )), model.feature_importances_ )
pyplot.show()
```





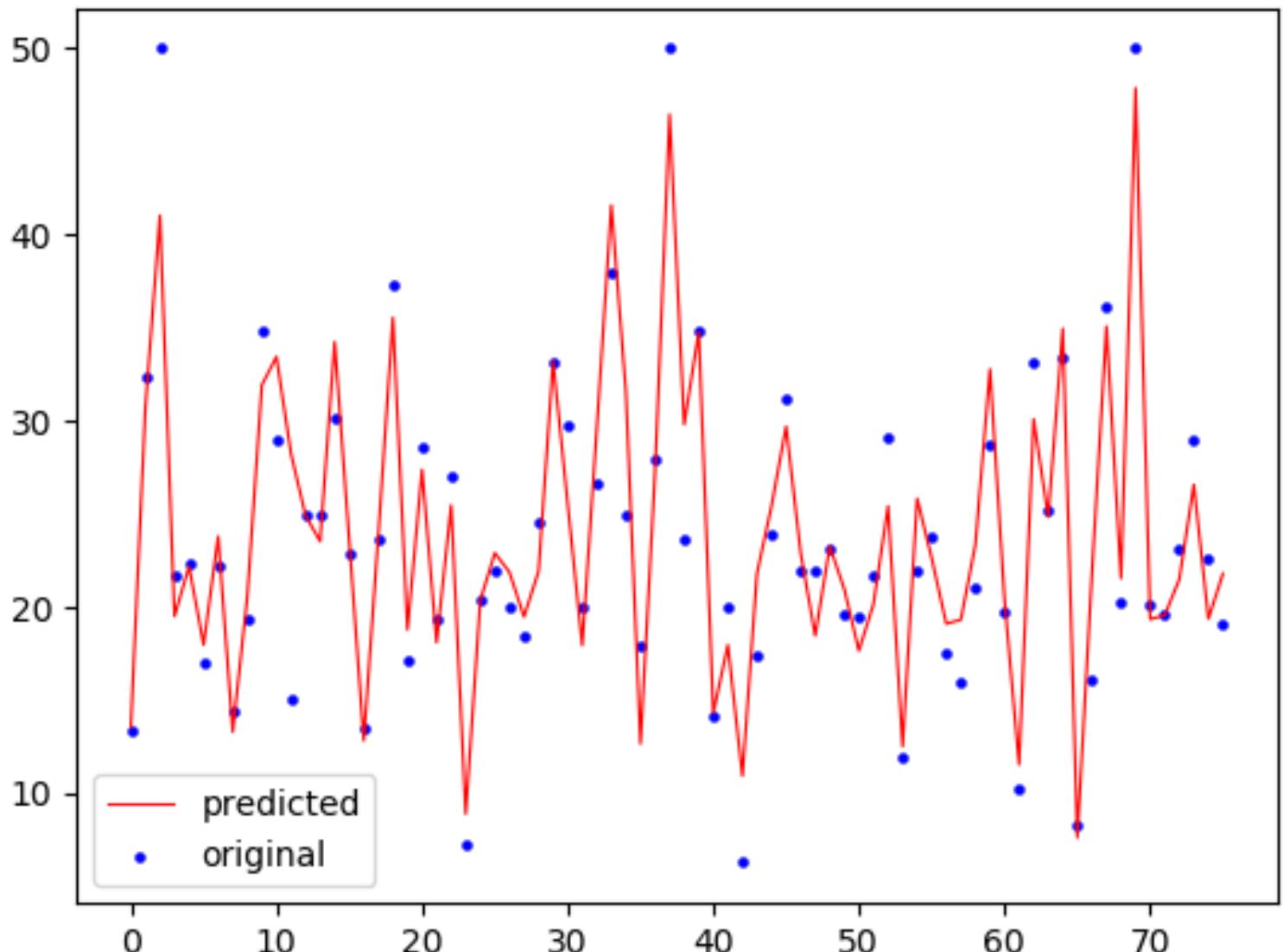
# Tree

```
# plot decision tree
from numpy import loadtxt
from xgboost import XGBClassifier
from xgboost import plot_tree
import matplotlib.pyplot as plt
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split data into X and y
X = dataset[:,0:8]
y = dataset[:,8]
# fit model no training data
model = XGBClassifier()
model.fit(X, y)
# plot single tree
plot_tree(model)
plt.show()
```



# Regression

```
import xgboost as xgb
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np
boston = load_boston()
x, y = boston.data, boston.target
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)
xgbr = xgb.XGBRegressor()
print(xgbr)
xgbr.fit(xtrain, ytrain)
# - cross validation
scores = cross_val_score(xgbr, xtrain, ytrain, cv=5)
print("Mean cross-validation score: %.2f" % scores.mean())
kfold = KFold(n_splits=10, shuffle=True)
kf_cv_scores = cross_val_score(xgbr, xtrain, ytrain, cv=kfold)
print("K-fold CV average score: %.2f" % kf_cv_scores.mean())
y_pred = xgbr.predict(xtest)
mse = mean_squared_error(ytest, y_pred)
print("MSE: %.2f" % mse)
print("RMSE: %.2f" % np.sqrt(mse))
x_ax = range(len(ytest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, y_pred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```





# XGBoost model save/load

```
# Train XGBoost model, save to file using pickle, load and make predictions
from sklearn.model_selection import train_test_split
from numpy import loadtxt
import xgboost
import pickle
from sklearn import model_selection
from sklearn.metrics import accuracy_score
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
# fit model no training data
model = xgboost.XGBClassifier()
model.fit(X_train, y_train)
# save model to file
pickle.dump(model, open("pima.pickle.dat", "wb"))

# some time later...

# load model from file
loaded_model = pickle.load(open("pima.pickle.dat", "rb"))
# make predictions for test data
y_pred = loaded_model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```





# Hyper-parameter tuning

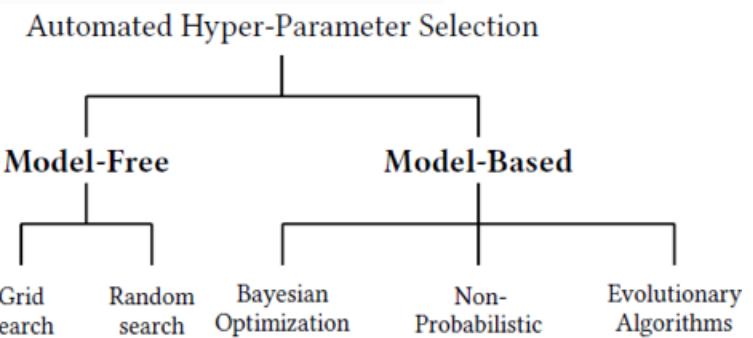
```

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
pipe_svc = Pipeline([('scl', StandardScaler()), ('clf', SVC(random_state=1))])
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [ {'clf__C': param_range, 'clf__kernel': ['linear']},
{'clf__C': param_range, 'clf__gamma': param_range, 'clf__kernel': ['rbf']}]
gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid, scoring='accuracy', cv=10, n_jobs=1)
gs = gs.fit(X, y)

print(gs.best_score_)
print(gs.best_params_)

c=gs.best_estimator_
c.fit(x_train,y_train)
pred=c.predict(x_test)
print(accuracy_score(y_test,pred))

```



$$\begin{aligned}
\text{CVLoss}(\mathbf{h}) &:= \text{Loss}(\mathbf{X}_v, \mathbf{Y}_v; \mathbf{a}^*(\mathbf{h}), \mathbf{h}), \quad \text{where} \\
\mathbf{a}^*(\mathbf{h}) &:= \operatorname{argmin}_{\mathbf{a}} \{ \text{Loss}(\mathbf{X}_t, \mathbf{Y}_t, \mathbf{a}, \mathbf{h}) \}
\end{aligned}$$

[https://en.wikipedia.org/wiki/Hyperparameter\\_optimization](https://en.wikipedia.org/wiki/Hyperparameter_optimization)

<https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/>

<https://mlfromscratch.com/gridsearch-keras-sklearn/#/>

<https://www.kaggle.com/cesartrevisan/scikit-learn-and-gridsearchcv>

<https://github.com/dmlc/xgboost/tree/master/demo/guide-python>

<https://www.youtube.com/watch?v=R5eJgjn9bnw>



# Hyper-parameter tuning

```
def algorithm_pipeline(X_train, X_test, y_train, y_test, model, param_grid, cv=10, scoring_fit='neg_mean_squared_error', do_prob = False):
    gs = GridSearchCV( estimator=model, param_grid=param_grid, cv=cv, n_jobs=-1, scoring=scoring_fit, verbose=2 )
    fitted_model = gs.fit(X_train, y_train)
    if do_prob:
        pred = fitted_model.predict_proba(X_test)
    else:
        pred = fitted_model.predict(X_test)
    return fitted_model, pred
```

```
param_grid = { 'epochs':[1,2,3], 'batch_size':[128]
    #'epochs' : [100,150,200],
    #'batch_size' : [32, 128],
    #'optimizer' : ['Adam', 'Nadam'],
    #'dropout_rate' : [0.2, 0.3],
    #'activation' : ['relu', 'elu']
    }
model = KerasClassifier(build_fn = build_cnn, verbose=0)
model, pred = algorithm_pipeline(X_train, X_test, y_train, y_test, model, param_grid, cv=5, scoring_fit='neg_log_loss')

print(model.best_score_)
print(model.best_params_)
```

```
model = xgb.XGBRegressor()
param_grid = { 'n_estimators': [400, 700, 1000], 'colsample_bytree': [0.7, 0.8], 'max_depth': [15,20,25], 'reg_alpha': [1.1, 1.2, 1.3],
    'reg_lambda': [1.1, 1.2, 1.3], 'subsample': [0.7, 0.8, 0.9] }
model, pred = algorithm_pipeline(X_train, X_test, y_train, y_test, model, param_grid, cv=5)
# Root Mean Squared Error
print(np.sqrt(-model.best_score_))
print(model.best_params_)
```

<https://mlfromscratch.com/gridsearch-keras-sklearn/#/>

<https://www.kaggle.com/cesartrevisan/scikit-learn-and-gridsearchcv>

<https://github.com/dmlc/xgboost/tree/master/demo/guide-python>

<https://www.youtube.com/watch?v=R5eJgjn9bnw>



# Hyper-parameter tuning

## GBDT Hyper Parameter Tuning

Hyper Parameter	Tuning Approach	Range	Note
# of Trees	Fixed value	100-1000	Depending on datasize
Learning Rate	Fixed => Fine Tune	[2 - 10] / # of Trees	Depending on # trees
Row Sampling	Grid Search	[.5, .75, 1.0]	
Column Sampling	Grid Search	[.4, .6, .8, 1.0]	
Min Leaf Weight	Fixed => Fine Tune	3/(% of rare events)	Rule of thumb
Max Tree Depth	Grid Search	[4, 6, 8, 10]	
Min Split Gain	Fixed	0	Keep it 0

Best GBDT implementation today: <https://github.com/tqchen/xgboost>

by Tianqi Chen (U of Washington)



## Tuning XGBoost

- **eta (learning rate) + num\_round (number of trees)**
  - Examine objective metric in training/validation to quickly find good configuration
  - Target around 100 trees
- **max\_depth (start with 6) -- This is different from R GBM**
- **min\_child\_weight (start with 1/sqrt(event rate))**
- **colsample\_bytree (.3-.5)**
- **subsample (leave at 1.0)**
- **gamma (usually is it OK to leave at 0.0)**

Xgboost	<ul style="list-style-type: none"> <li>• Eta</li> <li>• Gamma</li> <li>• Max_depth</li> <li>• Min_child_weight</li> <li>• Subsample</li> <li>• Colsample_bytree</li> <li>• Lambda</li> <li>• alpha</li> </ul>	<ul style="list-style-type: none"> <li>• 0.01,0.015, 0.025, 0.05, 0.1</li> <li>• 0.05-0.1,0.3,0.5,0.7,0.9,1.0</li> <li>• 3, 5, 7, 9, 12, 15, 17, 25</li> <li>• 1, 3, 5, 7</li> <li>• 0.6, 0.7, 0.8, 0.9, 1.0</li> <li>• 0.6, 0.7, 0.8, 0.9, 1.0</li> <li>• 0.01-0.1, 1.0 , RS*</li> <li>• 0, 0.1, 0.5, 1.0 RS*</li> </ul>
---------	---	---

<https://machinelearningmastery.com/configure-gradient-boosting-algorithm/>

[https://github.com/aarshayj/Analytics\\_Vidhya/blob/master/Articles/Parameter\\_Tuning\\_XGBoost\\_with\\_Example/XGBoost%20models.ipynb](https://github.com/aarshayj/Analytics_Vidhya/blob/master/Articles/Parameter_Tuning_XGBoost_with_Example/XGBoost%20models.ipynb)

<https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/>

<https://machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/>

<https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/>

<https://statklee.github.io/model/model-python-xgboost-hyper.html>



# Hyper-parameter tuning

```
# XGBoost on Otto dataset, Tune n_estimators
from pandas import read_csv
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
# split data into X and y
X = dataset[:,0:8]
y = dataset[:,8]
# grid search
model = XGBClassifier()
n_estimators = range(10, 400, 20)
param_grid = dict(n_estimators=n_estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
# plot
pyplot.errorbar(n_estimators, means, yerr=stds)
pyplot.title("XGBoost n_estimators vs Log Loss")
pyplot.xlabel('n_estimators')
pyplot.ylabel('Log Loss')
pyplot.savefig('n_estimators.eps')
```

by preserving the percentage of samples for each class.



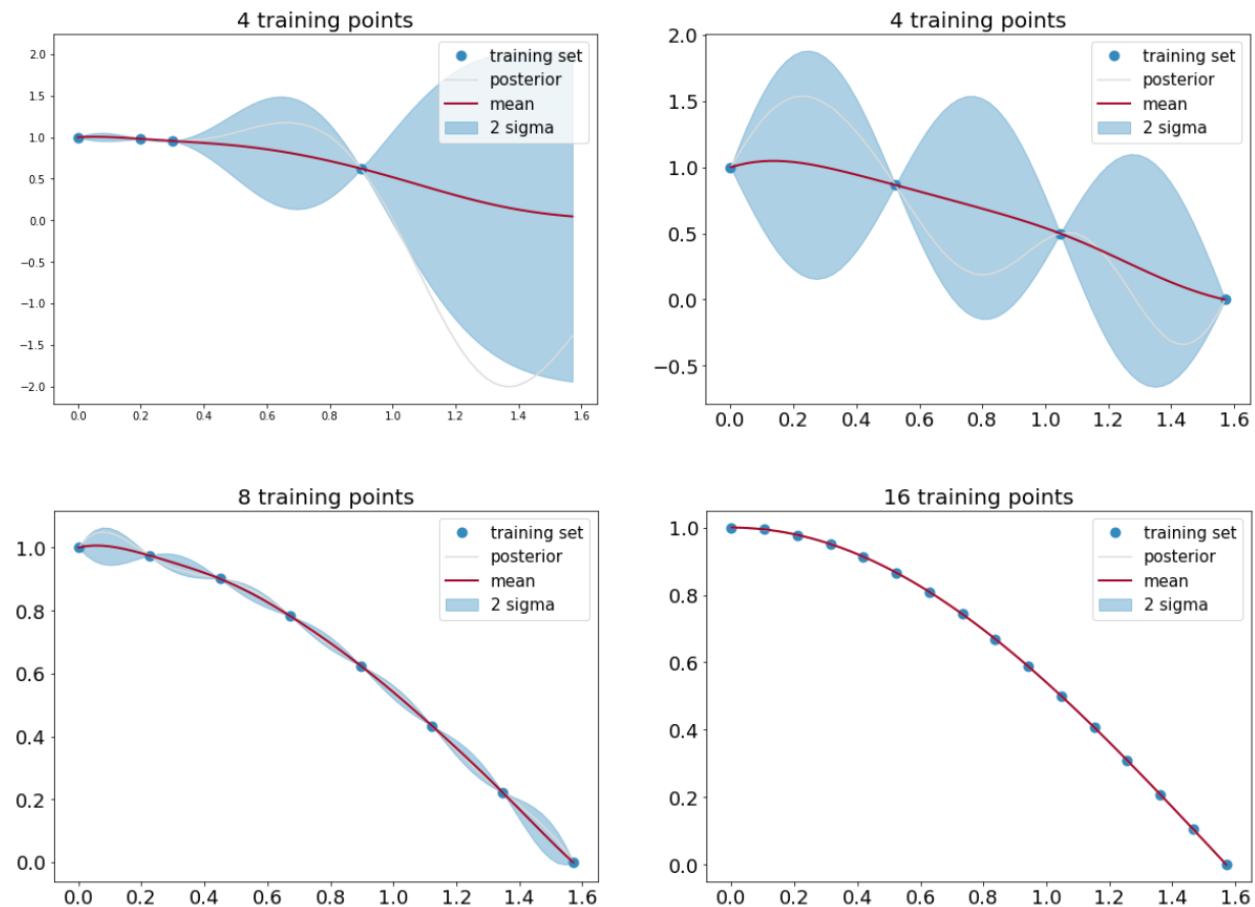
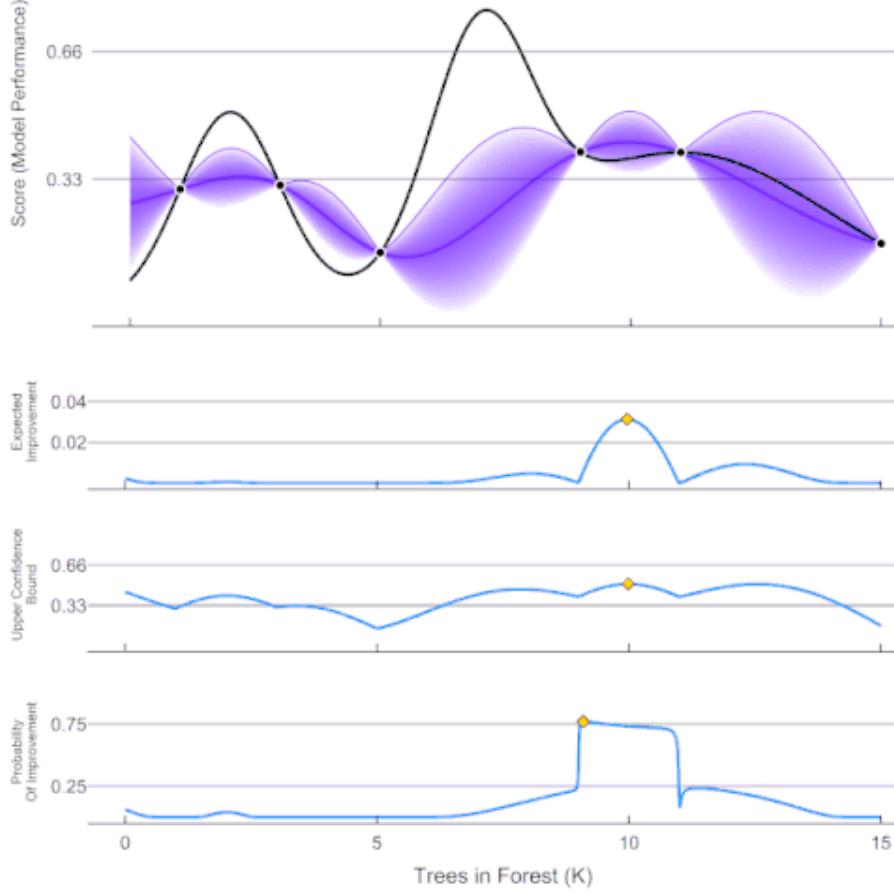
# Hyper-parameter tuning

```
if True:  
    test0=-1.  
    for i in (0.01, 0.05, 1.0):  
        for j in (100, 500, 1000):  
            for k in (3, 5, 10):  
                for l in (1.0, 1.0):  
                    for m in (1.0, 1.0):  
                        for n in (0, 2, 5):  
                            clf = XGBClassifier( max_depth=k, learning_rate=i, n_estimators=j, subsample=l, colsample_bytree=m, gamma=n)  
                            clf.fit(X,y)  
                            test=np.mean(cross_val_score(clf, X, y, cv=3, scoring='roc_auc'))  
                            if test0 < test:  
                                print(i,j,k,l,m,n,test)  
                                test0=test  
                            pickle.dump(clf, open("pima.pickle.dat", "wb"))  
  
# some time later...  
# load model from file  
loaded_model = pickle.load(open("pima.pickle.dat", "rb"))  
# make predictions for test data  
y_pred = loaded_model.predict(X)  
predictions = [round(value) for value in y_pred]  
# evaluate predictions  
accuracy = accuracy_score(y, predictions)  
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```



# Gaussian Processes regression: basic introductory example

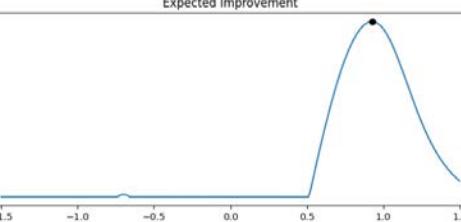
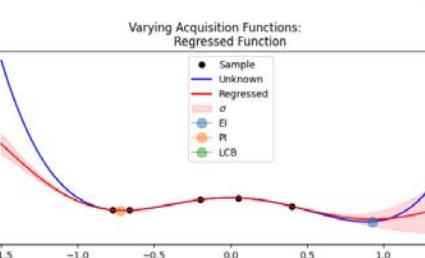
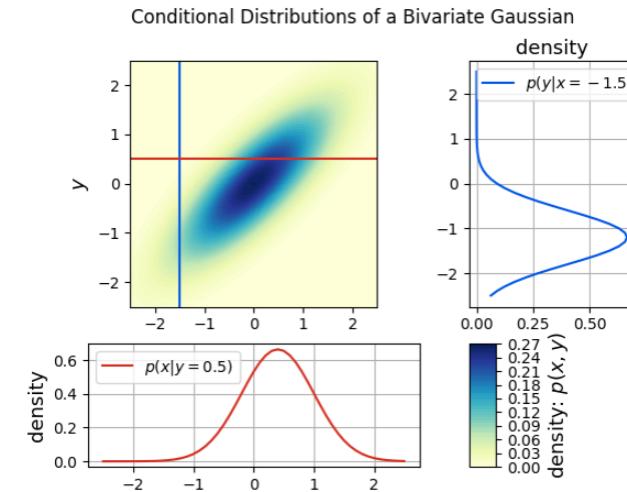
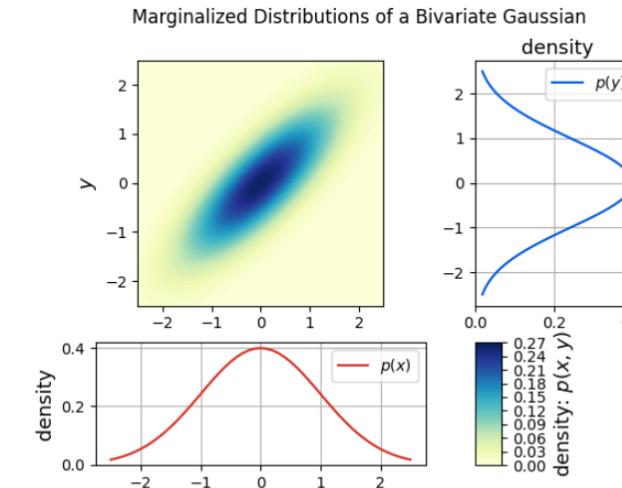
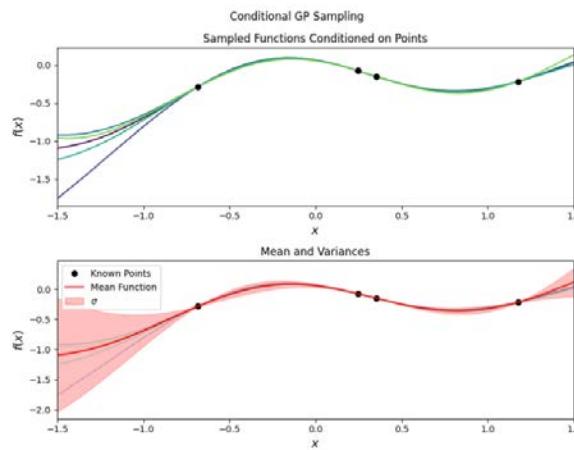
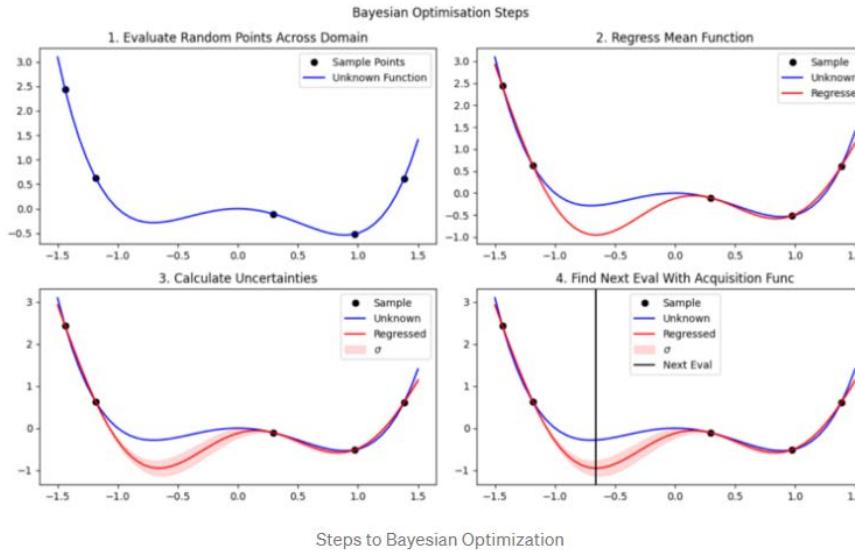
ParBayesianOptimization in Action (Round 1)



The figures illustrate the interpolating property of the Gaussian Process model as well as its probabilistic nature in the form of a pointwise 95% confidence interval.



# Gaussian Processes regression: basic introductory example



$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K})$$

$$\begin{aligned} p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{y}) &= \int p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{f}) p(\mathbf{f}|\mathbf{X}, \mathbf{y}) d\mathbf{f} \\ &= \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \end{aligned}$$

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N} \left( \mathbf{0}, \begin{pmatrix} \mathbf{K}_y & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix} \right)$$

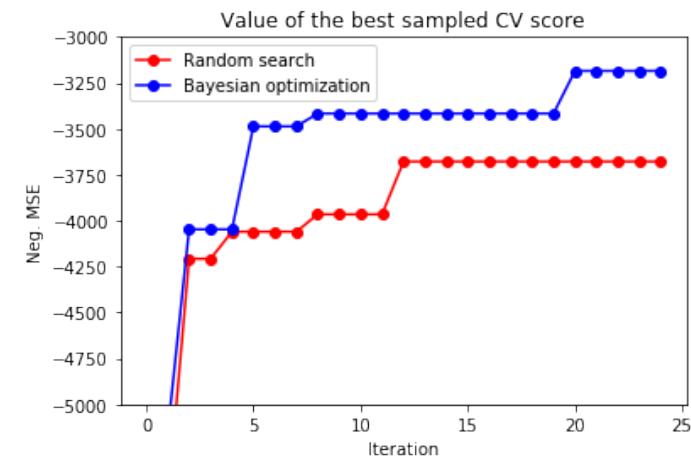
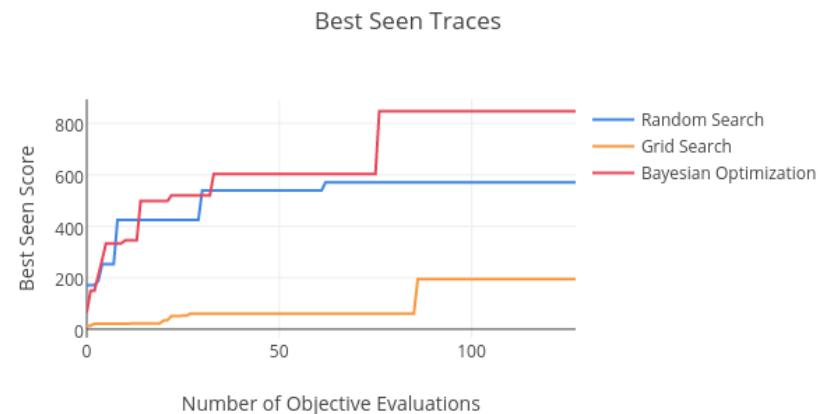
$$\boldsymbol{\mu}_* = \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{y}$$

$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_*$$



# Hyper-parameter tuning

```
import numpy as np
from numpy import loadtxt
from xgboost import XGBClassifier
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score
pbounds = {
    'learning_rate': (0.01, 1.0),
    'n_estimators': (100, 1000),
    'max_depth': (3,10),
    'subsample': (1.0, 1.0), # Change for big datasets
    'colsample': (1.0, 1.0), # Change for datasets with lots of features
    'gamma': (0, 5)
}
def xgboost_hyper_param(learning_rate, n_estimators, max_depth, subsample, colsample, gamma):
    dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
    X = dataset[:,0:8]
    y = dataset[:,8]
    max_depth = int(max_depth)
    n_estimators = int(n_estimators)
    clf = XGBClassifier( max_depth=max_depth, learning_rate=learning_rate, n_estimators=n_estimators,
        subsample=subsample, colsample=colsample, gamma=gamma)
    return np.mean(cross_val_score(clf, X, y, cv=3, scoring='roc_auc'))
optimizer = BayesianOptimization( f=xgboost_hyper_param, pbounds=pbounds, random_state=1)
optimizer.maximize(init_points=3, n_iter=24, acq='ei', xi=0.01)
```





# Hyper-parameter tuning

OPTUNA

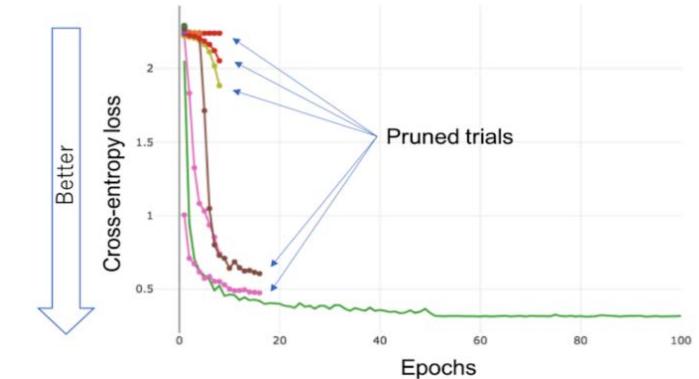
<https://optuna.org/>

```
!pip install optuna
import optuna
```

Optuna is framework agnostic. You can use it with any machine learning or deep learning framework.

[Quick Start](#) [PyTorch](#) [Chainer](#) [TensorFlow](#) [Keras](#) [MXNet](#) [Scikit-Learn](#) [XGBoost](#) [LightGBM](#) [Other](#)

```
def objective(trial,data=data,target=target):
    train_x, test_x, train_y, test_y = train_test_split(data, target, test_size=0.15,random_state=42)
    param = { 'tree_method':'gpu_hist', #this parameter means using the GPU when training our model to speedup the training process
              'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),
              'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0),
              'colsample_bytree': trial.suggest_categorical('colsample_bytree', [0.3,0.4,0.5,0.6,0.7,0.8,0.9, 1.0]),
              'subsample': trial.suggest_categorical('subsample', [0.4,0.5,0.6,0.7,0.8,1.0]),
              'learning_rate': trial.suggest_categorical('learning_rate', [0.008,0.009,0.01,0.012,0.014,0.016,0.018, 0.02]),
              'n_estimators': 4000, 'max_depth': trial.suggest_categorical('max_depth', [5,7,9,11,13,15,17,20]),
              'random_state': trial.suggest_categorical('random_state', [24, 48,2020]), 'min_child_weight': trial.suggest_int('min_child_weight', 1, 300), }
    model = xgb.XGBRegressor(**param)
    model.fit(train_x,train_y,eval_set=[(test_x,test_y)],early_stopping_rounds=100,verbose=False)
    preds = model.predict(test_x)
    rmse = mean_squared_error(test_y, preds,squared=False)
    return rmse
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)
print('Number of finished trials:', len(study.trials))
print('Best trial:', study.best_trial.params)
study.trials_dataframe()
optuna.visualization.plot_optimization_history(study)
optuna.visualization.plot_parallel_coordinate(study)
optuna.visualization.plot_slice(study)
optuna.visualization.plot_contour(study, params=['alpha', #'max_depth', 'lambda', 'subsample', 'learning_rate', 'subsample'])
optuna.visualization.plot_param_importances(study)
optuna.visualization.plot_edf(study)
```





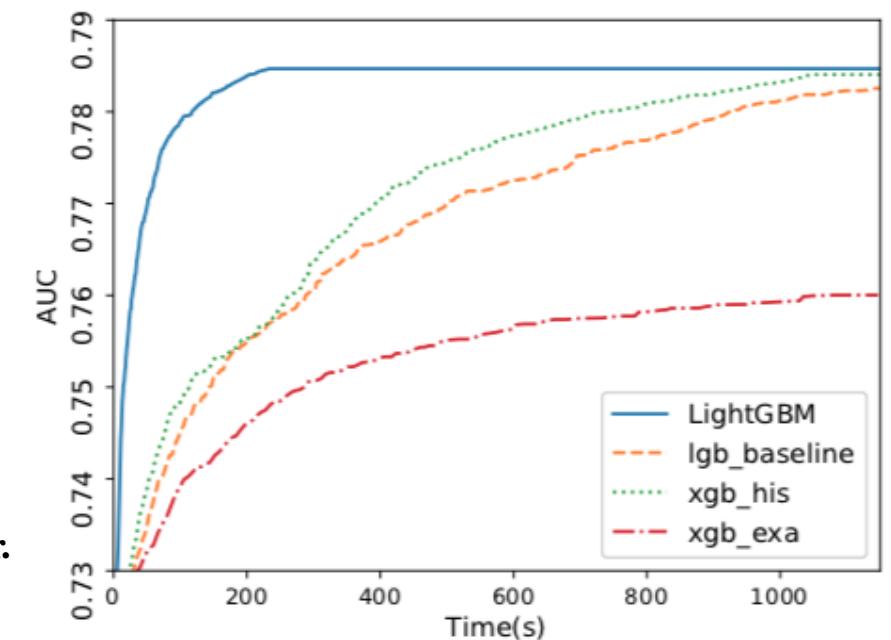
# LightGBM Better accuracy

## Advantages of LightGBM (2017)

1. **Faster training speed and higher efficiency:** Light GBM use histogram based algorithm i.e it buckets continuous feature values into discrete bins which fasten the training procedure.
2. **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.
3. **Better accuracy than any other boosting algorithm:** It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max\_depth parameter.
4. **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.
5. **Parallel learning supported.**

LightGBM vs XGBoost

**Catboost improves over LightGBM by handling categorical features better.**



<https://medium.com/kaggle-nyc/gradient-boosting-decision-trees-xgboost-vs-lightgbm-and-catboost-72df6979e0bb>

<https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/>

<https://lightgbm.readthedocs.io/en/latest/Experiments.html>

<https://github.com/Microsoft/LightGBM/blob/master/docs/Experiments.rst#comparison-experiment>



1	0.644	0.247	-0.447	0.862	0.374	0.854	-1.126	-0.798	2.173	1.015	-0.201	1.408	0.800	1.575	1.807	1.607	0.800	1.585	-0.198	-0.744	3.102	0.958	1.061	0.988	0.875	0.581	0.985	0.796	
2	0.385	1.800	1.037	1.004	0.349	1.502	-0.966	1.734	0.800	0.966	-1.968	-0.249	0.800	1.501	0.665	-0.354	2.548	0.834	-0.448	0.638	3.102	0.695	0.989	0.981	0.803	0.813	1.149	1.116	
3	1.214	-0.166	0.004	0.505	1.434	0.628	-1.174	-1.230	1.087	0.579	-1.047	-0.118	0.000	0.835	0.340	1.234	2.548	0.711	-1.383	1.355	0.000	0.848	0.911	1.043	0.931	1.058	0.744	0.696	
4	0.420	1.111	0.137	1.516	-1.657	0.854	0.623	1.605	1.087	1.511	-1.297	0.251	0.000	0.872	-0.368	-0.721	0.000	0.543	0.731	1.424	3.102	1.597	1.282	1.105	0.730	0.148	1.231	1.234	
5	0.897	-1.703	-1.306	1.022	-0.729	0.836	0.859	-0.333	2.173	1.336	-0.965	0.972	2.215	0.671	1.021	-1.439	0.000	0.493	-2.019	-0.289	0.000	0.805	0.930	0.984	1.438	2.198	1.934	1.684	
6	0.756	1.126	-0.945	2.355	-0.555	0.889	0.880	1.440	0.000	0.585	0.271	0.631	2.215	0.722	1.744	1.051	0.000	0.618	0.924	0.699	1.551	0.976	0.864	0.988	0.803	0.234	0.822	0.911	
7	1.141	-0.741	0.953	1.478	-0.524	1.197	-0.871	1.689	2.173	0.875	1.321	-0.518	1.107	0.548	0.037	-0.987	0.000	0.879	1.187	0.245	0.000	0.888	0.701	1.747	1.358	2.479	1.491	1.223	
8	0.606	0.603	-0.936	-0.384	1.257	-1.162	2.719	-0.600	0.100	2.173	3.303	-0.284	1.561	1.107	0.689	1.786	-0.326	0.000	0.788	-0.532	1.216	0.000	0.936	2.022	0.985	1.574	4.323	2.263	1.742
9	0.603	0.429	-0.279	1.448	1.381	1.088	2.423	-1.295	0.000	0.452	1.305	0.533	0.800	1.076	1.011	1.256	2.548	2.021	1.260	-0.343	0.000	0.899	0.969	1.281	0.763	0.652	0.827	0.785	
10	1.171	-0.962	0.521	0.841	-0.315	1.196	-0.744	-0.882	2.173	0.726	-1.305	1.377	1.107	0.643	-1.790	-1.264	0.000	1.257	0.222	0.817	0.000	0.862	0.911	0.987	0.846	1.293	0.899	0.756	
11	1.392	-0.358	0.235	1.494	-0.461	0.895	-0.848	1.549	2.173	0.841	-0.384	0.666	1.107	1.199	2.509	-0.891	0.000	1.109	-0.364	-0.945	0.000	0.693	2.135	1.178	1.362	0.959	2.056	1.842	
12	1.024	1.076	-0.886	0.851	1.530	0.673	-0.449	0.187	1.087	0.628	-0.895	1.176	2.215	0.696	-0.232	-0.875	0.000	0.411	1.501	0.048	0.000	0.842	0.919	1.063	1.193	0.777	0.964	0.807	
13	0.898	0.898	-0.760	1.182	1.369	0.751	0.696	-0.959	-0.710	1.087	0.775	-0.130	-1.409	2.215	0.701	-0.110	-0.739	0.000	0.508	-0.451	0.390	0.000	0.762	0.738	0.998	1.126	0.788	0.940	0.790
14	0.460	0.537	0.636	1.442	-0.269	0.585	0.323	-1.731	2.173	0.503	1.034	-0.927	0.000	0.928	-1.024	1.000	2.548	0.513	-0.618	-1.336	0.000	0.802	0.831	0.992	1.019	0.925	1.050	0.883	
15	0.364	1.648	0.560	1.720	0.829	1.110	0.811	-0.588	0.000	0.408	1.045	1.054	2.215	0.319	-1.138	1.545	0.000	0.423	1.025	-1.265	3.102	1.656	0.928	1.003	0.544	0.327	0.670	0.746	
16	0.525	-0.096	1.206	0.948	-1.103	1.519	-0.582	0.606	2.173	1.274	-0.572	-0.934	0.000	0.855	-1.028	-1.222	0.000	0.578	-1.000	-1.725	3.102	0.896	0.878	0.981	0.498	0.909	0.772	0.668	
17	0.536	-0.821	-1.029	0.703	1.113	0.363	-0.711	0.022	1.087	0.325	1.503	1.249	2.215	0.673	1.041	-0.401	0.000	0.480	2.127	1.681	0.000	0.767	1.034	0.990	0.671	0.836	0.669	0.663	

```
>>> import pandas as pd
>>> from pandas import DataFrame
>>> import numpy as np

df_train = pd.read_csv('C:/testAI/regression/regression.train', header=None, sep='\t')
df_test = pd.read_csv('C:/testAI/regression/regression.test', header=None, sep='\t')
```

```
y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)
```

```
>>> aa=np.zeros((3,3))
```

```
>>> aa[0,0]=1
```

```
>>> aa[0,1]=2
```

```
>>> aa[0,2]=3
```

```
>>> aa[1,0]=4
```

```
>>> aa[1,1]=5
```

```
>>> aa[1,2]=6
```

```
>>> aa[2,0]=7
```

```
>>> aa[2,1]=8
```

```
>>> aa[2,2]=9
```

```
>>> aa
```

```
array([[1., 2., 3.,
```

```
        4., 5., 6.,
```

```
        7., 8., 9.]])
```

```
>>> a=pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]])
```

```
>>> a
```

```
0 1 2
```

```
0 1 2 3
```

```
1 4 5 6
```

```
2 7 8 9
```

```
>>> aa
```

```
array([[1., 2., 3.,
```

```
        4., 5., 6.,
```

```
        7., 8., 9.]])
```

```
>>> a[0]
```

```
0 1
```

```
1 4
```

```
2 7
```

```
Name: 0, dtype: int64
```

```
>>> a.drop(0, axis=1)
```

```
1 2
```

```
0 2 3
```

```
1 5 6
```

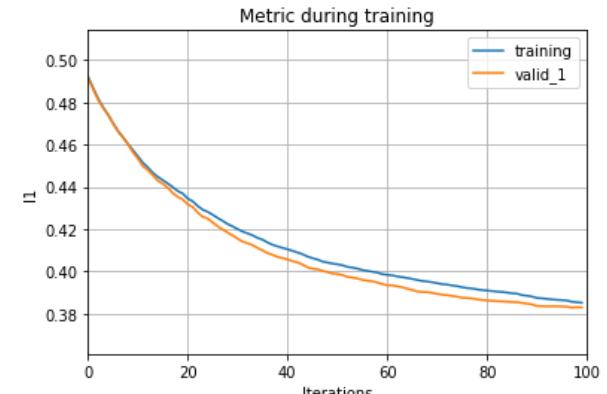
```
2 8 9
```



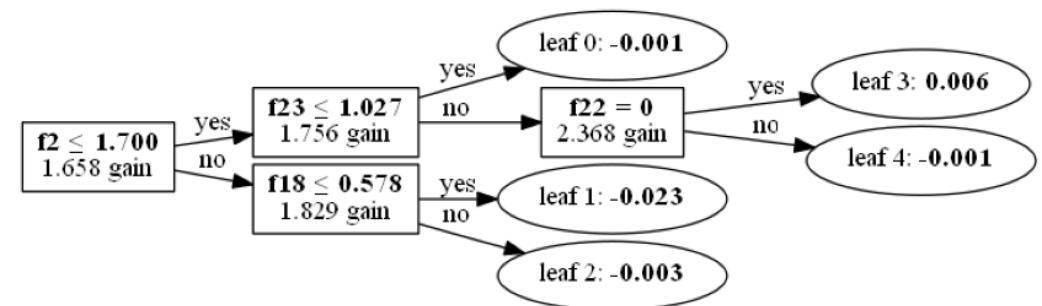
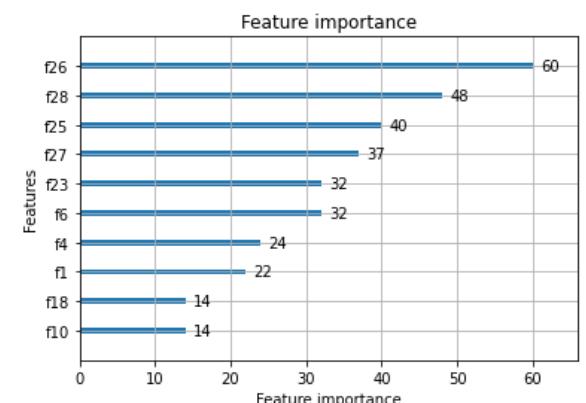
```

# coding: utf-8
import lightgbm as lgb
import pandas as pd
if lgb.compat.MATPLOTLIB_INSTALLED:
    import matplotlib.pyplot as plt
else:
    raise ImportError('You need to install matplotlib for plot_example.py.')
print('Loading data...')
#
df_train = pd.read_csv('C:/testAI/regression/regression.train', header=None, sep='\t')
df_test = pd.read_csv('C:/testAI/regression/regression.test', header=None, sep='\t')
y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)
# create dataset for lightgbm
lgb_train = lgb.Dataset(X_train, y_train)
lgb_test = lgb.Dataset(X_test, y_test, reference=lgb_train)
# specify your configurations as a dict
params = { 'num_leaves': 5, 'metric': ('l1', 'l2'), 'verbose': 0}
evals_result = {} # to record eval results for plotting
print('Starting training...')
# train
gbm = lgb.train(params, lgb_train, num_boost_round=100, valid_sets=[lgb_train, lgb_test],
                 feature_name=['f' + str(i + 1) for i in range(X_train.shape[-1])], categorical_feature=[21], evals_result=evals_result, verbose_eval=10)
print('Plotting metrics recorded during training...')
ax = lgb.plot_metric(evals_result, metric='l1')
plt.show()
print('Plotting feature importances...')
ax = lgb.plot_importance(gbm, max_num_features=10)
plt.show()
print('Plotting split value histogram...')
ax = lgb.plot_split_value_histogram(gbm, feature='f26', bins='auto')
plt.show()
print('Plotting 54th tree...') # one tree use categorical feature to split
ax = lgb.plot_tree(gbm, tree_index=53, figsize=(15, 15), show_info=['split_gain'])
plt.show()
print('Plotting 54th tree with graphviz...')
graph = lgb.create_tree_digraph(gbm, tree_index=53, name='Tree54')
graph.render(view=True)

```



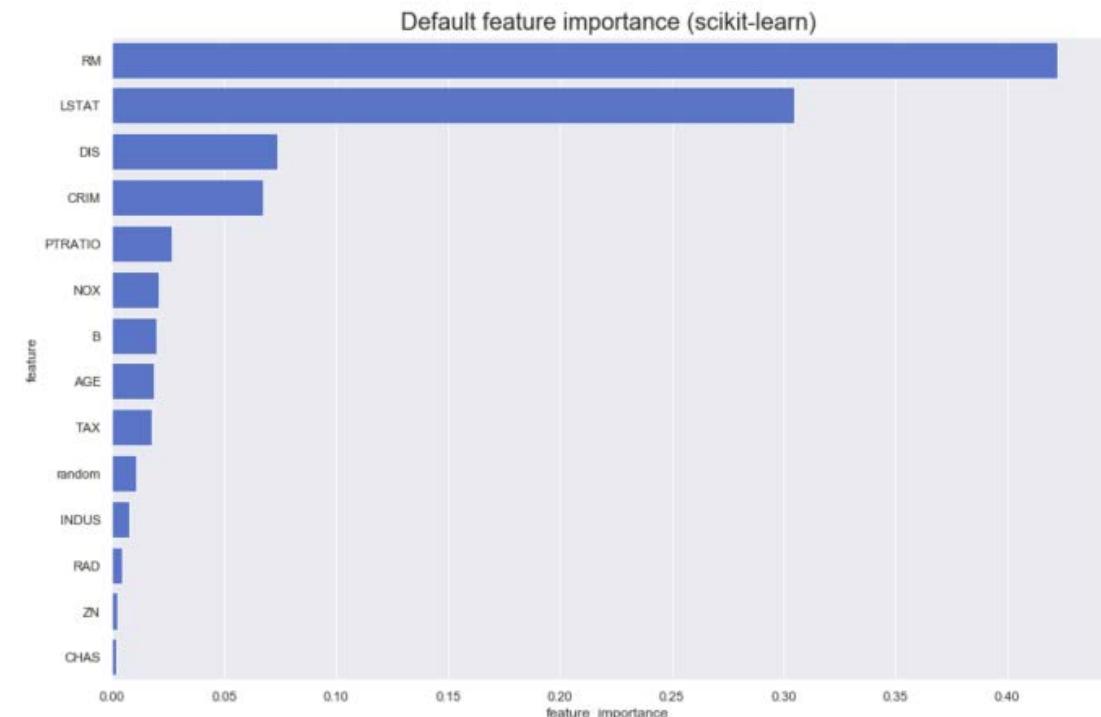
Plotting feature importances...





