

Statistical Learning and Deep Learning

Adriano Veloso

DCC-UFMG

References

- Yoshua Bengio's 2009 book: Learning Deep Architectures for AI¹
- Yann LeCun & Marco Aurelio Ranzato's ICML2013 tutorial²
- Richard Socher et. al.'s NAACL2013 tutorial³
- Hugo Larochelle⁴
- Bengio's Deep Learning book⁵
- Chris Bishop's Pattern Recognition and Machine Learning book⁶
- Nando de Freitas Youtube material⁷.
- Kevin Murphy's Machine Learning book ⁸

¹<http://www.iro.umontreal.ca/~bengioy/papers/ftml.pdf>

²<http://techtalks.tv/talks/deep-learning/58122/>

³<http://www.socher.org/index.php/DeepLearningTutorial/>

⁴<http://www.dmi.usherb.ca/~larocheh>

⁵www.deeplearning.org

⁶<http://research.microsoft.com/en-us/um/people/cmbishop/prml/>

⁷<https://www.youtube.com/user/ProfNandoDF>

⁸<http://mitpress.mit.edu/books/machine-learning-2>

A little from:

- Algebra:
 - Vectors
- Calculus:
 - Partial derivatives
- Probability:
 - Conditional probability
 - Bayes
- Optimization:
 - Gradient descent, Lagrange multipliers
- Computer Science:
 - Algorithms and data structures

What is Learning?

- Definition of learning:
 - Learning is an adaptive change in behaviour caused by experience.

What is Learning?

- Definition of learning:
 - Learning is an adaptive change in behaviour caused by experience.
- Pretty much all animals with a central nervous system are capable of learning (even the simplest ones).

What is Learning?

- Definition of learning:
 - Learning is an adaptive change in behaviour caused by experience.
- Pretty much all animals with a central nervous system are capable of learning (even the simplest ones).
- What does it mean for a computer to learn? Why would we want them to learn? How do we get them to learn?
 - We want computers to learn when it is too difficult or too expensive to program them directly to perform a task.
 - Get the computer to program itself by showing examples of inputs and outputs.
 - In reality: we will write a “parameterized” program, and let the learning algorithm find the set of parameters that best approximates the desired function or behavior.

Machine Learning

Arthur Samuel coined the name, 1959:

- “Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.”

Tom Mitchell, 1997:

- “A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .”

Why We Need Machine Learning?

Very hard to solve problems like recognizing two-dimensional or three-dimensional objects.

- We do not know what program to write because we do not know it is done in our brain.
- Even if we know, the program would be very complicated.

Why We Need Machine Learning?

Very hard to solve problems like recognizing two-dimensional or three-dimensional objects.

- We do not know what program to write because we do not know it is done in our brain.
- Even if we know, the program would be very complicated.

0 0 0 1 1 1 1 1 2

2 2 2 2 2 2 3 3 3

3 4 4 4 4 5 5 5 5

6 6 7 7 7 7 8 8 8

9 9 9 9 9 9 9 9 9

Why We Need Machine Learning?

The machine learning approach: Instead of writing a program by hand, we collect lots of examples that specify the correct output for a given input. A machine learning algorithm produces a program from this data.

- The program produced by a learning algorithm may look very different from a typical hand-written program.
- If we do it right, the program works for new cases as well as the ones we trained it on.

Massive amounts of computation/data are now cheaper than paying someone to write a program.

Different Types of Learning

- Supervised Learning: given training examples of inputs and corresponding outputs, produce the correct outputs for new inputs. Example: character recognition.

Different Types of Learning

- Supervised Learning: given training examples of inputs and corresponding outputs, produce the correct outputs for new inputs. Example: character recognition.
- Reinforcement Learning (similar to animal learning): an agent takes inputs from the environment, and takes actions that affect the environment. Occasionally, the agent gets a reward or punishment. The goal is to learn to produce action sequences that maximize the expected reward (e.g. driving a robot without bumping into obstacles).

Different Types of Learning

- Supervised Learning: given training examples of inputs and corresponding outputs, produce the correct outputs for new inputs. Example: character recognition.
- Reinforcement Learning (similar to animal learning): an agent takes inputs from the environment, and takes actions that affect the environment. Occasionally, the agent gets a reward or punishment. The goal is to learn to produce action sequences that maximize the expected reward (e.g. driving a robot without bumping into obstacles).
- Unsupervised Learning: given only inputs as training, find a structure: discover clusters, manifolds, embedding.

Related Fields

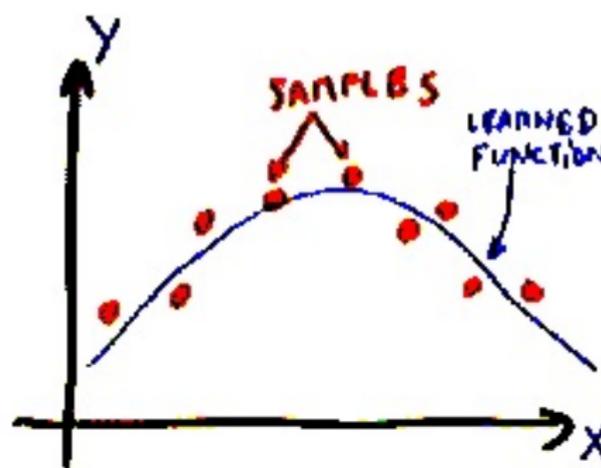
- Statistical Estimation: statistical estimation attempts to solve the same problem as machine learning. Most learning techniques are statistical in nature.
- Pattern Recognition: pattern recognition is when the output of the learning machine is a set of discrete categories.
- Neural Networks: neural nets are now one many techniques for statistical machine learning.
- Data Mining: data mining is a large application area for machine learning.
- Machine Learning methods are an essential ingredient in many fields: bio-informatics, natural language processing, web search and text classification, speech and handwriting recognition, fraud detection, financial time-series prediction, industrial process control, database marketing...

Applications

- Handwriting recognition, OCR: reading checks and zipcodes, handwriting recognition for tablet PCs.
- Speech recognition, speaker recognition/verification
- Security: face detection and recognition, event detection in videos.
- Text classification: indexing, web search.
- Computer vision: object detection and recognition.
- Diagnosis: medical diagnosis (e.g. pap smears processing)
- Fraud detection: e.g. detection of unusual usage patterns for credit cards.
- Financial prediction (many people on Wall Street use machine learning).

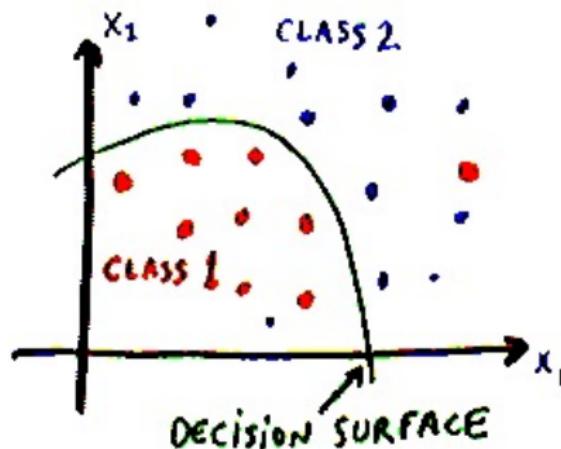
Supervised Learning

- Regression: also known as curve fitting or “function approximation”. Learn a continuous input-output mapping from a limited number of examples (possibly noisy).



Supervised Learning

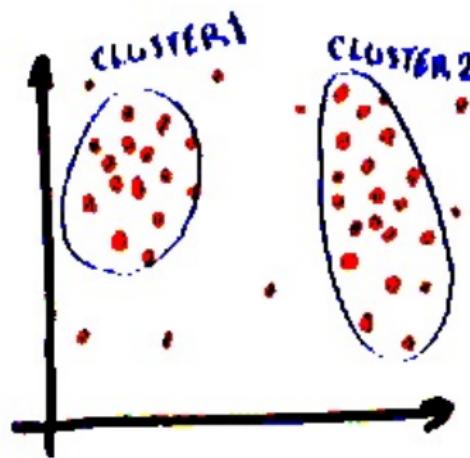
- Classification: outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other. Generally, a “confidence” is also desired (how sure are we that the input belongs to the chosen category).



Unsupervised Learning

This is a horrendously ill-posed problem in high dimension. To do it right, we must guess/discover the hidden structure of the inputs. Methods differ by their assumptions about the nature of the data.

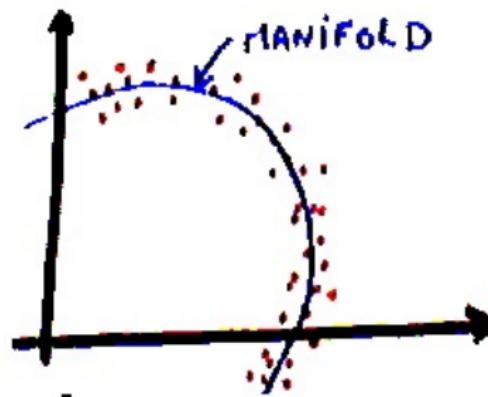
- Clustering: discover groups of points.



Unsupervised Learning

This is a horrendously ill-posed problem in high dimension. To do it right, we must guess/discover the hidden structure of the inputs. Methods differ by their assumptions about the nature of the data.

- Embedding: discover low-dimensional manifold.
 - Discover a function that for each input computes a code from which it can be reconstructed



Supervised Learning: Problem Setup

- Training set: a set of $(X, y)^m$ pairs, where input $X \in R^d$ and output $y \in \{0, 1\}$
- Goal: Learn function/model $f : X \rightarrow y$ to predict correctly on new inputs X

Supervised Learning: Problem Setup

- Training set: a set of $(X, y)^m$ pairs, where input $X \in R^d$ and output $y \in \{0, 1\}$
- Goal: Learn function/model $f : X \rightarrow y$ to predict correctly on new inputs X
 - Step 1: Choose a learning algorithm
 - logistic regression, SVMs, KNNs, decision trees, ANNs etc.
 - Step 2: Optimize parameters/weights W using the training set
 - Minimize a loss function: $\min_W \sum_{i=1}^m (f_w(x^{(i)}) - y^{(i)})^2$

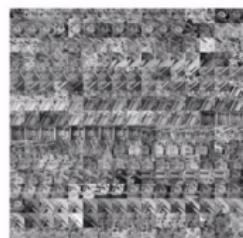
Supervised Learning: Applications

- ATCCGTATAGTCGATCAGTCAGCTACTATGGTAT **CANCER**
 - TGCATGCATGCAGATCGATCGATGCCAACGTAC **NO CANCER**
 - ATTATATTCTGCGATCGAAGCTATGCGATCGTGA **CANCER**
 - TATGCGCGAGTTTATGAGGCGATGATGCTA **CANCER**
 - ATCGCATCGACGTACGATGCTGATTATTAGCCG **NO CANCER**
 - GATCATGCTGCGAGAGGAGATTATGCGATAGA **CANCER**
- ...

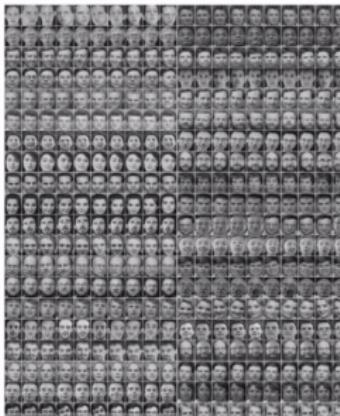
ATCGTCTGATGCAGCGAGCTATCGTACGTAGCA ????



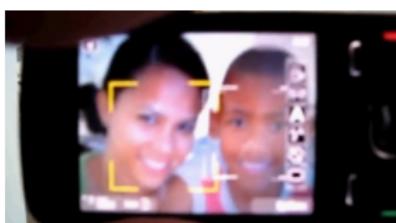
Supervised Learning: Applications



NOT FACES



FACES



Supervised Learning: Applications

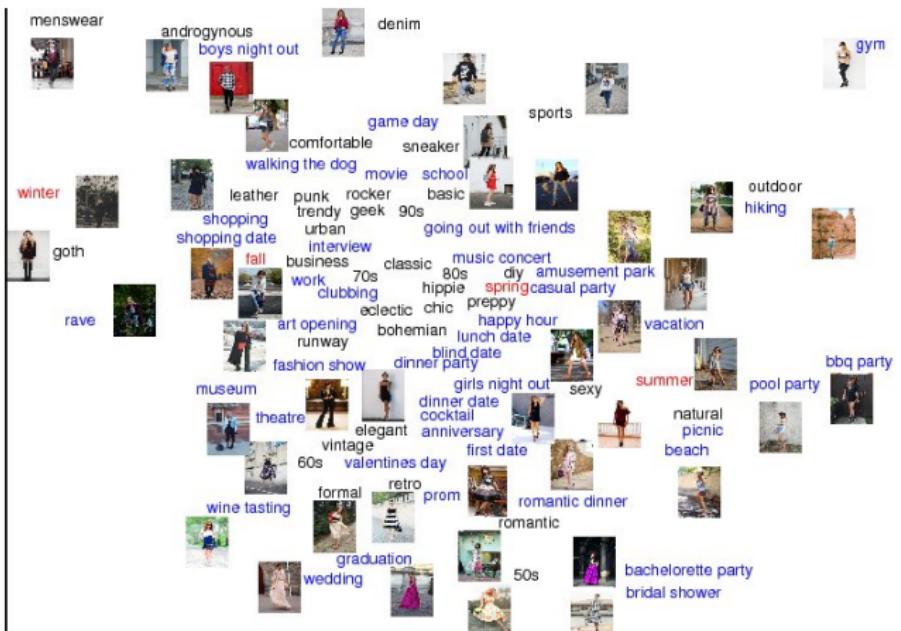
POSITIVE POLARITY (GOOD)

- a mesmerizing cinematic poem from the first frame to the last ."
- a well-put-together piece of urban satire .
- one can't deny its seriousness and quality .
- hard to resist .
- a naturally funny film , home movie makes you crave chris smith's next movie .
- a true-blue delight .
- a fun ride .
- a surprisingly funny movie .
- the script is smart and dark - hallelujah for small favors .
- a flick about our infantilized culture that isn't entirely infantile .

-
- unfortunately the story and the actors are served with a hack script .
 - too slow for a younger crowd , too shallow for an older one .
 - terminally brain dead production .
 - one lousy movie .
 - this movie . . . doesn't deserve the energy it takes to describe how bad it is .
 - a cleverly crafted but ultimately hollow mockumentary .
 - it's an 88-minute highlight reel that's 86 minutes too long .
 - the whole affair is as predictable as can be .

NEGATIVE POLARITY (BAD)

Supervised Learning: Applications



Statistical Learning and Neural Computing

Neural Computing

- Inspired on nervous system
 - Strongly connected to Neurosciences.
- In general:
 - Algorithms are proposed, then theory is created.
 - Perceptron/convnets.
 - Trying to mimic the operation of the brain.
- Many of the influential methods for both fronts were invented within a few meters of each other, and within a few years of each other, in AT&T labs during the 90's.
 - Lecun, Vapnik, Weston, Bengio, Bottou, Schepkopf, Chapelle.

Statistical Learning

- Learning as statistical modeling.
- In general:
 - Theory is created, then algorithms are proposed.
 - SVMs and decision trees.
 - Working to find shortcuts to brain-like behavior.

Statistical Learning and Neural Computing



Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next?

Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next? (There is no way to tell.)
 - 1, 2, 4, 7, 11, 16, ... : ($a_{n+1} = a_n + n$)

Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next? (There is no way to tell.)
 - $1, 2, 4, 7, 11, 16, \dots : (a_{n+1} = a_n + n)$
 - $1, 2, 4, 7, 12, 20, \dots : (a_{n+2} = a_{n+1} + a_n + 1)$

Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next? (There is no way to tell.)
 - 1, 2, 4, 7, 11, 16, ... : ($a_{n+1} = a_n + n$)
 - 1, 2, 4, 7, 12, 20, ... : ($a_{n+2} = a_{n+1} + a_n + 1$)
 - 1, 2, 4, 7, 13, 24, ... : “Tribonacci sequence”

Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next?(There is no way to tell.)
 - 1, 2, 4, 7, 11, 16, ... : ($a_{n+1} = a_n + n$)
 - 1, 2, 4, 7, 12, 20, ... : ($a_{n+2} = a_{n+1} + a_n + 1$)
 - 1, 2, 4, 7, 13, 24, ... : “Tribonacci sequence”
 - 1, 2, 4, 7, 14, 28 : divisors of 28

Learning is NOT Memorization

Problem: new inputs are different from training examples

- The ability to produce correct outputs on previously unseen inputs is called **generalization**.

Observe: 1, 2, 4, 7, ...

- What is the next? (There is no way to tell.)
 - 1, 2, 4, 7, 11, 16, ... : $(a_{n+1} = a_n + n)$
 - 1, 2, 4, 7, 12, 20, ... : $(a_{n+2} = a_{n+1} + a_n + 1)$
 - 1, 2, 4, 7, 13, 24, ... : "Tribonacci sequence"
 - 1, 2, 4, 7, 14, 28 : divisors of 28
 - 1, 2, 4, 7, 1, 1, 5, ... : decimal expansions of $\pi = 3.14159\dots$ and $e = 2.718\dots$ interleaved
- The big question of Learning Theory (and practice): how to get good generalization with a limited number of examples?

Components of Supervised Learning

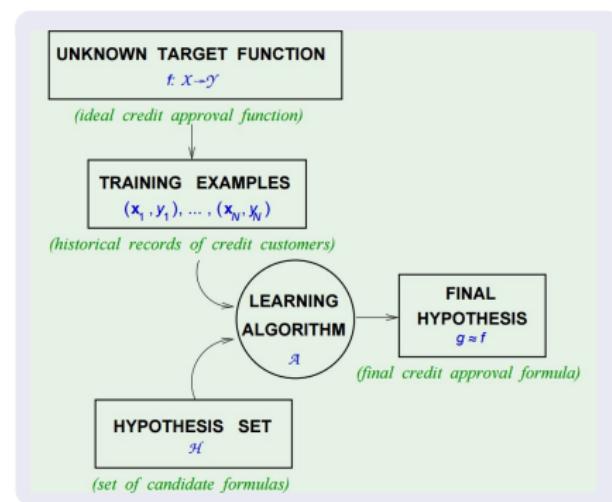
Table: Credit approval.

age	23 years
gender	male
annual salary	\$30,000
years in job	1 year
current debt	\$15,000
...	...

- Input: X (customer application)
- Output: y (good/bad customer?)
- Target function: $f : \mathcal{X} \rightarrow \mathcal{Y}$ (ideal credit approval formula)
- Data: $(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)$ (training set)
- Hypothesis: $g : \mathcal{X} \rightarrow \mathcal{Y}$ (g approximates f well)

Components of Supervised Learning

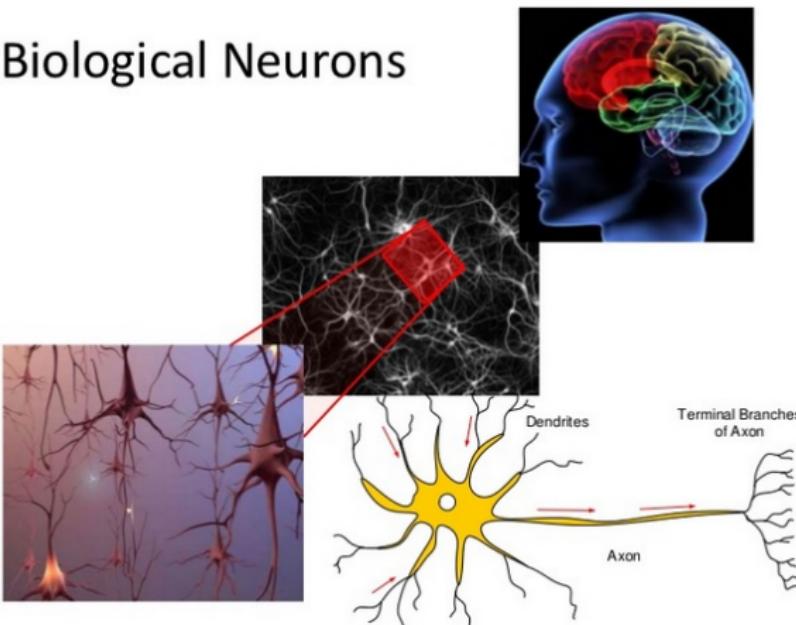
- We want to approximate a target function.
- Need a sample of data generated from the target function.
- A hypothesis set is used to avoid candidate functions that do not approximate well the target function.



- The learning algorithm searches for a good function in the hypothesis set (optimization or heuristics).

How Biology Does It

Biological Neurons



How Biology Does It

- The first attempts at machine learning in the 50's, and the development of artificial neural networks in the 80's and 90's were inspired by biology.
- Nervous Systems are networks of neurons interconnected through synapses
- Learning is based on changes in the “efficacy” of the synapses



How Biology Does It

- A neuron operates by receiving signals from other neurons through connections (synapses).
- The combination of these signals, in excess of a certain threshold or activation level, will result in the neuron firing.
 - Sending a signal on to other neurons connected to it.
- What we call “learning” is believed to be the collective effect of the presence or absence of these firings.

How Biology Does It

- A neuron operates by receiving signals from other neurons through connections (synapses).
- The combination of these signals, in excess of a certain threshold or activation level, will result in the neuron firing.
 - Sending a signal on to other neurons connected to it.
- What we call “learning” is believed to be the collective effect of the presence or absence of these firings.
- There are approximately 100,000,000,000 neurons each connected to 1,000 others in the human brain. The massive number of neurons and the complexity of their interconnections results in a “learning machine”, our brain.

How Biology Does It

- A cortical neuron has dendrites, that are receptors for signals generated by other neurons.
 - These signals are combined and the result will determine whether or not the neuron will fire.
 - If a neuron fires, an electrical impulse is generated. This impulse proceeds down the axon to its ends.
- The end of an axon is split into multiple ends, called boutons.
 - The boutons are connected to the dendrites of other neurons, resulting in synapses.
 - If the neuron has fired, the generated impulse stimulates the boutons and transmit the signal across the synapses to the receiving dendrites.

A Huge Simplification: First Generation of Neural Nets

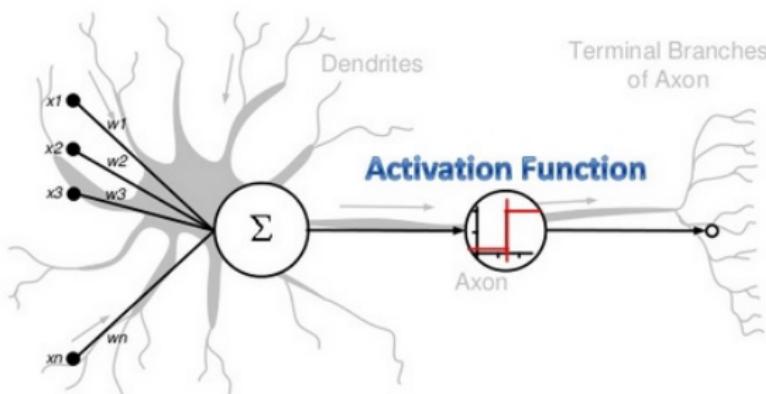


- A machine that learns from trial and error.

A Huge Simplification: First Generation of Neural Nets

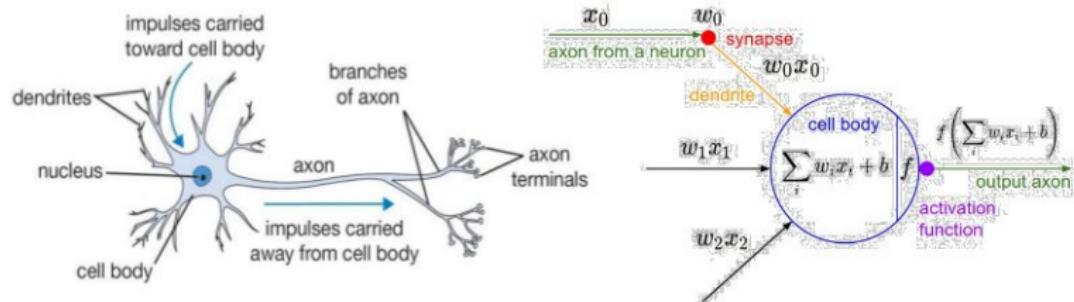
- The neuron consists of multiple inputs and a single output.
- Each input has a weight, which multiplies the input value.
- The neuron combines these weighted inputs in order to determine its output, according to an activation function.

This behavior follows closely our understanding of how cortical neurons work.



A Huge Simplification: First Generation of Neural Nets

Basic model.



A Huge Simplification: First Generation of Neural Nets

Credit approval with a linear classifier.

- Given an input $X = (x_1, x_2, \dots, x_d)$:
 - Approve credit if $\sum_{i=1}^d w_i \times x_i > \text{threshold}$,
 - Deny credit if $\sum_{i=1}^d w_i \times x_i < \text{threshold}$.
- This rule can be written as:
 - $$h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) - \text{threshold} \right)$$

A Huge Simplification: First Generation of Neural Nets

Credit approval with a linear classifier.

- Given an input $X = (x_1, x_2, \dots, x_d)$:
 - Approve credit if $\sum_{i=1}^d w_i \times x_i > \text{threshold}$,
 - Deny credit if $\sum_{i=1}^d w_i \times x_i < \text{threshold}$.
- This rule can be written as:
 - $$h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) - \text{threshold} \right)$$
- Searching for h is to find optimum values for w_i and threshold
 - w_i is high if x_i is evidence for approval.
 - w_i is low if x_i is evidence for denial.
- $$W^T X = \sum_{i=1}^d w_i \times x_i = 0.$$

A Huge Simplification: First Generation of Neural Nets

Credit approval with a linear classifier.

- $$h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) - \text{threshold} \right)$$

A Huge Simplification: First Generation of Neural Nets

Credit approval with a linear classifier.

- $h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) - \text{threshold} \right)$
- $h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) + w_0 \right)$
- Threshold is $-w_0$.

A Huge Simplification: First Generation of Neural Nets

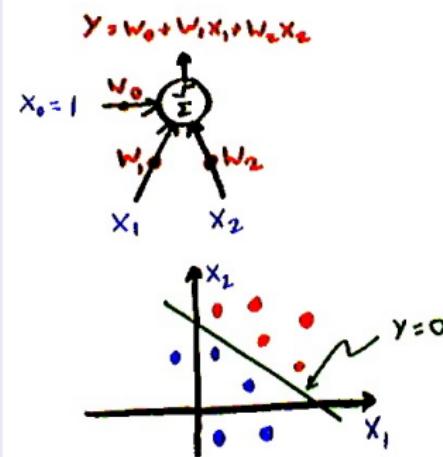
Credit approval with a linear classifier.

- $h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) - \text{threshold} \right)$
- $h(X) = \text{sign} \left(\left(\sum_{i=1}^d w_i \times x_i \right) + w_0 \right)$
 - Threshold is $-w_0$.
- Introduce an artificial coordinate x_0 which is always set to 1.
 - $h(X) = \text{sign} \left(\sum_{i=0}^d w_i \times x_i \right)$
- In vector form: $h(X) = \text{sign}(W^T X)$
 - This is the hypothesis set (e.g., linear separations).

A Huge Simplification: First Generation of Neural Nets

A **neuron** is a linear classifier $y = h(W^T X) = h(\sum_{i=0}^{i=d} w_i x_i)$

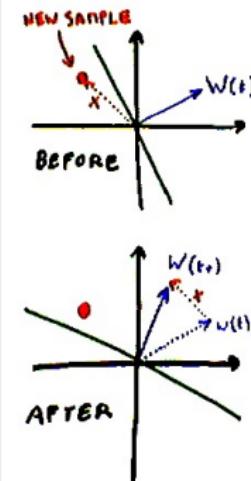
- h is the threshold function:
 $h(z) = 1$ iff $z > 0$, $h(z) = -1$
 otherwise (i.e., sign).
- We want to find a hyperplane
 $W^T X = 0$, which partitions the space in two categories.



Learning as Correcting Errors

We have a training set S of P input-output pairs:
 $S = (X^1, y^1), (X^2, y^2), \dots, (X^P, y^P)$.

- A very simple learning algorithm:
 - ① show each sample in sequence repetitively
 - ② if the output is correct: do nothing
 - ③ if the output is -1, and the correct one is +1: increase/decrease the weights whose inputs are positive/negative
 - ④ if the output is +1, and the correct one is -1: decrease/increase the weights whose inputs are positive/negative
- This algorithm is called Perceptron



Perceptron Learning Procedure: Example

Binary NAND on inputs x_1 and x_2 .

- Inputs: x_0, x_1, x_2 , with $x_0 = 1$
- Threshold: $t = 0.5$
- Bias: $b = 0$
- Learning rate: $r = 0.1$
- Training set:
((1, 0, 0), 1)
((1, 0, 1), 1)
((1, 1, 0), 1)
((1, 1, 1), 0)

Perceptron Learning Procedure: Example

Input			Initial weights			Output					Error	Correction	Final weights			
Sensor values	Desired output					Per sensor	Sum	Network								
x_0	x_1	x_2	z	w_0	w_1	w_2	c_0	c_1	c_2	s	n	e	d	w_0	w_1	w_2
							$x_0 * w_0$	$x_1 * w_1$	$x_2 * w_2$	$c_0 + c_1 + c_2$	if $s > \delta$ then 1, else 0	$z = n$	$r * e$	$\Delta(x_0 * d)$	$\Delta(x_1 * d)$	$\Delta(x_2 * d)$
1	0	0	1	0	0	0	0	0	0	0	0	1	+0.1	0.1	0	0
1	0	1	1	0.1	0	0	0.1	0	0	0.1	0	1	+0.1	0.2	0	0.1
1	1	0	1	0.2	0	0.1	0.2	0	0	0.2	0	1	+0.1	0.3	0.1	0.1
1	1	1	0	0.3	0.1	0.1	0.3	0.1	0.1	0.5	0	0	0	0.3	0.1	0.1
1	0	0	1	0.3	0.1	0.1	0.3	0	0	0.3	0	1	+0.1	0.4	0.1	0.1
1	0	1	1	0.4	0.1	0.1	0.4	0	0.1	0.5	0	1	+0.1	0.5	0.1	0.2
1	1	0	1	0.5	0.1	0.2	0.5	0.1	0	0.6	1	0	0	0.5	0.1	0.2
1	1	1	0	0.5	0.1	0.2	0.5	0.1	0.2	0.8	1	-1	-0.1	0.4	0	0.1
1	0	0	1	0.4	0	0.1	0.4	0	0	0.4	0	1	+0.1	0.5	0	0.1
1	0	1	1	0.5	0	0.1	0.5	0	0.1	0.6	1	0	0	0.5	0	0.1
1	1	0	1	0.5	0	0.1	0.5	0	0	0.5	0	1	+0.1	0.6	0.1	0.1
1	1	1	0	0.6	0.1	0.1	0.6	0.1	0.1	0.8	1	-1	-0.1	0.5	0	0
1	0	0	1	0.5	0	0	0.5	0	0	0.5	0	1	+0.1	0.6	0	0
1	0	1	1	0.6	0	0	0.6	0	0	0.6	1	0	0	0.6	0	0
1	1	0	1	0.6	0	0	0.6	0	0	0.6	1	0	0	0.6	0	0
1	1	1	0	0.6	0	0	0.6	0	0	0.6	1	-1	-0.1	0.5	-0.1	-0.1
1	0	0	1	0.5	-0.1	-0.1	0.5	0	0	0.5	0	1	+0.1	0.6	-0.1	-0.1
1	0	1	1	0.6	-0.1	-0.1	0.6	0	-0.1	0.5	0	1	+0.1	0.7	-0.1	0
1	1	0	1	0.7	-0.1	0	0.7	-0.1	0	0.6	1	0	0	0.7	-0.1	0



Perceptron: Many Variations

Many possible error correction procedures.

