# Numerical Analysis for Artificial Intelligence

UCSD Summer session II 2018

CSE 190

Jacek Cyranka

# The goal of this course

The goal of the course is to study mathematical fundamentals of machine learning in particular neural networks.
The emphasis will be given to optimization techniques.

# Example benefits of completing the course

- Deeper understanding from where the *power and limitations of Neural Networks* is coming from,
- See that applied math is really applied,
- Open a path for research in Machine Learning,

# Lecture Plan for Week 1&2

- *Review of Programming in Python+NumPy+IPython notebook and Calculus and Linear Algebra topics*
  - Python language basics,
  - Linear Algebra in NumPy,
  - Working with Jupyter notebooks,
  - Example problem of solving a linear regression analytically,
  - Functions,
  - Vector spaces,
  - Matrices,
  - Matrix times vector/matrix operation,
  - Matrix transpose/inverse,
  - Solving systems of linear equations,
  - Basic properties,
  - Partial Derivatives,
  - Critical points,
  - Chain rule and gradients,
  - Characterization of critical points as local/global minima/maxima.

# Lecture Plan for Week 3

- *Gradient descent and convex optimization*
  - Backpropagation algorithm,
  - gradient checking of a backpropagation implementation,
  - Avoiding problems with convergence by decreasing the learning rate,
  - *Accelerated gradient descent (Nesterov momentum method),
  - Minimizing a quadratic function,
  - Solving linear regression using gradient descent.

# Lecture Plan for Week 4&5

- *Nonconvex optimization : supervised learning of feed-forward Neural Networks*
  - Difference in Convex/Nonconvex optimization,
  - Classical Blum/Rivest proof that training a 3-node NN is NP-Complete,
  - Perceptrons
  - Single hidden layered networks,
  - Linear, ReLU , tangential networks,
  - Mean squared error loss, cross entropy loss,
  - *Multiple hidden layered feed forward networks,
  - *Newton's method (optional).

# Extra stuff for research oriented students

* Supplementary material for <u>research oriented students</u>.

# What’s the current Deep Neural Network hype about?

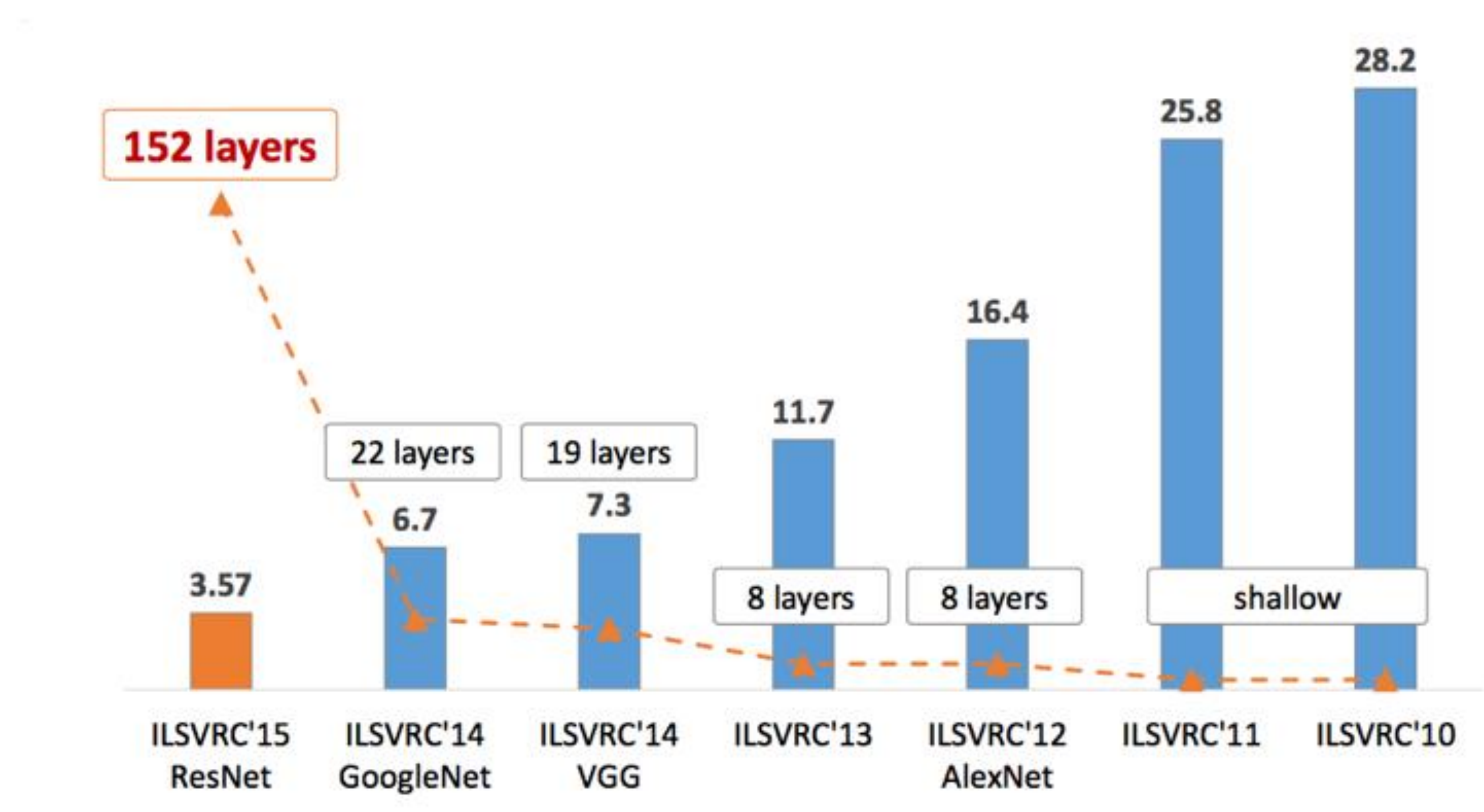
**<u>SUPERHUMAN</u> PERFORMANCE OF DEEP NEURAL NETWORKS ON SOME <u>SPECIALIZED TASKS.</u>**

1. **BRIEF RECAP OF SUCCESSFUL APPLICATIONS OF NEURAL NETWORKS.**
2. **DANGERS OF USING NEURAL NETWORKS AS BLACK BOX.**

# Stories of Success - IMAGENET Challenge

IMAGENET **Large Scale Visual Recognition Challenge**

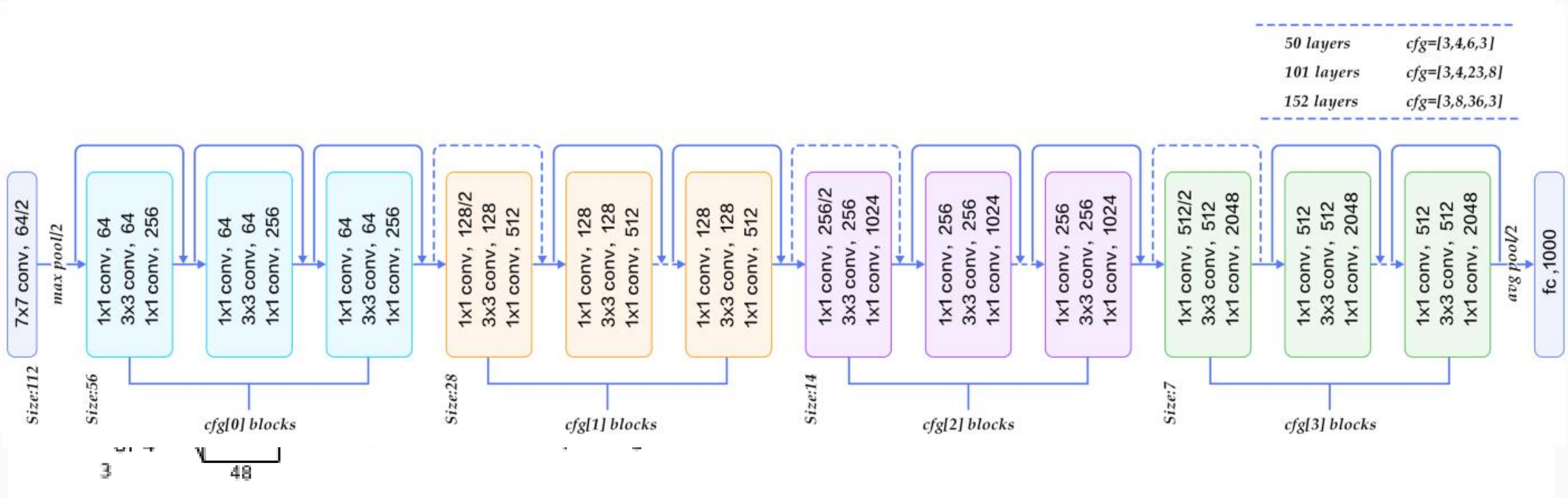
1.2 million training images, 1000 classes




Source: https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

# Stories of Success - *Deep Convolutional Neural Networks* applied to IMAGENET Challenge II

AlexNet    LeNet-5    ResNet




**applied to document recognition**, Proc. IEEE 86(11): 2278–2324, 1998

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition**. CVPR 2016

A. Krizhevsky, I. Sutskever, and G. Hinton, **ImageNet Classification with Deep Convolutional Neural Networks**, NIPS 2012.