# Explaining Feature Importance

- **Default Scikit-learn's feature importance**
- **Permutation feature importance**
- **Drop Column feature importance**
- **Observation level feature importance**





# LightGBM

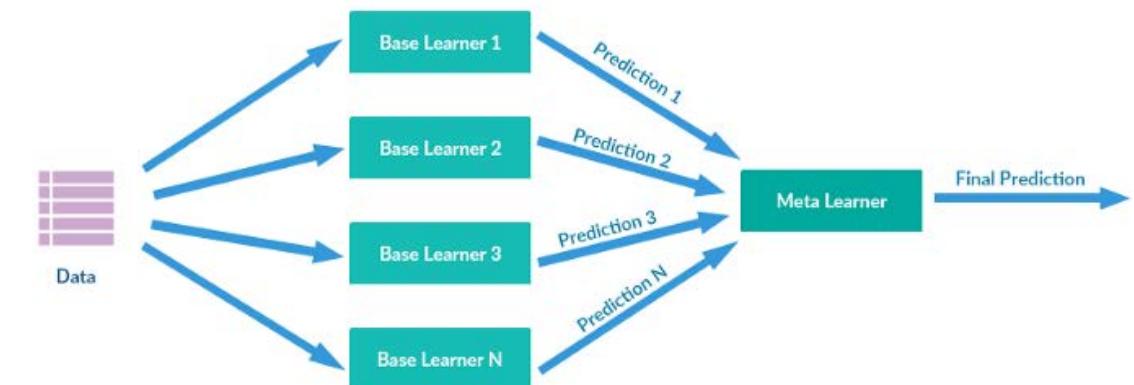
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import lightgbm as lgb
from numpy import loadtxt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,confusion_matrix
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)
# specify your configurations as a dict
params = { 'boosting_type': 'gbdt', 'objective': 'binary', 'metric': {'l2', 'l1'}, 'num_leaves':31, 'learning_rate': 0.05,
'feature_fraction':0.9, 'bagging_fraction': 0.8, 'bagging_freq': 5, 'verbose':0 }
print('Starting training...')
# train
gbm = lgb.train(params, lgb_train, num_boost_round=20, valid_sets=lgb_eval, early_stopping_rounds=5)
print('Saving model...')
# save model to file
gbm.save_model('model.txt')
print('Starting predicting...')
print(gbm.best_iteration)
# predict
y_pred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
y_pred = [round(value) for value in y_pred]
if True:
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print("Accuracy: ", accuracy)
    print("Confusion matrix:")
    print(cm)
```



# Stacking Ensemble

Don't select a model, combine them

```
# Voting Ensemble for Classification
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))
# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```



# Stacking Ensemble

Don't select a model, combine them

```
xtrain2 = pd.DataFrame(  
    {'XGB': xgb.predict(xtrain),  
     'NN': dl.predict(xtrain),  
     'LGB': lgb.predict(xtrain)}  
)  
xtrain2.head()  
from sklearn import linear_model  
reg = linear_model.LinearRegression()  
reg.fit(xtrain2, ytrain)
```

---

```
from sklearn.tree import DecisionTreeRegressor as DTR  
from sklearn.ensemble import StackingRegressor as SR  
model_sc = SR(estimators = [ ('xgb', model), ('lgb', model_lgb) ], final_estimator= DTR(random_state=17, max_depth=4, criterion= "mse"), cv = 8 )  
model_sc.fit(X_train, y_train)  
pred = model_sc.predict(X_test)  
print(regression_error(y_test, pred ))
```



# Stacking Ensemble

Don't select a model, combine them

```
## parameters derived from Bayesian Optimizaion fine-tuning
param_lgbm = {'bagging_fraction': 0.973905385549851, 'feature_fraction': 0.2945585590881137, 'learning_rate': 0.03750332268701348, 'max_depth': int(7.66), 'min_child_weight': int(41.36), 'min_split_gain': 0.04033836353603582, 'num_leaves': int(46.42), 'application': 'regression', 'num_iterations': 5000, 'metric': 'rmse'}
param_cat = {'bagging_temperature': 0.31768713094131684, 'depth': int(8.03), 'l2_leaf_reg': 1.3525686450404295, 'learning_rate': 0.18, 'iterations': 150, 'loss_function': 'RMSE', 'verbose': False}
param_xgb = {'colsample_bytree': 0.8119098377889549, 'gamma': 2.244423418642122, 'learning_rate': 0.015800631696721114, 'max_depth': int(9.846), 'min_child_weight': int(15.664), 'subsample': 0.82345, 'objective': 'reg:squarederror', 'eval_metric': 'rmse', 'num_boost_round': 500}
```

```
from sklearn.ensemble import StackingRegressor
from xgboost import XGBRegressor
```

```
estimators = [
    ('lgbm', lightgbm.LGBMRegressor(**param_lgbm, random_state=7, n_jobs=-1)),
    ('xgbr', XGBRegressor(**param_xgb, random_state=7, nthread=-1)),
    ('cat', CatBoostRegressor(**param_cat))]
reg=StackingRegressor(estimators=estimators,final_estimator=lightgbm.LGBMRegressor(),n_jobs=-1,cv=5)
train_X, val_X, train_Y, val_Y=train_test_split(train_feature,train['target'],test_size=0.2,shuffle=True)
reg.fit(train_X,train_Y)
val_pred = reg.predict(val_X)
score = np.sqrt(mean_squared_error(val_Y, val_pred))
print("Final model RMSE: ",end = "")
print(score)
```

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import StackingRegressor

final_estimator = GradientBoostingRegressor(n_estimators=200,random_state=42)

estimators = [('xgb', XGBRegressor(tree_method='gpu_hist', **XGB_params)),
    ('lgb', lgbm.LGBMRegressor(device_type='cpu', **lgb_params)),
    ('cat', CatBoostRegressor(verbose=0, task_type='CPU', **catboost_params))]

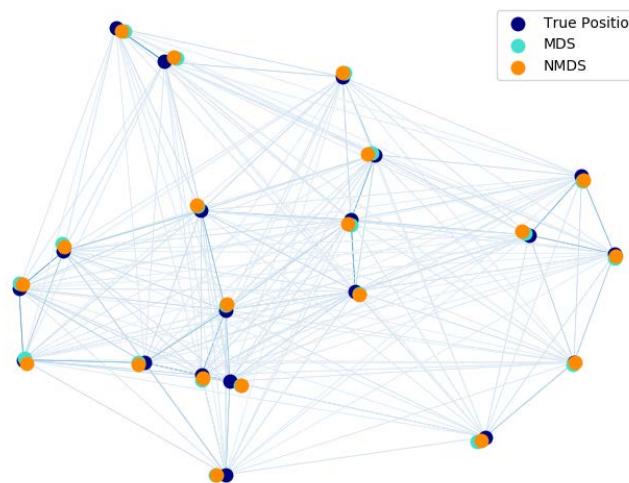
reg = StackingRegressor(estimators=estimators,final_estimator=final_estimator)
```



# Multi-dimensional scaling

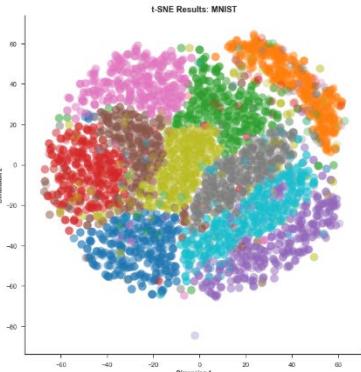
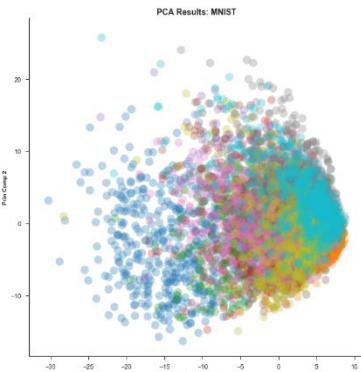
```
#     Author: Nelle Varoquaux <nelle.varoquaux@gmail.com>
#     License: BSD
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.collections import LineCollection
from sklearn import manifold
from sklearn.metrics import euclidean_distances
from sklearn.decomposition import PCA
n_samples = 20
seed = np.random.RandomState(seed=3)
X_true = seed.randint(0, 20, 2 * n_samples).astype(np.float)
X_true = X_true.reshape((n_samples, 2))
#     Center the data
X_true -= X_true.mean()
similarities = euclidean_distances(X_true)
#     Add noise to the similarities
noise = np.random.rand(n_samples, n_samples)
noise = noise + noise.T
noise[np.arange(noise.shape[0]), np.arange(noise.shape[0])] = 0
similarities += noise
mds = manifold.MDS(n_components=2, max_iter=3000, eps=1e-9, random_state=seed,
                   dissimilarity="precomputed", n_jobs=1)
pos = mds.fit(similarities).embedding_
nmuds = manifold.MDS(n_components=2, metric=False, max_iter=3000, eps=1e-12,
                     dissimilarity="precomputed", random_state=seed, n_jobs=1, n_init=1)
npos = nmuds.fit_transform(similarities, init=pos)
#     Rescale the data
pos *= np.sqrt((X_true ** 2).sum()) / np.sqrt((pos ** 2).sum())
npos *= np.sqrt((X_true ** 2).sum()) / np.sqrt((npos ** 2).sum())
#     Rotate the data
clf = PCA(n_components=2)
X_true = clf.fit_transform(X_true)
pos = clf.fit_transform(pos)
npos = clf.fit_transform(npos)
```

```
fig = plt.figure(1)
ax = plt.axes([0., 0., 1., 1.])
s = 100
plt.scatter(X_true[:, 0], X_true[:, 1], color='navy', s=s, lw=0, label='True Position')
plt.scatter(pos[:, 0], pos[:, 1], color='turquoise', s=s, lw=0, label='MDS')
plt.scatter(npos[:, 0], npos[:, 1], color='darkorange', s=s, lw=0, label='NMDS')
plt.legend(scatterpoints=1, loc='best', shadow=False)
similarities = similarities.max() / similarities * 100
similarities[np.isinf(similarities)] = 0
#     Plot the edges
start_idx, end_idx = np.where(pos)
# a sequence of (*line0*, *line1*, *line2*), where::
#     line = (x0, y0), (x1, y1), ... (xm, ym)
segments = [[X_true[i, :], X_true[j, :]] for i in range(len(pos)) for j in range(len(pos))]
values = np.abs(similarities)
lc = LineCollection(segments, zorder=0, cmap=plt.cm.Blues, norm=plt.Normalize(0,
values.max()))
lc.set_array(similarities.flatten())
lc.set_linewidths(np.full(len(segments), 0.5))
ax.add_collection(lc)
plt.show()
```

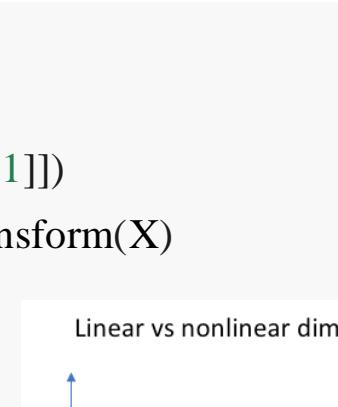
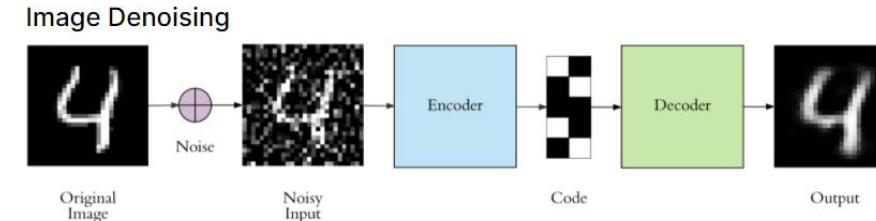


# PCA and tSNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.



```
>>> import numpy as np  
>>> from sklearn.manifold import TSNE  
>>> X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])  
>>> X_embedded = TSNE(n_components=2).fit_transform(X)  
>>> X_embedded.shape  
(4, 2)
```



PCA (principal component analysis) is a linear dimension reduction technique that seeks to maximize variance and preserves large pairwise distances.

<https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html>

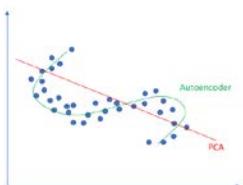
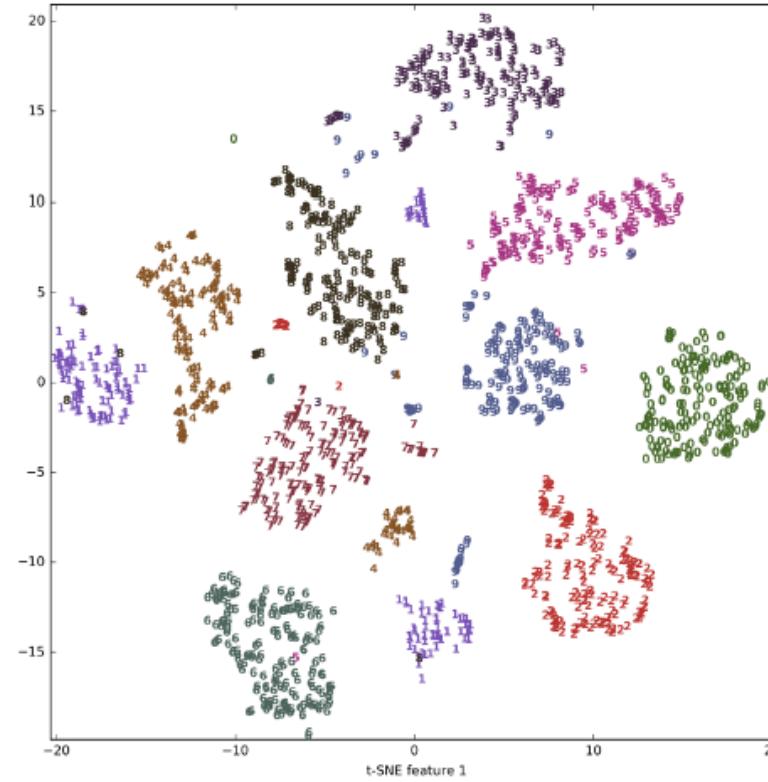
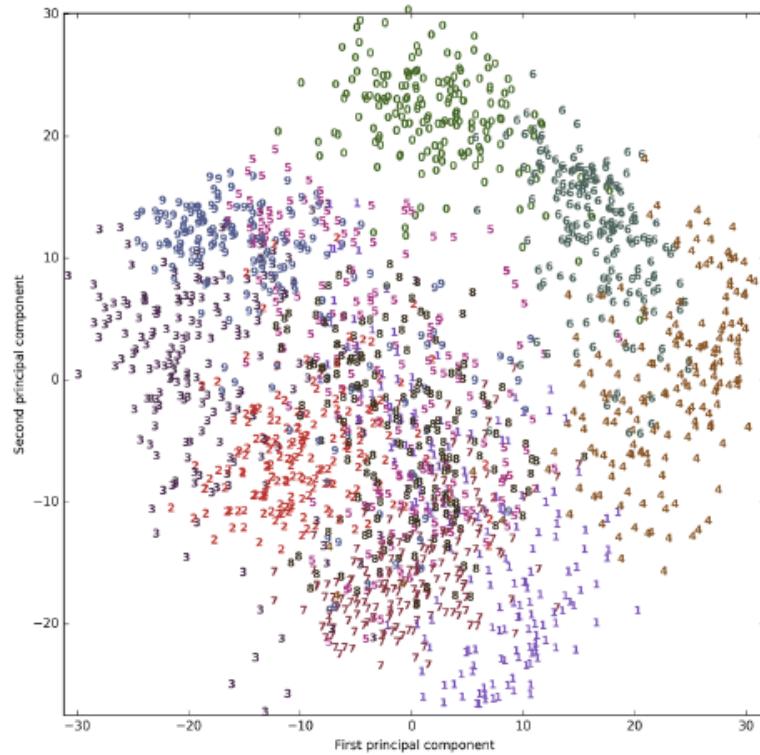
[https://en.wikipedia.org/wiki/T-distributed\\_stochastic\\_neighbor\\_embedding](https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding)

<https://lvdmaaten.github.io/tsne/>

<https://github.com/aviolante/sas-python-work/blob/master/tSNEExampleBlogPost.ipynb>

# PCA and tSNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.

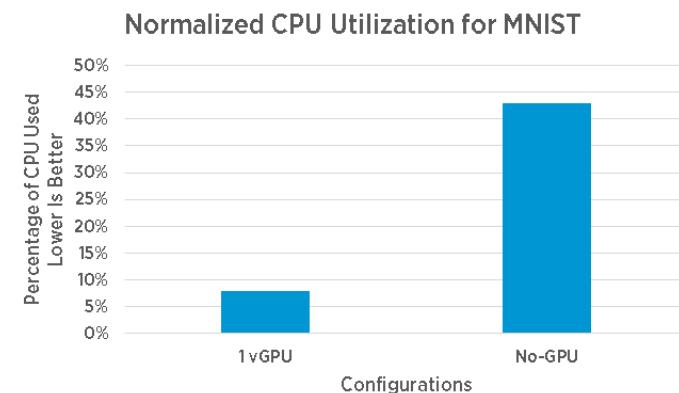
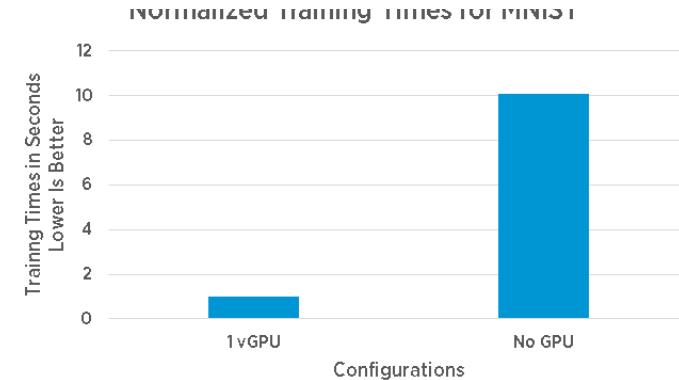




# Keras: The Python Deep Learning library

Use *Keras* if you need a *deep learning* library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and **GPU**.



Anaconda, Tensorflow, Keras Installation on Windows  
<https://www.youtube.com/watch?v=CcKf-iZ8umk&t=145s>

Anaconda, Tensorflow, Keras Installation on Linux  
<https://www.youtube.com/watch?v=lyYaZWxgods>

[Python for Data Science Cheat Sheet: Keras](#)

<https://keras.io>

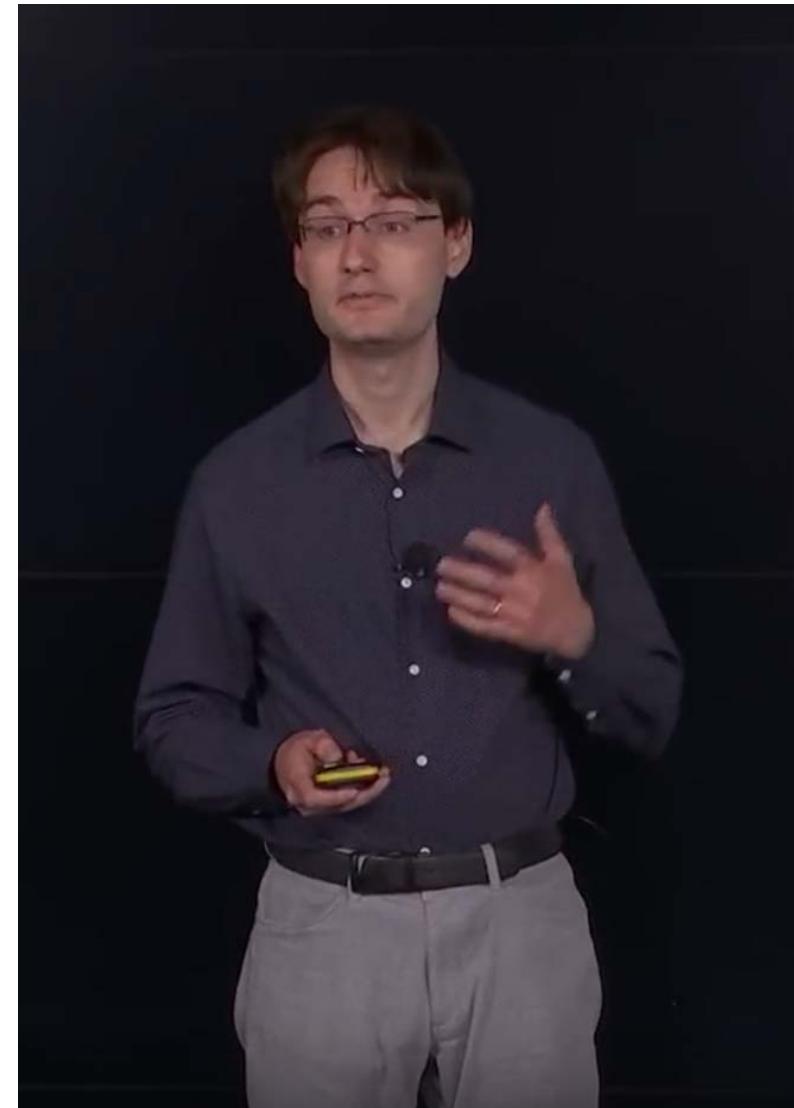
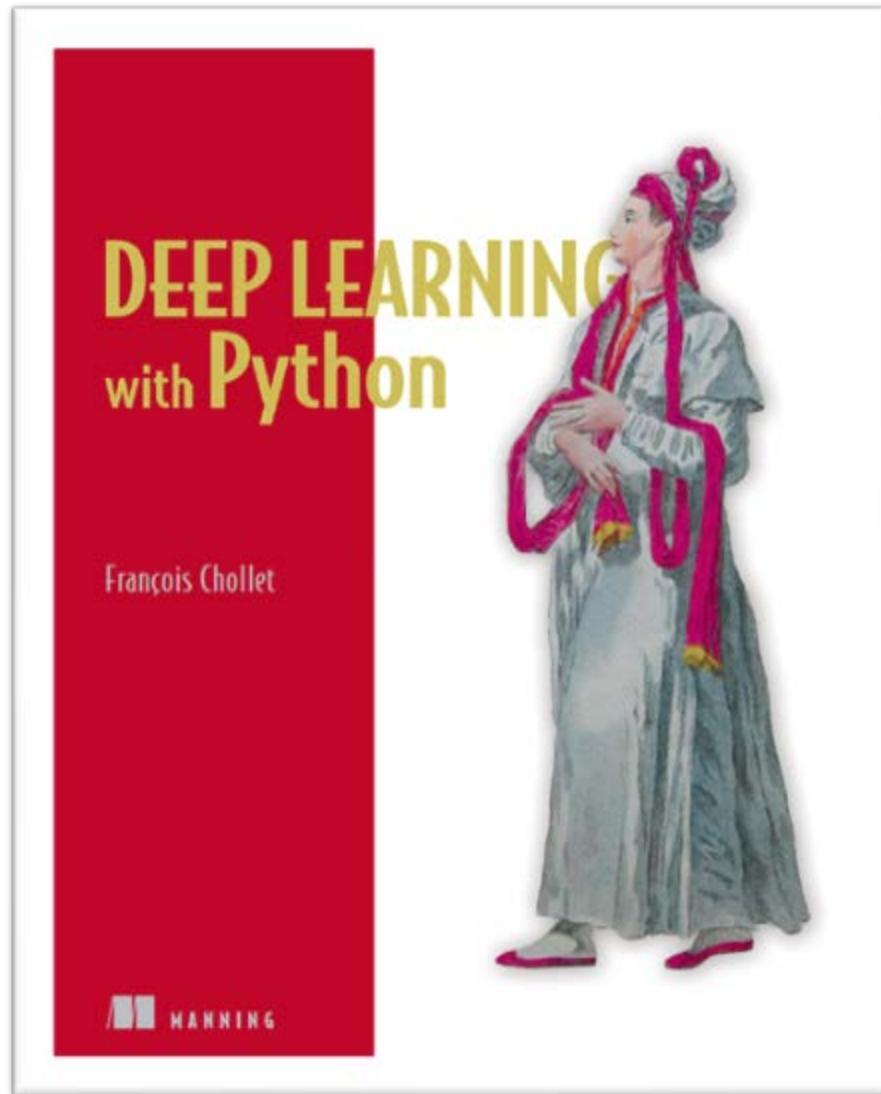


# Deep Learning packages

- TensorFlow : Google
- Keras : Google
- PyTorch : Facebook
- Chainer : Preferred Networks in partnership with IBM, Intel, Microsoft, and Nvidia
- Caffe : UC Berkeley
- CNTK : Microsoft



# Reference



164

<https://www.youtube.com/watch?v=2L2u303FAs8>

**KRISS**



Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers, with the aim of enabling fast experimentation.

Keras has the following key features:

- ?] It allows the same code to run seamlessly on CPU or GPU.
- ?] It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
- ?] It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- ?] It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.



# Keras: high-level NN API

Keras is a model-level library, providing high-level building blocks for developing DL models. It doesn't handle low-level operations such as tensor manipulation and differentiation.

Thanks to **symbolic differentiation**, you'll never have to implement the Backpropagation algorithm by hand.

Currently, the three existing backend implementations are the TensorFlow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend.

**High-level NN API**

**Runs on top of TF, Theano, CNTK**

**Simple: stacking layers, connecting graphs**

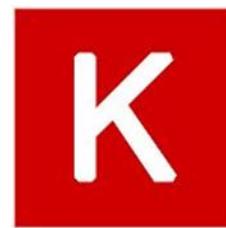
**Open source**

**High performance**

**400,000 developers**

**Linux/MacOs/Windows, automatic differentiation, Cuda**

# Keras



It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used.

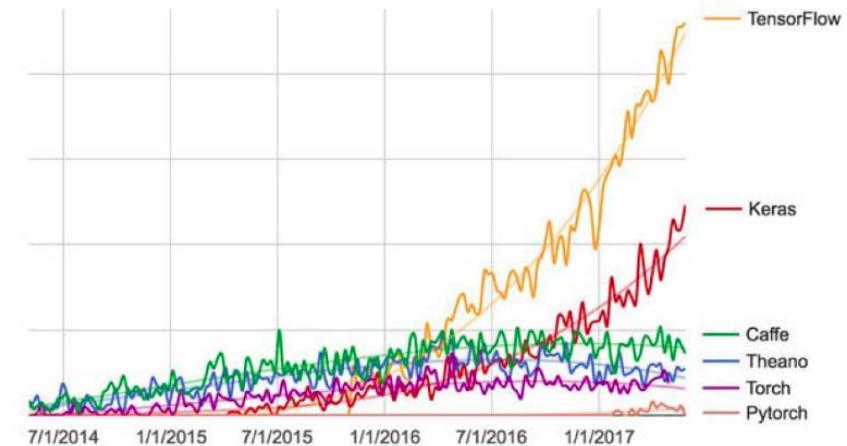
In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

## Comparison of deep-learning software

**Tensorflow 1.4:** Keras

# Keras has well over 250,000 users.

Keras is used at Google, Netflix, Uber, CERN, Yelp, Square, and hundreds of startups working on a wide range of problems.



# PyTorch

**PyTorch** is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it.





# Install Keras

- 1 Install the Python scientific suite—Numpy and SciPy—and make sure you have a Basic Linear Algebra Subprogram (BLAS) library installed so your models run fast on CPU.
- 2 Install two extras packages that come in handy when using Keras: HDF5 (for saving large neural-network files) and Graphviz (for visualizing neural-network architectures).
- 3 Make sure your GPU can run deep-learning code, by installing CUDA drivers and cuDNN.
- 4 Install a backend for Keras: TensorFlow, CNTK, or Theano.
- 5 Install Keras.

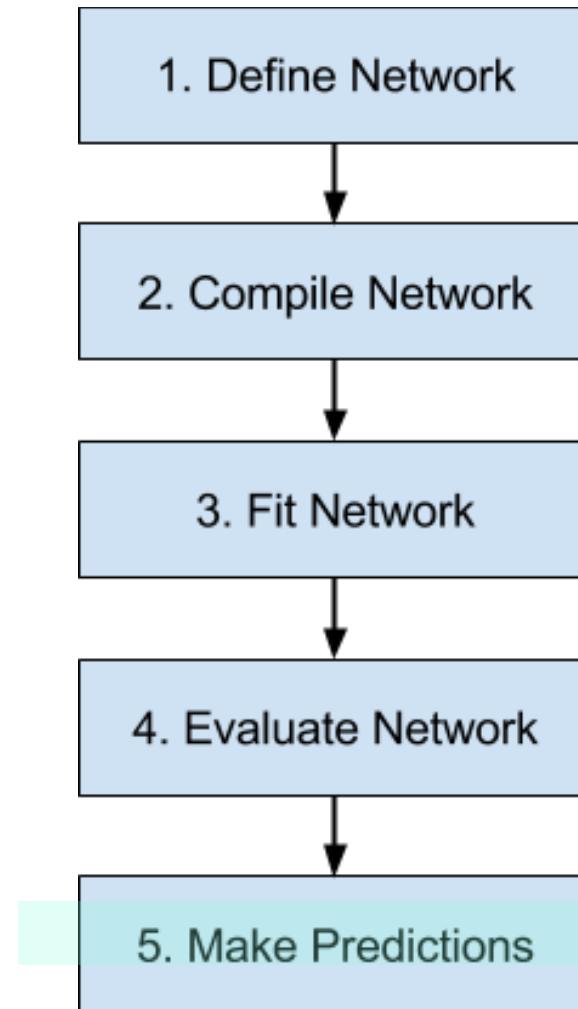
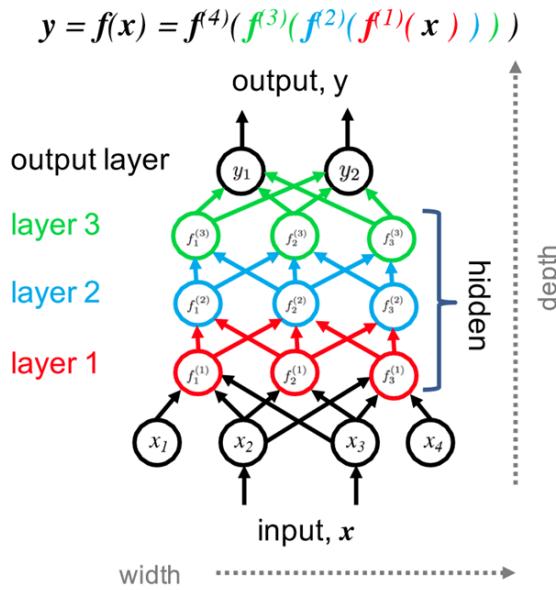


We can now dive into practical Keras examples.

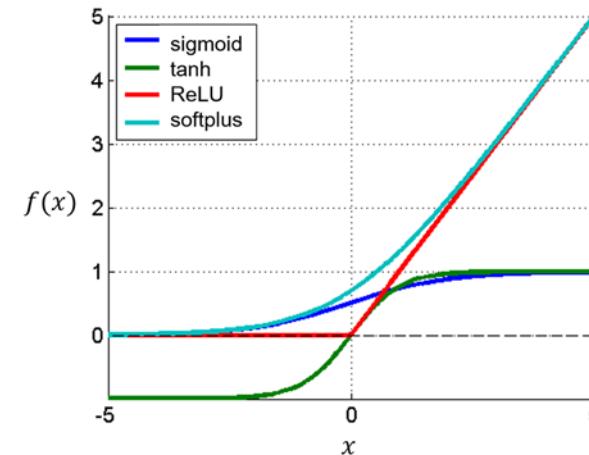
```
from keras.models import Sequential
```

(only for linear stacks of layers, which is the most common network architecture by far)

```
models.Sequential()  
  
layers.Input(shape=(784, ))  
  
layers.Dense(32, activation='relu')  
  
layers.Dense(10, activation='softmax')
```



‘relu’, ‘linear’, ‘sigmoid’, ‘softmax’  
 ‘uniform’, ‘normal’



$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$



# GPU (1/3)

Yes you can run keras models on GPU. Few things you will have to check first.

- 1.your system has GPU (Nvidia. As AMD doesn't work yet)
- 2.You have installed the GPU version of tensorflow
- 3.You have installed CUDA [installation instructions](#)
- 4.Verify that tensorflow is running with GPU [check if GPU is working](#)

```
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
```

OR

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

output will be something like this:

```
[ name: "/cpu:0" device_type: "CPU", name: "/gpu:0" device_type: "GPU" ]
```

Once all this is done your model will run on GPU:

To Check if keras(>=2.1.1) is using GPU:

```
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

You need to add the following block after importing keras. I am working on a machine which have 56 core CPU, and a GPU.

```
import keras
import tensorflow as tf
config = tf.ConfigProto( device_count = { 'GPU': 1 , 'CPU': 56} )
sess = tf.Session(config=config)
keras.backend.set_session(sess)
```



# GPU (2/3)

If you're going to buy a GPU, which one should you choose? The first thing to note is that it must be an NVIDIA GPU. NVIDIA is the only graphics computing company that has invested heavily in deep learning so far, and modern deep-learning frameworks can only run on NVIDIA cards.

## keras 2.0.9

```
from keras.utils.training_utils import multi_gpu_model
model = cnn_model()
model = multi_gpu_model(model, gpus=4)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(train_x, train_y, validation_data=(test_x, test_y), nb_epoch=20, batch_size=32, verbose=1)
```

[https://keras.io/utils/#multi\\_gpu\\_model](https://keras.io/utils/#multi_gpu_model)

<http://blog.naver.com/PostView.nhn?blogId=ossiriand&logNo=221010346484&parentCategoryNo=&categoryNo=48&viewDate=&isShowPopularPosts=true&from=search>

<https://www.youtube.com/watch?v=wQ8BIBpya2k&t=766s>

<https://www.youtube.com/watch?v=j-3vuBynnOE>

<https://www.youtube.com/watch?v=XNKeayZW4dY>



# GPU (3/3)

```
import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model
import numpy as np
num_samples = 1000
height = 224
width = 224
num_classes = 1000
# Instantiate the base model (or "template" model).
# We recommend doing this with under a CPU device scope,
# so that the model's weights are hosted on CPU memory.
# Otherwise they may end up hosted on a GPU, which would
# complicate weight sharing.
with tf.device('/cpu:0'):
    model = Xception(weights=None, input_shape=(height, width, 3), classes=num_classes)
# Replicates the model on 8 GPUs.
# This assumes that your machine has 8 available GPUs.
parallel_model = multi_gpu_model(model, gpus=8)
parallel_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
# Generate dummy data.
x = np.random.random((num_samples, height, width, 3))
y = np.random.random((num_samples, num_classes))
# This `fit` call will be distributed on 8 GPUs.
# Since the batch size is 256, each GPU will process 32 samples.
parallel_model.fit(x, y, epochs=20, batch_size=256)
# Save model via the template model (which shares the same weights):
model.save('my_model.h5')
```



# Practice

파일로부터 데이터를 읽는 방법, 예제:

파일속의 내용을 모두 스트링으로 읽어낸 다음, 숫자로 변환합니다.

```
aa=[]
bb=[]
colors = []
afile=open("fort.11","r")
ii=0
for line in afile:
    if len(line.split()) ==4:
        continue
    if len(line.split()) ==2:
        ii=ii+1
        bb.append(float(line.split()[1]))
        aa.append(int(line.split()[0]))
        colors.append(float(line.split()[0])/230.)
afile.close()
aa=np.array(aa)
bb=np.array(bb)
colors=np.array(colors)
```

```
list_of_lines=['First line', 'Second line', 'Third line']
append_multiple_lines('target00.txt', list_of_lines)
```

```
def append_new_line(file_name, text_to_append):
    with open(file_name, "a+") as file_object:
        file_object.seek(0)
        data = file_object.read(100)
        if len(data) > 0:
            file_object.write("\n")
        file_object.write(text_to_append)
def append_multiple_lines(file_name, lines_to_append):
    with open(file_name, "a+") as file_object:
        appendEOL = False
        file_object.seek(0)
        data = file_object.read(100)
        if len(data) > 0:
            appendEOL = True
        for line in lines_to_append:
            if appendEOL == True:
                file_object.write("\n")
            else:
                appendEOL = True
            file_object.write(line)
```

# Scikit-learn and Keras tutorial



Practice does not make perfect. Only perfect practice makes perfect.

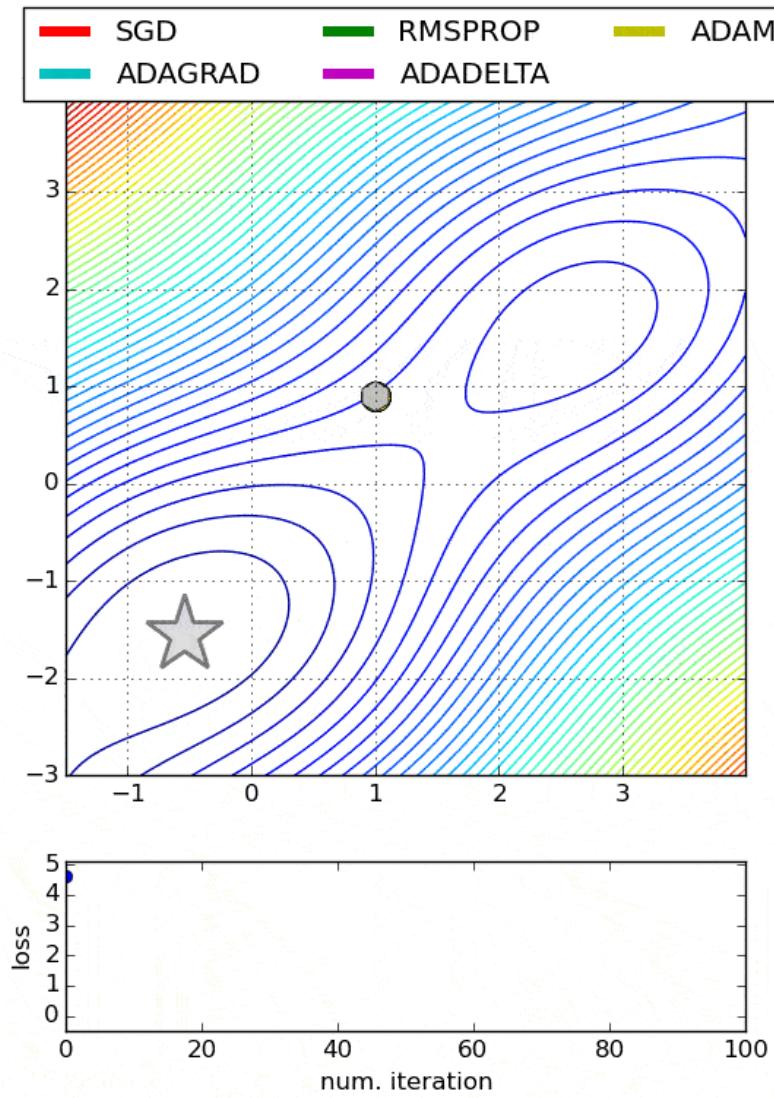
Vince Lombardi

BrainyQuote®



# Optimizers in Keras

- SGD
- SGD with momentum
- Adam
- AdaGrad
- RMSprop
- AdaDelta





```
import os
import tensorflow as tf

if True:
    model.save('my_model_tf', save_format='tf')
    print('writing')

if True:
    if os.path.exists('my_model_tf'):
        del model
    model = tf.keras.models.load_model('my_model_tf')
    print('reading')
```

# Iterative training in Keras

```
input_shape = X_train.shape
model.build(input_shape)
model.summary()
if True:
    plot_model(model, show_shapes=True, to_file='cnn1d_model.png')
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
if False:
    model.fit(X_train, y_train, epochs=500, batch_size=100, verbose=1)
if False:
    history = model.fit(X_train, y_train, epochs=500, batch_size=64, verbose=1)
if True:
    if os.path.exists('my_model_tf'):
        del model
        model = tf.keras.models.load_model('my_model_tf')
        print('reading')
if False:
    history = model.fit(X_train, y_train, batch_size = 100, epochs = 10, validation_split = 0.2, verbose = 1)
    _, accuracy = model.evaluate(X_test, y_test, batch_size=100, verbose=1)
if True:
    history = model.fit(X_train, y_train, batch_size = 100, epochs = 10, validation_data = (X_test,y_test), verbose = 1)
    _, accuracy = model.evaluate(X_test, y_test, batch_size=100, verbose=1)
```



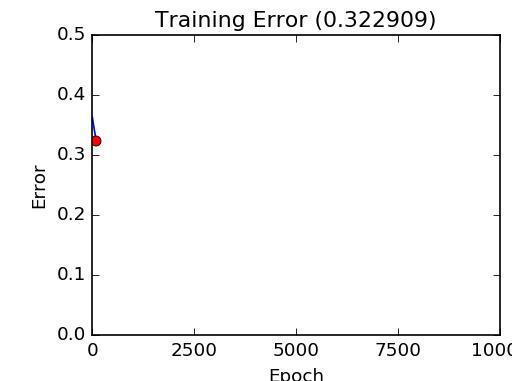
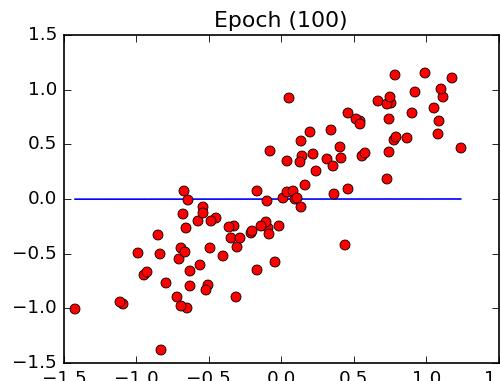
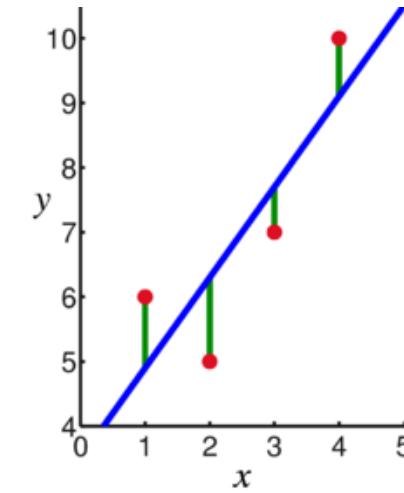
# Linear regression

## Linear regression, output : wide

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import numpy as np

trX=np.linspace(-1, 1, 101)
trY=2*trX+np.random.randn(*trX.shape)*0.33
```

```
model=Sequential()
model.add(Dense(output_dim=1, input_dim=1, init='normal', activation='linear'))
model.compile(optimizer=SGD(lr=0.01), loss='mean_squared_error', metrics=['accuracy'])
model.fit(trX,trY, epochs=100, verbose=1)
```





# Binary Classification

```
# Create your first MLP in Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
numpy.random.seed(7) # Fix random seed for reproducibility
# Load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:, 0:8]
Y = dataset[:, 8]
# Create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# Evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

# Multi-Class Classification

Iris-setosa,	Iris-versicolor,	Iris-virginica
1,	0,	0
0,	1,	0
0,	0,	1

```
# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:, 0:4].astype(float)
Y = dataset[:, 4]
```

```
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```





# LabelEncoder

```
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit([1, 2, 2, 6])
LabelEncoder()
>>> le.classes_
array([1, 2, 6])
>>> le.transform([1, 1, 2, 6])
array([0, 0, 1, 2]...)
>>> le.inverse_transform([0, 0, 1, 2])
array([1, 1, 2, 6])
```

```
>>> le = preprocessing.LabelEncoder()
>>> le.fit(["paris", "paris", "tokyo", "amsterdam"])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(["tokyo", "tokyo", "paris"])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```



# Housing

```
# load dataset
dataframe = pandas.read_csv("housing.csv", delim_whitespace=True, header=None)
dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:, 0:13]
Y = dataset[:, 13]

# define wider model
def wider_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=13, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

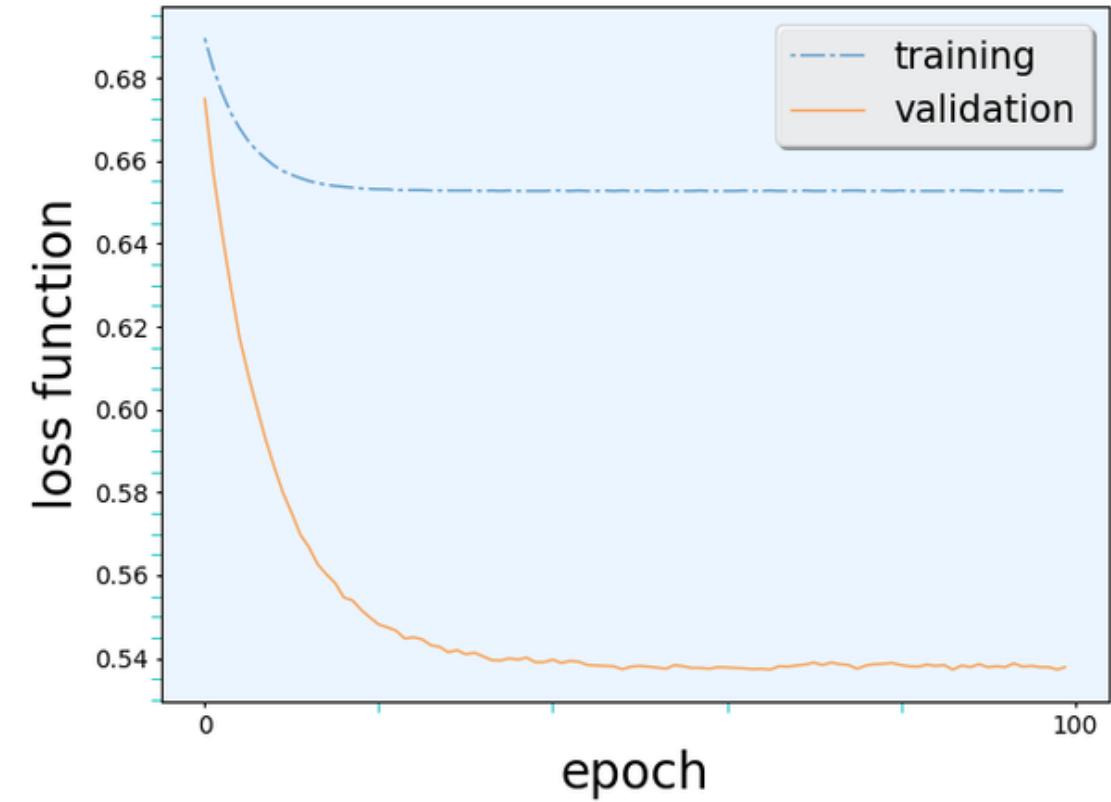
That means that by default it is a **linear** activation.



# Pima-indian/training

## Binary classification

```
import time
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import random
import tensorflow as tf
tf.set_random_seed(12)
np.random.seed(34)
random.seed(56)
random.seed(time.time())
start_time=time.clock()
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=',')
X = dataset[:,0:8]
Y = dataset[:,8]
X, Y = shuffle(X,Y,random_state=0)
x_train, x_test, y_train, y_test= train_test_split(X,Y, test_size=0.2)
model = Sequential()
model.add(Dense(12, input_dim=8, init='normal', activation='relu'))
for i in range(4):
    model.add(Dense(8, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, validation_split=0.10, epochs=100, batch_size=5, verbose=2)
scores = model.evaluate(x_test, y_test)
print("\n% s: % .2f% % " % (model.metrics_names[1], scores[1]*100))
predictions=model.predict(x_test)
rounded=[round(x[0]) for x in predictions]
print(rounded)
if True:
    # serialize model to JSON
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model.h5")
    print("Saved model to disk")
print((time.clock()-start_time)/60./60.,'hours')
```





# Pima-indian/prediction

## Binary classification

```
from keras.models import model_from_json
from sklearn.utils import shuffle
import time
import numpy as np
import random
import tensorflow as tf
tf.set_random_seed(12)
np.random.seed(34)
random.seed(56)
random.seed(time.time())
start_time=time.clock()
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter=',')
X = dataset[:,0:8]
Y = dataset[:,8]
X, Y = shuffle(X,Y,random_state=0)

# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
# evaluate loaded model on test data
loaded_model.compile(loss='mean_squared_error', optimizer='adam')
predicted = loaded_model.predict(X)
print((time.clock()-start_time),'sec')
```



# Iris/training

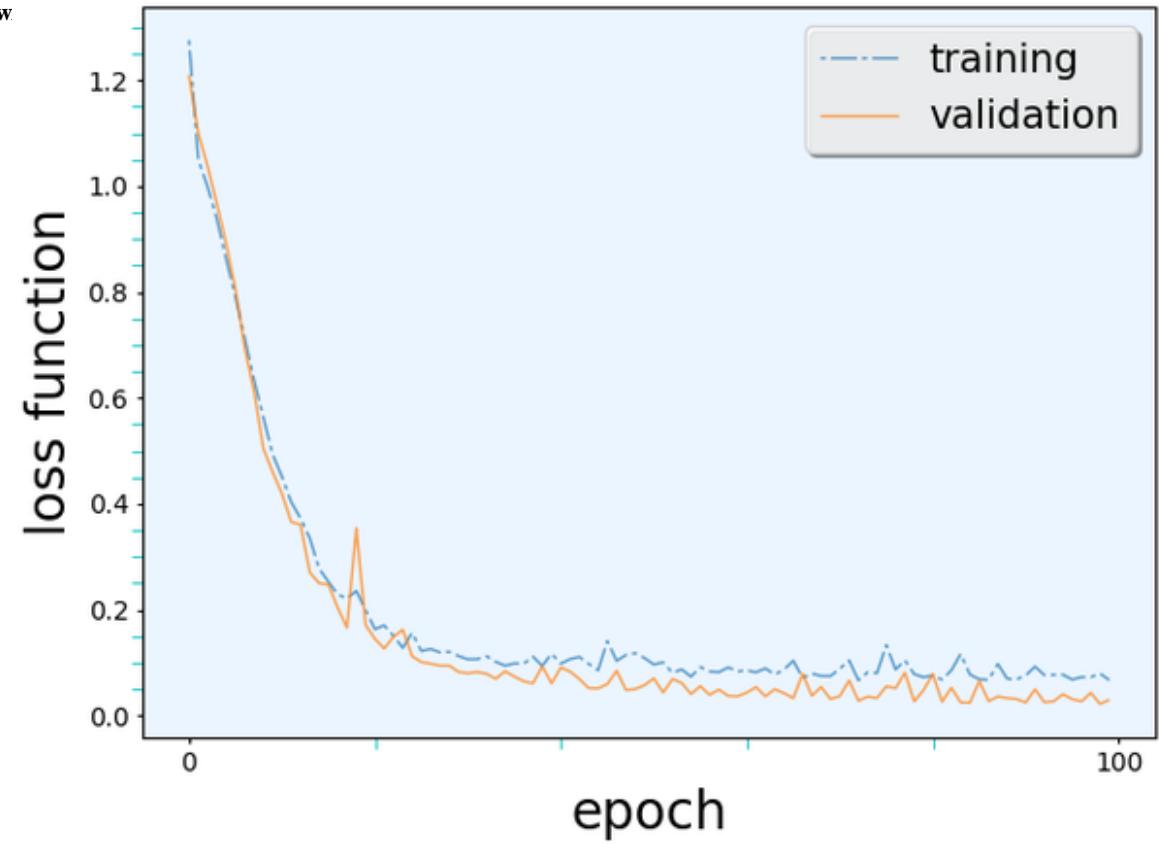
## Multi-class classification

```
df = pd.read_csv('iris.csv',names=["sepal_length", "sepal_width", "petal_length", "petal_w
data_set = df.values
X = data_set[:, 0:4].astype(float)
obj_y = data_set[:, 4]

encoder = LabelEncoder()
encoder.fit(obj_y)
Y_encoded = encoder.transform(obj_y)
Y = np_utils.to_categorical(Y_encoded)

X, Y = shuffle(X,Y,random_state=0)
x_train,x_test,y_train, y_test= train_test_split(X,Y, test_size=0.2)

model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
for i in range(3):
    model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train,y_train, validation_split=0.10, epochs=100, batch_size=5, verbose=2)
scores=model.evaluate(x_test,y_test)
print('\nTest: Loss: {:.4f}'.format(scores[0]))
print('\nTest: Accuracy: {:.4f}'.format(scores[1]))
y_pred=model.predict(x_test)
y_pred=np.argmax(y_pred, axis=1)
y_test=np.argmax(y_test, axis=1)
print(classification_report(y_test,y_pred))
print(confusion_matrix(y_test,y_pred))
if True:
    # serialize model to JSON
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model.h5")
    print("Saved model to disk")
print((time.clock()-start_time)/60./60.,'hours')
```





# Iris/prediction

## Multi-class classification

```
df = pd.read_csv('iris.csv',names=["sepal_length", "sepal_width", "petal_length",  
"petal_width", "species"])  
data_set = df.values  
X = data_set[:, 0:4].astype(float)  
obj_y = data_set[:, 4]  
  
encoder = LabelEncoder()  
encoder.fit(obj_y)  
Y_encoded = encoder.transform(obj_y)  
Y = np_utils.to_categorical(Y_encoded)  
  
X, Y = shuffle(X,Y,random_state=0)  
x_train, x_test,y_train, y_test= train_test_split(X,Y, test_size=0.2)  
  
# load json and create model  
json_file = open('model.json', 'r')  
loaded_model_json = json_file.read()  
json_file.close()  
loaded_model = model_from_json(loaded_model_json)  
# load weights into new model  
loaded_model.load_weights("model.h5")  
print("Loaded model from disk")  
# evaluate loaded model on test data  
loaded_model.compile(loss='mean_squared_error', optimizer='adam')  
predicted = loaded_model.predict(X)  
print((time.clock()-start_time),'sec')
```



# Categorical variables

There are many ways to encode categorical variables for modeling, although the three most common are as follows:

1. **Integer Encoding:** Where each unique label is mapped to an integer.
2. **One Hot Encoding:** Where each label is mapped to a binary vector.
3. **Learned Embedding:** Where a distributed representation of the categories is learned.



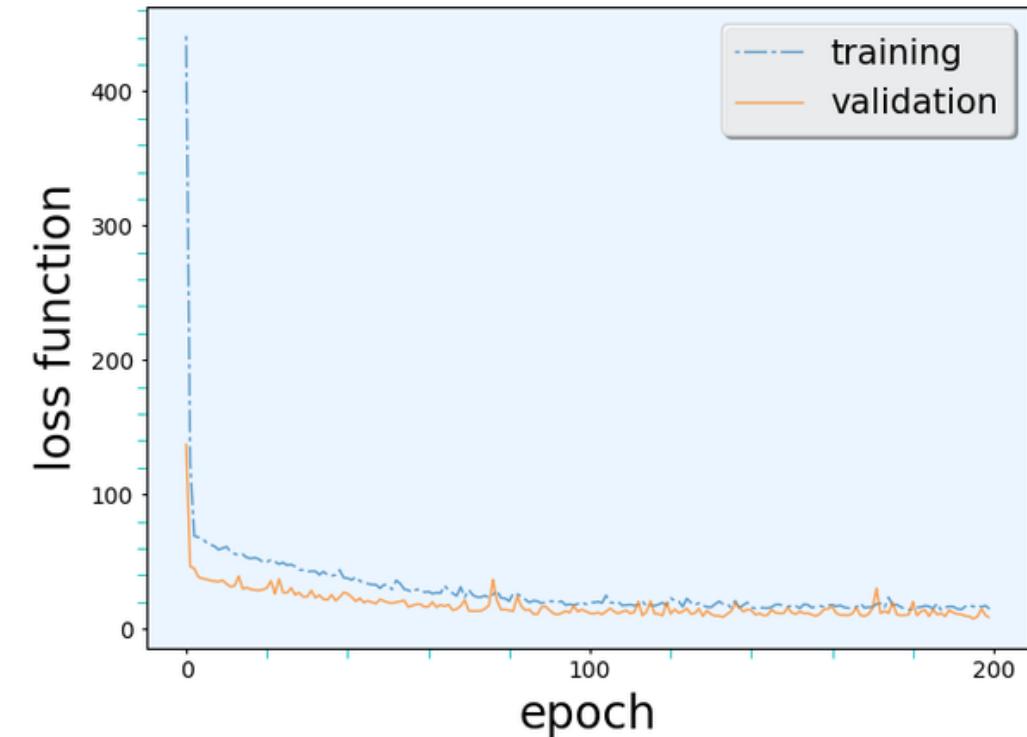
# House/training

## Regression

```
df = pd.read_csv('housing.csv', delim_whitespace=True, header=None)
data_set = df.values
X = data_set[:, 0:13]
Y = data_set[:, 13]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=0.2)
model = Sequential()
model.add(Dense(30, input_dim=13, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1))
model.summary()
model.compile(loss='mean_squared_error', optimizer='adam')
#model.fit(X_train, Y_train, epochs=200, batch_size=10)
model.fit(X_train, Y_train, validation_split=0.10, epochs=200, batch_size=10, verbose=2)

Y_prediction = model.predict(X_validation).flatten()
for i in range(10):
    real_price = Y_validation[i]
    predicted_price = Y_prediction[i]
    print('Real Price: {:.3f}, Predicted Price: {:.3f}'.format(real_price, predicted_price))

if True:
    # serialize model to JSON
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model.h5")
    print("Saved model to disk")
    print((time.clock()-start_time)/60./60.,'hours')
```



That means that by default it is a **linear** activation.



# House/prediction

## Regression

```
df = pd.read_csv('housing.csv', delim_whitespace=True, header=None)
data_set = df.values
X = data_set[:, 0:13]
Y = data_set[:, 13]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

if True:
    # load json and create model
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("model.h5")
    print("Loaded model from disk")

loaded_model.compile(loss='mean_squared_error', optimizer='adam')
Y_prediction = loaded_model.predict(X_test).flatten()
for i in range(10):
    real_price = Y_test[i]
    predicted_price = Y_prediction[i]
    print('Real Price: {:.3f}, Predicted Price: {:.3f}'.format(real_price, predicted_price))
print((time.clock()-start_time)/60./60.,'hours')
```



# Saving/Loading Keras models

- Use `.save` method to save the model
- Use `load_model` function to load saved model
- Saved file contains –
  - Architecture of the model
  - Weights and biases
  - State of the optimizer
- Saving weights
- Loading all the weights and loading weights layer wise

```
from keras.models import load_model

model.save('my_model.h5') # creates a HDF5 file 'my_model.h5'
del model # deletes the existing model

# returns a compiled model
# identical to the previous one
model = load_model('my_model.h5')

model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5', by_name=True)
```



# General layers

- Core layers
- Convolution layers
- Pooling layers
- Locally-connected layers
- Recurrent layers
- Merge layers
- Advanced activation layers
- Normalization layers
- Noise layers
- Layer wrappers
- Dropout
- Embedding



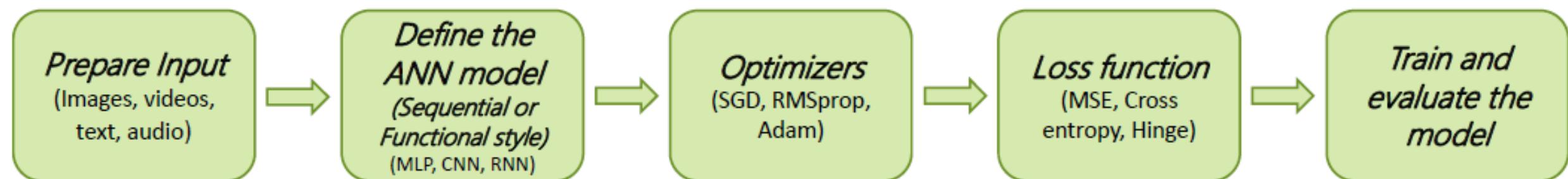
# GaussianNoise

```
# mlp overfit on the two circles dataset with hidden layer noise
from sklearn.datasets import make_circles
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Activation
from keras.layers import GaussianNoise
from matplotlib import pyplot
# generate 2d classification dataset
X, y = make_circles(n_samples=100, noise=0.1, random_state=1)
# split into train and test
n_train = 30
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
# define model
model = Sequential()
model.add(Dense(500, input_dim=2))
model.add(GaussianNoise(0.1))
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# fit model
history = model.fit(trainX, trainy, validation_data=(testX, testy), epochs=4000, verbose=0)
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f % (train_acc, test_acc)')
# plot history
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```

# Implementing a neural network in Keras

- Five major steps
  - Preparing the input and specify the input dimension (size)
  - Define the model architecture and build the computational graph
  - Specify the optimizer and configure the learning process
  - Specify the Inputs, Outputs of the computational graph (model) and the Loss function
  - Train and test the model on the dataset

**Note:** Gradient calculations are taken care by Auto – Differentiation and parameter updates are done automatically in the backend





# Models

Keras models – Sequential model API

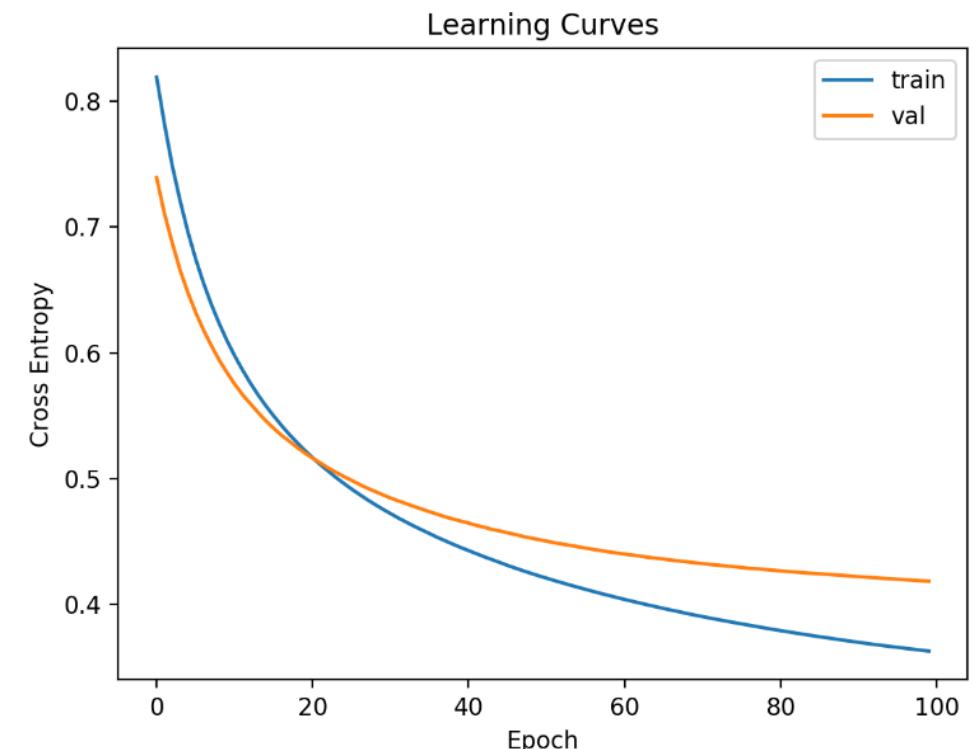
- Sequential model
- Linear stack of layers
- Useful for building simple models
- Simple classification network
- Encoder – Decoder models

Keras models – Functional API

- Functional *Model*
- Multi – input and Multi –output models
- Complex models which forks into 2 or more branches
- Models with shared (Weights)layers



```
# example of plotting learning curves
from sklearn.datasets import make_classification
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot
# create the dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# determine the number of input features
n_features = X.shape[1]
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(1, activation='sigmoid'))
# compile the model
sgd = SGD(learning_rate=0.001, momentum=0.8)
model.compile(optimizer=sgd, loss='binary_crossentropy')
# fit the model
history = model.fit(X, y, epochs=100, batch_size=32, verbose=0, validation_split=0.3)
# plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Cross Entropy')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()
```



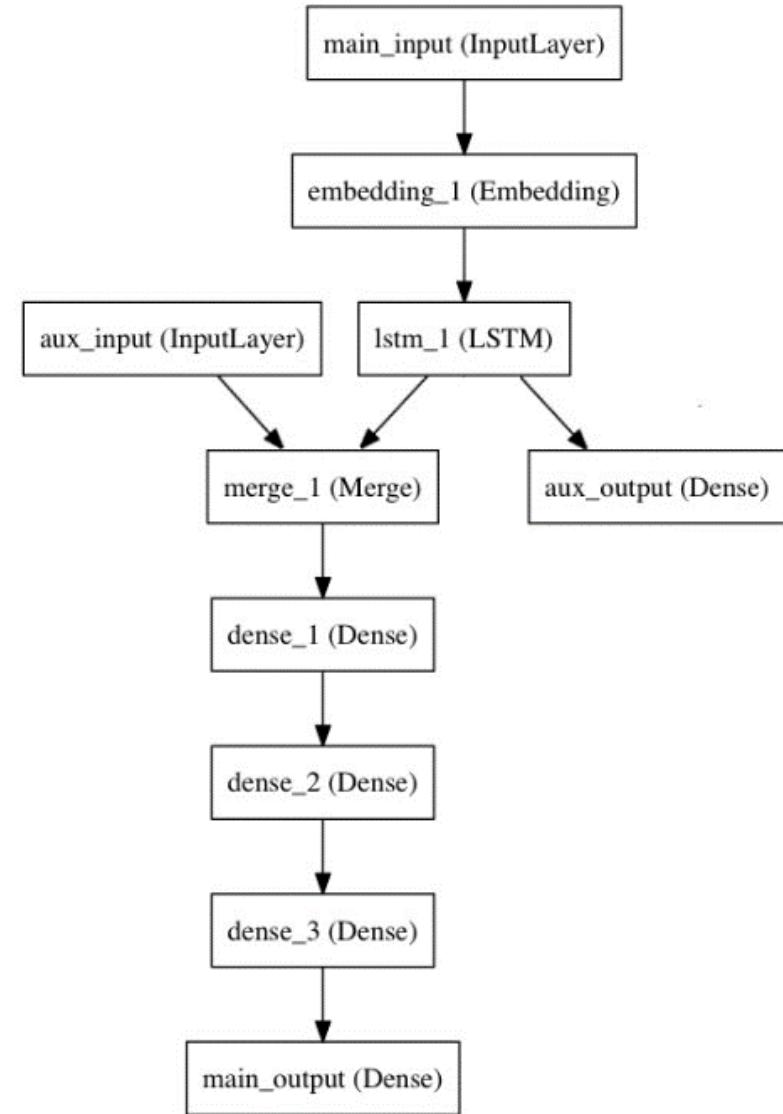


# Multi-input and multi-output

```
from keras.layers import Input, Embedding, LSTM, Dense
from keras.models import Model
import numpy as np
np.random.seed(0)
main_input = Input(shape=(100,), dtype='int32', name='main_input')
x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
lstm_out = LSTM(32)(x)
auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)
auxiliary_input = Input(shape=(5,), name='aux_input')
x = keras.layers.concatenate([lstm_out, auxiliary_input])
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
main_output = Dense(1, activation='sigmoid', name='main_output')(x)
model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output, auxiliary_output])
```

```
from keras.models import Model
from keras.layers import Input, Dense
a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)
```

```
model = Model(inputs=[a1, a2], outputs=[b1, b2, b3])
```





```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential, Model
#from sklearn.preprocessing import MinMaxScaler
%matplotlib inline

# synthetic classification dataset
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=2000, n_features=30, n_informative=10, n_redundant=10, random_state=7)
# summarize the dataset
print(X.shape, y.shape)

num_inputs = 30
num_hidden = 2
num_outputs = num_inputs

model = Sequential()
model.add(Dense(num_inputs, input_shape=[num_inputs]))
model.add(Dense(20, activation='relu')) # 5 < 20 < num_inputs
model.add(Dense(5, activation='relu')) # 2 < 5
model.add(Dense(num_hidden, activation='relu'))
model.add(Dense(5, activation='relu')) # 2 < 5
model.add(Dense(20, activation='relu')) # 2 < 5 < 20 < num_outputs=num_inputs
model.add(Dense(num_outputs))
model.compile(optimizer='adam', loss='mse')
model.summary()

history = model.fit(X, X, validation_split=0.20, epochs=250, batch_size=10, verbose=0)

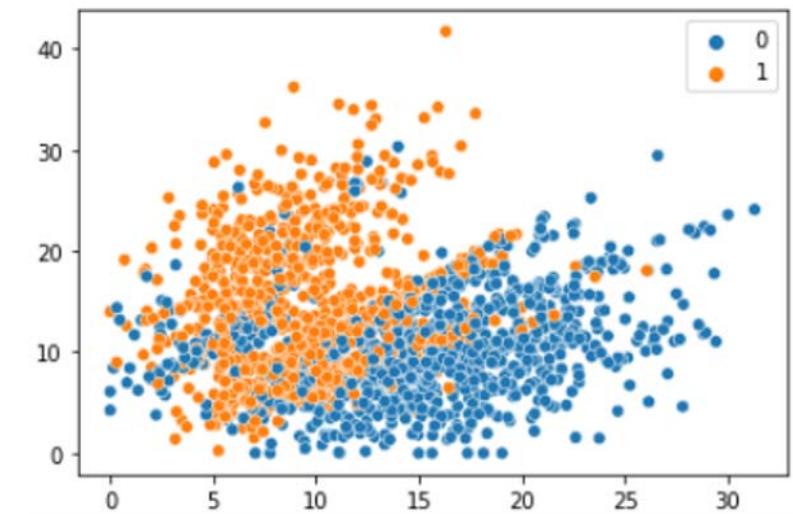
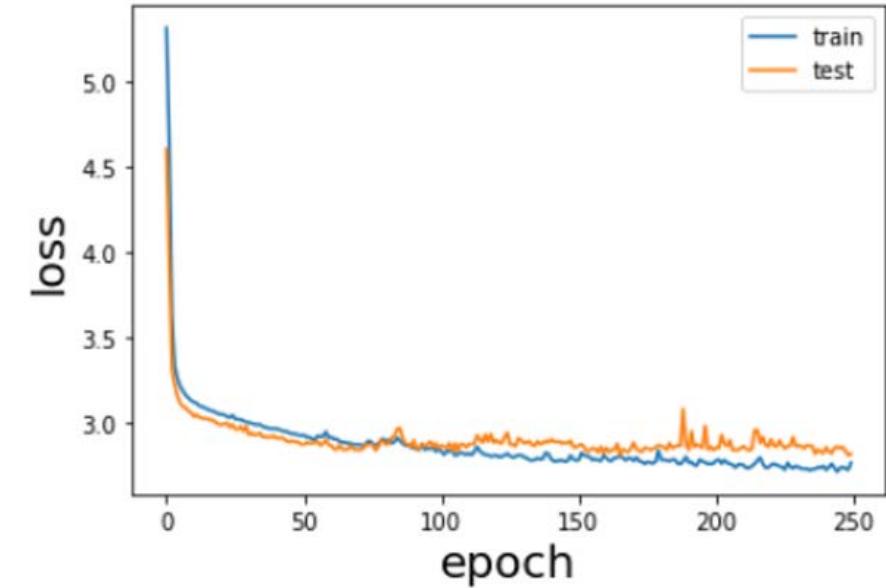
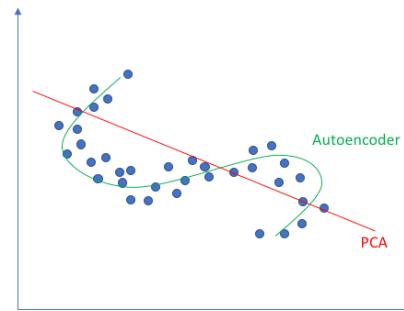
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss', fontsize=20)
plt.ylabel('loss', fontsize=20)
plt.xlabel('epoch', fontsize=20)
plt.legend(['train', 'test'], loc='upper right')
plt.show()

intermediate_layer_model = Model(inputs=model.input, outputs=model.get_layer(index=3).output)
intermediate_output = intermediate_layer_model.predict(X)

intermediate_output.shape

sns.scatterplot(intermediate_output[:,0], intermediate_output[:,1], hue=y )
```

# Autoencoder

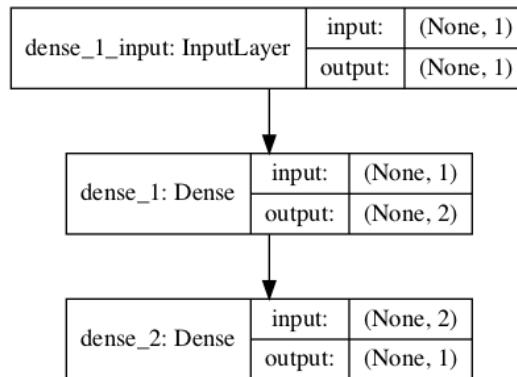




# Visualization of a DNN in Keras

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model

model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	4
dense_2 (Dense)	(None, 1)	3
<hr/>		
Total params: 7		
Trainable params: 7		
Non-trainable params: 0		
<hr/>		

```
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```



# MNIST

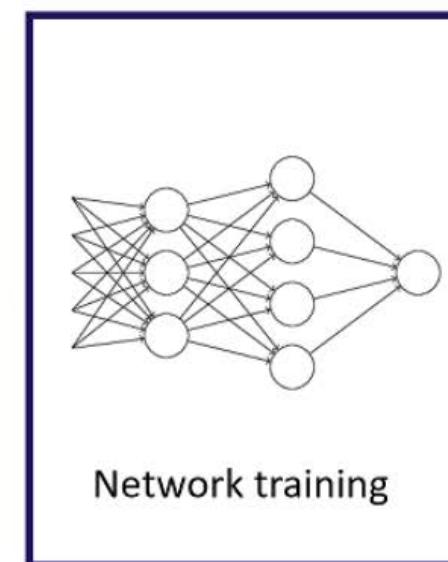
MNIST contains 70,000 images of handwritten digits: 60,000 for training and 10,000 for testing. The images are grayscale,  $28 \times 28$  pixels, and centered to reduce preprocessing and get started quicker.

```
>>> from keras.datasets import mnist
# Load pre-shuffled MNIST data into train and test sets
>>> (X_train, y_train), (X_test, y_test) = mnist.load_data()
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.pkl.gz
>>> print(X_train.shape)
(60000, 28, 28)
>>> print(X_test.shape)
(10000, 28, 28)
>>> print(y_train.shape)
(60000,)
>>> print(y_test.shape)
(10000,)
```



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

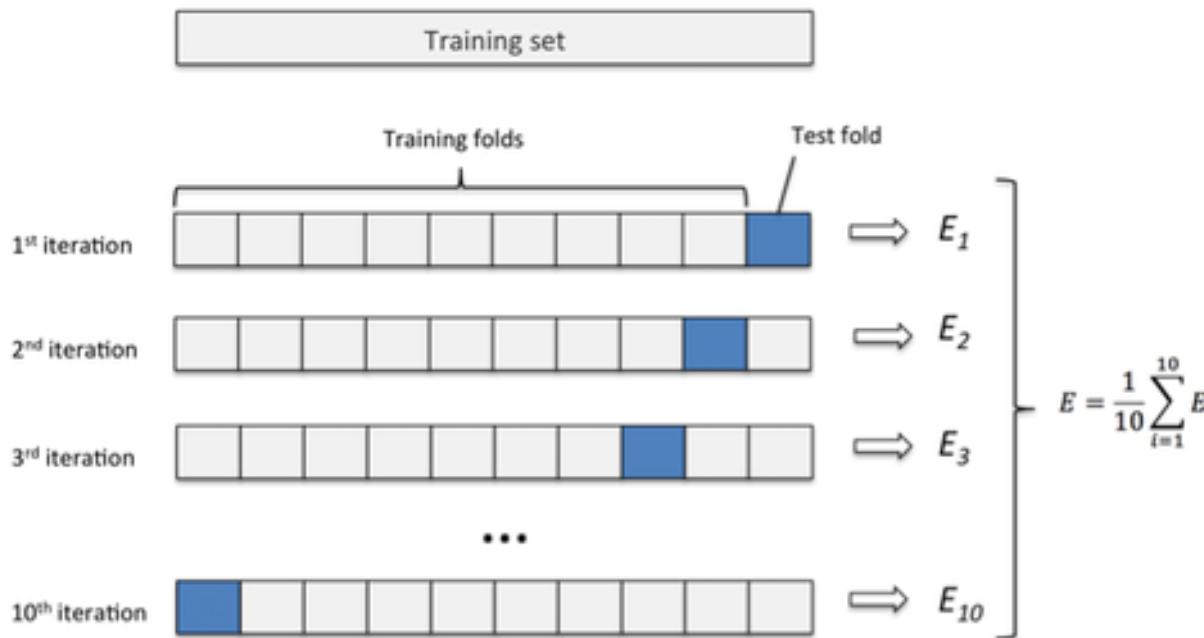
Data & Labels



0  
1  
2  
3  
4  
5  
6  
7  
8  
9



# K-fold cross-validation



```
for n in [3, 5]:  
    kfold = KFold(n_splits=n, shuffle=True, random_state=0)  
    scores = cross_val_score(logreg, iris.data, iris.target, cv=kfold)  
    print('n_splits={}', cross validation score: {}'.format(n, scores))
```

```
>>> from sklearn import datasets, linear_model  
>>> from sklearn.model_selection import cross_val_score  
>>> diabetes = datasets.load_diabetes()  
>>> X = diabetes.data[:150]  
>>> y = diabetes.target[:150]  
>>> lasso = linear_model.Lasso()  
>>> print(cross_val_score(lasso, X, y, cv=3))  
[0.33150734 0.08022311 0.03531764]
```

roc\_auc  
r2  
accuracy

```
np.mean(cross_val_score(clf, X_train, y_train, cv=3, scoring='roc_auc'))
```



# K-fold cross-validation

```
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test]))
```

```
train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

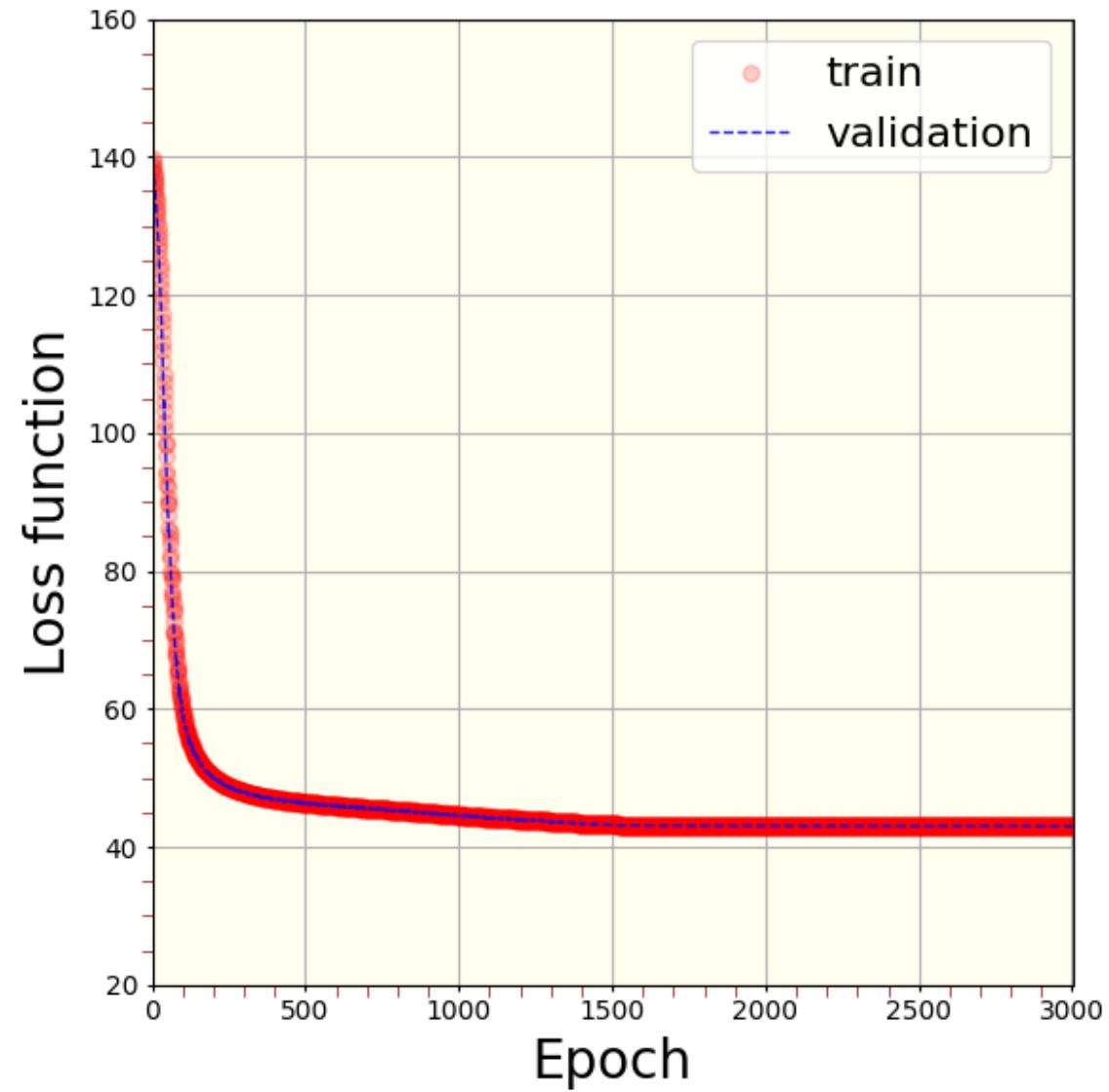
```
from sklearn.datasets import load_iris
iris=load_iris()
from sklearn.model_selection import cross_val_score
logreg=LogisticRegression()
score=cross_val_score(logreg, iris.data, iris.target, cv=5)
print(score)
```

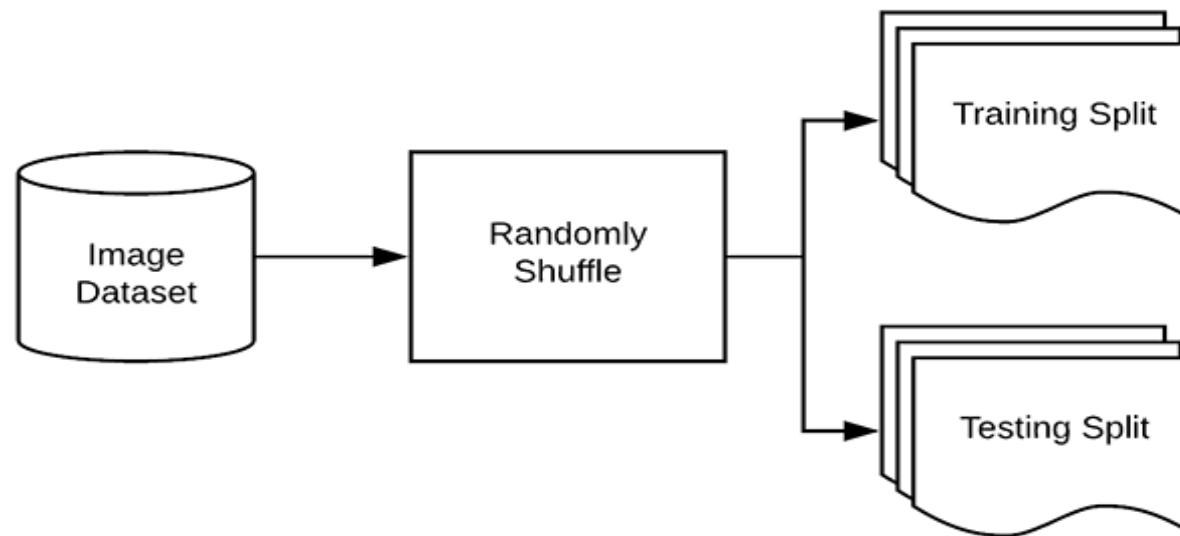
```
from sklearn.model_selection import Kfold
kfold=Kfold(n_splits=5, shuffle=True, random_state=1)
score=cross_val_score(logreg, iris.data,iris.target, cv=kfold)
```



# K-fold cross-validation

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, AutoMinorLocator
ii=[];aa=[];bb=[];kount=0
afile=open("g1","r")
for line in afile:
    if len(line.split())>=11:
        if line.split()[6]=='loss::':
            ii.append(kount)
            aa.append(float(line.split()[7]))
            bb.append(float(line.split()[10]))
            kount=kount+1
afile.close()
ii=np.array(ii)
aa=np.array(aa)
bb=np.array(bb)
plt.figure(figsize=(6,6))
ax=plt.axes()
ax.set_xlabel('Epoch', fontsize=20)
ax.set_ylabel('Loss function', fontsize=20)
majorLocator= MultipleLocator(500)
minorLocator= AutoMinorLocator()
majorFormatter= FormatStrFormatter('%d')
minorFormatter= FormatStrFormatter('%d')
ax.xaxis.set_major_locator(majorLocator)
ax.xaxis.set_major_formatter(majorFormatter)
ax.xaxis.set_minor_locator(minorLocator)
majorLocator= MultipleLocator(20)
minorLocator= AutoMinorLocator()
majorFormatter= FormatStrFormatter('%d')
minorFormatter= FormatStrFormatter('%d')
ax.xaxis.set_major_locator(majorLocator)
ax.xaxis.set_major_formatter(majorFormatter)
ax.xaxis.set_minor_locator(minorLocator)
ax.tick_params(which='major', length=2, color='black')
ax.tick_params(which='minor', length=4, color='brown')
ax.set_facecolor("ivory")      # ax.set_facecolor("beige")
plt.grid(True)
plt.plot(ii,aa,'o', linewidth=12, alpha=0.2, c='red')
plt.plot(ii,bb,'--', linewidth=1, alpha=0.9, c='blue')
plt.xlim(0, 3000)
plt.ylim(20, 160)
plt.legend(['train','validation'], loc="upper right", prop={'size': 16})
# plt.colorbar()
plt.tight_layout()
str1='conv.pdf'
plt.savefig(str1,dpi=150)
plt.show()
```





```
# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.25, random_state=42)
```

```
>>> import numpy as np
>>> from sklearn.model_selection import KFold
>>> X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
>>> y = np.array([1, 2, 3, 4])
>>> kf = KFold(n_splits=2)
>>> kf.get_n_splits(X)
2
>>> print(kf)
KFold(n_splits=2, random_state=None, shuffle=False)
>>> for train_index, test_index in kf.split(X):
...     print("TRAIN:", train_index, "TEST:", test_index)
...     X_train, X_test = X[train_index], X[test_index]
...     y_train, y_test = y[train_index], y[test_index]
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
```



# GridSearchCV

```
from sklearn.datasets import load_boston
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error, make_scorer, r2_score
import matplotlib.pyplot as plt
boston = load_boston()
x, y = boston.data, boston.target
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)
abreg = AdaBoostRegressor()
params = { 'n_estimators': [50, 100], 'learning_rate' : [0.01, 0.05, 0.1, 0.5], 'loss' : ['linear', 'square', 'exponential'] }
score = make_scorer(mean_squared_error)
gridsearch = GridSearchCV(abreg, params, cv=5, return_train_score=True)
gridsearch.fit(xtrain, ytrain)
print(gridsearch.best_params_)
best_estim = gridsearch.best_estimator_
print(best_estim)
best_estim.fit(xtrain, ytrain)
ytr_pred = best_estim.predict(xtrain)
mse = mean_squared_error(ytr_pred, ytrain)
r2 = r2_score(ytr_pred, ytrain)
print("MSE: %.2f" % mse)
print("R2: %.2f" % r2)
ypred = best_estim.predict(xtest)
mse = mean_squared_error(ytest, ypred)
r2 = r2_score(ytest, ypred)
print("MSE: %.2f" % mse)
print("R2: %.2f" % r2)
x_ax = range(len(ytest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
```



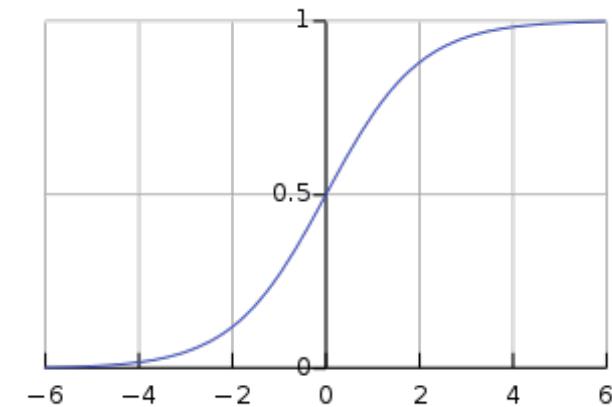
# Logistic regression

Logistic regression, output: [0,1]

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.datasets import mnist
from keras.utils import np_utils
batch_size=128 ; nb_classes=10
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train=X_train.reshape(6000,784)
X_test=X_test.reshape(1000,784)
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
Y_Train=np_utils.to_categorical(y_train, nb_classes)
Y_Test=np_utils.to_categorical(y_test, nb_classes)
model=Sequential()
model.add(Dense(output_dim=10, input_dim=(784,), init='normal', activation='softmax'))
model.compile(optimizer=SGD(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
history=model.fit(X_train, Y_Train, epochs=100, batch_size=128, verbose=1)
evaluation=model.evaluate(X_test,Y_Test,verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f', %(evaluation[0], evaluation[1]))
```

‘relu’, ‘linear’, ‘sigmoid’, ‘softmax’

‘uniform’, ‘normal’



The standard logistic function.