- If the output is correct: do nothing
- if $-1/+1$: add the input vector to the weight vector
- if $+1/-1$: subtract the input vector to the weight vector

Perceptron: Many Variations

Many possible error correction procedures.

- If the output is correct: do nothing
- if $-1/+1$: add the input vector to the weight vector
- if $+1/-1$: subtract the input vector to the weight vector

Suppose we have an input $X = (0.5, -0.5)$ connected with weights $W = (2, -1)$, and the bias $b = 0.5$. Furthermore, suppose the target for X is $t = 0$. The threshold function is given as:

$$y = \begin{cases} +1, & \text{if } X'W + b \geq 0. \\ -1, & \text{otherwise.} \end{cases} \quad (1)$$

What will be the weights after applying one step of the Perceptron? X is misclassified as $+1$, instead of -1 . So:

Perceptron: Many Variations

Many possible error correction procedures.

- If the output is correct: do nothing
- if $-1/+1$: add the input vector to the weight vector
- if $+1/-1$: subtract the input vector to the weight vector

Suppose we have an input $X = (0.5, -0.5)$ connected with weights $W = (2, -1)$, and the bias $b = 0.5$. Furthermore, suppose the target for X is $t = 0$. The threshold function is given as:

$$y = \begin{cases} +1, & \text{if } X'W + b \geq 0. \\ -1, & \text{otherwise.} \end{cases} \quad (1)$$

What will be the weights after applying one step of the Perceptron? X is misclassified as $+1$, instead of -1 . So:

- $(w, b) \leftarrow (w, b) - (x, 1) = (2, -1, 0.5) - (0.5, -0.5, 1) = (1.5, -0.5, -0.5)$

Perceptron: Convergence

Perceptron convergence theorem

- Novikoff, 1962

Theorem 11.1: Perceptron Convergence Theorem: Let F be a set of unit-length vectors. If there exists a unit vector A^* and a number $\delta > 0$ such that $A^* \cdot \Phi > \delta$ for all Φ in F , then the program

START: Set A to an arbitrary Φ of F .

TEST: Choose an arbitrary Φ of F , and
if $A \cdot \Phi > 0$ go to **TEST**;
otherwise go to **ADD**.

ADD: Replace A by $A + \Phi$.
Go to **TEST**.

will go to ADD only a finite number of times.

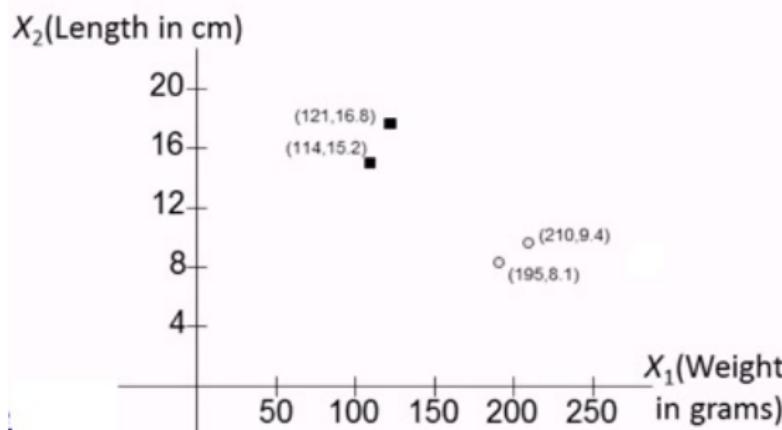
Perceptron: Step by Step

- The following table shows a training set obtained from two different fruits: C_1 and C_2 .

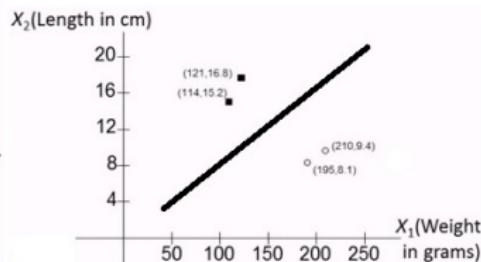
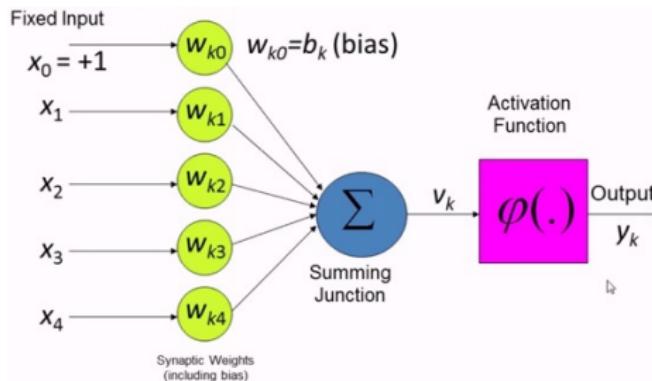
Table: Training set

Fruit	weight (grams)	length (cm)
C_1	121	16.8
C_1	114	15.2
C_2	210	9.4
C_2	195	8.1
?	140	17.9

Perceptron: Step by Step



Perceptron: Step by Step



- In mathematical terms, a neuron k can be described by:

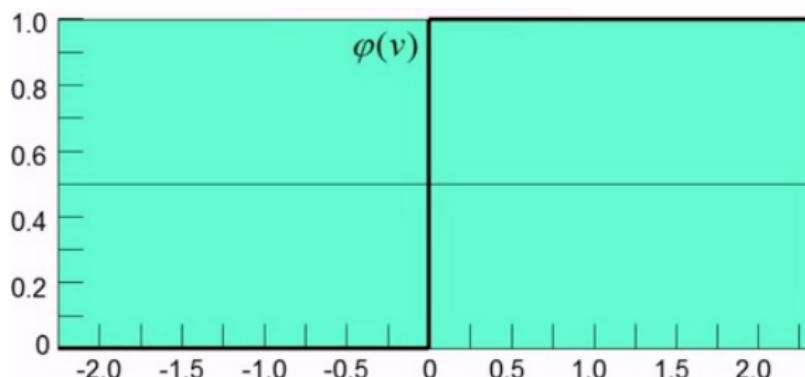
$$a(k) = \sum_{i=1}^n w_{ki}x_i, \text{ and } y(k) = \phi(a_k + b_k)$$

Perceptron: Step by Step

Activation function:

- Heaviside function.

$$\phi(v) = \begin{cases} 1, & \text{if } v \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

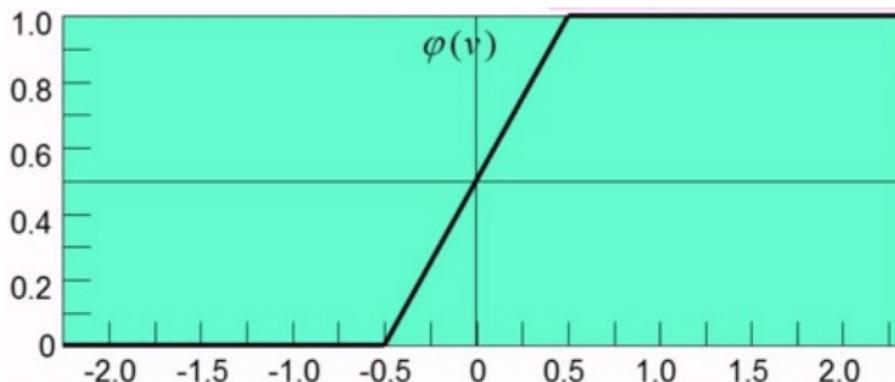


Perceptron: Step by Step

Activation function:

- Piecewise linear function.

$$\phi(v) = \begin{cases} 1, & \text{if } v \geq 0.5. \\ v, & \text{if } 0.5 > v > -0.5 \\ 0, & \text{if } v \leq -0.5. \end{cases}$$

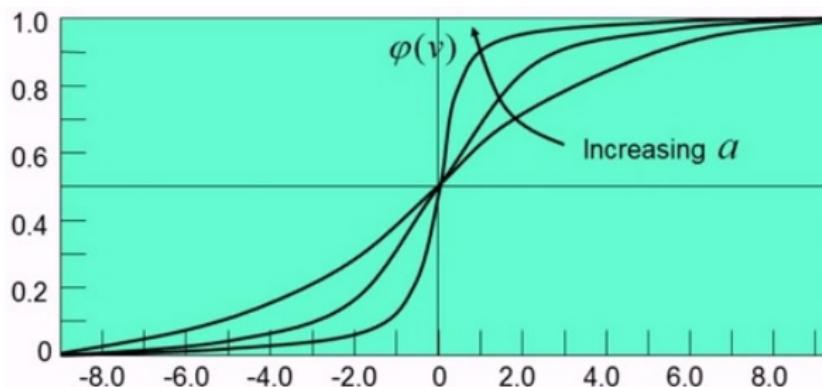


Perceptron: Step by Step

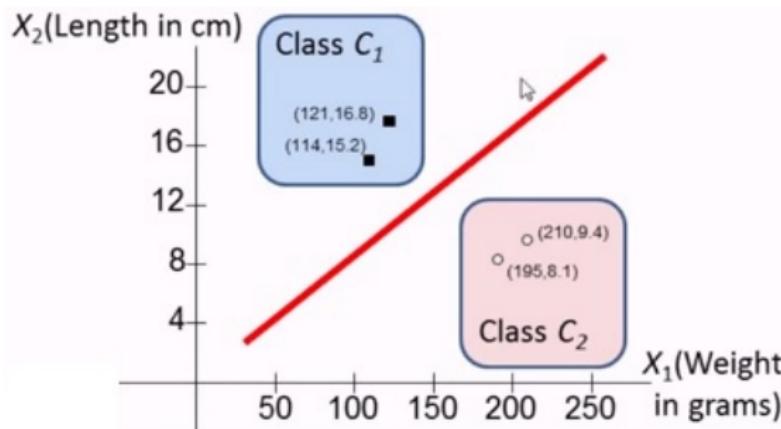
Activation function:

- Sigmoid function.

$$\phi(v) = \frac{1}{1 + \exp(-av)}$$



Perceptron: Step by Step



- Let's assume $y(n) = \text{sgn}[W^T X]$

where $\text{sgn}(\cdot)$ is the signal function:

$$\text{sgn}(x) = \begin{cases} +1, & \text{if } x \geq 0. \\ -1, & \text{otherwise.} \end{cases}$$

Perceptron: Step by Step

Updating the weights.

- $w \leftarrow w + r[d - y]X$

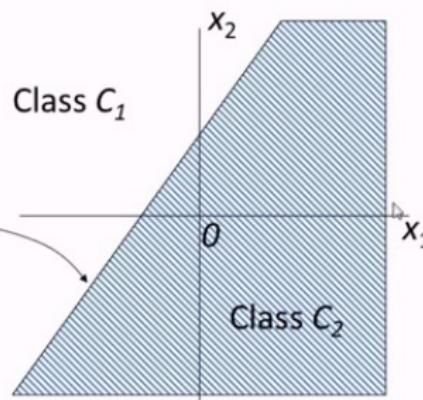
where

$$d = \begin{cases} +1, & \text{if } x \text{ belongs to class } C_1. \\ -1, & \text{otherwise.} \end{cases}$$

$$\sum_{i=1}^m w_i x_i + b = 0$$

or

$$w_1 x_1 + w_2 x_2 + b = 0$$



Perceptron: Step by Step

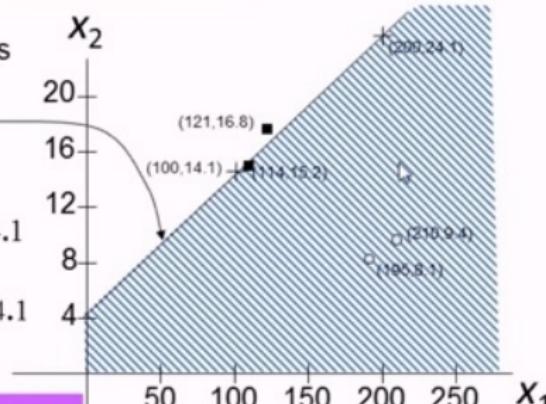
$$\left. \begin{array}{l} w_1(0) = -30, w_2(0) = 300, \\ b(0) = -1230, \eta = 0.01 \end{array} \right\} \text{given}$$

Therefore the Decision Boundary for this case:

$$-30x_1 + 300x_2 - 1230 = 0$$

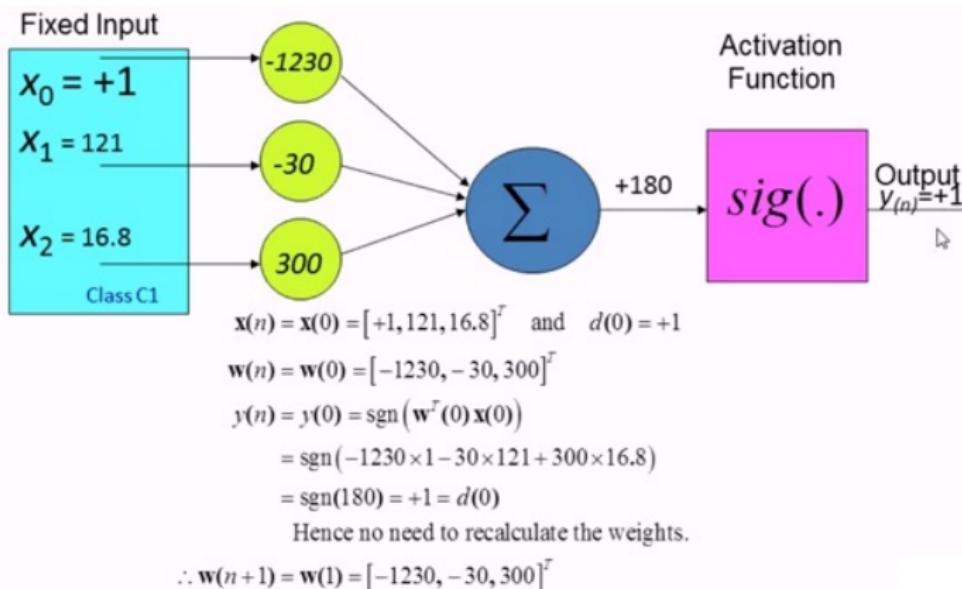
$$x_1 = 100, x_2 = \frac{30 \times 100 + 1230}{300} = 14.1$$

$$x_1 = 200, x_2 = \frac{30 \times 200 + 1230}{300} = 24.1$$

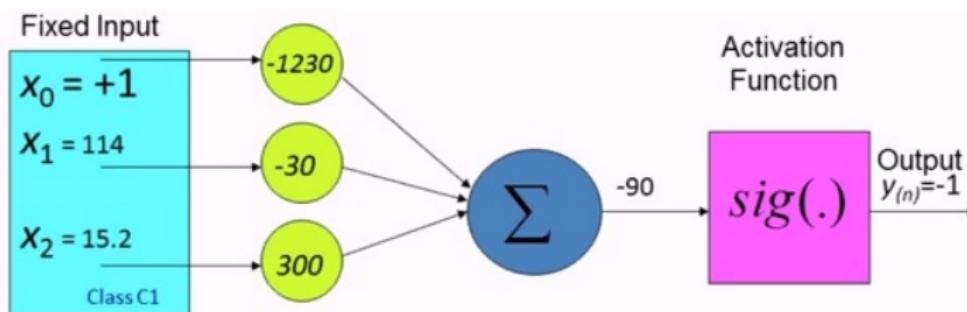


Initial hyperplane does not separate the two classes.
Therefore we need to **Train** the Neural Network

Perceptron: Step by Step



Perceptron: Step by Step



$$\mathbf{x}(l) = [+1, 114, 15.2]^T \quad \text{and} \quad d(l) = +1$$

$$\mathbf{w}(l) = [-1230, -30, 300]^T$$

$$\begin{aligned} y(l) &= \text{sgn}(\mathbf{w}^T(l)\mathbf{x}(l)) = \text{sgn}(-1230 \times 1 - 30 \times 114 + 300 \times 15.2) \\ &= \text{sgn}(-90) = -1 \neq d(l) \end{aligned}$$

Hence we have to recalculate the weights.

Perceptron: Step by Step

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta [d(n) - y(n)] \mathbf{x}(n)$$

$$\mathbf{w}(1) = [-1230, -30, 300]^T$$

$$\mathbf{x}(1) = [+1, 114, 15.2]^T$$

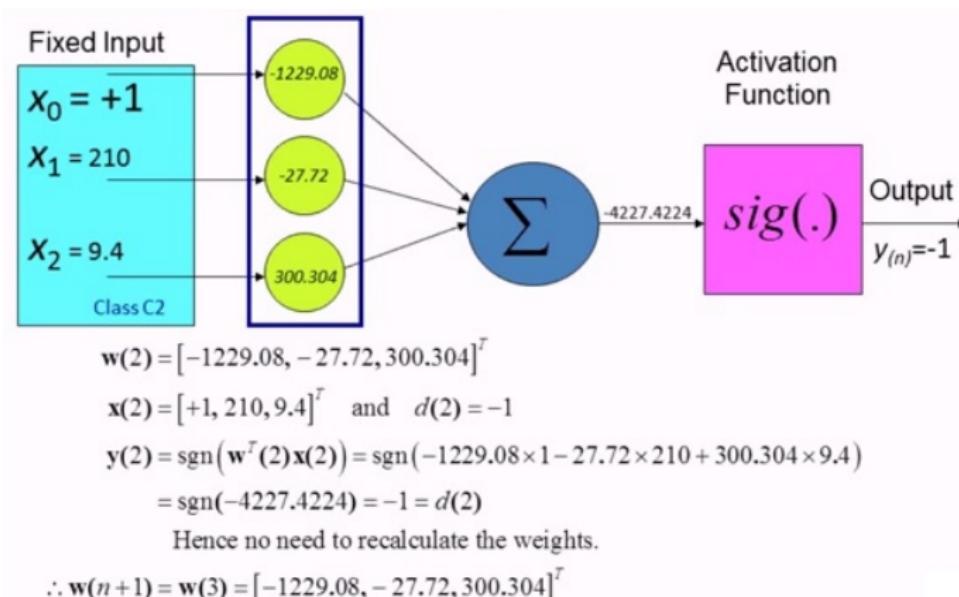
$$d(1) = +1, y(1) = -1, \eta = 0.01$$

$$\mathbf{w}(1+1) = \mathbf{w}(2) = [-1230, -30, 300]^T + 0.01[+1 - (-1)][+1, 114, 15.2]^T$$

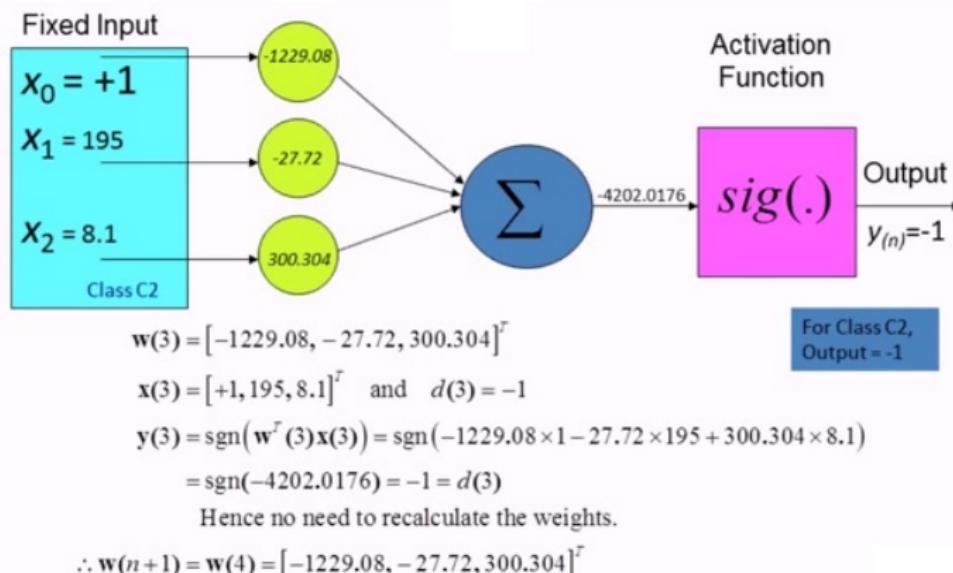
$$= [-1230, -30, 300]^T + [+0.02, 2.28, 0.304]^T$$

$$\therefore \mathbf{w}(2) = [-1229.08, -27.72, 300.304]^T$$

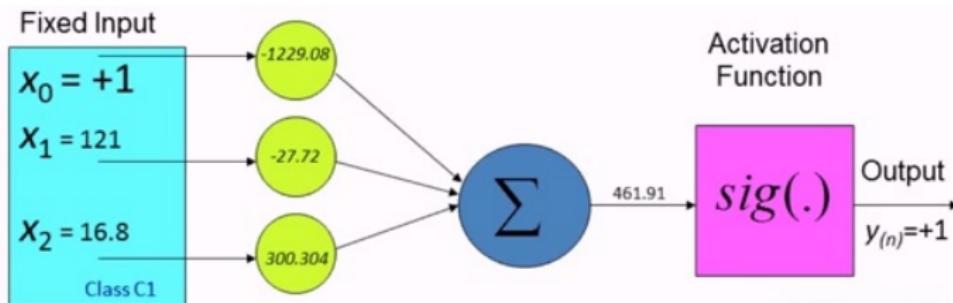
Perceptron: Step by Step



Perceptron: Step by Step



Perceptron: Step by Step



$$\mathbf{w}(4) = [-1229.08, -27.72, 300.304]^T$$

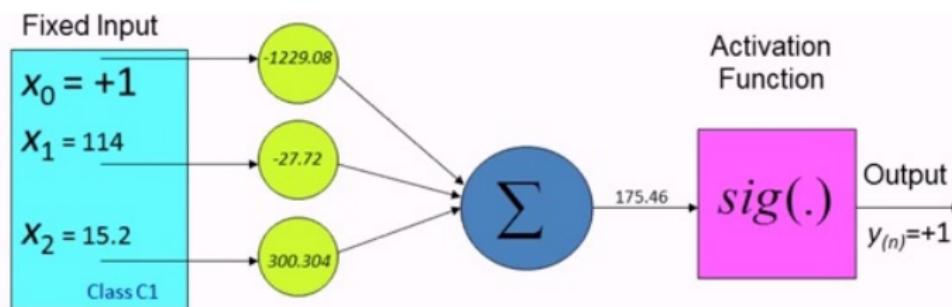
$$\mathbf{x}(4) = [+1, 121, 16.8]^T \quad \text{and} \quad d(4) = +1$$

$$\begin{aligned}y(4) &= \text{sgn}(\mathbf{w}^T(4)\mathbf{x}(4)) = \text{sgn}(-1229.08 \times 1 - 27.72 \times 121 + 300.304 \times 16.8) \\&= \text{sgn}(461.91) = +1 = d(4)\end{aligned}$$

Hence no need to recalculate the weights.

$$\therefore \mathbf{w}(n+1) = \mathbf{w}(5) = [-1229.08, -27.72, 300.304]^T$$

Perceptron: Step by Step



$$\mathbf{w}(5) = [-1229.08, -27.72, 300.304]^T$$

$$\mathbf{x}(5) = [+1, 114, 15.2]^T \quad \text{and} \quad d(5) = +1$$

$$\begin{aligned} \mathbf{y}(5) &= \text{sgn}(\mathbf{w}^T(5)\mathbf{x}(5)) = \text{sgn}(-1229.08 \times 1 - 27.72 \times 114 + 300.304 \times 15.2) \\ &= \text{sgn}(175.46) = +1 = d(5) \end{aligned}$$

Hence no need to recalculate the weights.

$$\therefore \mathbf{w}(n+1) = \mathbf{w}(6) = [-1229.08, -27.72, 300.304]^T$$

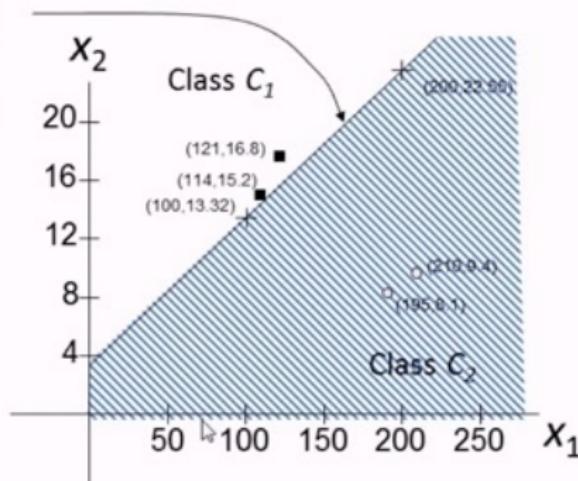
For Class C1,
Output = +1

Perceptron: Step by Step

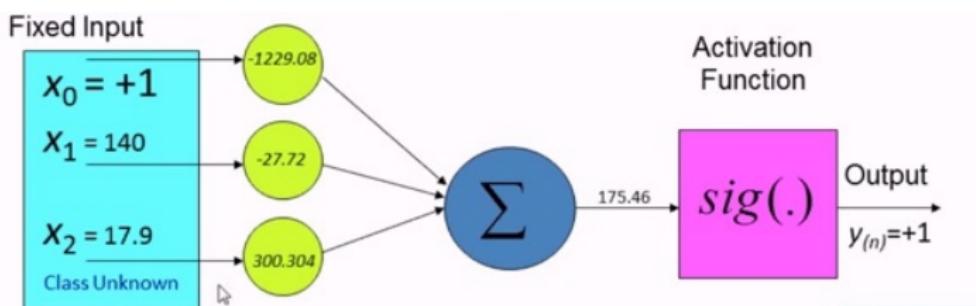
$$-27.72x_1 + 300.304x_2 - 1229.08 = 0$$

$$x_1 = 100, x_2 = \frac{27.72 \times 100 + 1229.08}{300.30} = 13.32$$

$$x_1 = 200, x_2 = \frac{27.72 \times 200 + 1229.08}{300.30} = 22.55$$



Perceptron: Step by Step



Now use the above model to classify the unknown fruit.

$$\mathbf{x}(\text{unknown}) = [+1, 140, 17.9]^T$$

$$\mathbf{w}(5) = [-1229.08, -27.72, 300.304]^T$$

$$\begin{aligned}\mathbf{y}(\text{unknown}) &= \text{sgn}(\mathbf{w}^T(5)\mathbf{x}(\text{unknown})) \\ &= \text{sgn}(-1229.08 \times 1 - 27.72 \times 140 + 300.304 \times 17.9) \\ &= \text{sgn}(265.56) = +1\end{aligned}$$

\therefore this unknown fruit belongs to the class C_1

Machine Learning: Historical Perspective

The Perceptron was the first approach to Machine Learning

- They were investigated in the early 60's by Frank Rosenblatt, and looked very promising as learning devices.
 - Lots of claims were made for what Perceptrons could learn.

NYT on a 1958 Press conference (Rosenblatt & ONR):

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.¹⁷

Machine Learning: Historical Perspective

The Perceptron was the first approach to Machine Learning

- But then the Perceptron fell into disfavor, since Minsky and Papert showed it is very restricted in what it could learn.
 - Researchers stopped pursuing Perceptron for the next decade.
 - In the 70's, the machine learning field was dormant.

Hecht-Nielssen (1990)

The campaign was waged by means of personal persuasion by Minsky and Papert and their allies, as well as by limited circulation of an unpublished technical manuscript (which was later de-venomized and, after further refinement and expansion, published in 1969 as the book *Perceptrons*).²⁰

Machine Learning: Historical Perspective

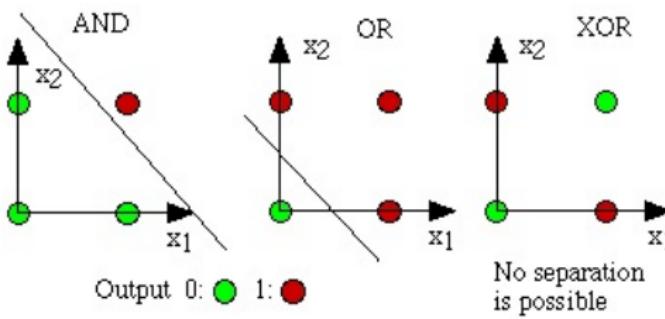
The Perceptron was the first approach to Machine Learning

Minsky & Papert (1969): *Perceptrons*

Perceptrons have been widely publicized as "pattern recognition" or "learning" machines and as such have been discussed in a large number of books, journal articles, and voluminous "reports." Most of this writing ... is without scientific value. (p. 4)

[We] became involved with a somewhat therapeutic compulsion: to dispel what we feared to be the first shadows of a "holistic" or "Gestalt" misconception that would threaten to haunt the fields of engineering and artificial intelligence... (p. 20)

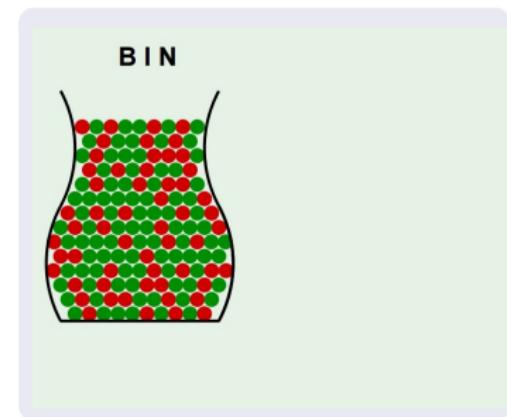
There is no reason to suppose that any of these virtues carry over to the many layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile. (p. 231)



Is Learning Feasible?

