Deep Learning CS583

Jia Xu

Stevens Institute of Technology

2021 Fall

about CS583: contacts

lecture time:
9:30-12:00PM Tuesdays or Thursdays
Recitation time: TBD

Office hours:
Instructor: Tuesdays 12:30-14:30
TAs:
Hanish Pallapothu Fridays 14:00-16:00
Randheer Vennapureddy Fridays 13:00-15:00
Om Bidhalan Mondays 11:00-13:00

Email:
Instructor: jxu70@stevens.edu
TAs:
Hanish Pallapothu hpallap1@stevens.edu
Randheer Vennapureddy rvennapu@stevens.edu
Om Bidhalan ombidhalan@gmail.com

about CS583: grading

Grading:

A: excellent empirical ability and deep understanding
B: good empirical ability and good understanding
C: meet basic requirement
D: fail in following up the course content

Grading proportional to
the ranking of your total score in the class

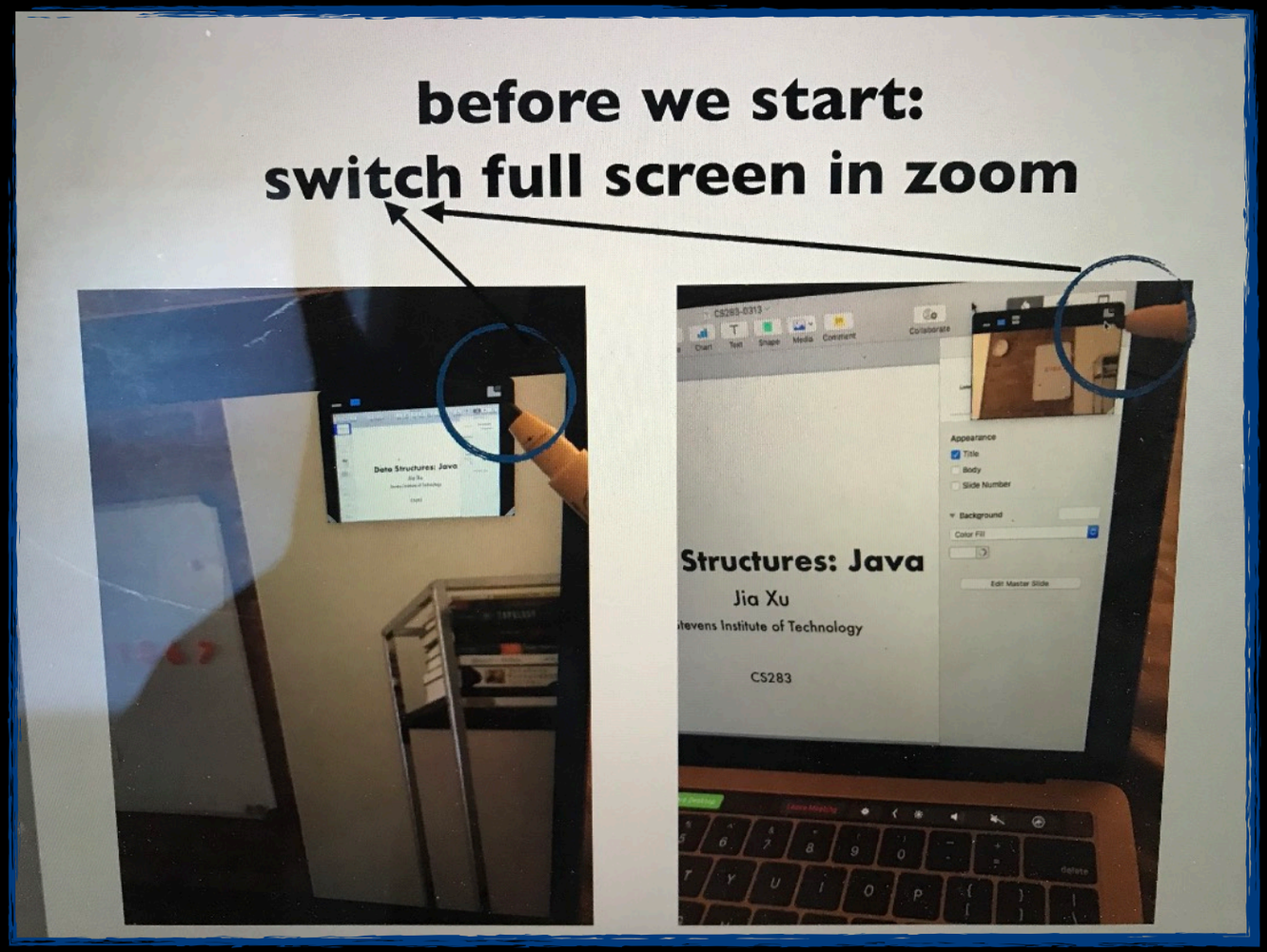
about CS583: how to score

Midterm exam: 10%
Final exam: 10%
Presentation: 15%
Q&A participation: 10%
Quizz (3-5): 15%
Homework assignments (4): 40%

about CS583: in-person and online class

Almost same exams/homework
No distinguish on participation status

Online:



about CS583: textbook

Textbook:

Deep Learning. Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
http://www.deeplearningbook.org

Other Readings:

S. Boyd and L. Vandenberghe. Introduction to Applied Linear Algebra. Cambridge University Press, 2018. (Available online.)

Y. Nesterov. Introductory Lectures on Convex Optimization Book. Springer, 2013. (Available online.)

D. S. Watkins. Fundamentals of Matrix Computations. John Wiley & Sons, 2004.

Francois Chollet. Deep learning with Python. Manning Publications Co., 2017. (Available online.)

M. Mohri, A. Rostamizadeh, and A. Talwalkar. Foundations of machine learning. MIT press, 2012.

J. Friedman, T. Hastie, and R. Tibshirani. The elements of statistical learning. Springer series in statistics, 2001. (Available online.)

about CS583: tentative topics

Lecture slides	Main topics	Sections in Textbook "Deep Learning" by Goodfellow, Bengio, Courville
Week 1	Introduction, ML basics	Section 5
Week 2	Math Background	Section 2-4
Week 3	MLP	Section 6
Week 4	Stochastic Gradient Descent	Section 6
Week 5	Deep Feedforward Networks	Section 6
Week 6	Regularization, Optimization	Section 7, 8
Week 7	Midterm Exam	All above
Week 8	RNN, LSTM	Section 10
Week 9	CNN	Section 9
Week 10	Attention	beyond textbook
Week 11	GAN	Section 20
Week 12	Auto-encoders	Section 14
Week 13	Thanksgiving	
Week 14	Reinforcement learning	beyond textbook
Week 15	Final exam	All above

Question?

Deep Learning: introduction

Now, deep learning is everywhere

nowadays deep learning
is applied in society, science,
arts, commerce and finance,
literature, military, …

Back in time,

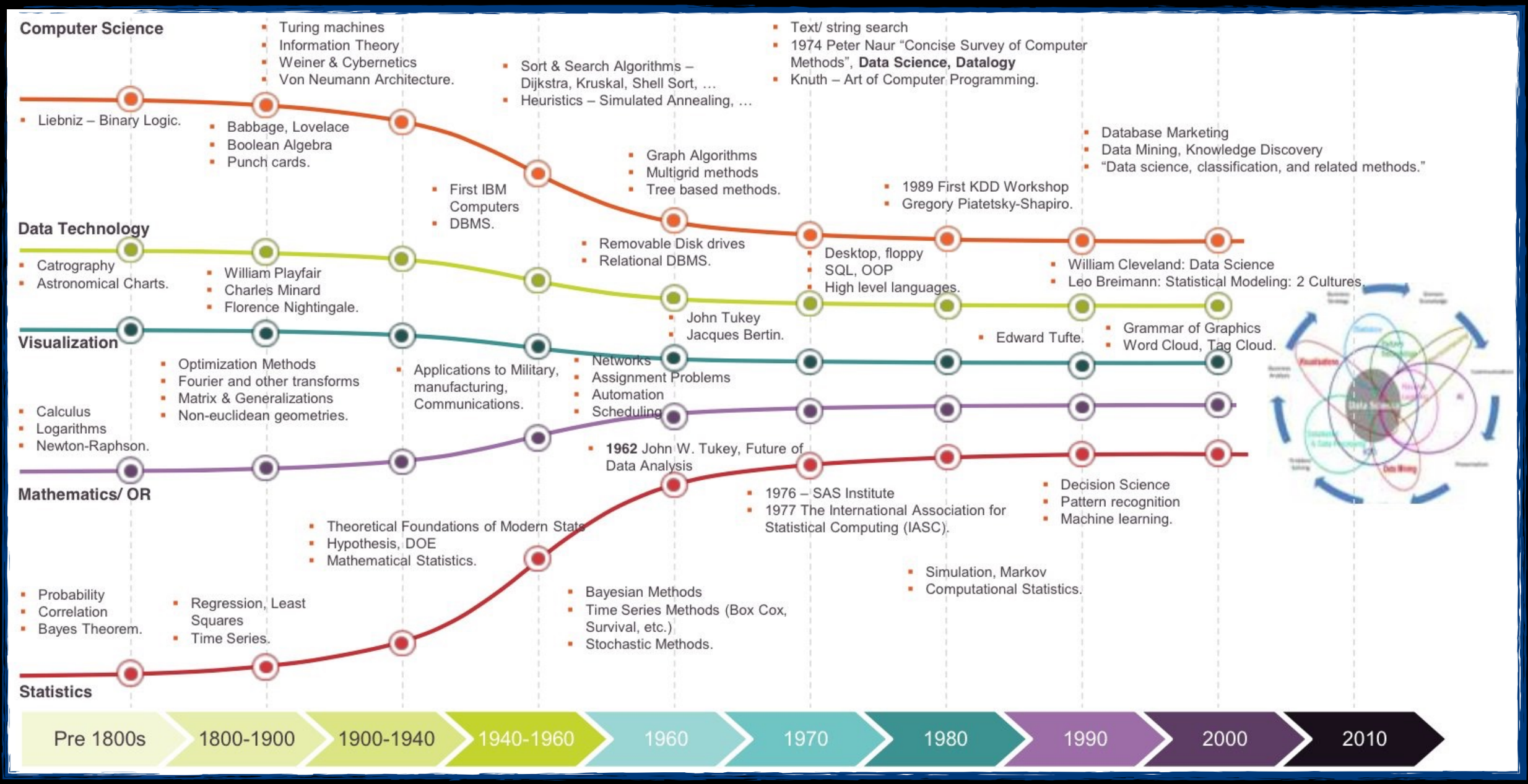
…

How was deep learning introduced and developed?

…

What is the connection of deep learning with other fields?

from CS, statistics to machine learning



by Kirk Borne

examples of AI, machine learning and deep learning

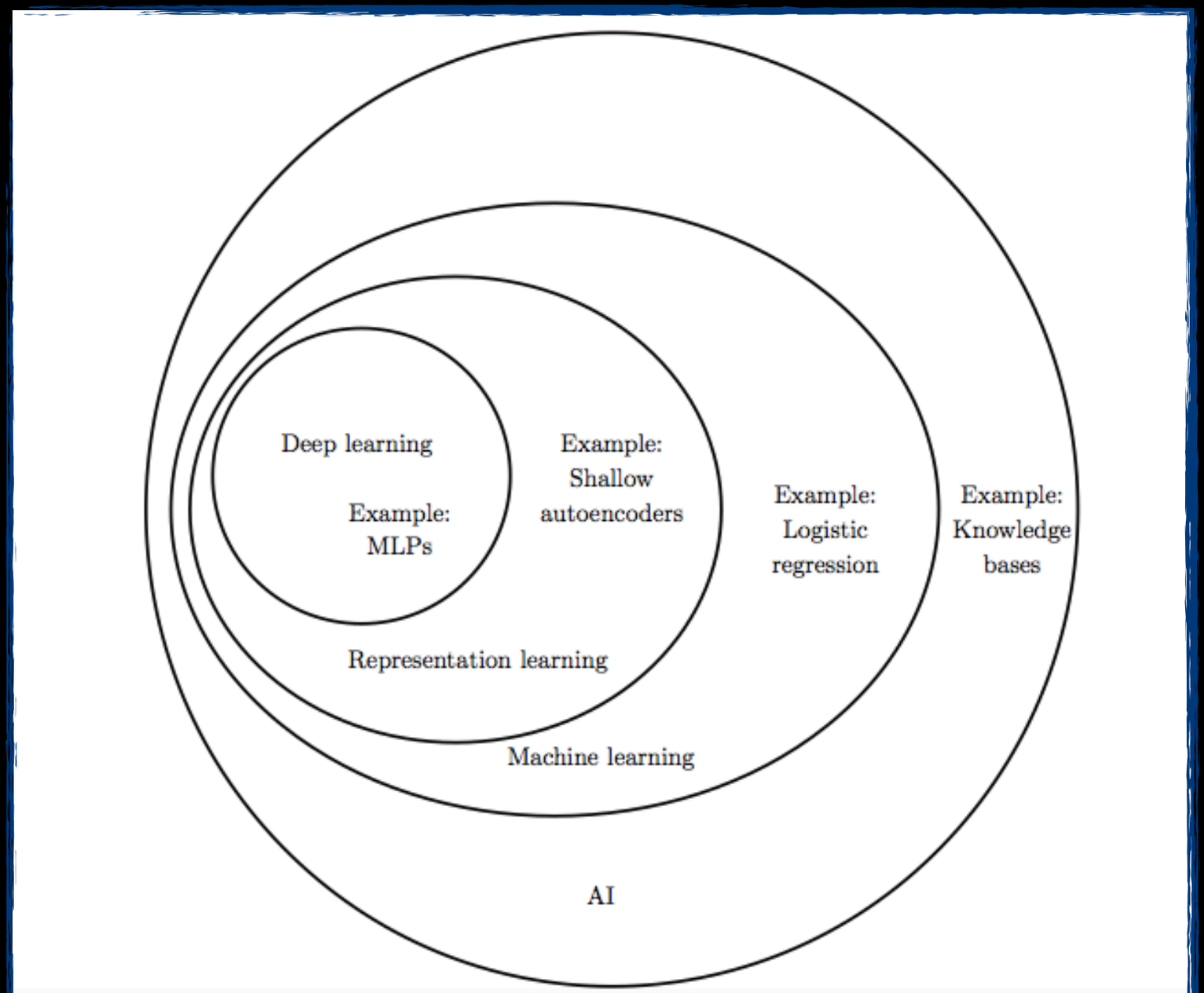
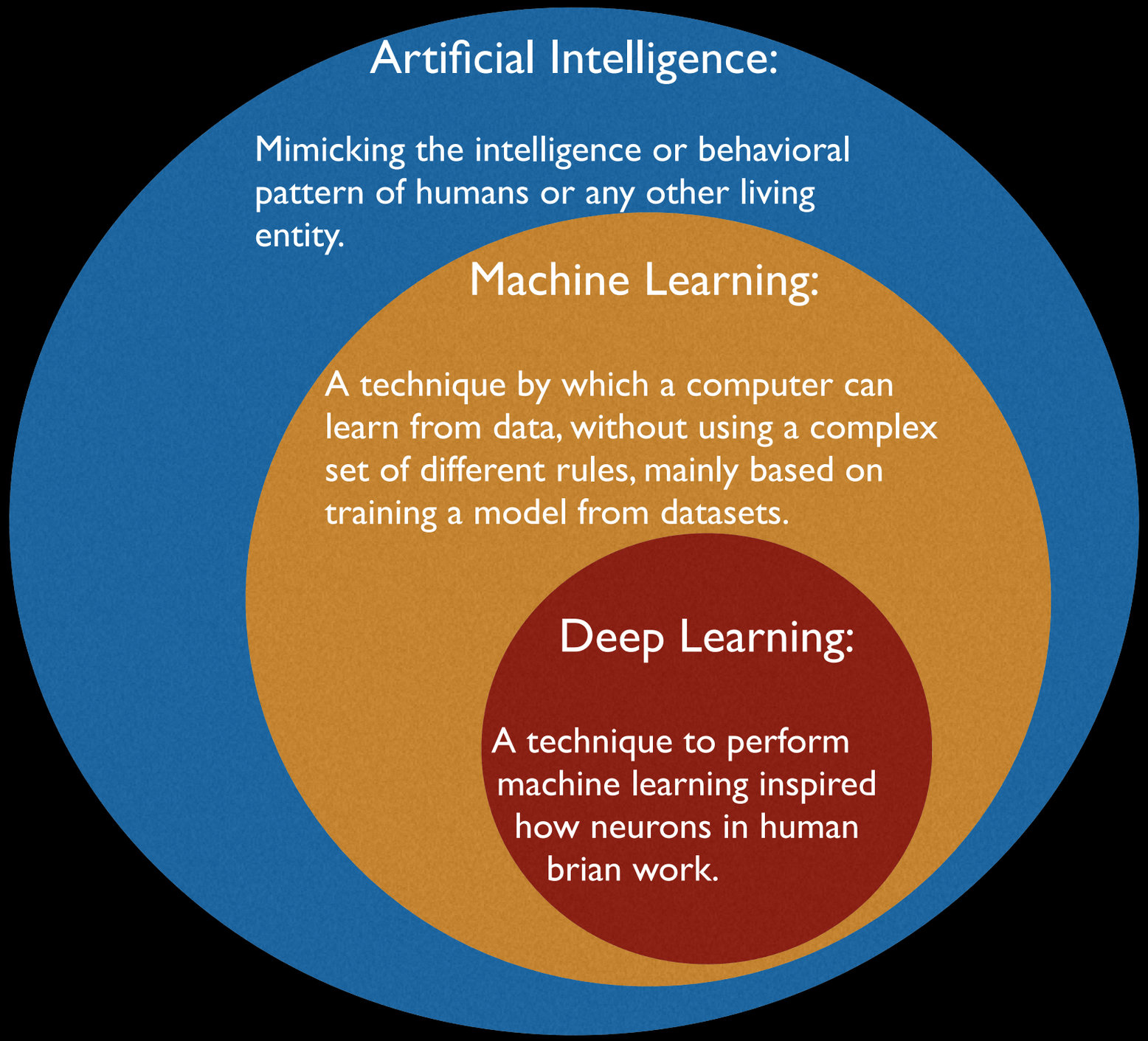
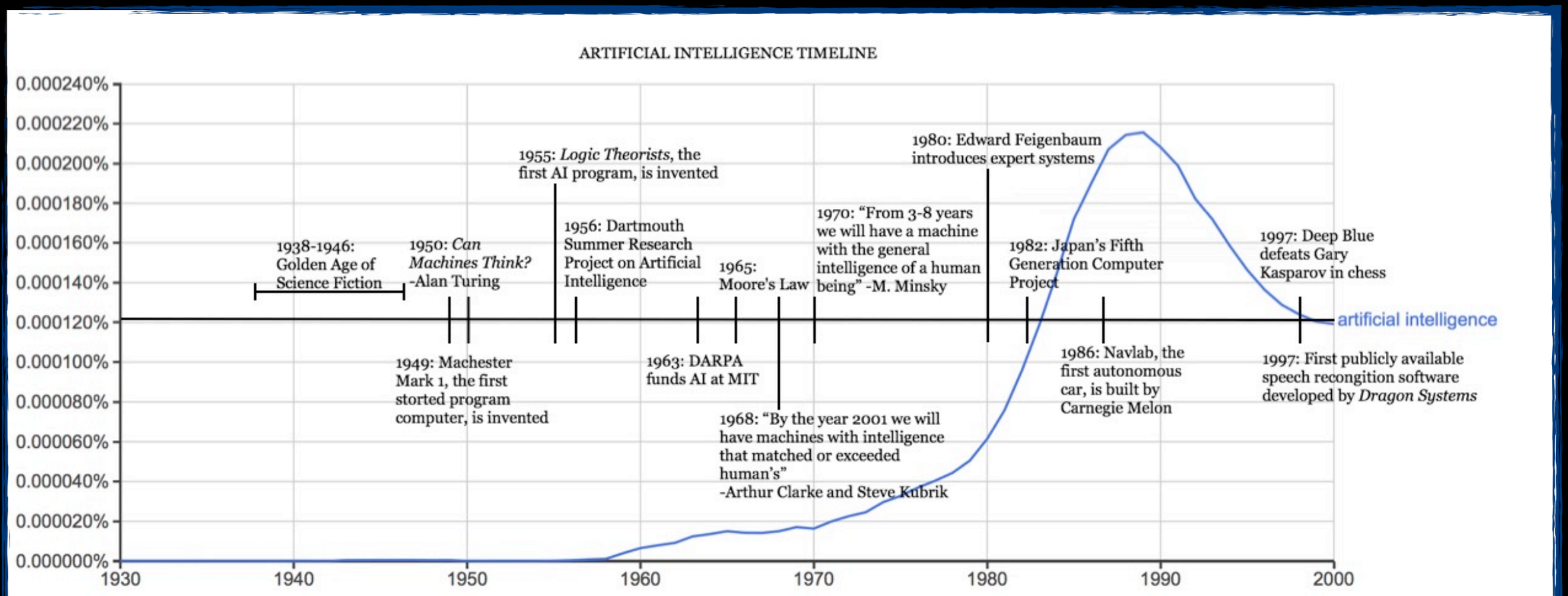