# Deep ffnnet/dropout

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import RMSprop
from keras.datasets import mnist
from keras.utils import np_utils
batch_size=128; nb_classes=10 ; nb_epoch=100
(X_train, y_train), (X_test, y_test)=mnist.load_data()
X_train=X_train.reshape(60000,784)
X_test=X_test.reshape(10000,784)
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
Y_train=np_utils.to_categorical(y_train, nb_classes)
Y_test=np_utils.to_categorical(y_test, nb_classes)
model=Sequential()
model.add(Dense(output_dim=625, input_dim=784, init='normal'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(output_dim=625, input_dim=625, init='normal'))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(output_dim=10, input_dim=625, init='normal'))
model.add(Activation('softmax'))
model.compile(optimizer=RMSprop(lr=0.001, rho=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
history=model.fit(X_train,Y_train, nb_epoch=100, batch_size=128, verbose=1)
evaluation=model.evaluate(X_test, Y_test, verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (evaluation[0], evaluation[1]))
```

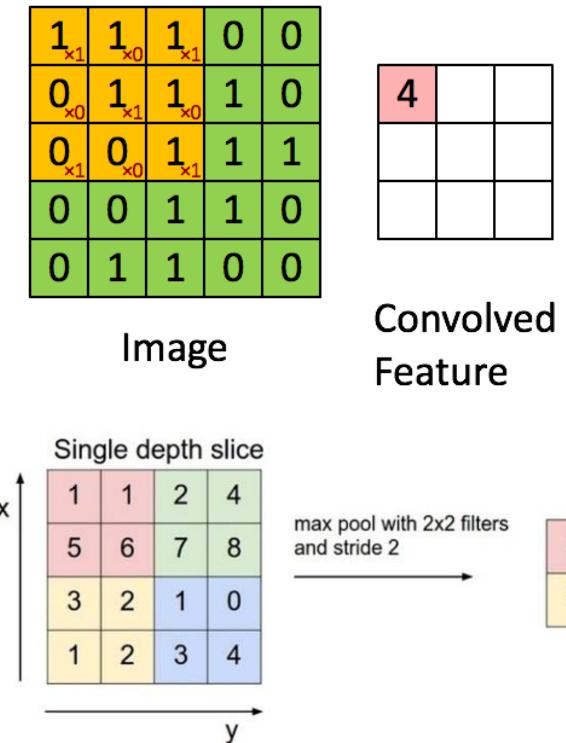
Use MLPs For:  
Tabular datasets  
Classification prediction problems  
Regression prediction problems



# convolution



$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t - \tau) d\tau.$$



Channel is the information that we are seeking in the image.

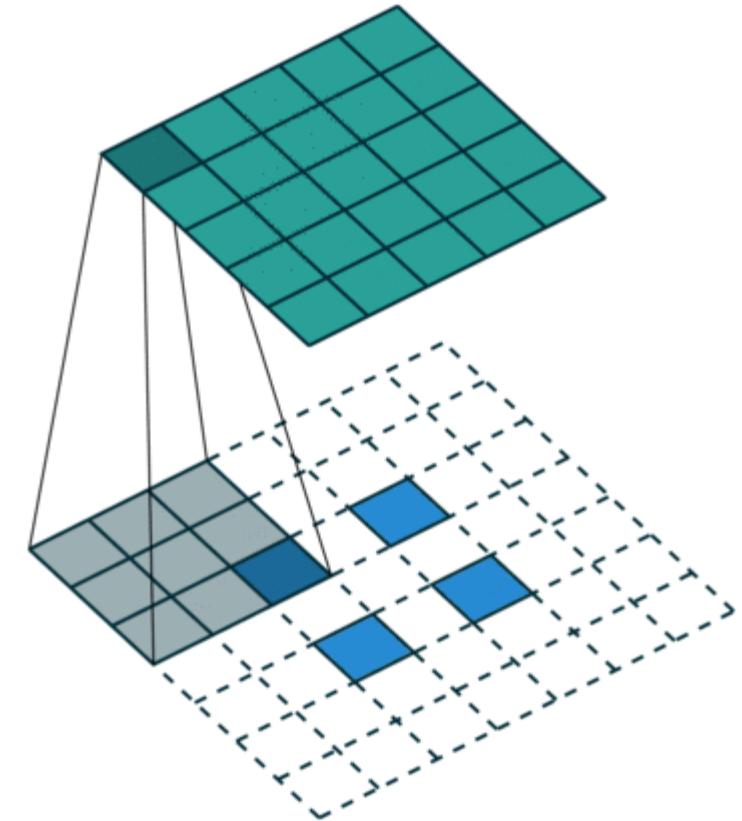
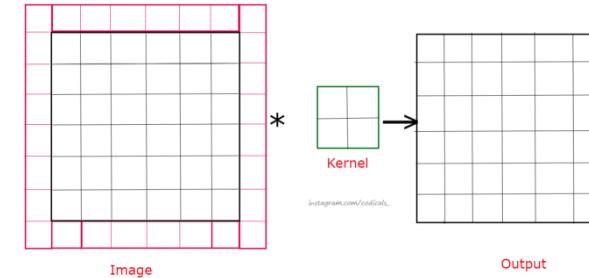
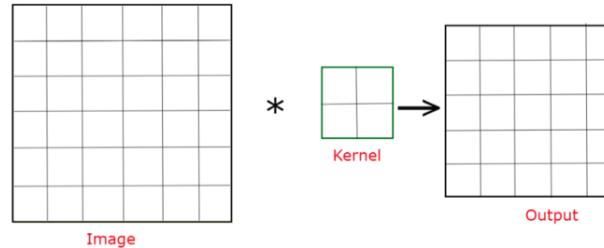
Kernel is the filter that filters images and gives important and desired information.

Image → Convolution process → Feature Map → Pooling process → Pooled Feature map → Flattening → One Dimensional Vector

Gaussian blur can be used to obtain a smooth grayscale digital image of a **halftone** print.



# Convolution/Deconvolution



**Convolution** is extracting important information from images using Kernel, and make an output matrix often known as a ***Feature map***. Image's dimensions get reduced and important information is also retained.

`o/p_size = img_size - kernel_size +1 +(2* padding_size)`

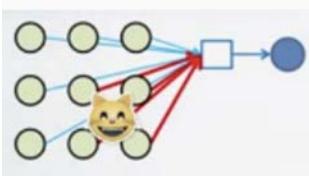
`padding_size=(kernel_size - 1) / 2`

**Convolution** adds up the information spread in various pixels to one pixel, whereas **DeConvolution** spreads the information present in one pixel to various pixels.

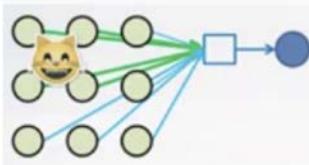
# convolution

The general idea of CNN's is to intelligently adapt to the properties of images:

- Pixel position and neighborhood have semantic meanings
- Elements of interest can appear anywhere in the image

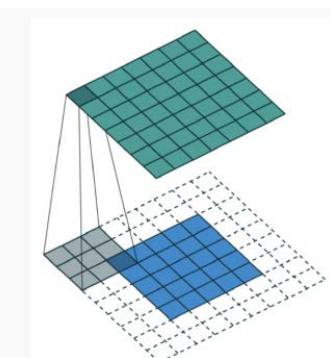


In this case, the **red weights** will be modified to better recognize cats

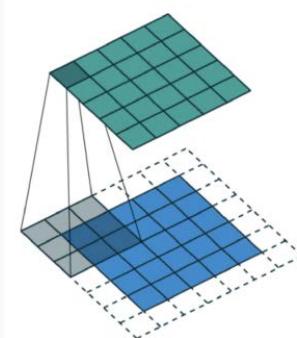


In this case, the **green weights** will be modified.

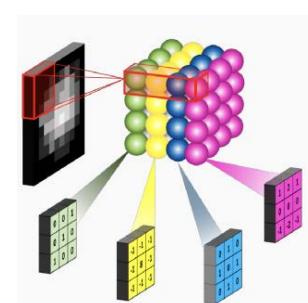
A cat detector using an MLP which changes as the position of the cat changes.



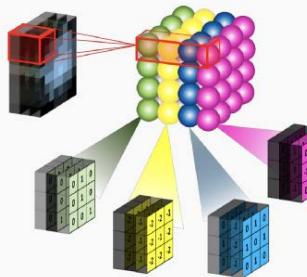
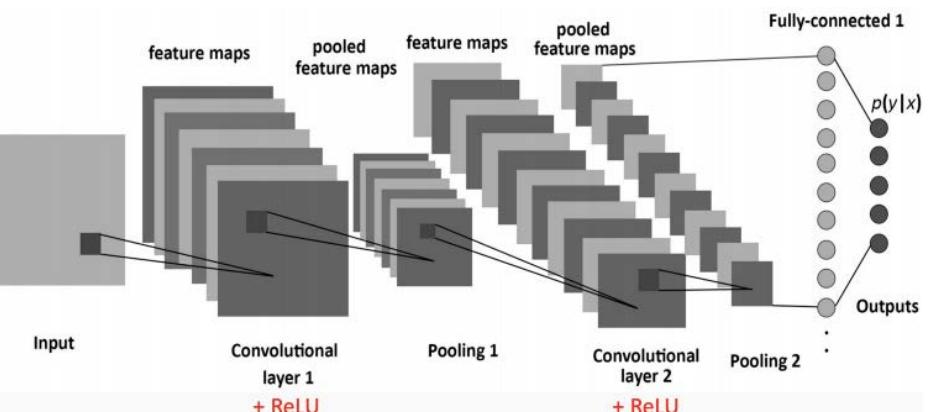
**Full padding.** Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



**Same padding.** Ensures that the output has the same size as the input.



Convolutional layer with four 3x3 filters on a **black and white image** (just one channel)



Convolutional layer with four 3x3 filters on an **RGB image**. As you can see, the filters are now cubes, and they are applied on the full depth of the image..



# convolution

## Convolutional Layers

- Action
- Apply filters to extract features
  - Filters are composed of small kernels, learned.
  - One bias per filter.
  - Apply activation function on every value of feature map

- Parameters
- Number of kernels
  - Size of kernels (W and H only, D is defined by input cube)
  - Activation function
  - Stride
  - Padding
  - Regularization type and value

- I/O
- Input: 3D cube, previous set of feature maps
  - Output: 3D cube, one 2D map per filter

## Pooling Layers

- Action
- Reduce dimensionality
  - Extract maximum or average of a region
  - Sliding window approach

- Parameters
- Stride
  - Size of window

- I/O
- Input: 3D cube, previous set of feature maps
  - Output: 3D cube, one 2D map per filter, reduced spatial dimensions

## Fully connected Layers

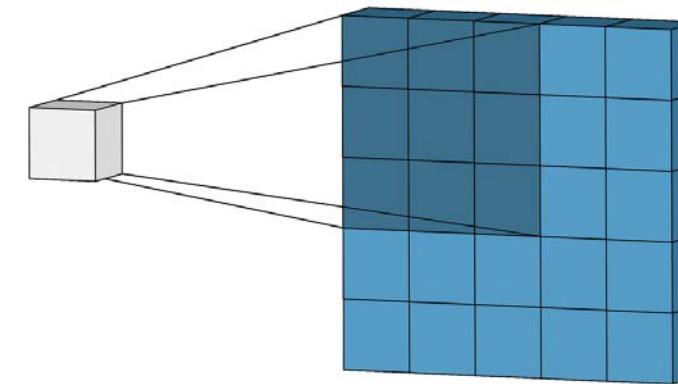
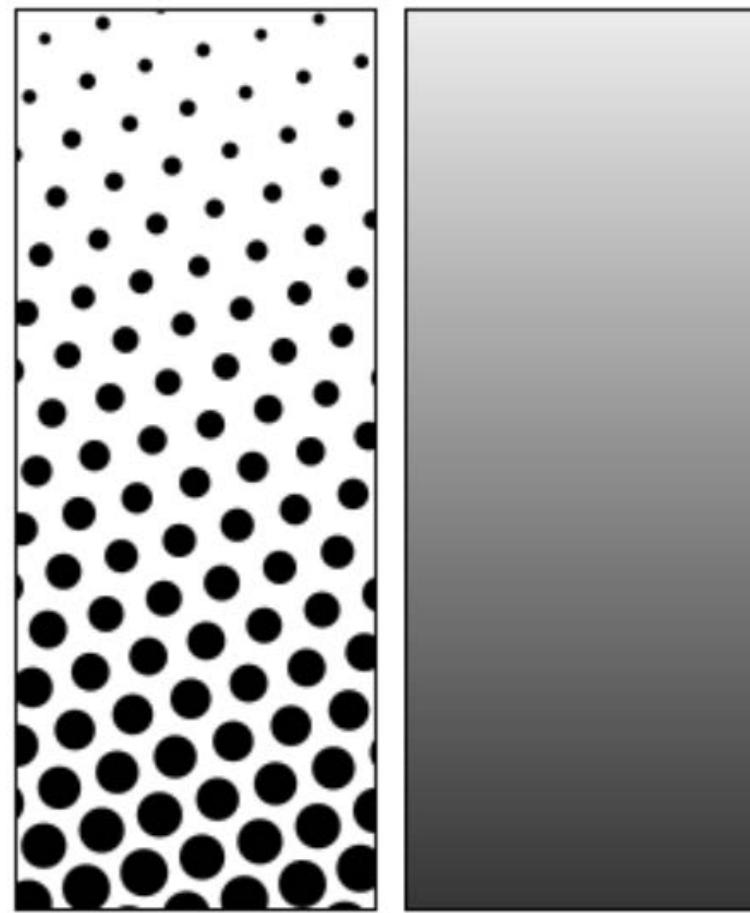
- Action
- Aggregate information from final feature maps
  - Generate final classification

- Parameters
- Number of nodes
  - Activation function: usually changes depending on role of layer. If aggregating info, use ReLU. If producing final classification, use Softmax.

- I/O
- Input: FLATTENED 3D cube, previous set of feature maps
  - Output: 3D cube, one 2D map per filter

<https://www.cs.ryerson.ca/~aharley/vis/conv/>

# convolution



1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2  
window and stride 2

6	8
3	4

Left: Halftone dots. Right: How the human eye would see this sort of arrangement from a sufficient distance.



# CNN (ConvNet)

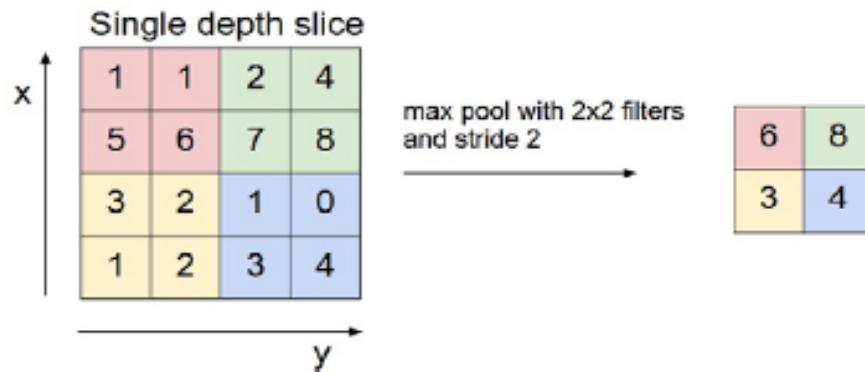
```
import numpy as np
np.random.seed(123)                                # for reproducibility
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, pooling
from keras.utils import np_utils
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1)
print(X_train.shape)
X_train = X_train.astype('float32') /255.
X_test = X_test.astype('float32') /255.               32 (3×3) filters
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)
model = Sequential()
model.add(Conv2D(32, 3, 3, activation='relu', input_shape=(28,28, 1)))
print(model.output_shape)
model.add(Conv2D(32, 3, 3, activation='relu'))
model.add(pooling.MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=32, nb_epoch=10, verbose=1)
score = model.evaluate(X_test, Y_test, verbose=0)
print(model.metrics_names)
print(score)
```

Use CNNs For:  
Image data  
Classification prediction problems  
Regression prediction problems  
More generally, CNNs work well with data that has a spatial relationship.

# Pooling layers

- Max pool

```
keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')
```

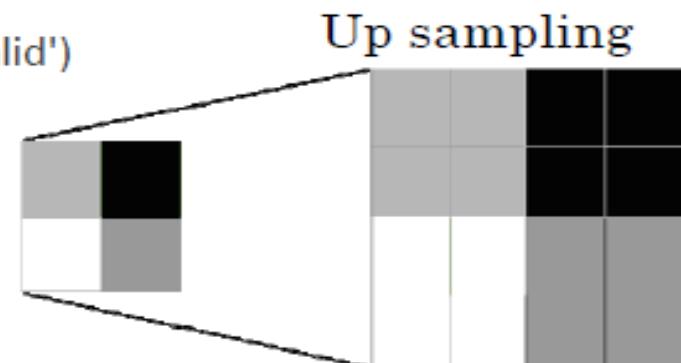


- Average pool

```
keras.layers.pooling.AveragePooling2D(pool_size=(2, 2), strides=None, padding='valid')
```

- Up sampling

```
keras.layers.convolutional.UpSampling2D(size=(2, 2))
```



*Two reasons for applying Max Pooling :*

1. **Downscaling Image** by extracting most important feature
2. **Keeping Invariances** like shift, rotational and scale



# Time series

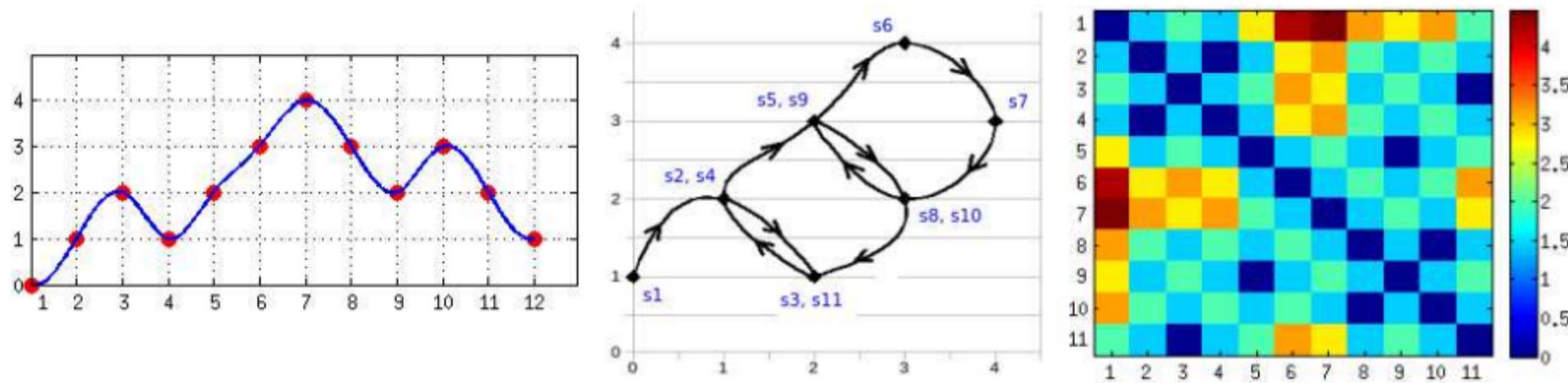


Figure 1: From time-series signal to recurrence plot. Left: A simple example of time-series signal ( $x$ ) with 12 data points. Middle: The 2D phase space trajectory is constructed from  $x$  by the time delay embedding ( $\tau = 1$ ). States in the phase space are shown with bold dots:  $s_1 : (x_1, x_2), s_2 : (x_2, x_3), \dots, s_{11} : (x_{11}, x_{12})$ . Right: The recurrence plot  $R$  is a  $11 \times 11$  square matrix with  $R_{i,j} = \text{dist}(s_i, s_j)$ .

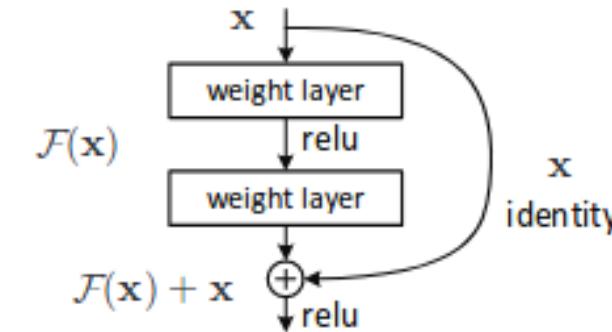


# Residual Network, ResNet

```
# example of a CNN model with an identity or projection residual module
from keras.models import Model
from keras.layers import Input
from keras.layers import Activation
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import add
from keras.utils import plot_model

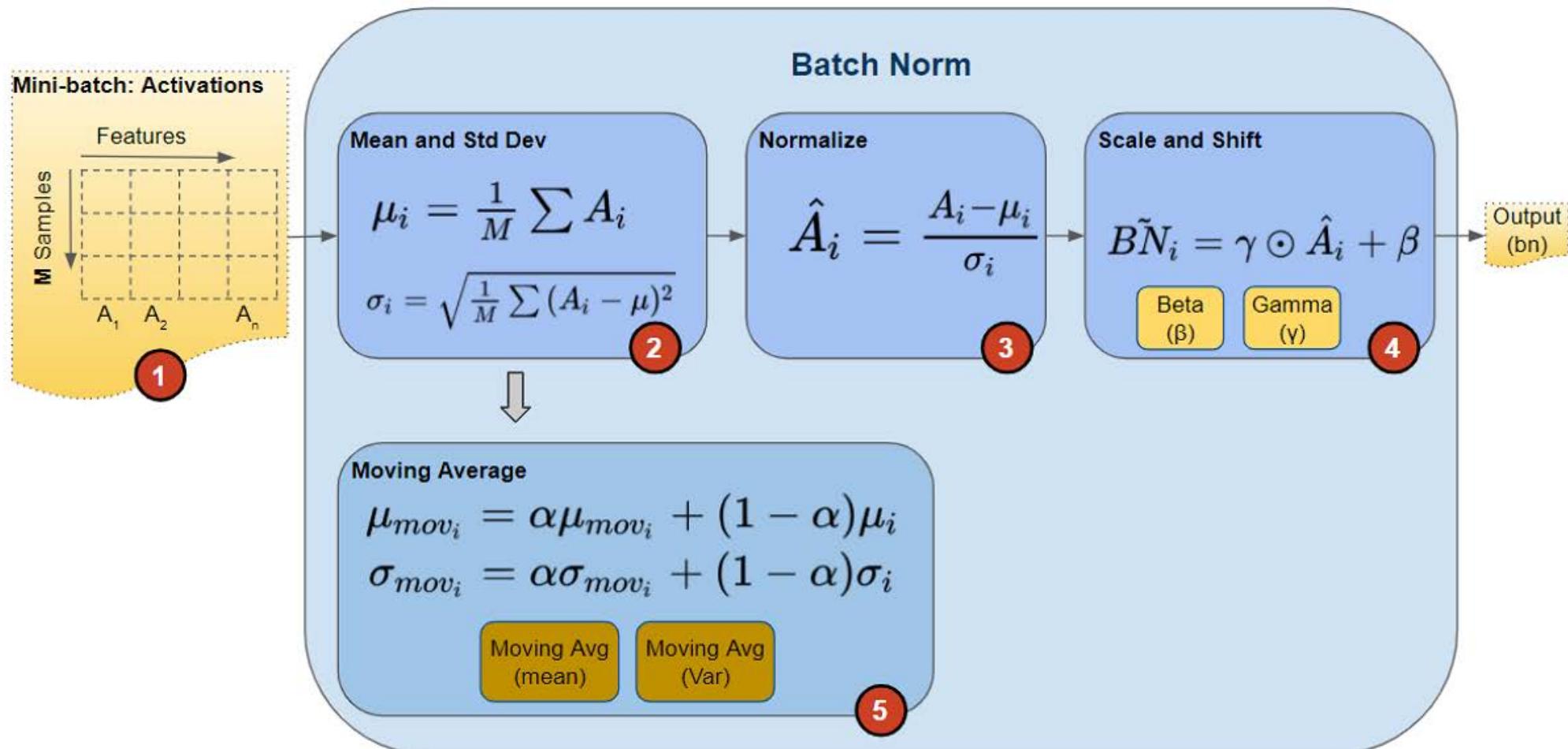
# function for creating an identity or projection residual module
def residual_module(layer_in, n_filters):
    merge_input = layer_in
    # check if the number of filters needs to be increase, assumes channels last format
    if layer_in.shape[-1] != n_filters:
        merge_input = Conv2D(n_filters, (1,1), padding='same', activation='relu', kernel_initializer='he_normal')(layer_in)
    # conv1
    conv1 = Conv2D(n_filters, (3,3), padding='same', activation='relu', kernel_initializer='he_normal')(layer_in)
    # conv2
    conv2 = Conv2D(n_filters, (3,3), padding='same', activation='linear', kernel_initializer='he_normal')(conv1)
    # add filters, assumes filters/channels last
    layer_out = add([conv2, merge_input])
    # activation function
    layer_out = Activation('relu')(layer_out)
    return layer_out

# define model input
visible = Input(shape=(256, 256, 3))
# add vgg module
layer = residual_module(visible, 64)
# create model
model = Model(inputs=visible, outputs=layer)
# summarize model
model.summary()
# plot model architecture
plot_model(model, show_shapes=True, to_file='residual_module.png')
```



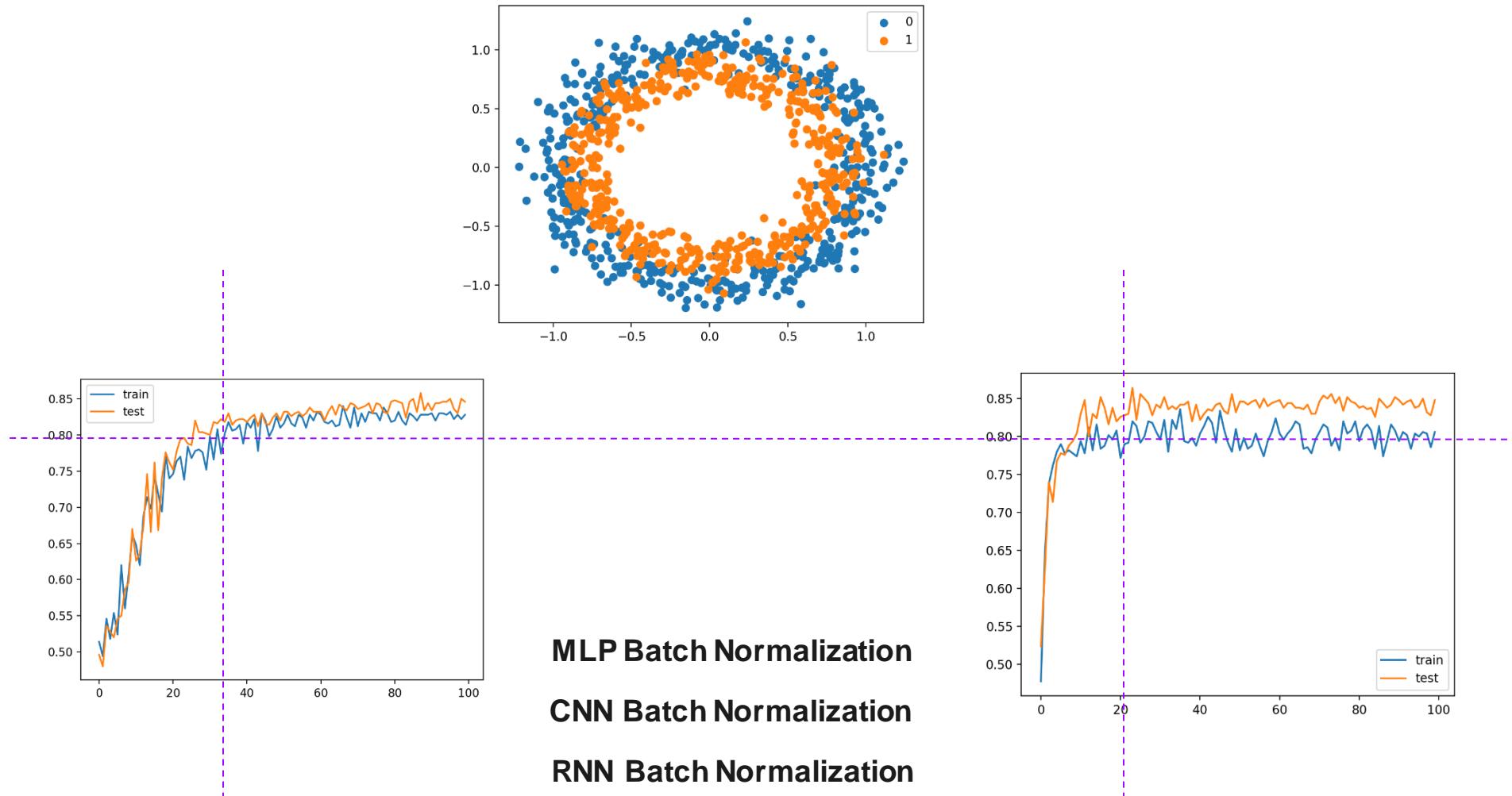


# Batch normalization





# Batch normalization



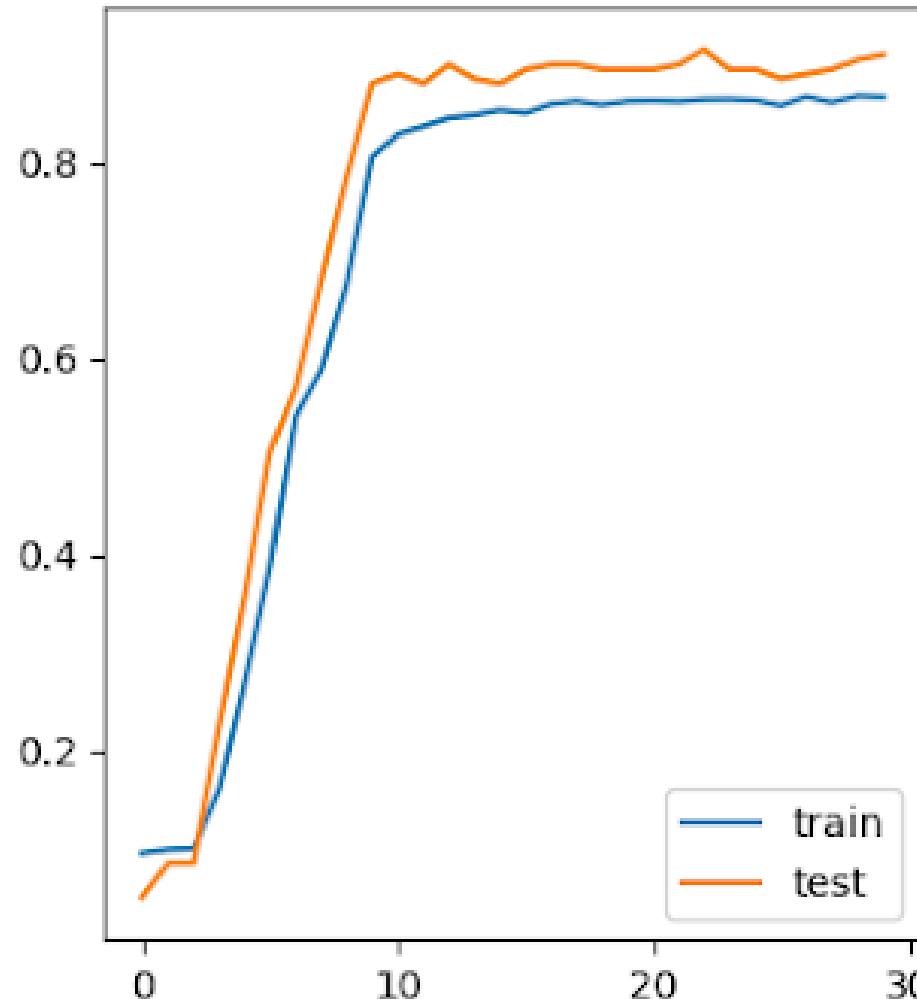
Batch normalization is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network.

Specifically, we can see that classification accuracy on the train and test datasets leaps above 80% within the first 20 epochs, as opposed to 30-to-40 epochs in the model without batch normalization.

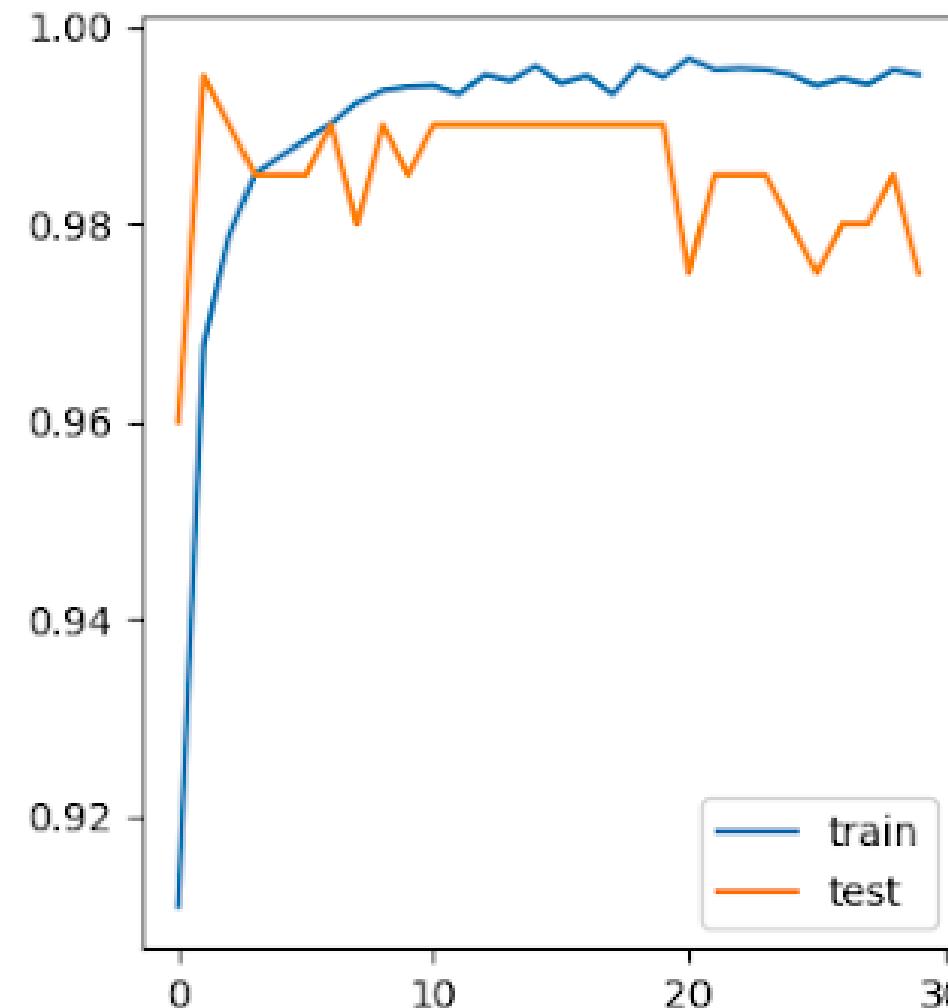


# Batch normalization

Train without Batch Normalization



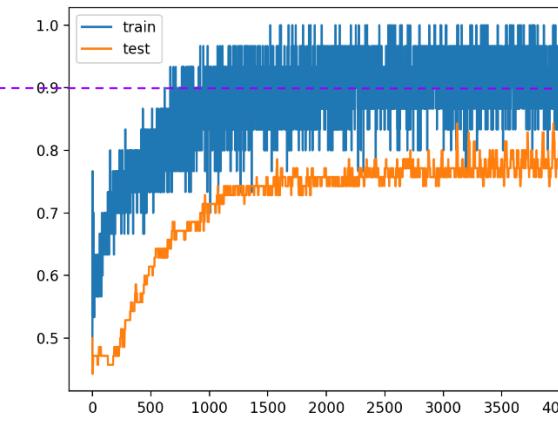
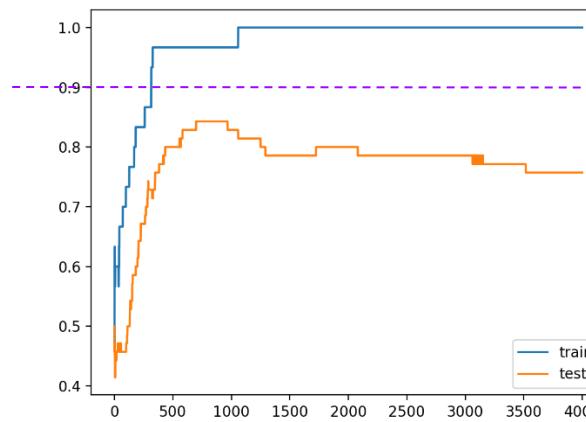
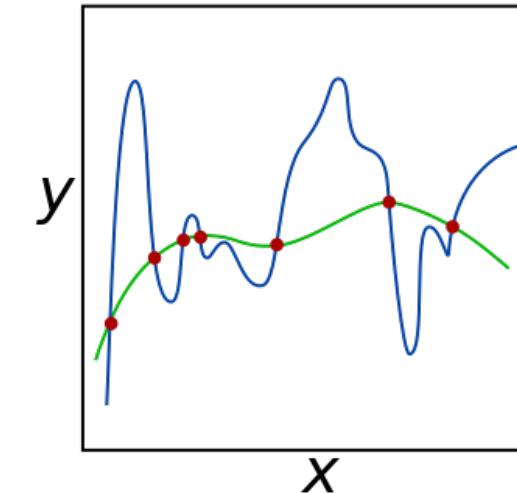
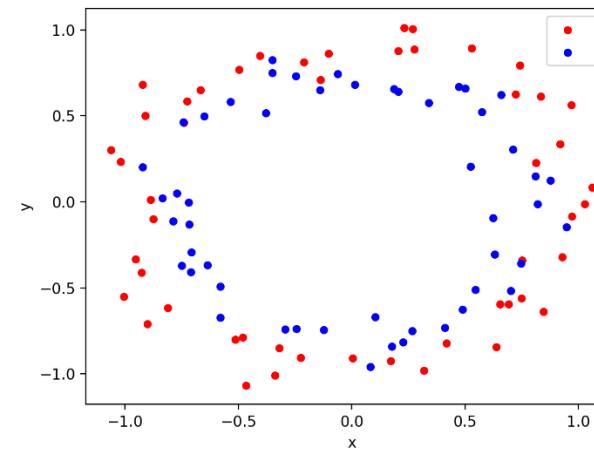
Train with Batch Normalization





# Dropout regularization

**Dropout(rate=0.1, seed=2020)**



We have only generated 100 samples, which is small for a neural network, providing the opportunity to overfit the training dataset and have higher error on the test dataset: a good case for using regularization. Further, the samples have noise, giving the model an opportunity to learn aspects of the samples that don't generalize.



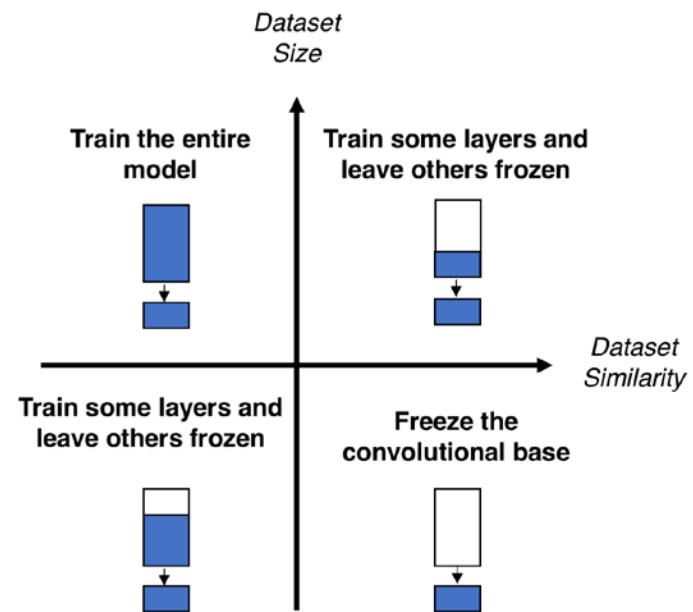
# Keras built-in VGG-16 net module: Transfer learning

```
from keras.applications.vgg16 import VGG16
import cv2
```

```
from keras.applications.inception_v3 import InceptionV3
```

## Convolution layers

1D  
2D  
3D



Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-



# Keras built-in MobileNetV2 module: Transfer learning

```
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,  include_top=False, weights='imagenet')
base_model.trainable = False
base_model.summary()
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
prediction_layer = keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
model = tf.keras.Sequential([ base_model, global_average_layer, prediction_layer])
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate), loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),  metrics=['accuracy'])
model.summary()
len(model.trainable_variables)
initial_epochs = 10
validation_steps=20
loss0,accuracy0 = model.evaluate(validation_batches, steps = validation_steps)
history = model.fit(train_batches, epochs=initial_epochs, validation_data=validation_batches)

base_model.trainable = True ; print("Number of layers in the base model: ", len(base_model.layers))
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),optimizer=tf.keras.optimizers.RMSprop(lr=base_learning_rate/10),  metrics=['accuracy'])
model.summary() ; len(model.trainable_variables) ; fine_tune_epochs = 10 ; total_epochs = initial_epochs + fine_tune_epochs
history_fine = model.fit(train_batches, epochs=total_epochs, initial_epoch = history.epoch[-1], validation_data=validation_batches)
```

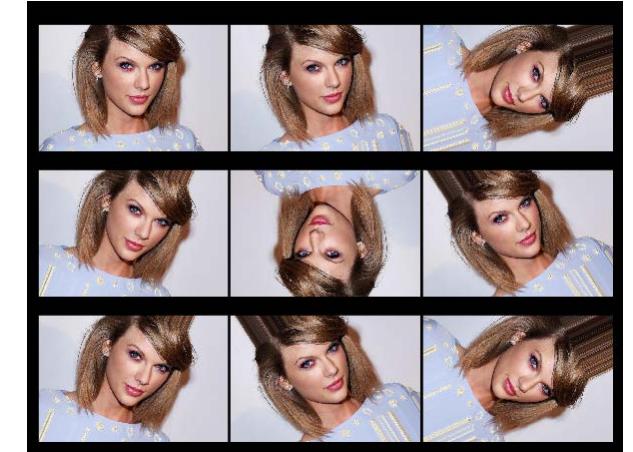


multi-class classification의 경우, 5종 분류인 경우, 폴더 5개를 만들어서 준비한다.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import np_utils
from keras.models import Sequential
from keras.preprocessing.image import array_to_img, img_to_array, load_img
from keras.layers import *
from keras.callbacks import ModelCheckpoint, EarlyStopping
import numpy as np
from sklearn.metrics import accuracy_score

trainDataGen = ImageDataGenerator(rescale=1./255, rotation_range
=30, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.2, zoom_range=0.2, horizontal_flip=False, vertical_flip=False, fill_mode='nearest')
trainGenSet = trainDataGen.flow_from_directory(path + 'train', batch_size=32, target_size=(28,28), class_mode='categorical')

model = Sequential()
model.add(Conv2D(64, kernel_size=(3,3), padding='same', input_shape=(28,28,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(5, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit_generator( trainGenSet, steps_per_epoch=20, epochs=200, validation_data=testGenSet, validation_steps=10)
scores = model.evaluate_generator(testGenSet)
print(scores)
```



```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Binary classification의 경우, 2종 분류인 경우, 폴더 2개를 만들어서 준비한다.

```
datagen = ImageDataGenerator(
```

```
    samplewise_center=True, # set each sample mean to 0
    rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1, # Randomly zoom image
    width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
    horizontal_flip=True, # randomly flip images
    vertical_flip=False) # we don't expect Bo to be upside-down so we will not flip vertically
```

```
# load and iterate training dataset
```

```
train_it = datagen.flow_from_directory('./COVID/train/',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=8)
```

```
# load and iterate validation dataset
```

```
valid_it = datagen.flow_from_directory('./COVID/test/',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='binary',
    batch_size=8)
```

```
batch_size = 16
# 학습 이미지에 적용한 augmentation 인자를 지정해줍니다.
train_datagen = ImageDataGenerator( rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
# 검증 및 테스트 이미지는 augmentation을 적용하지 않습니다. 모델 성능을 평가할 때에는 이미지 원본을 사용합니다.
validation_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
# 이미지를 배치 단위로 불러와 줄 generator입니다.
train_generator = train_datagen.flow_from_directory( `data/train` ,                                     # this is the target directory
target_size=(150, 150),                                         # 모든 이미지의 크기가 150x150로 조정됩니다.
batch_size=batch_size, class_mode='binary' )
# binary_crossentropy 손실 함수를 사용하므로 binary 형태로 라벨을 불러와야 합니다.
validation_generator = validation_datagen.flow_from_directory( `data/validation` , target_size=(150, 150), batch_size=batch_size, class_mode='binary' )
test_generator = test_datagen.flow_from_directory( `data/validation` , target_size=(150, 150), batch_size=batch_size, class_mode='binary' )
```



# RNN, LSTM

[10, 20, 30, 40, 50, 60, 70, 80, 90]

X,	y
10, 20, 30	40
20, 30, 40	50
30, 40, 50	60
...	

Use RNNs For:  
Text data  
Speech data  
Classification prediction problems  
Regression prediction problems  
Generative models

Don't Use RNNs For:  
Tabular data  
Image data

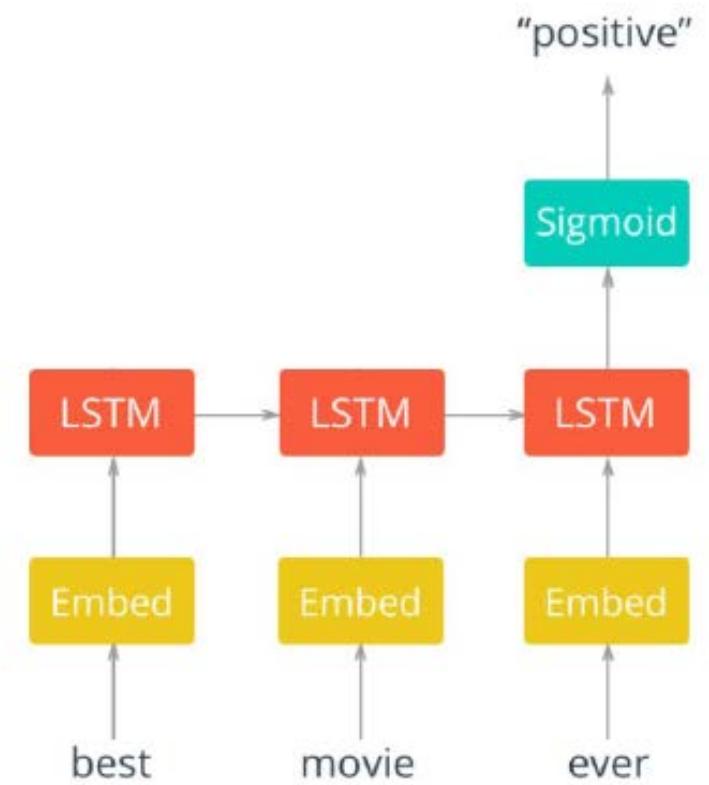
## LSTM Models for Time Series Forecasting

Long Short-Term Memory networks (LSTMs) can be applied to time series forecasting.  
There are many types of LSTM models that can be used for each specific type of time series forecasting problem.



# RNN, LSTM

- RNNs are used on sequential data – Text, Audio, Genomes etc.
- Recurrent networks are of three types
- Vanilla RNN
- LSTM
- GRU
- They are feedforward networks with internal feedback
- The output at time “t” is dependent on current input and previous values



**Generative models : RNN, GAN, VAE**

**Discriminant models : Regression, Classification**



# Popular Deep learning Architectures

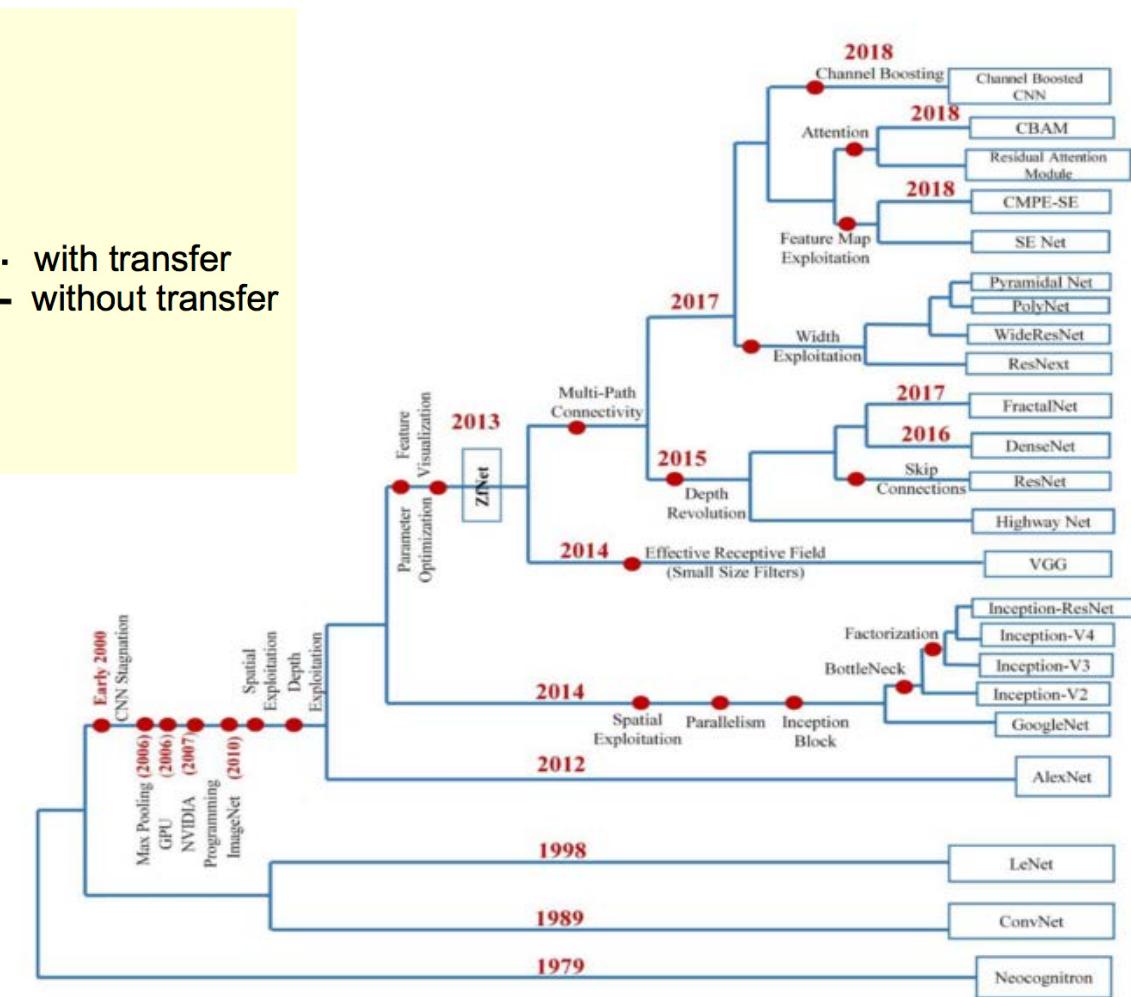
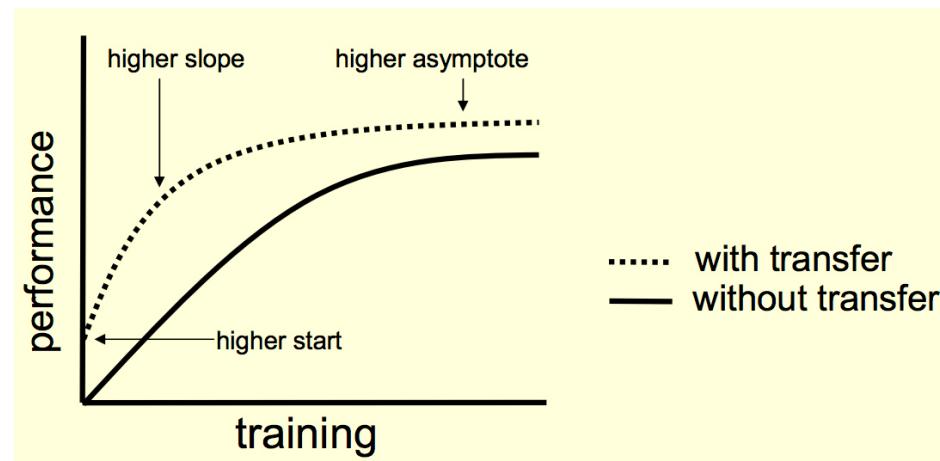
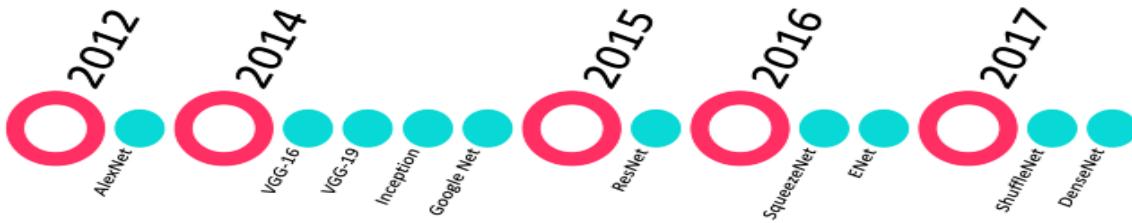
---

- Popular Convolution networks
  - Alex net
  - VGG
  - Res-Net
  - DenseNet
- Generative models
  - Autoencoders
  - Generative adversarial networks



# Evolutionary history of deep CNNs

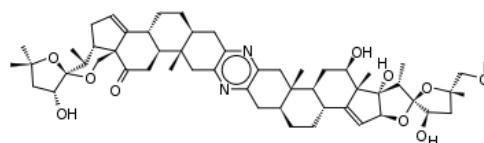
LeNet-5 (1990s)  
 AlexNet (2012)  
 VGG-16 (2014)  
 VGG-19  
 Inception and GoogLeNet  
 ResNet (2016)  
 Squeeze Net  
 DenseNet (2017)  
 Shuffle Net  
 ENet





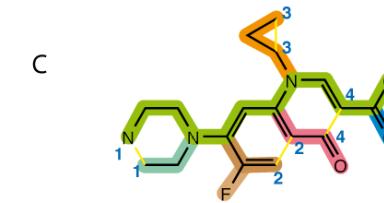
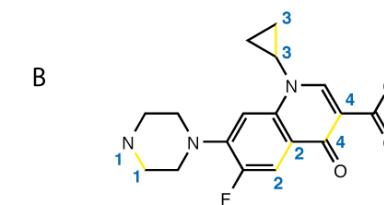
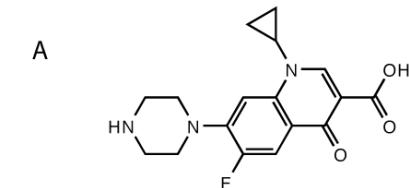
# SMILES

- Simplified molecular-input line-entry system ([SMILES](#))
  - SMILES → 2D drawings, 3D models
  - SYBYL line notation



Starting with the left-most methyl group in the figure

CC(C)(O)C[C@H](O)[C@H](O)[C@H](O)(C)[C@H](O)3CC=C4[C@H]3(C2C)(=O)C[C@H]5[C@H]4CC[C@H](O)C6[C@H]5(C)Cc(n7)c6nc(C[C@H]89(C)c7C[C@H]8CC[C@H]9[C@H](O)[C@H]9[C@H]10(C)C[C@H]10=C[C@H](O)X12)[C@H]9(C)[C@H]12(O)X13)[O@H]X13(C)C9

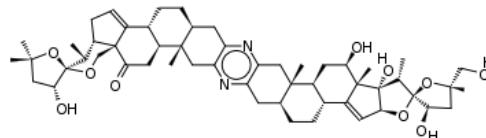


$$\text{N1CCN(CC1)C(C(F)=C2)=CC(=C2C4=O)N(C3CC3)C=C4C(=O)C}$$



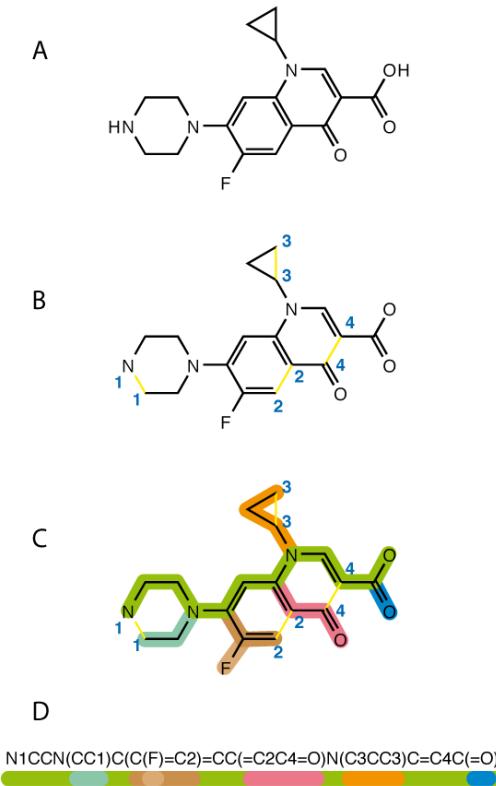
# SMILES

- Simplified molecular-input line-entry system (**SMILES**)
- SMILES → 2D drawings, 3D models
- SYBYL line notation



Starting with the left-most methyl group in the figure:

```
CC(C)(O1)C[C@H](O)[C@H]1O[C@H]1(C)[C@H]3C=C4[C@H]3[C2]C(=O)C[C@H]5[C@H]4CC[C@H](C6)[C@H]5(C)Cc(n7)c6nc(C[C@H]89(C)c7C[C@H]8CC[C@H]9C[C@H](O)[C@H]x11(C)Cx10=C[C@H](Ox12)[C@H]x11(O)[C@H](C)[C@H]x12(Ox13)[C@H](O)C[C@H]x13(C)CO
```



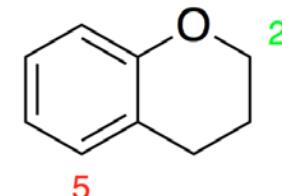


# to be or not to be that is the question

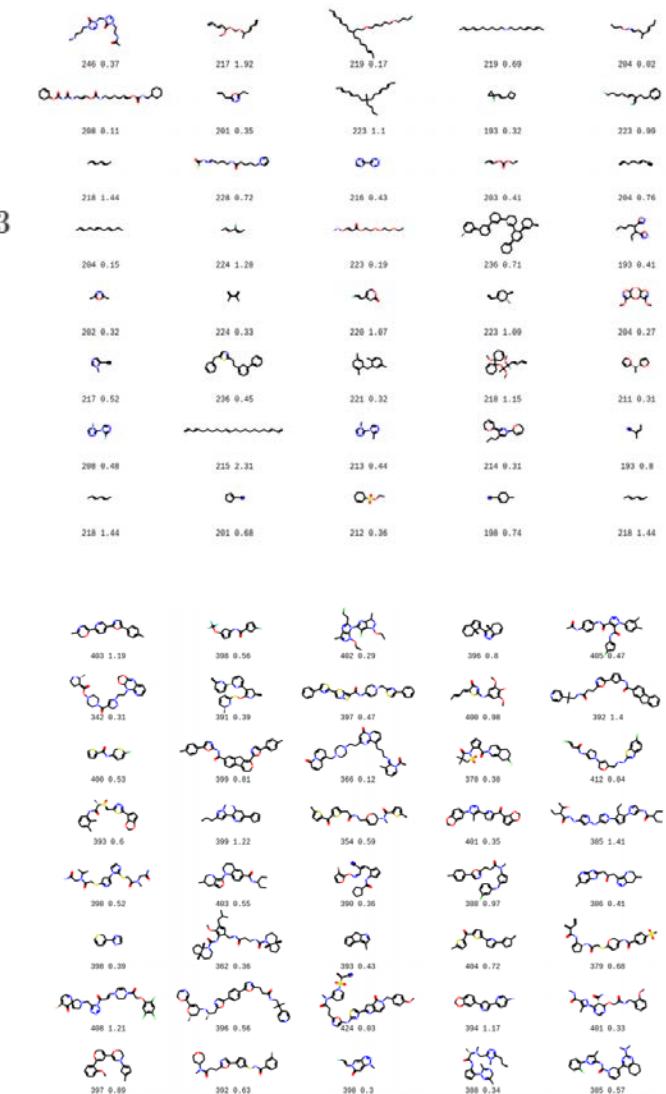
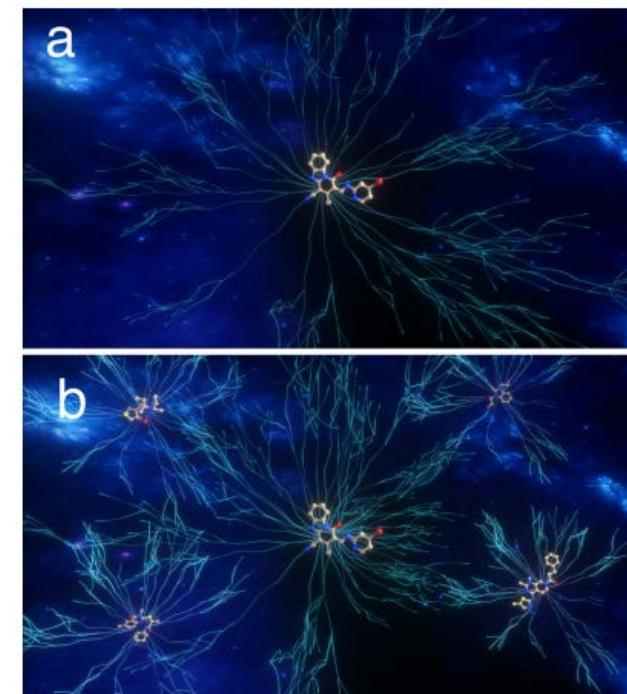
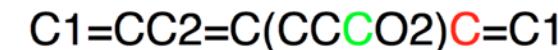
$$27^{39} = 6.7 \times 10^{55}$$

-7.0 dczrwbcupkohijqqyefthfibousoxquyjtvha  
 -8.0 yoyg wybynsw ttba ftcmo lbdgrkojheltrjl  
 -10.0 treweef gndtwt anb xbmtdimphxeujisqtqri  
 -11.0 treweefagndtwt ajb ybyt isbhxwktisqkqod  
 -12.0 mo weeftfndtnttaj tbyt ibbhfwknwsqtqns  
 -13.0 pppua ga zoh tox c fvdr sm cjpnuwekdioe  
 -15.0 pogbf opoxwkhponhg hlatoyj theuqusxeknn  
 -16.0 to oe gocnig novrektxdjurrvthk zhasfiou  
 -18.0 xnvs w n t qyzvr i ft in ihp quvstiop  
 -19.0 xnvseu w ntt qyzvr i at is ihp quvstgop  
 -22.0 tv be grnna qt xqihr atrismthekqunstion  
 -23.0 to se grcnatqt xji rgatqwsmtthequestqon  
 -25.0 do ne or notqtaaqzotyat iefthe qmfstiin  
 -27.0 so be or nob ko rejtdyt ws the phestunn  
 -28.0 ro be or notgjodbgotvatfisvh question  
 -30.0 to be gw nob to re tdat ws the phestinn  
 -31.0 rozbe or notnto kvothat isvjhe question  
 -32.0 to be orgnot to ke thltcisvdhe questiod  
 -33.0 to be vrgnot to be thltcis dhe questiod  
 -34.0 ro be orvnot to je thah isbthe question  
 -35.0 to be or notpto re that is thesquestiof  
 -36.0 to be or notpgo be that is the questiof  
 -37.0 to be or notgto be that isvthe question  
 -38.0 to be or not to be that isvthe question  
 -39.0 **to be or not to be that is the question**

$$10^{60} \rightarrow 10^3 \rightarrow 1$$



$$e^{-\frac{1}{2}\left(\frac{\lambda - \lambda_t}{\sigma}\right)^2} + \min(\omega, 0.3)/0.3$$



```

import numpy as np
from scipy.optimize import dual_annealing
from scipy.optimize import minimize
def num2abc(i):
    char=''
    if i >= 0 and i < 26 :
        char = f'{chr(i + 97)}'
    return char
def abc2num(char):
    i=ord(char)-97
    if char == ' ' :
        i=26
    return i
def f(x):
#  astring="to be or not to be that is the question whether it is nobler in"
#  astring="to be or not to be that is the question"
    tmp=0.
    for i in range(len(x)):
        if abc2num(asrtng[i]) == int(x[i]) :
            tmp=tmp+1.
    return -tmp

```

```

def in_bounds(point, bounds):
    # enumerate all dimensions of the point
    for d in range(len(bounds)):
        # check if out of bounds for this dimension
        if point[d] < bounds[d, 0] or point[d] > bounds[d, 1]:
            return False
    return True

def csadistance(x,y):
    tmp=0.
    for i in range(len(x)):
        tmp=tmp+np.abs(x[i]-y[i])
    return tmp

def localquench(x):
    fx=f(x)
    if True:
        lw = [0] * len(x)
        up = [26] * len(x)
        x0=[x[i] for i in range(len(x))]
        ret = dual_annealing(f, bounds=list(zip(lw, up)), seed=1234,
no_local_search=False, maxiter=100,x0=x0)
        return ret.fun,ret.x

```

```

astring="to be or not to be that is the question"
#astring="to be or not to be"
ndim=len(astring)
if False:
    for i in range(ndim):
        print(astring[i],abc2num(astring[i]))
print(ndim)
npop=100
static=np.zeros((npop,ndim)) ; ostatic=np.zeros(npop) ;
xvector=np.zeros(ndim) ; yvector=np.zeros(ndim)
dynamic=np.zeros((npop,ndim)) ; odynamic=np.zeros(npop)
for i in range(npop):
    for j in range(ndim):
        static[i,j]=np.random.randint(27)
        xvector[:]=static[i,:]
        trial,xvector=localquench(xvector)
        ostatic[i]=trial ; static[i,:]=xvector[:]
        ind = ostatic.argsort() ; ostatic=ostatic[ind] ; static=static[ind,:]
        dynamic[:,:]=static[:,:] ; odynamic[:]=ostatic[:]
        print(odynamic)
        best=odynamic[0] ; davg=0. ; kount=0
    for i in range(npop):
        for j in range(npop):
            if i > j:
                davg=davg+csadistance(static[i,:],static[j,:]) ; kount=kount+1
    davg=davg/kount ; dcut=davg/2

```

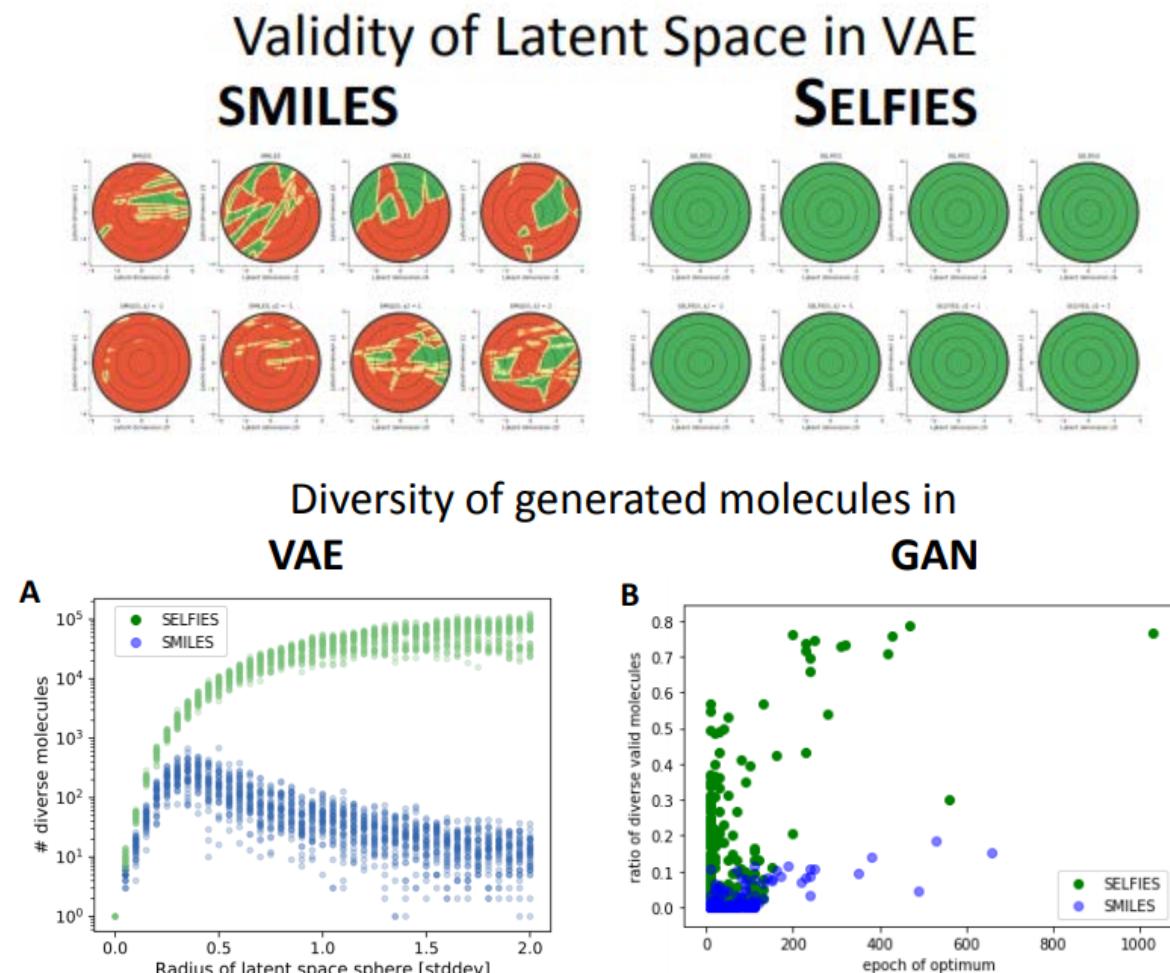
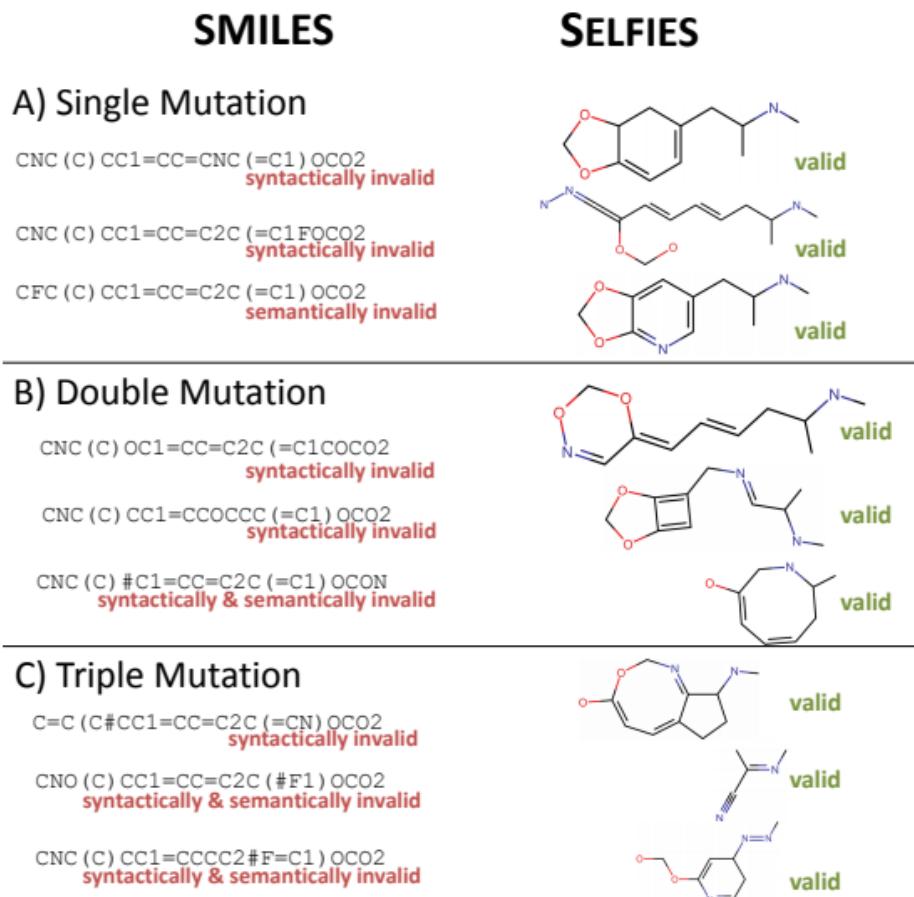
```

for kter in range(200*npop):
    k0=0 ; k1=0
    while k0==k1 :
        i=np.random.randint(npop) ; j=np.random.randint(npop) ; k0=j
        if i < j:
            k0=i
        i=np.random.randint(npop) ; j=np.random.randint(npop) ; k1=j
        if i < j:
            k1=i
        xvector[:,]=dynamic[k0,:]
        if np.random.random()< 0.1:
            xvector[:,]=static[k0,:]
        if np.random.random()> 0.5:
            tmp=np.random.random()/4.+0.1
            for i in range(ndim):
                if np.random.random() < tmp:
                    xvector[i]=np.random.randint(27)
        else:
            yvector[:,]=dynamic[k1,:]; tmp=np.random.random()/2.+0.1
            for i in range(ndim):
                if np.random.random() < tmp:
                    xvector[i]=yvector[i]
        trial,xvector=localquench(xvector)
        dsmall=1e99 ; i0=npop-1
        for i in range(npop):
            tmp=csadistance(xvector,dynamic[i,:])
            if dsmall > tmp:
                i0=i ; dsmall=tmp
        if dcut > dsmall :
            if odynamic[i0]>trial:
                odynamic[i0]=trial ; dynamic[i0,:]=xvector[:,]
            if False:
                print(trial,'old')
        else:
            i0=npop-1
            if odynamic[i0]>trial:
                odynamic[i0]=trial ; dynamic[i0,:]=xvector[:,]
            if False:
                print(trial,'new')
        ind=odynamic.argsort(); odynamic=odynamic[ind] ; dynamic=dynamic[ind,:]
        if best>odynamic[0] or kter==0:
            best=odynamic[0]; xvector[:,]=dynamic[0,:]
            bstring=""
            for i in range(ndim):
                bstring=bstring+num2abc(int(xvector[i]))
            print(best, bstring)
#            print(xvector)
        if kter > npop and np.mod(kter,npop) ==0:
            dcut=dcut*0.99
        if dcut < davg/5 :
            dcut=davg/5

```



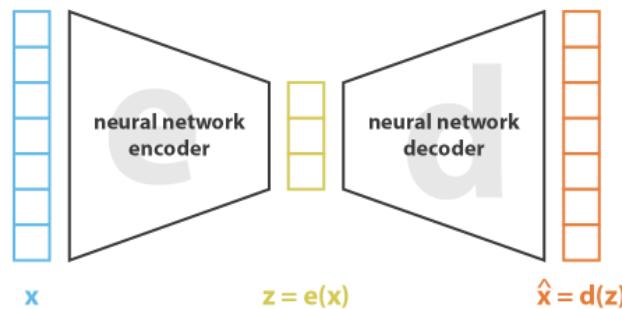
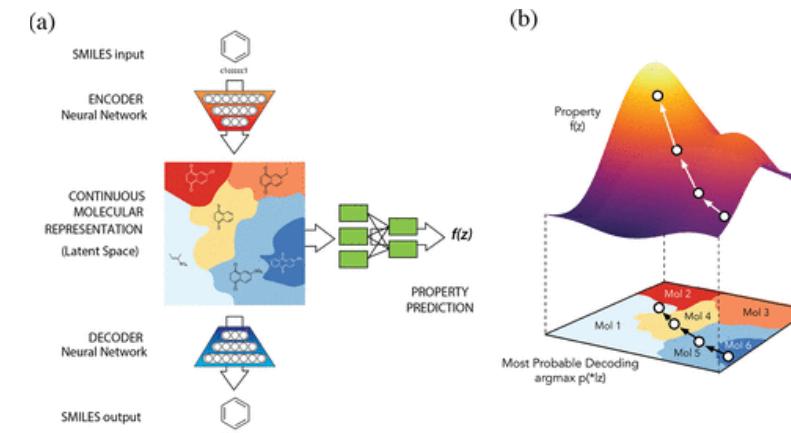
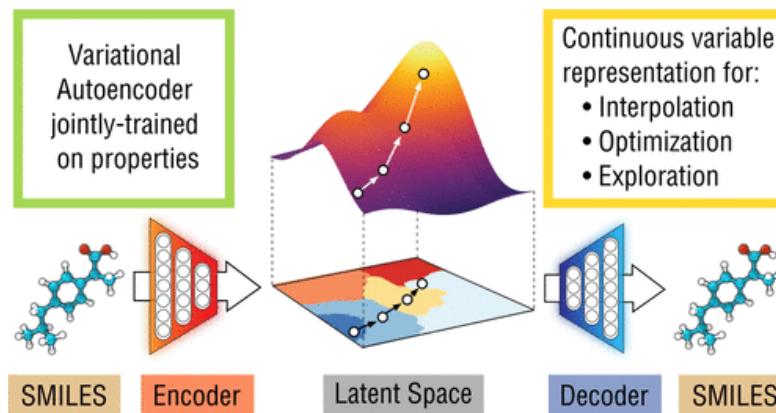
# Self-Referencing Embedded Strings (SELFIES): A 100% robust molecular string representation



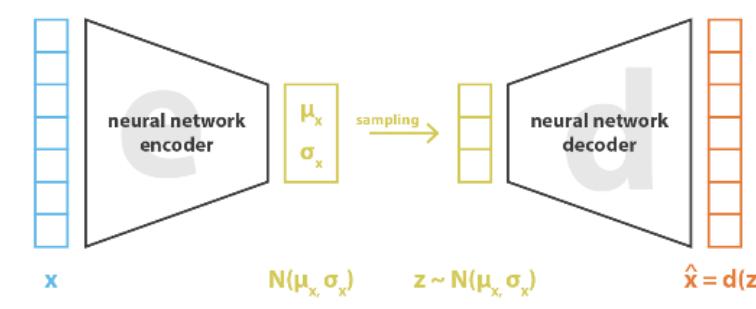


# Generative models

## Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$



$$\text{loss} = \| x - \hat{x} \|^2 + \text{KL}[ N(\mu_x, \sigma_x), N(0, I) ] = \| x - d(z) \|^2 + \text{KL}[ N(\mu_x, \sigma_x), N(0, I) ]$$



# Autoencoder (AE)

```

from keras.layers import Input, Dense
from keras.models import Model
# this is the size of our encoded representations
encoding_dim = 32 # 32 floats -> compression of factor 24.5, assuming the input is
784 floats, 28*28/32 = 24.5
# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
# create a placeholder for an encoded 32D input
encoded_input = Input(shape=(encoding_dim,))
# retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape(len(x_train), np.prod(x_train.shape[1:]))
x_test = x_test.reshape(len(x_test), np.prod(x_test.shape[1:]))
print(x_train.shape)
print(x_test.shape)
autoencoder.fit(x_train, x_train, epochs=50, batch_size=256, shuffle=True,
validation_data=(x_test, x_test))
# encode and decode some digits
# note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

```

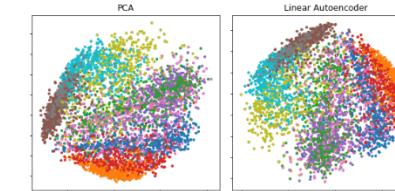
```

# use matplotlib
import matplotlib.pyplot as plt
n = 10 # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

```

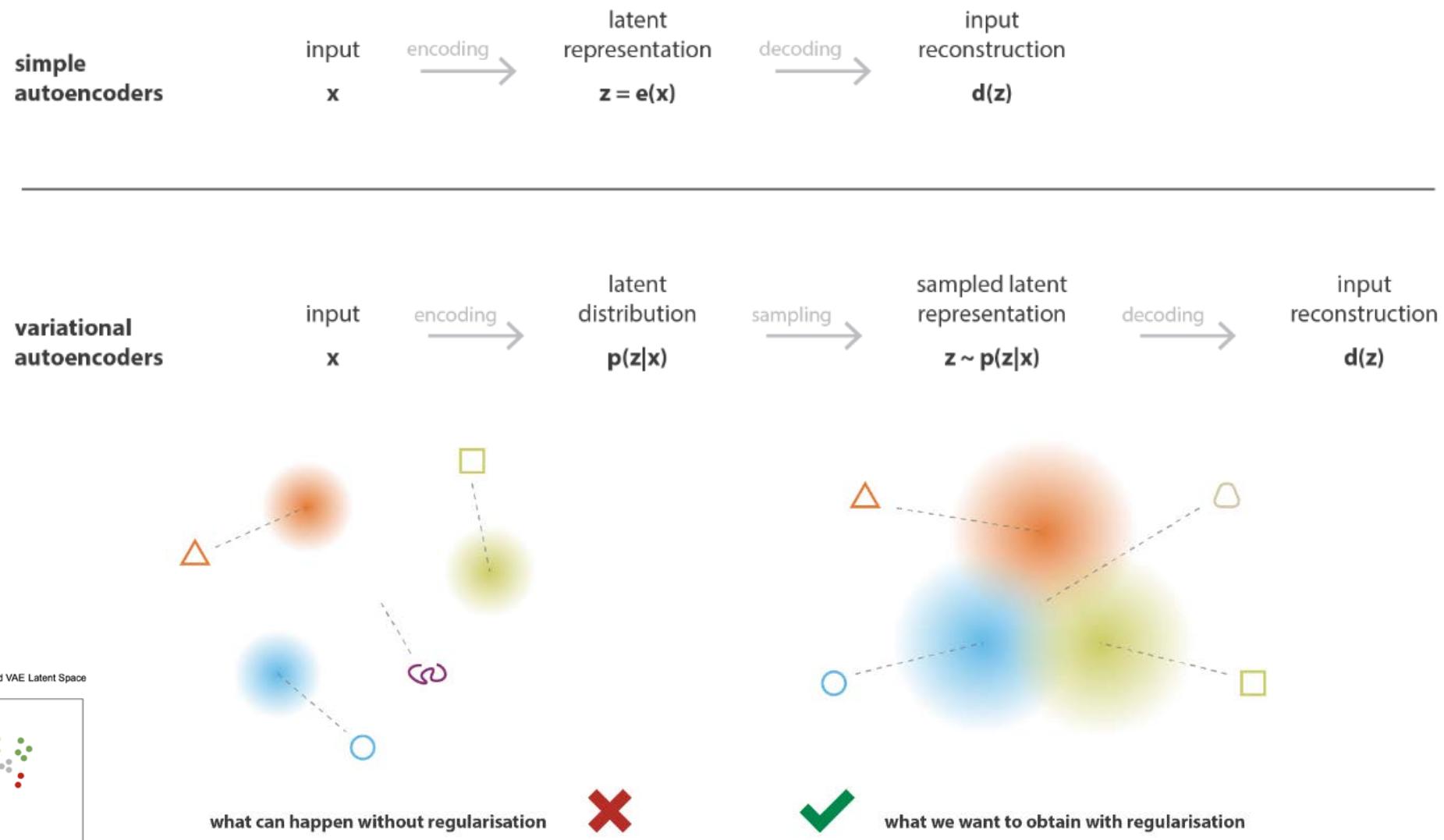
## Denoising AE



There exist several methods to design forms with fields to be surrounded by bounding boxes, by light rectangles or methods specify where to write and, therefore, minimize the effort with other parts of the form. These guides can be located on a separate sheet or much better from the top of view of the user but it's not always appropriate and, more importantly, this type of acquisition is used. Guiding rulers printed on the used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, be taken into account: The best way to print these light rectan;



# AE and VAE





# Variational autoencoder (VAE)

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from keras import backend as K
from keras.layers import Input, Dense, Lambda, Layer, Add, Multiply
from keras.models import Model, Sequential
from keras.datasets import mnist
original_dim = 784
intermediate_dim = 256
latent_dim = 2
epsilon_std = 1.0
def nll(y_true, y_pred):
    """ Negative log likelihood (Bernoulli). """
    # keras.losses.binary_crossentropy gives the mean
    # over the last axis. we require the sum
    return K.sum(K.binary_crossentropy(y_true, y_pred), axis=-1)
class KLDivergenceLayer(Layer):
    """ Identity transform layer that adds KL divergence to the final model loss. """
    def __init__(self, *args, **kwargs):
        self.is_placeholder = True
        super(KLDivergenceLayer, self).__init__(*args, **kwargs)
    def call(self, inputs):
        mu, log_var = inputs
        kl_batch = - .5 * K.sum(1+log_var-K.square(mu)-K.exp(log_var), axis=-1)
        self.add_loss(K.mean(kl_batch), inputs=inputs)
        return inputs
decoder = Sequential([
    Dense(intermediate_dim, input_dim=latent_dim, activation='relu'),
    Dense(original_dim, activation='sigmoid') ])
x = Input(shape=(original_dim,))
h = Dense(intermediate_dim, activation='relu')(x)
z_mu = Dense(latent_dim)(h)
z_log_var = Dense(latent_dim)(h)
z_mu, z_log_var = KLDivergenceLayer()([z_mu, z_log_var])
z_sigma = Lambda(lambda t: K.exp(.5*t))(z_log_var)
eps = Input(tensor=K.random_normal(stddev=1.,shape=(K.shape(x)[0], latent_dim)))
z_eps = Multiply()([z_sigma, eps])
z = Add()([z_mu, z_eps])
x_pred = decoder(z)
vae = Model(inputs=[x, eps], outputs=x_pred)
vae.compile(optimizer='rmsprop', loss=nll)
# train the VAE on MNIST digits
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(-1, original_dim) / 255.
x_test = x_test.reshape(-1, original_dim) / 255.
vae.fit(x_train, x_train, shuffle=True, epochs=50, batch_size=100, validation_data=(x_test, x_test))
encoder = Model(x, z_mu)
# display a 2D plot of the digit classes in the latent space
z_test = encoder.predict(x_test, batch_size=100)
plt.figure(figsize=(6, 6))
plt.scatter(z_test[:, 0], z_test[:, 1], c=y_test, alpha=.4, s=3**2, cmap='viridis')
plt.colorbar()
plt.show()
# display a 2D manifold of the digits
n = 15 # figure with 15x15 digits
digit_size = 28
# linearly spaced coordinates on the unit square were transformed
# through the inverse CDF (ppf) of the Gaussian to produce values
# of the latent variables z, since the prior of the latent space is Gaussian
u_grid = np.dstack(np.meshgrid(np.linspace(0.05, 0.95, n), np.linspace(0.05, 0.95, n)))
z_grid = norm.ppf(u_grid)
x_decoded = decoder.predict(z_grid.reshape(n*n, 2))
x_decoded = x_decoded.reshape(n, n, digit_size, digit_size)
plt.figure(figsize=(10, 10))
plt.imshow(np.block(list(map(list, x_decoded))), cmap='gray')
plt.show()

```



input image



reproduced image  
(2D latent space)



reproduced image  
(20D latent space)

encoder, decoder, and predictor

<https://arxiv.org/abs/1312.6114>

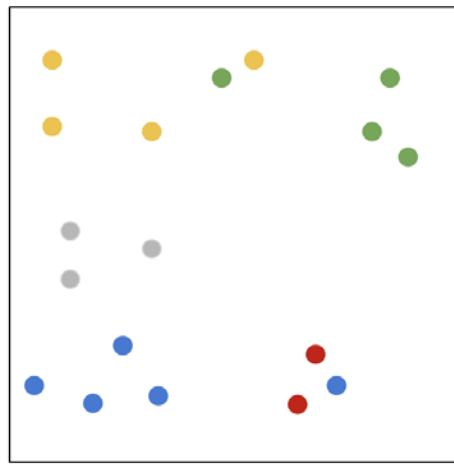
<https://blog.keras.io/building-autoencoders-in-keras.html>

[https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB20\\_CXVII-Keras\\_VAE\\_ising.html](https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB20_CXVII-Keras_VAE_ising.html)

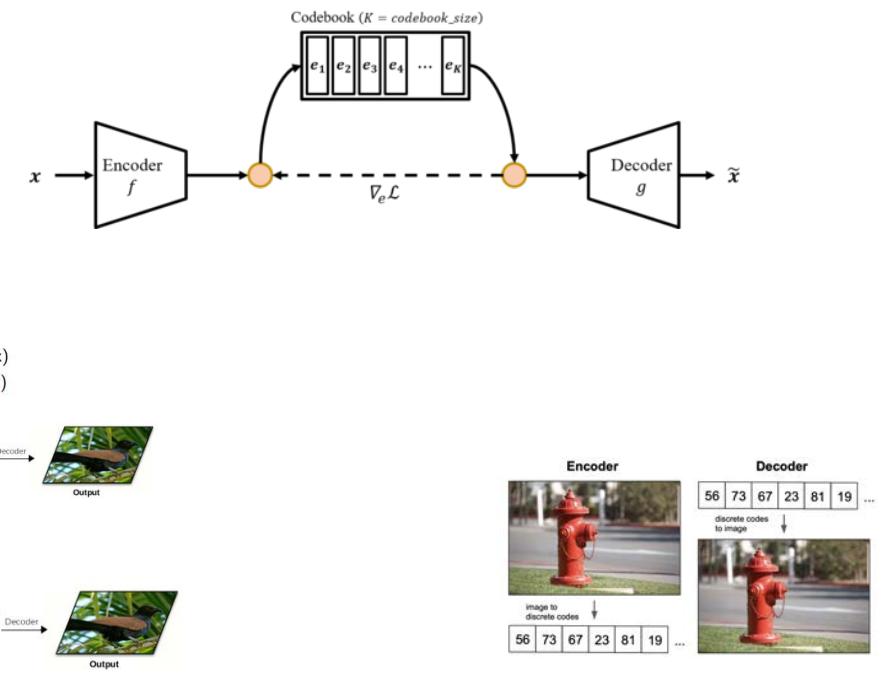
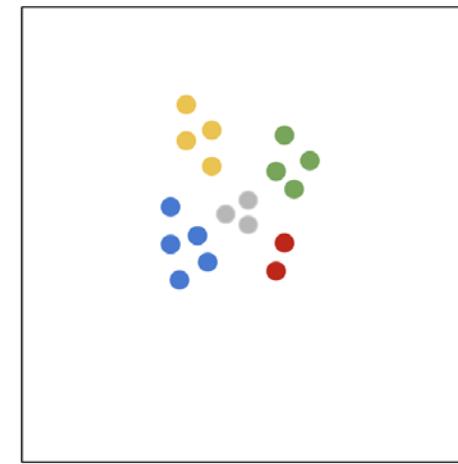


# VAE and VQ-VAE

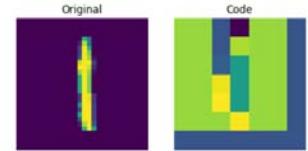
Messy Autoencoder Latent Space



Well Distributed VAE Latent Space



VQ-VAEs can represent diverse, complex data distributions better than pretty much any other algorithm out currently. And since these models play so nicely with transformers, the generative possibilities can be scaled almost arbitrarily given a large enough compute budget (unfortunately, for state of the art results, this is a budget that very few individuals or even organizations can afford). For these reasons, I expect VQ-VAEs to remain a popular component in the deep learning ecosystem for quite a while.



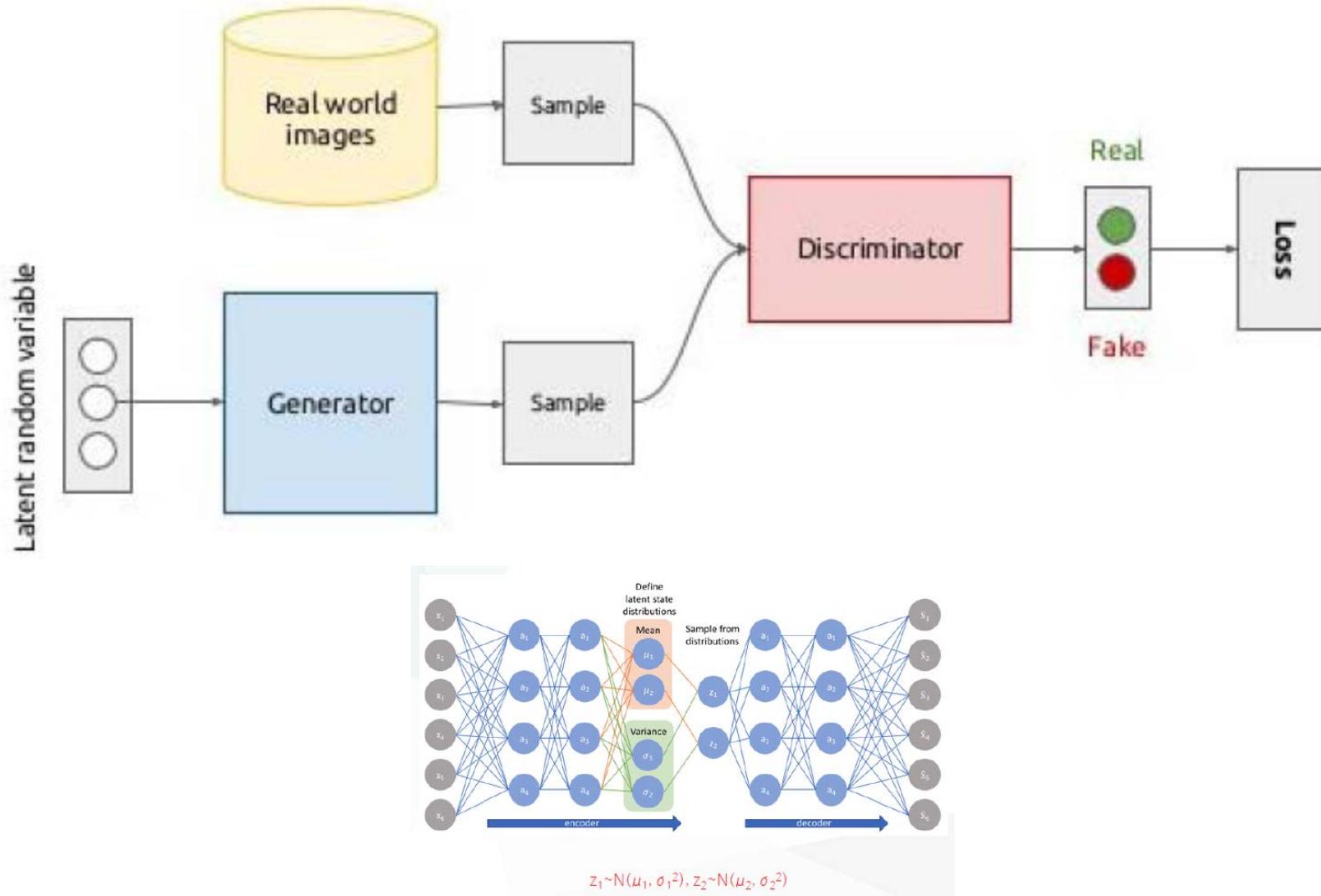
<https://shashank7-iitd.medium.com/understanding-vector-quantized-variational-autoencoders-vq-vae-323d710a888a>

[https://keras.io/examples/generative/vq\\_vae/](https://keras.io/examples/generative/vq_vae/)

<https://ml.berkeley.edu/blog/posts/vq-vae/>



# Generative Adversarial Network (GAN)





- $Z$ : noise vector
  - $G(Z)$ : output of the generator
  - $X$ : real training data
  - $D(G(Z))$ : output of the discriminator
  - $D(X)$ : output of the discriminator
- $D(X) = 1$  and  $D(G(Z)) = 0$  {We want our discriminator to label all  $D(X)$  as 1 and all  $D(G(Z))$  as 0.}

Given,  $X$  as input, the predicted value is  $D(X)$  and the ground truth is 1. Then,

$$\therefore L(D(X), 1) = 1 \cdot \log D(X) + (1 - 1) \cdot \log (1 - D(X))$$

Or,  $L(D(X), 1) = \log D(X)$

Given,  $G(Z)$  as input, the predicted value is  $D(G(Z))$  and the ground truth label is 0. Then,

$$\therefore L(D(G(Z)), 0) = 0 \cdot \log D(G(Z)) + (1 - 0) \cdot \log (1 - D(G(Z)))$$

Or,  $L(D(G(Z)), 0) = \log (1 - D(G(Z)))$

Maximise  $\left[ \log D(X) + \log (1 - D(G(Z))) \right]$

The discriminator will maximize  $D(X)$  and minimize  $D(G(Z))$  to overall maximize the above loss function.



The loss function of the discriminator over a batch is,

$$\text{Max} \left[ \mathbb{E}_{(X \sim P(X))} \left[ \log D(X) \right] + \mathbb{E}_{(Z \sim P(Z))} \left[ \log (1 - D(G(Z))) \right] \right]$$

Where,  $P(X)$  is the probability distribution of real training data and  $P(Z)$  is the probability distribution of the noise vector  $Z$ . Typically,  $P(Z)$  is Gaussian or uniform.

$$\text{Min} \left[ \mathbb{E}_{(X \sim P(X))} \left[ \log D(X) \right] + \mathbb{E}_{(Z \sim P(Z))} \left[ \log (1 - D(G(Z))) \right] \right]$$

Note that, the generator has no control over the first term so the **generator will only minimize the second term**.

Assume that,  $D$  is the parameters of the Discriminator and  $G$  is the parameters of the generator. So, we can write the loss function as,

$$\underset{G}{\text{Min}} \underset{D}{\text{Max}} \left[ \mathbb{E}_{(X \sim P(X))} \left[ \log D(X) \right] + \mathbb{E}_{(Z \sim P(Z))} \left[ \log (1 - D(G(Z))) \right] \right]$$

This means the  $D$  will maximize the loss function and the  $G$  will minimize the loss function.