Experiment:

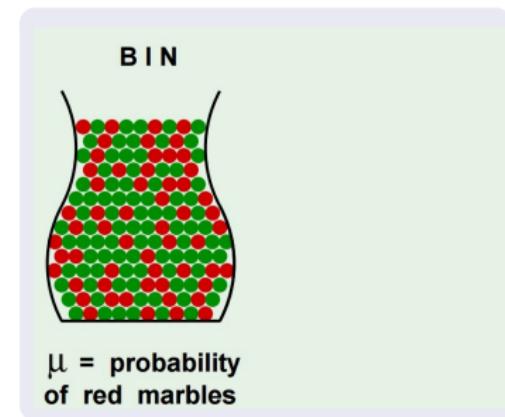
- Consider a bin with red and green marbles.
 - $p[\text{picking a red marble}]$
 - $p[\text{picking a green marble}]$



Is Learning Feasible?

Experiment:

- Consider a bin with red and green marbles.
 - $p[\text{picking a red marble}] = \mu$
 - $p[\text{picking a green marble}] = 1 - \mu$

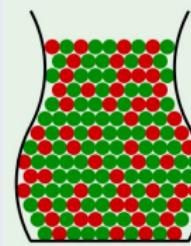


Is Learning Feasible?

Experiment:

- Consider a bin with red and green marbles.
 - $p[\text{picking a red marble}] = \mu$
 - $p[\text{picking a green marble}] = 1 - \mu$
- The value of μ is unknown to us.

BIN

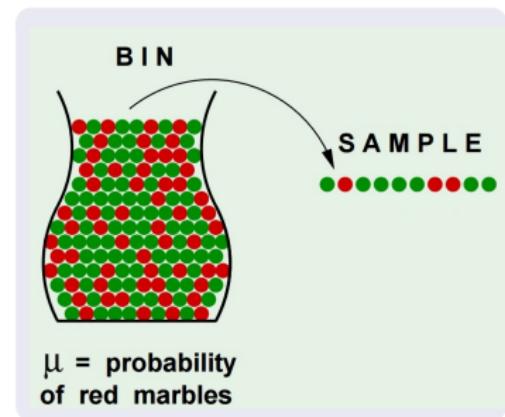


μ = probability
of red marbles

Is Learning Feasible?

Experiment:

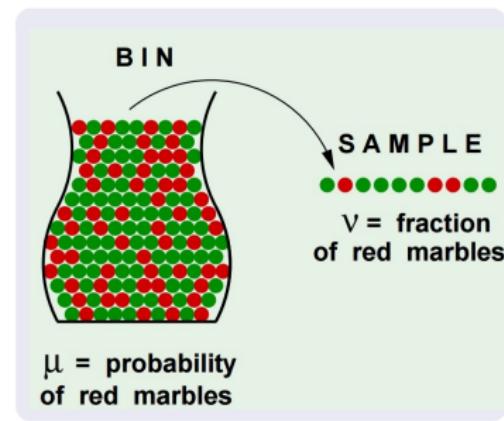
- Consider a bin with red and green marbles.
 - $p[\text{picking a red marble}] = \mu$
 - $p[\text{picking a green marble}] = 1 - \mu$
- The value of μ is unknown to us.
 - But we pick N marbles independently.



Is Learning Feasible?

Experiment:

- Consider a bin with red and green marbles.
 - $p[\text{picking a red marble}] = \mu$
 - $p[\text{picking a green marble}] = 1 - \mu$
- The value of μ is unknown to us.
 - But we pick N marbles independently.
 - The fraction of red marbles in sample is ν .



Is Learning Feasible?

Does ν says anything about μ ?

Is Learning Feasible?

Does ν says anything about μ ?

- No!
 - It is **possible** that you have a bin with 90 red marbles and 10 green marbles, and you get a sample of 10 green marbles.

Is Learning Feasible?

Does ν says anything about μ ?

- No!
 - It is **possible** that you have a bin with 90 red marbles and 10 green marbles, and you get a sample of 10 green marbles.
- Yes!
 - Sample frequency ν is **probably** close to bin frequency μ .

Is Learning Feasible?

Does ν says anything about μ ?

- No!
 - It is **possible** that you have a bin with 90 red marbles and 10 green marbles, and you get a sample of 10 green marbles.
- Yes!
 - Sample frequency ν is **probably** close to bin frequency μ .
- Possible vs. probable!
 - Absolutely certain vs. almost certain. (poll during elections)

Is Learning Feasible?

What does ν say about μ ?

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[\quad] \leq$ (the probability of something is small)

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[\text{ bad event }] \leq$

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[|\nu - \mu| > \epsilon] \leq$

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[|\nu - \mu| > \epsilon] \leq e^{-N}$ (how small it can be)

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[|\nu - \mu| > \epsilon] \leq e^{-\epsilon^2 N}$

Is Learning Feasible?

What does ν say about μ ?

- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$ (Hoeffding's Inequality)

Is Learning Feasible?

What does ν say about μ ?

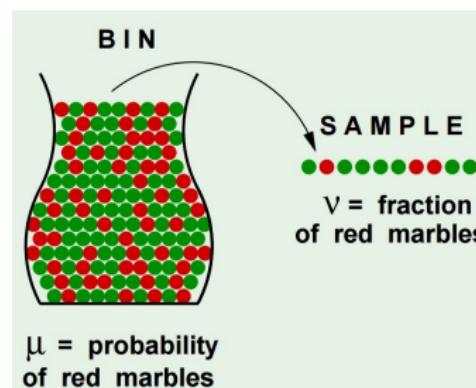
- In a big sample, ν is probably close to μ .
 - Big sample: N is large.
 - ν is probably close to μ : within ϵ .
- Formally:
 - $p[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

Thus, the statement $\mu = \nu$ is P.A.C.

Is Learning Feasible?

$$P[|\nu - \mu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$$

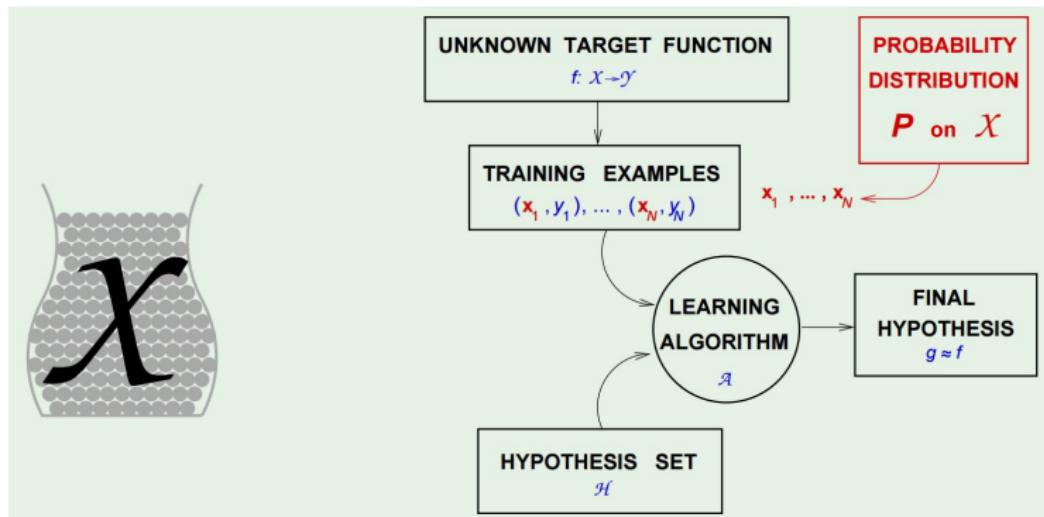
- Valid for all N and ϵ (good because it is an exponential)
- Bound does not depend on μ (good because μ is unknown)
- Tradeoff between N and ϵ (small ϵ demands large N)



Is Learning Feasible?

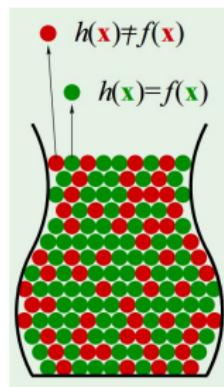
Connection to machine learning.

- Bin: the unknown is a number μ
- ML: the unknown is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$



Is Learning Feasible?

Now we know how to verify a hypothesis h (e.g., a function)

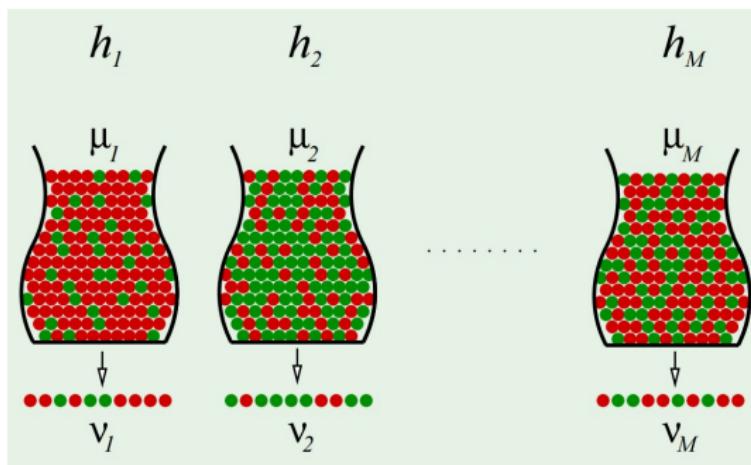


- No guarantee that ν will be small.
- We need to choose the hypothesis from multiple h 's.

Is Learning Feasible?

Try multiple hypothesis (or functions/models)

- The hypothesis can be evaluated with the sample.
 - Choose the better hypothesis among all the evaluated ones.



Is Learning Feasible?

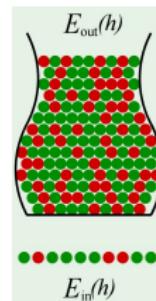
Notation.

- Both μ and ν depends on the choice of h .
 - ν is an error **in sample**, $E_{in}(h)$
 - If $E_{in}(h)$ is low then

Is Learning Feasible?

Notation.

- Both μ and ν depends on the choice of h .
 - ν is an error **in sample**, $E_{in}(h)$
 - If $E_{in}(h)$ is low then h may be a good hypothesis
 - μ is an error **out sample**, $E_{out}(h)$
 - If $E_{out}(h)$ is low then h is really a good hypothesis.



- $p[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

Is Learning Feasible?

Notation.

- Both μ and ν depends on the choice of h .
 - ν is an error **in sample**, $E_{in}(h)$
 - If $E_{in}(h)$ is low then h may be a good hypothesis
 - μ is an error **out sample**, $E_{out}(h)$
 - If $E_{out}(h)$ is low then h is really a good hypothesis.
- $p[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}$

Is Learning Feasible?

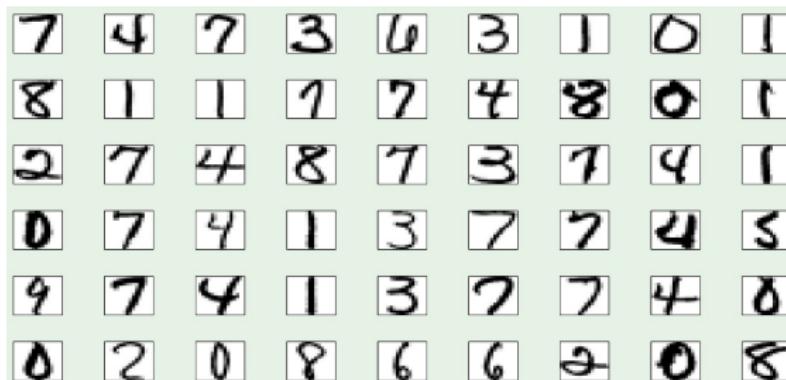
Notation.

- Both μ and ν depends on the choice of h .
 - ν is an error **in sample**, $E_{in}(h)$
 - If $E_{in}(h)$ is low then h may be a good hypothesis
 - μ is an error **out sample**, $E_{out}(h)$
 - If $E_{out}(h)$ is low then h is really a good hypothesis.
- $p[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2Me^{-2\epsilon^2N}$ (Union bound)
 - M is the number of hypothesis.
 - The more hypothesis you try, the higher is the chance of choosing a complex one.
 - $E_{in}(h)$ and $E_{out}(h)$ deviates as h becomes complex.

The Linear Model

16 × 16 images from zip code.

- Human error is 2.2%.



The Linear Model

Input representation.

- Each pixel is an attribute of the input
 - The dimension is 256, $X = (x_1, x_2, \dots, x_{256})$.

The Linear Model

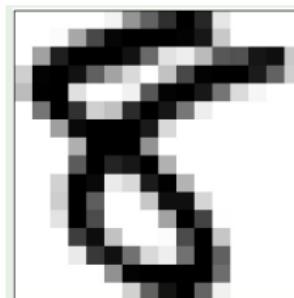
Input representation.

- Each pixel is an attribute of the input
 - The dimension is 256, $X = (x_1, x_2, \dots, x_{256})$.
 - Too long representation for a so simple object.
 - 256 parameters.
- We may know something from the problem.

The Linear Model

Input representation.

- Each pixel is an attribute of the input
 - The dimension is 256, $X = (x_1, x_2, \dots, x_{256})$.
 - Too long representation for a so simple object.
 - 256 parameters.
- We may know something from the problem.
 - Features: intesinty and symmetry.
 - Now the dimension is 2, $X = (x_1, x_2)$



The Linear Model

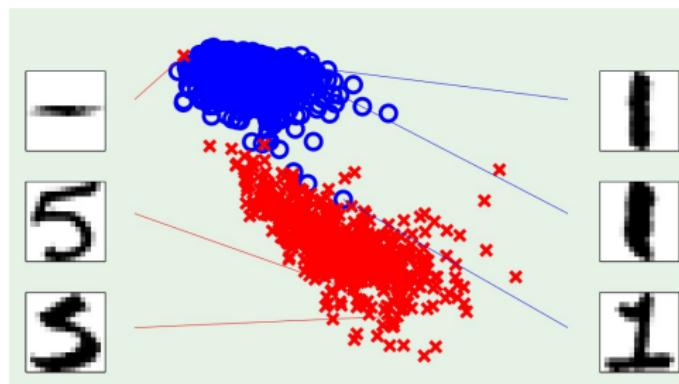
Input representation.

- Extracting features is a controversial subject.
 - Approach 1:
 - Help the machine with prior knowledge, and manually extract the features.
 - This is the dominant approach.
 - Approach 2:
 - Let the machine find the best representation with no human intervention.
 - This is gaining popularity.

The Linear Model

Input representation.

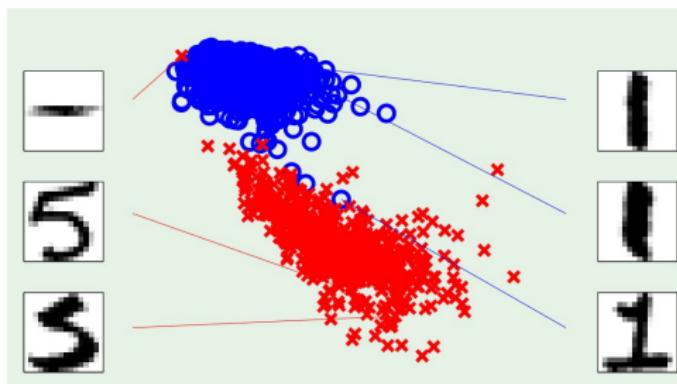
- x_1 : intensity, x_2 : symmetry.



The Linear Model

Input representation.

- x_1 : intensity, x_2 : symmetry.

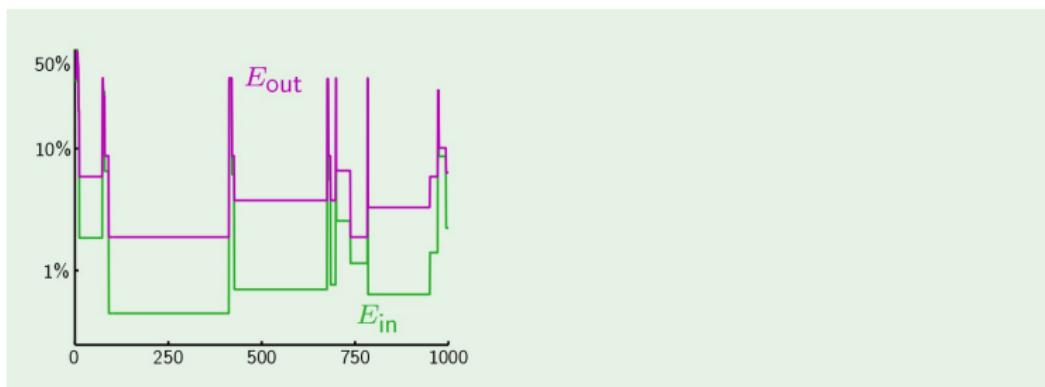


- What happens when we run Perceptron?

The Linear Model

What the Perceptron does?

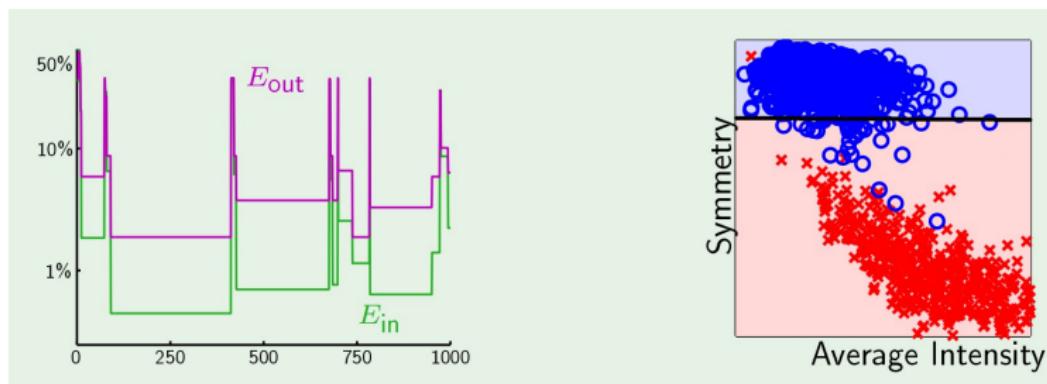
- Evolution of E_{in} and E_{out} .



The Linear Model

What the Perceptron does?

- Evolution of E_{in} and E_{out} .



The Linear Model

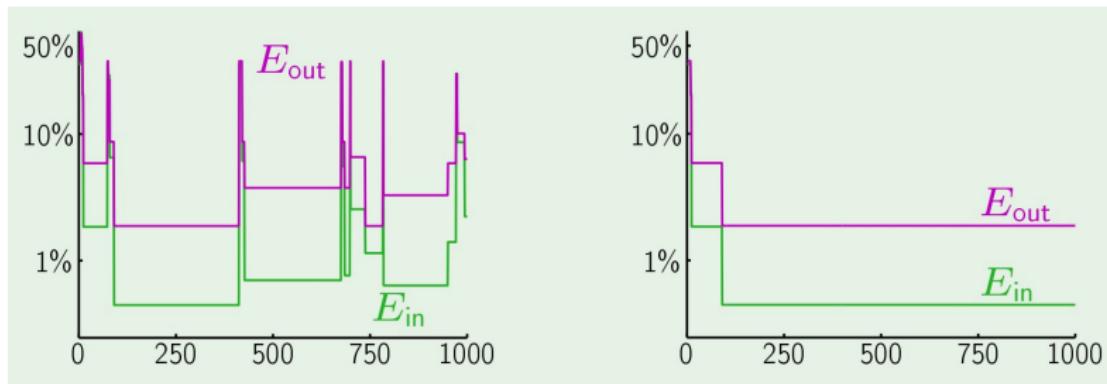
The pocket Perceptron.

- Always keep in the pocket the function with the lowest E_{in} .

The Linear Model

The pocket Perceptron.

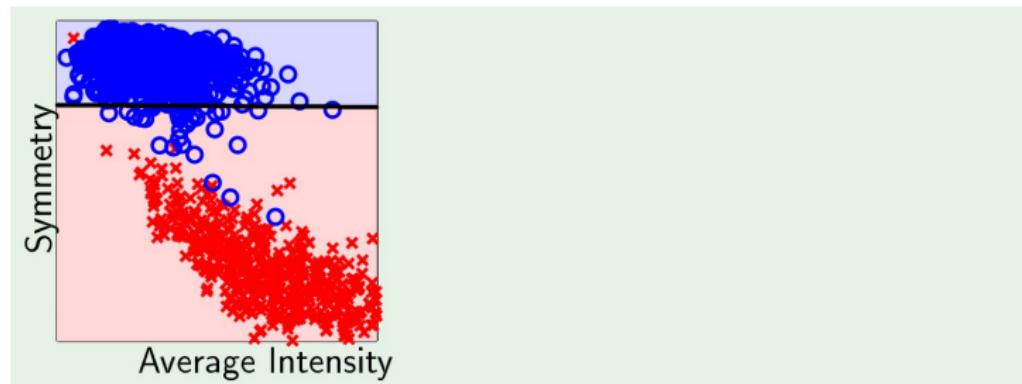
- Always keep in the pocket the function with the lowest E_{in} .



The Linear Model

The pocket Perceptron.

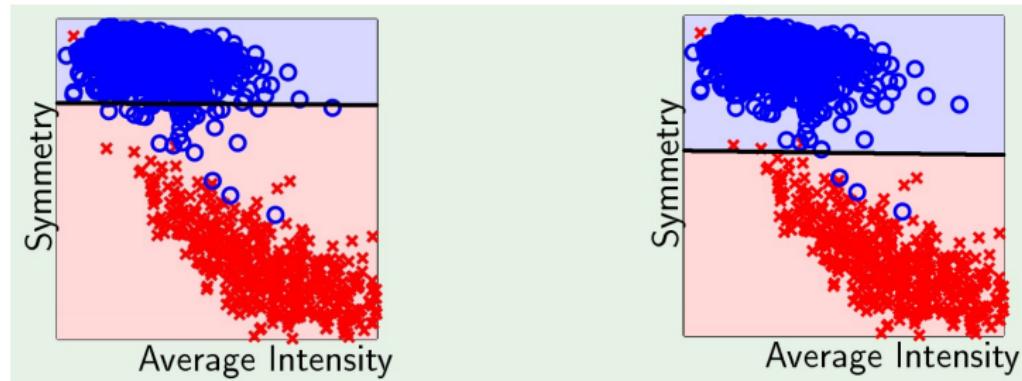
- Always keep in the pocket the function with the lowest E_{in} .



The Linear Model

The pocket Perceptron.

- Always keep in the pocket the function with the lowest E_{in} .



The Linear Model

The pocket Perceptron.

- Empirical Error Minimization.
 - Searching for the best function based solely on E_{in} .

The Linear Model

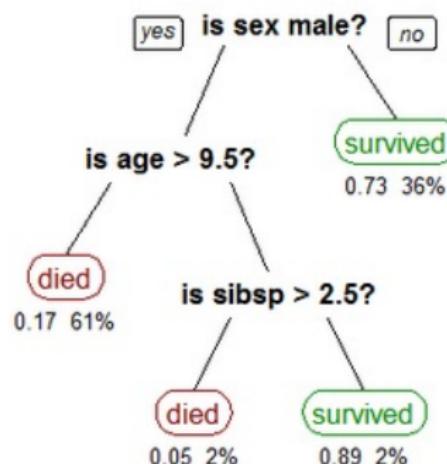
The pocket Perceptron.

- Empirical Error Minimization.
 - Searching for the best function based solely on E_{in} .
 - E_{in} is also called **empirical error**.

Machine Learning: Historical Perspective

The revival of machine learning came in mid-80's, when the decision tree model was invented and distributed as software.

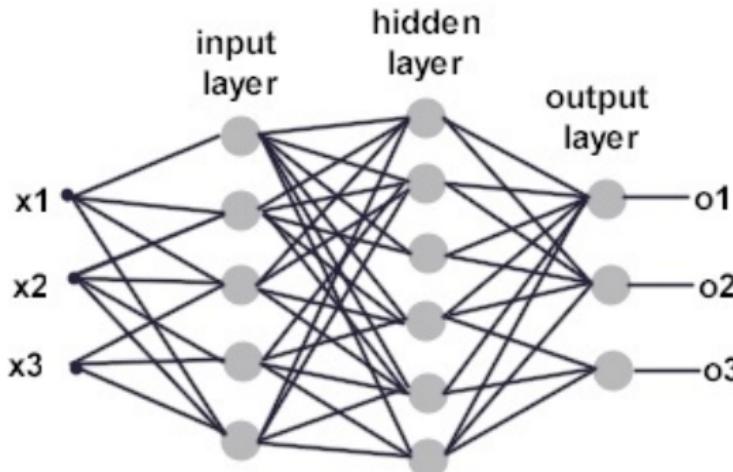
- The model can be viewed by a human and is easy to explain.
- It is also very versatile and can adapt to different problems.



Machine Learning: Historical Perspective

It is also in mid 80's that multi-layer neural networks were created.

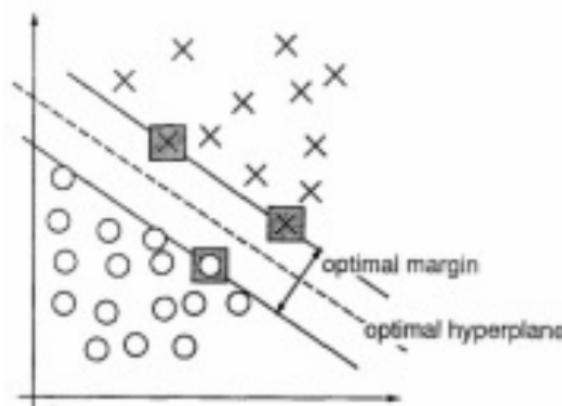
- With enough hidden layers, a neural network can express any function, thus overcoming the limitations of Perceptron.
- With Backpropagation, we see a revival of neural networks.



Machine Learning: Historical Perspective

Machine learning saw rapid growth in the 90's, due to the invention of World-Wide-Web and large data gathered on the Internet.

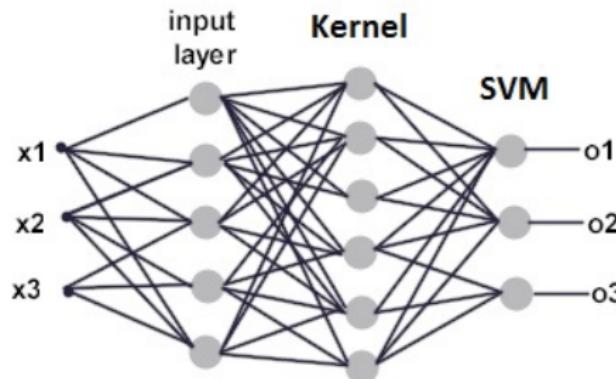
- Around 1995, SVM was proposed and have become adopted.
- SVM packages like libSVM and SVM light make it popular.



Machine Learning: Historical Perspective

In early 00's the Machine Learning community was divided into two crowds: neural networks or SVM advocates.

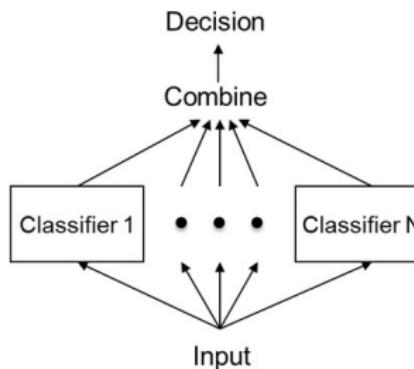
- It was not easy for the neural network side, after the kernelized version of SVM.
- SVM got the best of many tasks that were occupied by neural networks models before.



Machine Learning: Historical Perspective

In addition to the individual methods, we have seen the invention of ensemble learning, where several methods are used together.

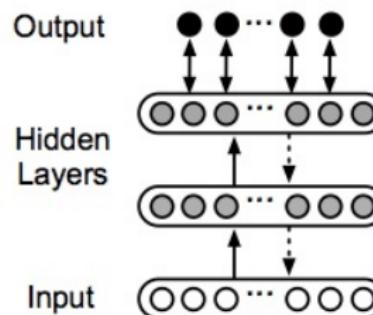
- Adaboost trains a weak set of classifiers, by giving more importance to hard instances.
- Random forests are a combination of tree predictors, showing endurance against overfitting.



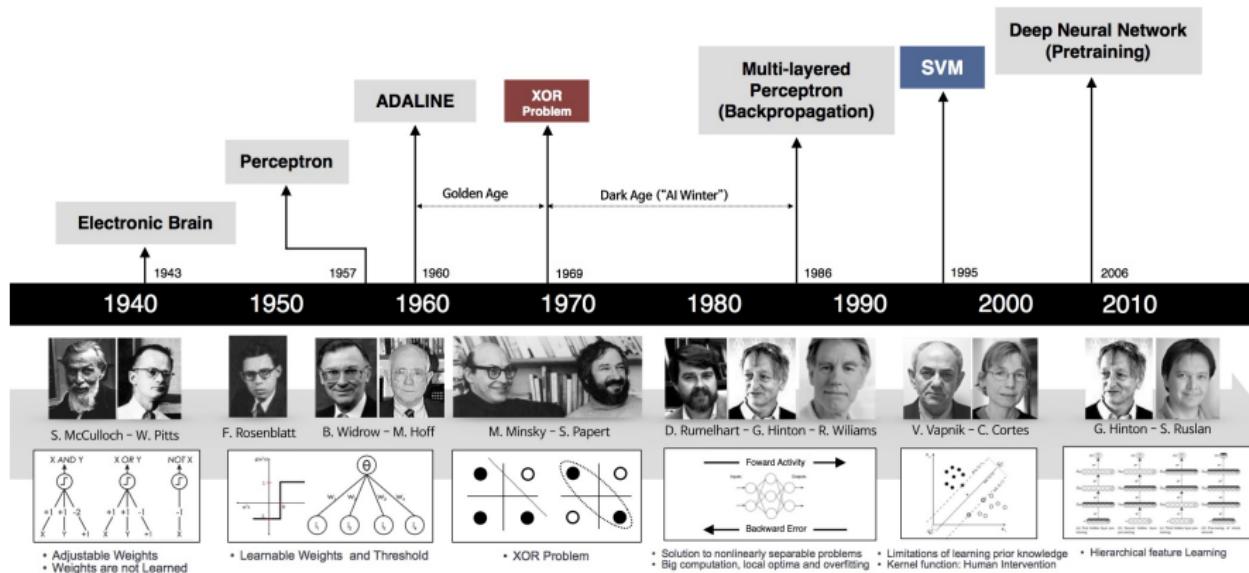
Machine Learning: Historical Perspective

As we come closer today, a new era called Deep Learning has been commerced.