Figure 1.4

AI, machine learning, deep learning

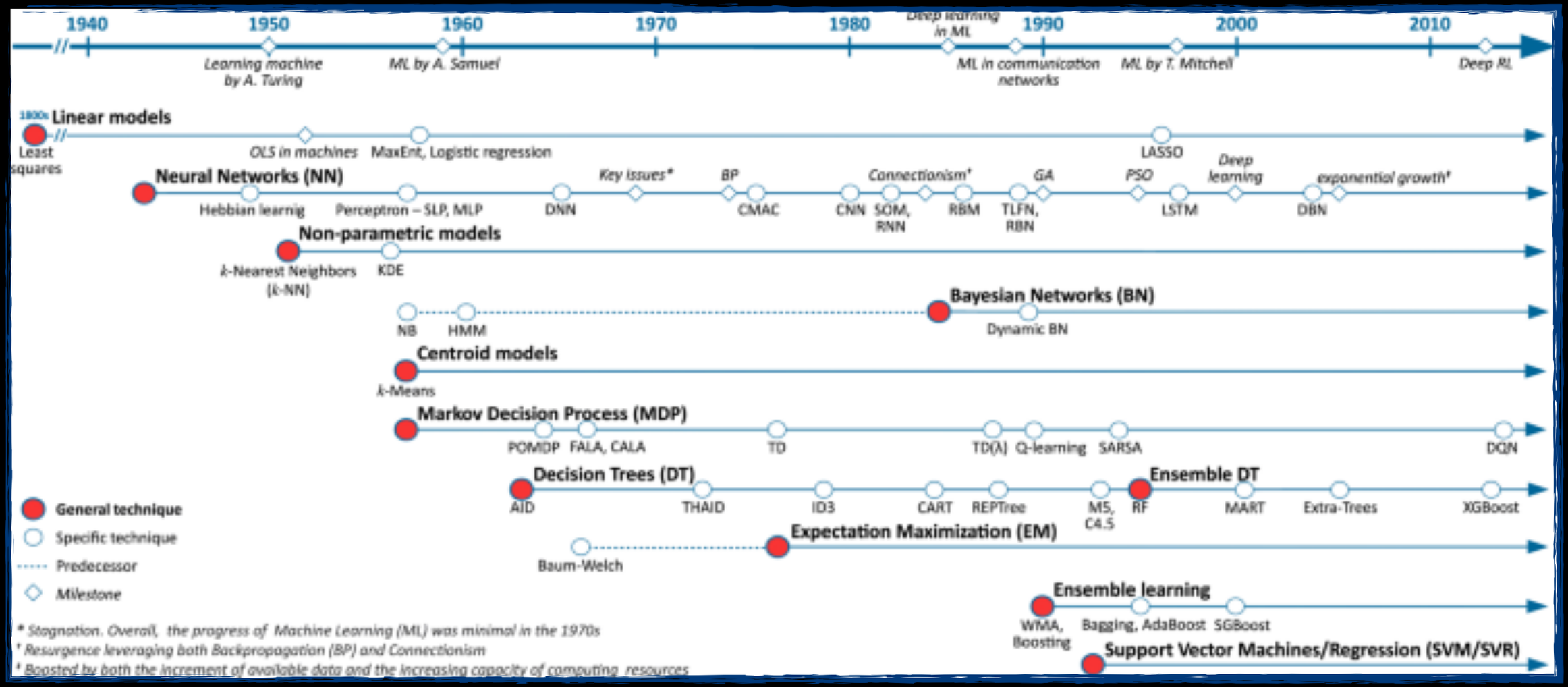


AI history



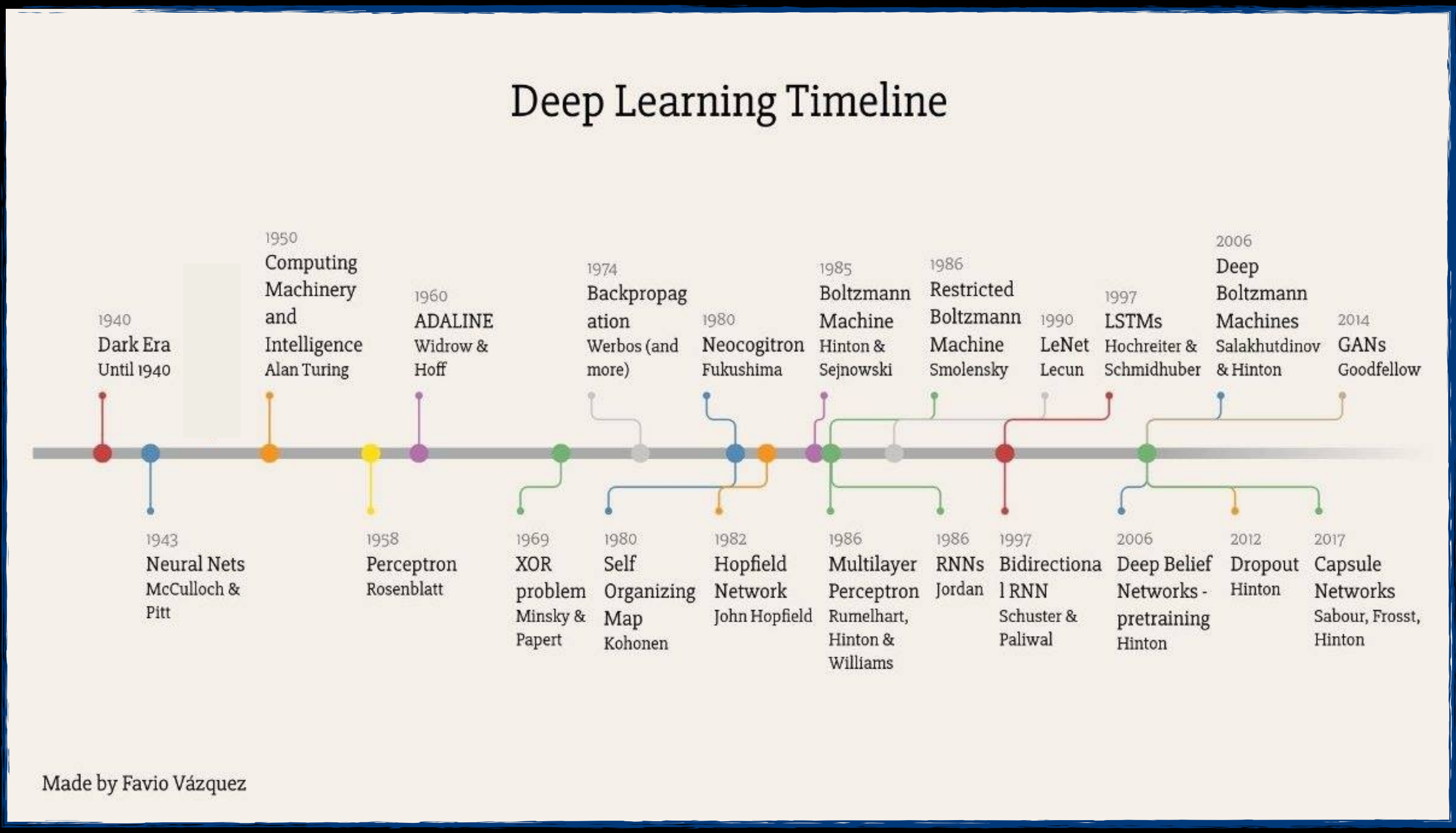
http://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/

machine learning history



https://jisajournal.springeropen.com/articles/10.1186/s13174-018-0087-2

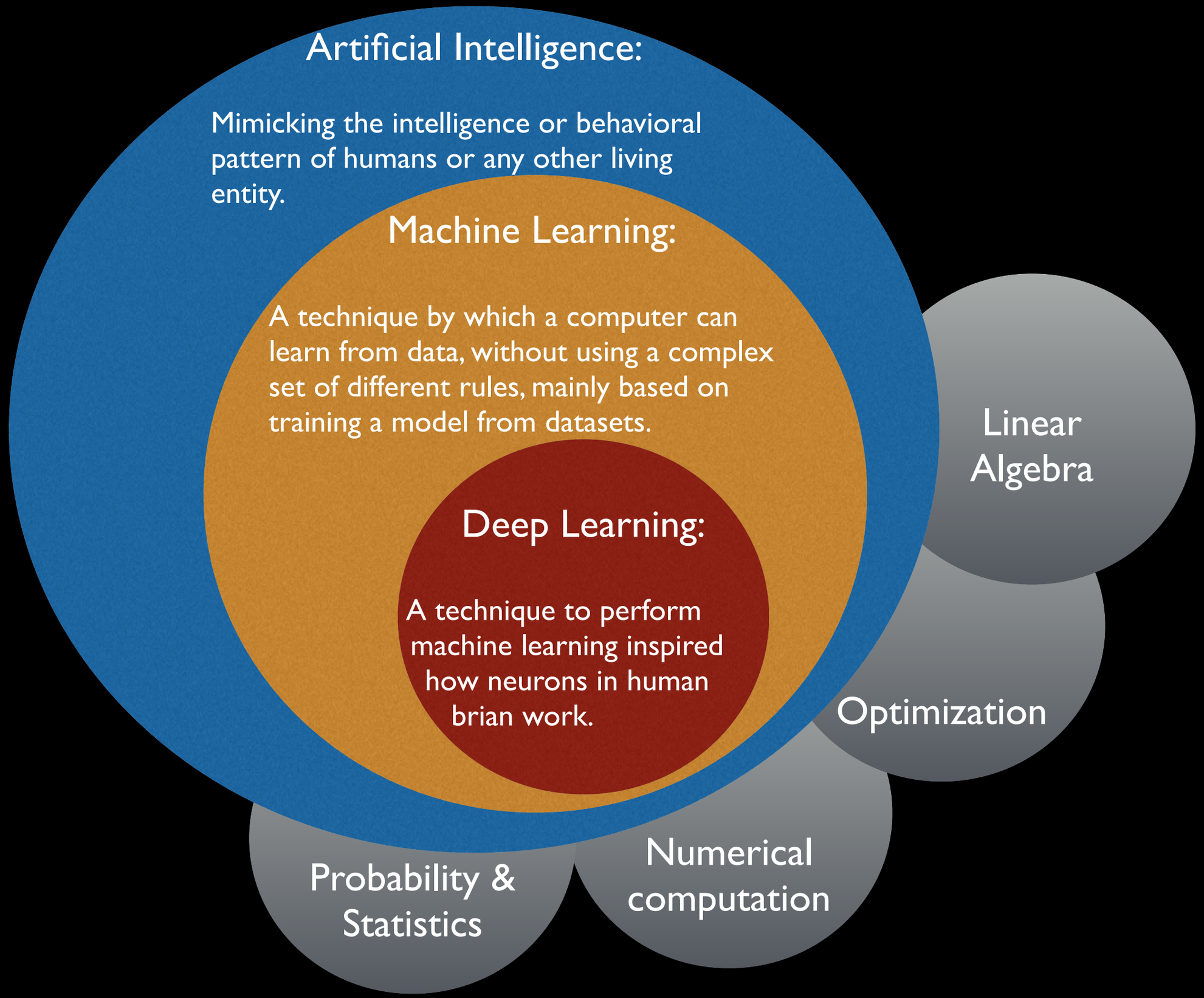
deep learning history



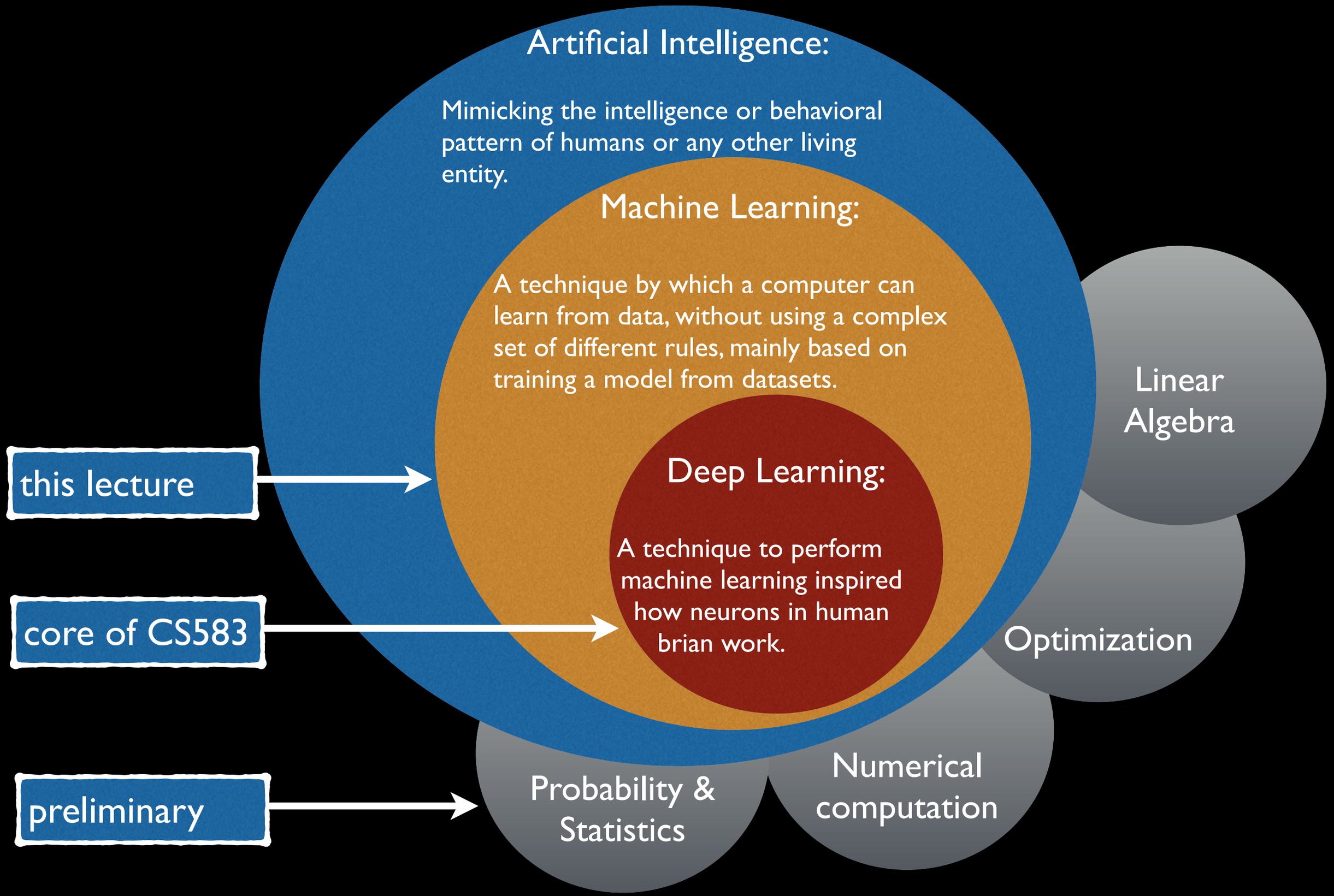
deep learning history

1943: McCulloch and Pitts, a computational model for neural networks based on threshold logic.
1958: Frank Rosenblatt, the perceptron, a two-layer computer neural network using simple addition and subtraction.
1980: Kunihiko Fukushima proposes the Neoconitron, a hierarchical, multilayered artificial neural network for handwriting recognition.
Mid-2000s: Geoffrey Hinton and Ruslan Salakhutdinov, “deep learning” — a many-layered neural network could be pre-trained one layer at a time.
2009: NIPS Workshop on Deep Learning for Speech Recognition
2015: Facebook DeepFace
2016: Google DeepMind’s algorithm AlphaGo
…

AI, machine learning, deep Learning



AI, machine learning, deep Learning



Machine Learning

learning from examples

- detect credit card fraud

given past transactions
and
current purchase

current purchase
valid or fraud ?

predict the answer
from past examples

input

label

- regression tasks in healthcare
- face rec e.g. AiCure in healthcare)
- natural language processing

…

example = input + label

can big data help to **generically**
improve the prediction accuracy ?

machine learning example

binary classification

label (classify)
emails as
spam or ham

From: howtobeabillioner.com
Subject: lottery

You've been selected
among millions to receive....

...viagra...

...Rolex...

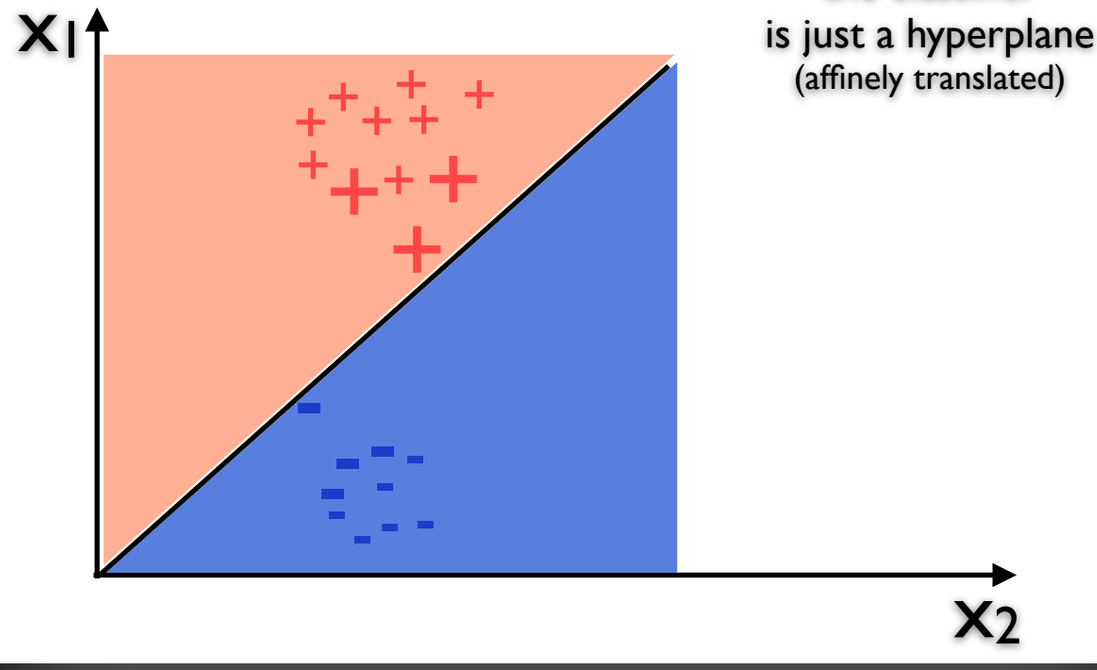
From: mike@tsinghua.edu.cn
Subject: manuscript

hi Nick,

Please find attached the
corrections to our
manuscript.
...

given an email
extract two features:
x_1 and x_2 (real numbers)

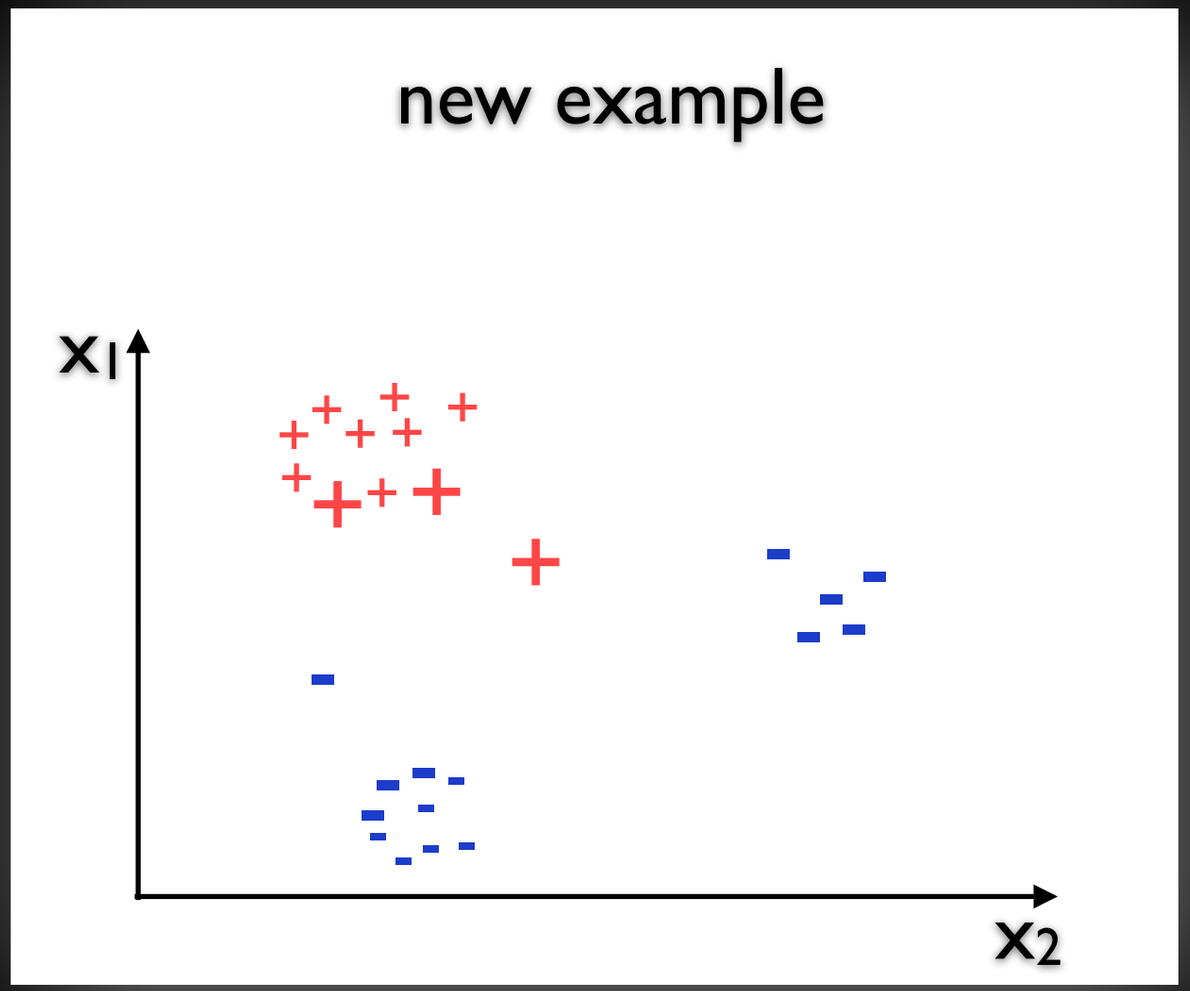
observations (training set)
generalize



this example: no generalization error

machine learning example

who constructs the classifier?



label {red, blue}

instance:(x_1,x_2)

$(Y, (x_1,x_2))$, ...

learner

classifier

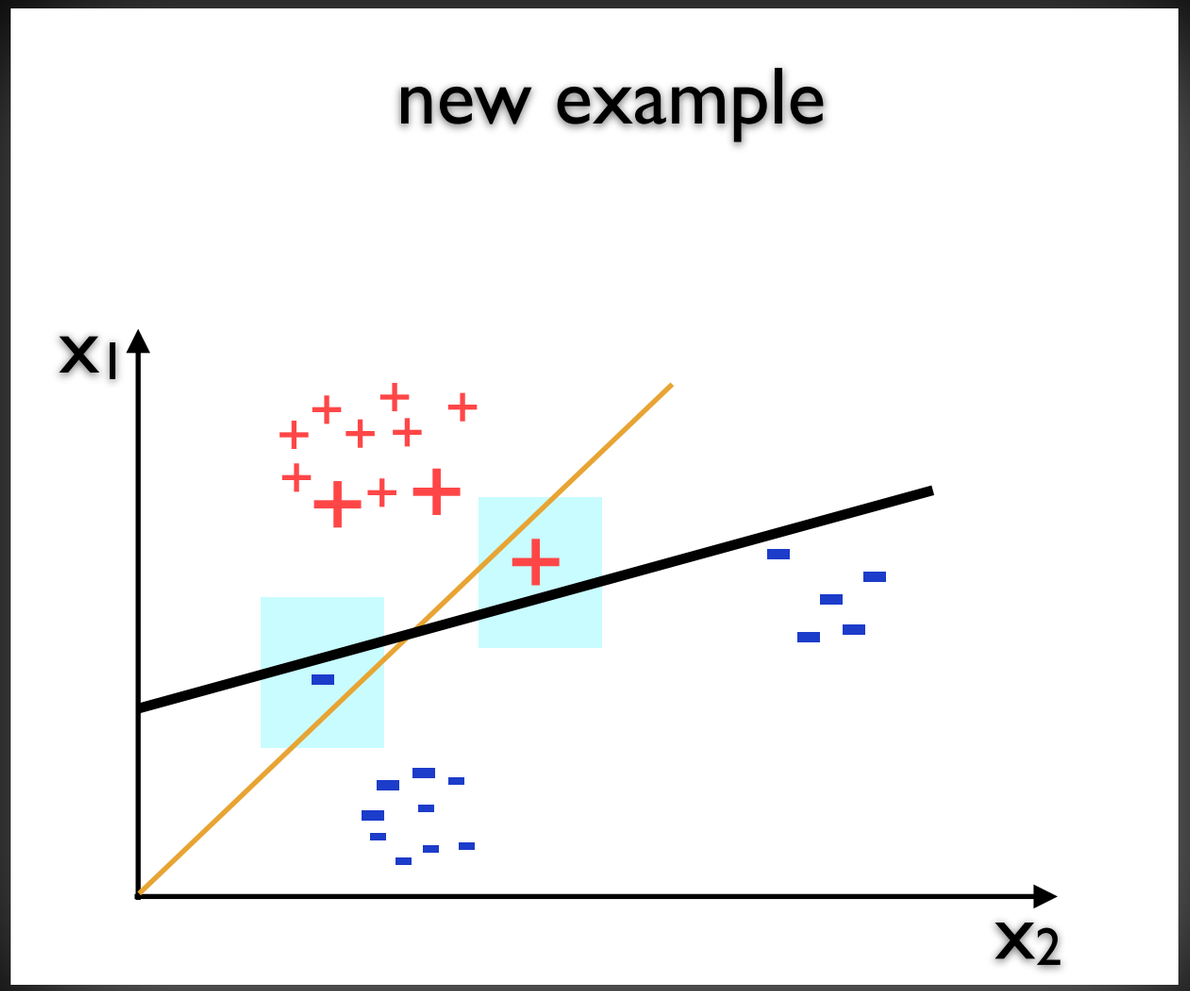
observations
iid distribution μ

target concept: unknown
concept class: known
goal: find out the target
among the possible concepts

machine learning example

who constructs the classifier?

an ensemble can be better than any of the trained classifiers alone



the black line has the form $0.9x + 1.03$

but…

$0.2(x+2)$

$+ \quad 0.2(x+1.2)$

$+ \quad 0.3(x+1.3)$

$0.9x + 1.03$

label {red, blue}

instance:(x_1,x_2)

$(Y, (x_1,x_2)), \ldots$ → learner → classifier

observations
iid distribution μ

but...our learner can only learn affine lines

$x+\beta$, where $|\beta| \leq 2$

workflow of machine learning

instance → (features, spam or not) , (feature, spam or not) , ... ← **class**

Please 101
find 33
attached 25
the 948
corrections 11
to 253
our 92
manuscript 2
...