# Stories of Success - *Deep Neural Networks for game playing*

AlphaGo: MCTS + supervised learning + reinforcement learning

AlphaGo Zero: MCTS + reinforcement learning

ARTICLE

doi:10.1038/nature24270

## Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

David Silver1, Julian Schrittwieser1, Karen Simonyan1, Ioannis Antonoglou1, Aja Huang1, Arthur Guez1, Thomas Hubert1, Lucas Baker1, Matthew Lai1, Adrian Bolton1, Yutian Chen1, Timothy Lillicrap1, Fan Hui1, Laurent Sifre1, George van den Driessche1, Thore Graepel1 & Demis Hassabis1, **Mastering the game of Go without human knowledge**, NATURE | VOL 550 | 19 october 2017.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis, **Mastering the game of Go with deep neural networks and tree search**, NATURE | VOL 529 | 28 JANUARY 2016.

# Stories of Success - *Deep Neural Networks for game playing II*

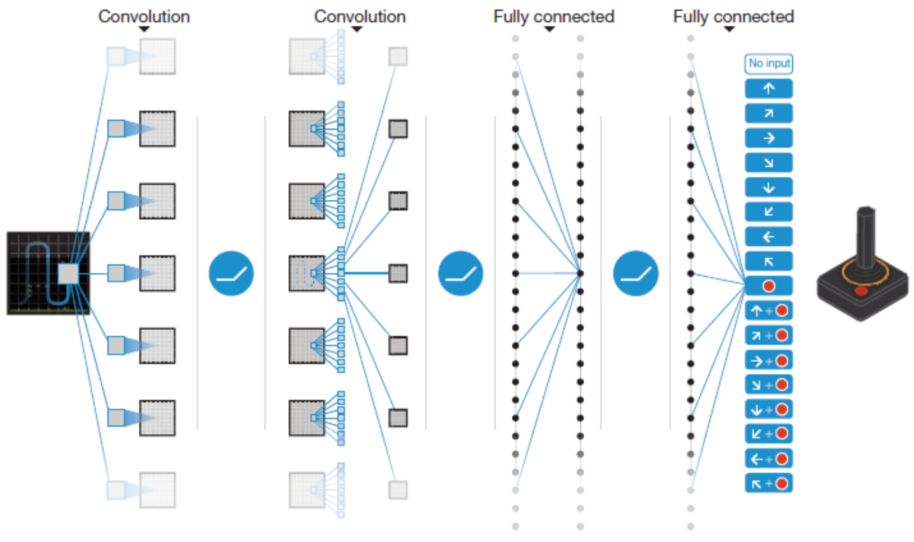
LETTER

doi:10.1038/nature14236

## Human-level control through deep reinforcement learning

Volodymyr Mnih[1]*, Koray Kavukcuoglu[1]*, David Silver[1]*, Andrei A. Rusu[1], Joel Veness[1], Marc G. Bellemare[1], Alex Graves[1], Martin Riedmiller[1], Andreas K. Fidjeland[1], Georg Ostrovski[1], Stig Petersen[1], Charles Beattie[1], Amir Sadik[1], Ioannis Antonoglou[1], Helen King[1], Dharshan Kumaran[1], Daan Wierstra[1], Shane Legg[1] & Demis Hassabis[1]

Training a Deep Neural Network to play Atari games using reinforcement learning.



# Dangers

Hoooorayy

Deep Learning can solve any problem!!

Maybe not yet?

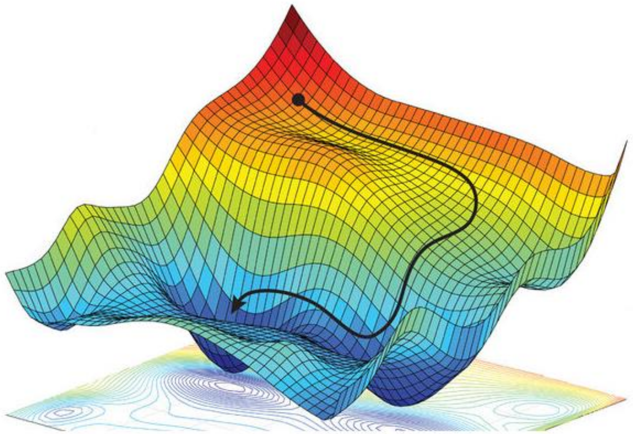
1. Available technologies (e.g. TensorFlow) provide NN (DNN) tools as a **black box**.
2. Current architectures of NN (DNN) are **too easy to fool**.
3. Lack of **interpretability**.

# Problems with deep learning

SHARE

13K

1K



Gradient descent relies on trial and error to optimize an algorithm, aiming for minima in a 3D landscape.
ALEXANDER AMINI, DANIELA RUS, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, ADAPTED BY M. ATAROD/*SCIENCE*

## AI researchers allege that machine learning is alchemy

By **Matthew Hutson** | May. 3, 2018 , 11:15 AM

Ali Rahimi, a researcher in artificial intelligence (AI) at Google in San Francisco, California, took a swipe at his field last December—and received a 40-second ovation for it. Speaking at an AI conference, Rahimi charged that machine learning algorithms, in which computers learn through trial and error, **have become a form of "alchemy."** Researchers, he said, do not know why some algorithms work and others don't, nor do they have rigorous criteria for choosing one AI architecture over another. Now, in a paper presented on 30 April at the International Conference on

**Medium**

Michael Jordan Follow
Michael I. Jordan is a Professor in the Department of Electrical Engineering and Computer Sciences and the Department of Statistics at UC Berkeley.
Apr 18 · 16 min read

Listen to this story
0:00 22:31

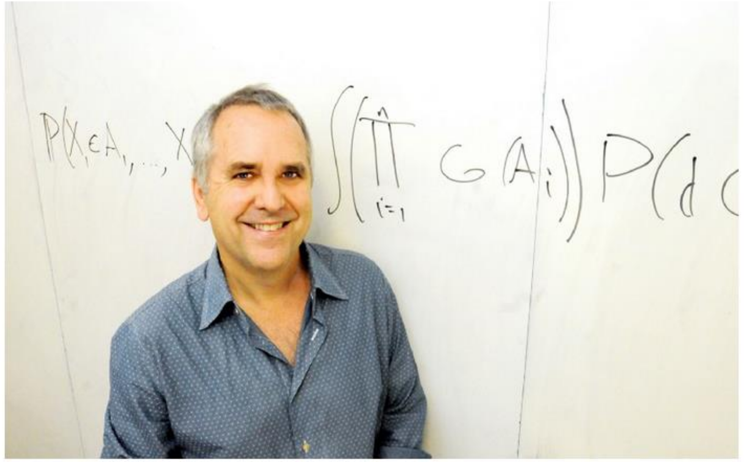


**Photo credit:** Peg Skorpinski

## Artificial Intelligence—The Revolution Hasn't Happened Yet

## The Machine Learning Bubble?

JANUARY 26, 2017 | MICHAEL MOZER

Written by: Michael Mozer, Scientific Advisor at AnswerOn and Professor at the University of Colorado