- Neural network models with many wide successive layers.
- Deep Learning models are able to beat off state of art at very different tasks such as Object Recognition, NLP etc.
 - However, there are many critics directed to training cost and tuning exogenous parameters of these models.
 - Moreover, **SVM is used more commonly owing to its simplicity.**



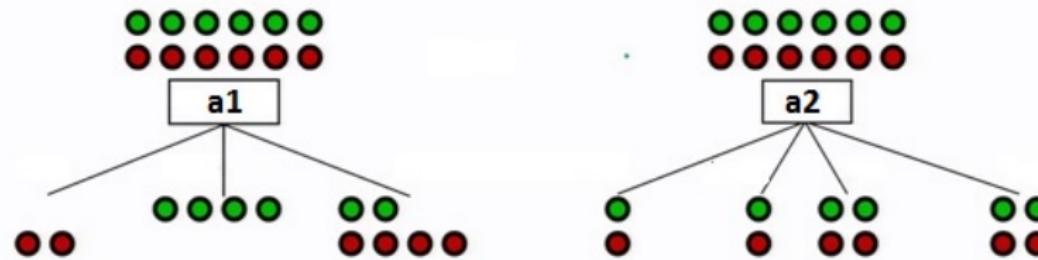
Machine Learning: Historical Perspective



Learning Decision Trees

How do we construct a good tree?

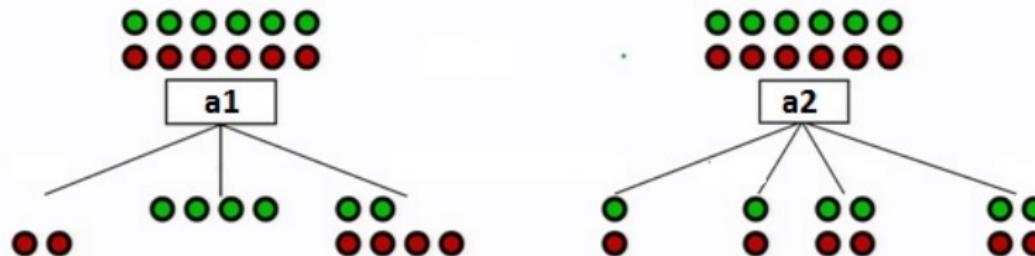
- A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.



Learning Decision Trees

How do we construct a good tree?

- A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.



- For a training set containing p positive examples and n negative examples, we have:

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log \frac{p}{p+n} - \frac{n}{p+n} \log \frac{n}{p+n}$$

Learning Decision Trees



Leo Breiman 1928-2005

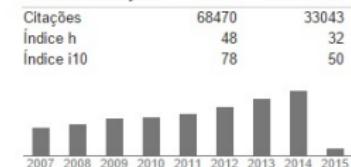
Professor of Statistics, UC Berkeley
 Data Analysis, Statistics, Machine Learning
 E-mail confirmado em stat.berkeley.edu - Página inicial

[Seguir](#) ▾

Google Acadêmico



Índices de citações	Todos	Desde 2010
Citações	68470	33043
Índice h	48	32
Índice i10	78	50



Título	1–20	Citado por	Ano
Classification and Regression Trees L Breiman, JH Friedman, RA Olshen, CJ Stone CRC Press, New York		26911 *	1999
Classification and regression trees L Breiman Chapman & Hall/CRC		26911	1984

Learning Decision Trees

- A chosen attribute A , with k distinct values, divide the training set S into subsets S_1, S_2, \dots, S_k .
- The expected entropy remaining after trying attribute A (with branches $i = 1, 2, \dots, k$) is:

$$EH(A) = \sum_{i=1}^k \frac{p_i + n + i}{p + n} H\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- The information gain I (or the reduction in entropy) for A is:

$$I(A) = H\left(\frac{p}{p + n}, \frac{n}{p + n}\right) - EH(A)$$

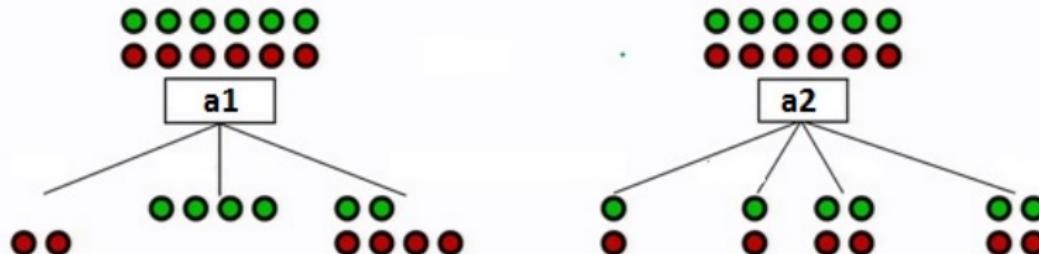
- Choose the attribute with the largest I

Learning Decision Trees

- $p = n = 6$, $H(\frac{6}{12}, \frac{6}{12}) = 1$ bit.
- Consider attributes a_1 and a_2 .

$$I(a_1) = 1 - \left[\frac{2}{12} H(0, 1) + \frac{4}{12} H(1, 0) + \frac{6}{12} H\left(\frac{2}{6}, \frac{4}{6}\right) \right] = 0.541$$

$$I(a_2) = 1 - \left[\frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} H\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} H\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0$$



Learning Decision Trees

QUESTION: When do we stop growing the decision tree?

- One approach is to continue growing the tree until the leaf nodes are empty.
- Or until the leaf nodes are pure.
 - Training error is 0.

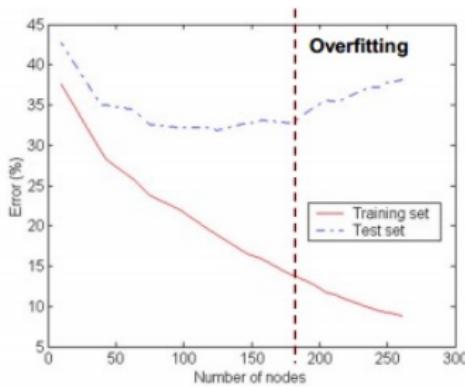
this approach leads to large trees with many leaf nodes, many of which contain small samples.

- Using small samples to assign the class of a leaf node is unreliable. This can yield high apparent accuracy/performance on the training set, but may generalize poorly to future unseen test cases.

Learning Decision Trees



Learning Decision Trees



- Overfitting: When the tree becomes too large, its test error begins increasing while its training error continues to decrease.
 - Training error is not a good approximation of the test error.
- Underfitting: when the tree is too simple, both training and test errors are large

Learning Decision Trees

Early stopping: An approach to early stopping is to stop adding nodes when no attribute test yields an improvement in class purity.

- This prevents us from adding nodes that do not appear to improve classification accuracy.
- One problem with this approach is that it is difficult to judge if a test is useful without seeing the splits that might be installed after it.

In general, early stopping is difficult when combinations of attributes can yield good classification, but those attributes in isolation look poor.

Learning Decision Trees

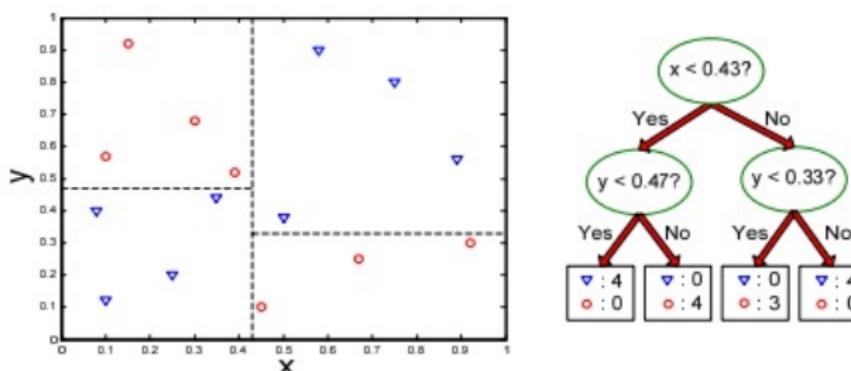
Post pruning: A better approach to overgrowing the tree and overfitting, is to grow the tree to maximum size, and then prune it back.

- One way to prune leafs is reduced error pruning: prune away any node that does not make the accuracy of the tree worse when it is pruned.

Learning Decision Trees

Post pruning: A better approach to overgrowing the tree and overfitting, is to grow the tree to maximum size, and then prune it back.

- One way to prune leafs is reduced error pruning: prune away any node that does not make the accuracy of the tree worse when it is pruned.



Training Error and Test Error

What we are really interested in is good accuracy on unseen data.

- In practice, we split the data into two subsets: a training set and a test set.
- We learn a model using the training set, and evaluate its accuracy on the test set.

Training Error and Test Error

What we are really interested in is good accuracy on unseen data.

- In practice, we split the data into two subsets: a training set and a test set.
- We learn a model using the training set, and evaluate its accuracy on the test set.
 - The accuracy of the model is assessed using a **loss function** which summarizes the errors produced by the model.
 - If we evaluate the model using the training set, we get the training error, or the **empirical risk**.
 - The test error is often called the **expected risk**.

Training Error and Test Error

What we are really interested in is good accuracy on unseen data.

- In practice, we split the data into two subsets: a training set and a test set.
- We learn a model using the training set, and evaluate its accuracy on the test set.
 - The accuracy of the model is assessed using a **loss function** which summarizes the errors produced by the model.
 - If we evaluate the model using the training set, we get the training error, or the **empirical risk**.
 - The test error is often called the **expected risk**.
- The number of examples for which the training error and test error start converging toward each other is called the **capacity** of the model.

Capacity

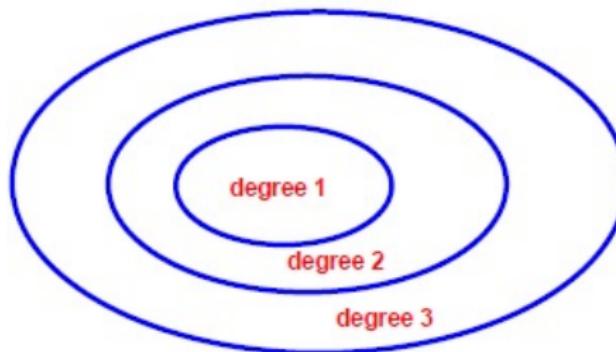
Capacity is defined as the largest n such that there exist a set of examples D_n such that one can always find an f which gives the correct answer for all examples in D_n , for any possible labeling.

- Example: for the set of linear functions ($y = wx + b$) in d dimensions, the capacity is $d + 1$.

Capacity

In many situations we can construct a sequence of models of increasing capacity.

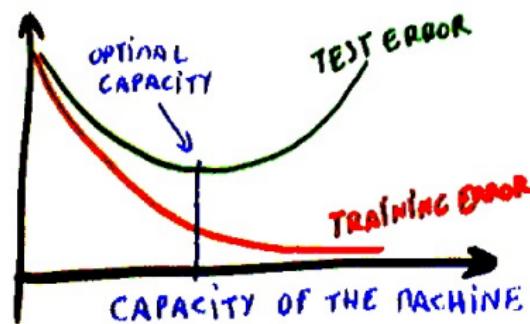
- Capacity is related to the size of the model.
 - The depth of a decision tree.
- Capacity is related to when the training error is a good approximation for the test error.



Model Selection

The curve of training error and test error as a function of the capacity of the model has a minimum.

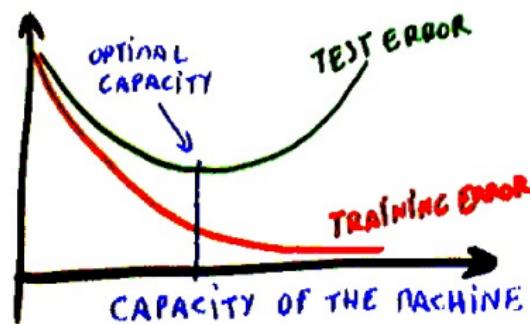
- This is the optimal size for the model.



Model Selection

The curve of training error and test error as a function of the capacity of the model has a minimum.

- This is the optimal size for the model.

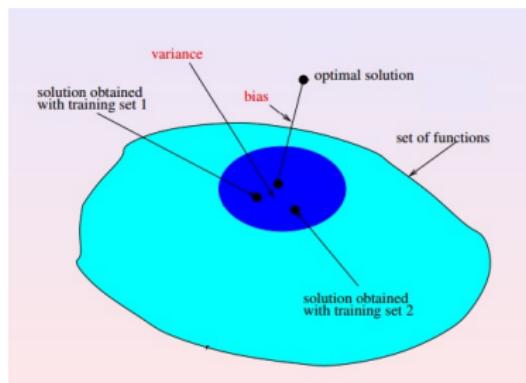


We want to minimize the test error.

- But we cannot assess it, we can only assess the training error.
- Underfitting: capacity is low. Overfitting: capacity is large.

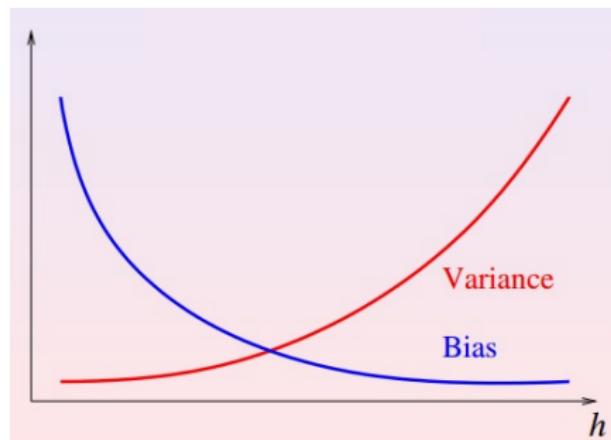
Bias and Variance

- Bias:
 - Error due to the fact that the set of functions does not contain the target function.
- Variance:
 - Error due to the fact that if we had been using another training set drawn from the same distribution, we would have obtained another function.



Bias and Variance

- As the capacity grows:
 - Bias goes down.
 - Variance goes up.



Bias and Variance

Learning is searching in a set of functions H .

- Underfitting: H is too small.
- Overfitting: H is too large.

Bias and Variance

Learning is searching in a set of functions H .

- Underfitting: H is too small.
- Overfitting: H is too large.

We can decompose E_{out} into:

- How well H can approximate the target function f .
- How well we can zoom in on a good $h \in H$.

Bias and Variance

Learning is searching in a set of functions H .

- Underfitting: H is too small.
- Overfitting: H is too large.

We can decompose E_{out} into:

- How well H can approximate the target function f .
- How well we can zoom in on a good $h \in H$.

Regularization: trading-off accuracy for simplicity.

- Minimizes the training error, as long as it is still a good approximation for the test error.
 - Considering only simple models provides good approximation, but the training error is high.
 - Considering only complex models provides low training error, but the approximation is not good.

Regularization

Occam's Razor.

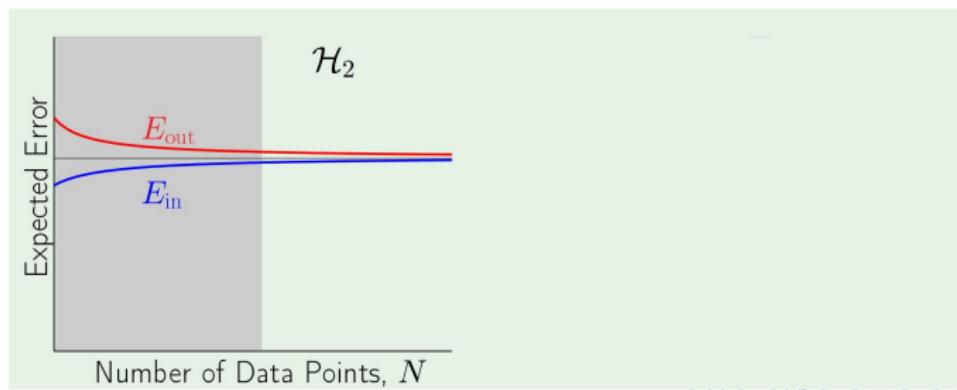
- When given the choice between several models that explain the data equally well, choose the “simplest” one.
- Choose a trade off between how well the model fits the training set and how “simple” that model is.

Regularization

Occam's Razor.

- When given the choice between several models that explain the data equally well, choose the “simplest” one.
- Choose a trade off between how well the model fits the training set and how “simple” that model is.

Overfitting ($f \in H_{10}$).

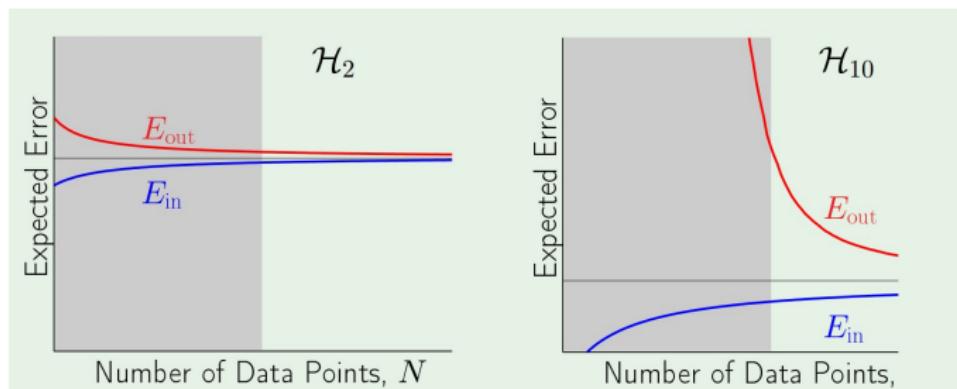


Regularization

Occam's Razor.

- When given the choice between several models that explain the data equally well, choose the “simplest” one.
- Choose a trade off between how well the model fits the training set and how “simple” that model is.

Overfitting ($f \in \mathcal{H}_{10}$).



Regularization

Back to decision trees.

- ① Fully-grown tree: no training error, $E_{in} = 0$.
 - But test error is probably high, since leaf nodes are composed by few training examples.
- ② Need a regularizer, say $\Omega(T) =$ the number of leaves in T .
- ③ Regularized decision tree:

$$\operatorname{argmin}_T E_{in}(T) + \lambda \Omega(T), \text{ for all possible } T$$

- λ is a parameter which trades training error for simplicity.

Regularization

The principle is applicable to any learning algorithm.

- Minimizes a loss function.
- Penalizes complex models.

Regularization

The principle is applicable to any learning algorithm.

- Minimizes a loss function.
- Penalizes complex models.
- Let's pick a measure of the "complexity" of model W : $C(W)$.
Then, we minimize:

$$\mathcal{L} = \frac{1}{n} \left(\sum_{i=1}^{i=n} L(W, y^i, X^i) + \lambda C(W) \right)$$

where L is a conventional loss function (i.e., accuracy) and λ is a well chosen positive constant.

- How we pick the regularization term $C(W)$ is entirely up to us. No theory tells us how to build the regularization function.

Induction Principles

As said before, we cannot minimize the test error (expected risk) directly.

- The method we will employ to replace the expected risk by another quantity that we can minimize is called the **induction principle**.

There are two induction principles:

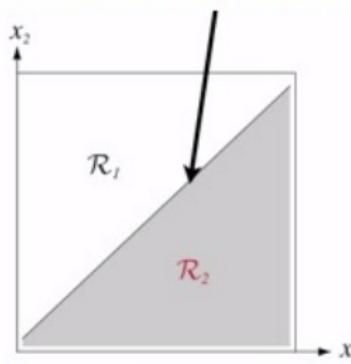
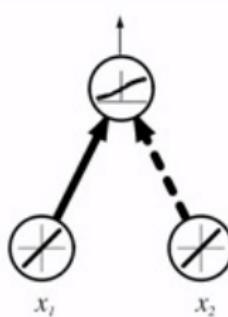
- Empirical Risk Minimization:
 - The simplest principle, which simply consists in minimizing the loss on the training set (training error).
- Structural Risk Minimization:
 - The alternative, which generally consists in using a regularization term to penalize complex models.

1-Layer Neural Networks

Capacity of a single neuron:

- With sigmoid, we can interpret a neuron as estimating $p(y = 1|X)$
 - This is also known as **logistic regression classifier**
- If greater than 0.5, predict class 1. Otherwise, predict class 0.

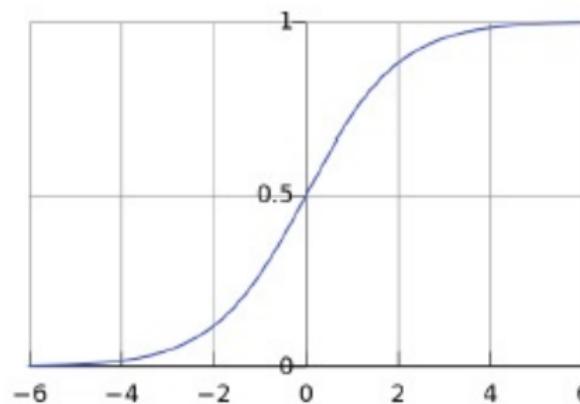
decision boundary is linear



1-Layer Neural Networks

Model: $f(X) = \sigma(W^T X + b)$

- Vector $W \in R^d$, and b is a scalar bias term.
- σ is the sigmoid function.
- For simplicity:
 - Write $f(X) = \sigma(W^T X)$, where $W = [W; b]$ and $X = [X, 1]$



1-Layer Neural Networks

The single-layer neural network algorithm is different from a Perceptron.

- A Perceptron finds weights that are closer to “a good set of weights”.
 - A set of weights forming a boundary that separates the points.

1-Layer Neural Networks

The single-layer neural network algorithm is different from a Perceptron.

- A Perceptron finds weights that are closer to “a good set of weights”.
 - A set of weights forming a boundary that separates the points.
- In contrast, a single-layer neural network finds the weights so that the predicted output always get closer to the desired output.
 - This can work even for non-convex problems.

1-Layer Neural Networks

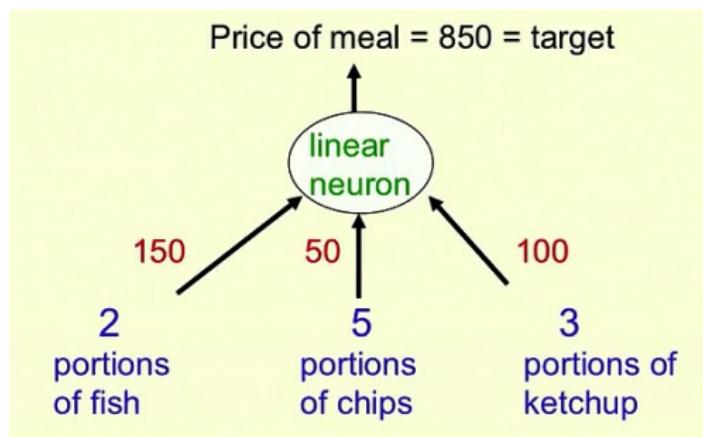
The fish-chips-ketchup example (from Hinton's slides).

- Each day you get lunch in the cafeteria.
 - Your diet consists of fish, chips, and ketchup.
 - You get several portions of each.
- The cashier only tells you the total price of the meal
 - After several days, you should be able to figure out the price of each portion.
- The iterative approach:
 - ① Start with random guesses for the prices.
 - ② Adjust them to get a better fit to the observed prices of the meal.

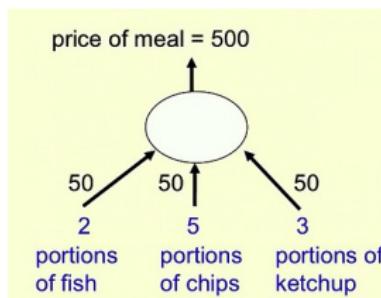
1-Layer Neural Networks

- Each meal price gives a linear constraint on the prices of the portions:
 - $price = x_{fish} \times w_{fish} + x_{chips} \times w_{chips} + x_{ketchup} \times w_{ketchup}$The prices of the portions are like the weights of a linear neuron.
- $W = \{w_{fish}, w_{chips}, w_{ketchup}\}$
- We will start with guesses for the weights and then adjust the guesses slightly to give a better fit to the prices given by the cashier.

1-Layer Neural Networks



1-Layer Neural Networks



- Residual error is 350
- Adjust the weights: $\Delta w_i = r \times x_i(t - y)$
- If $r = \frac{1}{35}$, the weights changes are
 - +20, +50, +30
- This gives new weights of:
 - 70, 100, 80
 - Notice that the weight (price) for chips got worse.

1-Layer Neural Networks

Training 1-layer nets: $z = W^T X^{(i)} + b$.

- Assume squared-error loss:

$$L(W) = \frac{1}{2} \sum_i^m (\sigma(W^T X^{(i)}) - y^{(i)})^2$$

1-Layer Neural Networks

Training 1-layer nets: $z = W^T X^{(i)} + b$.

- Assume squared-error loss:

$$L(W) = \frac{1}{2} \sum_i^m (\sigma(W^T X^{(i)}) - y^{(i)})^2$$

- Gradient: $\nabla_W L$ (minimize $L(W)$ with respect to W)
 - To minimize $L(W)$, find the gradient and adjust W
 - Calculate $\nabla_W L = \frac{\partial L(W)}{\partial W} = \sum \frac{\partial \sigma(W^T X^{(i)})}{\partial W} \frac{\partial L(W)}{\partial \sigma(W^T X^{(i)})}$
 - $\frac{\partial \sigma(W^T X^{(i)})}{\partial W} = \frac{\partial z}{\partial W} \frac{\sigma(W^T X^{(i)})}{\partial z} = X^{(i)} \sigma'(W^T X^{(i)})$
 - $\frac{\partial L(W)}{\partial \sigma(W^T X^{(i)})} = \sigma(W^T X^{(i)}) - y^{(i)}$

$$\nabla_W L = \sum_i^m [\sigma(W^T X^{(i)}) - y^{(i)}] \sigma'(W^T X^{(i)}) X^{(i)} \text{ (chain rule)}$$

1-Layer Neural Networks

Derivative of sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$

- $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

1-Layer Neural Networks

Derivative of sigmoid $\sigma(z) = \frac{1}{1+\exp(-z)}$

- $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Thus, the gradient becomes:

$$\nabla_W L = \sum_i^m [\sigma(W^T X^{(i)}) - y^{(i)}] \sigma(W^T X^{(i)})(1 - \sigma(W^T X^{(i)})) X^{(i)}$$

- Delta rule: $(\sigma(W^T X^{(i)}) - y^{(i)}) X^{(i)}$
- Slope of logistic: $\sigma(W^T X^{(i)})(1 - \sigma(W^T X^{(i)}))$

1-Layer Neural Networks

Training 1-layer nets.

- Gradient Descent algorithm:
 - Initialize W

1-Layer Neural Networks

Training 1-layer nets.

- Gradient Descent algorithm:
 - Initialize W
 - Compute $\nabla_W L$

1-Layer Neural Networks

Training 1-layer nets.

- Gradient Descent algorithm:

- 1 Initialize W
- 2 Compute $\nabla_W L$
- 3 $W \leftarrow W - r \times (\nabla_W L) = \sum_i Error(i) \times \sigma'(W^T X^{(i)}) \times X^{(i)}$

1-Layer Neural Networks

Training 1-layer nets.

- Gradient Descent algorithm:

- Initialize W
- Compute $\nabla_W L$
- $W \leftarrow W - r \times (\nabla_W L) = \sum_i Error(i) \times \sigma'(W^T X^{(i)}) \times X^{(i)}$
- Repeat steps 2 and 3 until some condition is satisfied

1-Layer Neural Networks

Training 1-layer nets.

- Gradient Descent algorithm:
 - ① Initialize W
 - ② Compute $\nabla_W L$
 - ③ $W \leftarrow W - r \times (\nabla_W L) = \sum_i Error(i) \times \sigma'(W^T X^{(i)}) \times X^{(i)}$
 - ④ Repeat steps 2 and 3 until some condition is satisfied
- You should plot $L(W)$ after each iteration:
 - If $L(W)$ is converging = learning rate is ok
 - If $L(W)$ is diverging = learning rate is too large
- If the learning rate is too small = slow to converge
- Stopping condition: W does not change anymore (10^{-5})
- Needs feature scaling: $x'_i = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$
- Step 3 is too expensive for large training sets.

1-Layer Neural Networks

Training 1-layer nets.

- Stochastic Gradient Descent algorithm:
 - ① Randomly shuffle the training set
 - ② Initialize W
 - ③ For each example $(X^{(i)}, y^{(i)})$ in the training set:
 - ④ $W \leftarrow W - r \times (\text{Error}(i) \times \sigma'(W^T X^{(i)}) \times X^{(i)})$
 - ⑤ Repeat steps 2 and 3 until some condition is satisfied
- Convergence is not so obvious. Plot $L(W)$ after 1,000 iterations:
 - If $L(W)$ is converging = learning rate is ok
 - If $L(W)$ is diverging = learning rate is too large
- Stops close to the minimum.

1-Layer Neural Networks

Intuition behind SGD update.

- For some example $(X^{(i)}, y^{(i)})$:
 - $W \leftarrow W - r \times ((\sigma(W^T X^{(i)})) - y^{(i)}) \times \sigma'(W^T X^{(i)}) \times X^{(i)}$

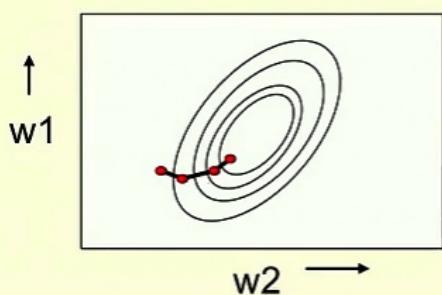
Table: Possible updates

$\sigma(W^T X^{(i)})$	$y^{(i)}$	Error $^{(i)}$	new W	new prediction
0	0	0	no change	0
1	1	0	no change	1
0	1	-1	$W + r \times \sigma'(W^T X^{(i)}) \times X^{(i)}$	≥ 0
1	0	+1	$W - r \times \sigma'(W^T X^{(i)}) \times X^{(i)}$	≤ 1

- $\sigma'(W^T X^{(i)})$ is near 0 = confident, near 0.25 = uncertain.
- Large r values = aggressive, small r values = conservative.

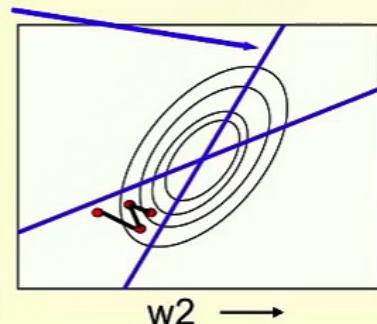
1-Layer Neural Networks

This travels perpendicular to the contour lines.



constraint from training case 1

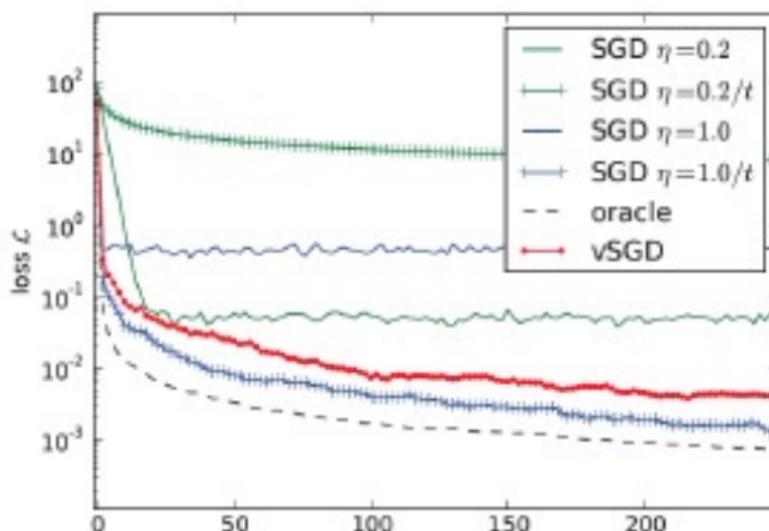
constraint from training case 2



1-Layer Neural Networks

Adaptive learning rate.