1 (spam)

pool of observations

↓

an algorithm

learner

learning method, model fitting, …

↓

instance → **classifier** → prediction or classification

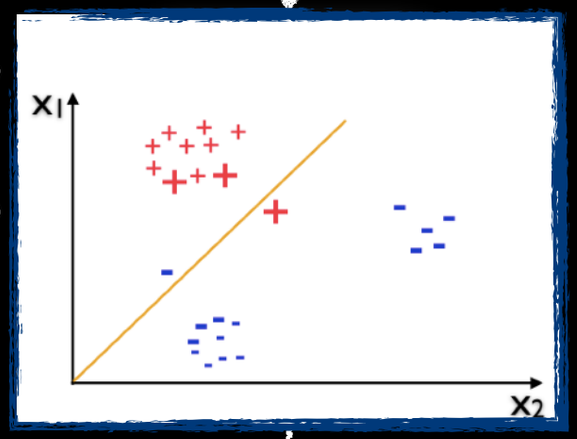
another algorithm constructed by the learner

workflow of machine learning

instance (features, charge or not) , (feature, charge or not) , ... **class**

pool of observations

hyperplane



instance → classifier → prediction or classification

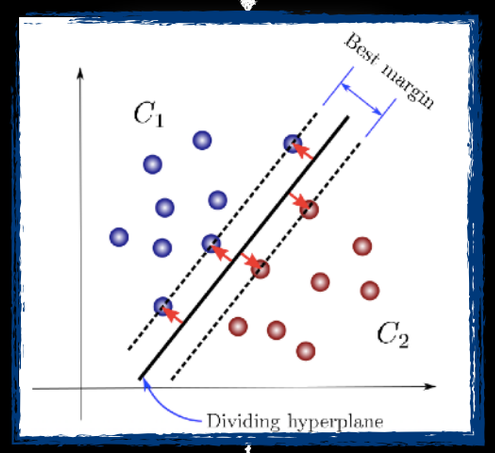
another algorithm constructed by the learner

workflow of machine learning

instance (features, charge or not) , (feature, charge or not) , ... **class**

pool of observations

SVM



instance → **classifier** → prediction or classification

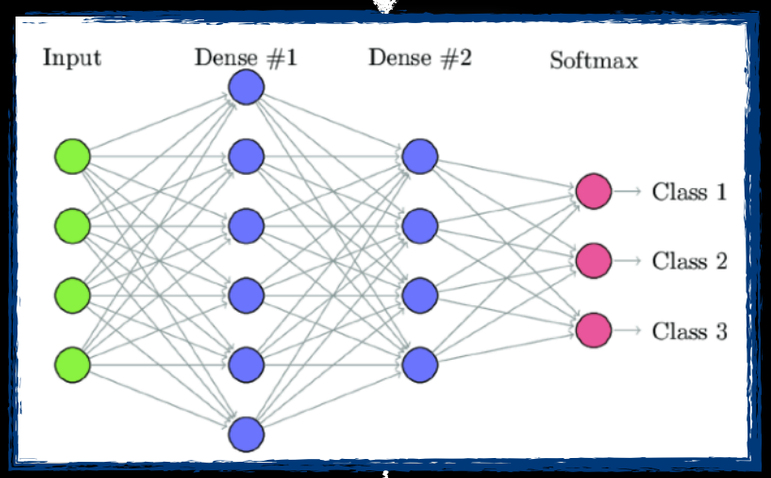
another algorithm constructed by the learner

workflow of machine learning

instance (features, charge or not) , (feature, charge or not) , ... **class**

pool of observations

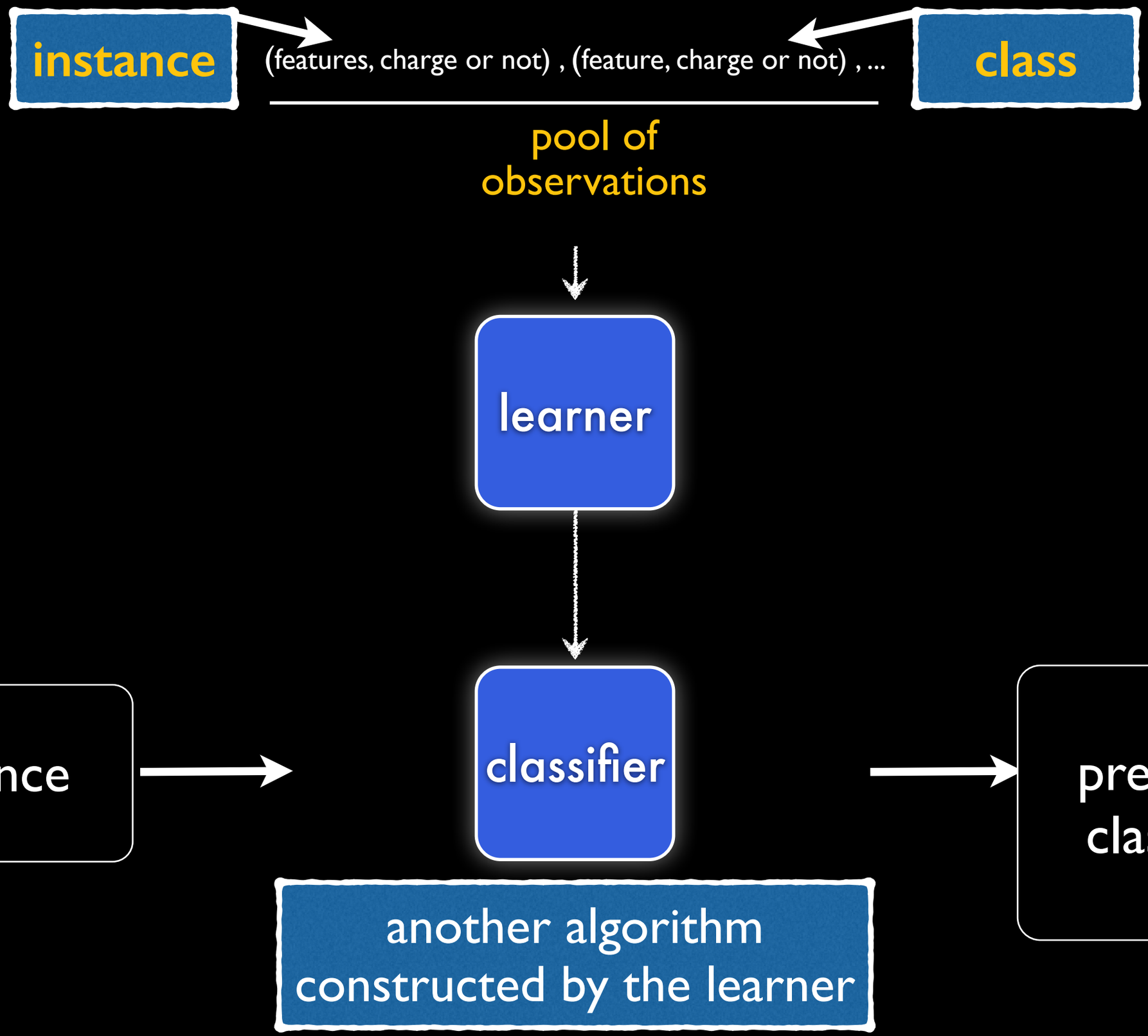
Neural Networks (NN)



instance → classifier → prediction or classification

another algorithm constructed by the learner

workflow of machine learning



workflow of machine learning

instance (features, charge or not) , (feature, charge or not) , ... **class**

- input: observation ***x***
- output: decision ***c***
- Bayes decision rule assuming probabilistic dependencies between ***x*** and ***c***:

$$x \rightarrow \hat{c} = \arg\max_c \left\{ pr(c|x) \right\}$$
$$= \arg\max_c \left\{ pr(c) \cdot pr(x|c) \right\}$$

discriminative

generative

instance → classifier → prediction or classification

another algorithm constructed by the learner

workflow of machine learning

instance (features, charge or not) , (feature, charge or not) , ... **class**

- input: observation ***x***
- output: decision ***c***
- Bayes decision rule assuming probabilistic dependencies between ***x*** and ***c***:

$$x \rightarrow \hat{c} = \arg\max_c \{pr(c|x)\}$$
$$= \arg\max_c \{pr(c) \cdot pr(x|c)\}$$

disc… ge…

modeling

- define model of Pr(c|x) or Pr(c) and Pr(x|c)

learning/training

- learn model parameters

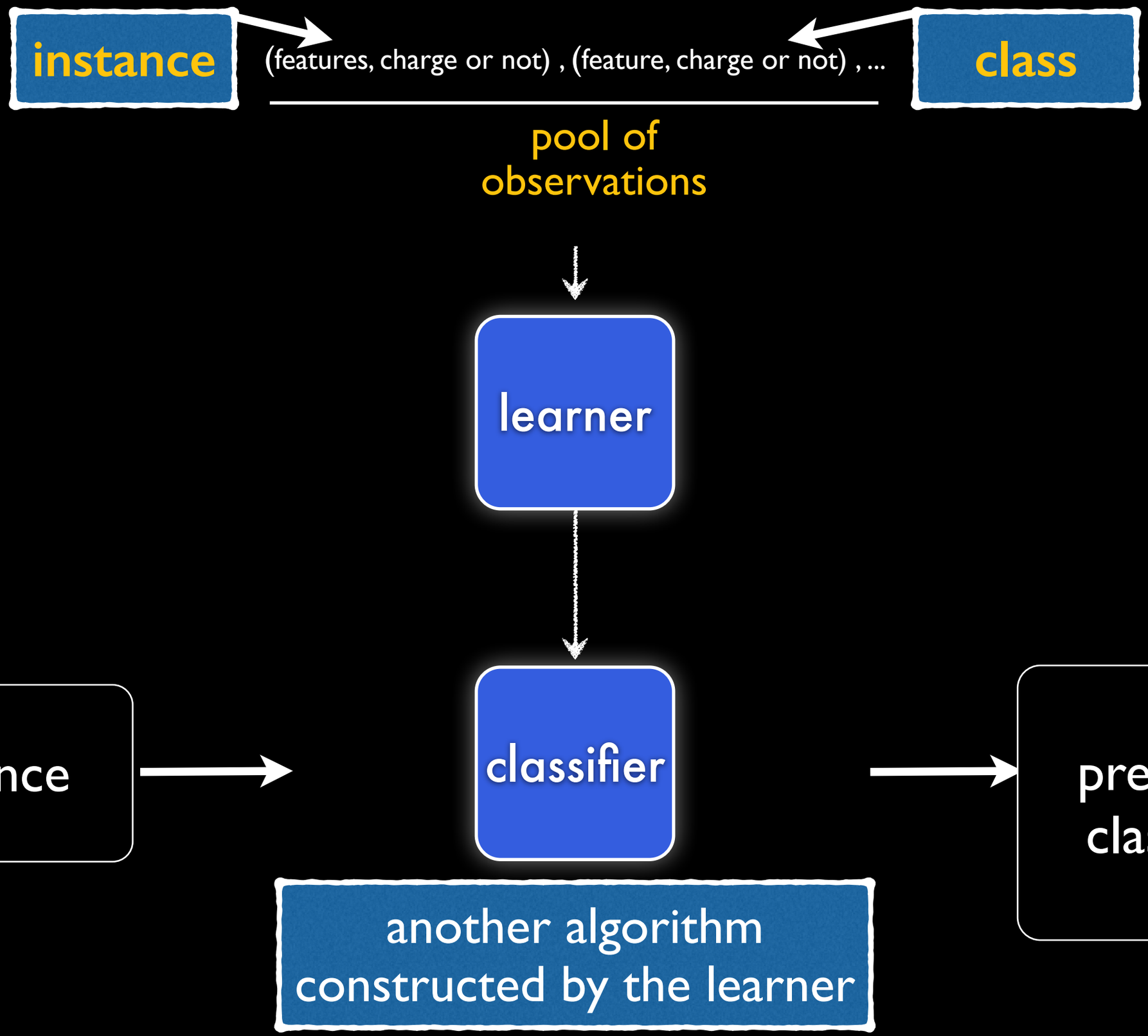
search/decoding

- find the class maximizing the probability distribution

instance → classifier → prediction or classification

another algorithm constructed by the learner

workflow of machine learning



Machine Learning tasks

Classification: spam filtering, face recognition

Osman Khan to Carlos show details Jan 7 (6 days ago) Reply

sounds good
+ok

Carlos Guestrin wrote:
> Let's try to chat on Friday a little to coordinate and more on Sunday in person?
>
> Carlos

Welcome to New Media Installation: Art that Learns

Carlos Guestrin to 10615-announce, Osman, Miche show details 3:15 PM (8 hours ago) Reply

Hi everyone,

Welcome to New Media Installation:Art that Learns

The class will start tomorrow.
Make sure you attend the first class, even if you are on the Wait List.
The classes are held in Doherty Hall C316, and will be Tue, Thu 01:30-4:20 PM.

By now, you should be subscribed to our course mailing list: 10615-announce@cs.cmu.edu.
You can contact the instructors by emailing: 10615-instructors@cs.cmu.edu

Natural _LoseWeight SuperFood Endorsed by Oprah Winfrey, Free Trial 1 bottle, pay only $5.95 for shipping mfw rlk Spam | X

Jaquelyn Halley to nherrlein, bcc: thehorney, bcc: anç show details 9:52 PM (1 hour ago) Reply

=== Natural WeightL0SS Solution ===

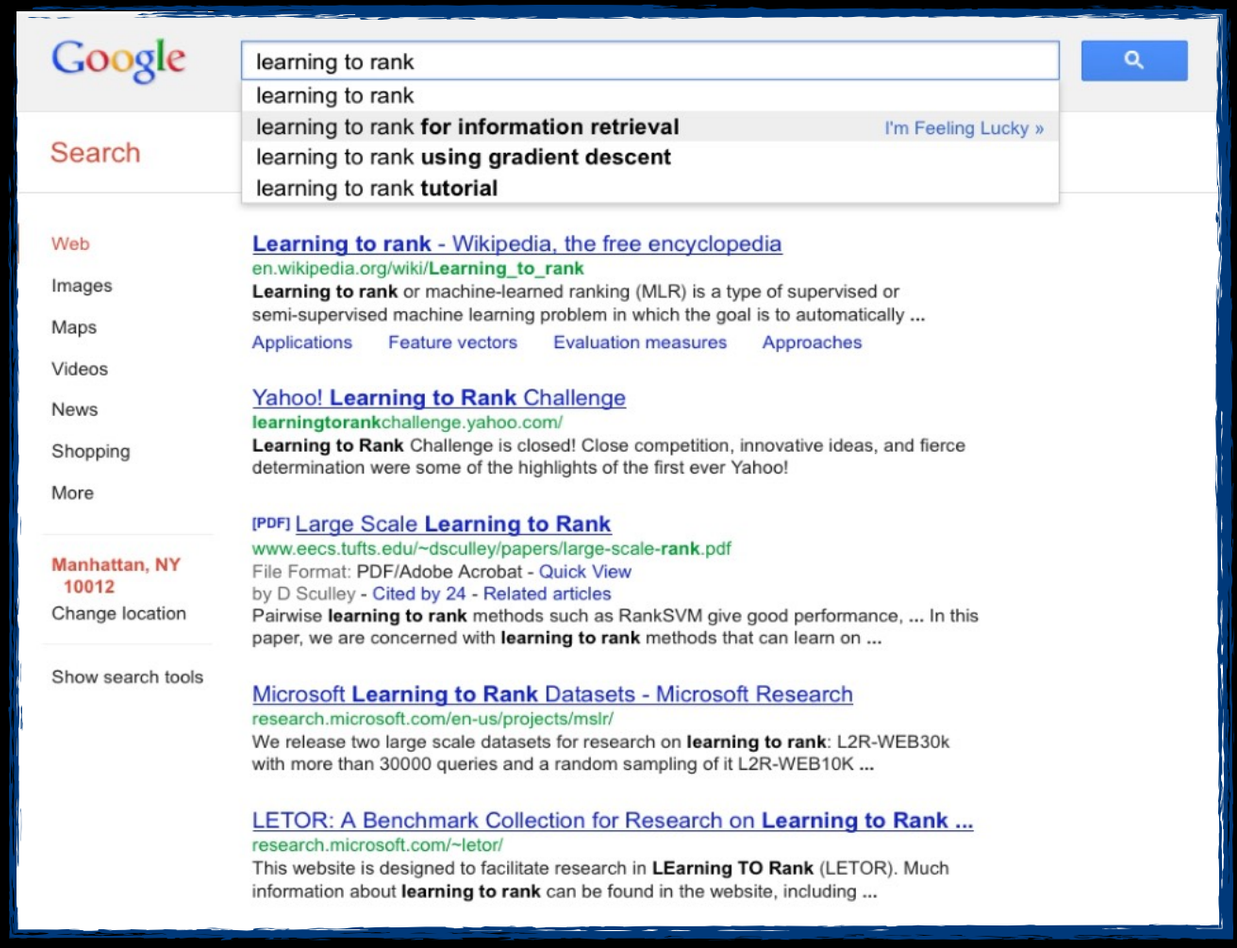
Vital Acai is a natural WeightL0SS product that Enables people to lose wieght and cleansing their bodies faster than most other products on the market.

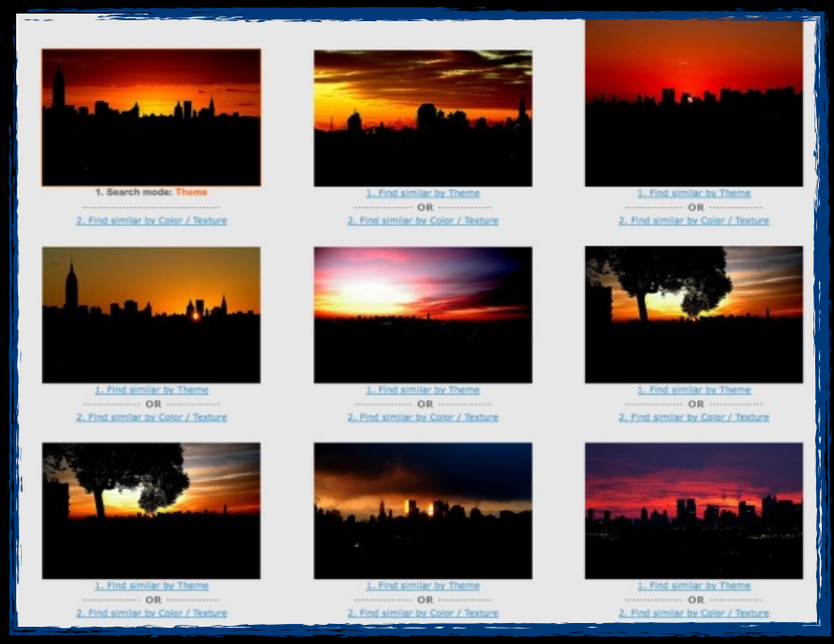
Here are some of the benefits of Vital Acai that You might not be aware of. These benefits have helped people who have been using Vital Acai daily to Achieve goals and reach new heights in there dieting that they never thought they could.