# Generative adversarial network (GAN) 1/3

```

# train a GAN on a 1D function, y=sin(x)
from numpy import hstack
from numpy import zeros, ones, sin, pi
from numpy.random import rand, randn
from keras.models import Sequential
from keras.layers import Dense
from matplotlib import pyplot
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, AutoMinorLocator

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n):
    # generate points in the latent space
    z_input = randn(latent_dim * n)
    # reshape into a batch of inputs for the network
    z_input = z_input.reshape(n, latent_dim)
    return z_input

# define the standalone generator model
def define_generator(latent_dim, n_outputs=2):
    model = Sequential()
    # model.add(Dense(15, activation='relu', kernel_initializer='he_uniform', input_dim=latent_dim))
    model.add(Dense(15, activation='relu', input_dim=latent_dim))
    model.add(Dense(n_outputs, activation='linear'))
    return model

# define the standalone discriminator model
def define_discriminator(n_inputs=2):
    model = Sequential()
    # model.add(Dense(25, activation='relu', kernel_initializer='he_uniform', input_dim=n_inputs))
    model.add(Dense(25, activation='relu', input_dim=n_inputs)) (2)
    model.add(Dense(1, activation='sigmoid')) (1)
    # compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

```

(100)

(2)

(2)

(1)

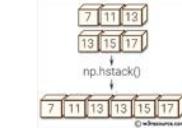
```

# define the combined generator and discriminator model, for updating the generator
def define_gan(generator, discriminator):
    # make weights in the discriminator not trainable
    discriminator.trainable = False
    # connect them
    model = Sequential()
    # add generator
    model.add(generator)
    # add the discriminator
    model.add(discriminator)
    # compile model
    model.compile(loss='binary_crossentropy', optimizer='adam')
    return model

# generate n real samples with class labels
def generate_real_samples(n):
    # generate inputs in [-0.5, 0.5]
    p = (rand(n) - 0.5)*2.*pi
    # generate outputs sin(x)
    q = sin(p)
    # stack arrays
    p = p.reshape(n, 1)
    q = q.reshape(n, 1)
    x = hstack((p, q))
    # generate class labels
    y = ones((n, 1))
    return x, y

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n):
    # generate points in latent space
    z_input = generate_latent_points(latent_dim, n)
    # predict outputs
    x = generator.predict(z_input)
    # create class labels
    y = zeros((n, 1))
    return x, y

```



(n,2), (n,1)

(n,2), (n,1)

# Generative adversarial network (GAN) 2/3

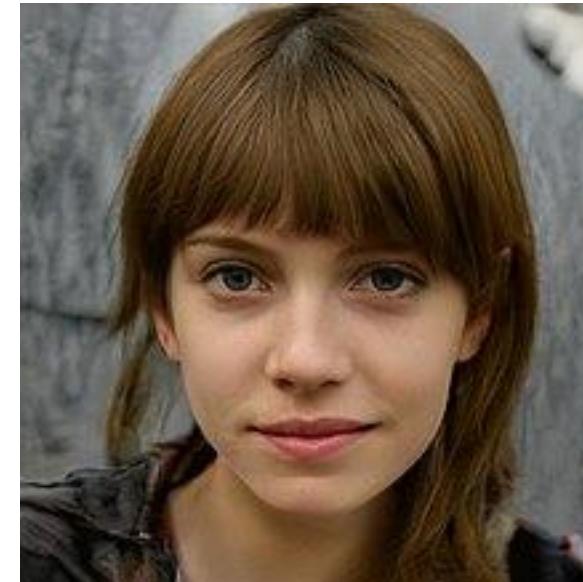
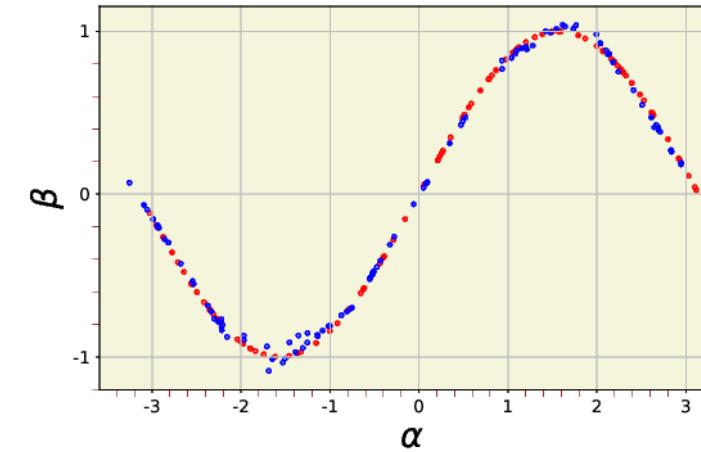
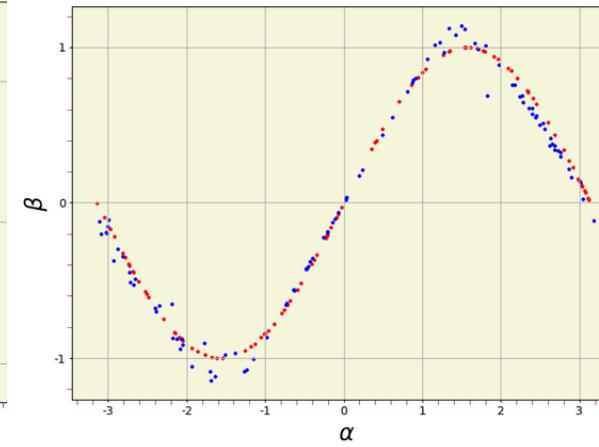
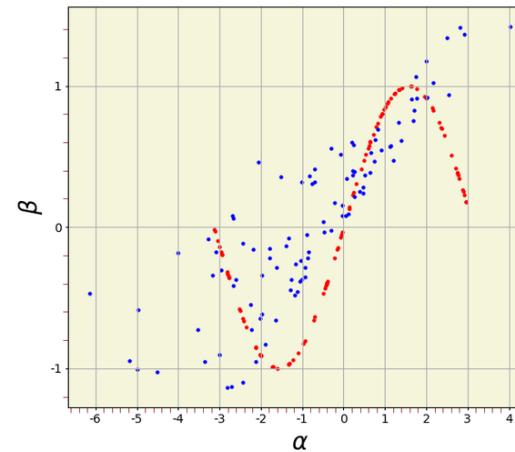
```
# evaluate the discriminator and plot real and fake points
def summarize_performance(epoch, generator, discriminator, latent_dim, n=100):
    # prepare real samples
    x_real, y_real = generate_real_samples(n)
    # evaluate discriminator on real examples
    _, acc_real = discriminator.evaluate(x_real, y_real, verbose=0)
    # prepare fake examples
    x_fake, y_fake = generate_fake_samples(generator, latent_dim, n)
    # evaluate discriminator on fake examples
    _, acc_fake = discriminator.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print(epoch, acc_real, acc_fake)
    # scatter plot real and fake data points
    pyplot.figure(figsize=(6,4))
    ax=pyplot.axes()
    ax.set_xlabel(r'$\alpha$', fontsize=20)
    ax.set_ylabel(r'$\beta$', fontsize=20)
    # ax.legend(fancybox=True, shadow=True, fontsize=15, framealpha=0.8)
    majorLocator= MultipleLocator(1)
    minorLocator= AutoMinorLocator()
    majorFormatter= FormatStrFormatter('%d')
    minorFormatter= FormatStrFormatter('%d')
    ax.xaxis.set_major_locator(majorLocator)
    ax.xaxis.set_major_formatter(majorFormatter)
    ax.xaxis.set_minor_locator(minorLocator)
    majorLocator= MultipleLocator(1)
    minorLocator= AutoMinorLocator()
    majorFormatter= FormatStrFormatter('%d')
    minorFormatter= FormatStrFormatter('%d')
    ax.yaxis.set_major_locator(majorLocator)
    ax.yaxis.set_major_formatter(majorFormatter)
    ax.yaxis.set_minor_locator(minorLocator)
    ax.tick_params(which='major', length=2, color='black')
    ax.tick_params(which='minor', length=4, color='brown')
    ax.set_facecolor("beige")
    pyplot.grid(True)
    pyplot.scatter(x_real[:, 0], x_real[:, 1], color='red', s=5)
    pyplot.scatter(x_fake[:, 0], x_fake[:, 1], color='blue', s=5)
    pyplot.tight_layout()
    str1='sin'+str(epoch)+'.pdf'
    pyplot.savefig(str1,dpi=150)
    # pyplot.show()

# train the generator and discriminator
def train(g_model, d_model, gan_model, latent_dim, n_epochs=100000, n_batch=128, n_eval=2000):
    # determine half the size of one batch, for updating the discriminator
    half_batch = int(n_batch / 2)
    # manually enumerate epochs
    for i in range(n_epochs):
        # prepare real samples
        x_real, y_real = generate_real_samples(half_batch)
        # prepare fake examples
        x_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
        # update discriminator
        d_model.train_on_batch(x_real, y_real)
        d_model.train_on_batch(x_fake, y_fake)
        # prepare points in latent space as input for the generator
        x_gan = generate_latent_points(latent_dim, n_batch)
        # create inverted labels for the fake samples
        y_gan = ones((n_batch, 1))
        # update the generator via the discriminator's error
        gan_model.train_on_batch(x_gan, y_gan)
        # evaluate the model every n_eval epochs
        if (i+1) % n_eval == 0:
            summarize_performance(i, g_model, d_model, latent_dim)

# size of the latent space
latent_dim = 10
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the GAN
gan_model = define_gan(generator, discriminator)
# train model
train(generator, discriminator, gan_model, latent_dim)
```

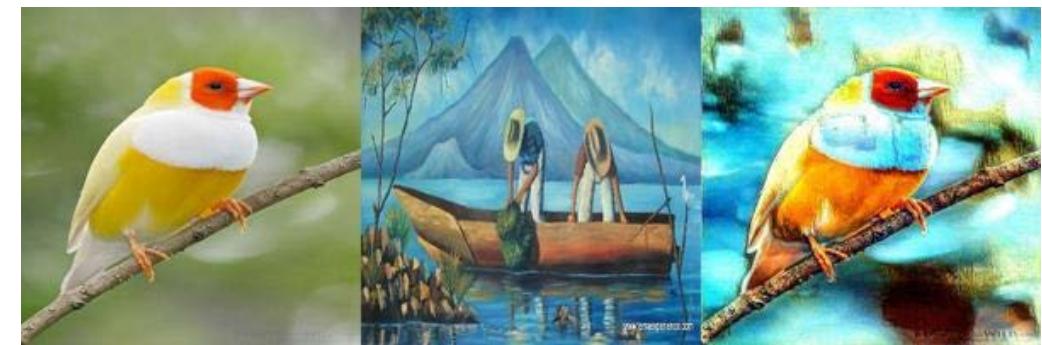


# Generative adversarial network (GAN) 3/3





ThisPersonDoesNotExist.com uses AI to generate endless fake faces

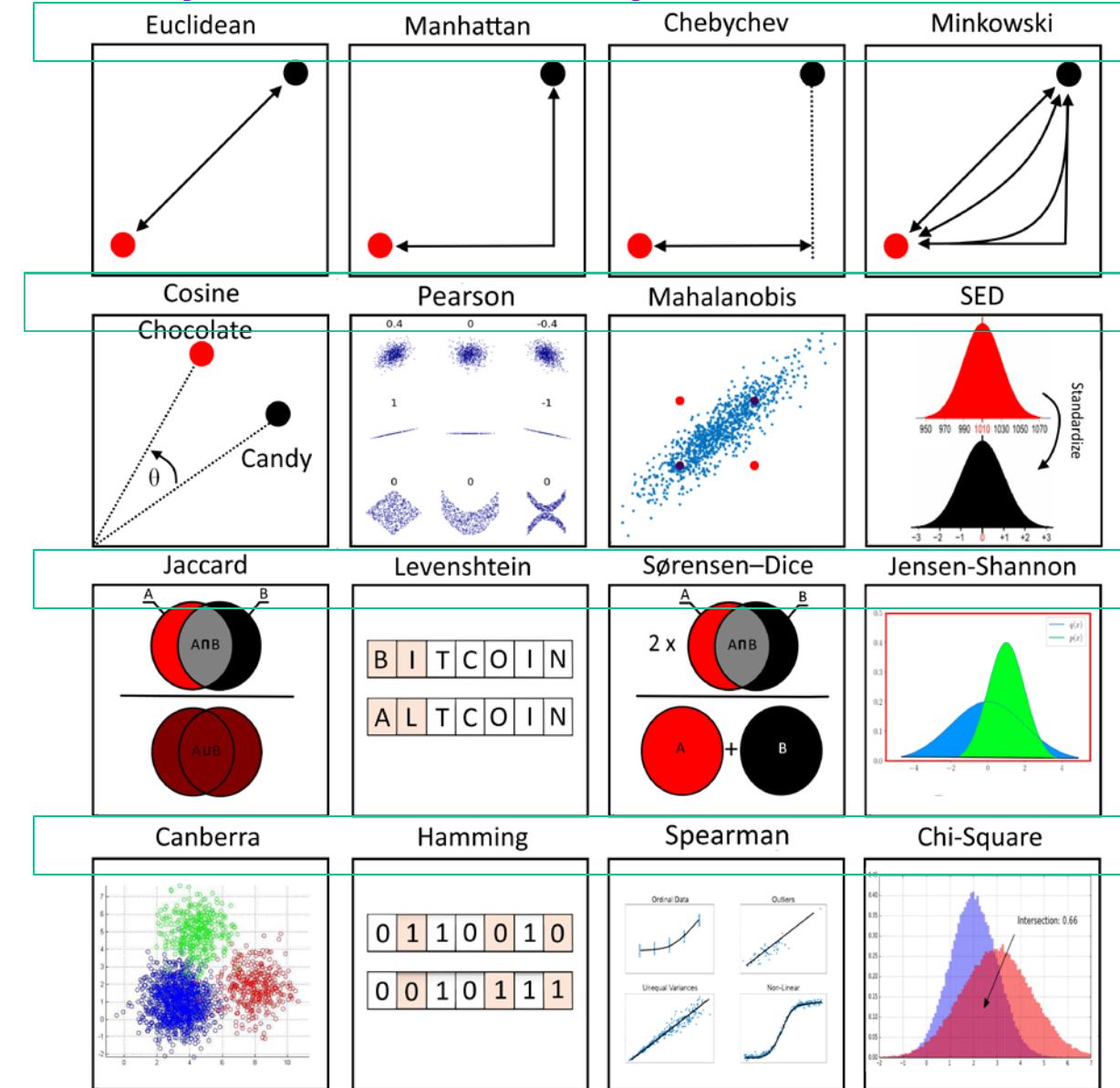


Neural Style Transfer  
Gram matrix

Generative models (**unsupervised learning**):  
Boltzmann machine, GAN, VAE, RNN, PixelCNN



# 16 types of similarity and dissimilarity measures used in data science





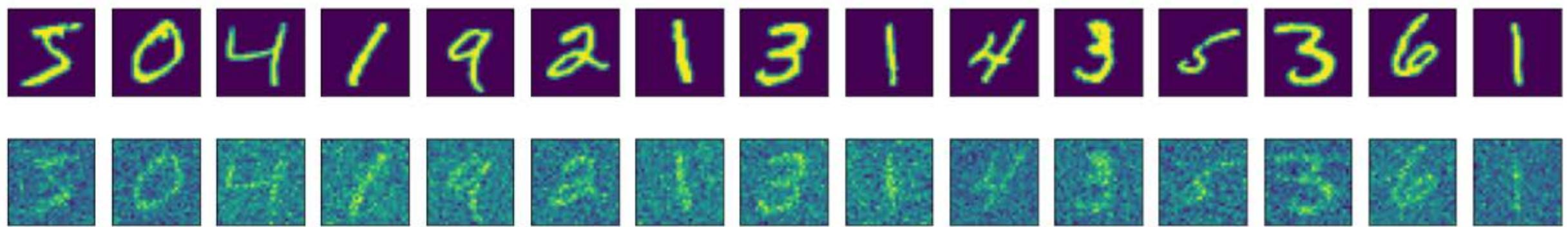
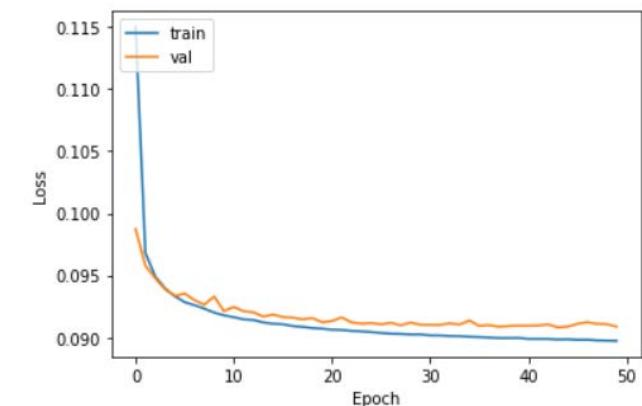
# Keras GAN

```
from keras_adversarial import AdversarialModel, simple_gan, gan_targets  
  
.adversarial_compile
```



# AE, denosing

```
model = Sequential()  
model.add(Conv2D(64,kernel_size=(3,3),kernel_constraint=max_norm(max_norm_value),activation = 'relu',kernel_initializer= 'he_uniform',input_shape = input_shape))  
model.add(Conv2D(32,kernel_size=(3,3),kernel_constraint=max_norm(max_norm_value),activation='relu',kernel_initializer='he_uniform'))  
model.add(Conv2DTranspose(32,kernel_size=(3,3),kernel_constraint=max_norm(max_norm_value),activation='relu',kernel_initializer='he_uniform'))  
model.add(Conv2DTranspose(64, kernel_size=(3,3),kernel_constraint= max_norm(max_norm_value),activation='relu',kernel_initializer= 'he_uniform'))  
model.add(Conv2D(1,kernel_size=(3,3),kernel_constraint=max_norm(max_norm_value), activation='sigmoid',padding = 'same'))  
input_shape = x_train.shape  
model.build(input_shape)  
model.summary()  
model.compile(optimizer='adam',loss='binary_crossentropy')  
history=model.fit(noise_train, x_train, validation_split= validation_splits, epochs=no_epochs, batch_size=batch_size)
```





# PixelCNN

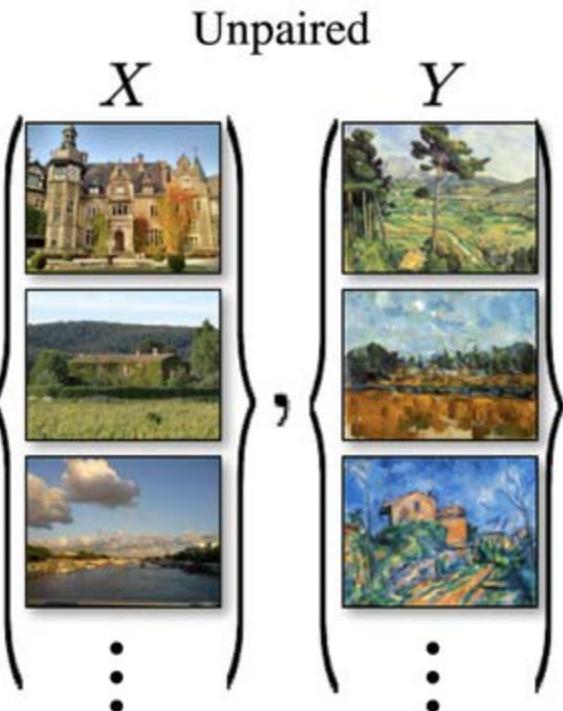
3 7

254

**KRISS**

# CycleGAN

There exist several methods to design forms with fields to fields may be surrounded by bounding boxes, by light rectangles or methods specify where to write and, therefore, minimize the effect with other parts of the form. These guides can be located on a separate sheet is much better from the point of view of the quality but requires giving more instructions and, more importantly, rest this type of acquisition is used. Guiding rulers printed on the used for this reason. Light rectangles can be removed more easily whenever the handwritten text touches the rulers. Nevertheless, be taken into account: The best way to print these light rectan



## ▼ Objective function:

### Consistency losses:

forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$

backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

$$L_{cyc}(G, F) = E_x \sim p_{data(x)}[||F(G(x)) - x||] + E_y \sim p_{data(y)}[||G(F(y)) - y||]$$

### Adversarial Losses:

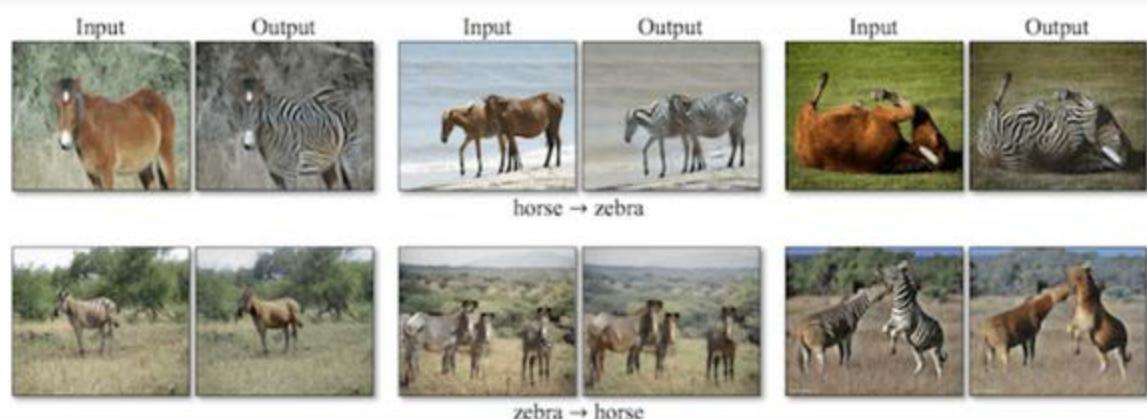
$$L_{GAN}(G, D_Y, X, Y) = E_y \sim p_{data(y)}[\log D_Y(y)] + E_x \sim p_{data(x)}[\log(1 - D_Y(G(x)))]$$

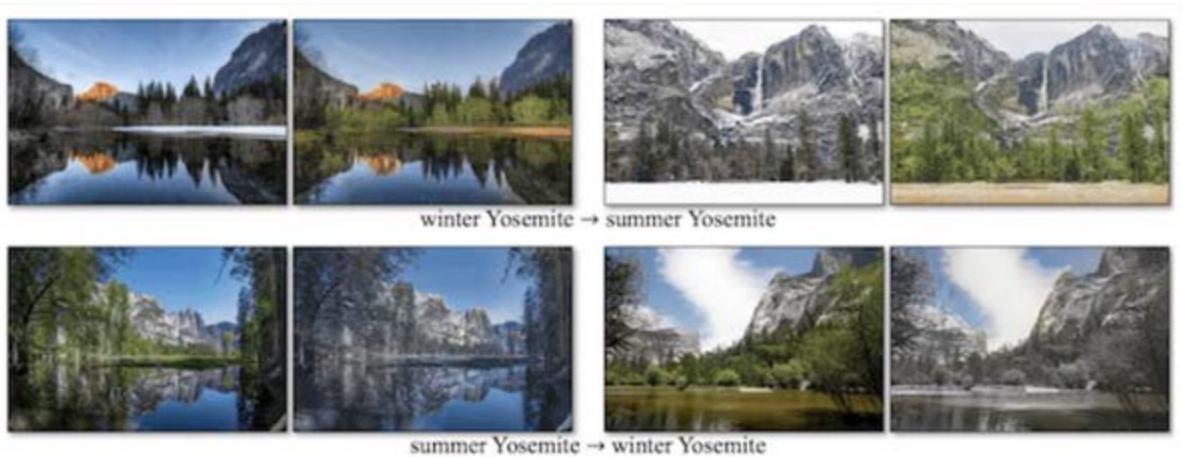
Similarly, the second adversarial loss  $L_{GAN}(F, D_X, Y, X)$

### Combined Objective function :

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \lambda L_{cyc}(G, F)$$

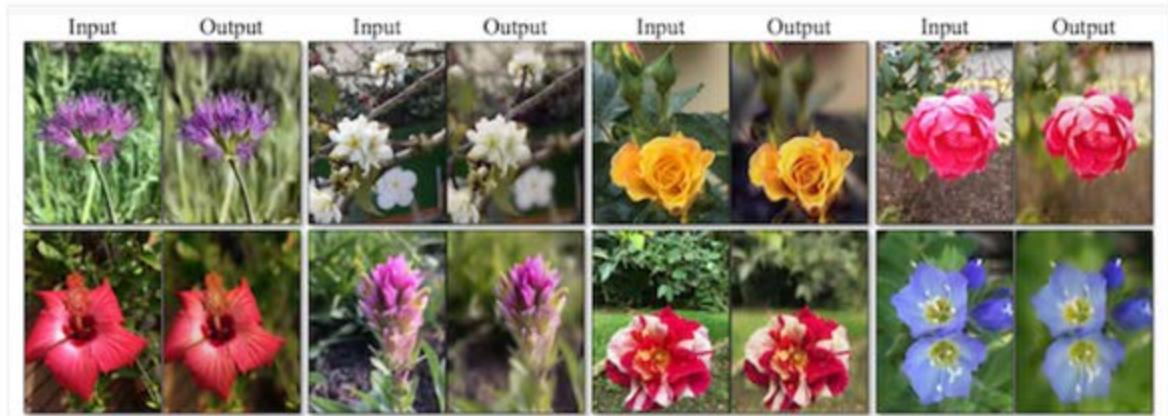
```
# load image data
dataset = load_real_samples('horse2zebra_256.npz')
print('Loaded', dataset[0].shape, dataset[1].shape)
# define input shape based on the loaded dataset
image_shape = dataset[0].shape[1:]
# generator: A → B
g_model_AtoB = define_generator(image_shape)
# generator: B → A
g_model_BtoA = define_generator(image_shape)
# discriminator: A → [real/fake]
d_model_A = define_discriminator(image_shape)
# discriminator: B → [real/fake]
d_model_B = define_discriminator(image_shape)
# composite: A → B → [real/fake, A]
c_model_AtoB = define_composite_model(g_model_AtoB, d_model_B, g_model_BtoA, image_shape)
# composite: B → A → [real/fake, B]
c_model_BtoA = define_composite_model(g_model_BtoA, d_model_A, g_model_AtoB, image_shape)
# train models
train(d_model_A, d_model_B, g_model_AtoB, g_model_BtoA, c_model_AtoB, c_model_BtoA, dataset)
```





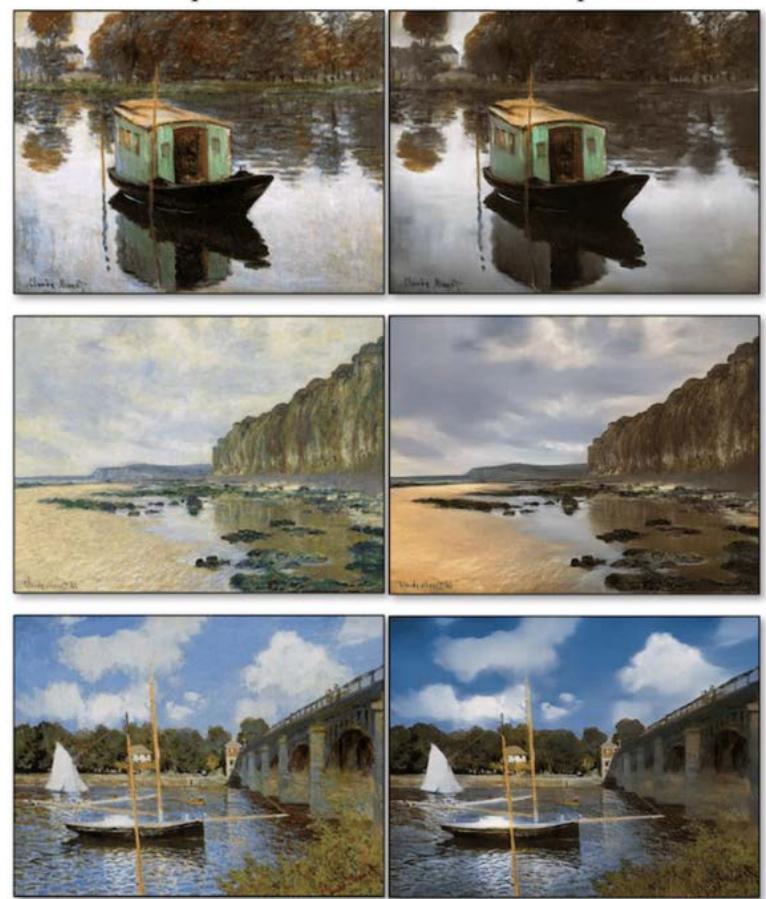
winter Yosemite → summer Yosemite

summer Yosemite → winter Yosemite



## Input

## Output



cycleGAN이 이미지 속의 "인공적인 부분들"을 제거하는 목적으로 활용된 경우:

[Optica Publishing Group \(osapublishing.org\)](#)

cycleGAN이 구조 최적화에 활용된 경우:

[Mol-CycleGAN: a generative model for molecular optimization | ScienceGate](#)

defect segmentation에 활용된 경우:

[IEEE Xplore Full-Text PDF:](#)

음악, 고전음악 → 재즈음악:

[MIDI-VAE: Modeling Dynamics and Instrumentation of Music with Applications to Style Transfer | Paper | Microsoft Academic](#)

Revisiting CycleGAN for semi-supervised segmentation

From the reported results, we have shown that this strategy improves segmentation performance, particularly when annotated data is scarce.

<https://arxiv.org/pdf/1908.11569.pdf>

2D-to-3D CycleGAN (2D → 3D)

현재 사진 → 10년 후 사진

사진 → 아바타

안전을 위해서 약한 신호로 획득한 의료정보를 강한 신호로 받아 보았다고 가정하고 이를 예측하는 기술.

강한 선량을 유지할 경우 시료가 변질 될 수 있는 경우. 약한 선량으로 측정하고 강한 선량으로 측정한 데이터를 예측하는 기술.

## Principal component analysis

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
print(pca.components_)
print(pca.explained_variance_)

def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate(' ', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');

pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape: ", X.shape)
print("transformed shape:", X_pca.shape)
```

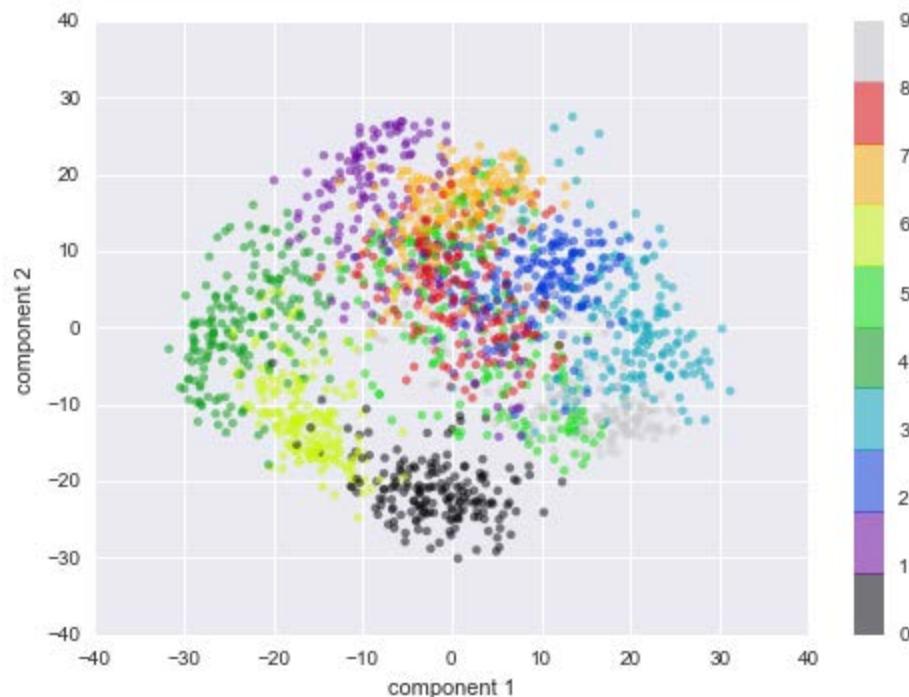


## Principal component analysis

```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape

pca = PCA(2) # project from 64 to 2 dimensions
projected = pca.fit_transform(digits.data)
print(digits.data.shape)
print(projected.shape)

plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



## Mahalanobis distance

```
import numpy as np
from scipy.spatial.distance import euclidean
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import mahalanobis
def mahalanobis_distance(p, distr):
    # p: a point
    # distr : a distribution
    # covariance matrix
    cov = np.cov(distr, rowvar=False)
    # average of the points in distr
    avg_distri = np.average(distr, axis=0)
    dis = mahalanobis(p, avg_distri, cov)
    return dis

X = np.array([[1,2], [2,2], [3,3]])
cov = np.cov(X, rowvar=False)
covI = np.linalg.inv(cov)
mean=np.mean(X)
maha = mahalanobis(X[0], X[1], covI)
pca = PCA(whiten=True)
X_transformed= pca.fit_transform(X)
print('Mahalanobis distance: '+str(maha))
print('Euclidean distance: '+str(euclidean(X_transformed[0],X_transformed[1])))

p=[1,2]
dist= mahalanobis_distance(p, X_transformed)
print(dist)

Mahalanobis distance: 2.0
Euclidean distance: 2.0000000000000004
2.2360797749979
```

<https://www.statology.org/mahalanobis-distance-python/>

<https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.mahalanobis.html>

[https://en.wikipedia.org/wiki/Mahalanobis\\_distance](https://en.wikipedia.org/wiki/Mahalanobis_distance)



## What is probability?

Fundamentally related to the frequencies of repeated events.

*- Frequentists*

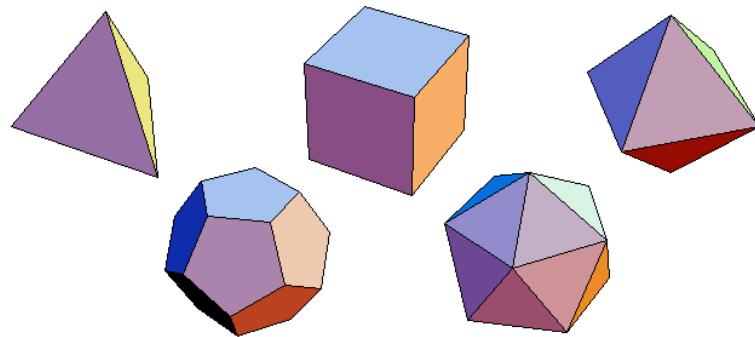


Fundamentally related to our own certainty or uncertainty of events.

*- Bayesians*

For very simple problems,  
frequentist & Bayesian results are  
often practically indistinguishable

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)}$$



4면체, 6면체, 8면체, 12면체, 그리고 20면체 주사위가 든 상자가 있다.  
상자에서 임의로 주사위 하나를 집어서 던졌더니 "12"가 나왔습니다.  
그렇다면 각각의 주사위를 선택했었을 확률은?

	tetrahedron	hexahedron	octahedron	dodecahedron	icosahedron
a priori $P(H)$	1/5	1/5	1/5	1/5	1/5
likelihood $P(D H)$	0	0	0	1/12	1/20
$a \text{ priori} \times \text{likelihood}$	0	0	0	1/60	1/100
$a \text{ posteriori } P(H D)$	0	0	0	5/8	3/8

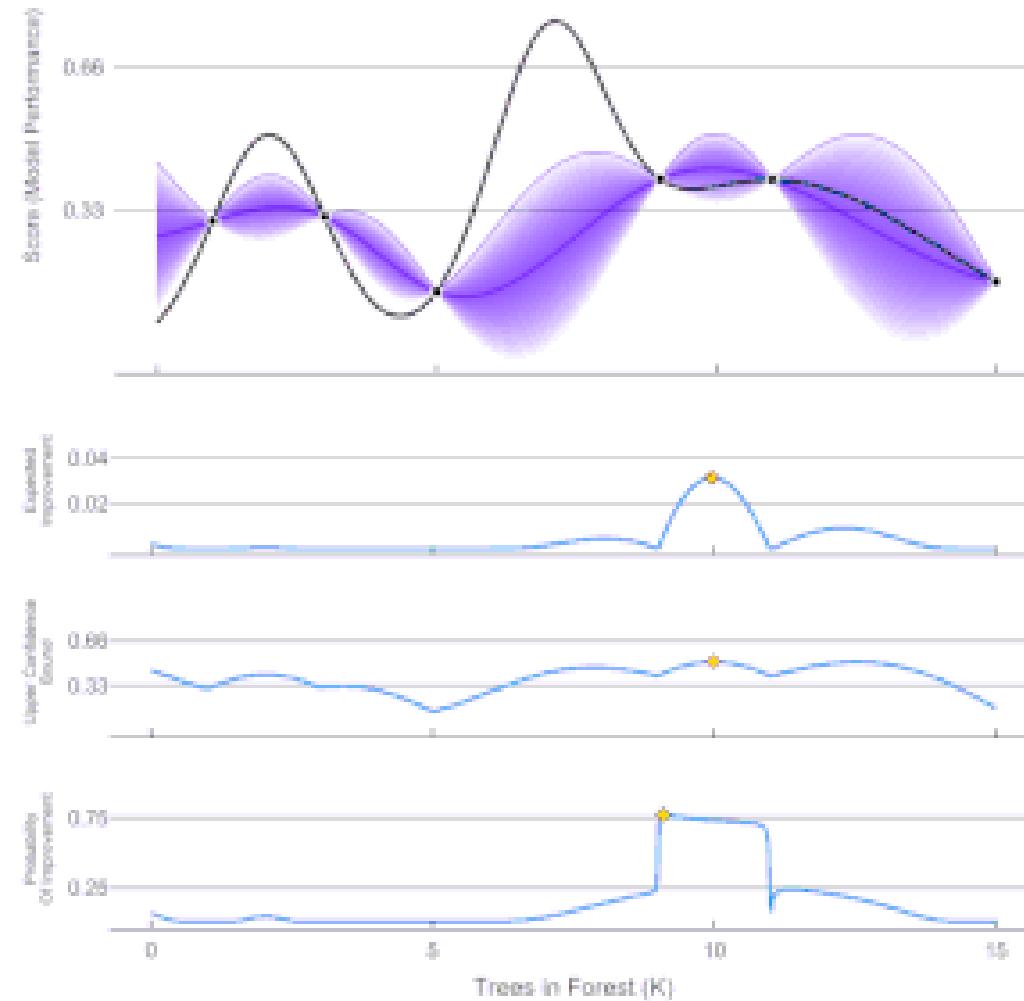
1/60 + 1/100 = (10+6)/600 = 16/600

600/16 \* 1/60 = 10/16 = 5/8

600/16 \* 1/100 = 6/16 = 3/8

Bayesian optimization

### ParBayesianOptimization in Action (Round 1)



- 베이지안 옵티마이제이션(Bayesian optimization) 방법의 일반적 성질들:
  1. 순차적 접근법이다. 계산이 복잡화되지 않는다.
  2. 목적함수의 도함수를 이용하지 않는다. [도함수를 알 수 없을 정도로 이산적인 경우에 적합하다.]
  3. 문제에 따라서는 도함수가 정의되지 않을 수 있다.
  4. 물론 도함수가 정의되어도 계산하는 것이 복잡할 때에도 적용할 수 있다. 이 경우, 자동구배를 활용할 수도 있겠다.]
  5. 목적함수를 이용해서 보다 더 좋은 해가 있는 곳을 예측한다. (surrogate model; 대리 모델)[인공지능을 이용한다.]
  6. 여러 가지 model이 가능하다. 또한 model 선택에 민감하게 결과가 변화할 수 있다.
  7. 목적함수 계산을 너무 많이 할 수 없는 상황에 적절한 방법으로 알려져 있다.
  8. [즉, 계산량이 너무 많을 경우에 해당한다. 목적함수가 아주 비싼 대가를 치르는 경우는 무수히 많다.]
  9. 다시 말해서, 기계학습 방법으로 surrogate model을 만드는 비용이 아주 싸게 먹히는 경우에 적합한 최적화 방법이다.
  10. (하지만, 데이터 수가 너무 많을 경우 문제가 될 수 있다.)
  11. 데이터 갯수의 삼승에 비례하는 문제가 발생한다. 이것은 역행렬 계산과 연관이 있는 복잡도이다.)
  12. 목적함수 계산에 얼마나 많은 시간이 소모되는지? 기대 향상을 얻어내는데 걸리는 시간을 직접 비교해 보아야 한다.
  13. 목적함수가 노이즈를 가지고 있을 때 사용할 수 있는 일반적인 최적화 알고리듬이다.
  14. 목적함수가 단순한 컴퓨터 계산으로 마무리 되는 것이라고 단정하지 말라. 이것이 실험 결과일 수도 있다. 아주 많은 노력이 필요한 것일 수도 있다.
  15. 추가적인 데이터의 보강이 이루어 질 때, 지속적으로 기계학습 방법을 이용할 수 있다.
- <https://github.com/fmfn/BayesianOptimization>
-

# What is Bayesian inference?

Prior assumptions → Look at evidence from data → Update prior assumptions → Posterior distribution

Main differences with classical (frequentist) statistics:

- Bayesians account for the prior distribution (prior belief). (Frequentist models are, in a way, Bayesian models with a flat prior: example of the coin toss)
- Bayesians preserve a notion of uncertainty about estimates, frequentist models tend to collapse these uncertainties into decisions or single-point estimates.



$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta')p(\theta')d\theta'}$$

more than 1 observation

$$P(y|\theta) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\left(\frac{1}{2\sigma^2}\right)(y-\theta)^2\right\} \quad \text{Likelihood}$$

NOW/ Remember the Bayesian relationship:

$$\text{Posterior} \leftarrow P(\theta|y) \propto P(y|\theta)P(\theta) \rightarrow \text{Prior}$$

Consider a **prior** of the following form:

$$P(\theta) \propto \exp\left\{-\left(\frac{1}{2\tau_0^2}\right)(\theta - \mu_0)^2\right\} \quad \theta \sim N(\mu_0, \tau_0^2)$$

$$P(\theta|y) \propto \exp\left\{-\frac{1}{2}\left[\frac{(y-\theta)^2}{\sigma^2} + \frac{(\theta - \mu_0)^2}{\tau_0^2}\right]\right\}$$

$$P(\theta|y) \propto \exp\left\{-\frac{(\theta - \mu_1)^2}{2\tau_1^2}\right\}$$

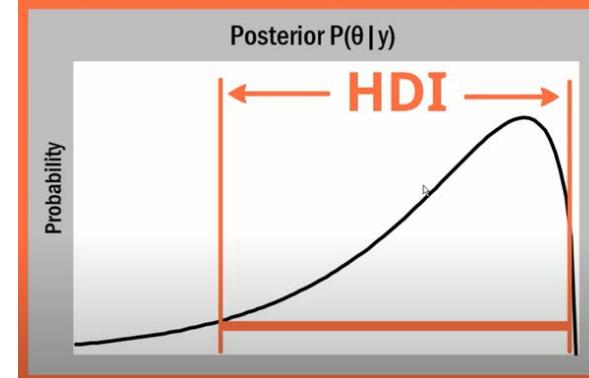
$$\text{Where } \mu_1 = \frac{(\mu_0/\tau_0^2) + (y/\sigma^2)}{(1/\tau_0^2) + (1/\sigma^2)} \quad \frac{1}{\tau_1^2} = \frac{1}{\tau_0^2} + \frac{1}{\sigma^2}$$

$$P(\theta|y) \propto \exp\left\{-\frac{1}{2}\left[\frac{(y-\theta)^2}{\sigma^2} + \frac{(\theta - \mu_0)^2}{\tau_0^2}\right]\right\}$$

$$P(\theta|y) \propto \exp\left\{-\frac{(\theta - \mu_1)^2}{2\tau_1^2}\right\}$$

$$\text{Where } \mu_n = \frac{(\mu_0/\tau_0^2) + \bar{y}(n/\sigma^2)}{(1/\tau_0^2) + (n/\sigma^2)} \quad \frac{1}{\tau_n^2} = \frac{1}{\tau_0^2} + \frac{n}{\sigma^2}$$

## Interval estimation



Likelihood : Conjugate prior  
Normal : Normal

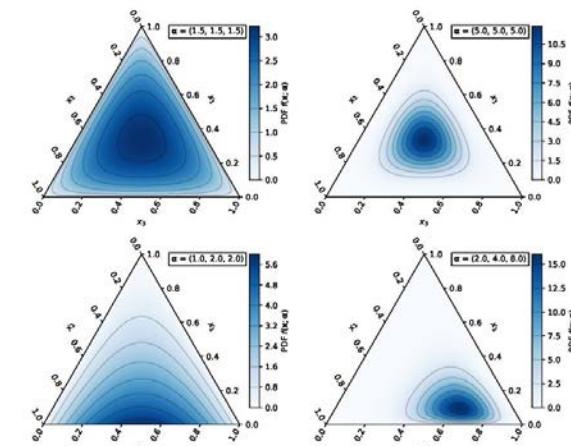
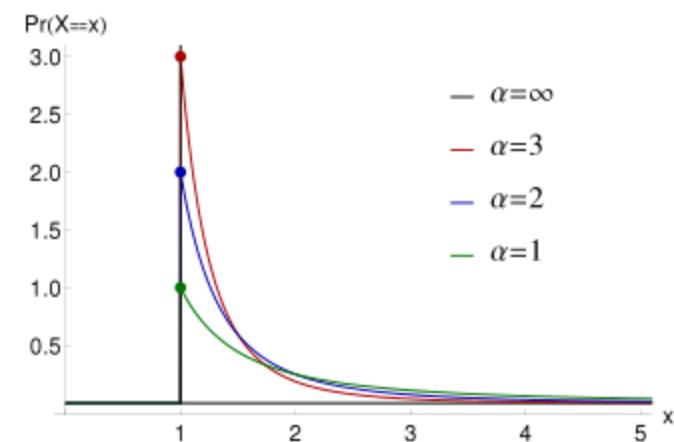
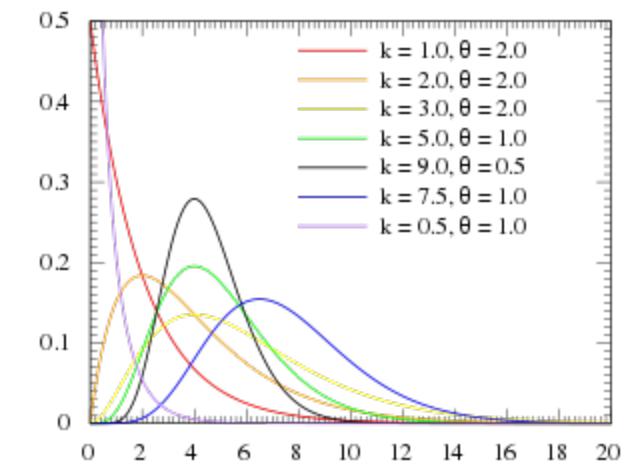
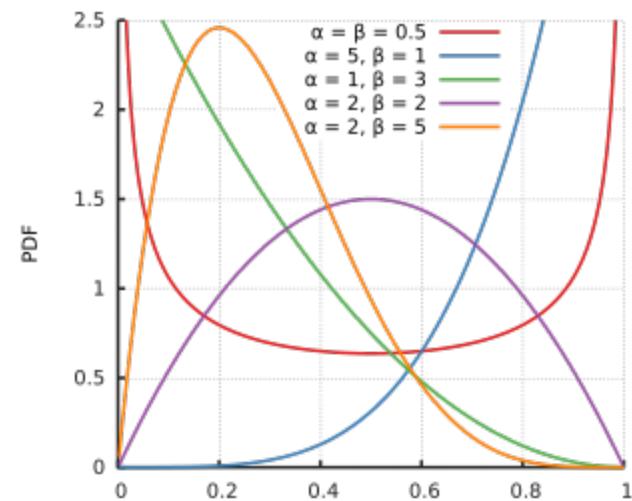
Binomial/Bernoulli : Beta  
Exponential/Poisson : Gamma

Uniform : Pareto

Multinomial : Dirichlet



Likelihood : Conjugate prior  
Normal : Normal  
Binomial/Bernoulli : Beta  
Exponential/Poisson : Gamma  
Uniform : Pareto  
Multinomial : Dirichlet



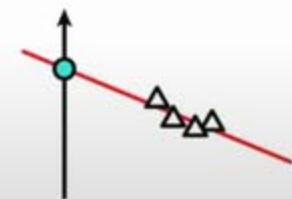
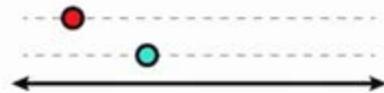
# Bayesian statistical modeling and probabilistic machine learning

- PyMC3 (a PP framework)
- Statistical distributions
- Sampling algorithms
- Syntax

```
conda create -n pymc_env -c conda-forge python libpython mkl-service numba python-graphviz scipy arviz
conda activate pymc_env
pip install pymc3
```

```
conda remove theano
conda install -c conda-forge theano-pymc
```

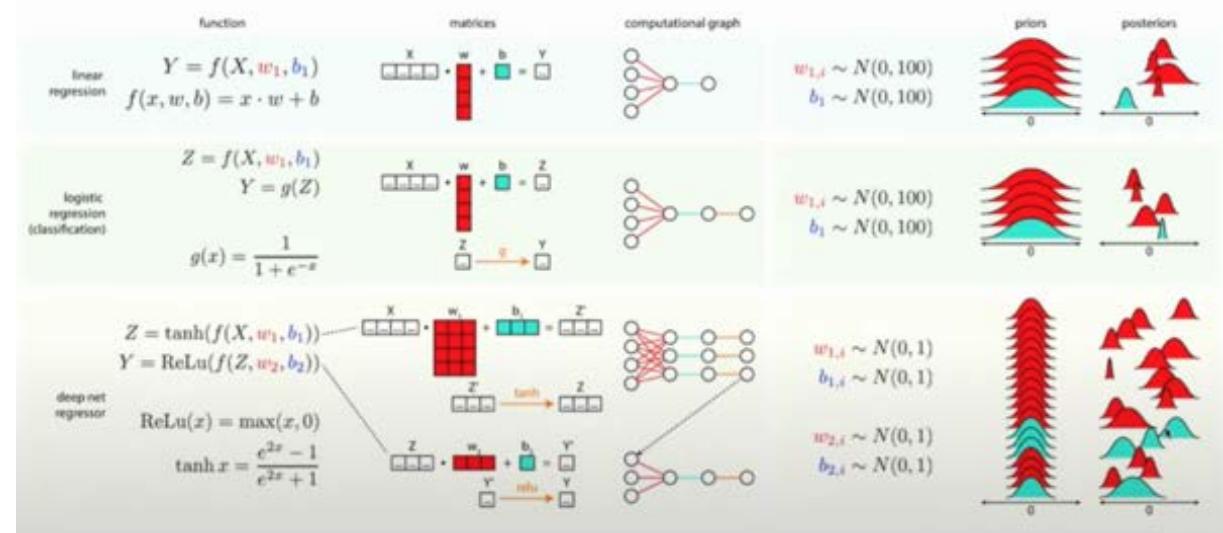
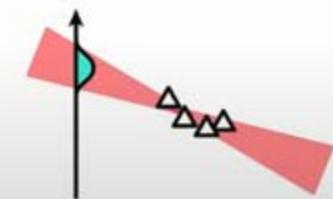
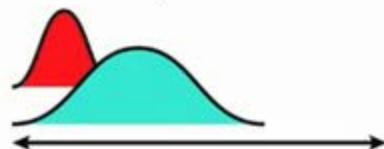
## Non-Bayesian

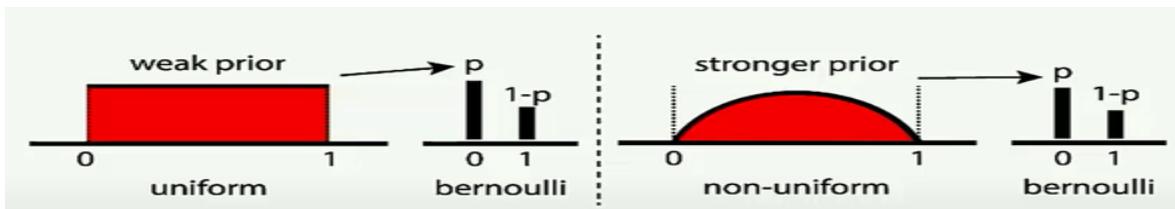


$$X \quad \quad \quad W \quad \quad \quad b \quad \quad \quad Y$$

$$\begin{bmatrix} \dots & \dots & \dots & \dots \end{bmatrix} \quad \bullet \quad \begin{bmatrix} \text{red triangle} \\ \text{red triangle} \\ \text{red triangle} \\ \text{red triangle} \end{bmatrix} \quad + \quad \begin{bmatrix} \text{red triangle} \\ \text{red triangle} \end{bmatrix} \quad = \quad \begin{bmatrix} \text{red triangle} \\ \text{red triangle} \end{bmatrix}$$

## Bayesian





```

# Context manager syntax. `coin_model` is **just**
# a placeholder
with pm.Model() as coin_model:
    # Distributions are PyMC3 objects.
    # Specify prior using Uniform object.
    p_prior = pm.Uniform('p', 0, 1)

    # Specify likelihood using Bernoulli object.
    like = pm.Bernoulli('likelihood', p=p_prior,
                        observed=tosses)
    # "observed=data" is key
    # for likelihood.

    with coin_model:
        # don't worry about this:
        step = pm.Metropolis()

        # focus on this, the Inference Button:
        coin_trace = pm.sample(2000, step=step)

    pm.traceplot(coin_trace)
    plt.show()

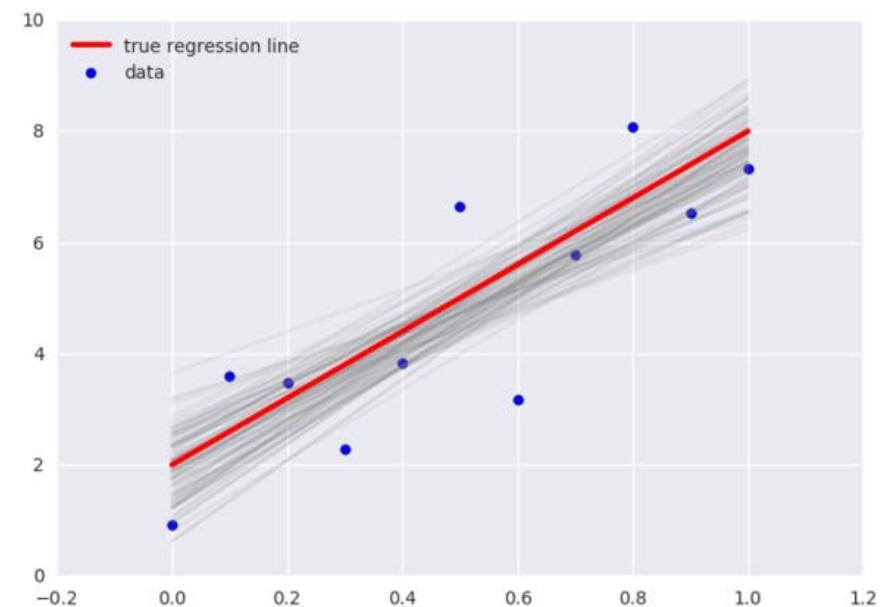
    pm.plot_posterior(coin_trace[100:], color='#87ceeb',
                      rope=[0.48, 0.52], point_estimate='mean',
                      ref_val=0.5)
    plt.show()

```

```

# observed data
np.random.seed(123)
n = 11
_a = 6
_b = 2
x = np.linspace(0, 1, n)
y = _a*x + _b + np.random.randn(n)
niter = 10000
with pm.Model() as linreg:
    a = pm.Normal('a', mu=0, sd=100)
    b = pm.Normal('b', mu=0, sd=100)
    sigma = pm.HalfNormal('sigma', sd=1)
    y_est = a*x + b
    likelihood = pm.Normal('y', mu=y_est, sd=sigma, observed=y)
    trace = pm.sample(niter, random_seed=123)
    t = trace[niter//2:]
    pm.traceplot(trace, varnames=['a', 'b'])
    pass
    plt.scatter(x, y, s=30, label='data')
    for a_, b_ in zip(t['a'][-100:], t['b'][-100:]):
        plt.plot(x, a_*x + b_, c='gray', alpha=0.1)
    plt.plot(x, _a*x + _b, label='true regression line', lw=3., c='red')
    plt.legend(loc='best')
    pass

```



```

import pymc3 as pm
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')

# Initialize random number generator
np.random.seed(123)

# True parameter values
alpha, sigma = 1, 1
beta = [1, 2.5]

# Size of dataset
size = 1000

# Predictor variable
X1 = np.random.randn(size)
X2 = np.random.randn(size) * 0.2

# Simulate outcome variable
Y = alpha + beta[0]*X1 + beta[1]*X2 + np.random.randn(size)*sigma

fig, axes = plt.subplots(1, 2, sharex=True, figsize=(10,4))
axes[0].scatter(X1, Y)
axes[1].scatter(X2, Y)
axes[0].set_ylabel('Y'); axes[0].set_xlabel('X1'); axes[1].set_xlabel('X2');

basic_model = pm.Model()

with basic_model:

    # Priors for unknown model parameters
    alpha = pm.Normal('alpha', mu=0, sigma=100)
    beta = pm.Normal('beta', mu=0, sigma=100, shape=2)
    sigma = pm.HalfNormal('sigma', sigma=100)

    # Expected value of outcome
    mu = alpha + beta[0]*X1 + beta[1]*X2

    # Likelihood (sampling distribution) of observations
    Y_obs = pm.Normal('Y_obs', mu=mu, sigma=sigma, observed=Y)

with basic_model:
    # draw 5000 posterior samples
    trace = pm.sample(5000)

    pm.traceplot(trace);

    pm.summary(trace).round(2)

```



# Markov-based Monte Carlo



- abandoning the standard Markov-based Monte Carlo methods
- introducing a feedback mechanism

*New types of  $x \rightarrow x'$*   
 **$M$  copies of system**

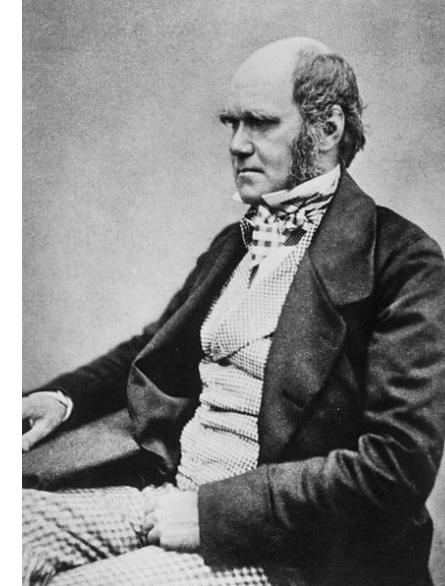


# Lamarckian and Darwinian



J.-B. Lamarck (1744-1829)

cf. epigenetics



C. Darwin (1809-1882)

*New types of  $x \rightarrow x'$*   
 **$M$  copies of system**

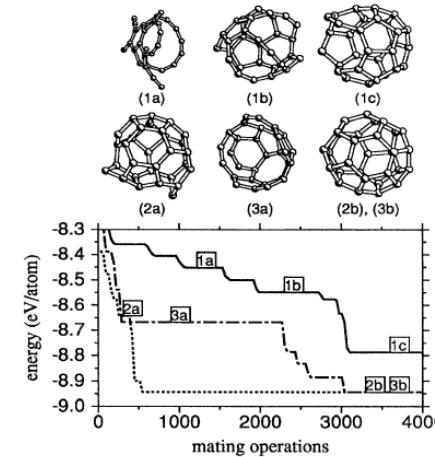
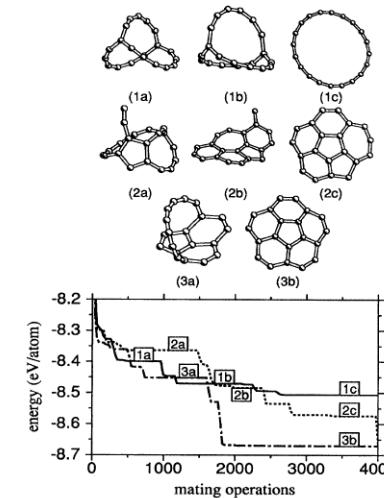
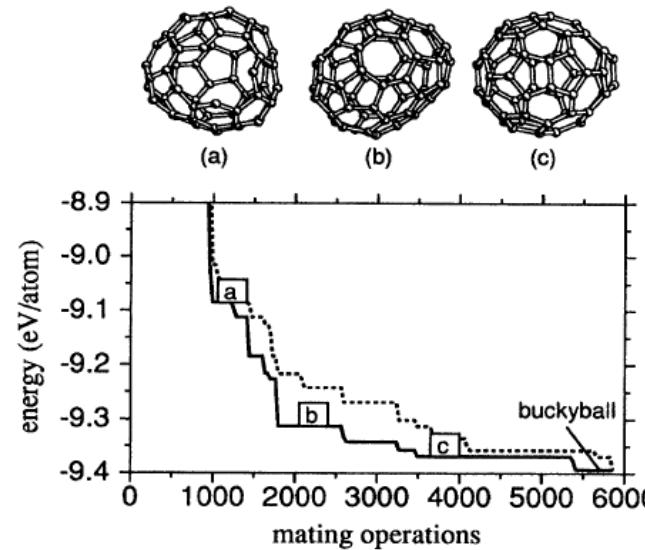
# Particle Swarm Optimization (PSO)

- PSO is similar to the Genetic Algorithm (GA) in the sense that these two evolutionary heuristics are population-based search methods.
- In other words, PSO and the GA move from a set of points (population) to another set of points in a single iteration with likely improvement using a combination of deterministic and probabilistic rules.