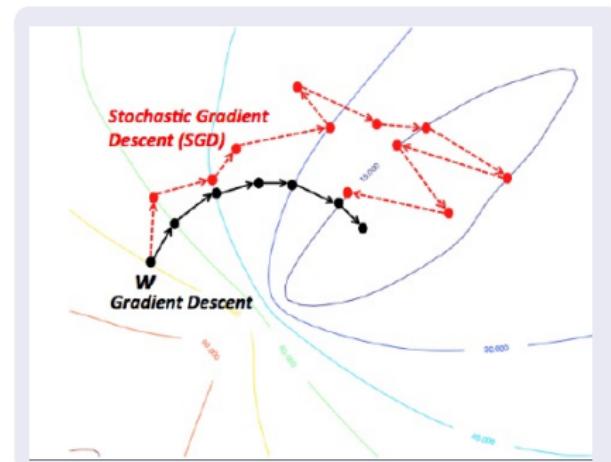
- Decrease r as we see more data.



1-Layer Neural Networks

Geometric view of Stochastic Gradient Descent updates.

- Contour plot: $\frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$
- Gradient descent goes in steepest descent direction, but it is slower to compute.
- SGD can be viewed as noisy descent (different samples may demand different updates), but it is faster.
- A good trade-off is mini-batch SGD.



1-Layer Neural Networks

- (Batch) gradient descent: use all examples in each iteration.
- Stochastic gradient descent: use only one example in each iteration.
- Mini-batch SGD: use b training examples in each iteration
 - b is typically set to 10 examples

$$W \leftarrow W - r \times \frac{1}{b} \sum_i^b Error(i) \times \sigma'(W^T X^{(i)}) \times X^{(i)}$$

- May be even faster than SGD.

1-Layer Neural Networks

Regularization.

- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!

1-Layer Neural Networks

Regularization.

- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!
- Regularization:

1-Layer Neural Networks

Regularization.

- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!
- Regularization: $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2 + ||W||$
 - If the weights are large, the loss is big (weight decay penalty).
 - Regularization prevents large weights.

1-Layer Neural Networks

Regularization.

- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!
- Regularization: $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2 + ||W||$
 - If the weights are large, the loss is big (weight decay penalty).
 - Regularization prevents large weights.
 - If the weights are large, the outputs will be very sensitive to small changes in X .
 - Similar inputs will be associated with different outputs.

1-Layer Neural Networks

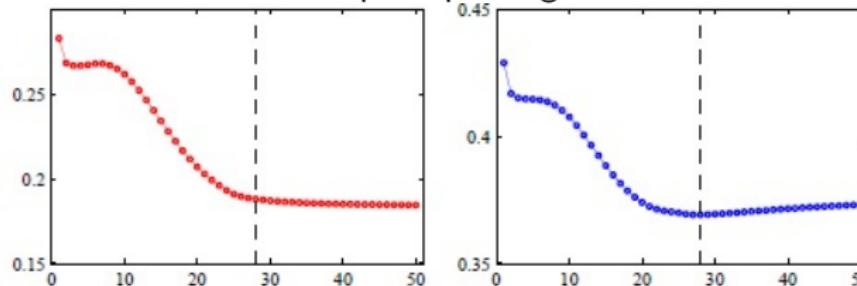
Early stopping.

- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!

1-Layer Neural Networks

Early stopping.

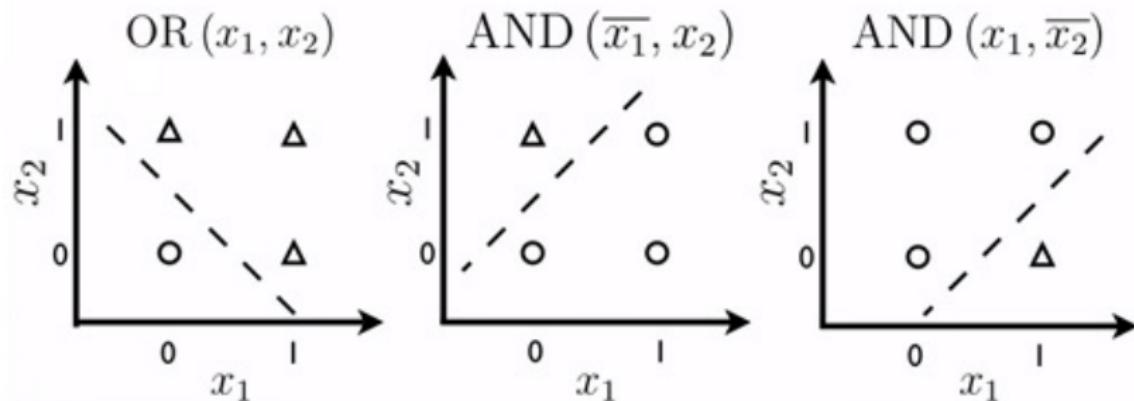
- Minimize $L(W) = \frac{1}{2} \sum_i (\sigma(W^T X^{(i)}) - y^{(i)})^2$ on training set
 - Not necessarily leads to generalization. Recall decision trees!
- Early stopping (cross-validation):
 - Separate the data into training and validation sets.
 - Minimize $L(W)$ on the training set, but stops when $L(W)$ on the validation set stops improving.



1-Layer Neural Networks

Capacity of a single neuron:

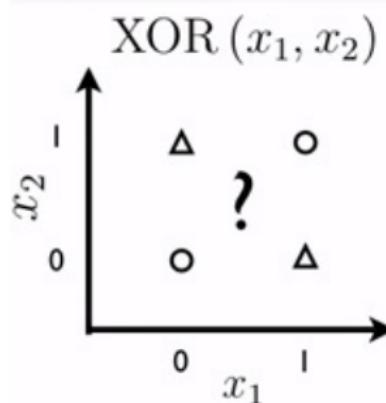
- Can solve linearly separable problems.



1-Layer Neural Networks

Capacity of a single neuron:

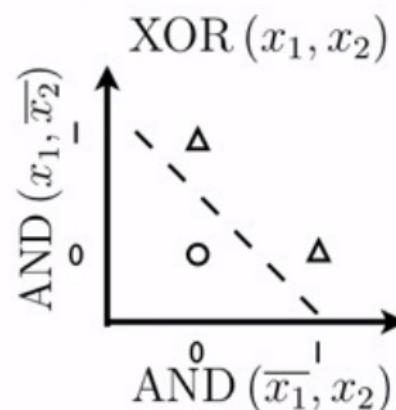
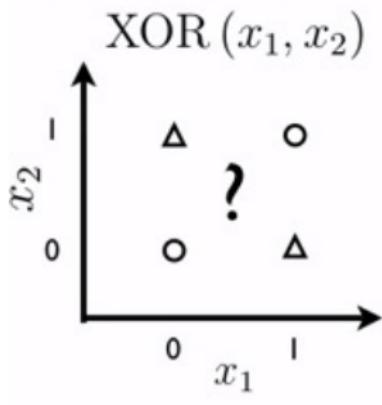
- Cannot solve non linearly separable problems.



1-Layer Neural Networks

Capacity of a single neuron:

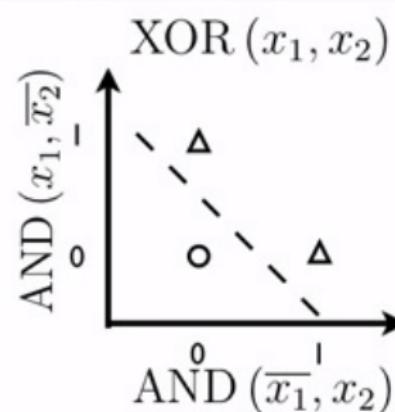
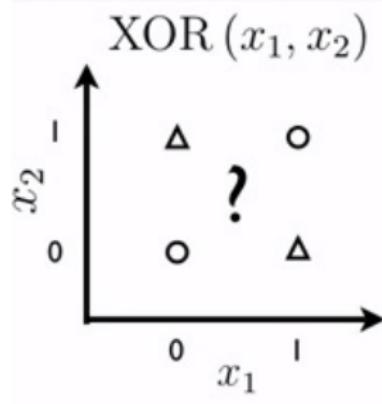
- Cannot solve non linearly separable problems.



2-Layer Neural Networks

Multilayer neural network.

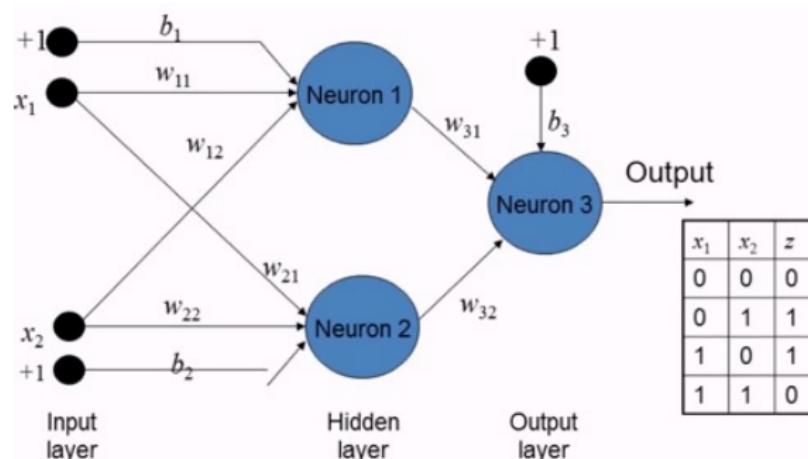
- We can employ two neurons (ANDs), connected to a third one (XOR).
 - This is the inspiration behind multilayer neural network.



2-Layer Neural Networks

Multilayer neural network.

- Input layer with two units.
- One hidden layer with two units.
- Output layer with one unit.



2-Layer Neural Networks

Multilayer neural network.

- Verify that the network solves the XOR problem with the following weights and biases.
 - $w_{11} = w_{12} = w_{21} = w_{22} = w_{32} = +1, w_{31} = -2$
 - $b_1 = -1.5, b_2 = b_3 = -0.5$
- Assume that the activation function $\phi(v)$ is:

$$\phi(v) = \begin{cases} 1, & \text{if } v \geq 0. \\ 0, & \text{otherwise.} \end{cases}$$

2-Layer Neural Networks

$w_{11} = w_{12} = w_{21} = w_{22} = w_{32} = +1$, $w_{31} = -2$, $b_1 = -1.5$ and $b_2 = b_3 = -0.5$

For [0,1] input:

$$w_H(n) = \begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \quad x(n) = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

$$y^H(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} -1.5 & -0.5 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -1.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right] = \varphi \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} y_1^H \\ y_2^H \end{bmatrix}$$

$$y^o(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} b_3 \\ w_{31} \\ w_{32} \end{bmatrix}^T \begin{bmatrix} 1 \\ y_1^H \\ y_2^H \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -0.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right] = \varphi [0.5] = [1]$$

2-Layer Neural Networks

$w_{11} = w_{12} = w_{21} = w_{22} = w_{32} = +1$, $w_{31} = -2$, $b_1 = -1.5$ and $b_2 = b_3 = -0.5$

For [0,0] input:

$$w_H(n) = \begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \quad x(n) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$y^H(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} -1.5 & -0.5 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -1.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right] = \varphi \begin{bmatrix} -1.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ y_2^H \end{bmatrix}$$

$$y^o(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} b_3 \\ w_{31} \\ w_{32} \end{bmatrix}^T \begin{bmatrix} 1 \\ y_1^H \\ y_2^H \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -0.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right] = \varphi[-0.5] = [0]$$

2-Layer Neural Networks

$w_{11} = w_{12} = w_{21} = w_{22} = w_{32} = +1$, $w_{31} = -2$, $b_1 = -1.5$ and $b_2 = b_3 = -0.5$

For [1,0] input:

$$w_H(n) = \begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \quad x(n) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

$$y^H(n) = \varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right] = \varphi\left[\begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right] = \varphi\left[\begin{bmatrix} -1.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right] = \varphi\left[\begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}\right] = \begin{bmatrix} y_1^H \\ y_2^H \end{bmatrix}$$

$$y^\circ(n) = \varphi\left[\mathbf{w}^T(n)\mathbf{x}(n)\right] = \varphi\left[\begin{bmatrix} b_3 \\ w_{31} \\ w_{32} \end{bmatrix}^T \begin{bmatrix} 1 \\ y_1^H \\ y_2^H \end{bmatrix}\right] = \varphi\left[\begin{bmatrix} -0.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}\right] = \varphi[0.5] = [1]$$



2-Layer Neural Networks

$w_{11} = w_{12} = w_{21} = w_{22} = w_{32} = +1$, $w_{31} = -2$, $b_1 = -1.5$ and $b_2 = b_3 = -0.5$

For [1,1] input:

$$w_H(n) = \begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \quad x(n) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

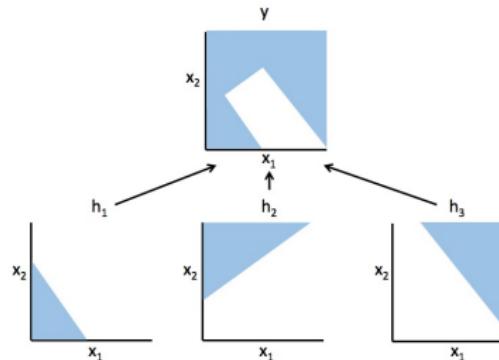
$$y^H(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} -1.5 & -0.5 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -1.5 & 1 & 1 \\ -0.5 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right] = \varphi \begin{bmatrix} 0.5 \\ 1.5 \end{bmatrix} = \begin{bmatrix} y_1^H \\ y_2^H \end{bmatrix}$$

$$y^o(n) = \varphi \left[\mathbf{w}^T(n) \mathbf{x}(n) \right] = \varphi \left[\begin{bmatrix} b_3 \\ w_{31} \\ w_{32} \end{bmatrix}^T \begin{bmatrix} 1 \\ y_1^H \\ y_2^H \end{bmatrix} \right] = \varphi \left[\begin{bmatrix} -0.5 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right] = \varphi[-1.5] = [0]$$

2-Layer Neural Networks

Modeling complex non-linearities.

- 1-layer nets only model linear hyperplanes.
- 2-layer nets are universal function approximators.
 - Given infinite number of hidden units, it can express any continuous function.
- ≥ 3 -layer nets can do so with fewer nodes/weights.
 - Faster and less prone to overfitting.

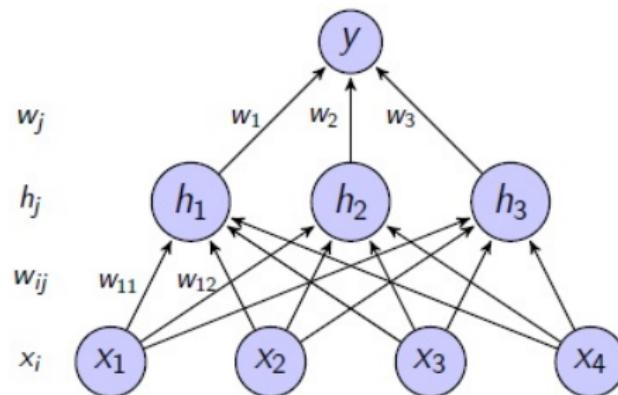


2-Layer Neural Networks

Networks without hidden units are very limited in the input-output mappings they can model.

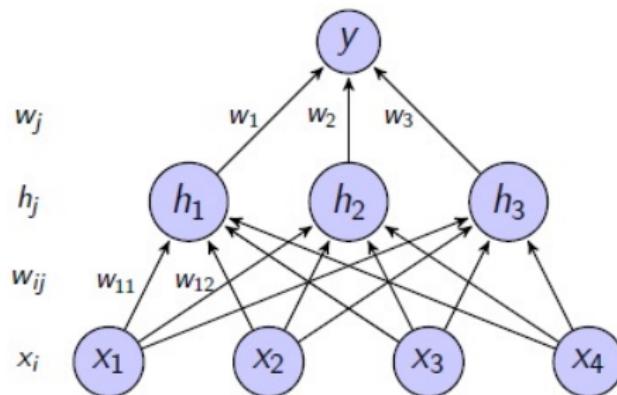
- Learning by perturbing weights (i.e., like mutations).
 - Randomly perturb one weight and see if it improves the performance. If so, keep the change.
 - Very inefficient. Several forward passes on the training examples just to change one weight.
 - Further, weights need to have relative values. They cannot be evaluated independently.

2-Layer Neural Networks



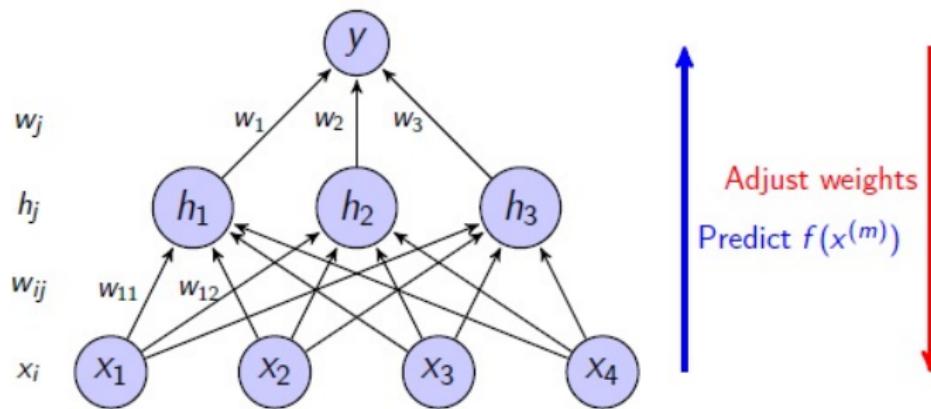
- $f(X) = \sigma\left(\sum_j w_j \times h_j\right) = \sigma\left(\sum_j w_j \times \sigma\left(\sum_i w_{ij} \times x_i\right)\right)$

2-Layer Neural Networks



- $f(X) = \sigma\left(\sum_j w_j \times h_j\right) = \sigma\left(\sum_j w_j \times \sigma\left(\sum_i w_{ij} \times x_i\right)\right)$
- Hidden units h_j 's can be viewed as new “features” from combining x_i 's (i.e., multiple logistic regressions).

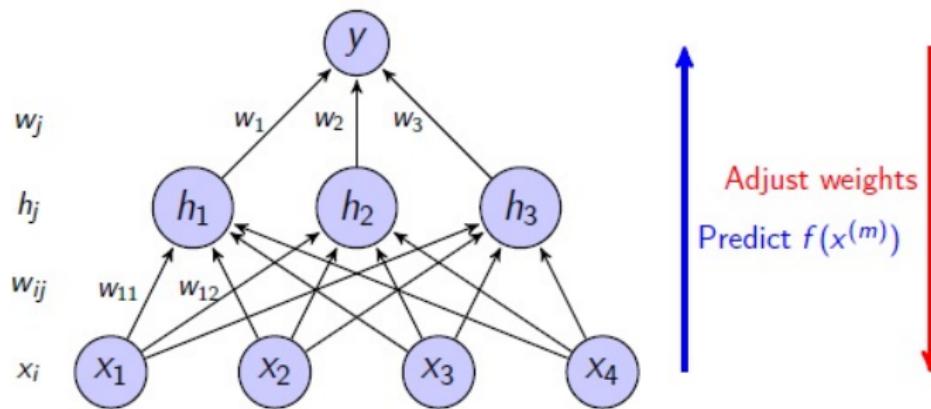
2-Layer Neural Networks



- ① For each example $X^{(m)}$:

$$\text{Compute } f(X^{(m)}) = \sigma\left(\sum_j w_j \times \sigma\left(\sum_i w_{ij} \times X_i^{(m)}\right)\right)$$

2-Layer Neural Networks



- ➊ For each example $X^{(m)}$:
- ➋ Compute $f(X^{(m)}) = \sigma\left(\sum_j w_j \times \sigma\left(\sum_i w_{ij} \times X_i^{(m)}\right)\right)$
- ➌ If $f(X^{(m)}) \neq y^{(m)}$, back-propagate error and adjust weights.

Backpropagation



Geoffrey Hinton

Professor of Computer Science, University of Toronto
 machine learning, neural networks, artificial intelligence, cognitive science, object
 recognition
 E-mail confirmado em cs.toronto.edu - Página inicial

[Seguir](#)

Google Acadêmico



[Obter meu próprio perfil](#)

Índices de citações	Todos	Desde 2010
Citações	104003	33996
Índice h	107	76
Índice i10	261	181



Título 1–20

Citado por Ano

Parallel distributed processing

DE Rumelhart, JL McClelland, PDP Research Group
 IEEE 1, 354-362

18612 1988

Learning internal representations by error-propagation

DE Rumelhart, GE Hinton, RJ Williams
 Parallel Distributed Processing: Explorations in the Microstructure of ...

17911 1986

Backpropagation

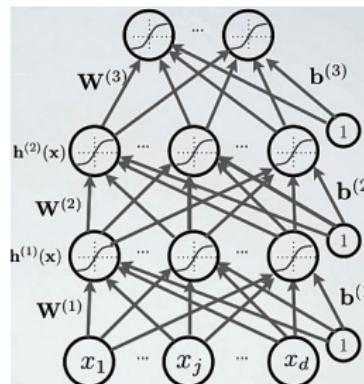
Algorithm:

- ① Randomly initialize the weights
- ② For each example $(x^{(i)}, y^{(i)})$
 - ③ Calculate the error (forward pass)
 - ④ Calculate local gradients for each node
 - ⑤ Update the weights (backward pass)

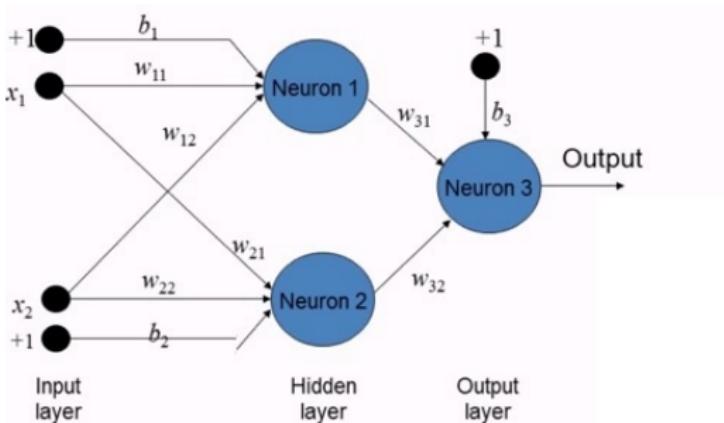
Backpropagation

Algorithm:

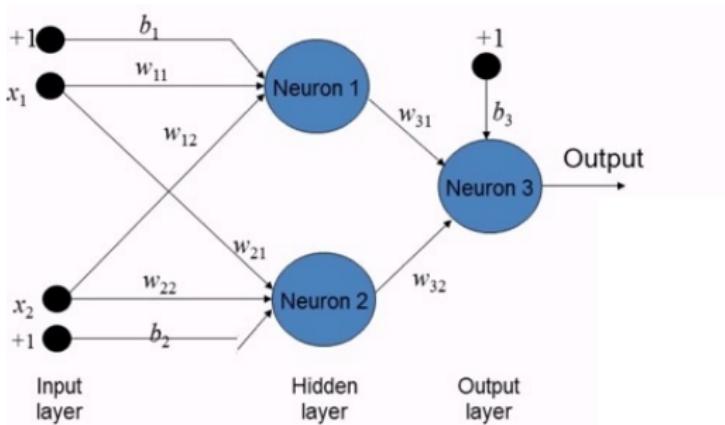
- ① Randomly initialize the weights
- ② For each example $(x^{(i)}, y^{(i)})$
 - ③ Calculate the error (forward pass)
 - ④ Calculate local gradients for each node
 - ⑤ Update the weights (backward pass)



Backpropagation



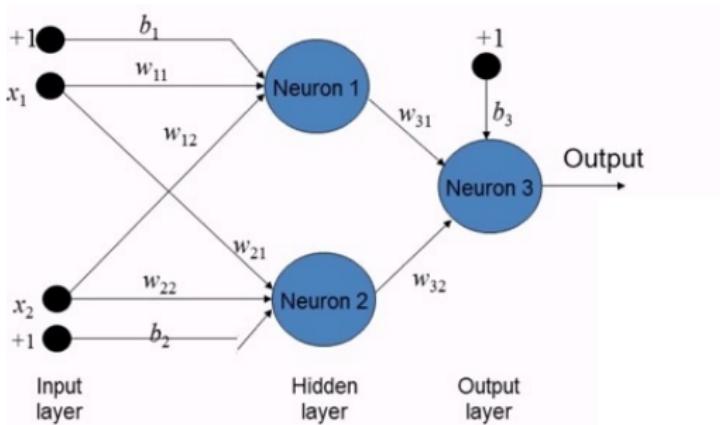
Backpropagation



After initialization:

- $w_{11} = -0.2, w_{12} = 0.1, w_{21} = -0.1, w_{22} = 0.3, w_{31} = 0.2, w_{32} = 0.3$
- $b_1 = 0.1, b_2 = 0.1, b_3 = 0.2$

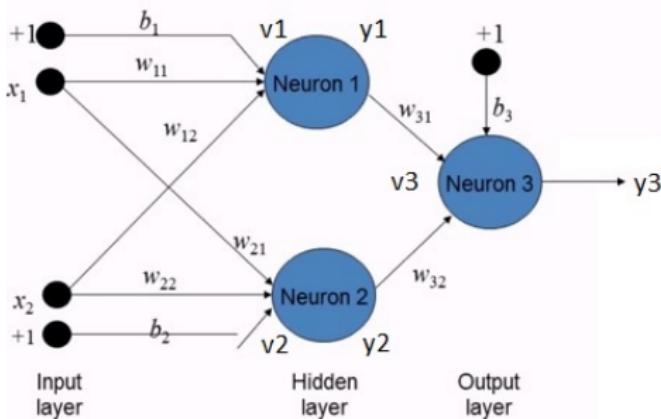
Backpropagation



Assume:

- Target output: $o = 0.9$
- Learning rate $r = 0.25$

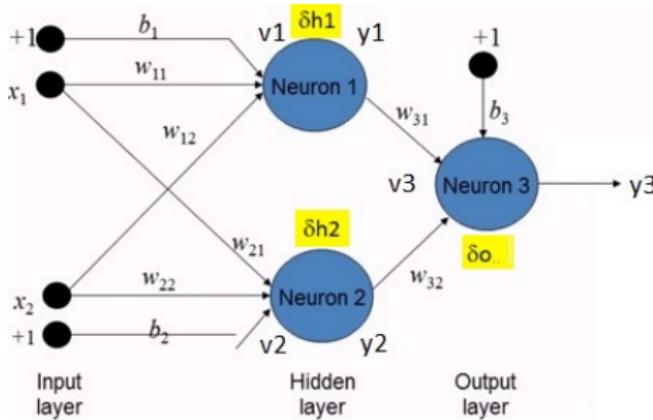
Backpropagation



Assume:

- Activation function: $\phi(v) = \frac{1}{1+\exp(-v)}$
- $\phi'(v) = \phi(v) \times (1 - \phi(v))$

Backpropagation



Local gradients:

- $\delta o = \phi'(1 * b_3 + y_1 \times w_{31} + y_2 \times w_{32}) \times (o - y_3)$
- $\delta h_1 = \phi'(1 * b_1 + x_1 \times w_{11} + x_2 \times w_{21}) \times (\delta o \times w_{31})$
- $\delta h_2 = \phi'(1 * b_2 + x_2 \times w_{12} + x_2 \times w_{22}) \times (\delta o \times w_{32})$

Backpropagation

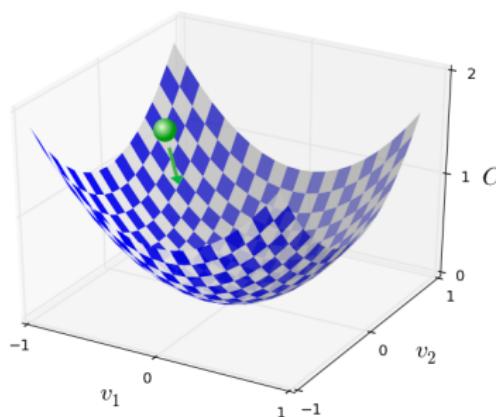
Updating the weights.

- Δ rule: $w_{i+1} = w_i + \gamma \times w_{i-1} + r \times \delta \times x$

Backpropagation

Updating the weights.