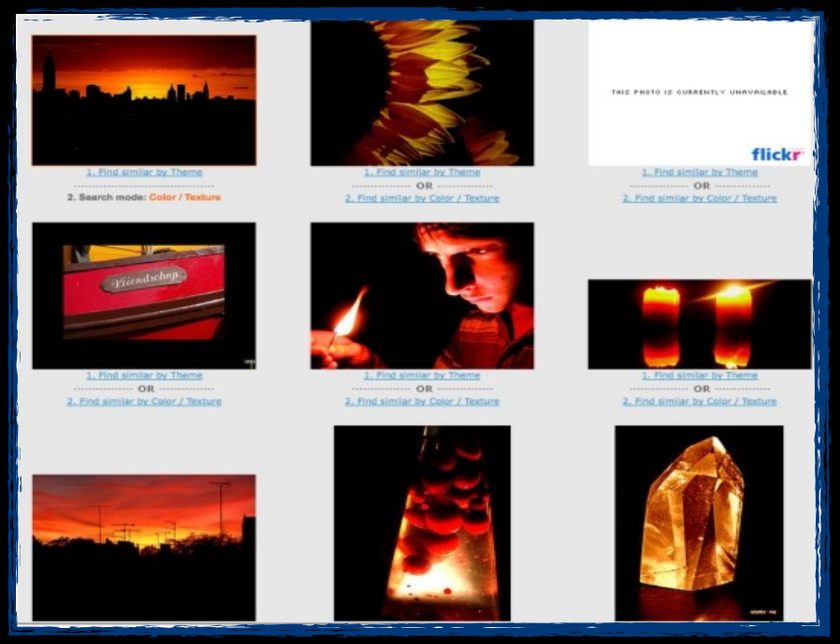
* Rapid WeightL0SS
* Increased metabolism - BurnFat & calories easily!
* Better Mood and Attitude
* More Self Confidence
* Cleanse and Detoxify Your Body
* Much More Energy
* BetterSexLife
* A Natural Colon Cleanse

Regression: weather, stock trend prediction

Ranking: web search, find similar images

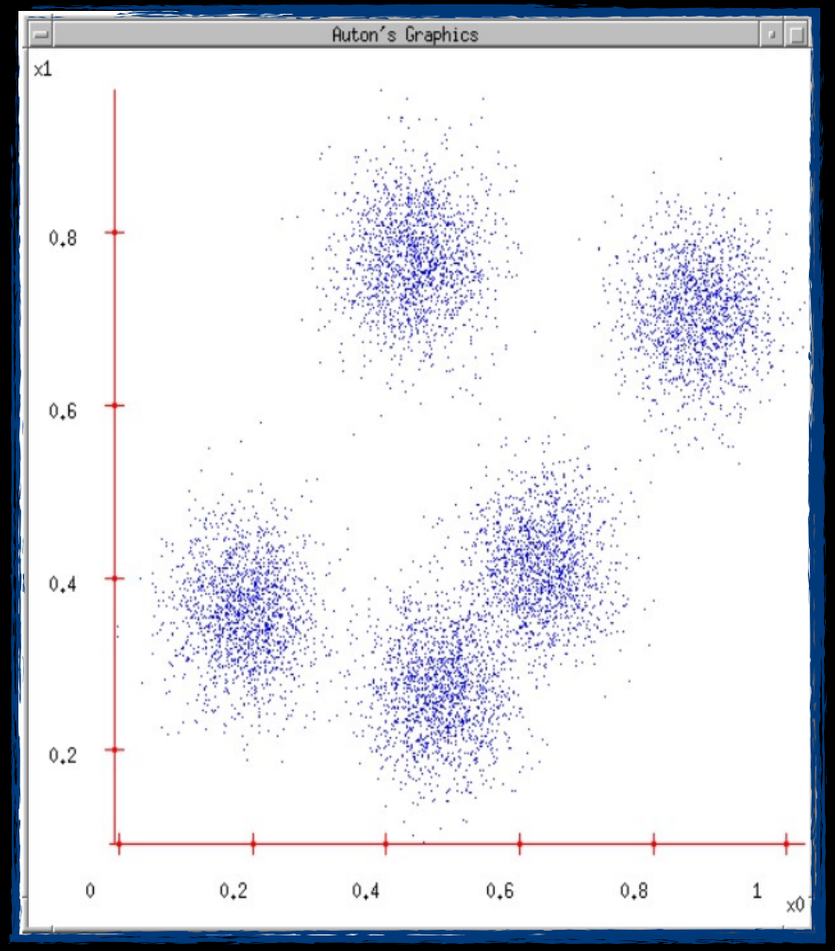




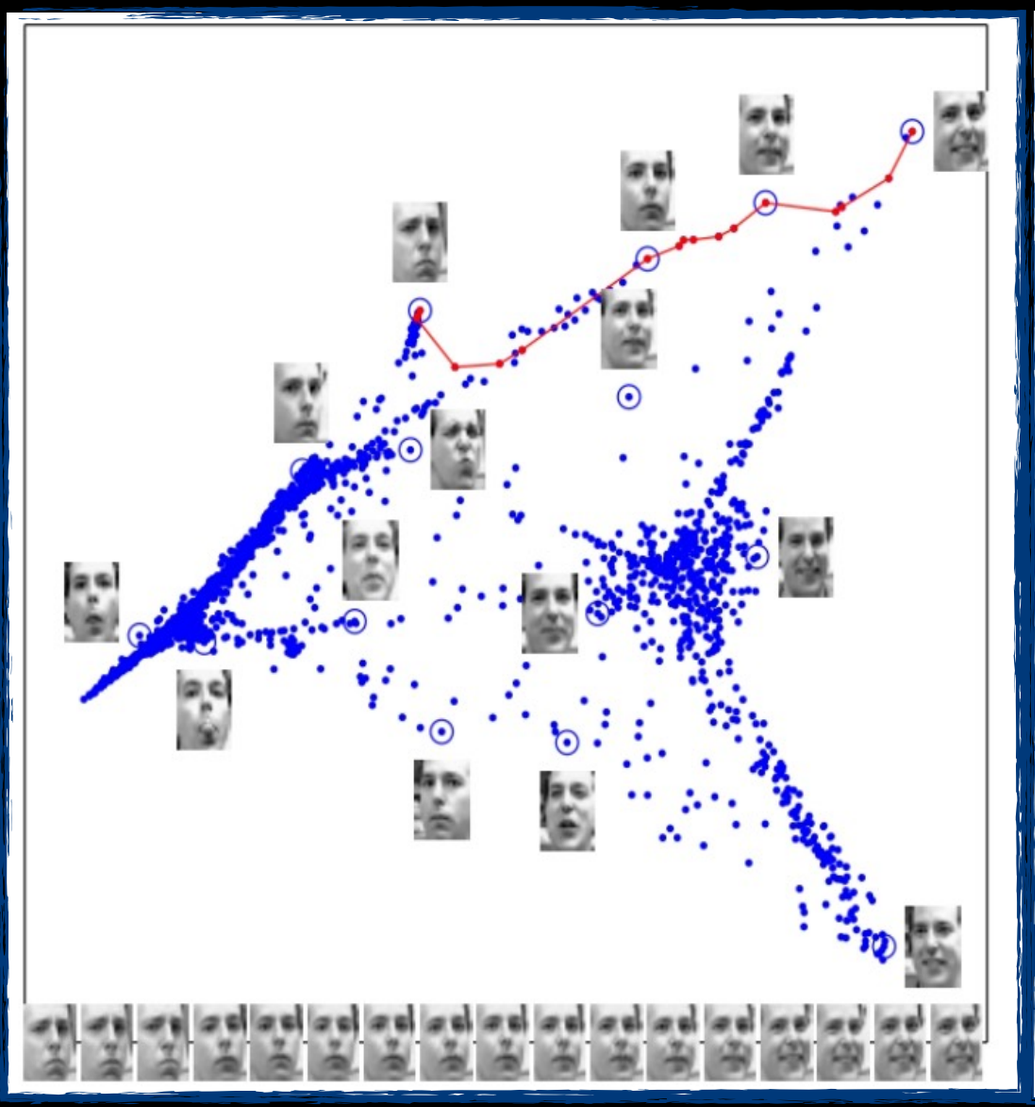


Collaborative filtering: recommendation

Clustering: group similar things



embedding: visualize data



Machine learning applications

Machine learning is preferred approach to

- Speech recognition
- Natural language processing
- Computer vision
- Medical outcomes analysis
- Robot control
- Computational biology
- Sensor networks
- …

This trend is accelerating

- Big data
- Improved machine learning algorithms
- Faster computers
- Good open-source software

Machine Learning problem

Supervised Learning

- Given: Training set $\{(x_i, y_i) \mid i = 1 \ldots N\}$
- Find: A good approximation to $f : X \rightarrow Y$

Examples: what are X and Y ?

- Spam Detection
  - Map email to {Spam, Not Spam}
- Digit recognition
  - Map pixels to {0,1,2,3,4,5,6,7,8,9}
- Stock Prediction

A Supervised Learning Problem

Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

- Our goal is to find a function f : X->Y
  - X = {0,1}^4
  - Y = {0,1}
- Question 1: How should we pick the hypothesis space, the set of possible functions f ?
- Question 2: How do we find the best f in the hypothesis space?

Most General Hypothesis Space

Consider all possible boolean functions over four

- 2^16 possible hypotheses
- 2^9 are consistent with our dataset
- How do we choose the best one?

x_1	x_2	x_3	x_4	y
0	0	0	0	?
0	0	0	1	?
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	?
1	0	0	0	?
1	0	0	1	1
1	0	1	0	?
1	0	1	1	?
1	1	0	0	0
1	1	0	1	?
1	1	1	0	?
1	1	1	1	?

Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

A Restricted Hypothesis Space

Consider all conjunctive boolean functions.

- 16 possible hypotheses
- None are consistent with our dataset
- How do we choose the best one?

Rule	Counterexample
$\Rightarrow y$	1
$x_1 \Rightarrow y$	3
$x_2 \Rightarrow y$	2
$x_3 \Rightarrow y$	1
$x_4 \Rightarrow y$	7
$x_1 \wedge x_2 \Rightarrow y$	3
$x_1 \wedge x_3 \Rightarrow y$	3
$x_1 \wedge x_4 \Rightarrow y$	3
$x_2 \wedge x_3 \Rightarrow y$	3
$x_2 \wedge x_4 \Rightarrow y$	3
$x_3 \wedge x_4 \Rightarrow y$	4
$x_1 \wedge x_2 \wedge x_3 \Rightarrow y$	3
$x_1 \wedge x_2 \wedge x_4 \Rightarrow y$	3
$x_1 \wedge x_3 \wedge x_4 \Rightarrow y$	3
$x_2 \wedge x_3 \wedge x_4 \Rightarrow y$	3
$x_1 \wedge x_2 \wedge x_3 \wedge x_4 \Rightarrow y$	3

Dataset:

Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Occam's Razor Principle

- William of Occam: Monk living in the 14th century
- Principle of parsimony:

"One should not increase, beyond what is necessary, the number of entities required to explain anything"

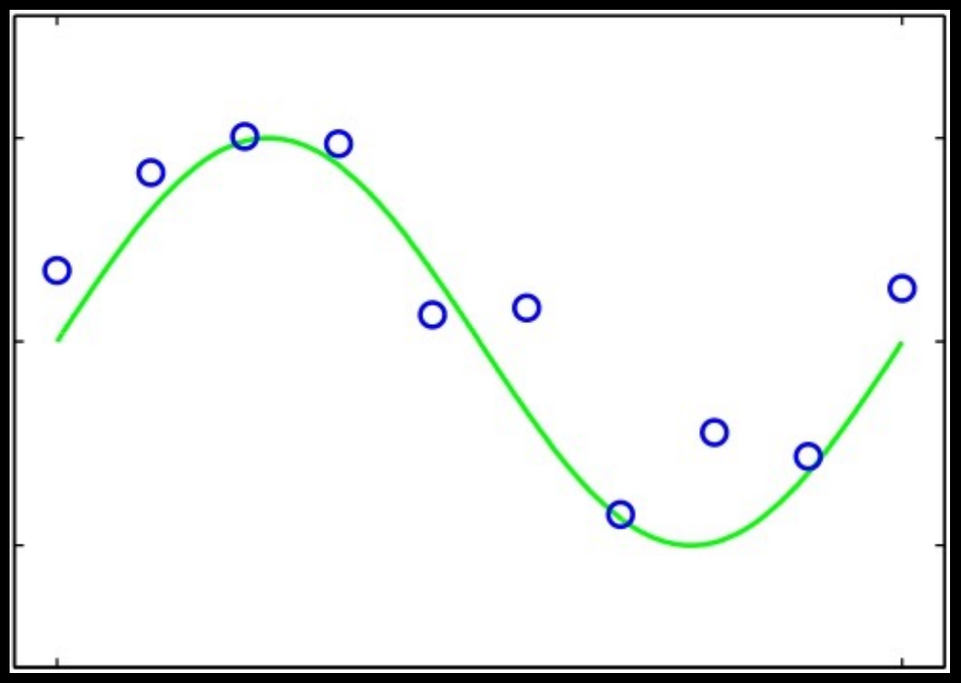
- When many solutions are available for a given problem, we should select the simplest one
- But what do we mean by simple?
- We will use prior knowledge of the problem to define what is a simple solution

Example of a prior: smoothness

[Samy Bengio]

Another Example: Regression

Dataset: 10 (X,Y) points generated from a sin function, with noise

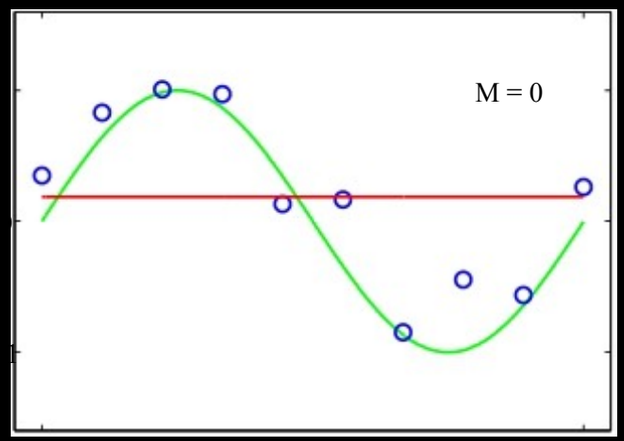


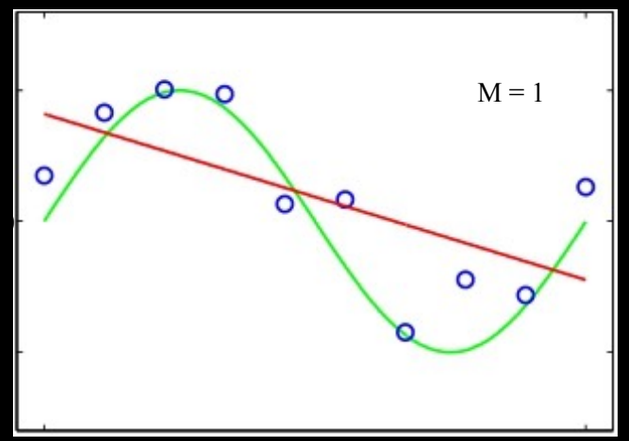
- Regression:
  - $f : X \rightarrow Y$
  - $X = \Re$
  - $Y = \Re$

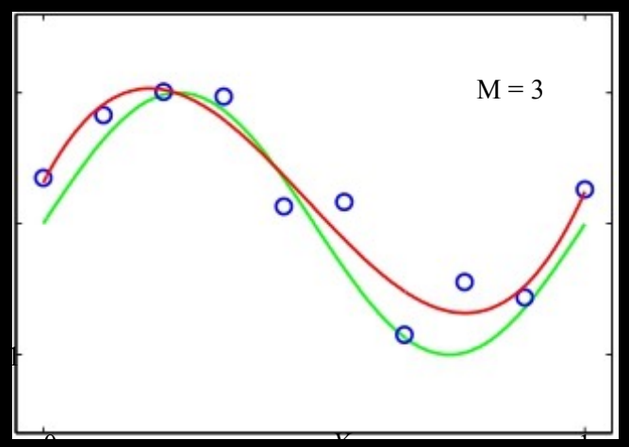
Degree-M Polynomials

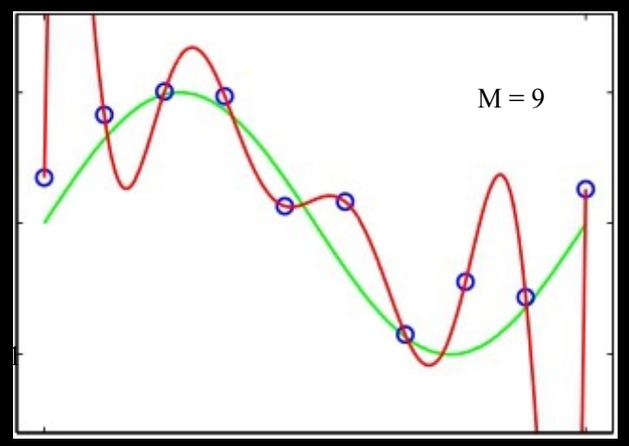
How about letting *f* be a degree M polynomial?

- Which one is **best** ?

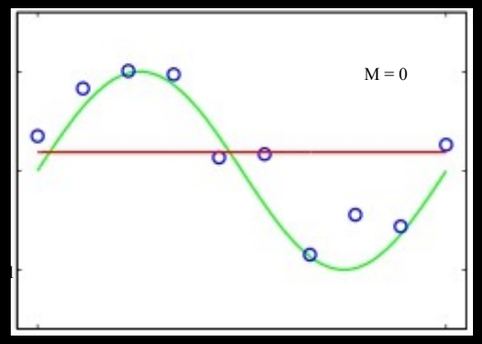


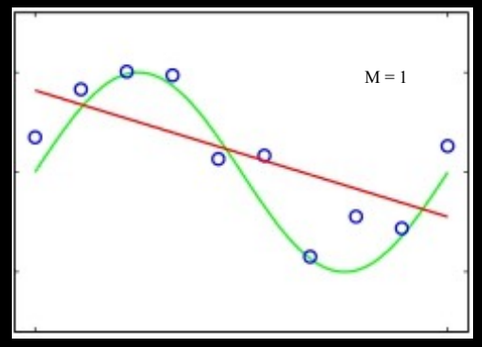


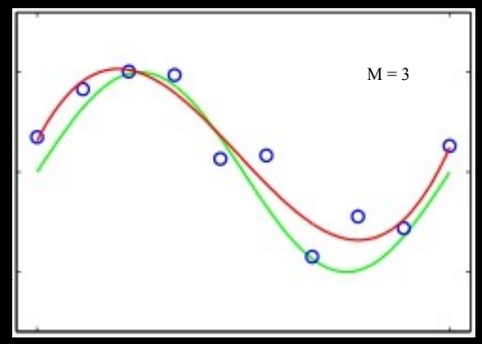


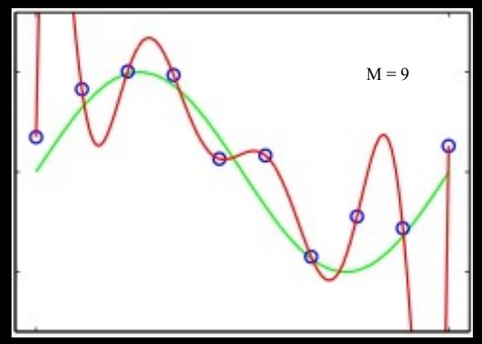


Hypo. Space: Degree-N Polynomials









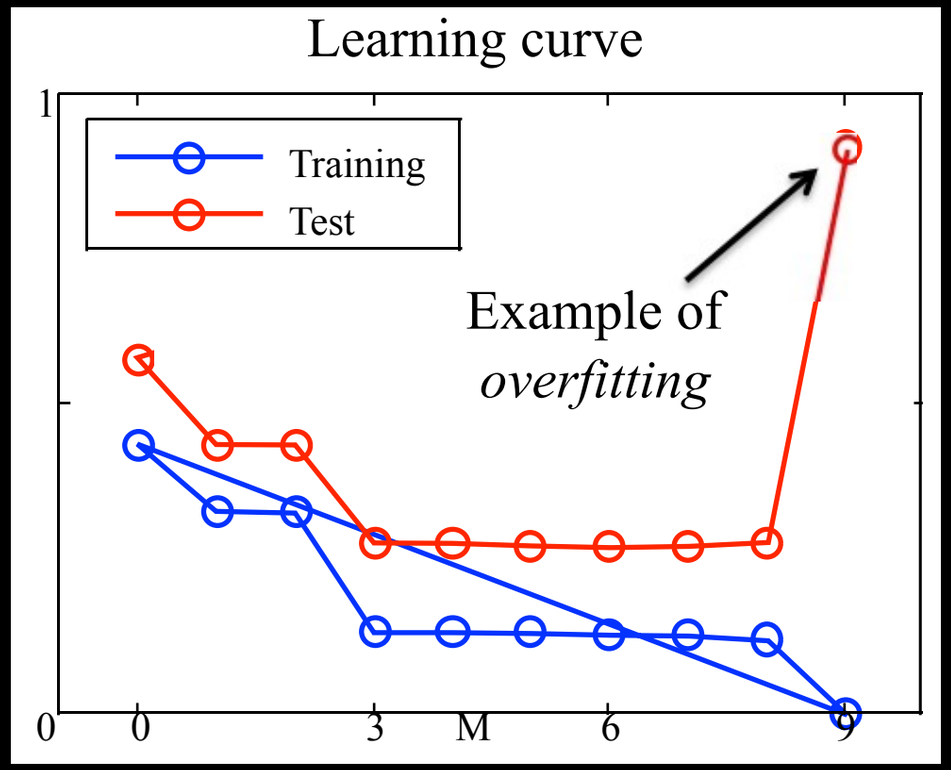
We measure error using a *loss function*

For regression, a common choice is squared loss:

$$L(y_i, f(x_i)) = (y_i - f(x_i))^2$$

The empirical loss of the function f applied to the training data is then:

$$\frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$



Key Issues in Machine Learning

- How do we choose a hypothesis space?
  - Often we use **prior knowledge** to guide this choice
- How can we gauge the accuracy of a hypothesis on unseen data?
  - **Occam's razor:** use the *simplest* hypothesis consistent with data! This will help us avoid overfitting.
  - ***Learning theory*** will help us quantify our ability to ***generalize*** as a function of the amount of training data and the hypothesis space
- How do we find the best hypothesis?
  - This is an **algorithmic** question, the main topic of computer science
- How to model applications as machine learning problems? (engineering challenge)

Perceptron algorithm

Binary classification

- Input: email
- Output: spam/ham
- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ ham”
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future emails

- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

Dear Sir.

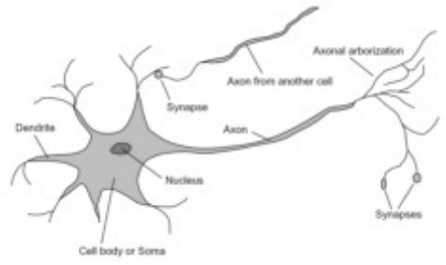
First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY $99

Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

The perceptron algorithm



- 1957: Perceptron algorithm invented by Rosenblatt

  Wikipedia: “A handsome bachelor, he drove a classic MGA sports… for several years taught an interdisciplinary undergraduate honors course entitled "Theory of Brain Mechanisms" that drew students equally from Cornell's Engineering and Liberal Arts colleges…this course was a melange of ideas .. experimental brain surgery on epileptic patients while conscious, experiments on .. the visual cortex of cats, ... analog and digital electronic circuits that modeled various details of neuronal behavior (i.e. the perceptron itself, as a machine).”

  - Built on work of Hebbs (1949); also developed by Widrow-Hoff (1960)

- 1960: Perceptron Mark 1 Computer – hardware implementation

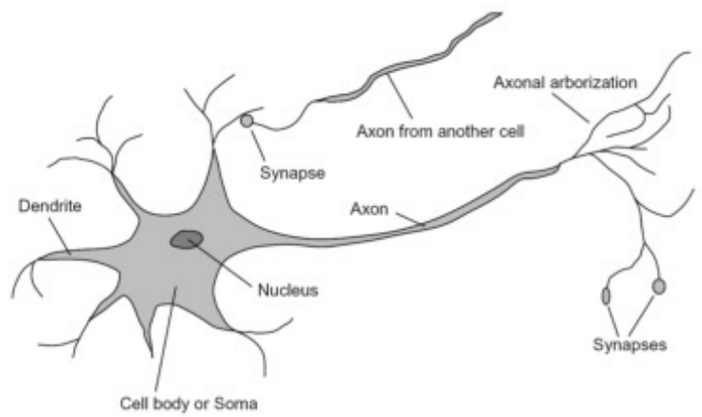
- 1969: Minksky & Papert book shows perceptrons limited to *linearly separable* data

- 1970’s: Learning methods for two-layer neural networks

[William Cohen]

Linear Classifiers

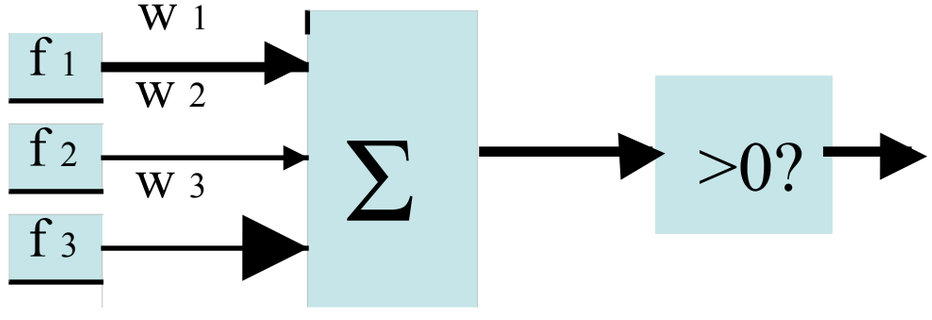
- Inputs are feature values
- Each feature has a weight
- Sum is the activation



Important note: changing notation!