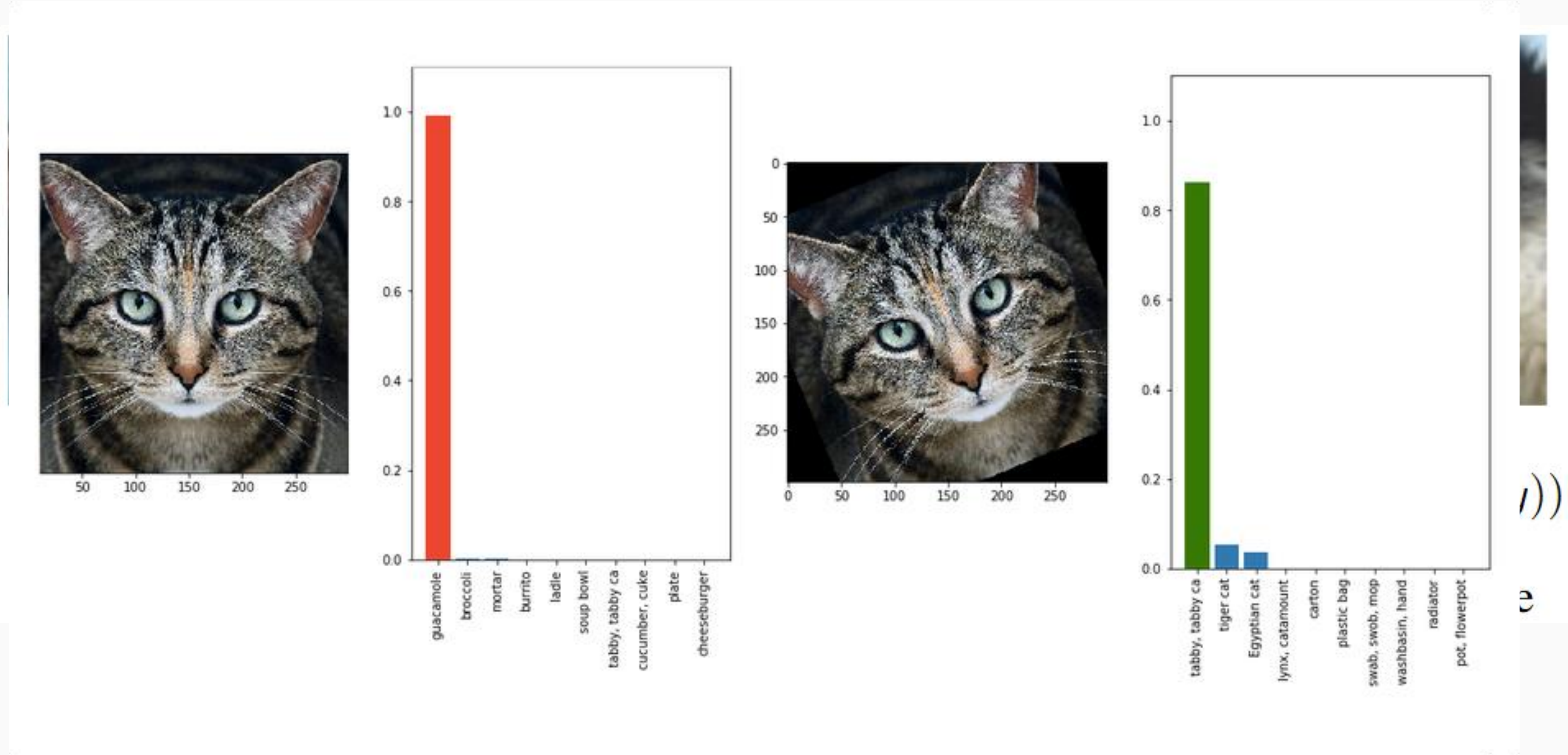
Five years ago, few had heard of the field of machine learning. Now, articles appear daily describing the accomplishments of machine learning, companies market their prowess in machine learning, and machine learning has become nearly synonymous with artificial intelligence. Here is a billboard I drove past on highway 80 in the East Bay (heading toward Silicon Valley), which was nearly as shocking as a billboard with my own name would have been:

In an earlier posting, I explained some of the reasons for the resurgence of AI and the interest in machine learning. Since then, major companies have invested even more in the technology, venture-capital backed start ups abound, and press releases and technical papers are regularly issued on astonishing results, such as AlphaGo, the system developed at DeepMind which plays the game of Go. (Go has been one of

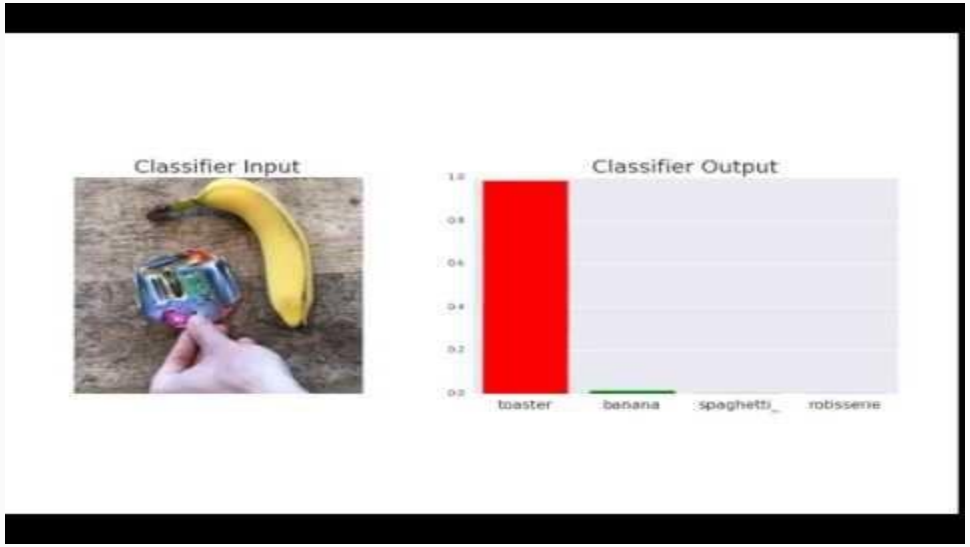
Sources:
http://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy
https://medium.com/@mijordan3/artificial-intelligence-the-revolution-hasnt-happened-yet-5e1d5812e1e7
http://answeron.com/machine-learning-bubble/

# Very big problem

Lack of robustness -- existence of adversarial examples



# ‘Adversarial patch’



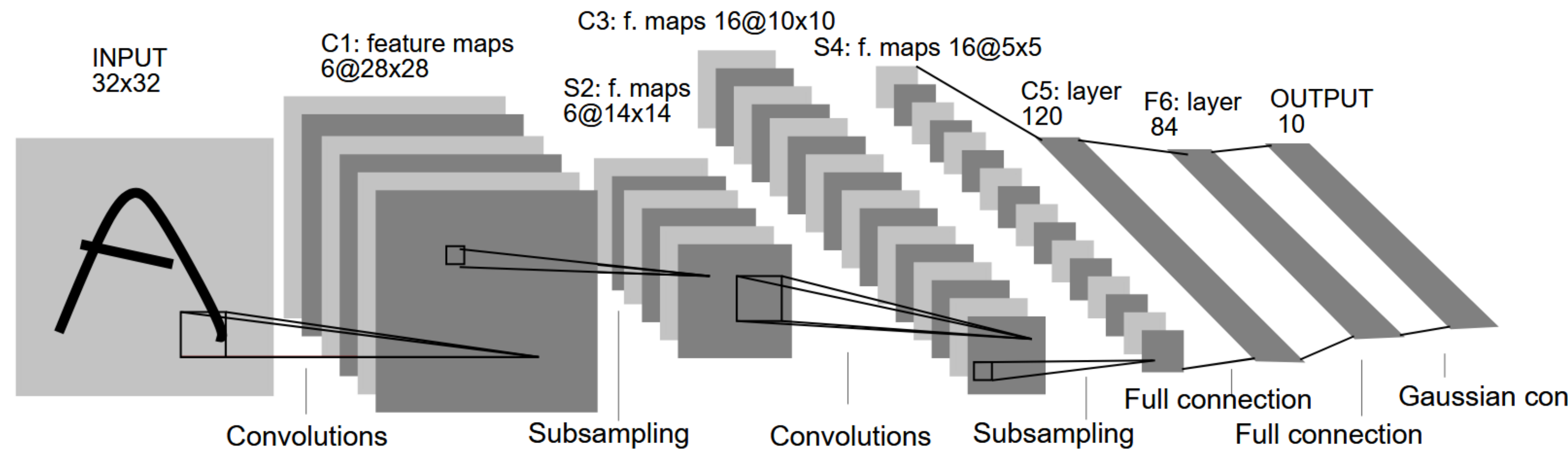
# Motivation of looking into the black box

2. black box – a trained neural network

1. black box – a function

1 2 3

$$x^2$$




**GO BACK TO PRINCIPLES**
**MATH, CALCULUS, LINEAR ALGEBRA**

# Look into the process of training of the 'black box'

| Training set $X$ | Labels $Y$ |
|---|---|
|  | 0 |
|  | 1 |
|  | 2 |
|  | 3 |
|  | 4 |
|  | 5 |
|  | 6 |
|  | 7 |
|  | 8 |
|  | 9 |

Neural Network
=
trainable function having a lot of parameters

Neural Network given input image $X_i$ computes $f_W(X_i)$

$X, Y$ can be thought as <u>matrices</u> of inputs and desired labels, and $W$ can be thought as <u>matrix of weights</u>.

The loss function

$$L(W, X, Y) = \sum_{i=1}^{N} \|f_W(X_i) - Y_i\|^2$$

We want to find

$$W^\star = \mathrm{argmin}_W \{L(W, X, Y)\}$$

using <u>optimization</u>.