*Sharing the information for  $x \rightarrow x'$   
 **$M$  copies of system***



# Genetic algorithm



*Crossover and mutation for  $x \rightarrow x'$   
 $M$  copies of system*

$x \leftarrow \text{chromosome}$

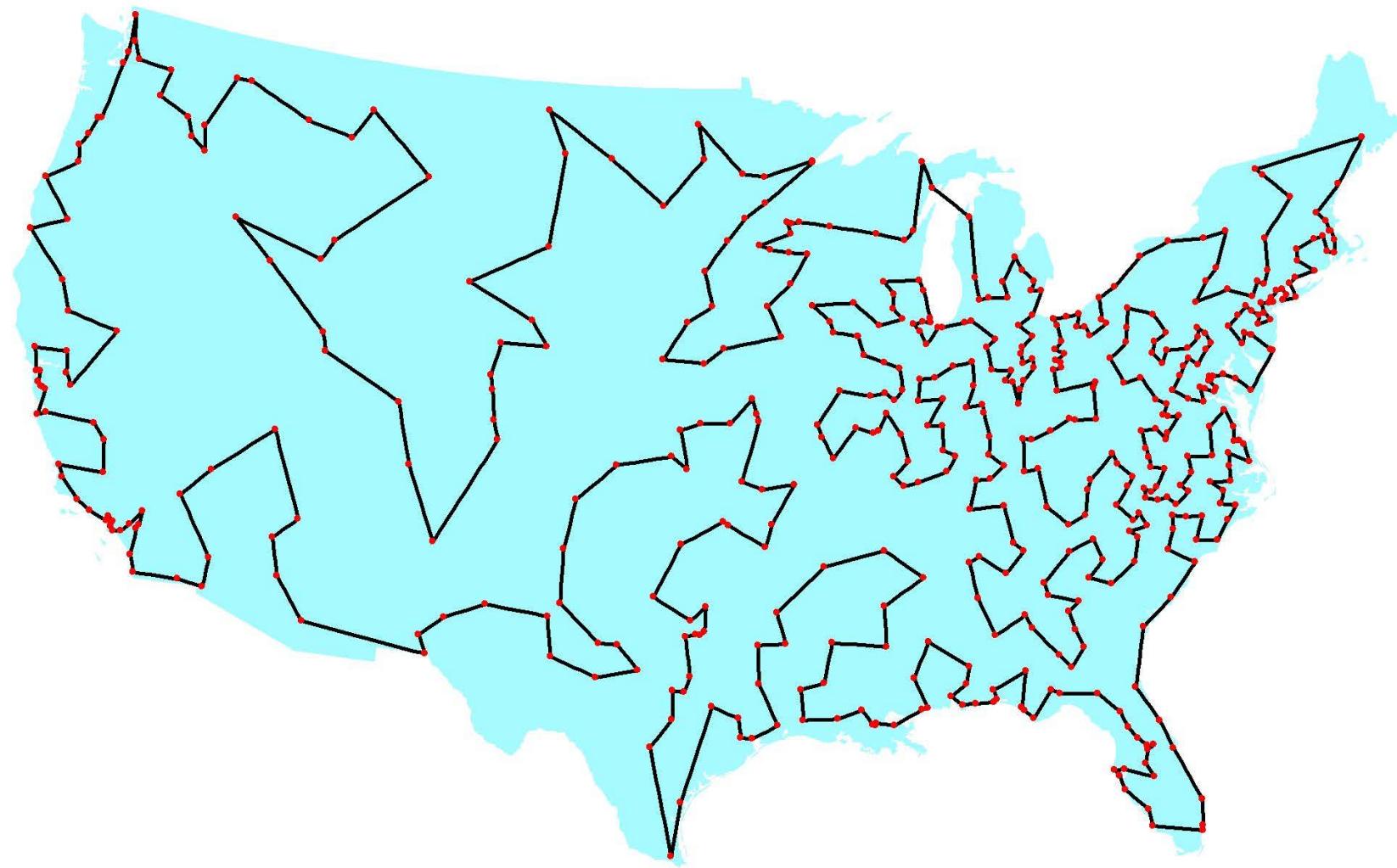
Exploration  $\leftarrow$  Mutation  
 Exploitation  $\leftarrow$  Crossover

Rechenberg, I. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution; Frommann-Holzboog: 1973.  
 Holland, J. H. Adaptation in Natural and Artificial Systems; The University of Michigan Press: Ann Arbor, MI, 1975.  
 J. H. Holland, Adaptation in natural and artificial systems (second ed.) MIT Press, Cambridge (1992).  
 Phys. Rev. Lett. 75, 288 (1995).



# att532

**Padberg and Rinaldi (1987) found the optimal tour of 532 AT&T switch locations in the USA.**

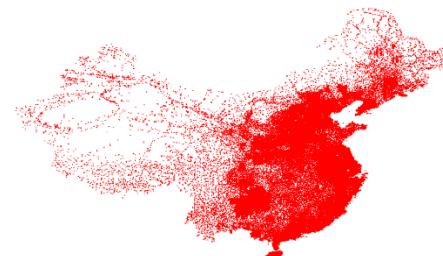


532 Cities

factorial(531)/2

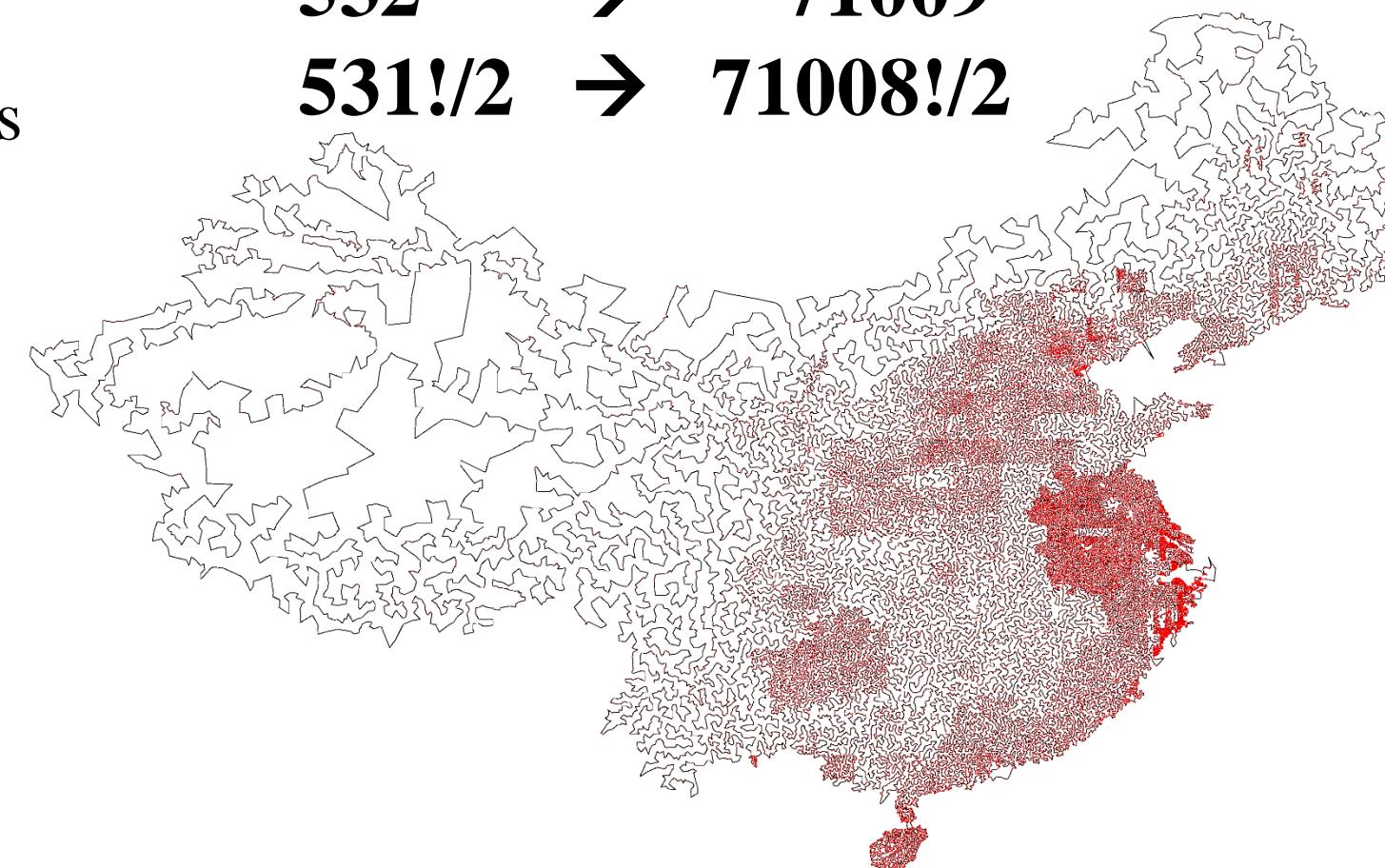


## CH71009 - China



71,009 Cities

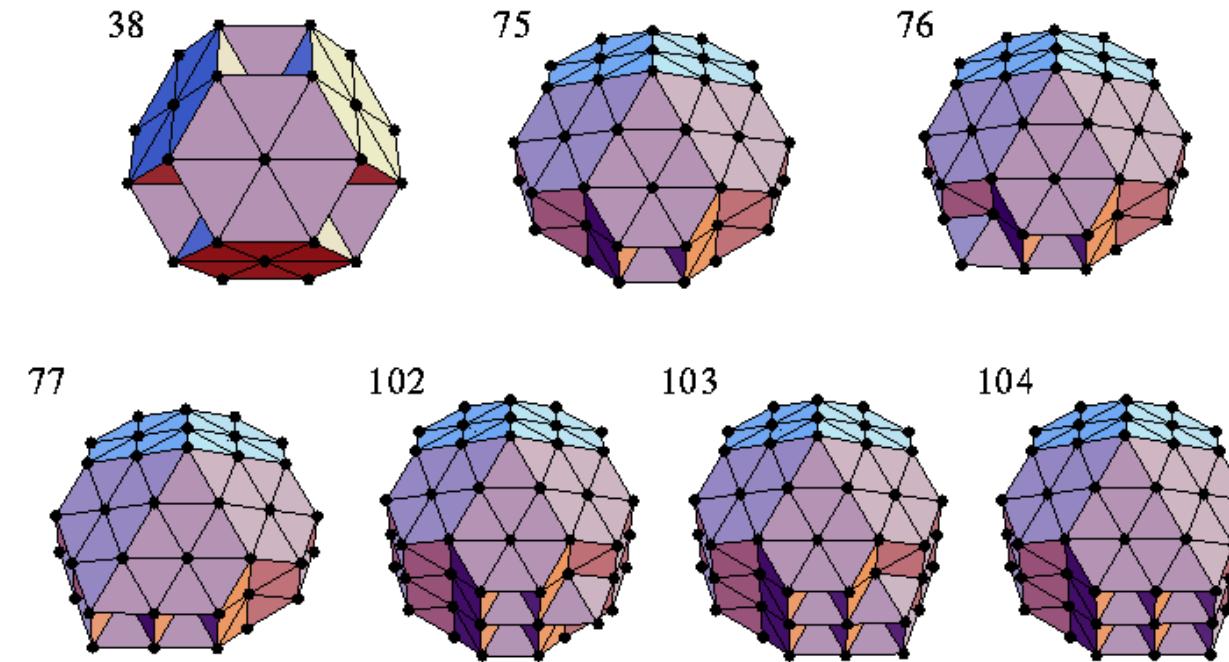
2003 → 2018  
532 → 71009  
531!/2 → 71008!/2





# Exponential multiplicity of inherent structures (potential energy minima)

NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".



Phys. Rev. Lett. **91**, 080201 (2003).

F. H. Stillinger, Phys. Rev. E **59**, 48 (1999).

[http://en.wikipedia.org/wiki/NP\\_hard](http://en.wikipedia.org/wiki/NP_hard)

<http://doye.chem.ox.ac.uk/jon/structures/LJ/pictures/LJ.nonicos.gif>

# Can We Design Personalized Chips?

*Systemic Complexity: 18-24 months and 100s of millions to bring a new chip to market*

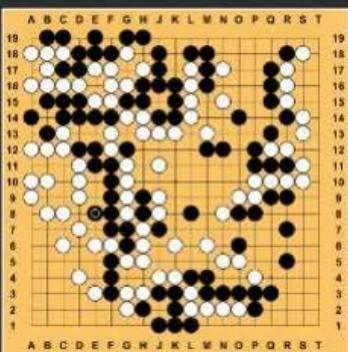
## Chess



Number  
of states:  $10^{123}$

Win / Lose

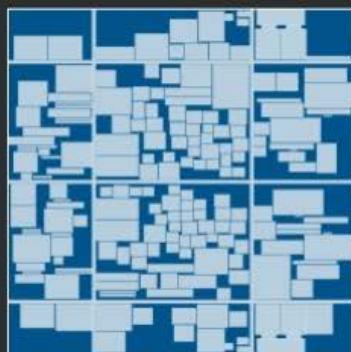
## Go



$\sim 10^{360}$

Win / Lose

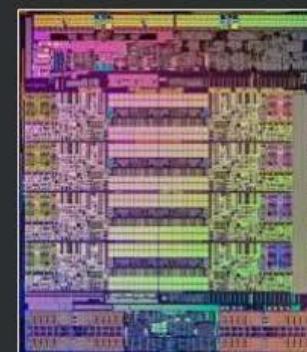
## Placement



$> 10^{90,000}$

Better / Worse

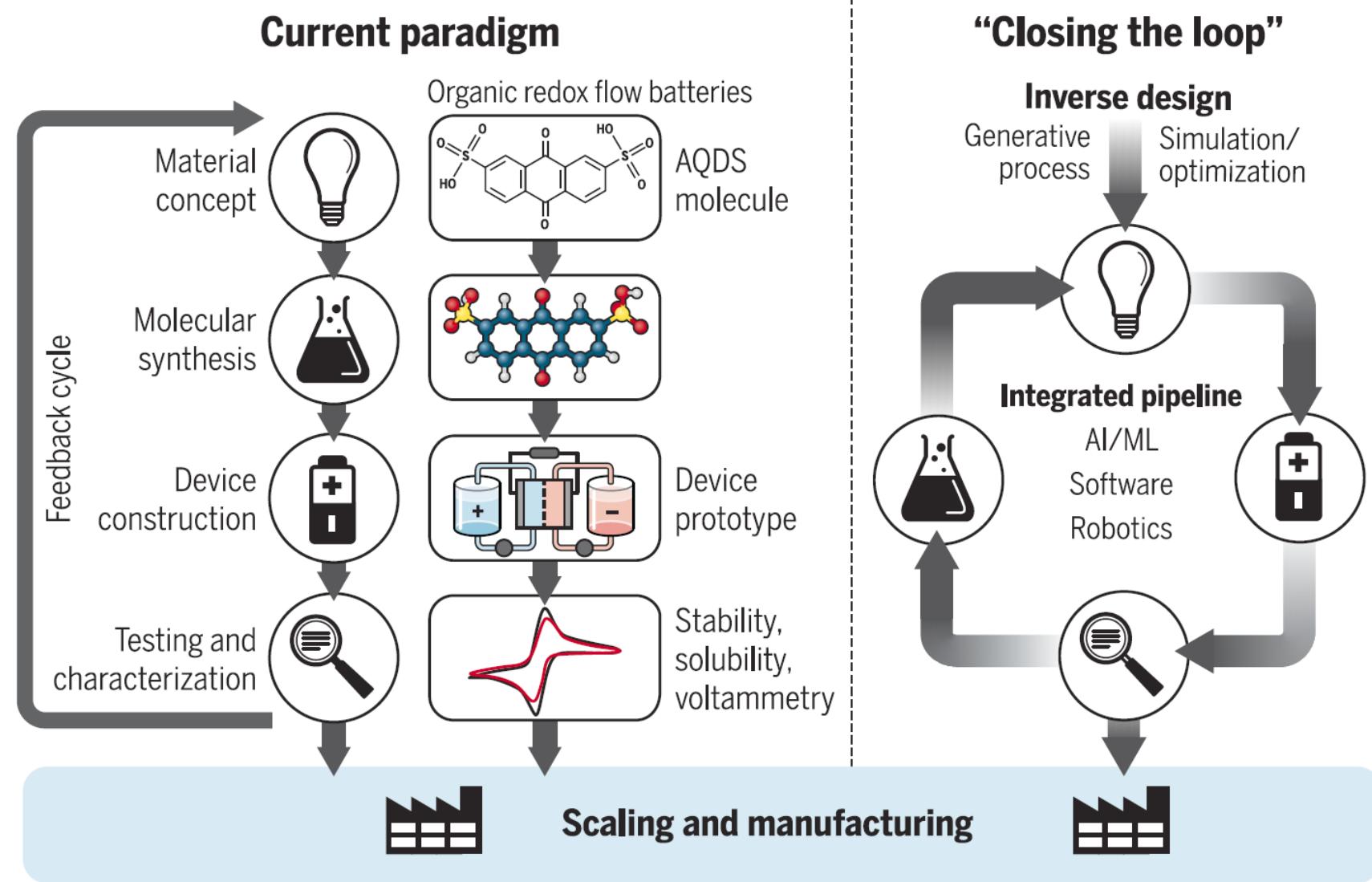
## Real Chips



$> 10^{??????}$

Impossible?

# Schematic comparison of material discovery paradigms





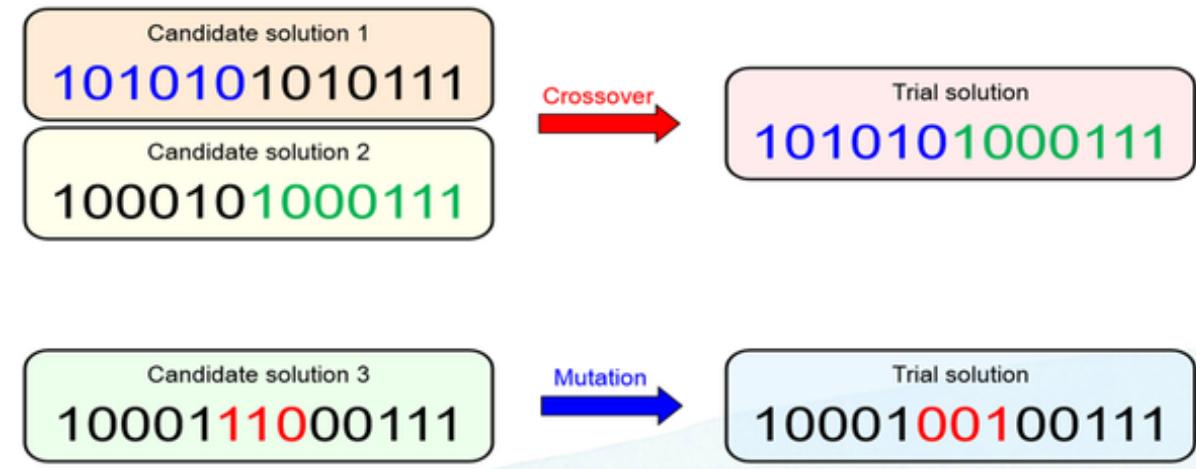
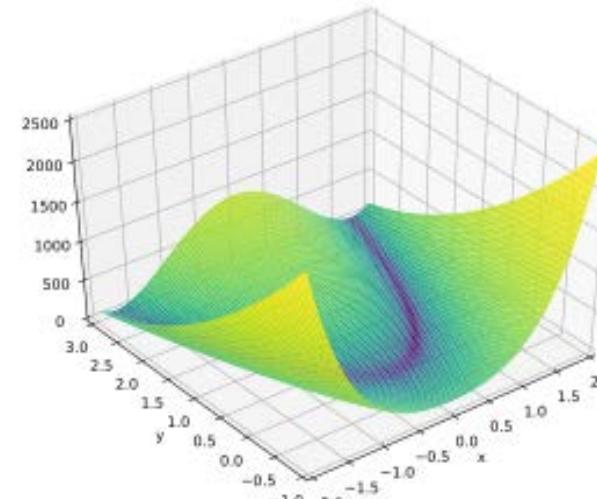
# Genetic algorithm (1/4)

```
import random
import numpy as np
from scipy.optimize import minimize

def functuser(x):
    case=3

    if case == 1:
        total=0.
        for j in range(len(x)):
            total+=(x[j])**2
    if case == 2:
        # Rastrigin
        total=10.*len(x)
        for j in range(len(x)):
            total+=x[j]**2-10.*np.cos(2.*np.pi*x[j])
    if case == 3:
        # Rosenbrock
        xarray0=np.zeros(len(x))
        for j in range(len(x)):
            xarray0[j]=x[j]
        total=sum(100.0*(xarray0[1:]-xarray0[:-1]**2.0)**2.0 + (1-xarray0[:-1])**2.0)
    if case == 4:
        # Styblinski-Tang
        total=0.
        for j in range(len(x)):
            total+=(x[j]**4-16.*x[j]**2+5.*x[j])/2.

    return total
```



# Genetic algorithm (2/4)

```
class PARTICLE:
    def __init__(self,startx0,ptbmp,pmut,pcross,xbounds,lverbo):
        self.position_i=[]
        self.position_best_i=[]
        self.obj_best_i=1e18
        self.obj_i=1e18
        self.dimensions=len(startx0)
        self.ptbmp=ptbmp+(random.random()-0.5)*3.5
        self.pmut=pmut+(random.random()-0.5)*0.1
        self.pcross=pcross+(random.random()-0.5)*0.1
        if self.pmut > 0.999 or self.pmut < 0.001:
            self.pmut=random.random()
        if self.pcross > 0.999 or self.pcross < 0.001:
            self.pcross=random.random()
        if lverbo:
            print(self.ptbmp,self.pmut,self.pcross)
        for j in range(self.dimensions):
            self.position_i.append(startx0[j]+(random.random()-0.5)*2.)
        if random.random() < 0.8:
            for j in range(self.dimensions):
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
        for j in range(self.dimensions):
            if self.position_i[j] > xbounds[j][1]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
            if self.position_i[j] < xbounds[j][0]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
        self.position_best_i=self.position_i.copy()
    def evaluate(self,objfunct):
        # self.obj_i=objfunct(self.position_i)
        xarray0=np.zeros(self.dimensions)
        for j in range(self.dimensions):
            xarray0[j]=self.position_i[j]
        res=minimize(objfunct,xarray0,method='nelder-mead',options={'xtol':1e-6,'disp':True})
        self.position_i=res.x.copy()
        self.obj_i=res.fun
        if self.obj_i < self.obj_best_i :
            self.position_best_i=self.position_i.copy()
            self.obj_best_i=self.obj_i
    def update_mutationcrossover(self,x1vec,x2vec):
        if random.random() < 0.5:
            for j in range(self.dimensions):
                self.position_i[j]=x1vec[j]
            if random.random() < self.pmut:
                self.position_i[j]=x1vec[j]+(random.random()-0.5)*self.ptbmp
        else:
            for j in range(self.dimensions):
                self.position_i[j]=x1vec[j]
            if random.random() < self.pcross:
                self.position_i[j]=x2vec[j]
    def update_position(self,xbounds):
        for j in range(self.dimensions):
            if self.position_i[j] > xbounds[j][1]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
            if self.position_i[j] < xbounds[j][0]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
```

# Genetic algorithm (3/4)

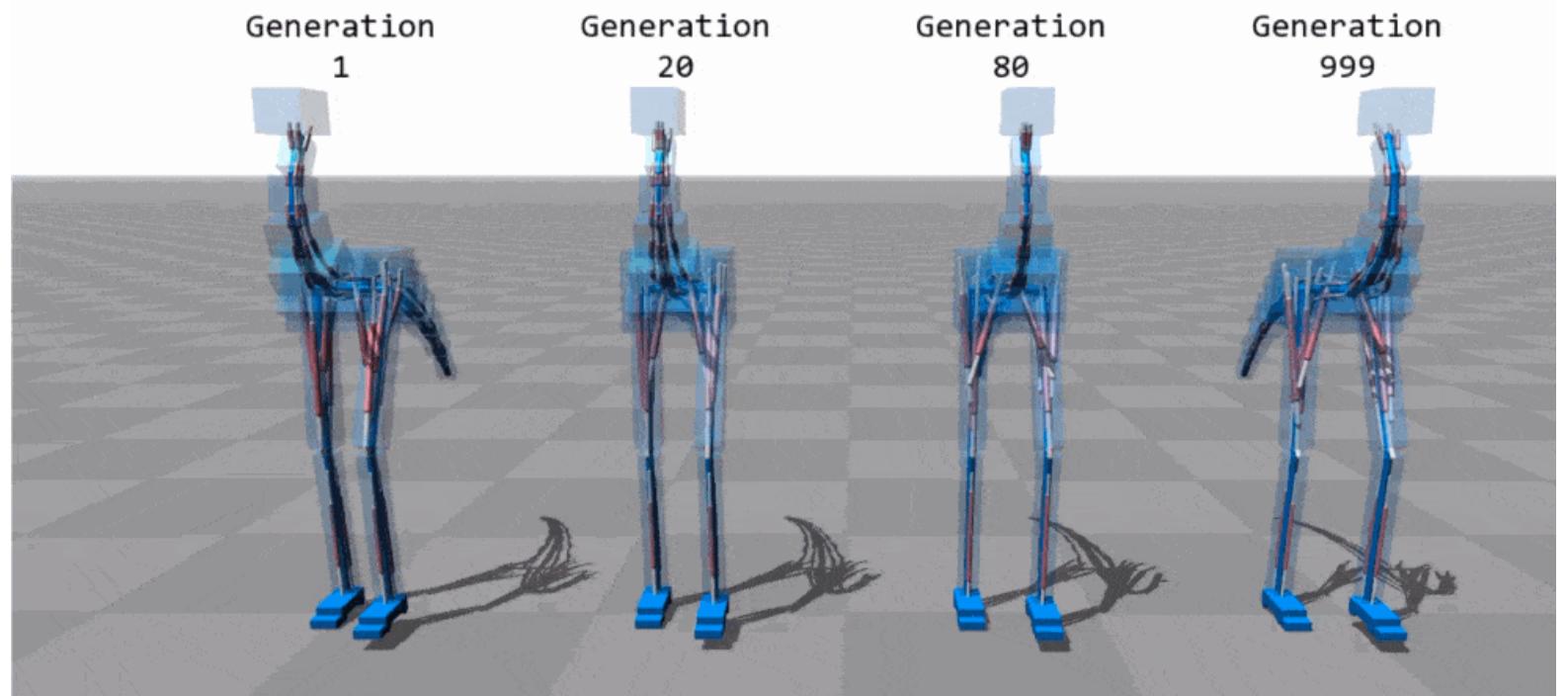
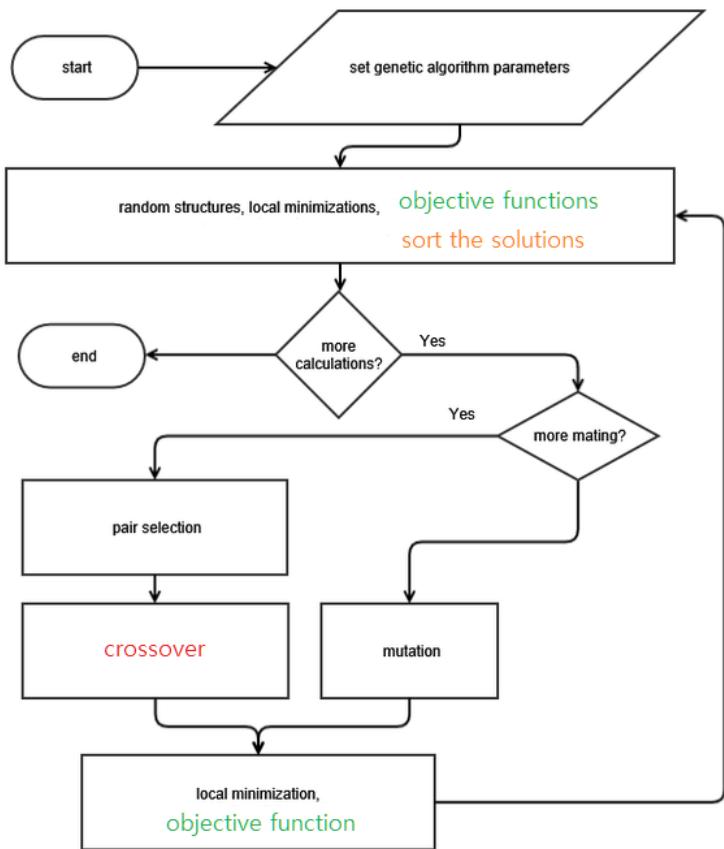
```

class GA():
    def __init__(self, objfunct, startx0, xbounds, ptbmp, pmut, pcross, nparticles, maxiter,
    verbose=False):
        obj_best_g=1e18
        position_best_g=[]
        swarm=[]
        x1vec=[]
        x2vec=[]
        nsubpop=0
        for _ in range(nparticles):
            swarm.append(PARTICLE(startx0,ptbmp,pmut,pcross,xbounds,verbose))
        it=0
        while it < maxiter:
            if verbose:
                print(f'iter: {it}>6d} best solution: {obj_best_g:16.8e}')
            if True and nparticles > 4:
                print('lowest five')
                abc=np.zeros(nparticles)
                abcvec=np.zeros((nparticles,len(startx0)))
                for i in range(nparticles):
                    abc[i]=swarm[i].obj_best_i
                    abcvec[i]=swarm[i].position_best_i
                idx=abc.argsort()
                abc=abc[idx]
                abcvec=abcvec[idx,:]
                print(abc[0],abc[1],abc[2],abc[3],abc[4])
                print(abcvec[0,:])
                print(abcvec[1,:])
                print(abcvec[2,:])
                print(abcvec[3,:])
                print(abcvec[4,:])
            for i in range(nparticles):
                swarm[i].evaluate(objfunct)
            if swarm[i].obj_i < obj_best_g :
                position_best_g=list(swarm[i].position_best_i)
                obj_best_g=float(swarm[i].obj_i)
            for i in range(nparticles):
                i1=int(random.random()*nparticles) ; i2=int(random.random()*nparticles) ; k1=i2
                if swarm[i1].obj_best_i < swarm[i2].obj_best_i :
                    k1=i1
                for _ in range(nsubpop-1):
                    i1=int(random.random()*nparticles)
                    if swarm[i1].obj_best_i < swarm[k1].obj_best_i :
                        k1=i1
                    i1=int(random.random()*nparticles) ; i2=int(random.random()*nparticles) ; k2=i2
                    if swarm[i1].obj_best_i < swarm[i2].obj_best_i :
                        k2=i1
                    for _ in range(nsubpop-1):
                        i1=int(random.random()*nparticles)
                        if swarm[i1].obj_best_i < swarm[k2].obj_best_i :
                            k2=i1
                        x1vec=list(swarm[k1].position_best_i)
                        x2vec=list(swarm[k2].position_best_i)
                        swarm[i].update_mutationcrossover(x1vec,x2vec)
                        swarm[i].update_position(xbounds)
                it+=1
                print('\nfinal solution:')
                print(f' > {position_best_g}')
                print(f' > {obj_best_g}\n')
            if True:
                abc=np.zeros(nparticles)
                abcvec=np.zeros((nparticles,len(startx0)))
                for i in range(nparticles):
                    abc[i]=swarm[i].obj_best_i
                    abcvec[i]=swarm[i].position_best_i
                idx=abc.argsort()
                abc=abc[idx]
                abcvec=abcvec[idx,:]
                for i in range(nparticles):
                    print(abc[i])
                    print(abcvec[i,:])

```

# Genetic algorithm (4/4)

```
startx0=[]
xbounds=[]
for j in range(10):
    startx0.append(0.)
for j in range(len(startx0)):
    xbounds.append((-20., 20.))
ptbmp=0.10
pmut=0.50
pcross=0.50
GA(functuser, startx0, xbounds, ptbmp, pmut, pcross, nparticles=50, maxiter=50000, verbose=True)
```





# Particle Swarm Optimization (1/2)

```
class PARTICLE:
    def __init__(self,startx0,ww,c1,c2,xbounds,lverb0):
        self.position_i=[]
        self.velocity_i=[]
        self.position_best_i=[]
        self.obj_best_i=1e18
        self.obj_i=1e18
        self.dimensions=len(startx0)
        self.ww=ww+(random.random()-0.5)*0.2
        self.c1=c1+(random.random()-0.5)*0.2*1.
        self.c2=c2+(random.random()-0.5)*0.2*1.
        if lverb0:
            print(self.ww,self.c1,self.c2)
        for j in range(self.dimensions):
            self.velocity_i.append(random.uniform(-1,1))
            self.position_i.append(startx0[j]+(random.random()-0.5)*2.)
        if random.random()<0.8:
            for j in range(self.dimensions):
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
        for j in range(self.dimensions):
            if self.position_i[j] > xbounds[j][1]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
            if self.position_i[j] < xbounds[j][0]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
        self.position_best_i=self.position_i.copy()
    def evaluate(self,objfunct):
        # self.obj_i=objfunct(self.position_i)
        xarray0=np.zeros(self.dimensions)
        for j in range(self.dimensions):
            xarray0[j]=self.position_i[j]
        res=minimize(objfunct,xarray0,method='nelder-mead',options={'xtol':1e-6,'disp':True})
        self.position_i=res.x.copy()
        self.obj_i=res.fun
        if self.obj_i < self.obj_best_i:
            self.position_best_i=self.position_i.copy()
            self.obj_best_i=self.obj_i
    def update_velocity(self,position_best_g):
        for j in range(self.dimensions):
            vc=self.c1*(self.position_best_i[j]-self.position_i[j])*random.random()
            vs=self.c2*(position_best_g[j]-self.position_i[j])*random.random()
            self.velocity_i[j]=self.ww*self.velocity_i[j]+vc+vs
    def update_position(self,xbounds):
        for j in range(self.dimensions):
            self.position_i[j]=self.position_i[j]+self.velocity_i[j]
            if self.position_i[j] > xbounds[j][1]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
            if self.position_i[j] < xbounds[j][0]:
                self.position_i[j]=xbounds[j][0]+(xbounds[j][1]-xbounds[j][0])*random.random()
```

# Particle Swarm Optimization (2/2)

```
class PSO():
    def __init__(self, objfunct, startx0, xbounds, ww, c1, c2, nparticles, maxiter, verbose=False):
        obj_best_g=1e18
        position_best_g=[]
        swarm=[]
        for _ in range(nparticles):
            swarm.append(PARTICLE(startx0,ww,c1,c2,xbounds,verbose))
        it=0
```

while it < maxiter:

```
    if verbose:
        print(f'iter: {it}>6d} best solution: {obj_best_g:16.8e}')
```

for i in range(nparticles):

```
    swarm[i].evaluate(objfunct)
```

if swarm[i].obj\_i < obj\_best\_g:

```
    position_best_g=list(swarm[i].position_i)
    obj_best_g=float(swarm[i].obj_i)
```

for i in range(nparticles):

```
    swarm[i].update_velocity(position_best_g)
```

```
    swarm[i].update_position(xbounds)
```

it+=1

print('final solution:')

print(f' >{position\_best\_g}')

print(f' >{obj\_best\_g}\n')

if True:

```
    abc=np.zeros(nparticles)
```

```
    abcvec=np.zeros((nparticles,len(startx0)))
```

for i in range(nparticles):

```
    abc[i]=swarm[i].obj_best_i
```

```
    abcvec[i]=swarm[i].position_best_i
```

idx=abc.argsort()

abc=abc[idx]

abcvec=abcvec[idx,:]

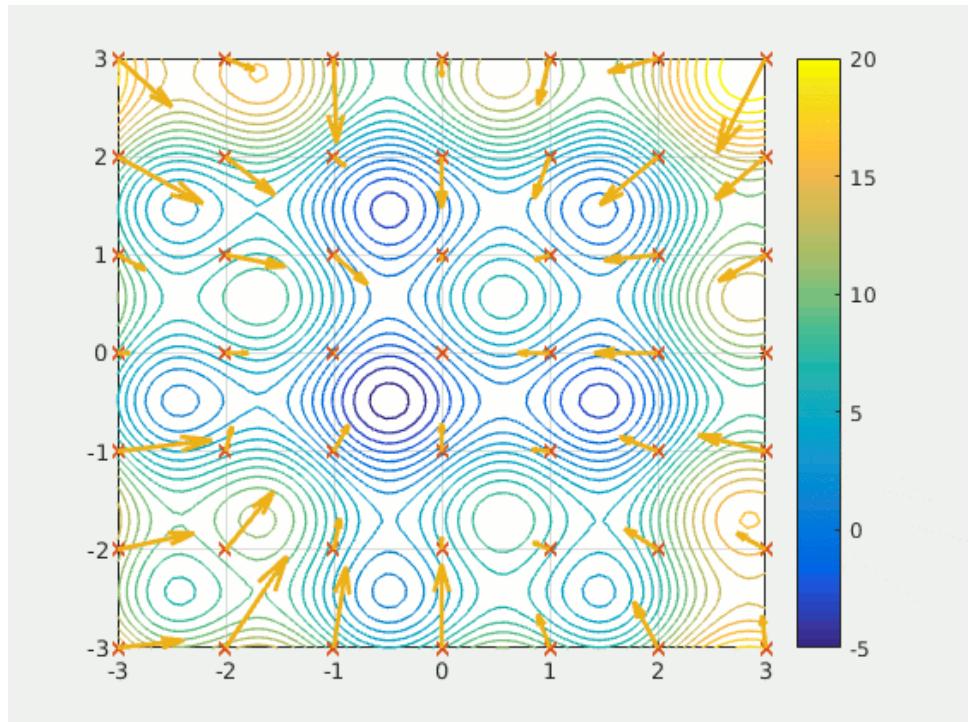
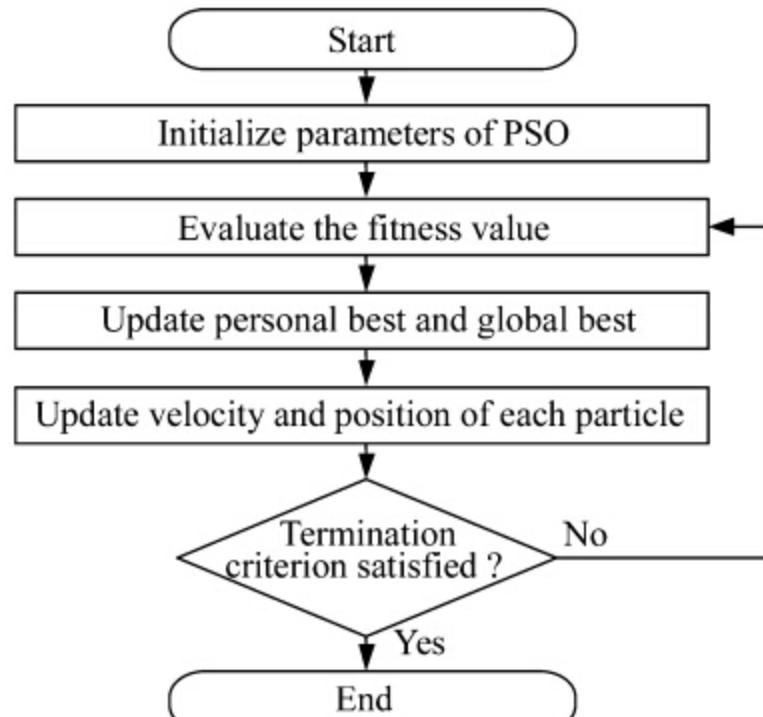
for i in range(nparticles):

```
    print(abc[i])
```

print(abcvec[i,:])

```
startx0=[]
xbounds=[]
for j in range(10):
    startx0.append(0.)
for j in range(len(startx0)):
    xbounds.append((-20., 20.))
ww=0.5
c1=1.0
c2=2.0
```

PSO(functuser, startx0, xbounds, ww, c1, c2, nparticles=50, maxiter=50000, verbose=True)



# Contents

- \*Modern introduction (3+3)
- \*Scikit-learn and Keras tutorial (3+3+3+3+3)
- \*Projects/Applications(3)

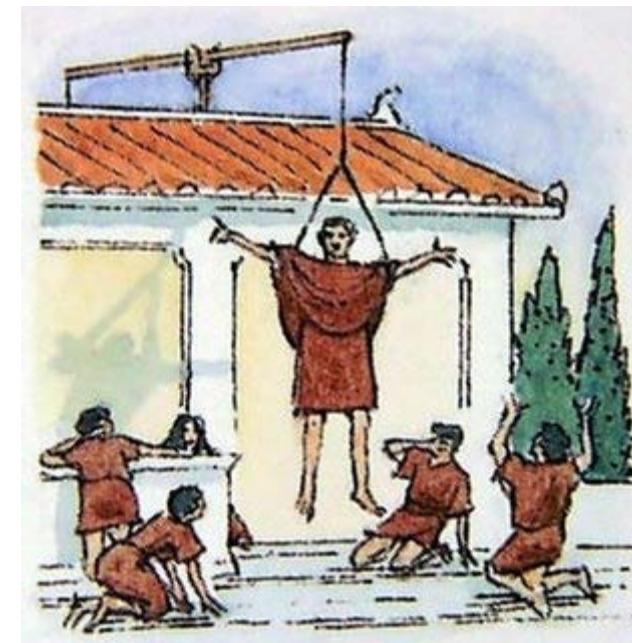




# 물질게놈, 기계학습 특집

- I.-H. Lee, J. Lee, and K. J. Chang, [Design and discovery of super functional materials: methods and applications](#), 물리학과 첨단기술 2017년 9월호, 2-11 페이지
- I.-H. Lee, [Modern Genetic Algorithms](#), 물리학과 첨단기술 2018년 1/2월호, 8-11 페이지