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- If the activation is:
  - Positive, output *class 1*
  - Negative, output *class 2*



Example: Spam

- Imagine 3 features (spam is “ positive” class):
  1. free (number of occurrences of “ free”)
  2. money (occurrences of “ money”)
  3. BIAS (intercept, always has value 1)

$$w \cdot f(x)$$
$$\sum_i w_i \cdot f_i(x)$$

x	$f(x)$	w
“ free money”	BIAS : 1 free : 1 money : 1 ...	BIAS : -3 free : 4 money : 2 ...

$$(1)(-3) + (1)(4) + (1)(2) + \ldots = 3$$

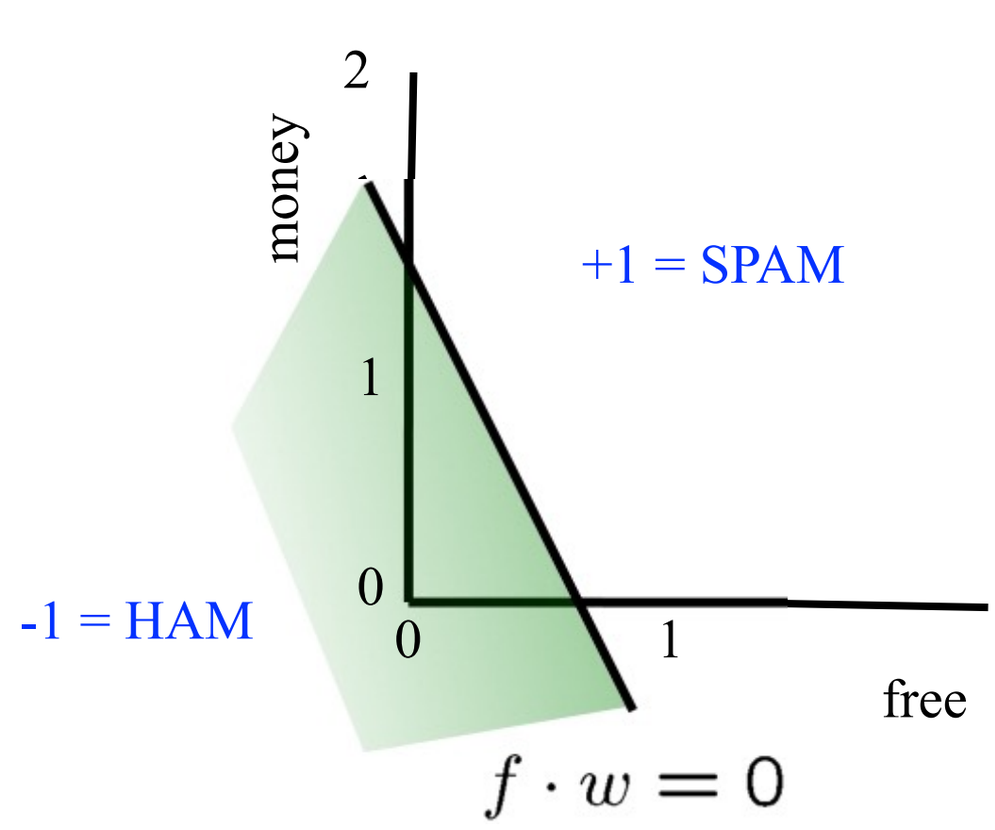
w.f(x) > 0 SPAM!!!

Binary Decision Rule

- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y=+1
  - Other corresponds to Y=-1

w

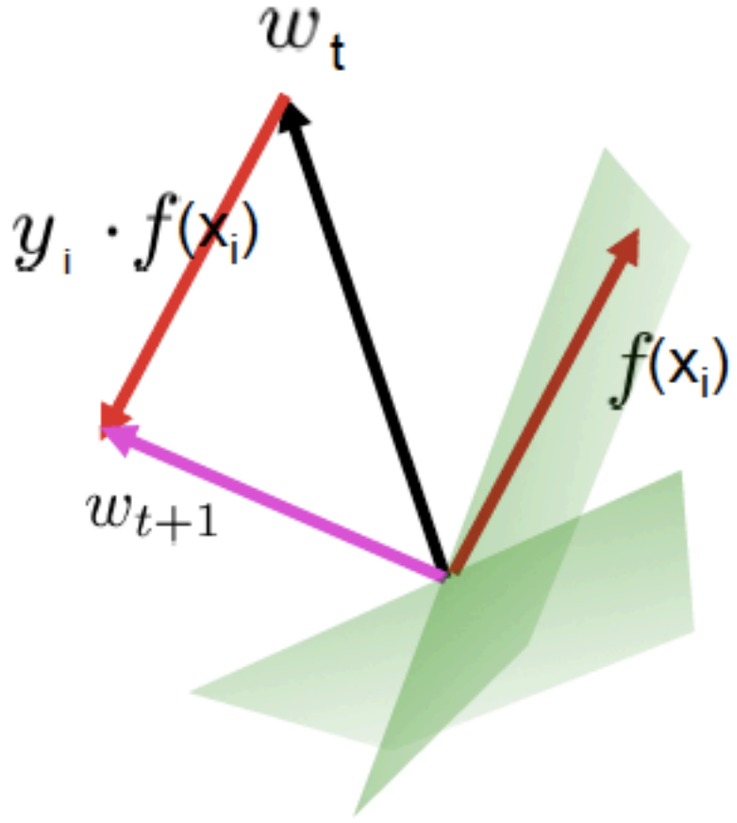
```
BIAS   : -3
free   : 4
money  :  2
...
```



The Perception algorithm

- Start with weight vector = $\vec{0}$
- For each training instance (x_i, y_i):
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x_i) \geq 0 \\ -1 & \text{if } w \cdot f(x_i) < 0 \end{cases}$$



  - If correct (i.e., $y=y_i$), no change!
  - If wrong: update

$$w = w + y_i \, f(x_i)$$

The Perceptron Algorithm:

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1. Also let's automatically scale all examples $\mathbf{x}$ to have (Euclidean) length 1, since this doesn't affect which side of the plane they are on.

2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.

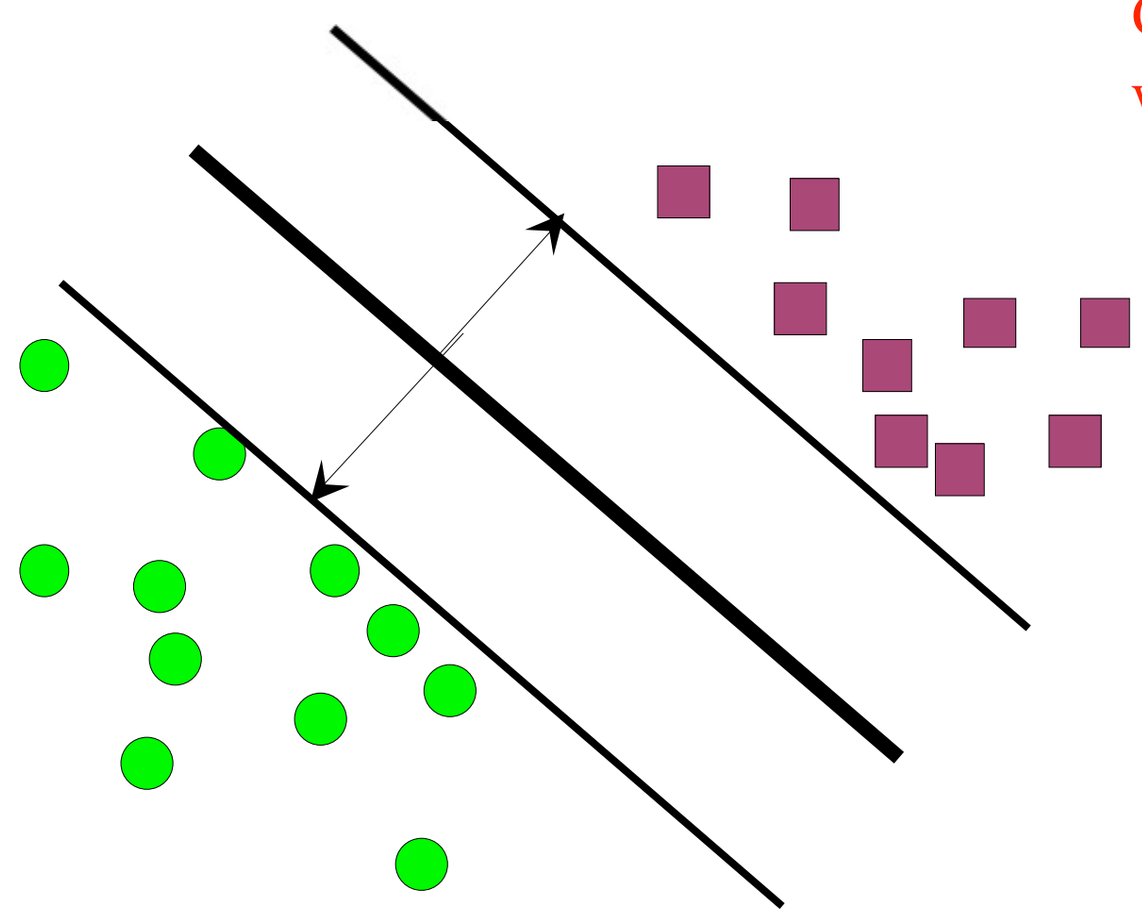
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

Linearly Separable

$\exists \mathbf{w}$ such that $\forall t$ $\quad y_t(\mathbf{w} \cdot \mathbf{x}_t) \geq \gamma > 0$

Called the *functional margin* with respect to the training set



Equivalently, for $y_t = +1$,

$$w \cdot x_t \geq \gamma$$

and for $y_t = -1$,

$$w \cdot x_t \leq -\gamma$$

Mistake Bound for Perceptron

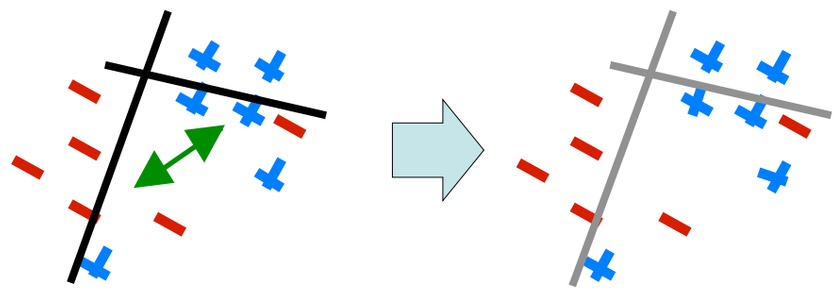
- Assume the data set *D* is linearly separable with *geometric* margin γ , i.e.,

$$\exists w^* \text{ s.t. } \|w^*\|_2 = 1 \text{ and } \forall t,\ y_t(w^* \cdot x_t) \geq \gamma$$

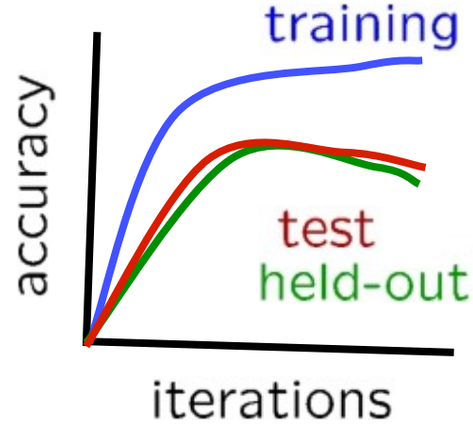
- Assume $\|x_t\|_2 \leq R,\ \forall t$
- Theorem: The maximum number of mistakes made by the perceptron algorithm is bounded by R^2/γ^2

Problems with the perceptron algorithm

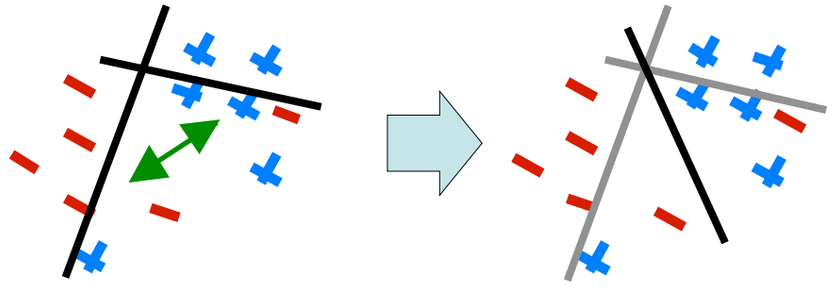
- If the data isn't linearly separable, no guarantees of convergence or training accuracy



- Even if the training data is linearly separable, perceptron can overfit



- **Averaged** perceptron is an algorithmic modification that helps with both issues
  - Averages the weight vectors across all iterations



ML Methodology

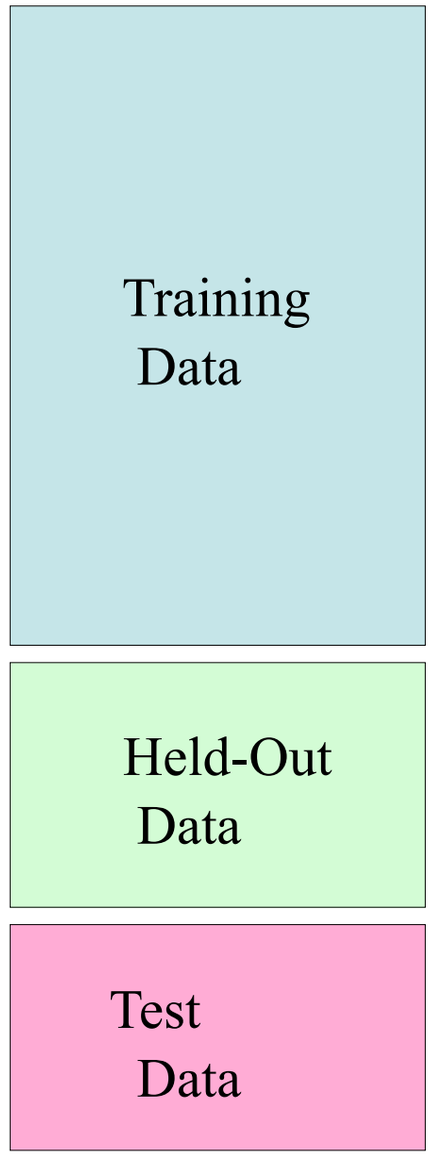
- Data: labeled instances, e.g. emails marked spam/ham
  - Training set
  - Held out set (sometimes call Validation set)
  - Test set

  Randomly allocate to these three, e.g. 60/20/20

- Features: attribute-value pairs which characterize each x

- Experimentation cycle
  - Select a hypothesis *f*
    (Tune hyper parameters on held-out or *validation* set)
  - Compute accuracy of test set
  - Very important: never " peek" at the test set!

- Evaluation
  - Accuracy: fraction of instances predicted correctly



Multi-Layer Perceptron (MLP)

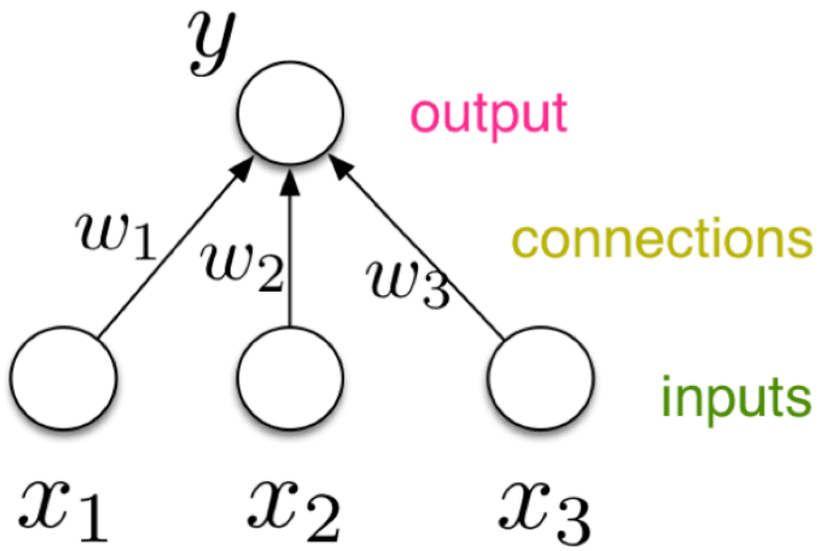
What are neural networks?

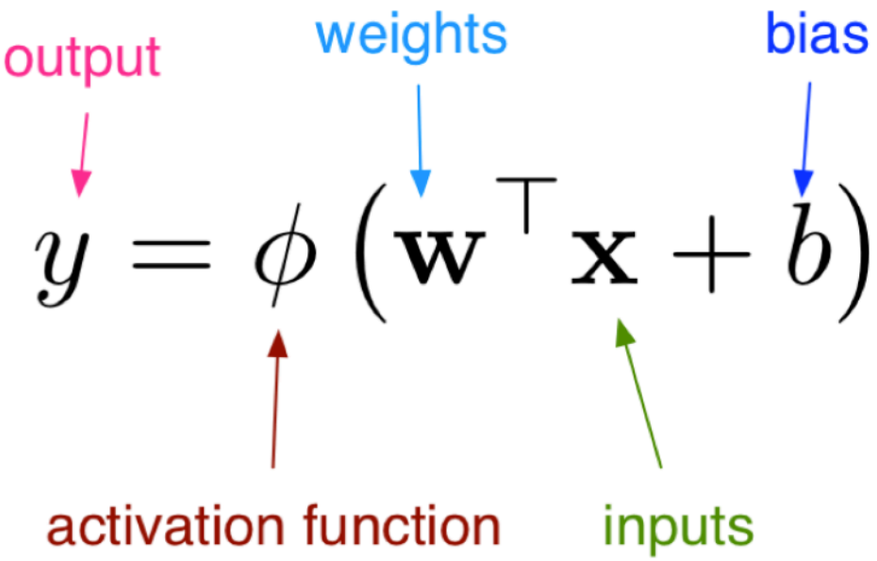
Why neural nets?

- inspiration from the brain
  - proof of concept that a neural architecture can see and hear!
- very effective across a range of applications (vision, text, speech, medicine, robotics, etc.)
- widely used in both academia and the tech industry
- powerful software frameworks (PyTorch, TensorFlow, etc.) let us quickly implement sophisticated algorithms

What are neural networks?

- Most of the biological details aren't essential, so we use vastly simplified models of neurons.
- While neural nets originally drew inspiration from the brain, nowadays we mostly think about math, statistics, etc.



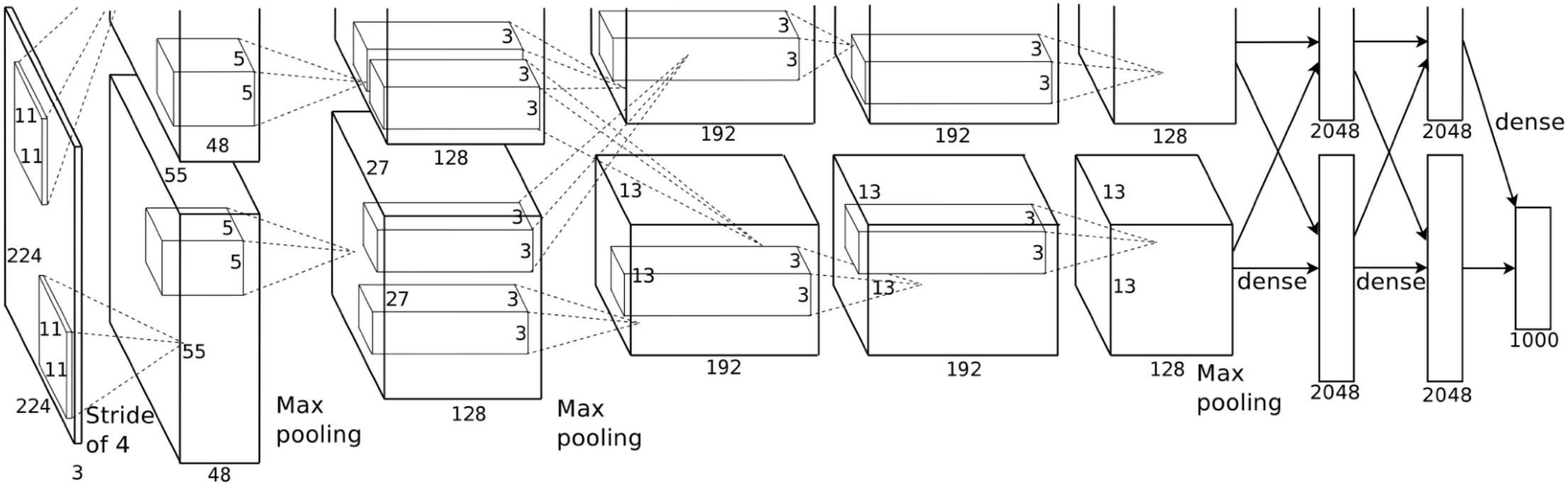


- Neural networks are collections of thousands (or millions) of these simple processing units that together perform useful computations.

“Deep learning”

Deep learning: many layers (stages) of processing

E.g. this network which recognizes objects in images:

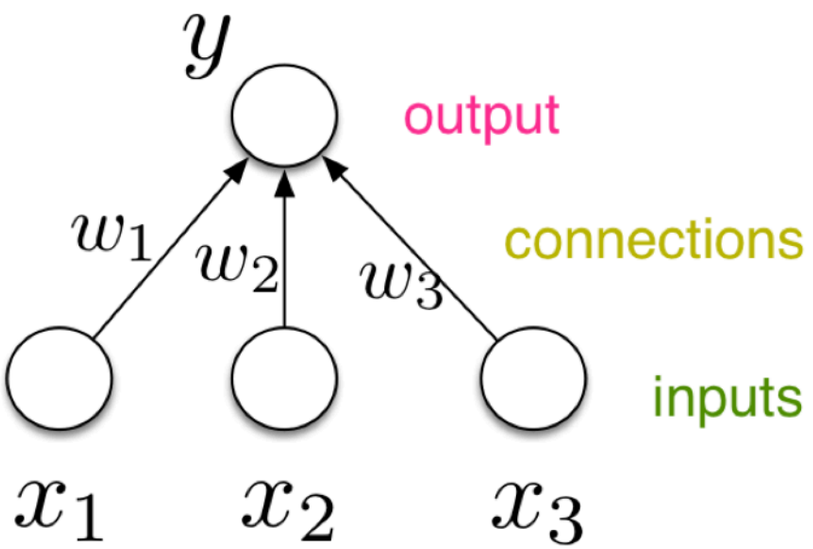


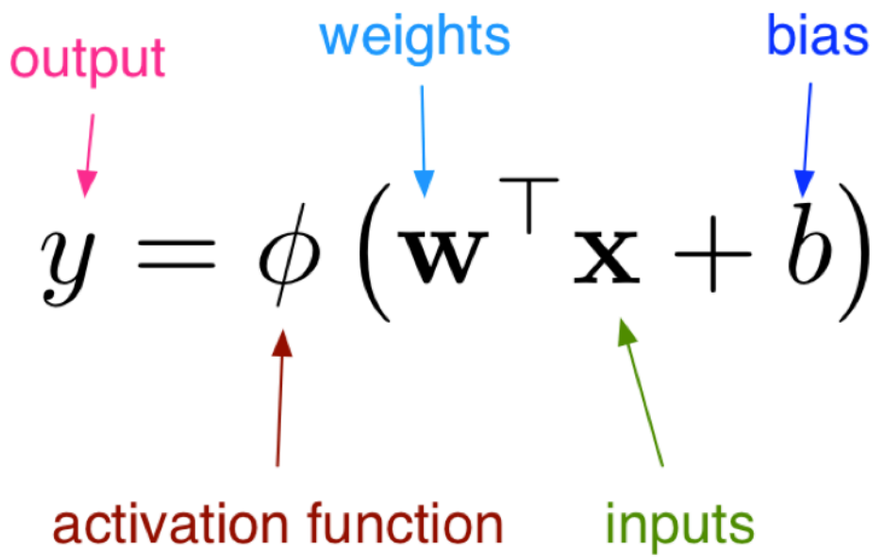
(Krizhevsky et al., 2012)

Each of the boxes consists of many neuron-like units similar to the one on the previous slide!