- Δ rule: $w_{i+1} = w_i + \gamma \times w_{i-1} + r \times \delta \times x$
- γ is the momentum
 - Prevents local optima



Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$

Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v_1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$

Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v_1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$
- $v_2 = 1 \times b_2 + x_1 \times w_{21} + x_2 \times w_{22} = 1 \times 0.1 + 0.1 \times (-0.1) + 0.9 \times 0.3 = 0.36$

Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v_1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$
- $v_2 = 1 \times b_2 + x_1 \times w_{21} + x_2 \times w_{22} = 1 \times 0.1 + 0.1 \times (-0.1) + 0.9 \times 0.3 = 0.36$
- $y_2 = \phi(v_2) = \phi(0.36) = \frac{1}{1+\exp(-0.36)} = 0.589$

Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v_1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$
- $v_2 = 1 \times b_2 + x_1 \times w_{21} + x_2 \times w_{22} = 1 \times 0.1 + 0.1 \times (-0.1) + 0.9 \times 0.3 = 0.36$
- $y_2 = \phi(v_2) = \phi(0.36) = \frac{1}{1+\exp(-0.36)} = 0.589$
- $v_3 = 1 \times b_3 + y_1 \times w_{31} + y_2 \times w_{32} = 1 \times 0.2 + 0.542 \times 0.2 + 0.589 \times 0.3 = 0.485$

Backpropagation

Forward pass:

- $v_1 = 1 \times b_1 + x_1 \times w_{11} + x_2 \times w_{12} = 1 \times 0.1 + 0.1 \times (-0.2) + 0.9 \times 0.1 = 0.17$
- $y_1 = \phi(v_1) = \phi(0.17) = \frac{1}{1+\exp(-0.17)} = 0.542$
- $v_2 = 1 \times b_2 + x_1 \times w_{21} + x_2 \times w_{22} = 1 \times 0.1 + 0.1 \times (-0.1) + 0.9 \times 0.3 = 0.36$
- $y_2 = \phi(v_2) = \phi(0.36) = \frac{1}{1+\exp(-0.36)} = 0.589$
- $v_3 = 1 \times b_3 + y_1 \times w_{31} + y_2 \times w_{32} = 1 \times 0.2 + 0.542 \times 0.2 + 0.589 \times 0.3 = 0.485$
- $y_3 = \phi(v_3) = \phi(0.485) = \frac{1}{1+\exp(-0.485)} = 0.619$

Thus, $error = (o - y_3) = 0.9 - 0.619 = 0.281$

Backpropagation

Backward pass:

- $\delta o = \phi'(v_3) \times (o - y_3) = \phi'(0.485) \times 0.281 = \phi(0.485) \times (1 - \phi(0.485)) \times 0.281 = 0.619 \times (1 - 0.619) \times 0.281 = 0.0663$

Backpropagation

Backward pass:

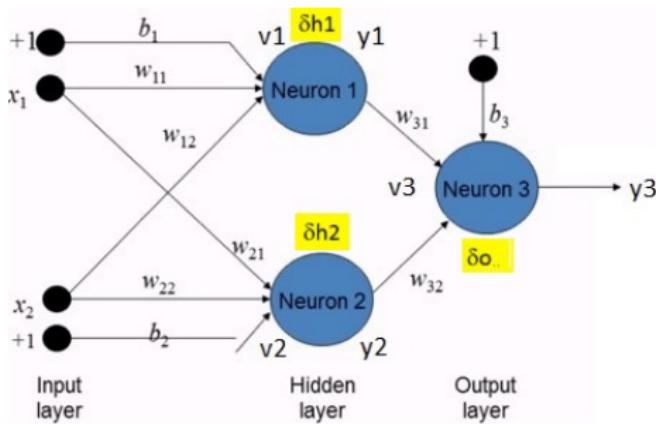
- $\delta o = \phi'(v_3) \times (o - y_3) = \phi'(0.485) \times 0.281 = \phi(0.485) \times (1 - \phi(0.485)) \times 0.281 = 0.619 \times (1 - 0.619) \times 0.281 = 0.0663$
- $\delta h_1 = \phi'(v_1) \times (\delta o \times w_{31}) = \phi'(0.17) \times 0.0663 \times 0.2 = \phi(0.17) \times (1 - \phi(0.17)) \times 0.01362 = 0.542 \times (1 - 0.542) \times 0.01362 = 0.0033$

Backpropagation

Backward pass:

- $\delta o = \phi'(v_3) \times (o - y_3) = \phi'(0.485) \times 0.281 = \phi(0.485) \times (1 - \phi(0.485)) \times 0.281 = 0.619 \times (1 - 0.619) \times 0.281 = 0.0663$
- $\delta h_1 = \phi'(v_1) \times (\delta o \times w_{31}) = \phi'(0.17) \times 0.0663 \times 0.2 = \phi(0.17) \times (1 - \phi(0.17)) \times 0.01362 = 0.542 \times (1 - 0.542) \times 0.01362 = 0.0033$
- $\delta h_2 = \phi'(v_2) \times (\delta o \times w_{32}) = \phi'(0.36) \times 0.0663 \times 0.3 = \phi(0.36) \times (1 - \phi(0.36)) \times 0.01989 = 0.589 \times (1 - 0.589) \times 0.01989 = 0.0049$

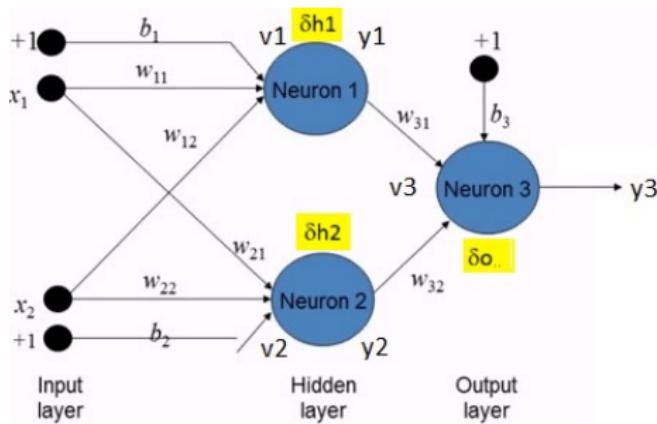
Backpropagation



Update the weights using the Δ rule. Assume $\gamma = 0.0001$

- $w(n+1) = w(n) + \gamma \times w(n-1) + r \times \delta \times x$

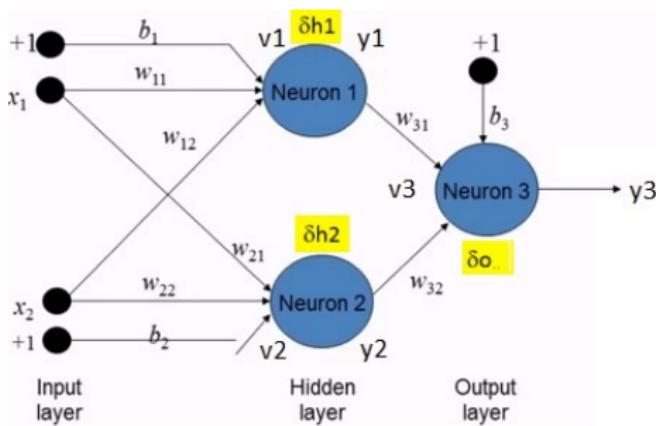
Backpropagation



Update the weights using the Δ rule. Assume $\gamma = 0.0001$

- $w(n+1) = w(n) + \gamma \times w(n-1) + r \times \delta \times x$
- $w_{31}(n+1) = 0.2 + 0.0001 \times 0.2 + 0.25 \times 0.0663 \times 0.542 = 0.2090$

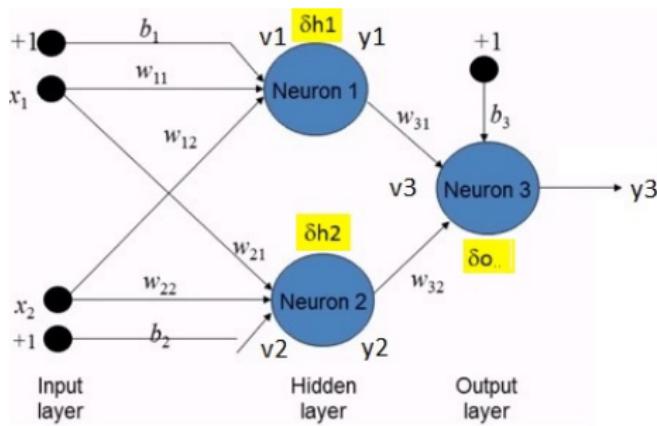
Backpropagation



Update the weights using the Δ rule. Assume $\gamma = 0.0001$

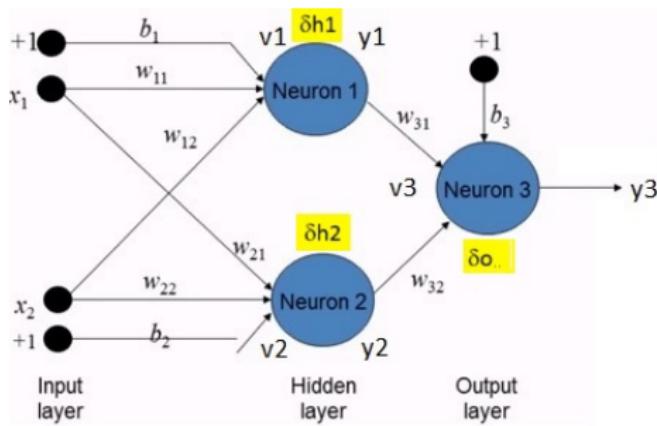
- $w(n+1) = w(n) + \gamma \times w(n-1) + r \times \delta \times x$
- $w_{31}(n+1) = 0.2 + 0.0001 \times 0.2 + 0.25 \times 0.0663 \times 0.542 = 0.2090$
- $w_{32}(n+1) = 0.3 + 0.0001 \times 0.3 + 0.25 \times 0.0663 \times 0.589 = 0.3098$

Backpropagation



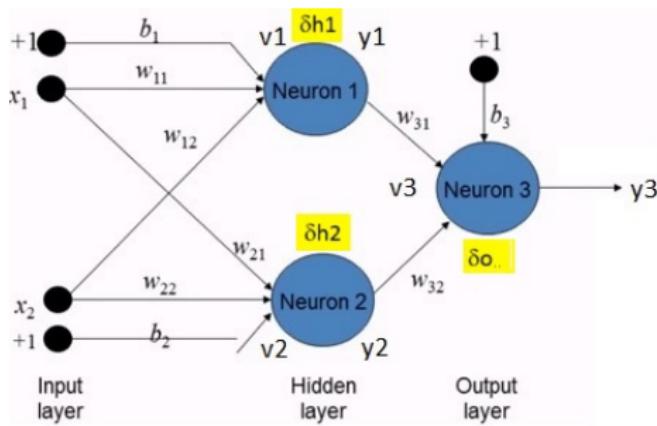
- $w_{11}(n+1) = -0.2 + 0.0001 \times (-0.2) + 0.25 \times 0.0033 \times 0.1 = -0.1999$

Backpropagation



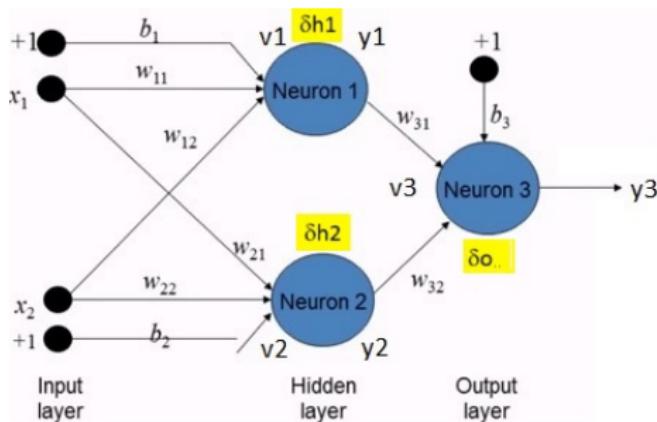
- $w_{11}(n+1) = -0.2 + 0.0001 \times (-0.2) + 0.25 \times 0.0033 \times 0.1 = -0.1999$
- $w_{21}(n+1) = -0.1 + 0.0001 \times (-0.1) + 0.25 \times 0.0049 \times 0.1 = -0.0999$

Backpropagation



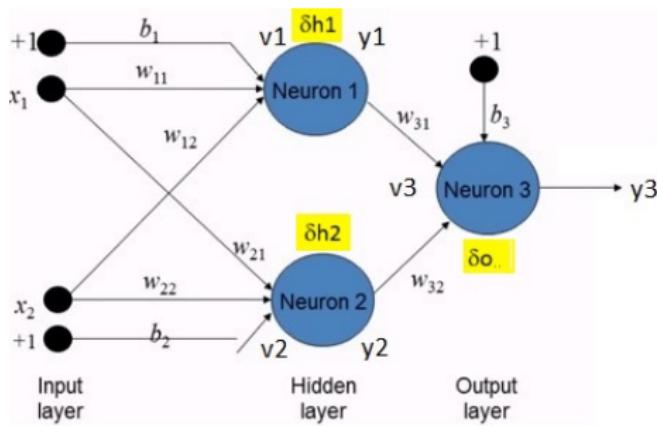
- $w_{11}(n+1) = -0.2 + 0.0001 \times (-0.2) + 0.25 \times 0.0033 \times 0.1 = -0.1999$
- $w_{21}(n+1) = -0.1 + 0.0001 \times (-0.1) + 0.25 \times 0.0049 \times 0.1 = -0.0999$
- $w_{12}(n+1) = 0.1 + 0.0001 \times (0.1) + 0.25 \times 0.0033 \times 0.9 = -0.1008$

Backpropagation



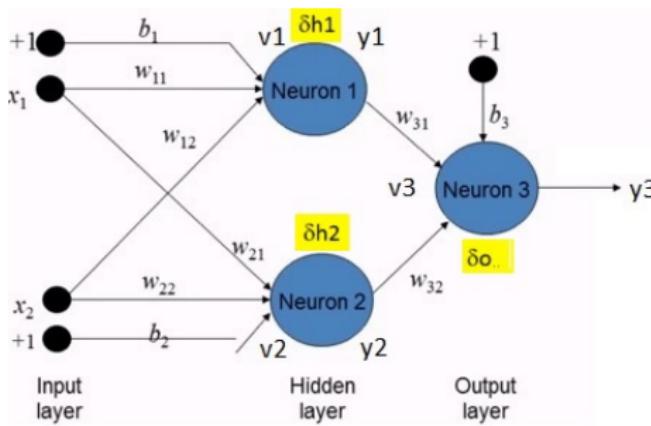
- $w_{11}(n+1) = -0.2 + 0.0001 \times (-0.2) + 0.25 \times 0.0033 \times 0.1 = -0.1999$
- $w_{21}(n+1) = -0.1 + 0.0001 \times (-0.1) + 0.25 \times 0.0049 \times 0.1 = -0.0999$
- $w_{12}(n+1) = 0.1 + 0.0001 \times (0.1) + 0.25 \times 0.0033 \times 0.9 = -0.1008$
- $w_{22}(n+1) = 0.3 + 0.0001 \times (0.3) + 0.25 \times 0.0049 \times 0.9 = -0.3011$

Backpropagation



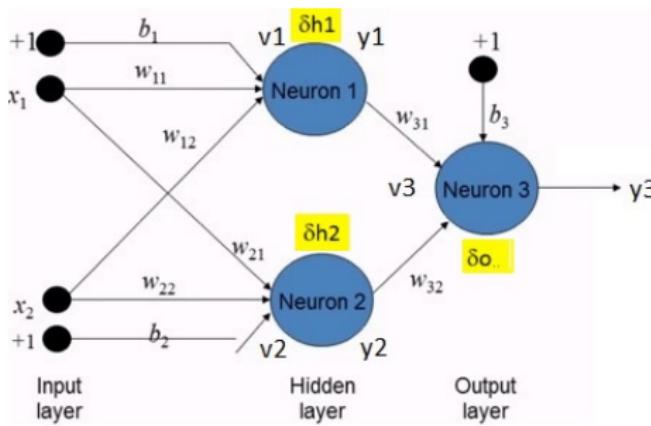
- $b(n+1) = b(n) + \gamma \times b(n-1) + r \times \delta \times 1$
- $b_3(n+1) = 0.2 + 0.0001 \times (0.2) + 0.25 \times 0.0663 \times 1 = 0.2166$

Backpropagation



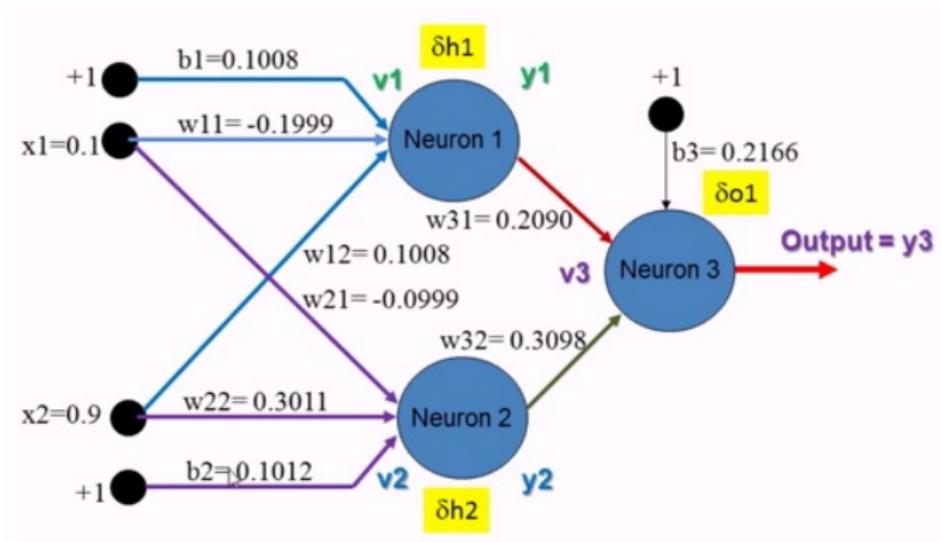
- $b(n+1) = b(n) + \gamma \times b(n-1) + r \times \delta \times 1$
- $b_3(n+1) = 0.2 + 0.0001 \times (0.2) + 0.25 \times 0.0663 \times 1 = 0.2166$
- $b_1(n+1) = 0.1 + 0.0001 \times (0.1) + 0.25 \times 0.0033 \times 1 = 0.1008$

Backpropagation



- $b(n+1) = b(n) + \gamma \times b(n-1) + r \times \delta \times 1$
- $b_3(n+1) = 0.2 + 0.0001 \times (0.2) + 0.25 \times 0.0663 \times 1 = 0.2166$
- $b_1(n+1) = 0.1 + 0.0001 \times (0.1) + 0.25 \times 0.0033 \times 1 = 0.1008$
- $b_2(n+1) = 0.1 + 0.0001 \times (0.1) + 0.25 \times 0.0049 \times 1 = 0.1012$

Backpropagation



Backpropagation

First iteration:

- $v_1 = 0.17 \rightarrow v_1 = 0.1715$
- $y_1 = 0.542 \rightarrow y_1 = 0.5428$
- $v_2 = 0.36 \rightarrow v_2 = 0.3622$
- $y_2 = 0.589 \rightarrow y_2 = 0.5896$
- $v_3 = 0.4851 \rightarrow v_3 = 0.5127$
- $y_3 = 0.619 \rightarrow y_3 = 0.6254$

Backpropagation

First iteration:

- $v_1 = 0.17 \rightarrow v_1 = 0.1715$
- $y_1 = 0.542 \rightarrow y_1 = 0.5428$
- $v_2 = 0.36 \rightarrow v_2 = 0.3622$
- $y_2 = 0.589 \rightarrow y_2 = 0.5896$
- $v_3 = 0.4851 \rightarrow v_3 = 0.5127$
- $y_3 = 0.619 \rightarrow y_3 = 0.6254$
- $error = (o - y_3) = 0.9 - 0.619 = 0.281 \rightarrow error = 0.9 - 0.6254 = 0.2746$

Backpropagation

After a few more passes:

- After second pass: $error = 0.2683$

Backpropagation

After a few more passes:

- After second pass: $\text{error} = 0.2683$
- After third pass: $\text{error} = 0.2623$

Backpropagation

After a few more passes:

- After second pass: $\text{error} = 0.2683$
- After third pass: $\text{error} = 0.2623$
- After fourth pass: $\text{error} = 0.2565$

Backpropagation

After a few more passes:

- After second pass: $\text{error} = 0.2683$
- After third pass: $\text{error} = 0.2623$
- After fourth pass: $\text{error} = 0.2565$
- After 100 passes: $\text{error} = 0.0693$

Backpropagation

After a few more passes:

- After second pass: $\text{error} = 0.2683$
- After third pass: $\text{error} = 0.2623$
- After fourth pass: $\text{error} = 0.2565$
- After 100 passes: $\text{error} = 0.0693$
- After 200 passes: $\text{error} = 0.0319$

Backpropagation

After a few more passes:

- After second pass: $\text{error} = 0.2683$
- After third pass: $\text{error} = 0.2623$
- After fourth pass: $\text{error} = 0.2565$
- After 100 passes: $\text{error} = 0.0693$
- After 200 passes: $\text{error} = 0.0319$
- After 500 passes: $\text{error} = 0.0038$
- Error is getting reduced after each pass.

Backpropagation

Minsky and Papert, again.

Minsky & Papert (1988, Perceptrons, 2nd ed.):

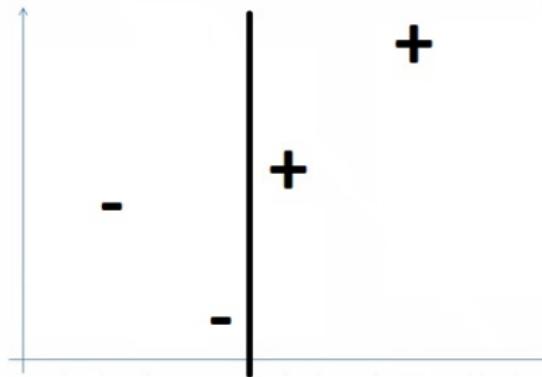
We have the impression that many people in the connectionist community do not understand that this [back-propagation] is merely a particular way to compute a gradient and have assumed instead that back-propagation is a new learning scheme that somehow gets around the basic limitations of hill-climbing.

situation. . . . We fear that its [back-propagation's] reputation also stems from unfamiliarity with the manner in which hill-climbing methods deteriorate when confronted with larger-scale problems. In any case, little good can come from statements like 'as a practical matter, GD leads to solutions in virtually every case' or 'GD can, in principle, learn arbitrary functions'. Such pronouncements are not merely technically wrong; more significantly, the pretense that problems do not exist can deflect us from valuable insights that could come from

Support Vector Machines

Intuition: to separate examples with a straight line.

- We have already seen this before.

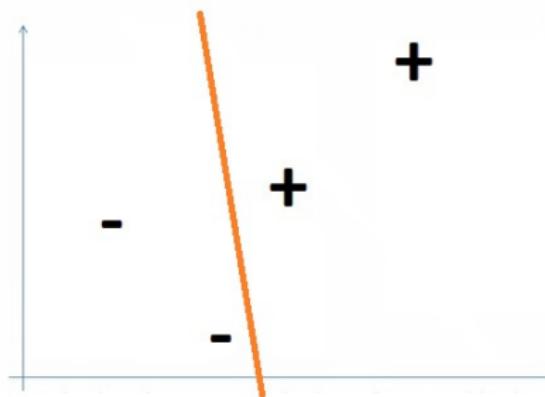


- Decision trees.

Support Vector Machines

Intuition: to separate examples with a straight line.

- We have already seen this before.

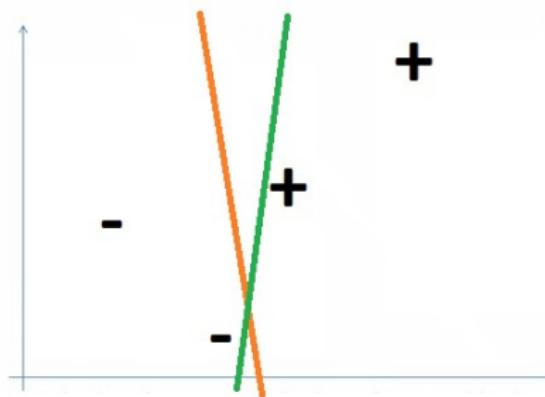


- Perceptron.

Support Vector Machines

Intuition: to separate examples with a straight line.

- We have already seen this before.



- Perceptron.

Support Vector Machines

Intuition: to separate examples with a straight line.

- We have already seen this before.



- Perceptron (depends on the initialization of the weights).

Support Vector Machines

Intuition: to separate examples with a straight line.

- But, which line?
- Are them equally good? (**note that all have no training error**)



vapnik

Professor of Columbia, Fellow of NEC Labs America,
machine learning, statistics, computer science
E-mail confirmado em nec-labs.com

[Seguir](#)

Google Acadêmico



Índices de citações	Todos	Desde 2010
Citações	145040	68836
Índice h	102	69
Índice i10	342	253

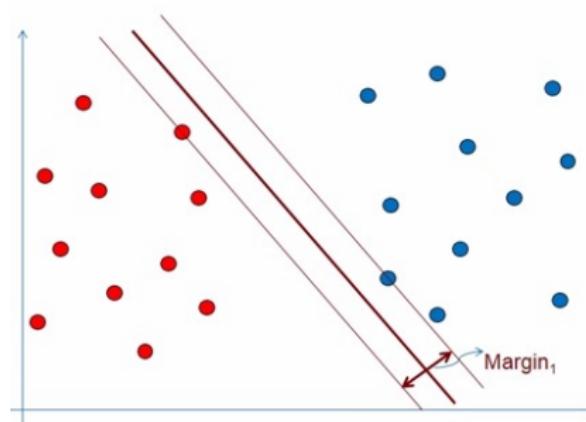


Coautores [Visualizar todos...](#)

Support Vector Machines

Margin:

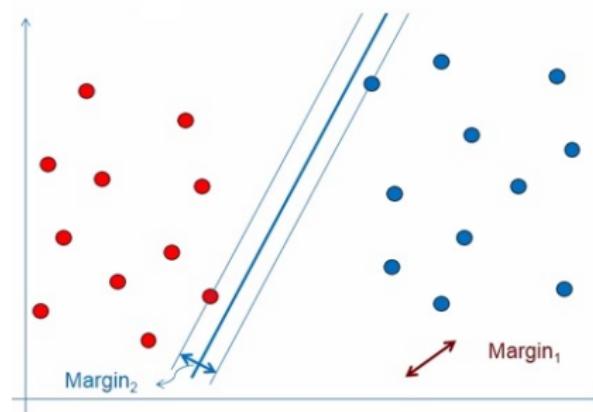
- The width of the street around the decision boundary.
 - No training examples within the street.



Support Vector Machines

Margin:

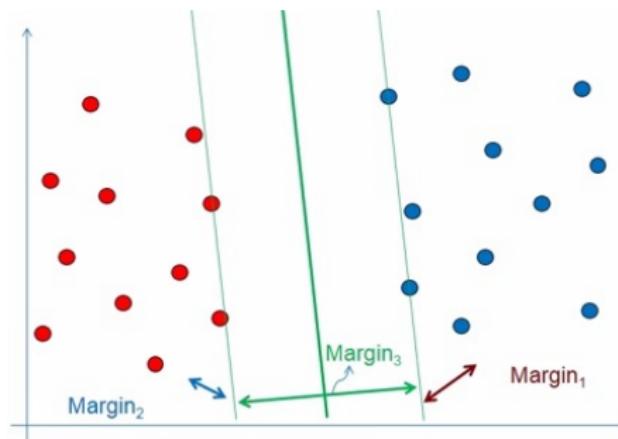
- The width of the street around the decision boundary.
 - No training examples within the street.



Support Vector Machines

Margin:

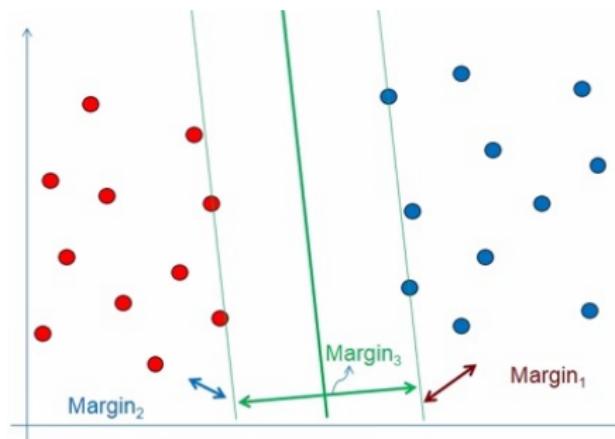
- The width of the street around the decision boundary.
 - No training examples within the street.



Support Vector Machines

Margin:

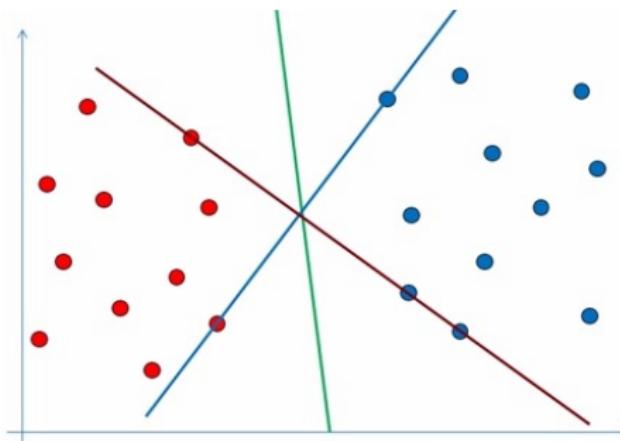
- Is a larger margin better?
- Why?