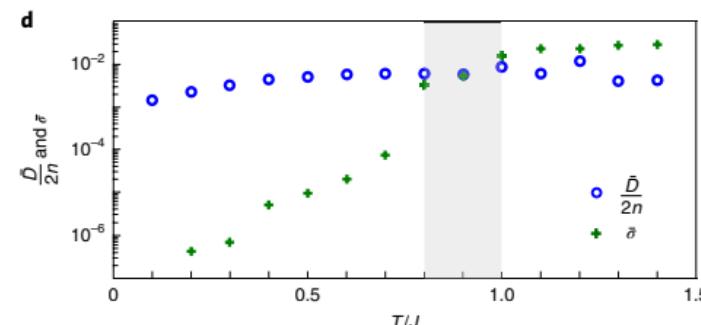
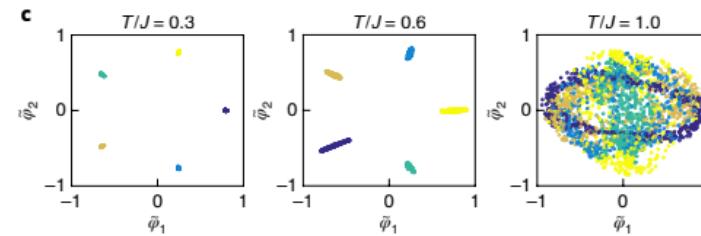
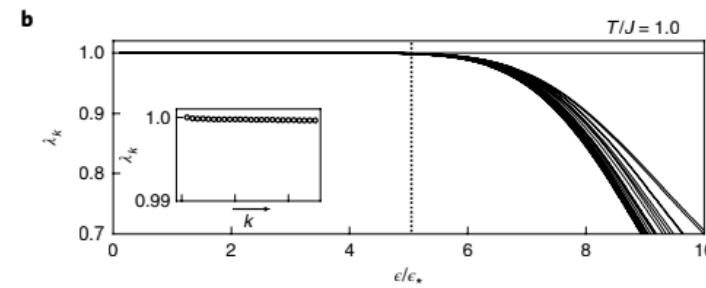
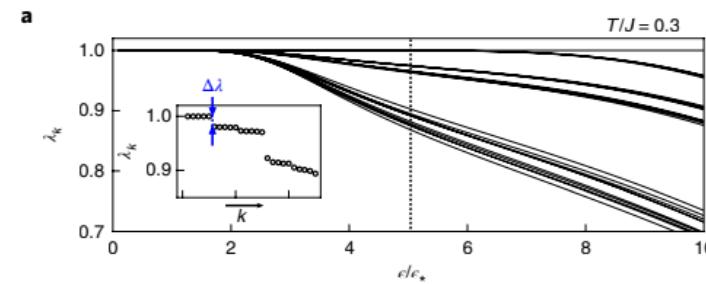
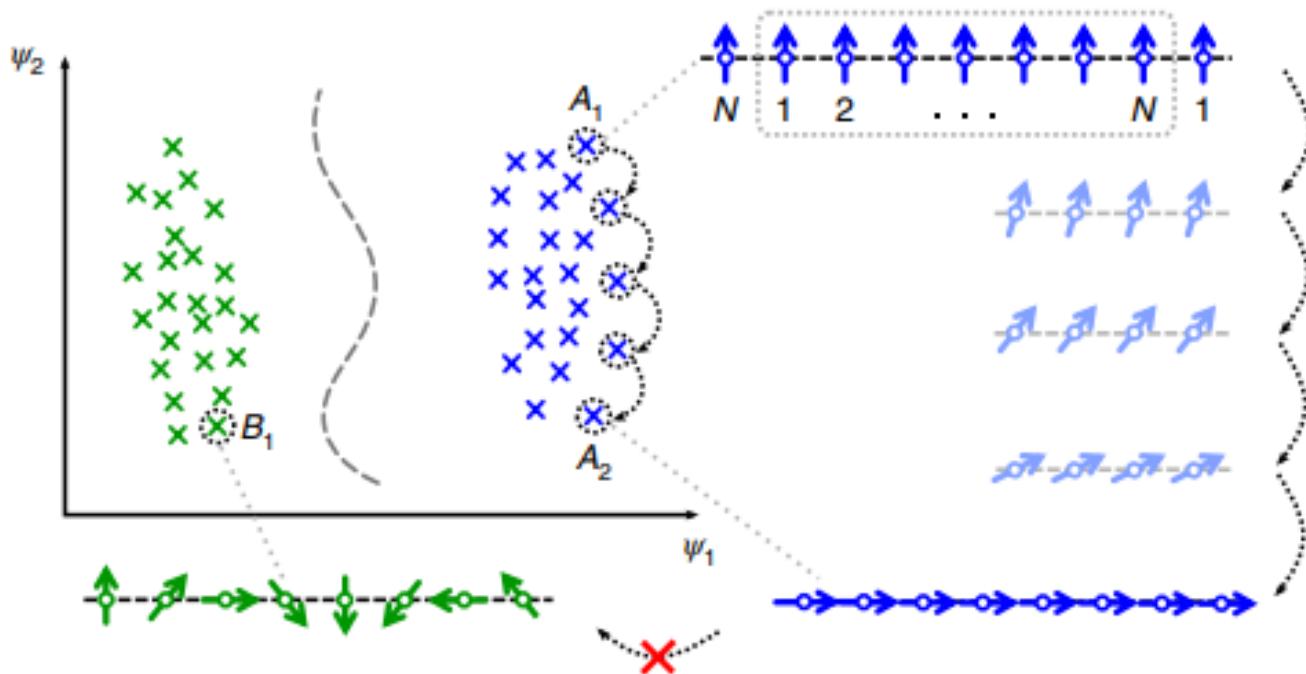
## DEUS EX MACHINA





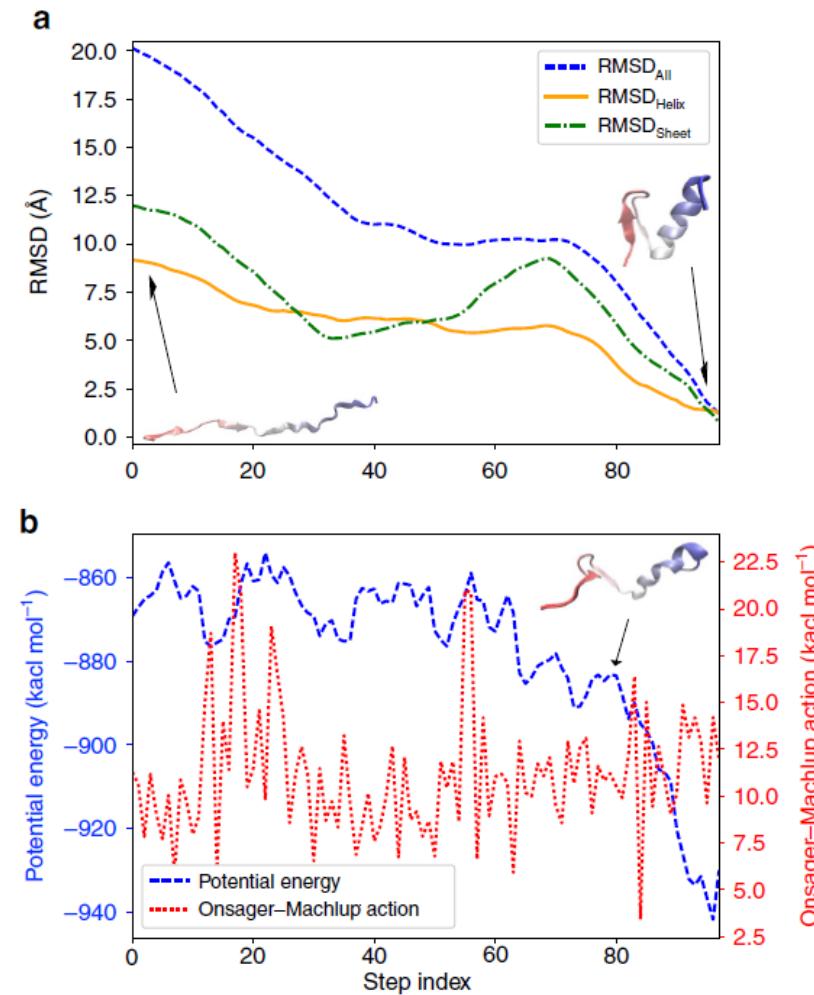
# Identifying topological order through unsupervised machine learning

## Diffusion map



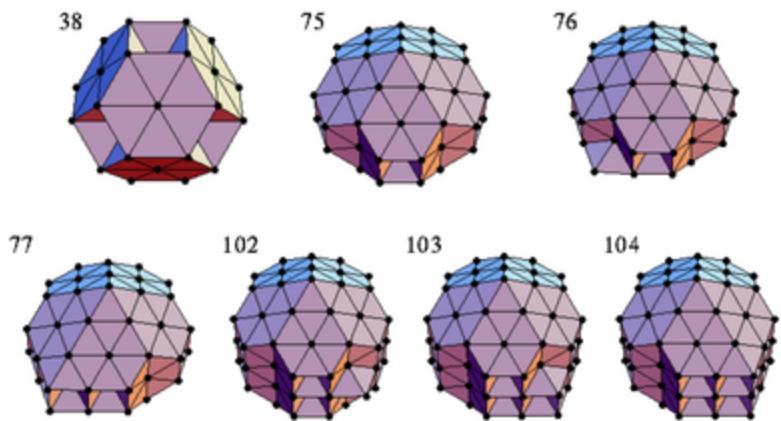


$$S_{OM} = \frac{\Delta V}{2} + \frac{1}{4} \int_0^t d\tau [\gamma m \dot{x}^2 + |\nabla V(x(\tau))|^2 / (\gamma m) - 2k_B T / (\gamma m) \nabla^2 V(x(\tau))]$$



## Exponential multiplicity of inherent structures (potential energy minima)

NP-hard (Non-deterministic Polynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP".



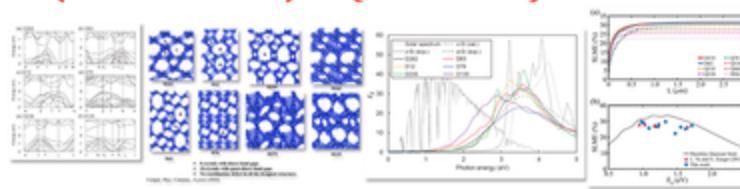
The inverse band-structure problem of finding an atomic configuration with given electronic properties

Process

$$f^{-1}(\text{Property}) = \{\text{Processing, Composition, Structure}\},$$

$$f^{-1}(\text{Performance}) = \{\text{Processing, Structure}\},$$

$$f^{-1}(\text{Performance}) = \{\text{Structure}\}.$$

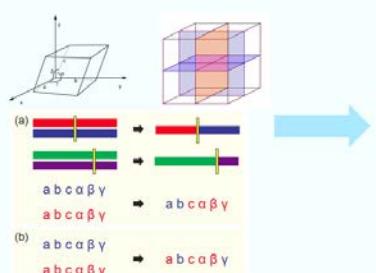


Atomic configuration  $\rightarrow$  Electronic structure (1)

Electronic structure  $\rightarrow$  Atomic configuration (2)

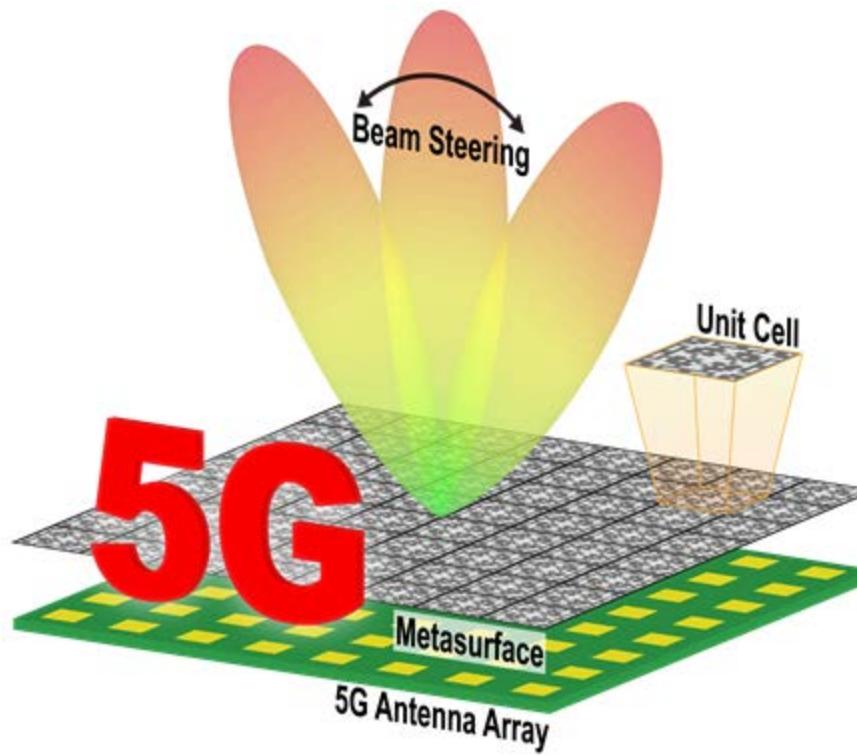
Objective function  $[a, b, c, \alpha, \beta, \gamma, \{R_j\}]$

- There could always be a better structure in a larger unit cell.
- We can predict structures that are far away from what chemical intuition would suggest.

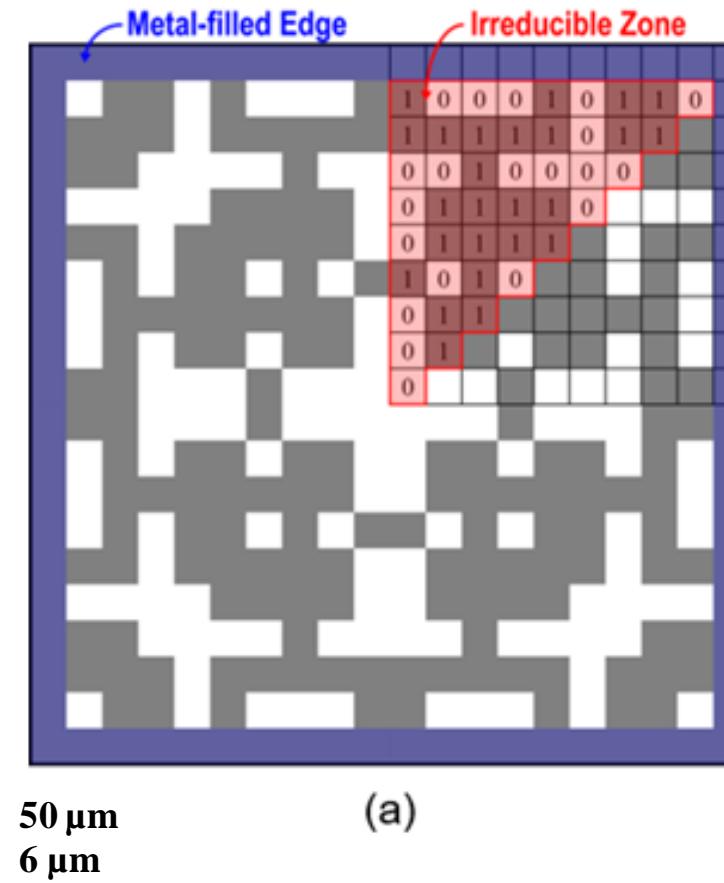




# Design of single-layer metasurface filter by CSA for 5G mm-wave communications

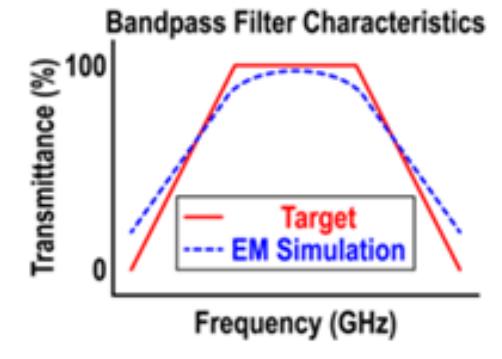


5G Antenna Array with Metasurface

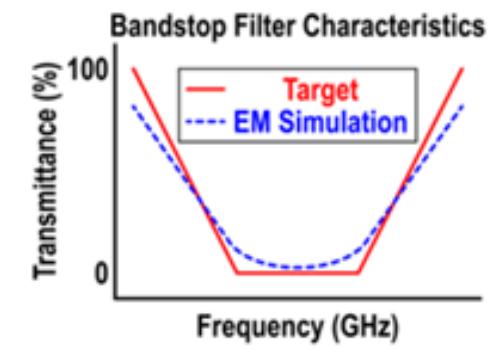


50  $\mu\text{m}$   
6  $\mu\text{m}$

(a)



(b)



(c)



$$2^{351} = 4586997231980143023221641790604173881593129978336562247475177678773845752176969616140037106220251373109248$$

$\sim 4.5 \times 10^{105}$

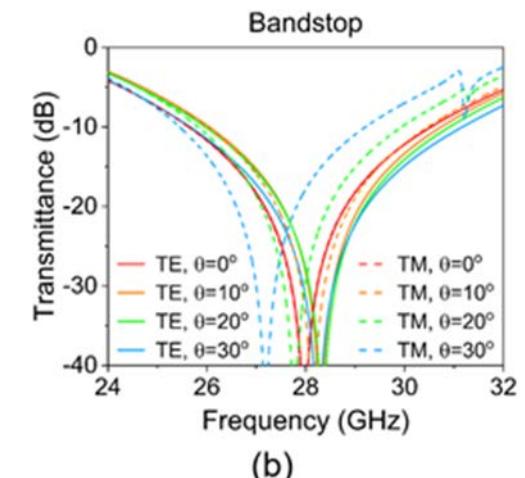
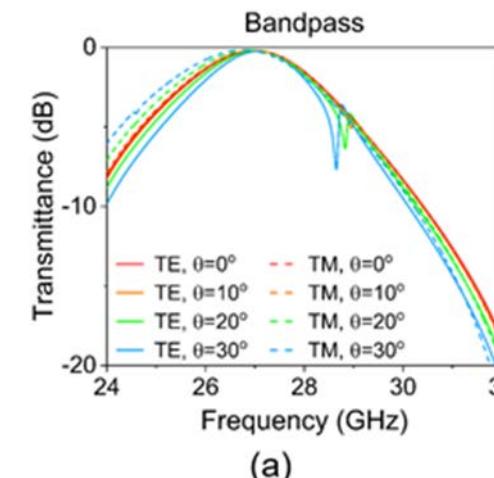
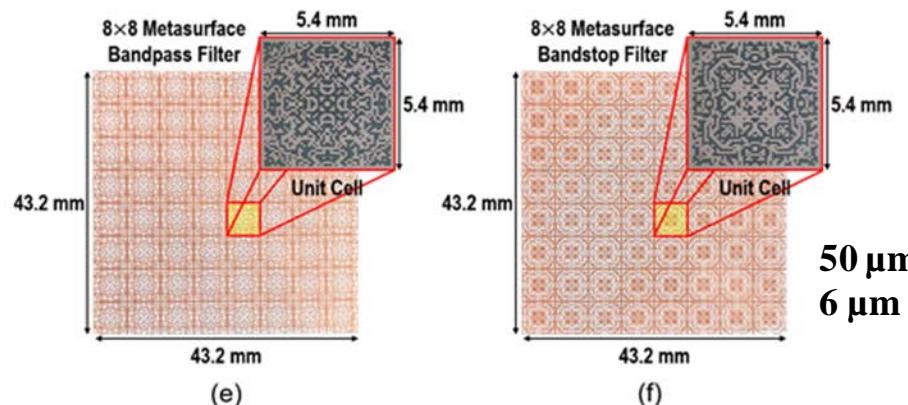
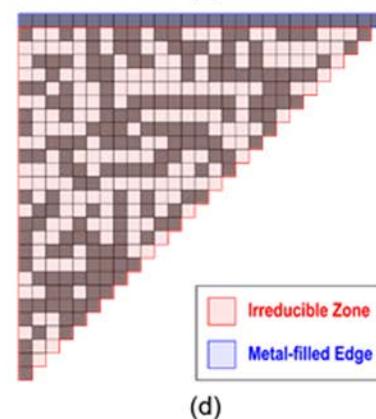
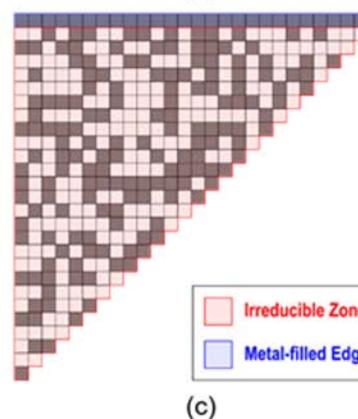
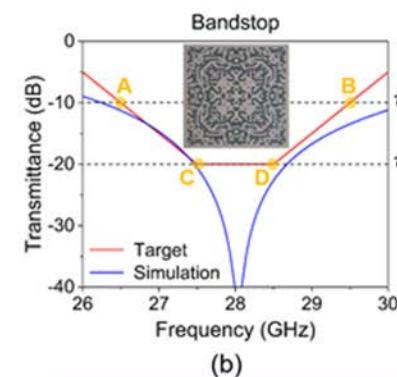
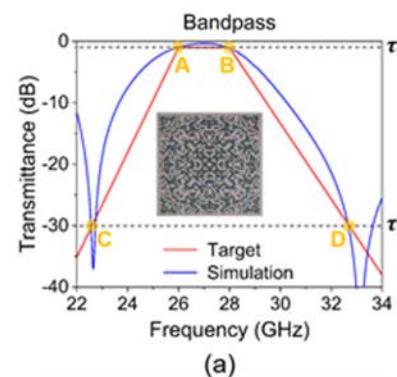
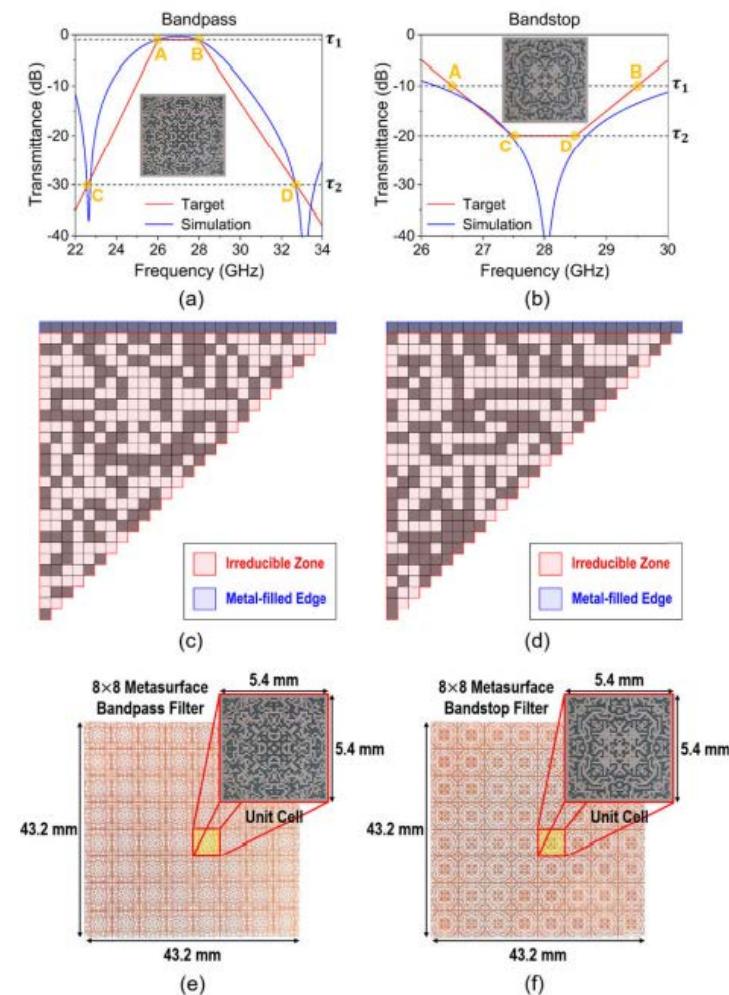
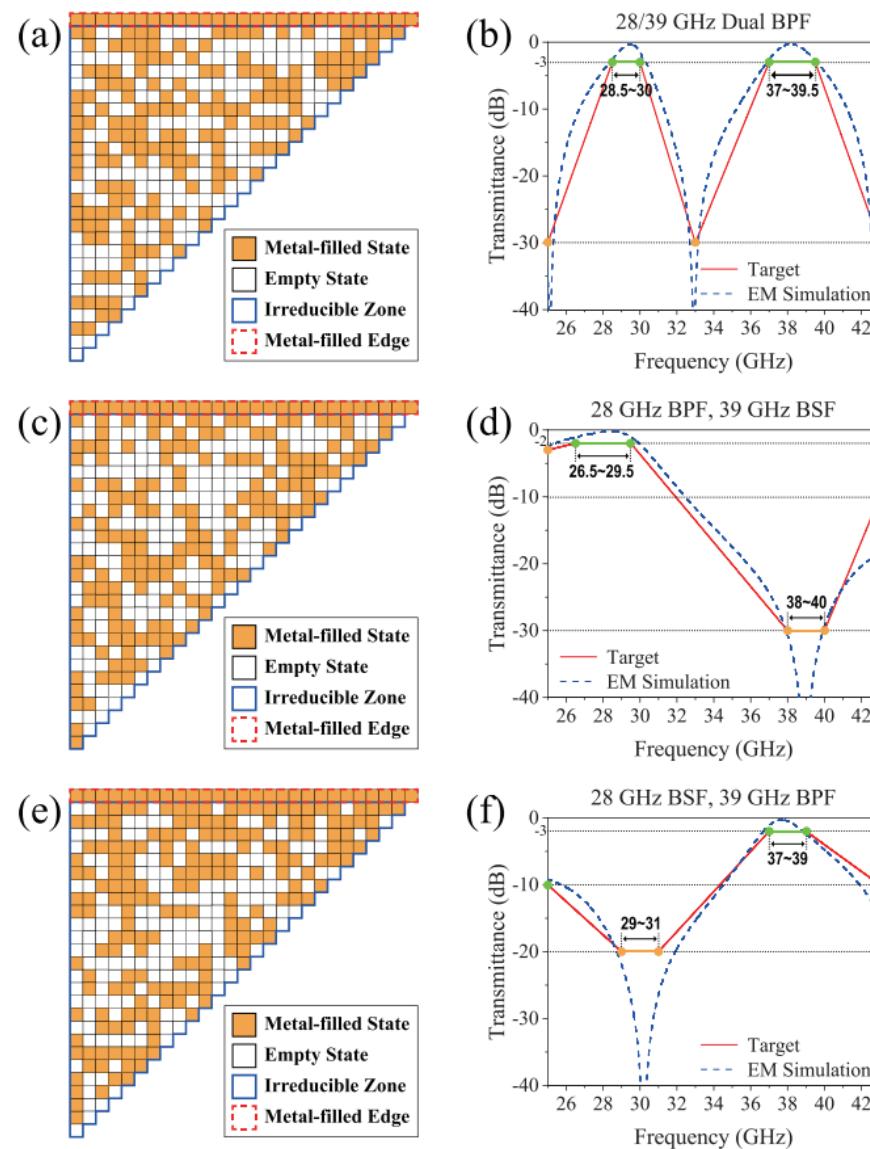


Table: Gain / loss in decibels

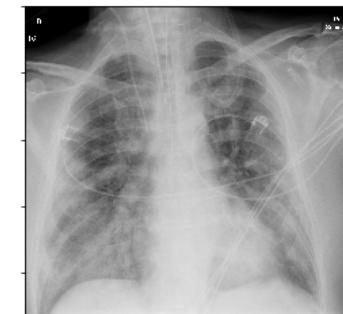
Loss/gain as a ratio	Loss/gain in decibels	Loss/gain as a ratio	Loss/gain in decibels
$\frac{P_{\text{output}}}{P_{\text{input}}}$	$10 \log \frac{P_{\text{output}}}{P_{\text{input}}}$	$\frac{P_{\text{output}}}{P_{\text{input}}}$	$10 \log \frac{P_{\text{output}}}{P_{\text{input}}}$
1000	30 dB	0.1	-10 dB
100	20 dB	0.01	-20 dB
10	10 dB	0.001	-30 dB
1 (no loss or gain)	0 dB	0.0001	-40 dB



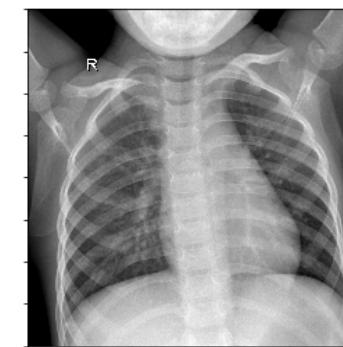




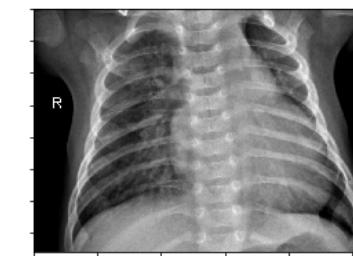
# COVID19, X-ray : classification



COVID19



Normal



Other pneumonia



# Denoising

Noise image: 101.png

There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

Denoised image: 101.png

There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

Noise image: 104.png

There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

Denoised image: 104.png

There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

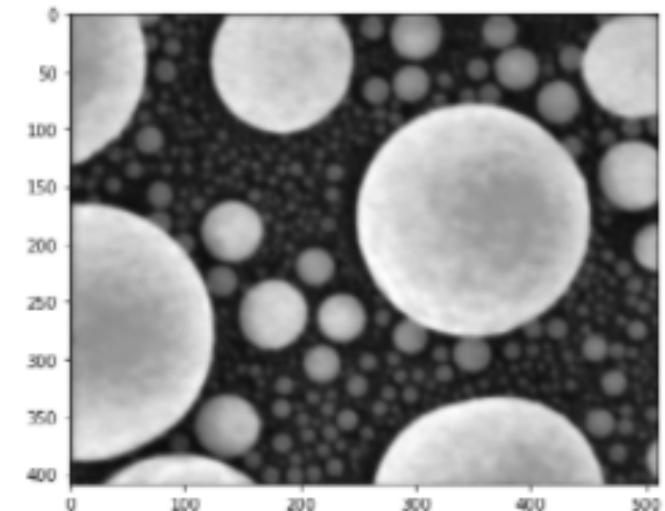
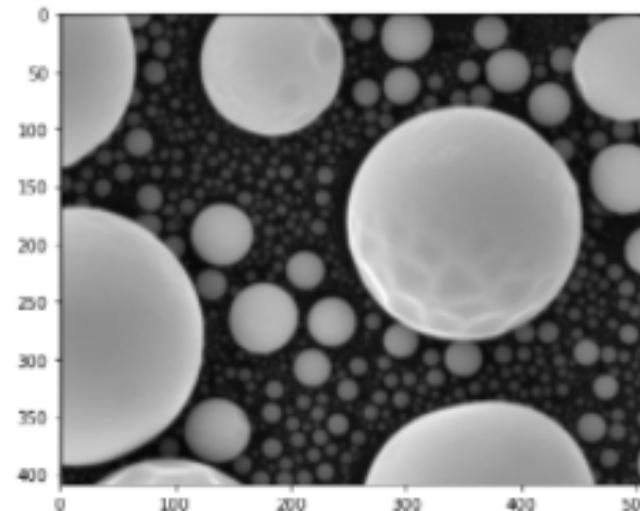
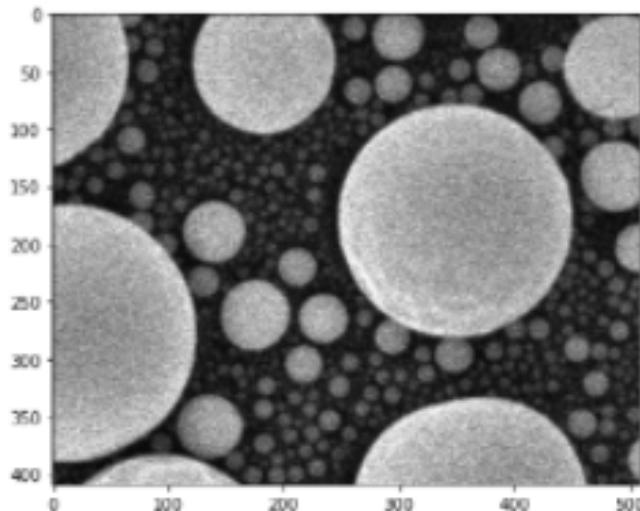
Noise image: 107.png

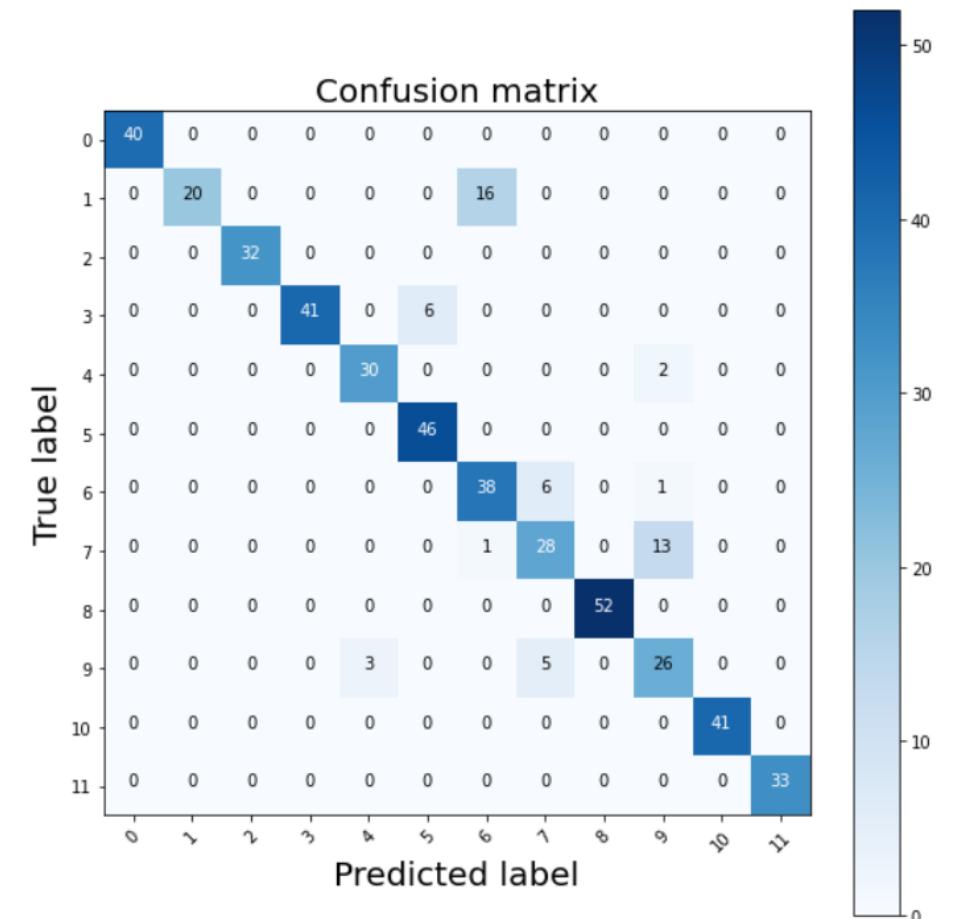
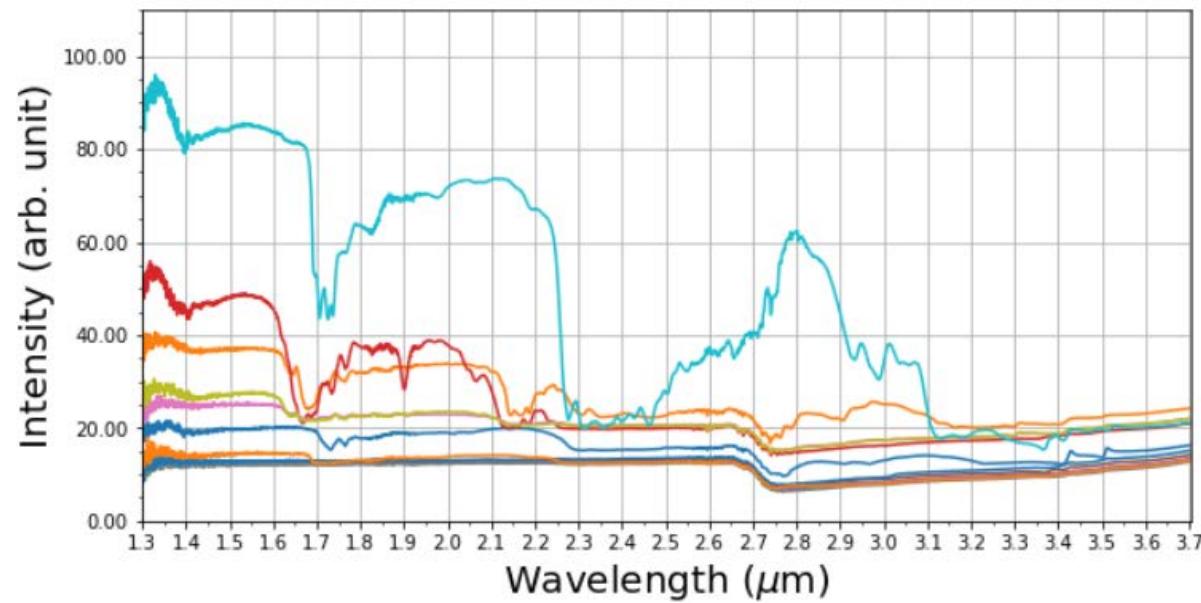
There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

Denoised image: 107.png

There exist several methods to design fields to be filled in. For instance, fields may be surrounding boxes, by light rectangles or by guiding rules specify where to write and, therefore, none of skew and overlapping with other parts of guides can be located on a separate sheet located below the form or they can be printed on the form. The use of guides on a separate sheet from the point of view of the quality of the form is good, but it is not always possible, but requires giving more instructions and, restricts its use to tasks where this type of a

# Denoising

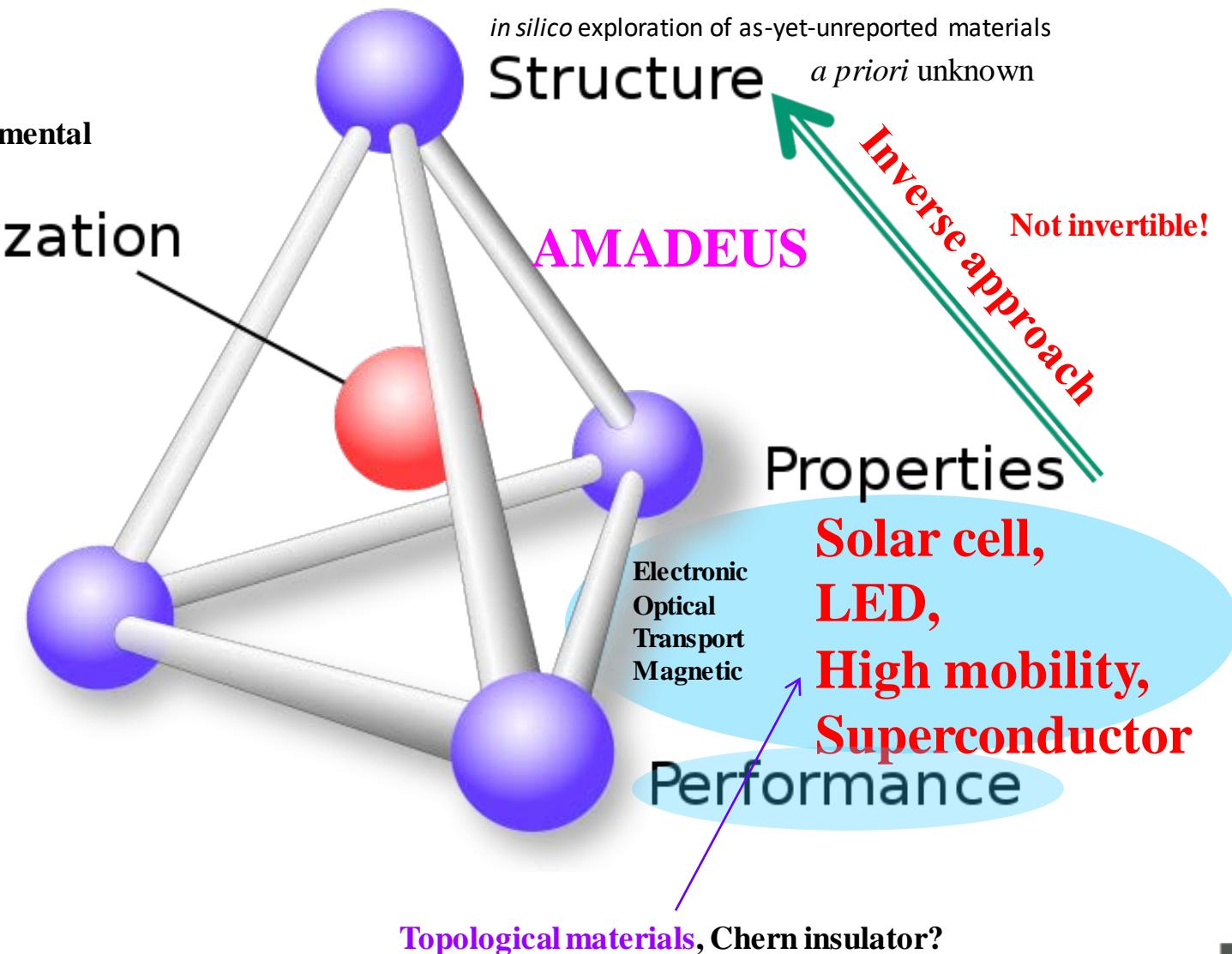




# The understanding of processing-structure-properties relationships is called the materials paradigm.

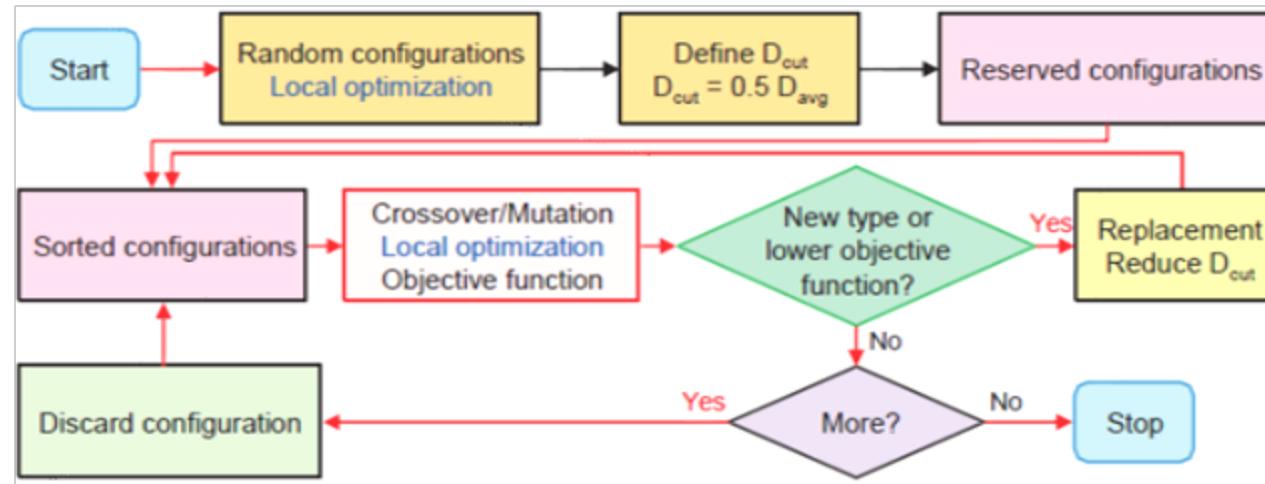
Materials Project → computational  
Cambridge Structural Database → experimental

Characterization  
Processing



# Development of AMADEUS code

AMADEUS (*Ab initio* MAterials DEnsiOn Using conformational Space annealing)



## Conformational space annealing (CSA) for global optimization

with three ingredients [Phys. Rev. Lett. **91**, 080201 (2003).]

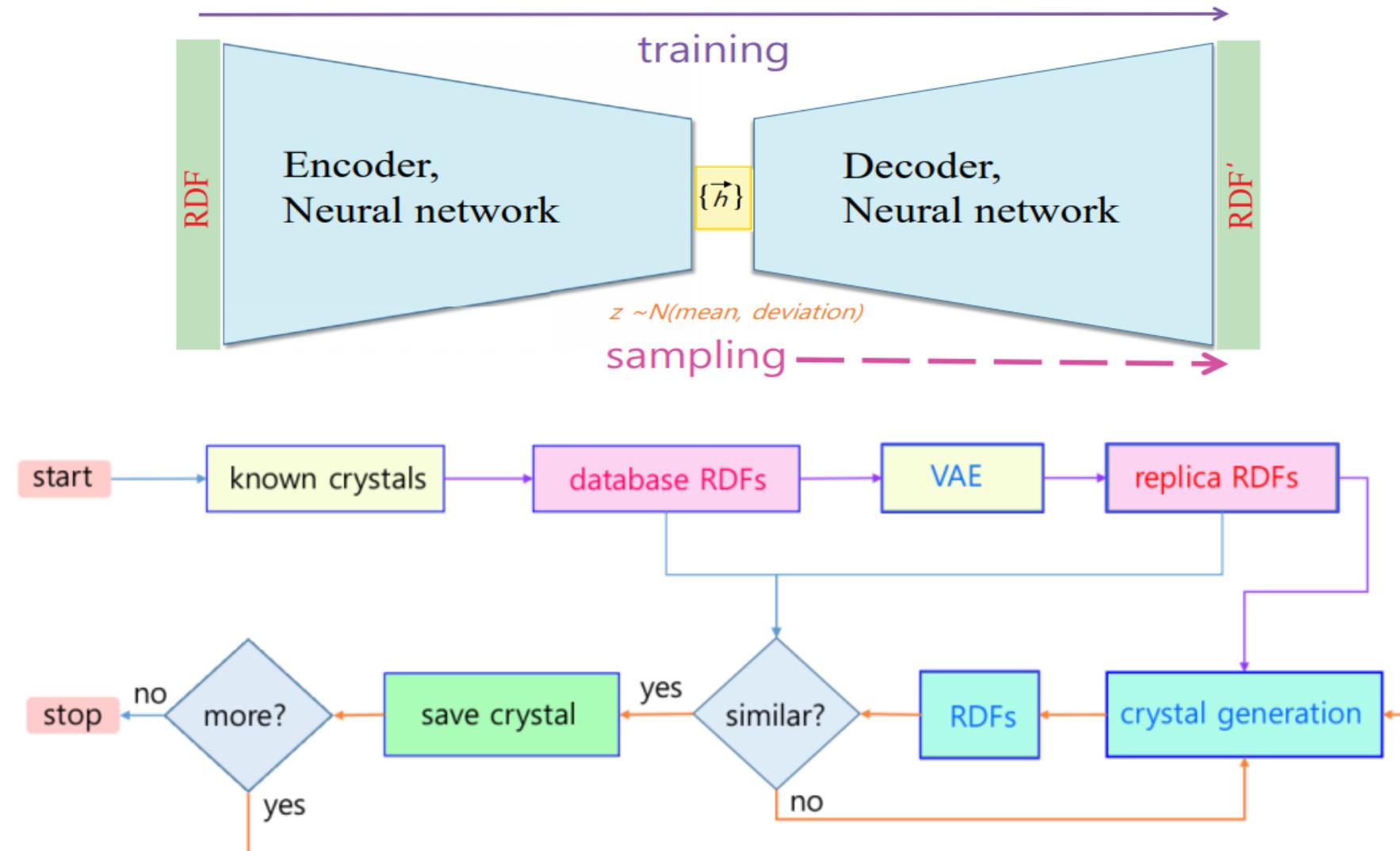
- Genetic algorithm
- Simulated annealing
- Local minimization

First-principles electronic structure calculations  
based on density functional theory

Mechanical stability  
Dynamical stability  
Thermal stability

VASP  
Quantum Espresso  
Wannier90  
WannierTools  
Phonopy  
VESTA  
PyProcar

# Crystal structure prediction in a continuous representation space



# Applications

- **Solar cell (Si):** Phys. Rev. B **90**, 115209 (2014), Sci. Rep. **5**, 18086 (2015), Comp. Phys. Commun. **203**, 110 (2016).
- **LED (C):** Phys. Rev. B **93**, 085201 (2016).
- **High-mobility [green phosphorus] (P):** J. Phys. Chem. Lett. **8**, 4627 (2017).
- **Topological materials (Si, C,  $\text{Ag}_2\text{Se}_{0.5}\text{Te}_{0.5}$ ):** J. Phys. Chem. C **123**, 1839 (2019), NPG Asia Materials **9**, e361 (2017), Nanoscale **11**, 5171 (2019).
- **Structure searches (B):** Sci. Rep. **7**, 7279 (2017), NPG Asia Materials **9**, e400 (2017), arXiv:1902.08390
- **Superconductivity (Si):** Phys. Rev. Lett. **120**, 157001 (2018).
- **Topological superconductivity (Si):** Phys. Rev. Mat. **5**, 104802 (2021).

# Projects/Applications

Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning

Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules

Science **365**, eaaw1147 (2019).

<https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

ACS Cent. Sci. **4**, 268 (2018).

<https://science.sciencemag.org/content/355/6325/602>