# Introductory Week 1&2

We like to use a good modern language **Python** is the choice today.

- Modern language,
- High-level,
- Easy to use,
- Availability of packages and tutorials,
- Critical parts are optimized in low level C++,

# Week 1 & 2

Linear algebra in NumPy

Scientific computing library

# A lot of linear algebra in deep learning

Deep Learning

Ian Goodfellow
Yoshua Bengio
Aaron Courville

# Vectors and matrices

$$v = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n.$$

$$M = \begin{bmatrix} m_{11} & m_{12} & \ldots & m_{1n} \\ m_{21} & m_{22} & \ldots & m_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ \boxed{m_{n1}} & m_{n2} & \ldots & m_{nn} \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

each entry
pixel value

(actually three matrices $M_1, M_2, M_3$ the three components of RGB )

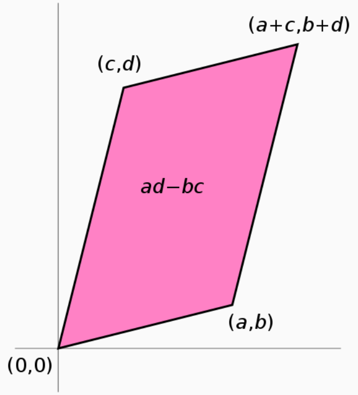
Show vectors and matrices in NumPy

week1.ipynb

# Interpretation of the matrix determinant

The Leibniz formula for the determinant of a 2 × 2 matrix is

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

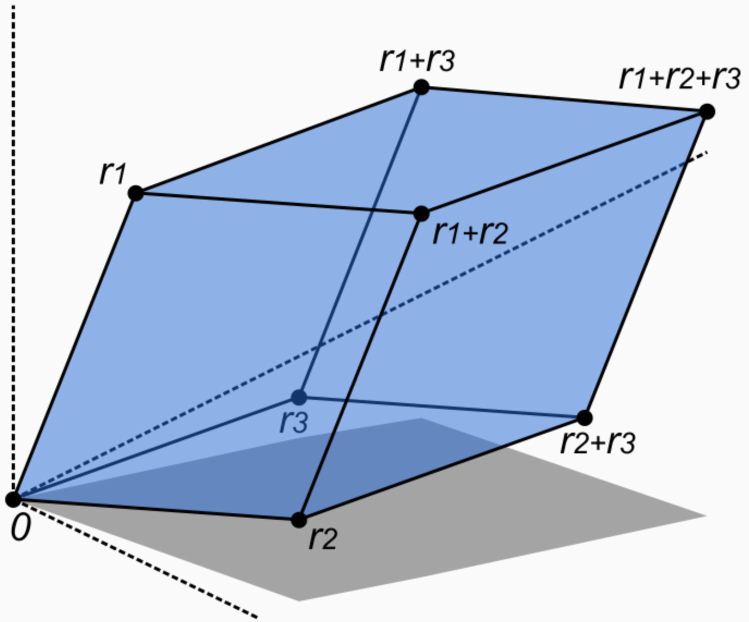



The Laplace formula for the determinant of a 3 × 3 matrix is

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

this can be expanded out to give

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a(ei - fh) - b(di - fg) + c(dh - eg)$$
$$= aei + bfg + cdh - ceg - bdi - afh.$$




Source: wikipedia.org

Show determinants in NumPy

week1.ipynb

# Solving linear systems of equations

$$Ax = b,$$

where $A$ is a matrix, $x$ is the unknown vector, $b$ is a fixed vector of values.

We have either

- one solution,
- none solutions,
- infinitely many solutions.

Many methods of solving:

- Inverting A,
- Gaussian elimination,
- LU factorization,
- Cramer's rule,
- Etc..

# Cramer's rule for systems of linear equations

## Cramer's Rule

In a nonsingular system $\mathbf{A}_{n\times n}\mathbf{x} = \mathbf{b}$, the $i^{th}$ unknown is

$$x_i = \frac{\det(\mathbf{A}_i)}{\det(\mathbf{A})},$$

where $\mathbf{A}_i = \left[\mathbf{A}_{*1}\middle|\cdots\middle|\mathbf{A}_{*i-1}\middle|\mathbf{b}\middle|\mathbf{A}_{*i+1}\middle|\cdots\middle|\mathbf{A}_{*n}\right]$. That is, $\mathbf{A}_i$ is identical to $\mathbf{A}$ except that column $\mathbf{A}_{*i}$ has been replaced by $\mathbf{b}$.

# Linear algebra for machine learning

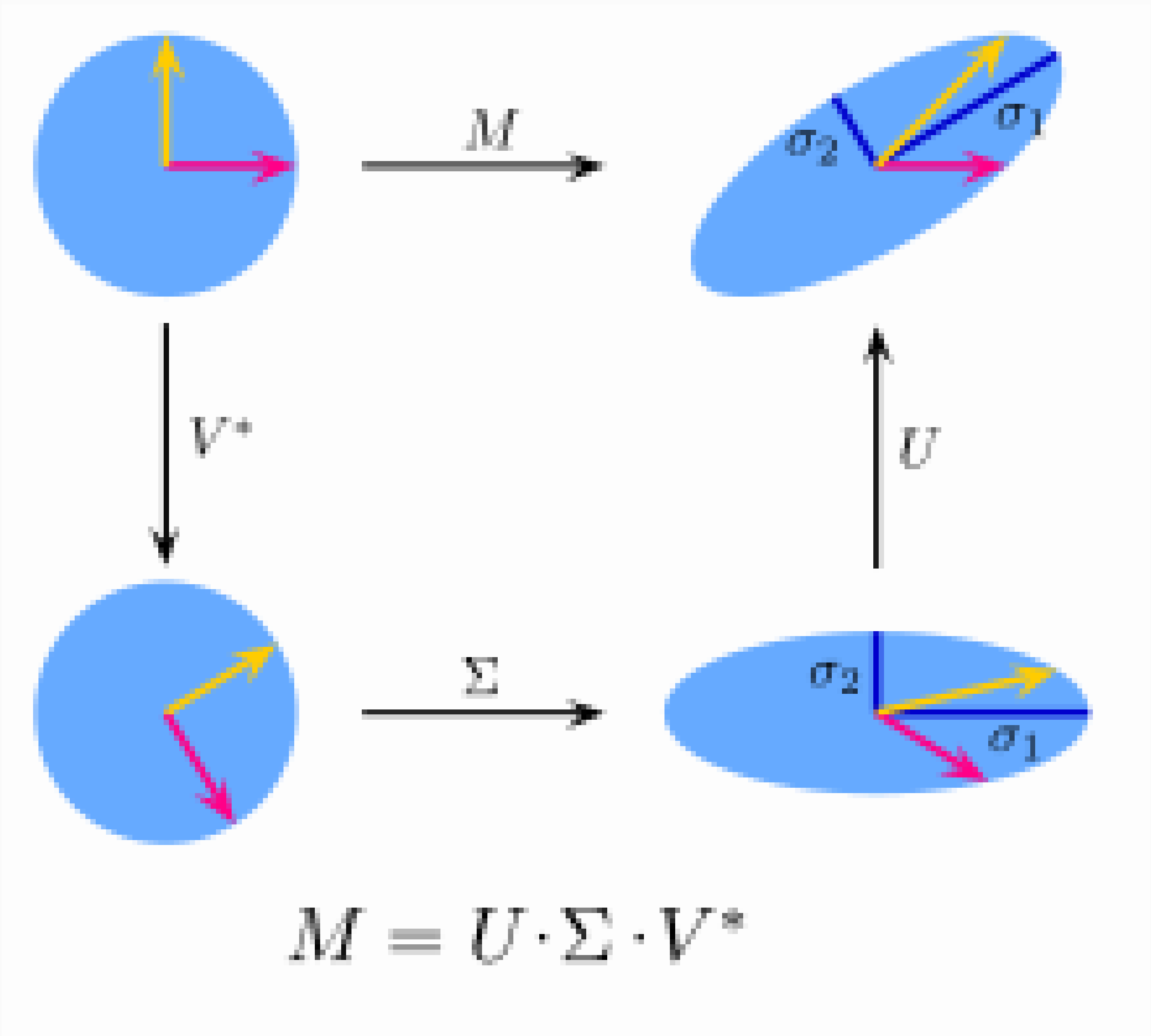
Some linear algebraic techniques that are very important for machine learning