Support Vector Machines

Margin:

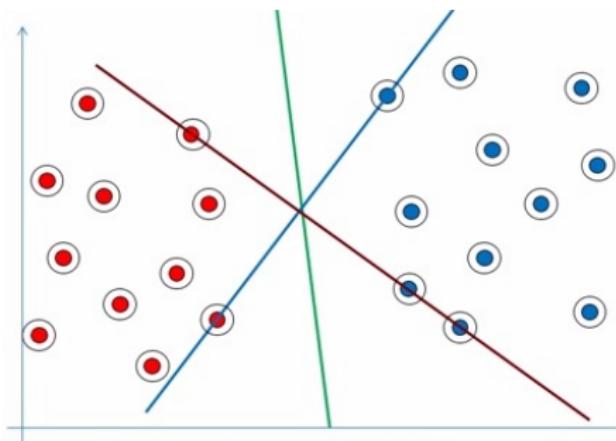
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

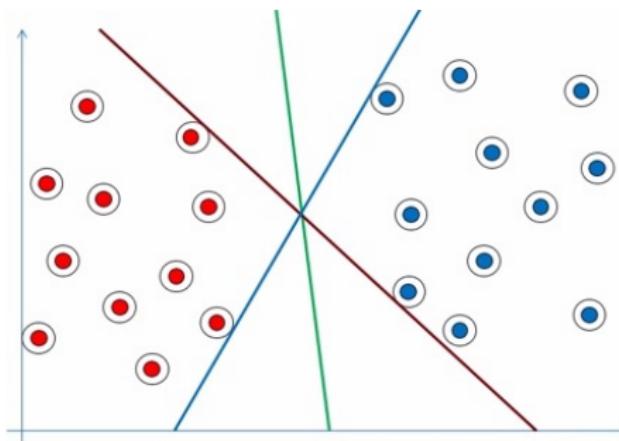
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

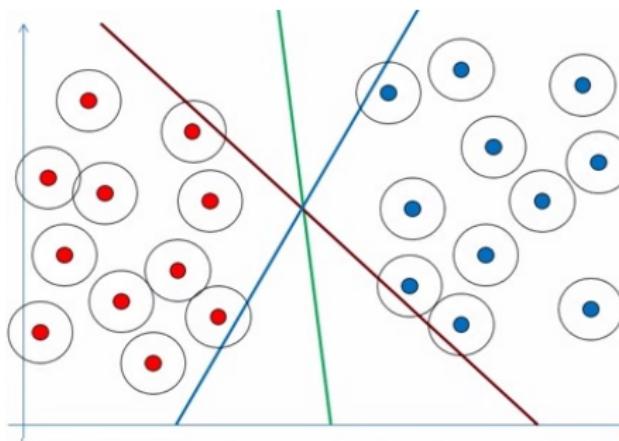
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

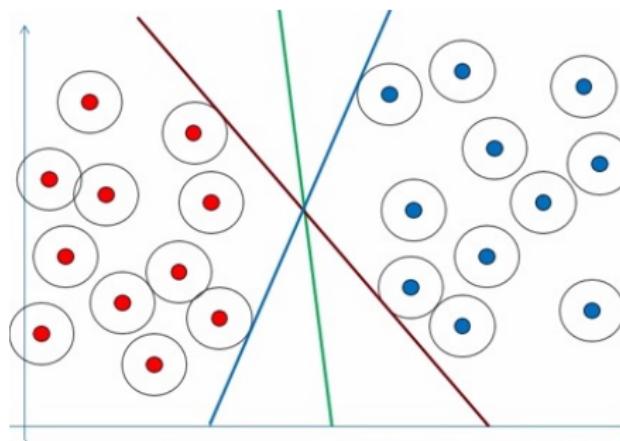
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

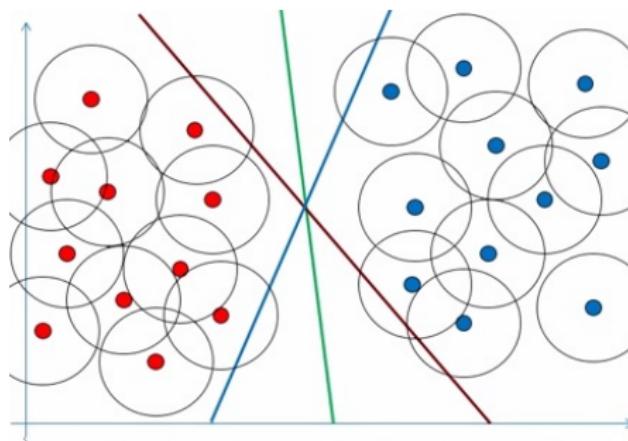
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

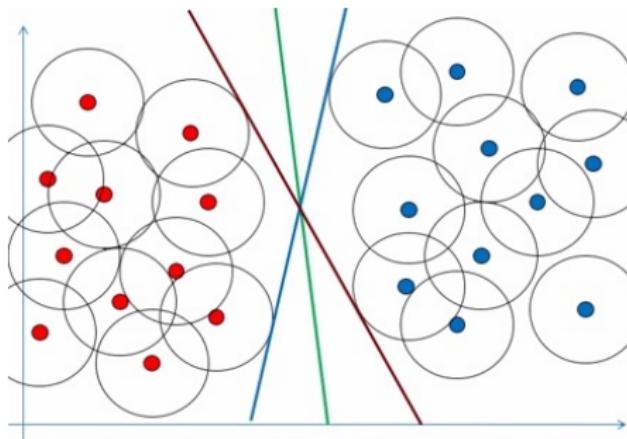
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

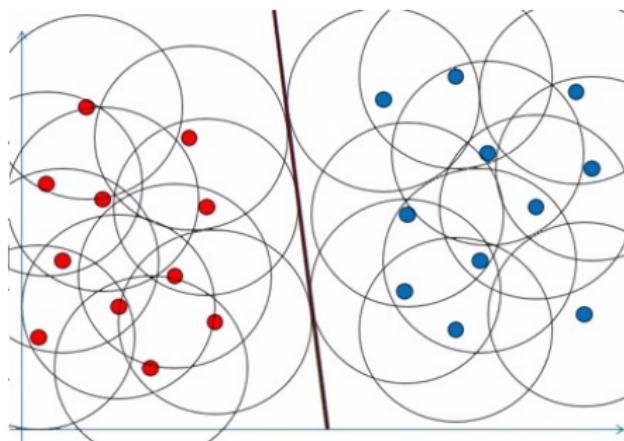
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

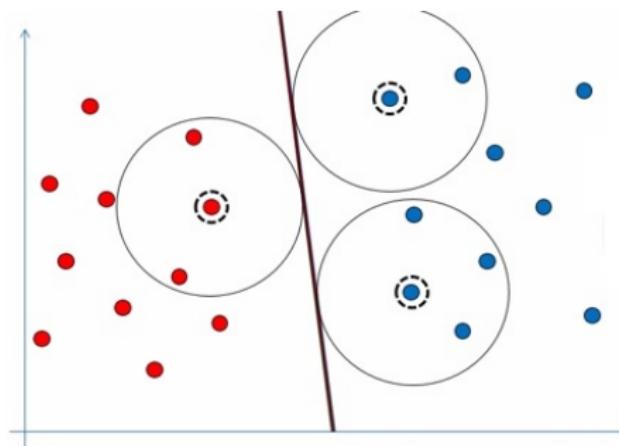
- Radius around each example, through which the decision boundary cannot pass



Support Vector Machines

Margin:

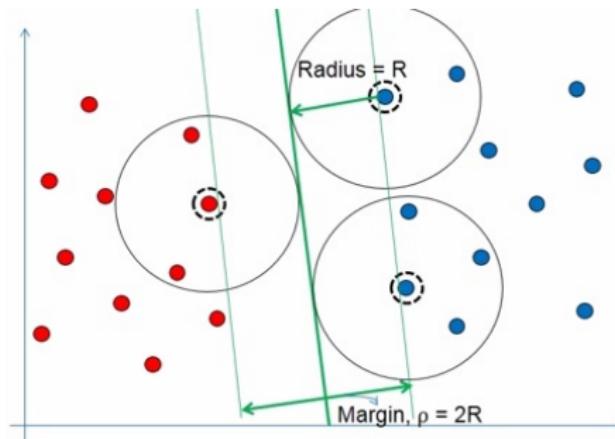
- Not all circles touch the boundary.



Support Vector Machines

Margin:

- Margin $\rho = 2 \times R$.



Support Vector Machines

The VC-dimension.

- Bound on the test error.
 - Based on the training error and the VC-dimension.

$$L(h)_{\text{test}} \leq L(h)_{\text{train}} + O\left(\sqrt{\frac{VC(h)}{n}}\right)$$

- Let us say we have a dataset containing m points/examples.
 - These m points can be labeled in

Support Vector Machines

The VC-dimension.

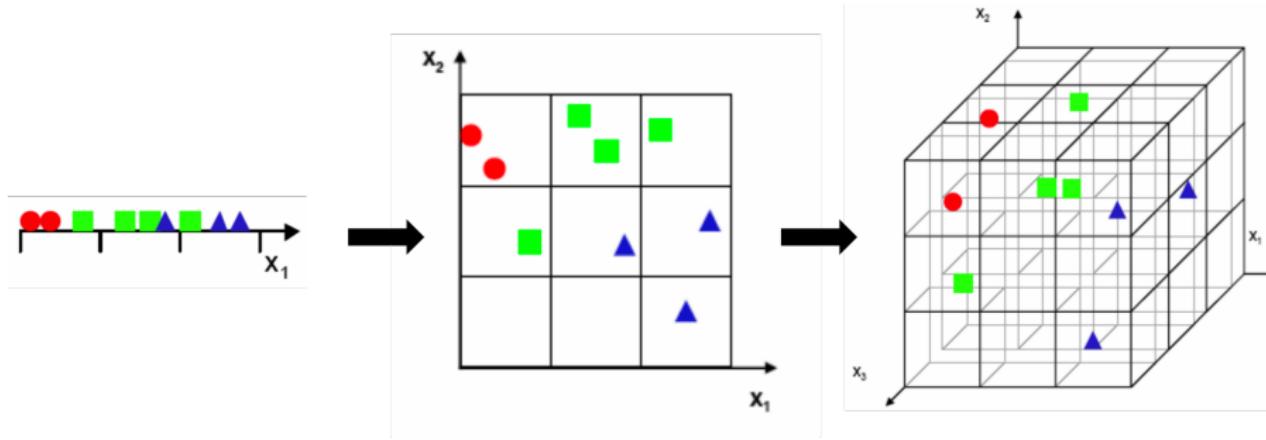
- Bound on the test error.
 - Based on the training error and the VC-dimension.

$$L(h)_{\text{test}} \leq L(h)_{\text{train}} + O\left(\sqrt{\frac{\text{VC}(h)}{n}}\right)$$

- Let us say we have a dataset containing m points/examples.
 - These m points can be labeled in 2^m ways as positive/negative.
 - Thus, 2^m different configurations can be defined by m points.
- If for any of these problems, we can find a model h that separates positives from negatives, then h **shatters** m points.
 - That is, any learning problem definable by m examples can be learned with no error by model h .
- The maximum number of points that can be shattered by h is called the VC-dimension of h .

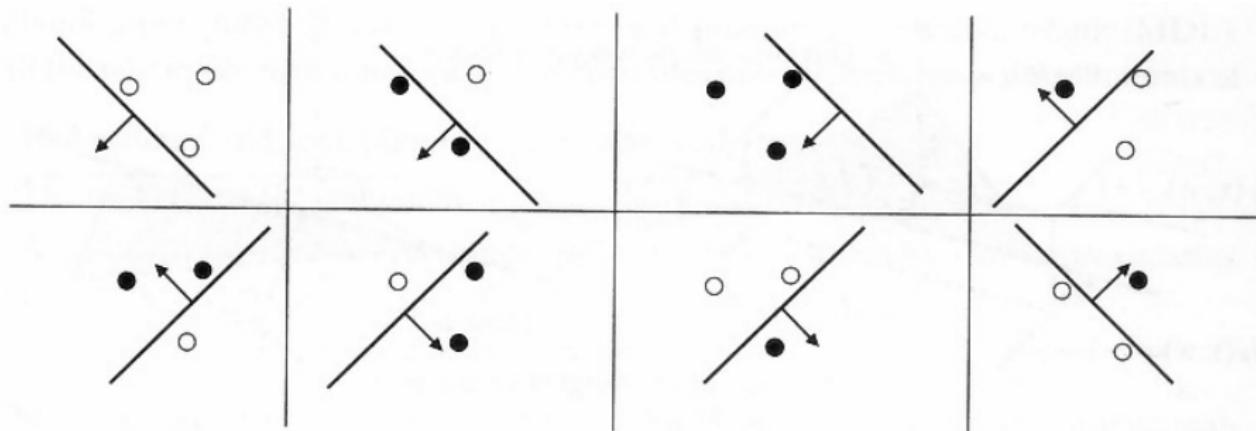
Support Vector Machines

Dimensionality and the VC-dimension.



Support Vector Machines

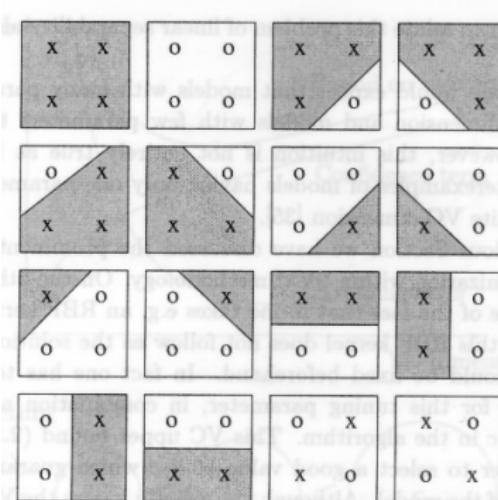
The VC-dimension.



- A Perceptron shatters any 3 points in a 2-D space.

Support Vector Machines

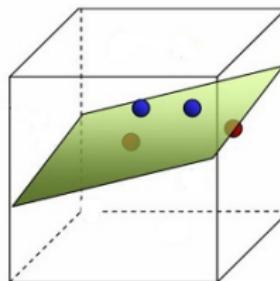
The VC-dimension.



- A Perceptron does not shatter 4 points in a 2-D space.
 - A Perceptron has VC-dimension 3 in a 2-D space.

Support Vector Machines

The VC-dimension.



- A Perceptron shatters 4 points in a 3-D space.
 - The VC-dimension increases with the dimensionality.
 - In general, for Perceptrons $VC(h) = d + 1$, where d is the dimension of the inputs.

Support Vector Machines

The VC-dimension.

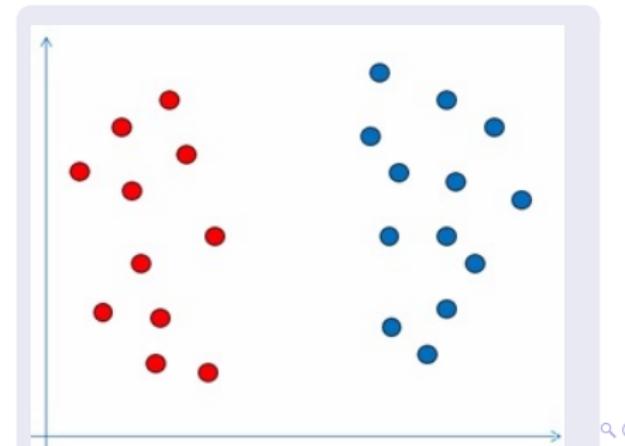
- To reduce the test error:
 - Keep training error low (i.e., 0)
 - Minimize $VC(h)$

Support Vector Machines

The VC-dimension.

- To reduce the test error:
 - Keep training error low (i.e., 0)
 - Minimize $VC(h)$
- Why maximize the margin is a good idea?

- Consider the training examples.

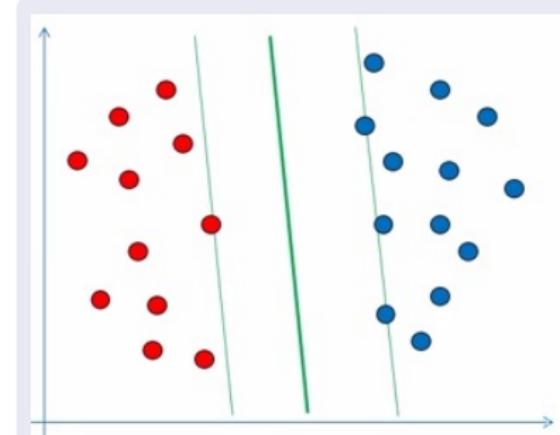


Support Vector Machines

The VC-dimension.

- To reduce the test error:
 - Keep training error low (i.e., 0)
 - Minimize $VC(h)$
- Why maximize the margin is a good idea?

- Suppose the decision boundary.

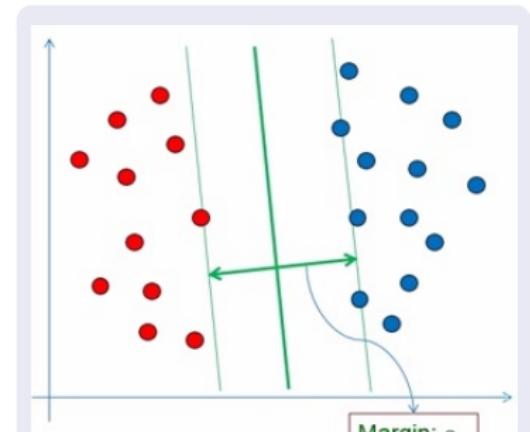


Support Vector Machines

The VC-dimension.

- To reduce the test error:
 - Keep training error low (i.e., 0)
 - Minimize $VC(h)$
- Why maximize the margin is a good idea?

- Margin depends on the scale of the points.
 - If we scale the points, the margin is also scaled.
 - Scale the points up, then the margin increases.

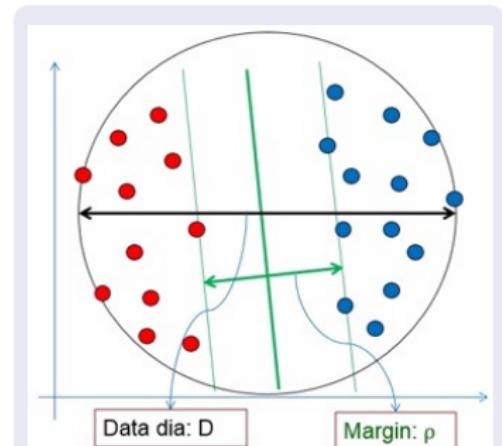


Support Vector Machines

The VC-dimension.

- To reduce the test error:
 - Keep training error low (i.e., 0)
 - Minimize $VC(h)$
- Why maximize the margin is a good idea?

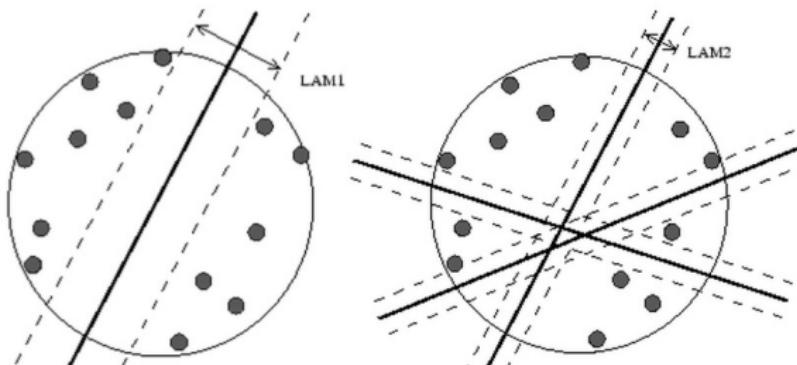
- The smallest sphere that encompasses all the examples.
 - Relative margin: $\frac{\rho}{D}$
 - $VC(h) \leq \min(d, \lceil \frac{D^2}{\rho^2} \rceil)$
 - Therefore, if we reduce $\frac{D^2}{\rho^2}$, or simply maximize ρ , $VC(h)$ becomes independent on the dimensionality of the data.



Support Vector Machines

The VC-dimension.

- Why maximize the margin is a good idea?
 - If we choose hyperplanes with a large margin, there is only a small number of possibilities to separate the data.
 - VC-dimension is smaller (remember sequence 1, 2, 4, 7, ...).
 - On the contrary, if we allow smaller margins there are many more possible separating hyperplanes.
 - VC-dimension is larger.

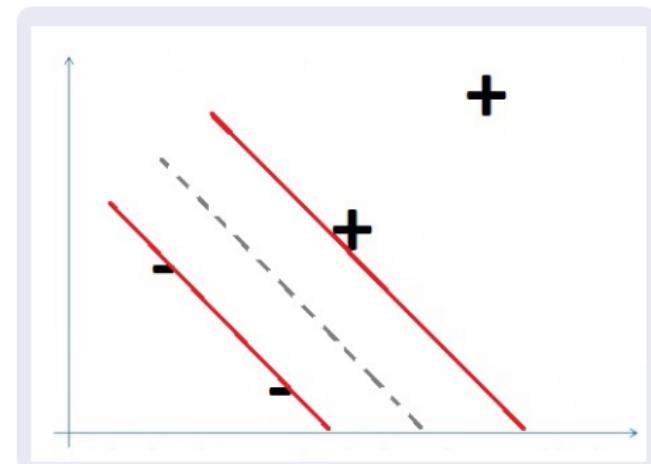


Support Vector Machines

Intuition: to separate examples with a straight line.

- But, which line?

- The one with the largest margin.
 - How to make a decision rule from the decision boundary?



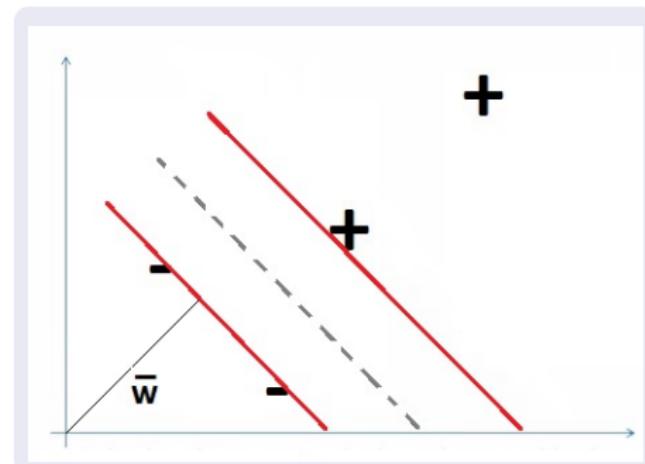
Support Vector Machines

Intuition: to separate examples with a straight line.

- But, which line?

- Decision rule.

- Suppose a vector \vec{w} which is perpendicular to the decision boundary.
- The length of \vec{w} is unknown.



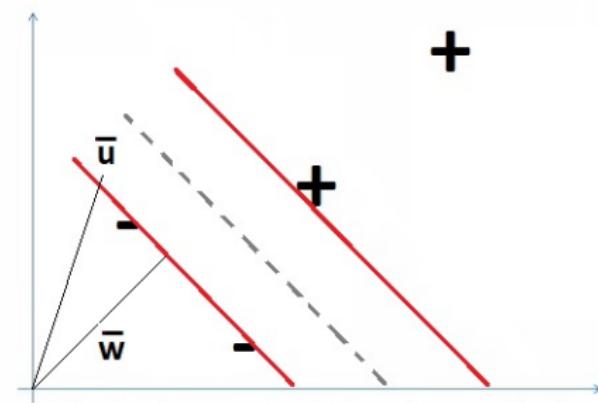
Support Vector Machines

Intuition: to separate examples with a straight line.

- But, which line?

- Decision rule.

- We also have some unknown \vec{u}
- In which side of the boundary is \vec{u} ?

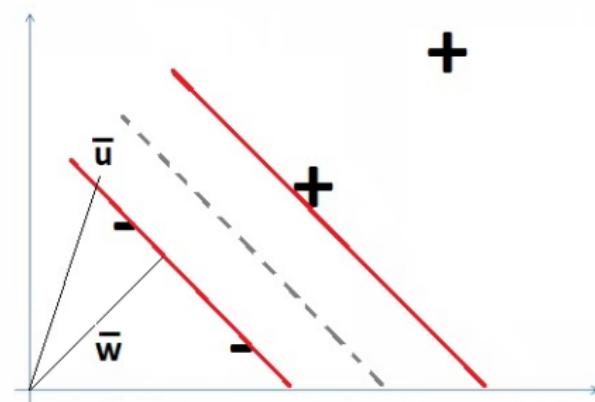


Support Vector Machines

Intuition: to separate examples with a straight line.

- But, which line?

- $\vec{w} \cdot \vec{u}$ gives the projection of \vec{u} in the direction of \vec{w} .
- If the projection is big \vec{u} is in the “+” side.
- If the projection is small \vec{u} is in the “-” side.

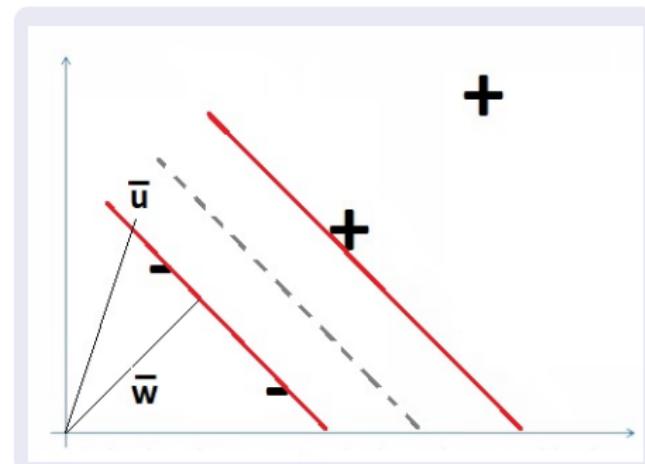


Support Vector Machines

Intuition: to separate examples with a straight line.

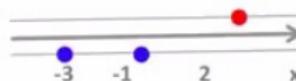
- But, which line?

- $\vec{w} \cdot \vec{u} \geq c$, and for $c = -b$
 - If $\vec{w} \cdot \vec{u} + b \geq 0 \rightarrow +$
- But we do not know the value of b
- And we do not know the length of \vec{w}
 - We only know its direction.

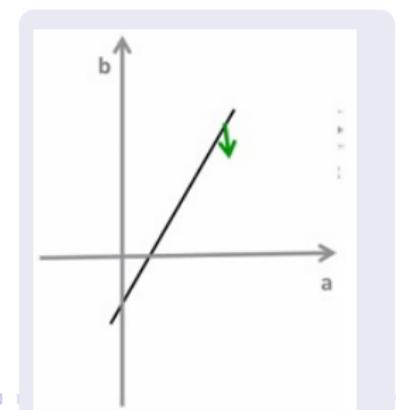


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$

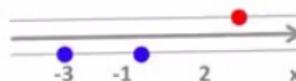


- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$

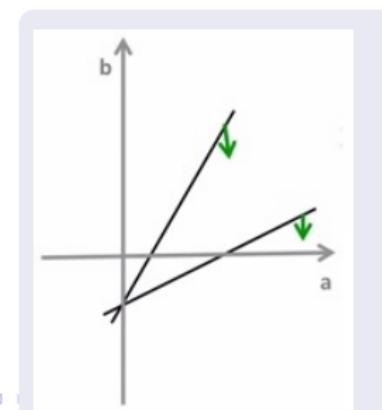


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$

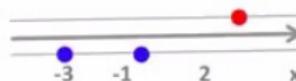


- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$

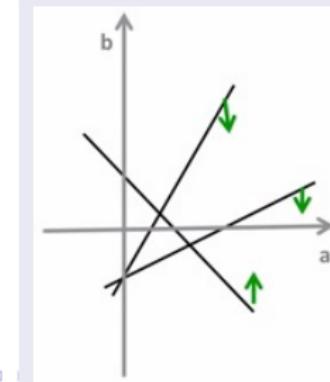


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$

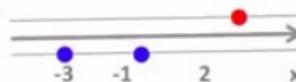


- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$

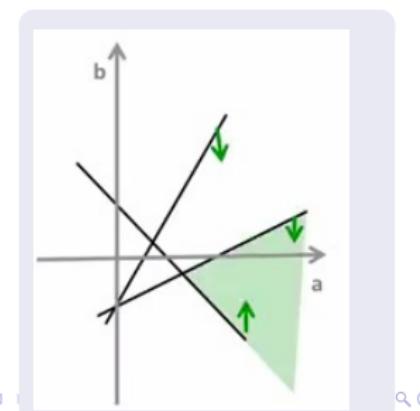


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$

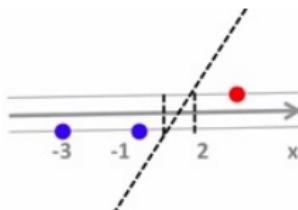


- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$

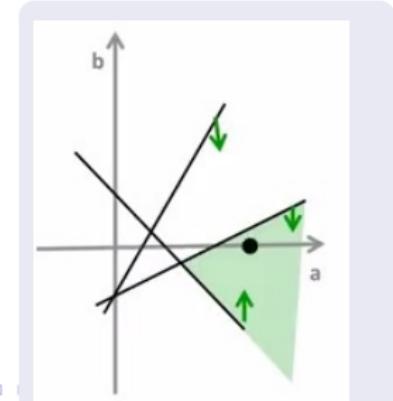


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$

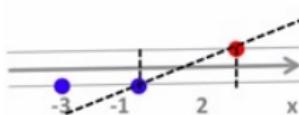


- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$
- x

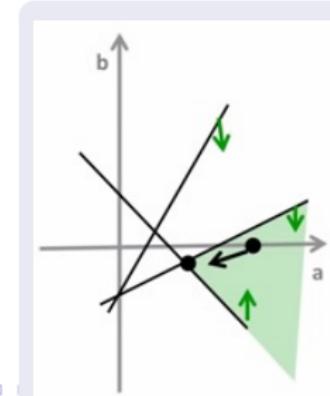


Support Vector Machines

- Suppose we have three examples: $(-3, -1)$, $(-1, -1)$, $(2, 1)$



- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $a \times (-3) + b \leq -1 \Rightarrow b \leq 3 \times a - 1$
 - $a \times (-1) + b \leq -1 \Rightarrow b \leq a - 1$
 - $a \times (2) + b \geq +1 \Rightarrow b \geq -2 \times a + 1$
- $0.66x - 0.33$



Support Vector Machines

Calculating b and \vec{w} .

- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
 - Since \vec{x}_+ is positive, the decision rule must evaluate $\geq +1$.
- $\vec{w} \cdot \vec{x}_- + b \leq -1$

Support Vector Machines

Calculating b and \vec{w} .

- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
 - Since \vec{x}_+ is positive, the decision rule must evaluate $\geq +1$.
- $\vec{w} \cdot \vec{x}_- + b \leq -1$

Include another variable for mathematical convenience.

- y_i such that

$$y_i = \begin{cases} +1, & \text{if } x_i \text{ is a positive example.} \\ -1, & \text{otherwise.} \end{cases} \quad (2)$$

- For each example we have a value for y_i

Support Vector Machines

Calculating b and \vec{w} .

- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
 - $y_i \times (\vec{w} \cdot \vec{x}_i + b) \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $y_i \times (\vec{w} \cdot \vec{x}_i + b) \geq +1$

Support Vector Machines

Calculating b and \vec{w} .

- $\vec{w} \cdot \vec{x}_+ + b \geq +1$
 - $y_i \times (\vec{w} \cdot \vec{x}_i + b) \geq +1$
- $\vec{w} \cdot \vec{x}_- + b \leq -1$
 - $y_i \times (\vec{w} \cdot \vec{x}_i + b) \geq +1$
- $y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0$
 - $y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0$, for all \vec{x}_i lying in the support.

Support Vector Machines

Calculating b and \vec{w} .

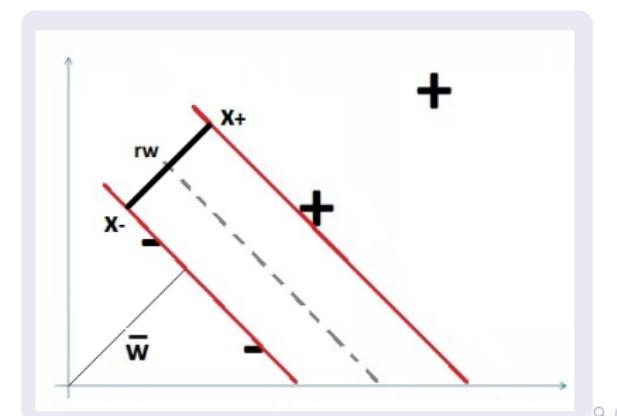
- To maximize the margin, we must know how to calculate it
- Choose x_- such that $\vec{w} \cdot \vec{x}_- = -1$
- Let x_+ be the closest point such that $\vec{w} \cdot \vec{x}_+ = +1$
 - $x_+ = x_- + r \times \vec{w}$

Support Vector Machines

Calculating b and \vec{w} .

- To maximize the margin, we must know how to calculate it
- Choose x_- such that $\vec{w} \cdot \vec{x}_- = -1$
- Let x_+ be the closest point such that $\vec{w} \cdot \vec{x}_+ = +1$
 - $x_+ = x_- + r \times \vec{w}$

- $\vec{w} \cdot \vec{x}_- + b = -1$, and
 $\vec{w} \cdot \vec{x}_+ + b = +1$
- $\Rightarrow r\|\vec{w}\|^2 + \vec{w} \cdot \vec{x}_- + b = +1$
- $\Rightarrow r\|\vec{w}\|^2 - 1 = +1$
- $\Rightarrow r = \frac{2}{\|\vec{w}\|^2}$
- $\rho = \|r \times \vec{w}\| = \frac{2\|\vec{w}\|}{\|\vec{w}\|^2} = \frac{2}{\|\vec{w}\|}$



Support Vector Machines

Calculating b and \vec{w} .