Overview

- Recall the simple neuron-like unit:

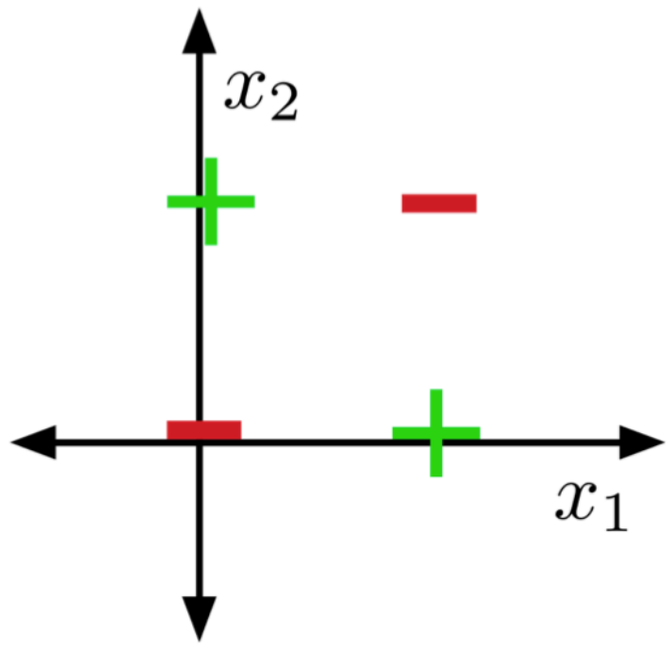




- Linear regression and logistic regression can each be viewed as a single unit.
- These units are much more powerful if we connect many of them into a neural network.

Limits of Linear Classification

- Single neurons (linear classifiers) are very limited in expressive power.
- **XOR** is a classic example of a function that's not linearly separable.

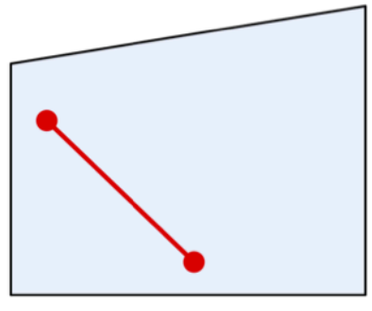


X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

- There's an elegant proof using convexity.

Limits of Linear Classification

Convex Sets



- A set $\mathcal{S}$ is convex if any line segment connecting points in $\mathcal{S}$ lies entirely within $\mathcal{S}$. Mathematically,

$$\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{S} \implies \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in \mathcal{S} \quad \text{for } 0 \leq \lambda \leq 1.$$

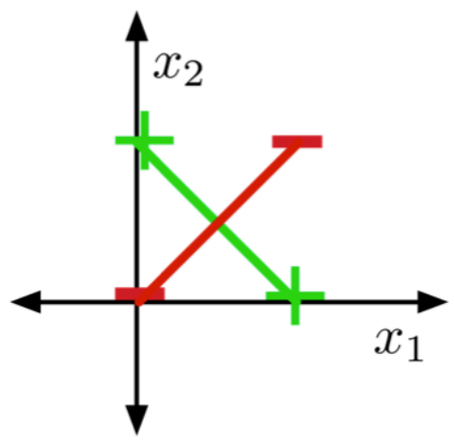
- A simple inductive argument shows that for $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{S}$, weighted averages, or convex combinations, lie within the set:

$$\lambda_1 \mathbf{x}_1 + \cdots + \lambda_N \mathbf{x}_N \in \mathcal{S} \quad \text{for } \lambda_i > 0, \ \lambda_1 + \cdots \lambda_N = 1.$$

Limits of Linear Classification

Showing that XOR is not linearly separable

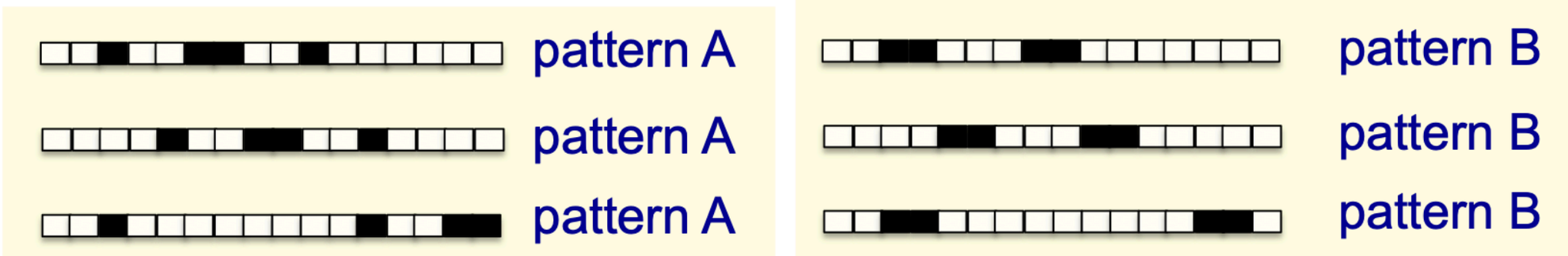
- Half-spaces are obviously convex.
- Suppose there were some feasible hypothesis. If the positive examples are in the positive half-space, then the green line segment must be as well.
- Similarly, the red line segment must line within the negative half-space.



- But the intersection can't lie in both half-spaces. Contradiction!

Limits of Linear Classification

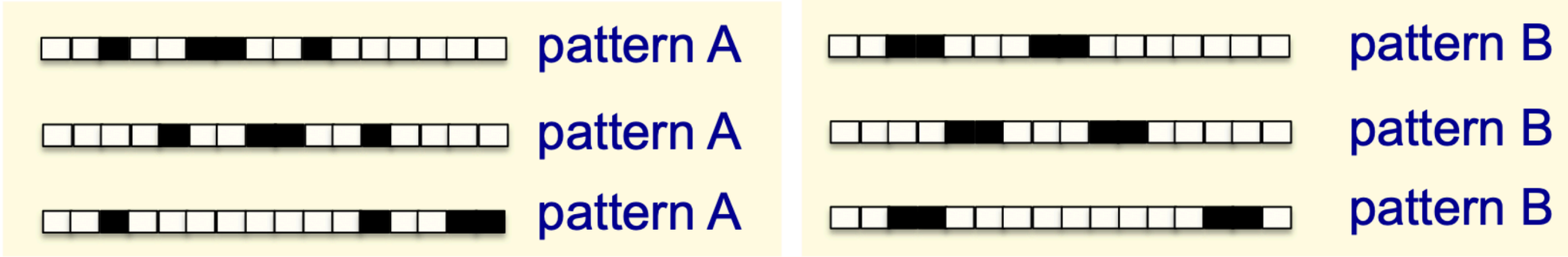
A more troubling example



- These images represent 16-dimensional vectors. White = 0, black = 1.
- Want to distinguish patterns A and B in all possible translations (with wrap-around)
- Translation invariance is commonly desired in vision!

Limits of Linear Classification

A more troubling example



- These images represent 16-dimensional vectors. White = 0, black = 1.
- Want to distinguish patterns A and B in all possible translations (with wrap-around)
- Translation invariance is commonly desired in vision!
- Suppose there's a feasible solution. The average of all translations of A is the vector $(0.25, 0.25, \ldots, 0.25)$. Therefore, this point must be classified as A.
- Similarly, the average of all translations of B is also $(0.25, 0.25, \ldots, 0.25)$. Therefore, it must be classified as B. Contradiction!

Limits of Linear Classification

- Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for **XOR**:

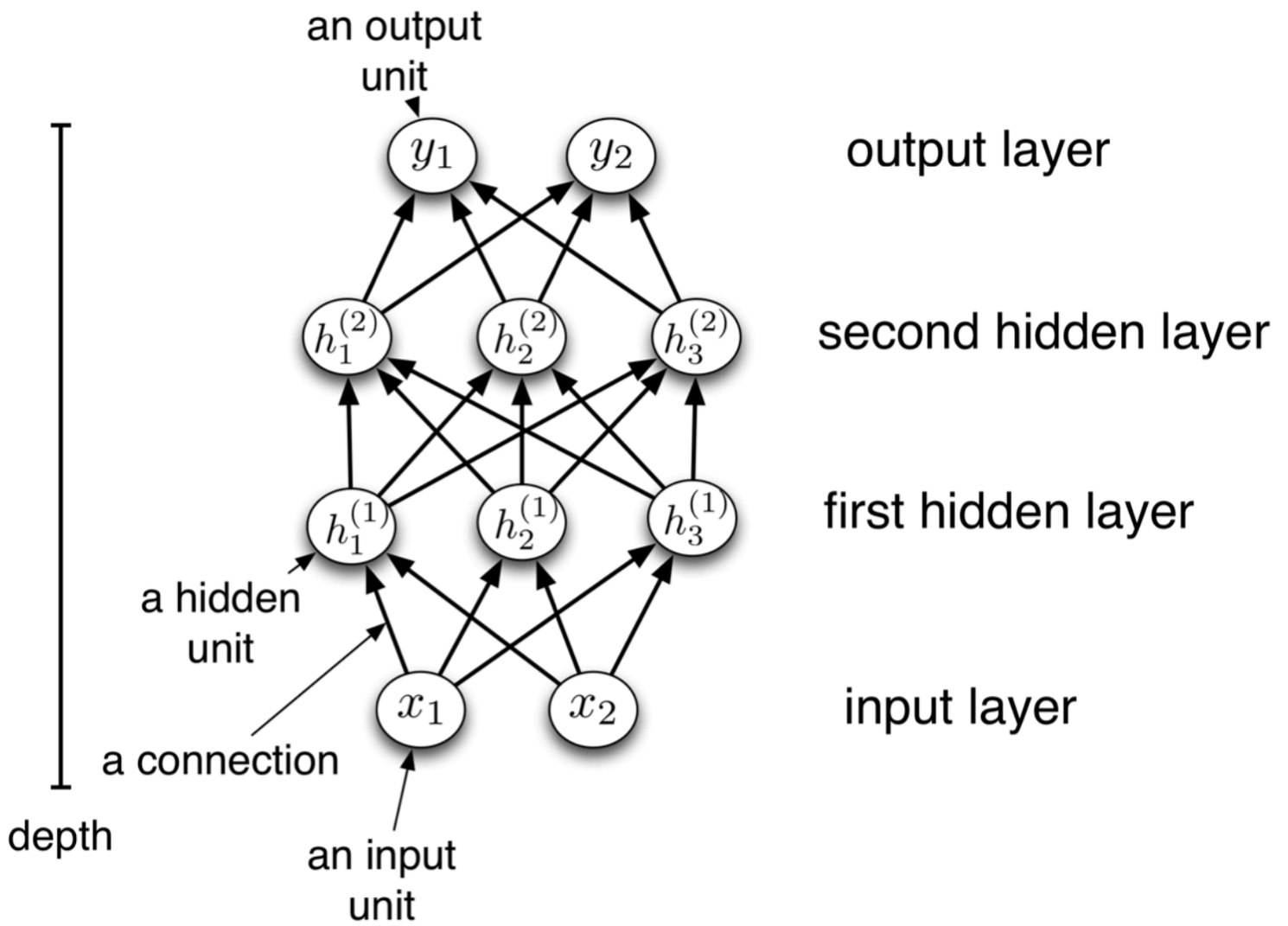
$$\boldsymbol{\psi}(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}$$

x_1	x_2	$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$	$\phi_3(\mathbf{x})$	t
0	0	0	0	0	0
0	1	0	1	0	1
1	0	1	0	0	1
1	1	1	1	1	0

- This is linearly separable. (Try it!)
- Not a general solution: it can be hard to pick good basis functions. Instead, we'll use neural nets to learn nonlinear hypotheses directly.

Multilayer Perceptrons

- We can connect lots of units together into a directed acyclic graph.
- This gives a feed-forward neural network. That's in contrast to recurrent neural networks, which can have cycles. (We'll talk about those later.)
- Typically, units are grouped together into layers.



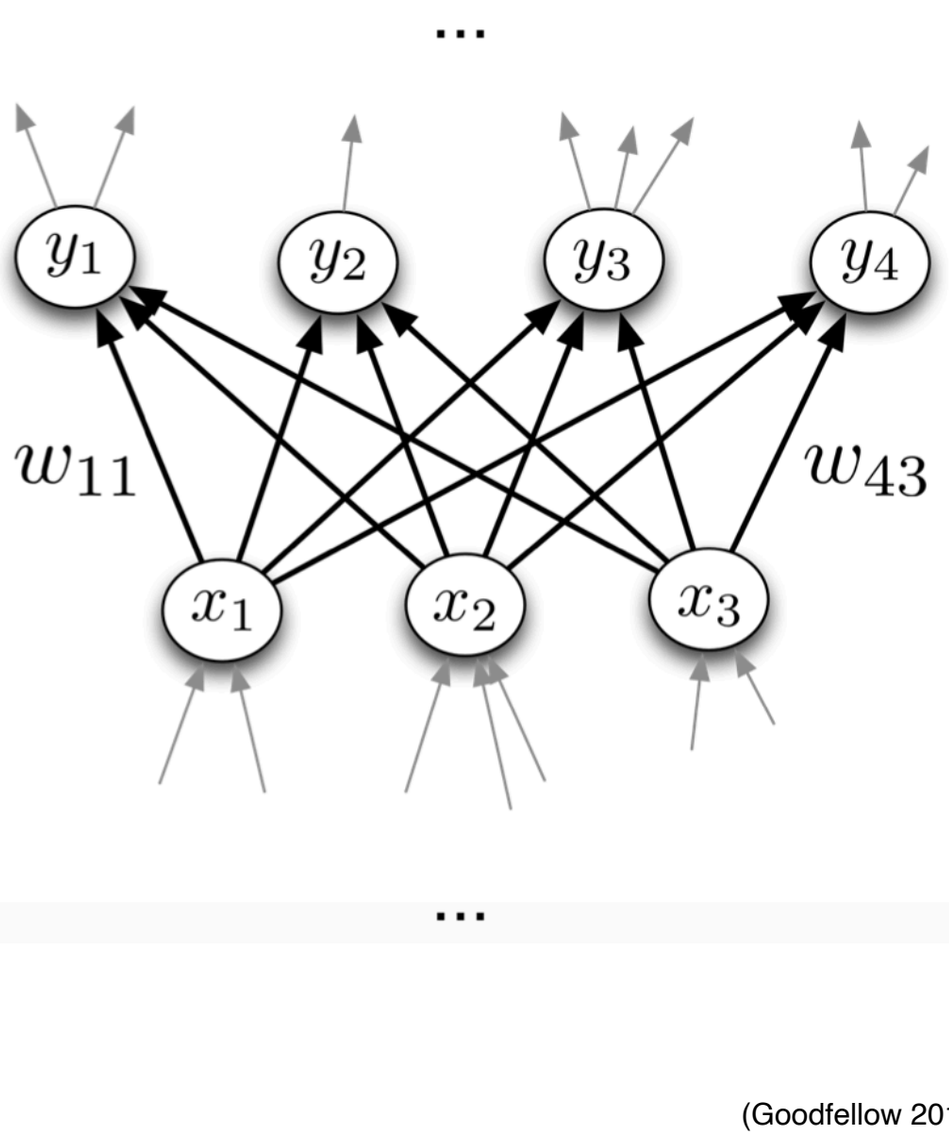
Multilayer Perceptrons

- Each layer connects N input units to M output units.
- In the simplest case, all input units are connected to all output units. We call this a fully connected layer. We'll consider other layer types later.
- Note: the inputs and outputs for a layer are distinct from the inputs and outputs to the network.

- Recall from softmax regression: this means we need an $M \times N$ weight matrix.
- The output units are a function of the input units:

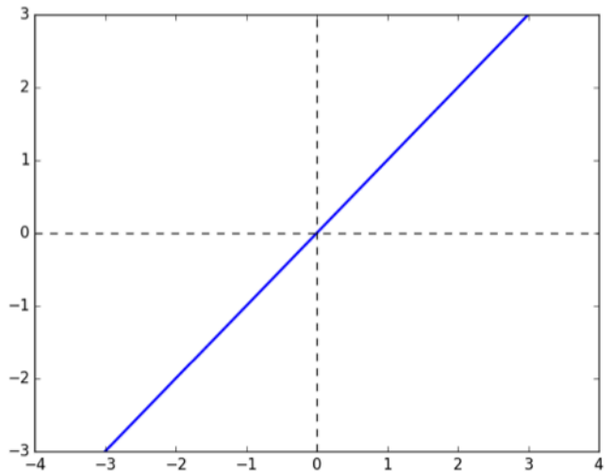
$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- A multilayer network consisting of fully connected layers is called a multilayer perceptron. Despite the name, it has nothing to do with perceptrons!



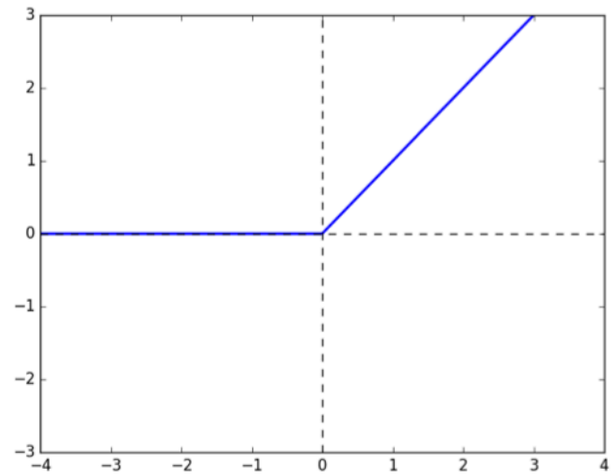
Multilayer Perceptrons

Some activation functions:



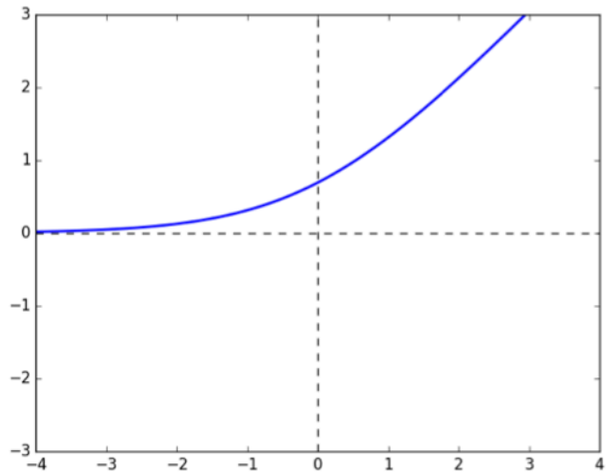
Linear

$$y = z$$



Rectified Linear Unit (ReLU)

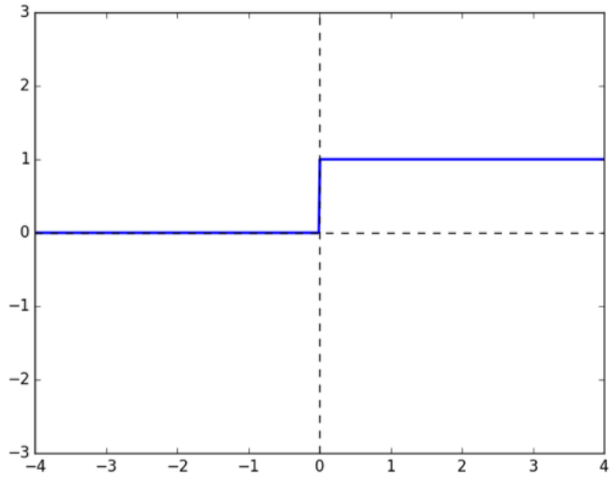
$$y = \max(0, z)$$



Soft ReLU

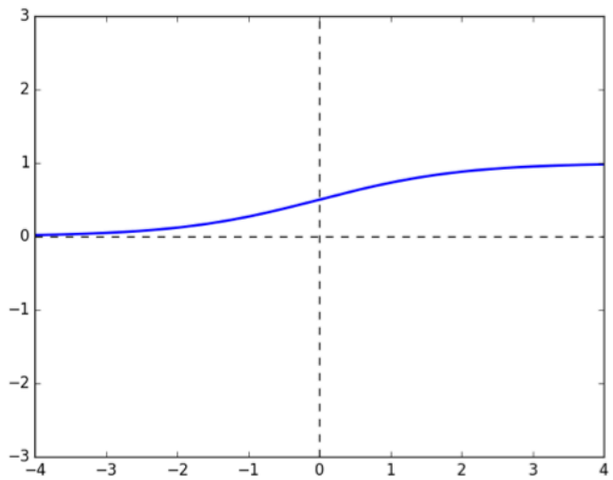
$$y = \log 1 + e^z$$

Some activation functions:



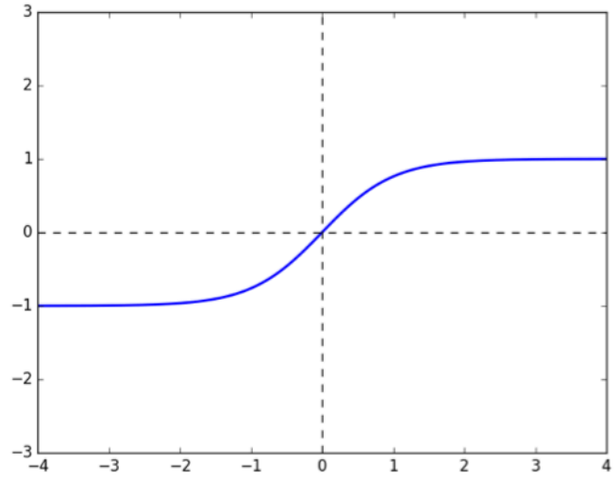
Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$y = \frac{1}{1 + e^{-z}}$$