- The SVD decomposition,
- Page-rank algorithm,
- Linear regression of data,

# Super important for ML is SVD decomposition

Source: wikipedia.org

# Interpretation of the SVD decomposition II




Source: wikipedia.org

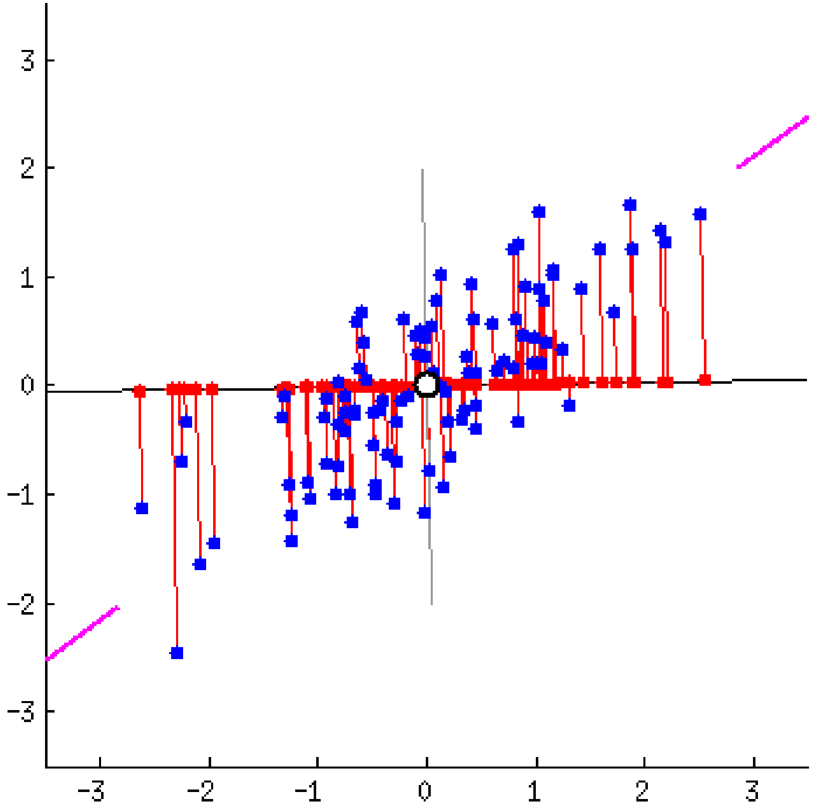
# Reminder – rotation matrix

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Hence those example SVD matrices U and V represent rotation, by what angle ?

```
U=[[ 0.22975292 -0.97324899]
 [ 0.97324899  0.22975292]]
V=[[ 0.52573111 -0.85065081]
 [ 0.85065081  0.52573111]]
```

# Application of SVD – dimensionality reduction



# Covariance

Let me start with PCA. Suppose that you have n data points comprised of d numbers (or dimensions) each. If you center this data (subtract the mean data point $\mu$ from each data vector $x_i$) you can stack the data to make a matrix

$$X = \left( \begin{array}{c} x_1^T - \mu^T \\ \hline x_2^T - \mu^T \\ \hline \vdots \\ \hline x_n^T - \mu^T \end{array} \right).$$

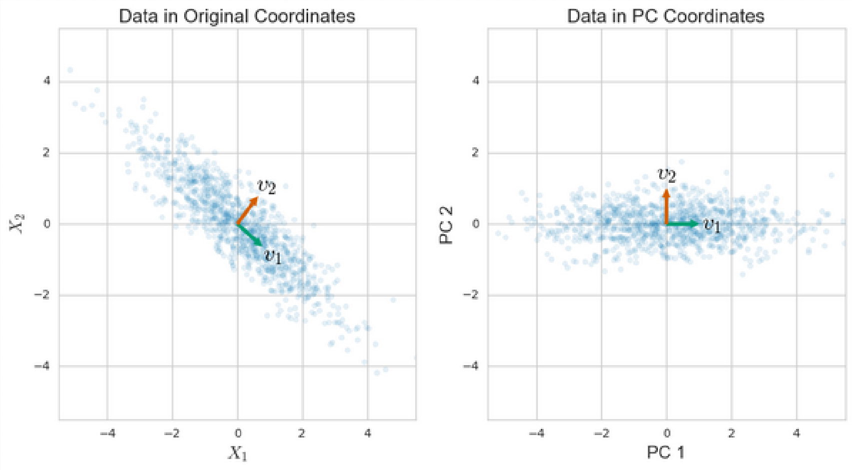
The covariance matrix

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T = \frac{1}{n-1} X^T X$$

measures to which degree the different coordinates in which your data is given vary together. So, it's maybe not surprising that PCA -- which is designed to capture the variation of your data -- can be given in terms of the covariance matrix. In particular, the eigenvalue decomposition of $S$ turns out to be

$$S = V \Lambda V^T = \sum_{i=1}^{r} \lambda_i v_i v_i^T ,$$

where $v_i$ is the $i$-th *Principal Component*, or PC, and $\lambda_i$ is the $i$-th eigenvalue of $S$ and is also equal to the variance of the data along the $i$-th PC. This decomposition comes from a general theorem in linear algebra, and some work *does* have to be done to motivate the relatino to PCA.




Source: https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca

# Application of SVD – dimensionality reduction II

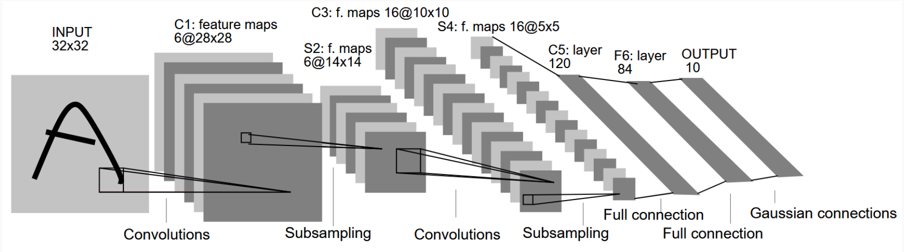
data matrix $X \in \mathbb{R}^{n\times p}$,
covariance matrix $C = X^T X/(n-1) \in \mathbb{R}^{p\times p}$

$$\text{SVD decomposition } C = VLV^T,$$

(observe that $C$ is a symmetric positive definite matrix
$V$ is a matrix of eigenvectors, $L$ is a diagonal,
Eigenvectors are principal axes of data.

# Reminder about the goal of the course

| Training set $X$ | Vectorized $X_i$ |
|---|---|
|  | [0,0,1,0,1,… |
|  | [0,0,1,1,1,… |
|  | [0,1,1,0,1,… |
|  | [1,1,1,0,1,… |
|  | [0,0,0,0,1,… |
|  | Etc.. |
|  | |
|  | |
|  | |
|  | |




=

Trainable very large function having a lot of parameters

Neural Network given input vector $X_i$ computes $f_W(X_i)$

$X, Y$ can be thought as <u>matrices</u> of inputs and desired labels, and $W$ can be thought as a <u>matrix of weights</u>.

The loss function

$$L(W, X, Y) = \sum_{i=1}^{N} \|f_W(X_i) - Y_i\|^2$$

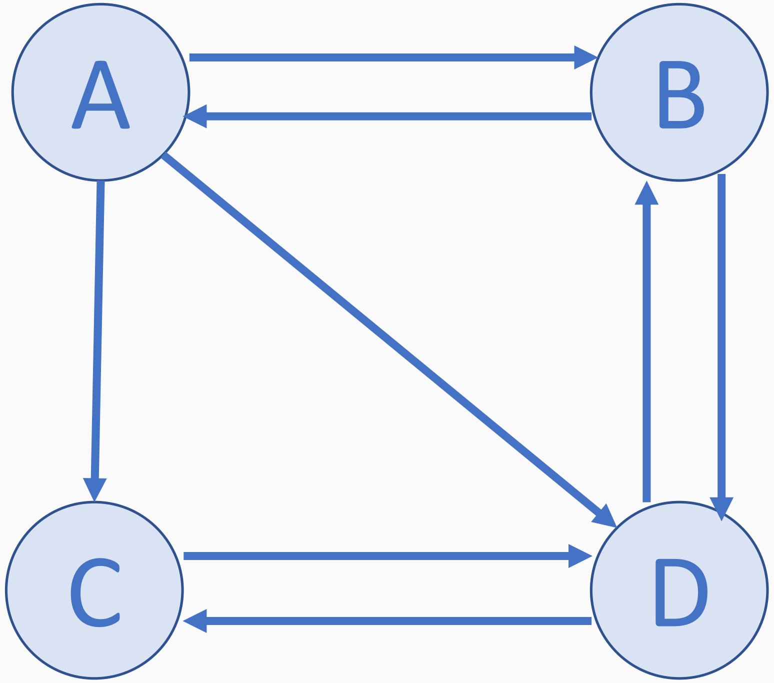
We want to find

$$W^\star = \mathrm{argmin}_W \{L(W, X, Y)\}$$

using <u>optimization</u>.

# Eigenvalues and eigenvectors and Introduction to the Page Rank algorithm

Given an abstract web of documents, linking to each other:




$$L_A = (0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}),$$
$$L_B = (\frac{1}{2}, 0, 0, \frac{1}{2}),$$
$$L_C = (0, 0, 0, 1),$$
$$L_D = (0, \frac{1}{2}, \frac{1}{2}, 0).$$

$$L = \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 1 & 0 \end{bmatrix}$$

Rank of the page $A$ $r_A = \sum_{j=1}^{n} L_{A,j} r_j$

The goal is to find the ranks of all pages in the web ($r$)

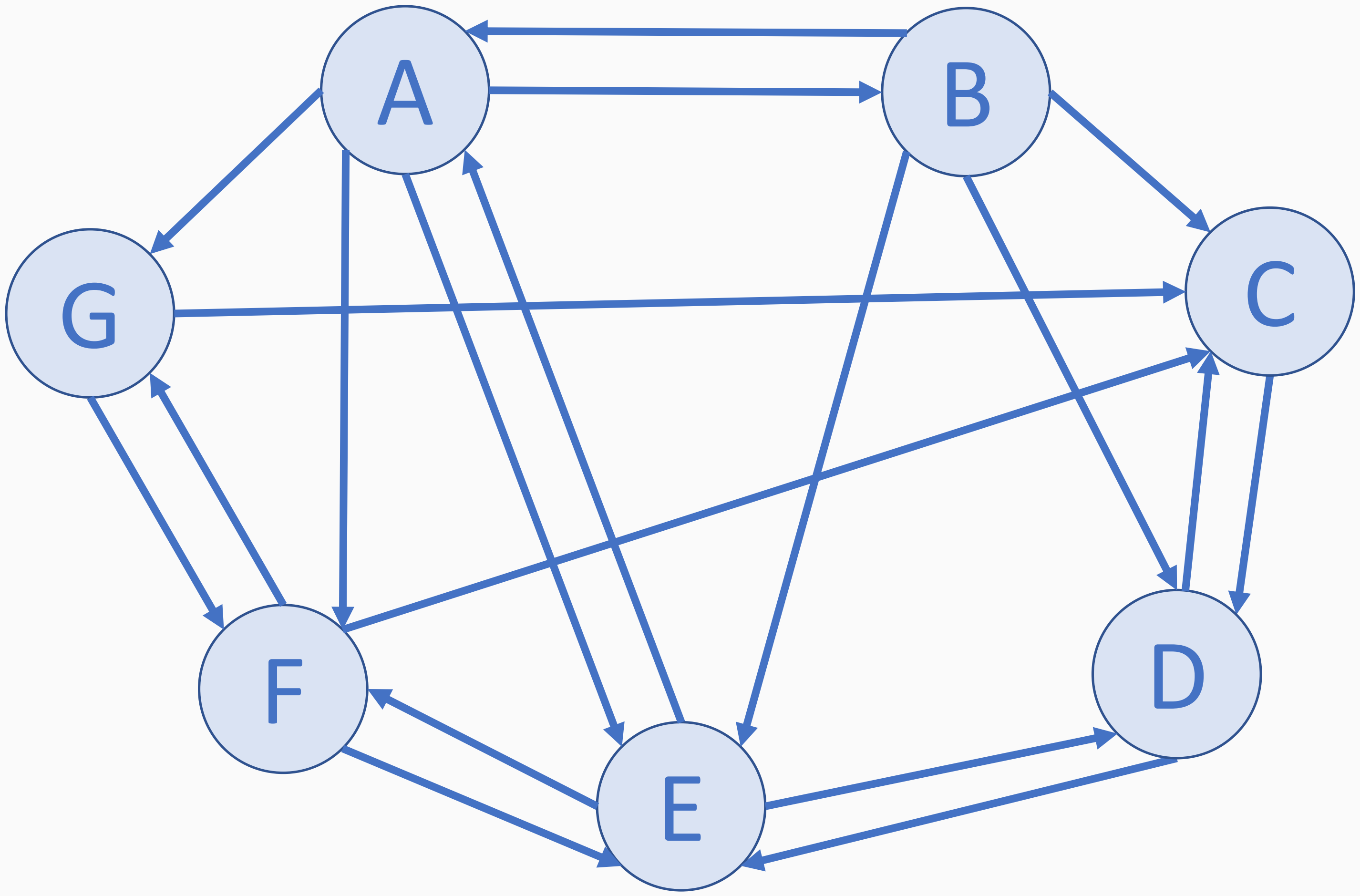
Solve iteratively $r_{k+1} = L r_k$.

Improvement: regularization

$$r_{k+1} = d(L r_k) + \frac{1-d}{n}$$

# Labwork task

Implement the page-rank algorithm for more complicated web of documents



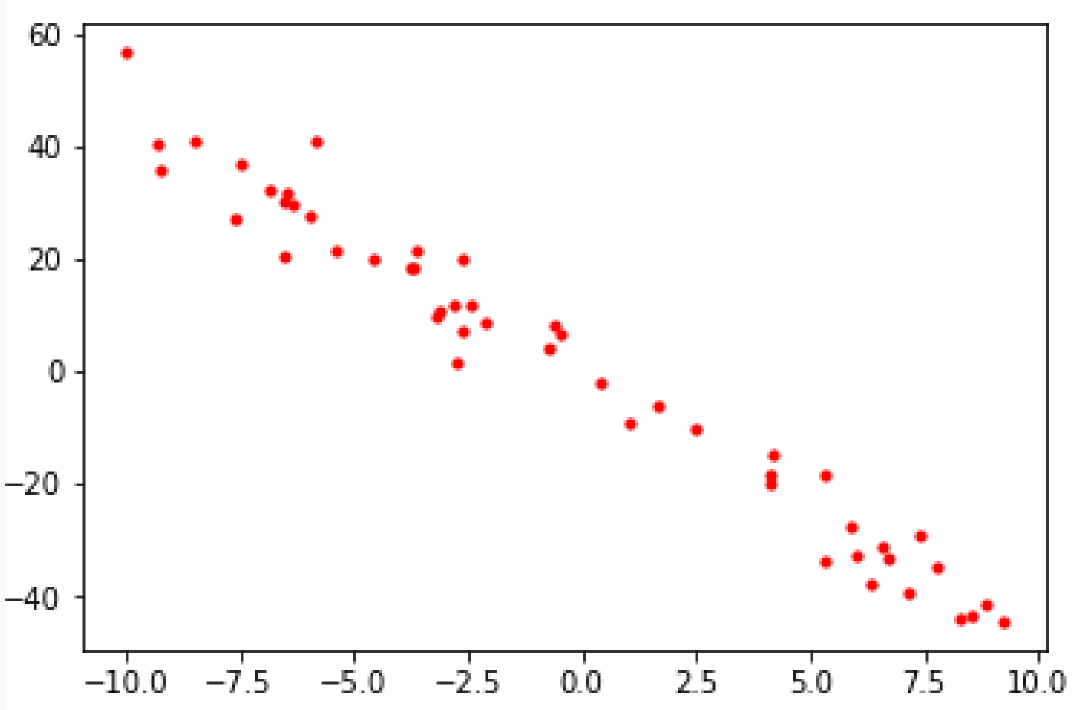

Find $r = ???$

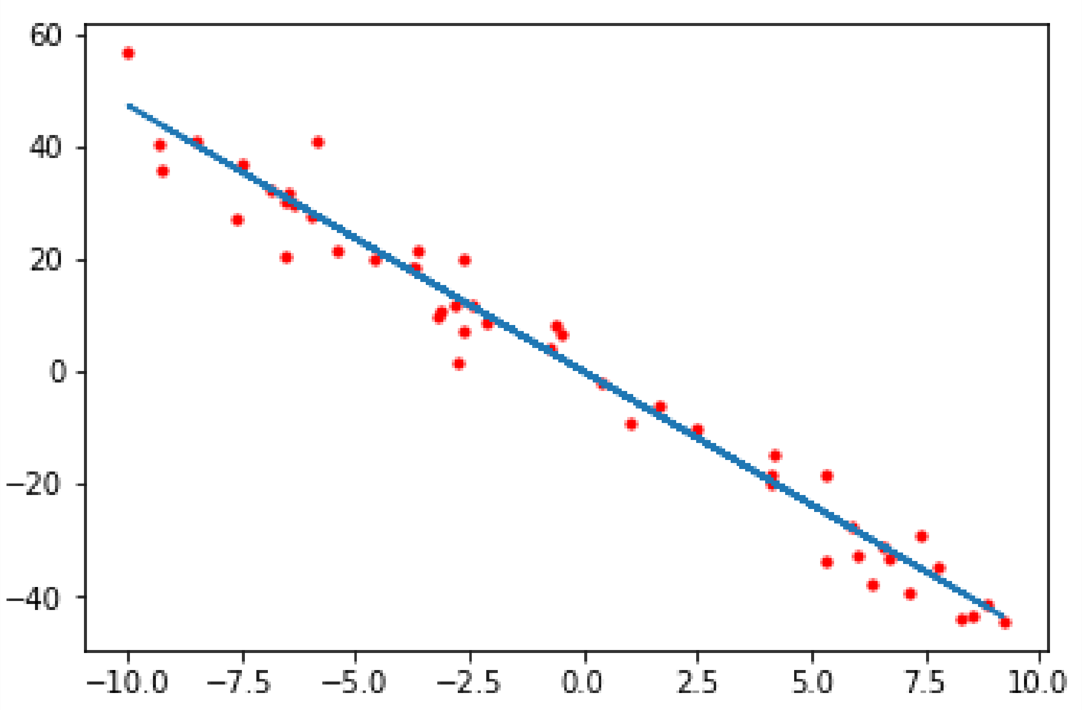
Page-rank in fact converges to the largest eigenvector of $L$ matrix.