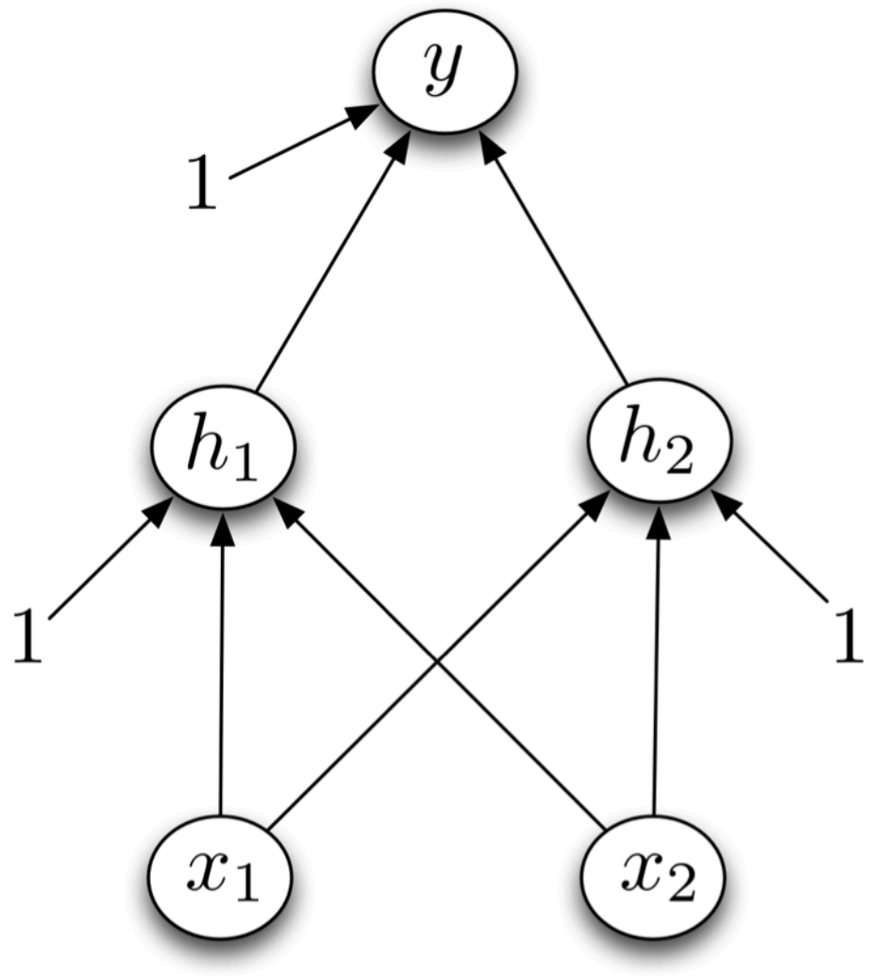
Hyperbolic Tangent (tanh)

$$y = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

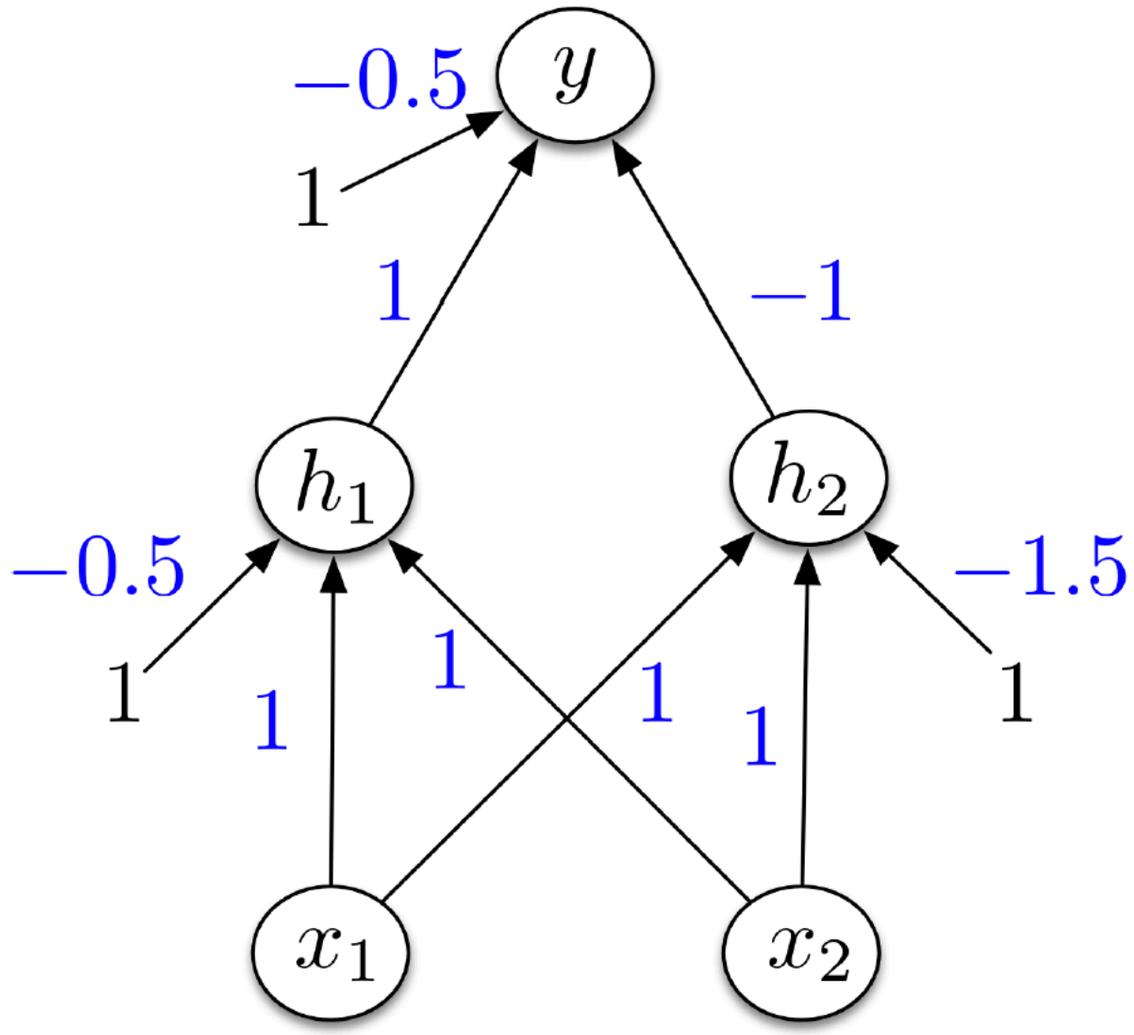
Multilayer Perceptrons

Designing a network to compute XOR:

Assume hard threshold activation function



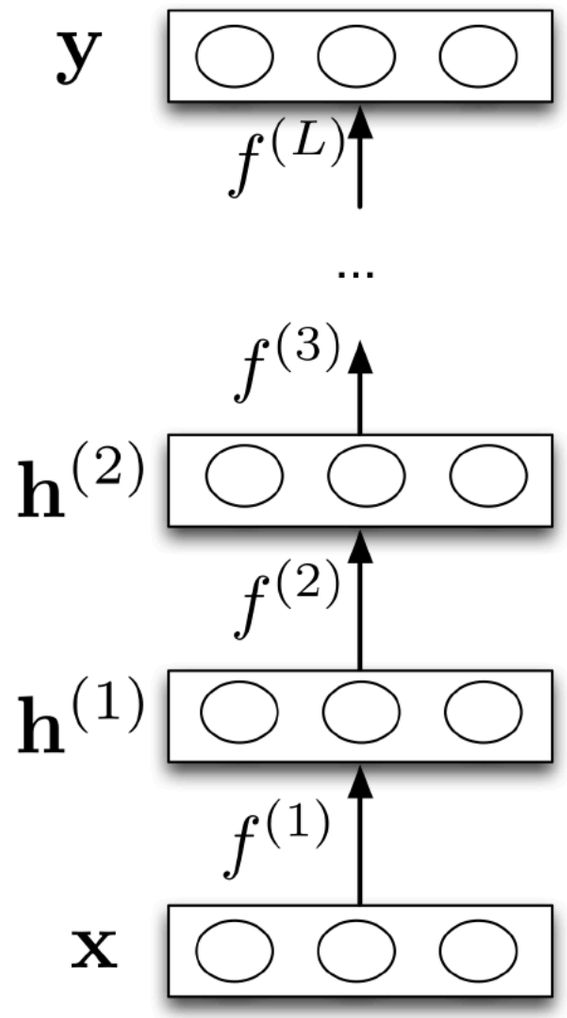
Multilayer Perceptrons



Multilayer Perceptrons

- Each layer computes a function, so the network computes a composition of functions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$



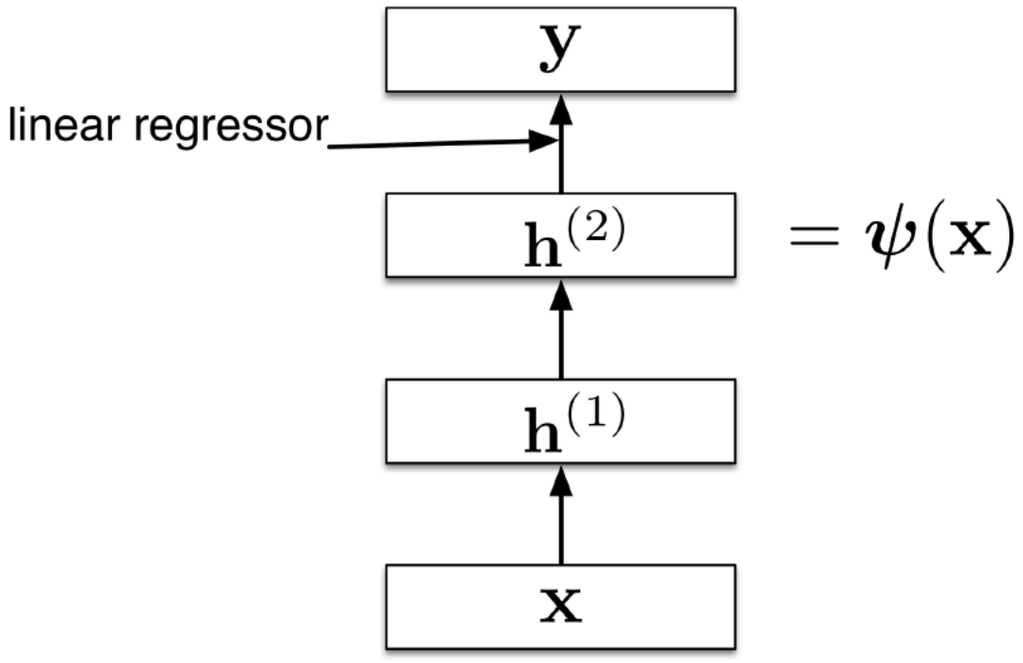
- Or more simply:

$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

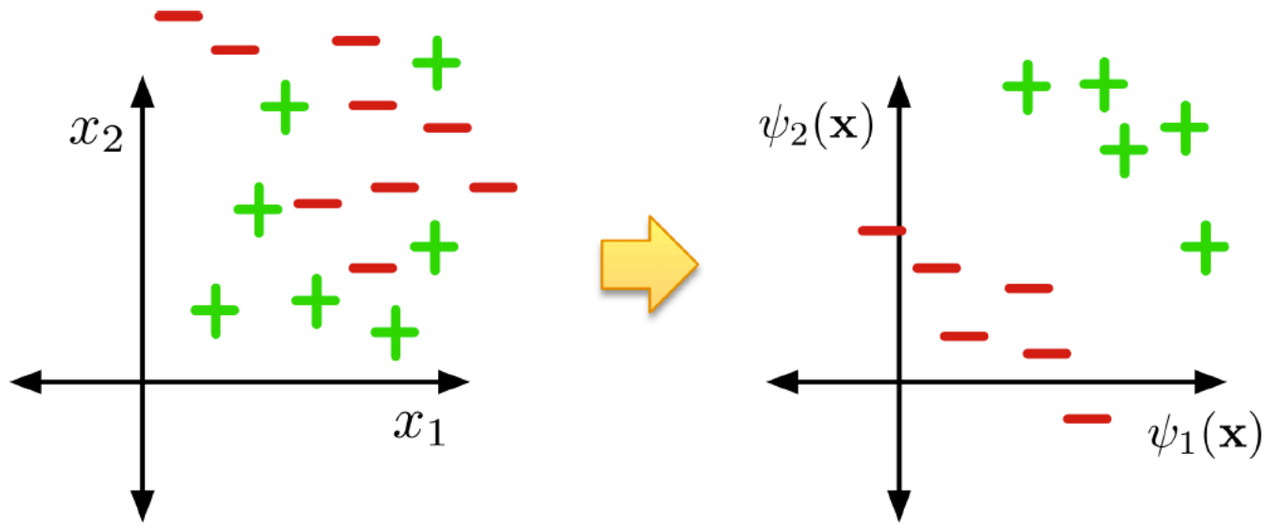
- Neural nets provide modularity: we can implement each layer's computations as a black box.

Feature Learning

- Neural nets can be viewed as a way of learning features:



- The goal:



Feature Learning

Input representation of a digit : 784 dimensional vector.

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	221.0	115.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	254.0	148.0	0.0	0.0	0.0	0.0	22.0	230.0	134.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	168.0	254.0	115.0	0.0	0.0	0.0	0.0	24.0	253.0	184.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	234.0	254.0	81.0	0.0	0.0	0.0	0.0	91.0	253.0	184.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.0	221.0	254.0	160.0	0.0	0.0	0.0	0.0	0.0	141.0	254.0	127.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	114.0	253.0	253.0	76.0	0.0	0.0	0.0	0.0	0.0	207.0	253.0	93.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	15.0	232.0	253.0	182.0	0.0	0.0	0.0	0.0	0.0	0.0	207.0	253.0	17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	229.0	253.0	202.0	19.0	0.0	0.0	0.0	0.0	0.0	34.0	240.0	253.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	7.0	170.0	254.0	254.0	46.0	0.0	0.0	0.0	0.0	0.0	0.0	47.0	254.0	254.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	24.0	253.0	254.0	253.0	234.0	163.0	47.0	47.0	26.0	0.0	0.0	130.0	253.0	253.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	21.0	246.0	254.0	253.0	253.0	253.0	254.0	253.0	232.0	174.0	208.0	232.0	253.0	177.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	84.0	161.0	211.0	219.0	219.0	254.0	253.0	253.0	253.0	254.0	253.0	244.0	69.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	93.0	142.0	142.0	93.0	170.0	254.0	230.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	153.0	253.0	213.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	170.0	253.0	137.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	220.0	253.0	137.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0	255.0	254.0	80.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	93.0	254.0	253.0	46.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	51.0	254.0	215.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	186.0	106.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

(Goodfellow 2016)

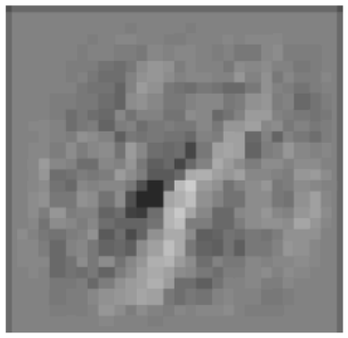
Feature Learning

Each first-layer hidden unit computes $\sigma(\mathbf{w}_i^T \mathbf{x})$

Here is one of the weight vectors (also called a feature).

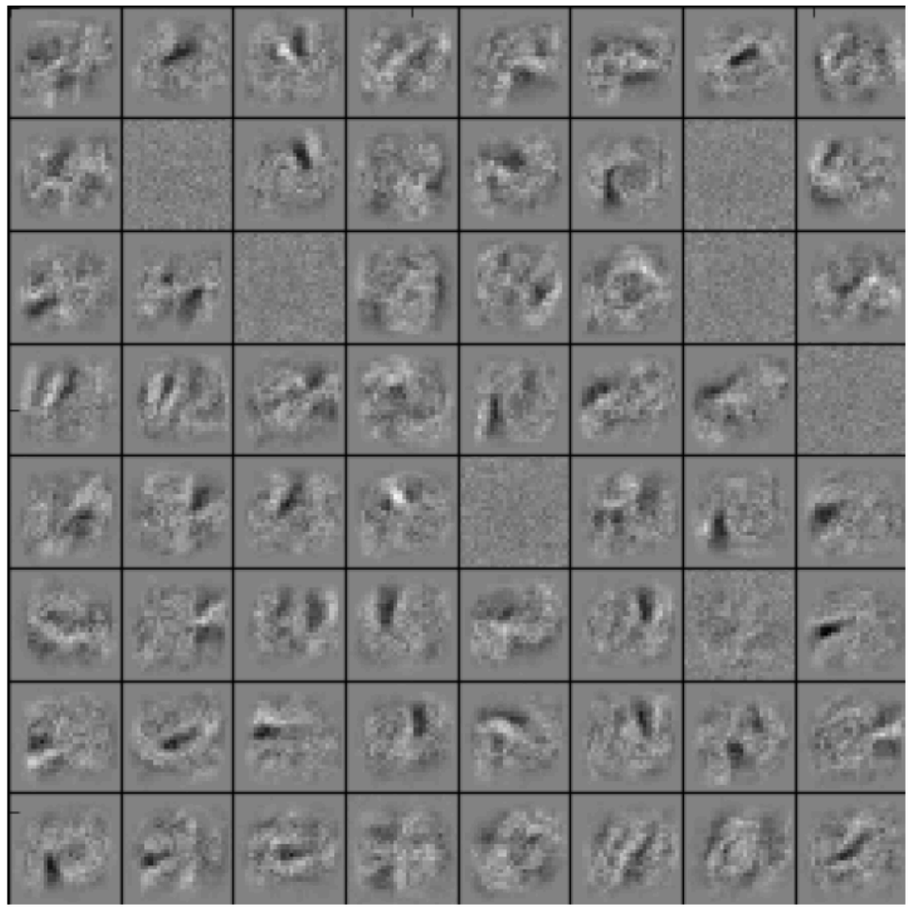
It's reshaped into an image, with gray = 0, white = +, black = -.

To compute $\mathbf{w}_i^T \mathbf{x}$, multiply the corresponding pixels, and sum the result.



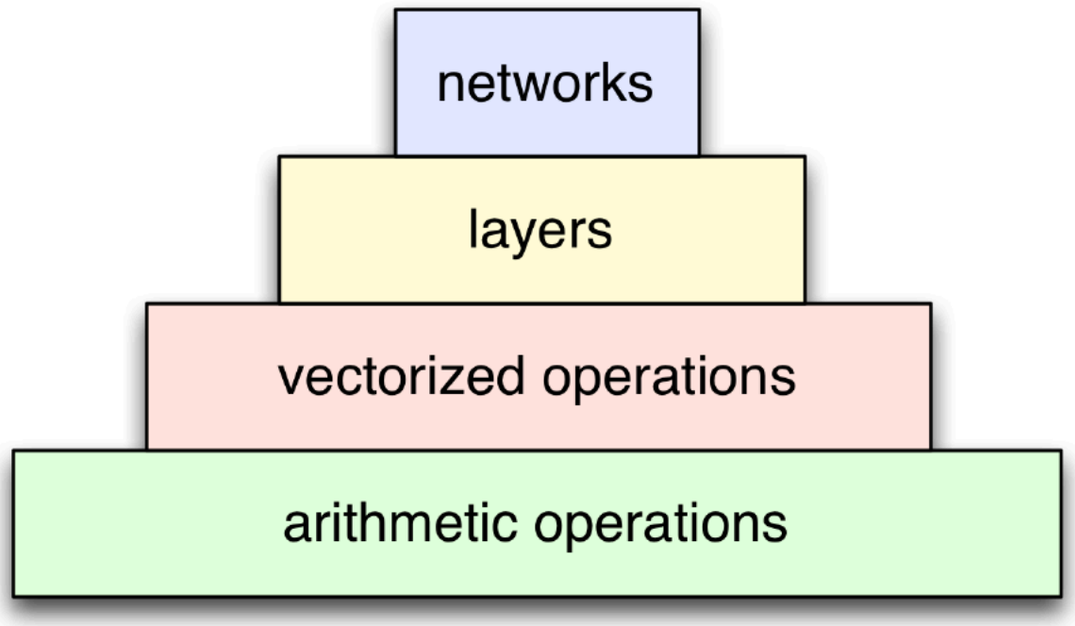
Feature Learning

There are 256 first-level features total. Here are some of them.



Levels of Abstraction

When you design neural networks and machine learning algorithms, you'll need to think at multiple levels of abstraction.



Expressive Power

- We've seen that there are some functions that linear classifiers can't represent. Are deep networks any better?
- Any sequence of *linear* layers can be equivalently represented with a single linear layer.

$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

- Deep linear networks are no more expressive than linear regression!
- Linear layers do have their uses — stay tuned!

Expressive Power

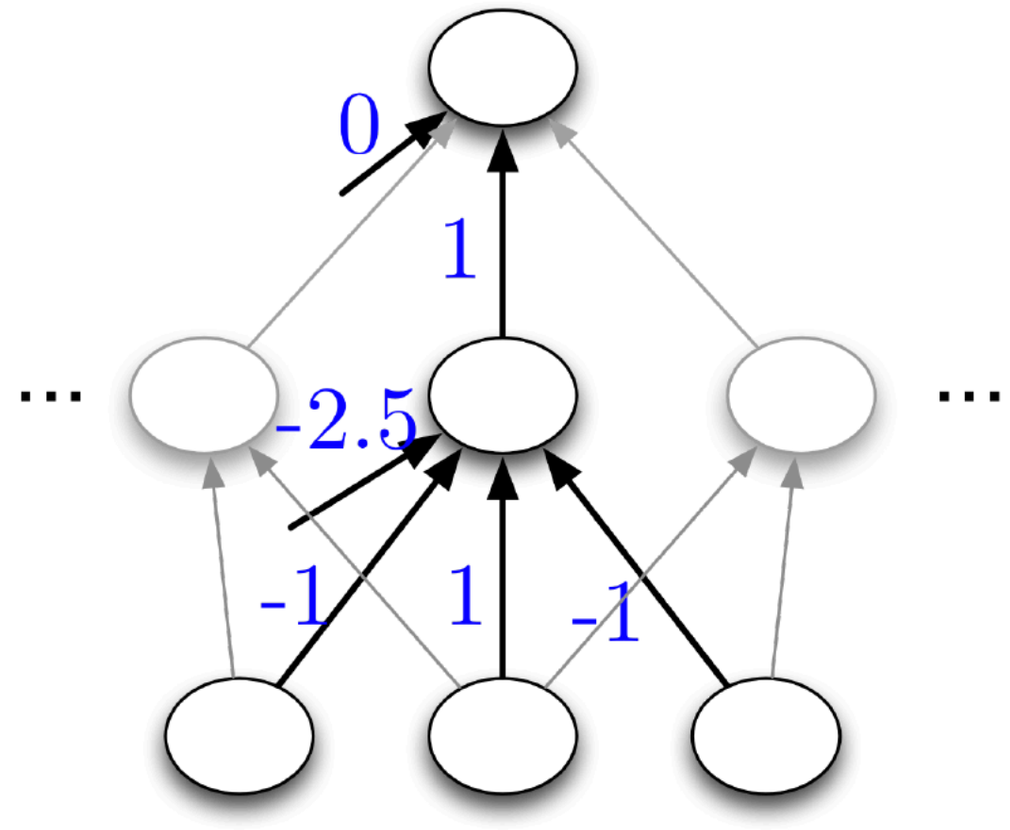
- Multilayer feed-forward neural nets with *nonlinear* activation functions are universal approximators: they can approximate any function arbitrarily well.
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though ReLU is "almost" linear, it's nonlinear enough!