It's a scalable way for computing the eigenvector directly.

# Linear Regression

Statement of the *linear regression problem*:
given data , find a <u>linear function</u> which is the best approximator of the data.

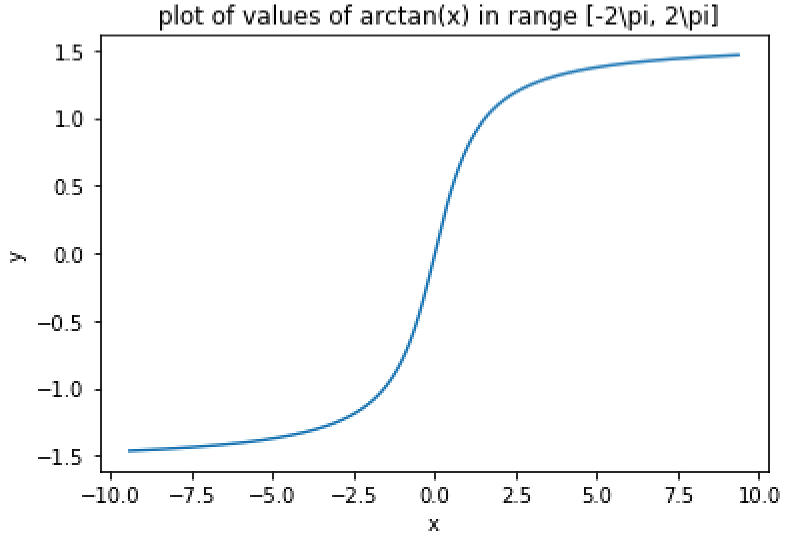


# Functions/data plotting using matplotlib

```
import numpy as np
import matplotlib.pyplot as pyp
import math
```

```
x = np.array(range(-150,150))*2*math.pi/100
y = np.arctan(x)
pyp.plot(x,y)
pyp.xlabel('x')
pyp.ylabel('y')
pyp.title(' plot of values of arctan(x) in range [-2\pi, 2\pi]')
pyp.show()
```



# Derivatives 1 (univariate)

## What is the derivative ?

Measures the rate of change of a function
(negative when function is decreasing
and positive when function is increasing)

Mathematica 1d gradient presentation

# Numerical vs symbolic derivatives

Derivative of function $f$

Derivative symbol

at $x$

$$f'(x) = \frac{df}{dx}(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

with respect to $x$

Limit as $h$ approaches 0

`Numerical derivative`

Estimate of the rate change of $f$ for fixed (small) $h$

$$\frac{f(x+h) - f(x)}{h}$$

# Numerical vs symbolic derivatives

*Computation of derivatives symbolically using the rules of differentiation*

- $\frac{d}{d(x)}(a) = 0$
- $\frac{d}{d(x)}(x) = 1$
- $\frac{d}{d(x)}(x^n) = n(x)^{n-1}$
- $\frac{d}{d(x)}[f(x) \pm g(x)] = f'(x) \pm g'(x)$
- $\frac{d}{d(x)}[c f(x)] = c f'(x)$
- $\frac{d}{d(x)}[f(x) \cdot g(x)] = f'(x) g(x) + g'(x) f(x)$ Product rule
- $\frac{d}{d(x)}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x) g(x) - g'(x) f(x)}{[g(x)]^2}$ Quotient rule
- $\frac{d}{d(x)} f[g(x)] = f'[g(x)] g'(x)$ Chain rule
- $\frac{d}{d(x)} f(x)^n = n f(x)^{n-1} \cdot f'(x)$ Power rule
- $\frac{d}{d(x)} f(kx + e) = k f'(kx + e)$
- $\frac{d}{d(x)} \ln[f(x)] = \frac{f'(x)}{f(x)}$

smtutor.com

$[x^a]' = a \cdot x^{a-1}$, $x \in \mathbb{R}$ for $a \in \mathbb{N}$, $x \in \mathbb{R} - \{0\}$ for $a \in \mathbb{Z}$, $x \in \mathbb{R}^+$ for $a \in \mathbb{R}$.

$[e^x]' = e^x$, $x \in \mathbb{R}$;

$[a^x]' = \ln(a) a^x$, $x \in \mathbb{R}$.

$[\ln(x)]' = \frac{1}{x}$, $x > 0$;

$[\log_a(x)]' = \frac{1}{\ln(a)} \frac{1}{x}$, $x > 0$.

$[\sin(x)]' = \cos(x)$, $x \in \mathbb{R}$;

$[\cos(x)]' = -\sin(x)$, $x \in \mathbb{R}$;

$[\tan(x)]' = \frac{1}{\cos^2(x)}$, $x \neq \frac{\pi}{2} + k\pi$;

$[\cot(x)]' = \frac{-1}{\sin^2(x)}$, $x \neq k\pi$.

$[\arcsin(x)]' = \frac{1}{\sqrt{1-x^2}}$, $x \in (-1, 1)$;

$[\arccos(x)]' = \frac{-1}{\sqrt{1-x^2}}$, $x \in (-1, 1)$;

$[\arctan(x)]' = \frac{1}{x^2+1}$, $x \in \mathbb{R}$;

$[\mathrm{arccot}(x)]' = \frac{-1}{x^2+1}$, $x \in \mathbb{R}$.

$[\sinh(x)]' = \cosh(x)$, $x \in \mathbb{R}$;

$[\cosh(x)]' = \sinh(x)$, $x \in \mathbb{R}$;

$[\tanh(x)]' = \frac{1}{\cosh^2(x)}$, $x \in \mathbb{R}$;

$[\coth(x)]' = \frac{-1}{\sinh^2(x)}$, $x \neq 0$.

$[\mathrm{argsinh}(x)]' = \frac{1}{\sqrt{x^2+1}}$, $x \in \mathbb{R}$;

$[\mathrm{argcosh}(x)]' = \frac{1}{\sqrt{x^2-1}}$, $x \in (1, \infty)$;

$[\mathrm{argtanh}(x)]' = \frac{1}{1-x^2}$, $x \in (-1, 1)$;

$[\mathrm{argcoth}(x)]' = \frac{1}{1-x^2}$, $x \in (-\infty, -1) \cup (1, \infty)$.

See the comparison of numerical and symbolic derivatives in week1_2.ipynb

# Chain rule

The rule for differentiating compositions of functions

inner function

$$\text{If } f = g(h) \text{ and } h = h(x), \text{ then } \frac{df}{dx} = \frac{dg}{dh} \times \frac{dh}{dx}$$

$f$ is a composed function

outer function

Derivative of the outer function

Derivative of the inner function

# Example for the chain rule in practice

$$f(x) = \arctan x^3$$

1. Decomposition of $f$ into the outer ($g$) / inner ($h$) functions.

$$f(x) = g(h(x))$$

| |
|---|
| $g(h) = \arctan h$ |
| $h(x) = x^3$ |

2. Differentiate $g$ and $h$.

$$\frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\frac{d}{dx}x^3 = 3x^2$$

| | |
|---|---|
| $g(h) = \arctan h$ | $g'(h) = \frac{1}{1+h^2}$ |
| $h(x) = x^3$ | $h'(x) = 3x^2$ |

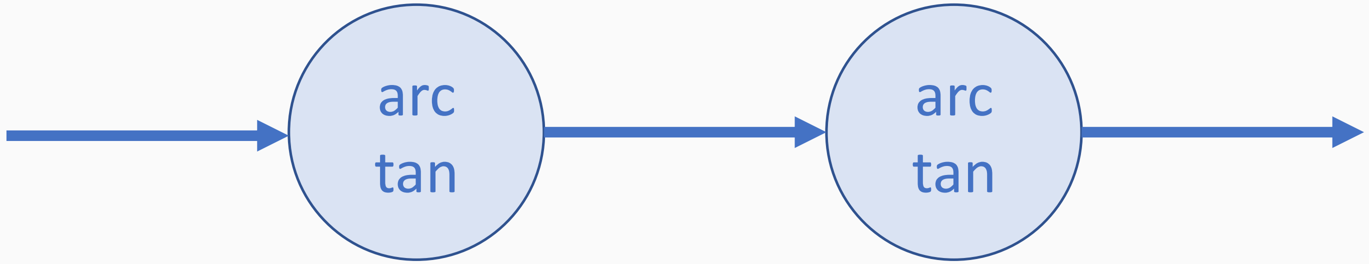
3. Compose the final result

$$\frac{df}{dx} = \frac{dg}{dh} \times \frac{dh}{dx} = \frac{1}{1+h^2} \cdot 3x^2 = \frac{1}{1+x^6} \cdot 3x^2.$$

See the comparison of this derivative with symbolic in week1_2.ipynb

# Another example

Two perceptrons linked together with one input and output



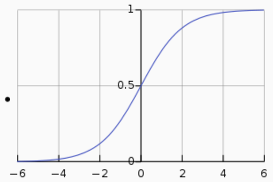

$$f(x) = \arctan(\arctan x)$$

<u>Lab assignment:</u> Compute $f'(x)$ by applying the chain rule, and then compare graph with the numerical derivative.

Some other functions for practicing the chain rule

$$(1 + x + 2x^2)^5, \ \sin x^3 + 1, \ \sqrt{x^4 + 1}, \ \ln t^2, \ \ln 1 + \frac{1}{t}$$

*The sigmoid function $\frac{1}{1+e^{-z}}$ (more advanced).



# Partial derivatives and gradients

Partial derivative with respect to $x$

$$\frac{\partial f}{\partial x}(x, y)$$

When computing, treat $y$ as a constant

Partial derivative with respect to $y$

$$\frac{\partial f}{\partial y}(x, y)$$

When computing, treat $x$ as a constant

gradient $$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

[Mathematica 2d gradient presentation]

# Computing partial derivatives analytically

$$f(x, y) = \sin xy$$

Compute $\frac{\partial f}{\partial x}$ applying the univariate chain rule (treating $y$ as constant)
we have $f(x,y) = g(h), \quad h = h(x)$

| | |
|---|---|
| $g(h) = \sin h$ | $g'(h) = \cos h$ |
| $h(x) = xy$ | $h'(x) = y$ |

$$\frac{\partial f}{\partial x} = \frac{dg}{dh} \times \frac{dh}{dx} = \cos(xy) \cdot y.$$

And analogously

$$\frac{\partial f}{\partial y} = \frac{dg}{dh} \times \frac{dh}{dy} = \cos(xy) \cdot x.$$

# Introduction to the backprop algorithm

The <u>fundamental algorithm</u> for the training of neural nets.

It boils down to executing the *chain rule in reverse* on a *DAG (Directed Acyclic Graphs)*.

Backpropagation is a **rediscovery**.
It has been known under the name *automatic differentiation* in numerical analysis.

**Learning representations by back-propagating errors**

**David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams***

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

**We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].**

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, $x_j$, to unit $j$ is a linear function of the outputs, $y_i$, of the units that are connected to $j$ and of the weights, $w_{ji}$, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, $y_j$, which is a non-linear function of its total input

## Automatic differentiation

From Wikipedia, the free encyclopedia

In mathematics and computer algebra, **automatic differentiation** (**AD**), also called **algorithmic differentiation** or **computational differentiation**,[1][2] is a set of techniques to numerically evaluate the derivative of a function specified by a computer program. AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, derivatives of arbitrary order can be computed automatically, accurately to working precision, and using at most a small constant factor more arithmetic operations than the original program.

Automatic differentiation is not:

- Symbolic differentiation, nor
- Numerical differentiation (the method of finite differences).

These classical methods run into problems: symbolic differentiation leads to inefficient code (unless done carefully) and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. Both classical methods have problems with calculating higher derivatives, where the complexity and errors increase. Finally, both classical methods are slow at computing the partial derivatives of a function with respect to *many* inputs, as is needed for gradient-based optimization algorithms. Automatic differentiation solves all of these problems, at the expense of introducing more software dependencies[*citation needed*].

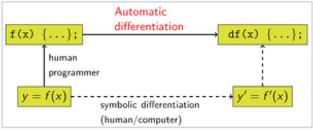
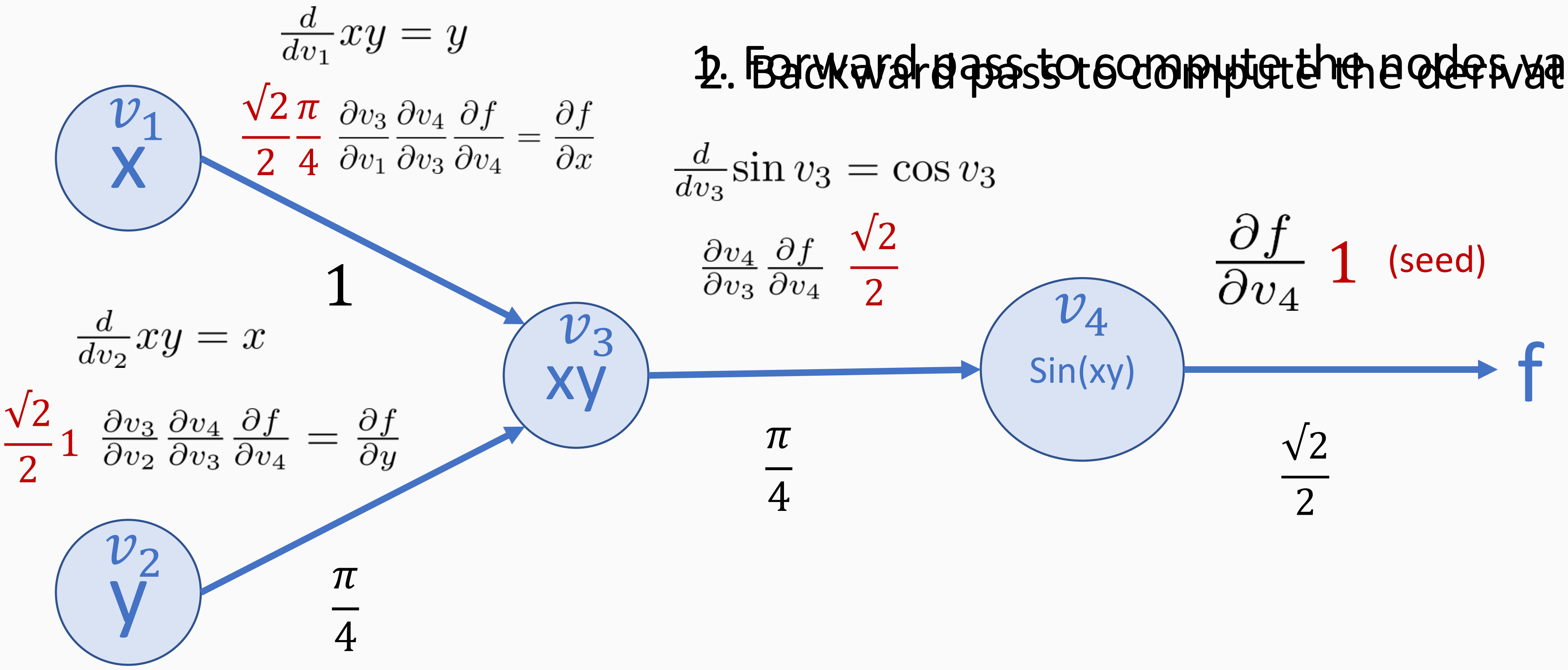



Figure 1: How automatic differentiation relates to symbolic differentiation

Contents [hide]

# Introduction to the backprop algorithm

1. Forward pass to compute the nodes values.
2. Backward pass to compute the derivatives.



# Computing 2D gradients numerically

Estimate the rate of change of $f(x, y)$
for fixed (small) $h$ in $x$ and $y$ direction

$$\frac{f(x+h,y)-f(x,y)}{h}$$

$$\frac{f(x,y+h)-f(x,y)}{h}$$

Those two quantities form a numerical gradient of $f(x, y)$

# Higher order numerical derivatives

Higher accuracy can be obtained by applying
*higher order numerical schemes*
like this five point scheme

$$\frac{-f(x+2h,y)+8f(x+h,y)-8f(x-h,y)+f(x-2h,y)}{12h}$$

$$\frac{-f(x,y+2h)+8f(x,y+h)-8f(x,y-h)+f(x,y-2h)}{12h}$$