Expressive Power

Universality for binary inputs and targets:

- Hard threshold hidden units, linear output
- Strategy: 2^D hidden units, each of which responds to one particular input configuration

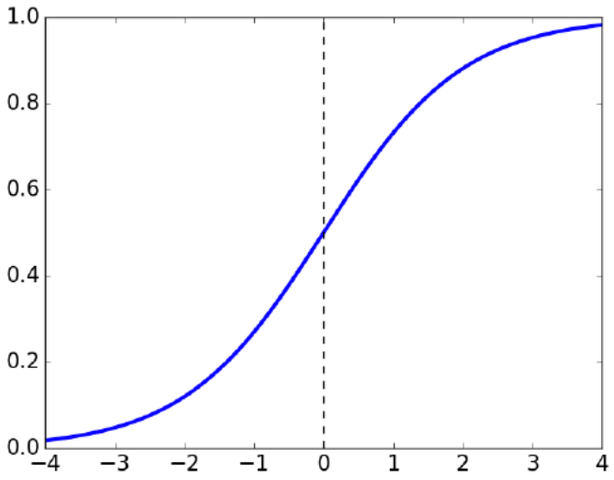
x_1	x_2	x_3	t
	$\vdots$		$\vdots$
-1	-1	1	-1
-1	1	-1	1
-1	1	1	1
	$\vdots$		$\vdots$



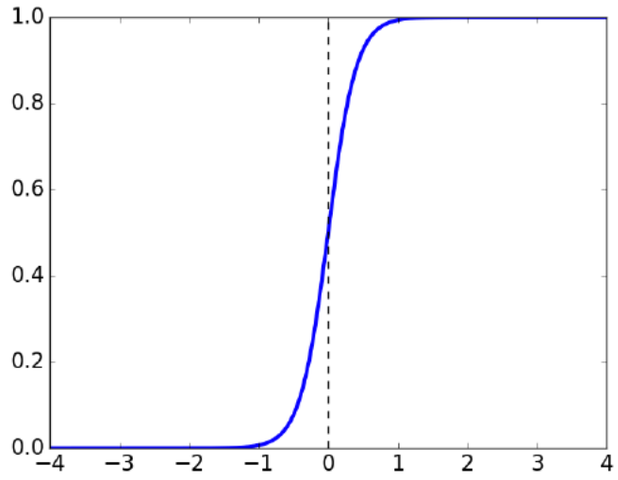
- Only requires one hidden layer, though it needs to be extremely wide!

Expressive Power

- What about the logistic activation function?
- You can approximate a hard threshold by scaling up the weights and biases:



$y = \sigma(x)$



$y = \sigma(5x)$

- This is good: logistic units are differentiable, so we can tune them with gradient descent. (Stay tuned!)

Expressive Power

- Limits of universality
  - You may need to represent an exponentially large network.
  - If you can learn any function, you'll just overfit.
  - Really, we desire a *compact* representation!
- We've derived units which compute the functions AND, OR, and NOT. Therefore, any Boolean circuit can be translated into a feed-forward neural net.
  - This suggests you might be able to learn *compact* representations of some complicated functions

Backpropagation

Overview

- We've seen that multilayer neural networks are powerful. But how can we actually learn them?
- Backpropagation is the central algorithm in this course.
  - It's is an algorithm for computing gradients.
  - Really it's an instance of reverse mode automatic differentiation, which is much more broadly applicable than just neural nets.
    - This is "just" a clever and efficient use of the Chain Rule for derivatives.

Univariate Chain Rule

- We've already been using the univariate Chain Rule.
- Recall: if $f(x)$ and $x(t)$ are univariate functions, then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \frac{\mathrm{d}x}{\mathrm{d}t}.$$

Univariate Chain Rule

Recall: Univariate logistic least squares model

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Let's compute the loss derivatives.

Univariate Chain Rule

How you would have done it in calculus class

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right]$$
$$= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2$$
$$= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t)$$
$$= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b)$$
$$= (\sigma(wx+b)-t)\sigma'(wx+b)x$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right]$$
$$= \frac{1}{2}\frac{\partial}{\partial b}(\sigma(wx+b)-t)^2$$
$$= (\sigma(wx+b)-t)\frac{\partial}{\partial b}(\sigma(wx+b)-t)$$
$$= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial b}(wx+b)$$
$$= (\sigma(wx+b)-t)\sigma'(wx+b)$$

What are the disadvantages of this approach?

Univariate Chain Rule

A more structured way to do it

Computing the loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

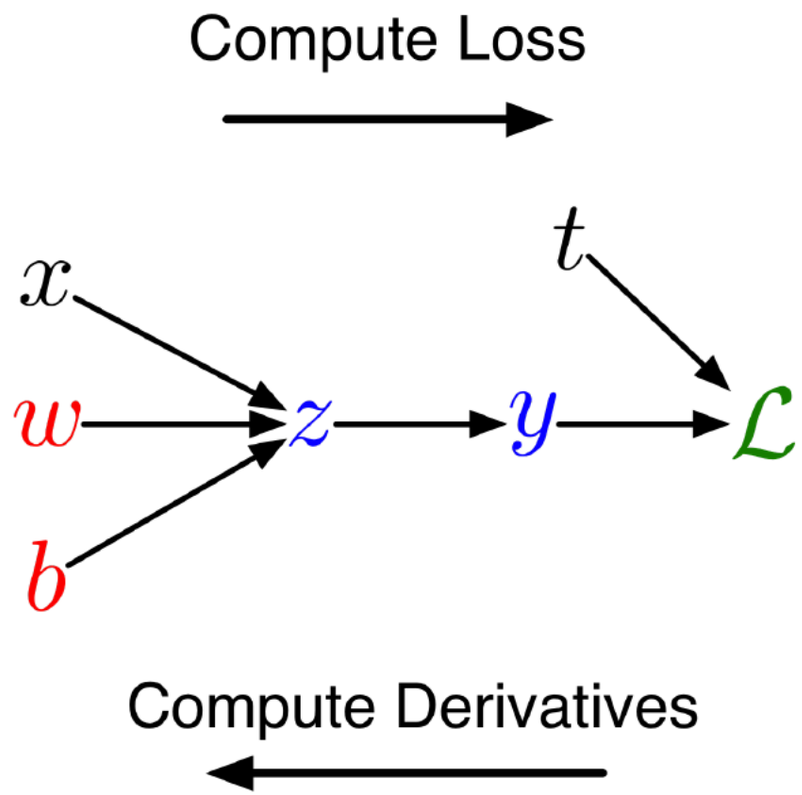
Computing the derivatives:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\,\sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\,x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

Univariate Chain Rule

- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



Univariate Chain Rule

A slightly more convenient notation:

- Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, sometimes called the error signal.
- This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).
- This is not a standard notation, but I couldn't find another one that I liked.

Computing the loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

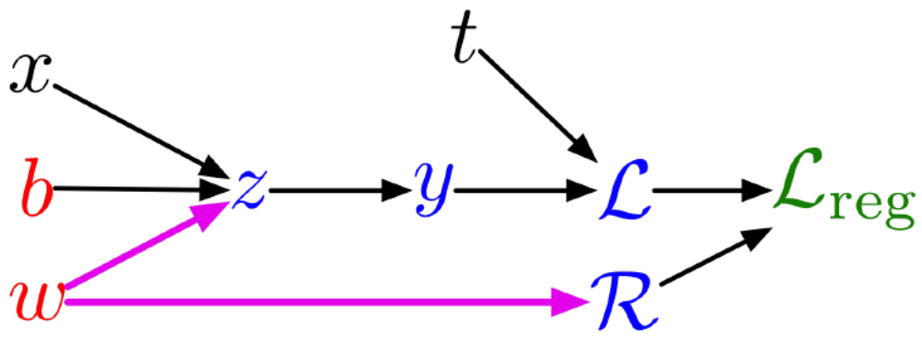
Computing the derivatives:

$$\overline{y} = y - t$$
$$\overline{z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\,x$$
$$\overline{b} = \overline{z}$$

Multivariate Chain Rule

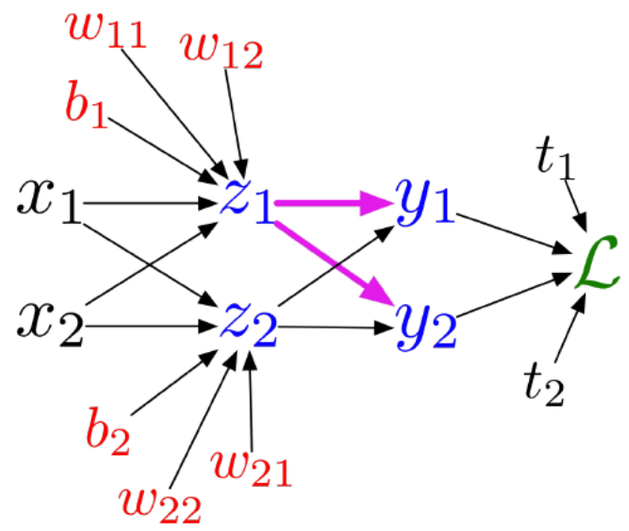
Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!

L_2-Regularized regression



$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Multiclass logistic regression

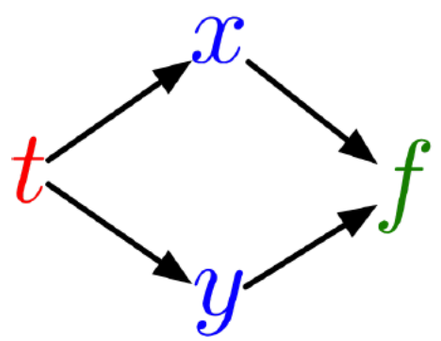


$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

Multivariate Chain Rule

- Suppose we have a function $f(x, y)$ and functions $x(t)$ and $y(t)$. (All the variables here are scalar-valued.) Then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$



- Example:

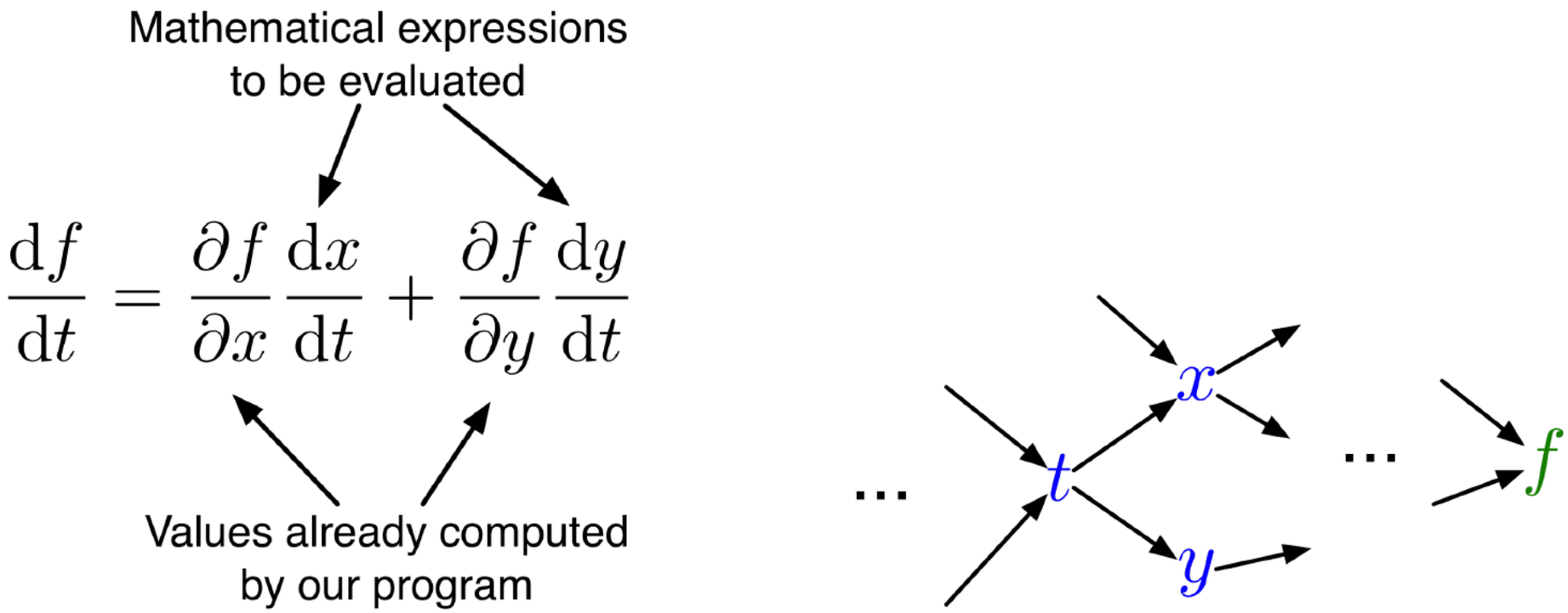
$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

- Plug in to Chain Rule:

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$
$$= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t$$

Multivariable Chain Rule

- In the context of backpropagation:



- In our notation:

$$\overline{t} = \overline{x}\frac{\mathrm{d}x}{\mathrm{d}t} + \overline{y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Backpropagation

Full backpropagation algorithm:

Let $v_1, \ldots, v_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

v_N denotes the variable we're trying to compute derivatives of (e.g. loss).

forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

backward pass

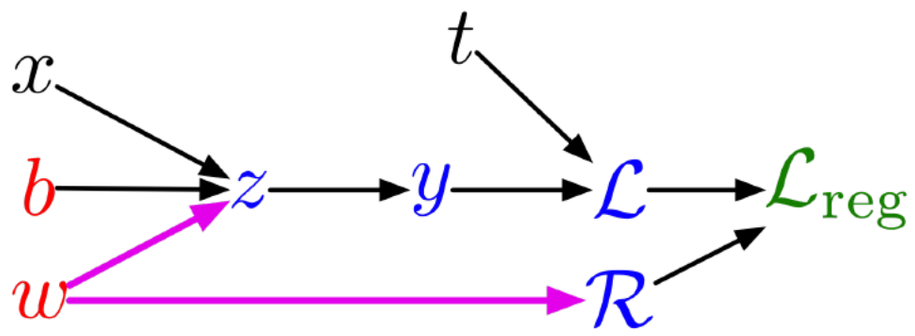
$\overline{v_N} = 1$

For $i = N-1, \ldots, 1$

$$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$$

Backpropagation

Example: univariate logistic least squares regression



Forward pass:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$

$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\lambda$$

$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}$$

$$\overline{y} = \overline{\mathcal{L}}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = \overline{\mathcal{L}}\,(y - t)$$

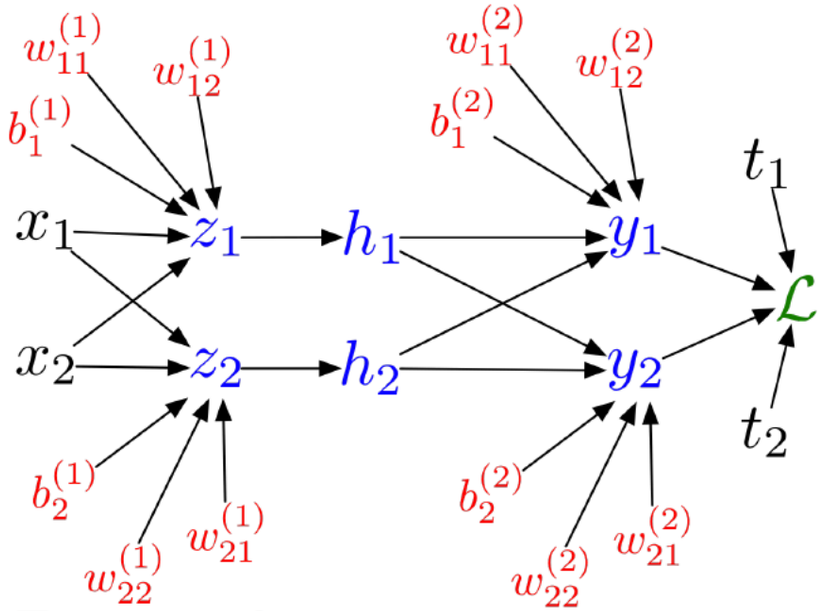
$$\overline{z} = \overline{y}\frac{\mathrm{d}y}{\mathrm{d}z} = \overline{y}\,\sigma'(z)$$

$$\overline{w} = \overline{z}\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} = \overline{z}\,x + \overline{\mathcal{R}}\,w$$

$$\overline{b} = \overline{z}\frac{\partial z}{\partial b} = \overline{z}$$

Backpropagation

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}} \, (y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k} \, h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

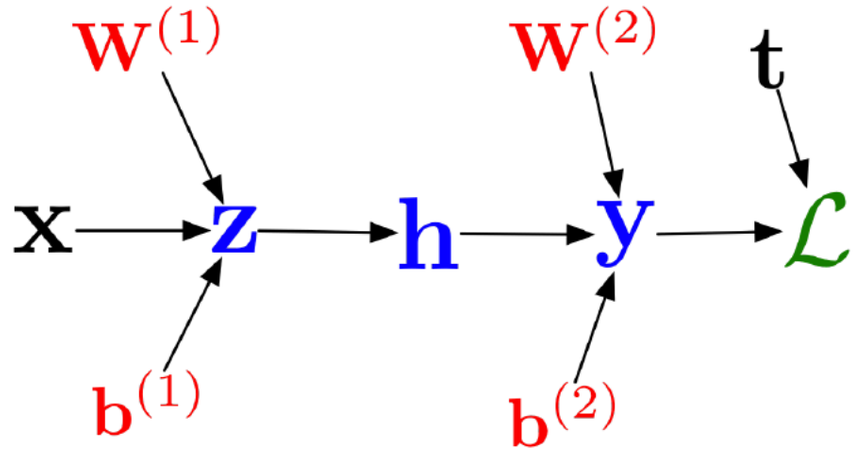
$$\overline{z_i} = \overline{h_i} \, \sigma'(z_i)$$

$$\overline{w_{ij}^{(1)}} = \overline{z_i} \, x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

Vector Form

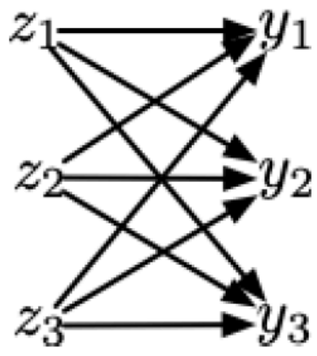
- Computation graphs showing individual units are cumbersome.
- As you might have guessed, we typically draw graphs over the vectorized variables.



- We pass messages back analogous to the ones for scalar-valued nodes.

Vector Form

- Consider this computation graph:



- Backprop rules:

$$\overline{z_j} = \sum_k \overline{y_k} \frac{\partial y_k}{\partial z_j} \qquad \overline{\mathbf{z}} = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{z}}\right)^\top \overline{\mathbf{y}},$$

where $\partial \mathbf{y} / \partial \mathbf{z}$ is the Jacobian matrix:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{z}} = \begin{pmatrix} \frac{\partial y_1}{\partial z_1} & \cdots & \frac{\partial y_1}{\partial z_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial z_1} & \cdots & \frac{\partial y_m}{\partial z_n} \end{pmatrix}$$

Vector Form

Examples

- Matrix-vector product

$$\mathbf{z} = \mathbf{W}\mathbf{x} \qquad \frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \mathbf{W} \qquad \overline{\mathbf{x}} = \mathbf{W}^\top \overline{\mathbf{z}}$$

- Elementwise operations

$$\mathbf{y} = \exp(\mathbf{z}) \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{z}} = \begin{pmatrix} \exp(z_1) & & 0 \\ & \ddots & \\ 0 & & \exp(z_D) \end{pmatrix} \qquad \overline{\mathbf{z}} = \exp(\mathbf{z}) \circ \overline{\mathbf{y}}$$

- Note: we never explicitly construct the Jacobian. It's usually simpler and more efficient to compute the VJP directly.

Vector Form

Full backpropagation algorithm (vector form):

Let $\mathbf{v}_1, \ldots, \mathbf{v}_N$ be a topological ordering of the computation graph (i.e. parents come before children.)

$\mathbf{v}_N$ denotes the variable we're trying to compute derivatives of (e.g. loss).

It's a scalar, which we can treat as a 1-D vector.

forward pass

For $i = 1, \ldots, N$

Compute $\mathbf{v}_i$ as a function of $\text{Pa}(\mathbf{v}_i)$

backward pass

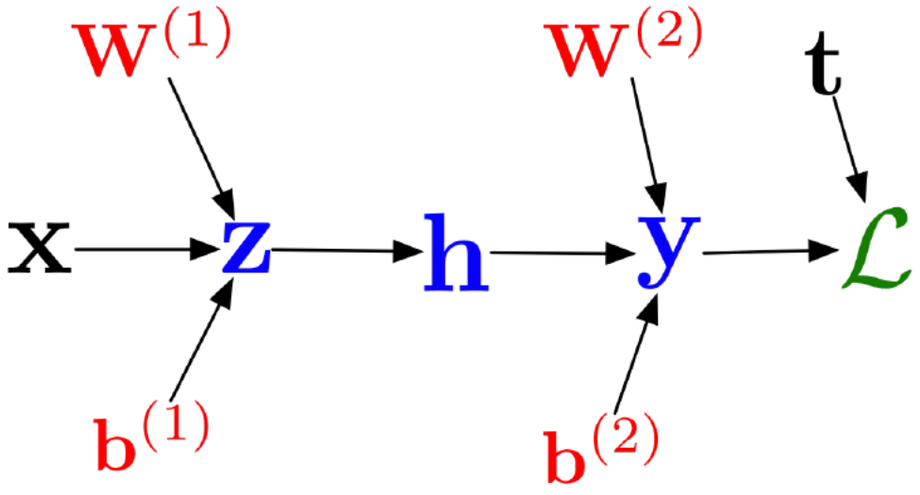
$\overline{\mathbf{v}_N} = 1$

For $i = N - 1, \ldots, 1$

$$\overline{\mathbf{v}_i} = \sum_{j \in \text{Ch}(\mathbf{v}_i)} \frac{\partial \mathbf{v}_j}{\partial \mathbf{v}_i}^\top \overline{\mathbf{v}_j}$$

Vector Form

MLP example in vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$

$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$

$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$

$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$

$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$

$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$

$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Computational Cost

- Computational cost of forward pass: one add-multiply operation per weight

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

- Computational cost of backward pass: two add-multiply operations per weight

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

- Rule of thumb: the backward pass is about as expensive as two forward passes.
- For a multilayer perceptron, this means the cost is linear in the number of layers, quadratic in the number of units per layer.

Closing Thoughts

- Backprop is used to train the overwhelming majority of neural nets today.
  - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.
- Despite its practical success, backprop is believed to be neurally implausible.
  - No evidence for biological signals analogous to error derivatives.
  - All the biologically plausible alternatives we know about learn much more slowly (on computers).
  - So how on earth does the brain learn?

Closing Thoughts

- By now, we've seen three different ways of looking at gradients:
  - **Geometric:** visualization of gradient in weight space
  - **Algebraic:** mechanics of computing the derivatives
  - **Implementational:** efficient implementation on the computer
- When thinking about neural nets, it's important to be able to shift between these different perspectives!