- $\max \frac{2}{||\vec{w}||}$

Support Vector Machines

Calculating b and \vec{w} .

- $\max \frac{2}{\|\vec{w}\|}$
- Which is the same as $\max \frac{1}{\|\vec{w}\|}$
 - Which is the same as $\min \|\vec{w}\|$
 - Which is the same as $\min \|\vec{w}\|^2$
 - Which is the same as $\min \frac{1}{2} \|\vec{w}\|^2$ (mathematical convenience)
 - such that "all points on the correct side of the boundary"

Support Vector Machines

Calculating b and \vec{w} .

- $\max \frac{2}{\|\vec{w}\|}$
- Which is the same as $\max \frac{1}{\|\vec{w}\|}$
 - Which is the same as $\min \|\vec{w}\|$
 - Which is the same as $\min \|\vec{w}\|^2$
 - Which is the same as $\min \frac{1}{2} \|\vec{w}\|^2$ (mathematical convenience)
 - such that "all points on the correct side of the boundary"
- We want to maximize a function:
 - given as a sum of quadratic terms
 - with linear constraints
- This is known as **quadratic program**.

Support Vector Machines

Calculating b and \vec{w} .

- QP is solved using Lagrangian multipliers.

- $$\vec{w}^* = \operatorname{argmin}_{\vec{w}} \max_{\alpha} f(\vec{w}) + \sum_i \alpha_i g_i(\vec{w})$$

- $$L = \frac{1}{2} \|\vec{w}\|^2 - \sum_i \alpha_i \times [y_i \times (\vec{w} \cdot \vec{x}_i + b) - 1]$$

- That is:

- $$\max \|\vec{w}\|$$
 with all points \vec{x}_i in the correct side of the boundary.

Support Vector Machines

Calculating b and \vec{w} .

- QP is solved using Lagrangian multipliers.

- $$\vec{w}^* = \operatorname{argmin}_{\vec{w}} \max_{\alpha} f(\vec{w}) + \sum_i \alpha_i g_i(\vec{w})$$

- $$L = \frac{1}{2} \|\vec{w}\|^2 - \sum_i \alpha_i \times [y_i \times (\vec{w} \cdot \vec{x}_i + b) - 1]$$

- That is:

- max $\|\vec{w}\|$ with all points \vec{x}_i in the correct side of the boundary.

- Find the derivative of L and set it to 0.

- $$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_i \alpha_i \times y_i \times \vec{x}_i = 0$$

Support Vector Machines

Calculating b and \vec{w} .

- QP is solved using Lagrangian multipliers.

- $$\vec{w}^* = \operatorname{argmin}_{\vec{w}} \max_{\alpha} f(\vec{w}) + \sum_i \alpha_i g_i(\vec{w})$$

- $$L = \frac{1}{2} \|\vec{w}\|^2 - \sum_i \alpha_i \times [y_i \times (\vec{w} \cdot \vec{x}_i + b) - 1]$$

- That is:

- \bullet max $\|\vec{w}\|$ with all points \vec{x}_i in the correct side of the boundary.

- Find the derivative of L and set it to 0.

- $$\frac{\partial L}{\partial \vec{w}} = \vec{w} - \sum_i \alpha_i \times y_i \times \vec{x}_i = 0 \Rightarrow \vec{w} = \sum_i \alpha_i \times y_i \times \vec{x}_i$$

- This means that \vec{w} is given as a linear combination of the input vectors x'_i s

Support Vector Machines

Calculating b and \vec{w} .

Substitute \vec{w} in L

- $L = \frac{1}{2} \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i \vec{x}_i \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$

Support Vector Machines

Calculating b and \vec{w} .

Substitute \vec{w} in L

- $L = \frac{1}{2} \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i \vec{x}_i \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$
- $L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j$
 - The optimal \vec{w} is a linear combination of the training examples
 - If x_i is positive, $\alpha_i \geq 0$
 - If x_i is negative, $\alpha_i < 0$
 - Only the support vectors have non-zero α coefficients

Support Vector Machines

Calculating b and \vec{w} .

Substitute \vec{w} in L

- $L = \frac{1}{2} \left(\sum_i \alpha_i y_i \vec{x}_i \right) \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i \vec{x}_i \cdot \left(\sum_j \alpha_j y_j \vec{x}_j \right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$
- $L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j$
 - The optimal \vec{w} is a linear combination of the training examples
 - If x_i is positive, $\alpha_i \geq 0$
 - If x_i is negative, $\alpha_i < 0$
 - Only the support vectors have non-zero α coefficients

Finally!

- Simply maximize L to find optimal α values.

Support Vector Machines

Decision rule:

- $\vec{w} \cdot \vec{x} + b \geq 0 \rightarrow +1$
- Substituting $\vec{w} = \sum_i \alpha_i y_i \vec{x}_i \Rightarrow \sum_i \alpha_i y_i \vec{x}_i \cdot \vec{u} + b \geq 0 \rightarrow +1$

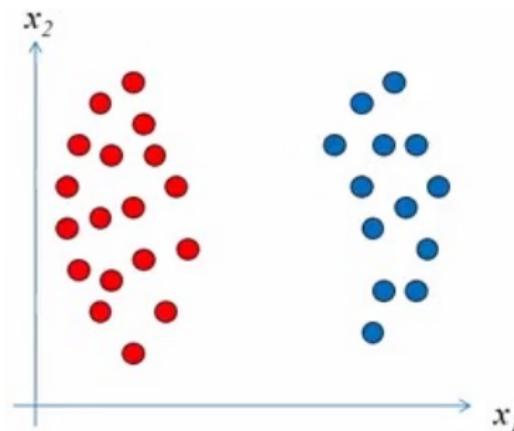
And since any support vector has $y_i = \vec{w} \cdot \vec{x} + b$, then:

- $b = \frac{1}{\#sv} \times \sum_{i \in SV} (y_i - \vec{w} \cdot \vec{x}_i)$

Support Vector Machines

Non-separable data.

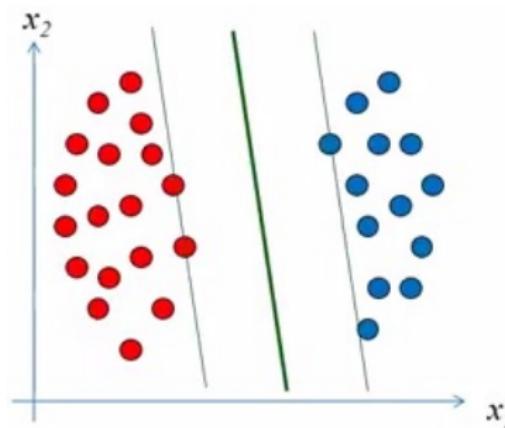
- SVMs work well on linearly separable data.



Support Vector Machines

Non-separable data.

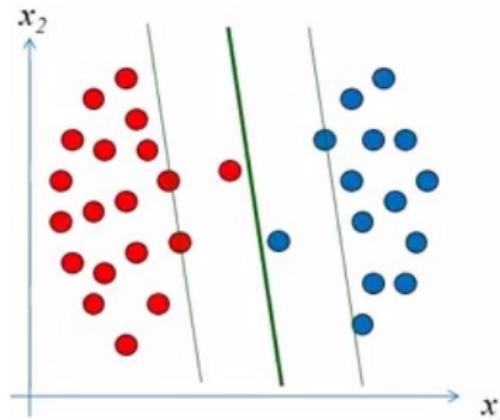
- SVMs work well on linearly separable data.



Support Vector Machines

Non-separable data.

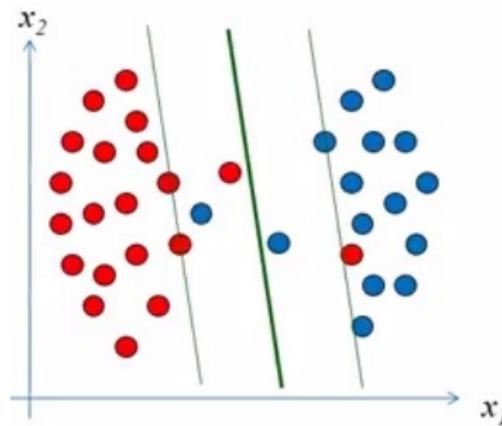
- Noisy data and bad features. Margin is compromised.



Support Vector Machines

Non-separable data.

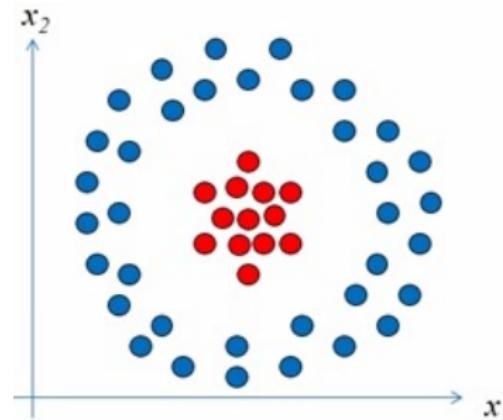
- Noisy data and bad features. The data is no longer separable.



Support Vector Machines

Non-separable data.

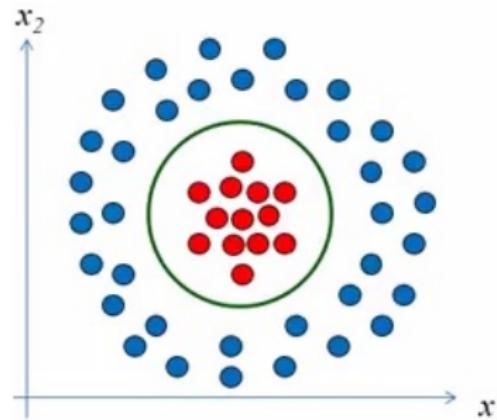
- There is no linear boundary.



Support Vector Machines

Non-separable data.

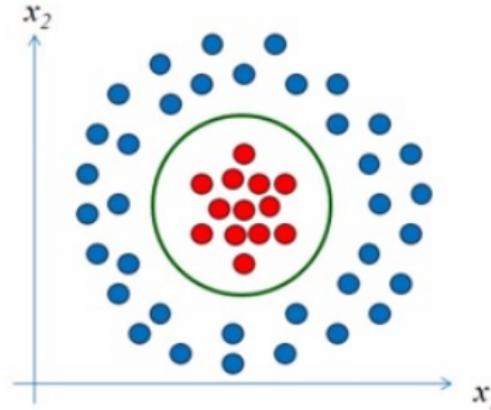
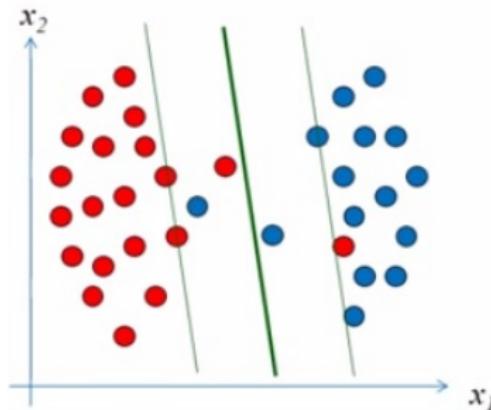
- There is a simple non-linear boundary.



Support Vector Machines

Two different reasons that make data not linearly separable:

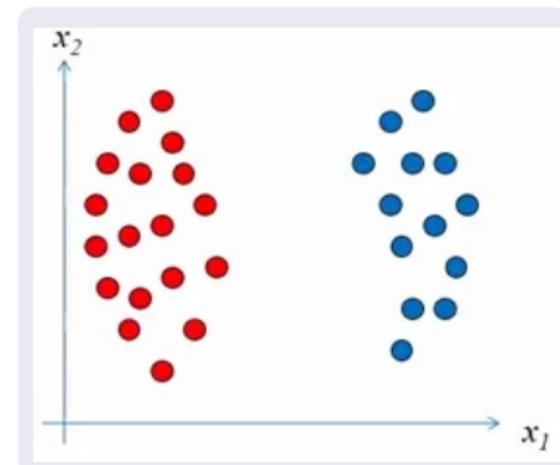
- In the first case, it is possible to find a complex non-linear separation. But it would probably overfit.
- In the second case, there is a simple non-linear boundary.



Support Vector Machines

Soft margin.

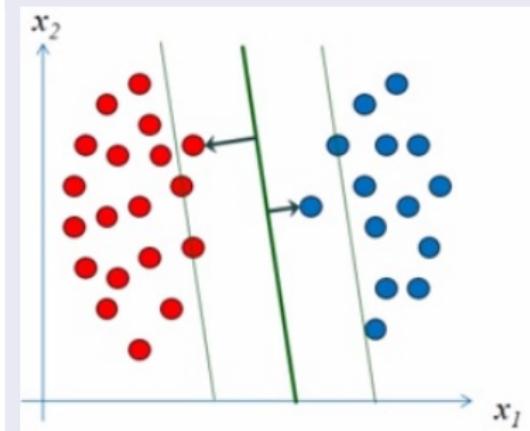
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
- Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$



Support Vector Machines

Soft margin.

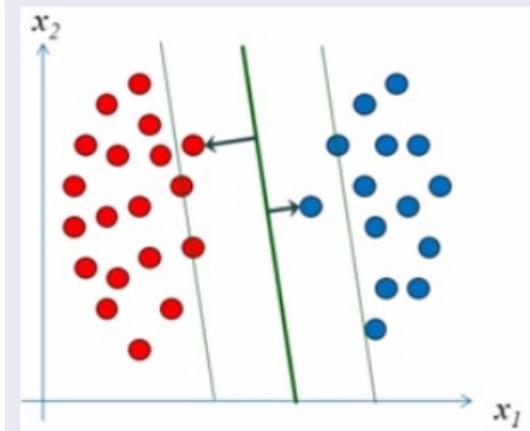
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
- Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$



Support Vector Machines

Soft margin.

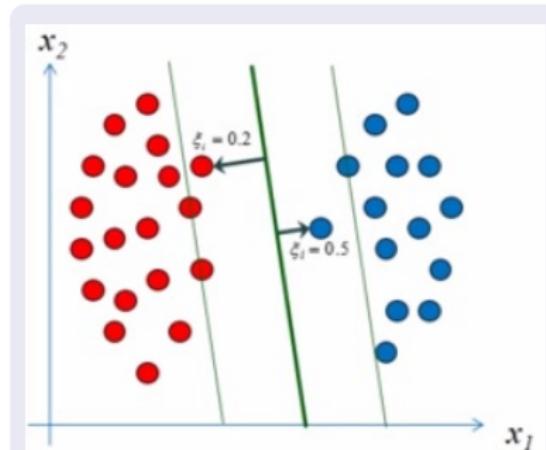
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$$
- Introduce slack variables: $\epsilon_i \geq 0$
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$$



Support Vector Machines

Soft margin.

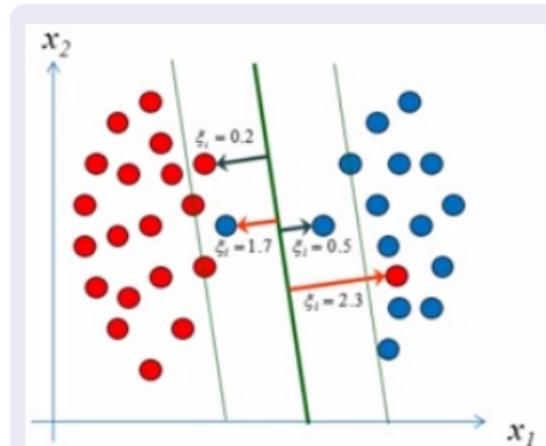
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$
 - Introduce slack variables: $\epsilon_i \geq 0$
 - Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$
 - $\epsilon_i \leq 1$, because points are still on the right side.



Support Vector Machines

Soft margin.

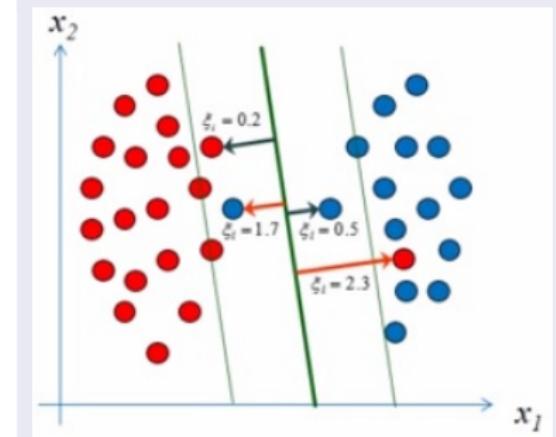
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$$
 - Introduce slack variables: $\epsilon_i \geq 0$
 - Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
$$y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$$
 - $\epsilon_i > 1$, because now points are on the wrong side.



Support Vector Machines

Soft margin.

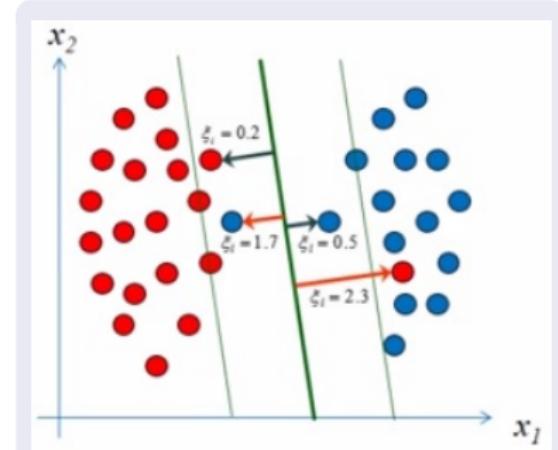
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 - $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$
- Introduce slack variables: $\epsilon_i \geq 0$
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 - $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$
- To minimize training error:
 - Also minimize $\sum_i \epsilon_i$



Support Vector Machines

Soft margin.

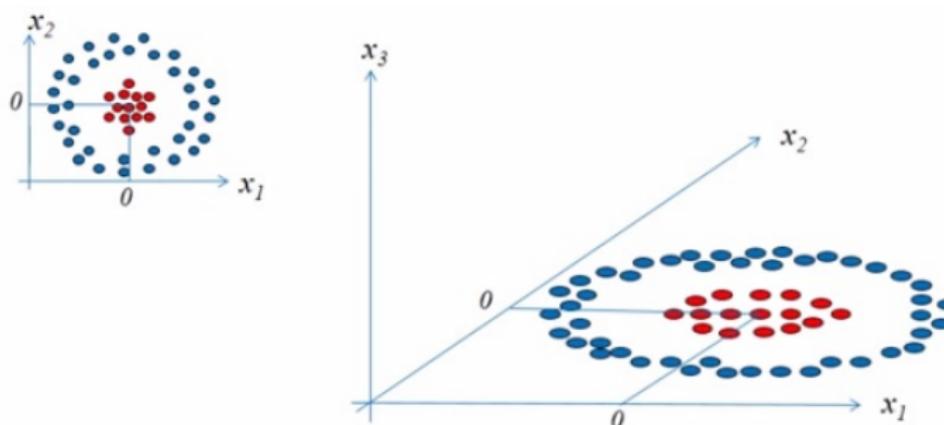
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \forall i$
- Introduce slack variables: $\epsilon_i \geq 0$
- Minimize: $\frac{1}{2} \|\vec{w}\|^2$
 - Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$
- Minimize: $\frac{1}{2} \|\vec{w}\|^2 + C \times \sum \epsilon_i$
 - Subject to:
 $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \epsilon_i \forall i$
- C trades margin for error.



Support Vector Machines

Kernels.

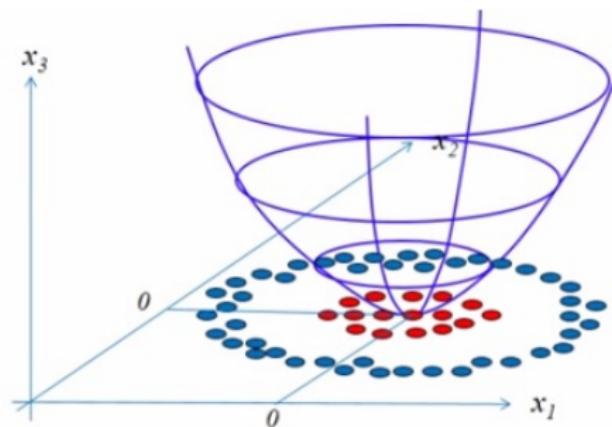
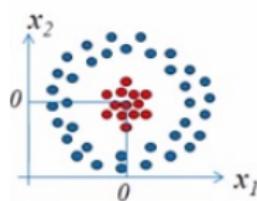
- Map the points to a higher dimensional feature space.
 - Data is linearly separable in the feature space.
 - $x_3 = f(x_1, x_2)$



Support Vector Machines

Kernels.

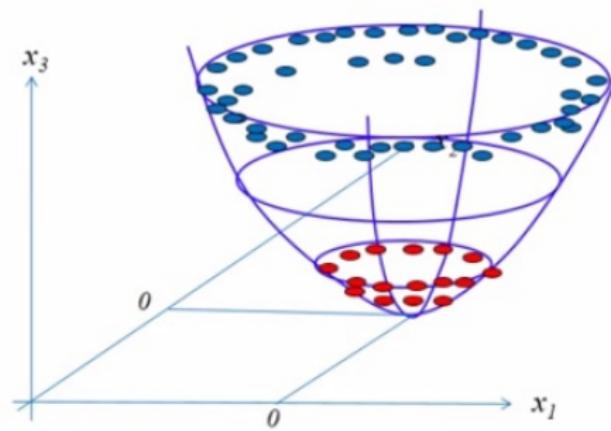
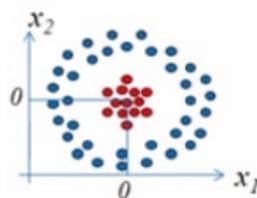
- Map the points to a higher dimensional feature space.
 - Data is linearly separable in the feature space.
 - $x_3 = x_1^2 + x_2^2$



Support Vector Machines

Kernels.

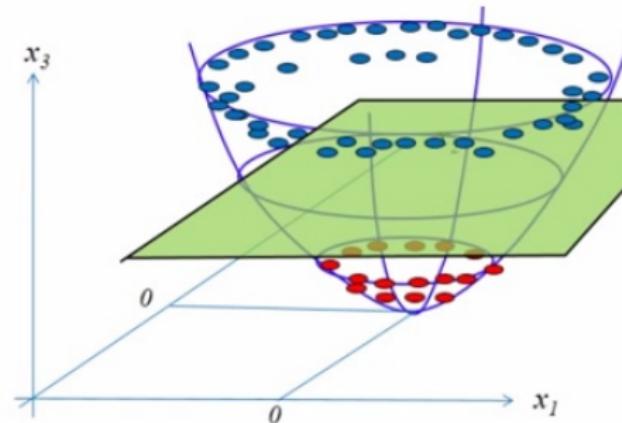
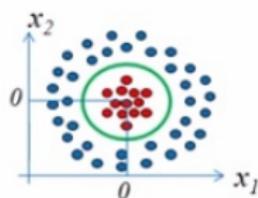
- Map the points to a higher dimensional feature space.
 - Data is linearly separable in the feature space.
 - $x_3 = x_1^2 + x_2^2$



Support Vector Machines

Kernels.

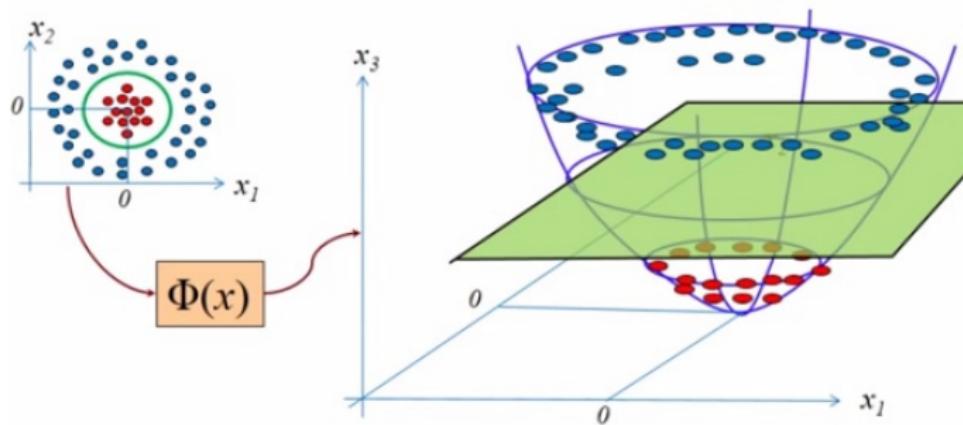
- Map the points to a higher dimensional feature space.
 - Data is linearly separable in the feature space.
 - $x_3 = x_1^2 + x_2^2$



Support Vector Machines

Kernels.

- Map the points to a higher dimensional feature space.
 - Φ is a non-linear mapping into a high-dimensional space.



Support Vector Machines

Kernels.

- Map the points to a higher dimensional feature space.

- $$L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$$

Support Vector Machines

Kernels.

- Map the points to a higher dimensional feature space.
 - $L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j)$
- If we can find a function $K(\vec{x}, \vec{u})$, which is equivalent to $\Phi(\vec{x}) \cdot \Phi(\vec{u})$, we can avoid explicit mapping to high dimensions
 - $L = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j)$

Support Vector Machines

Kernels.

- Let $\Phi(\vec{x}) = \Phi \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix}$

Support Vector Machines

Kernels.

- Let $\Phi(\vec{x}) = \Phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_2x_1 \\ x_2x_2 \end{bmatrix}$

- Let $K(\vec{x}, \vec{u}) = \Phi(\vec{x}) \cdot \Phi(\vec{u}) = \begin{bmatrix} x_1^2 \\ x_1x_2 \\ x_2x_1 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} u_1^2 \\ u_1u_2 \\ u_2u_1 \\ u_2^2 \end{bmatrix} = x_1^2u_1^2 + 2x_1x_2u_1u_2 + x_2^2u_2^2 = (x_1u_1 + x_2u_2)^2 = (\vec{x} \cdot \vec{u})^2$

Support Vector Machines

Popular Kernels.

- Polynomial kernel: $(\vec{x} \cdot \vec{u} + 1)^n$
- Radial Basis kernel: $e^{\frac{-1}{2\sigma^2} ||\vec{x} - \vec{u}||^2}$
- Hyperbolic Tangent kernel: $\tanh(\beta_0 \times \vec{x} \cdot \vec{u} + \beta_1)$

Relationship between Neural Nets and SVMs

Put simply:

- Linear SVMs are similar to a Perceptron, but with an optimal cost function.
- If a Kernel function is used, then SVMs are comparable to 2-layer neural networks.
 - The first layer is able to project data into some other space and next layer classifies the projected data.
- If one more layer is used then it might correspond to an ensemble of multiple Kernel SVMs.

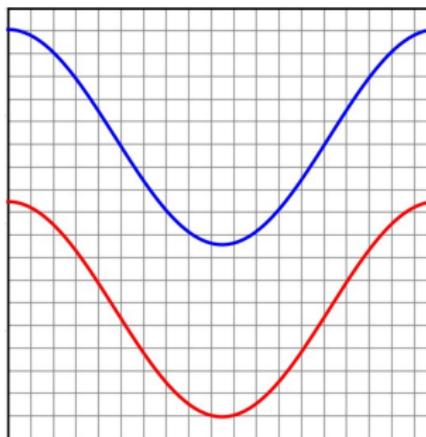
Relationship between Neural Nets and SVMs

Put simply:

- Linear SVMs are similar to a Perceptron, but with an optimal cost function.
- If a Kernel function is used, then SVMs are comparable to 2-layer neural networks.
 - The first layer is able to project data into some other space and next layer classifies the projected data.
- If one more layer is used then it might correspond to an ensemble of multiple Kernel SVMs.

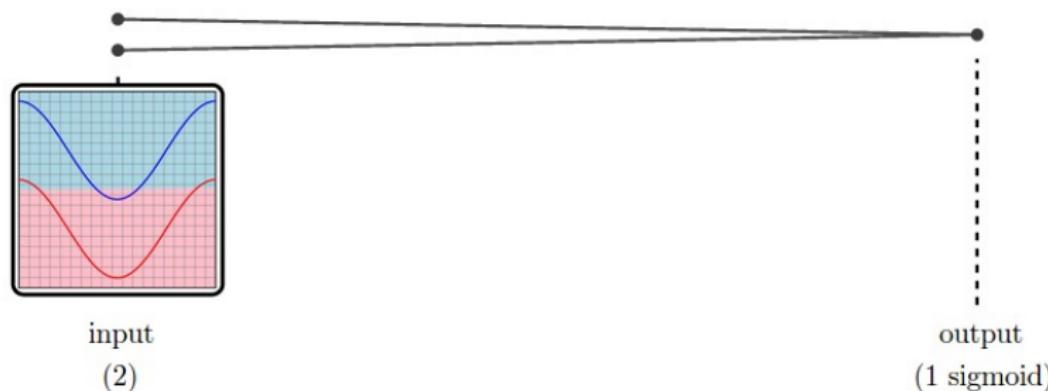
Relationship between Neural Nets and SVMs

- Consider the following dataset:
 - Two curves on the plane. Given a point on one of the curves, the network should predict which curve it came from.



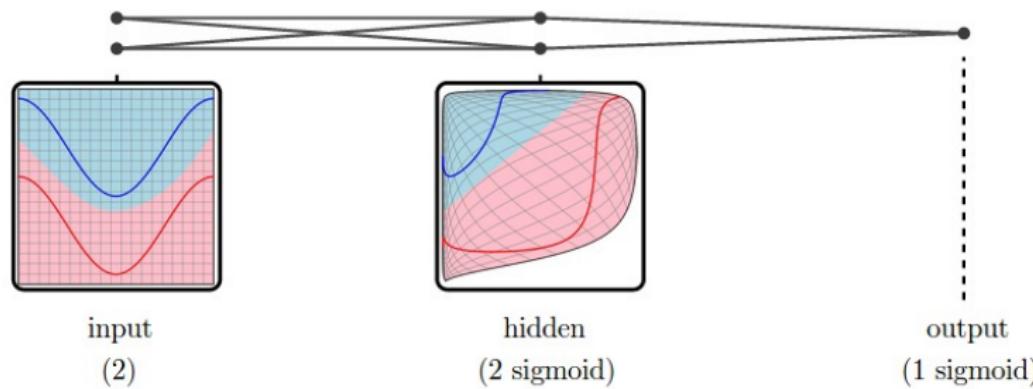
Relationship between Neural Nets and SVMs

- A network with just an input layer and an output layer divides the two classes with a straight line.
 - In this case, it is not possible to classify it perfectly by dividing it with a straight line.



Relationship between Neural Nets and SVMs

- A network with an additional hidden layer.
 - These layers reshape the data to make it easier to classify.



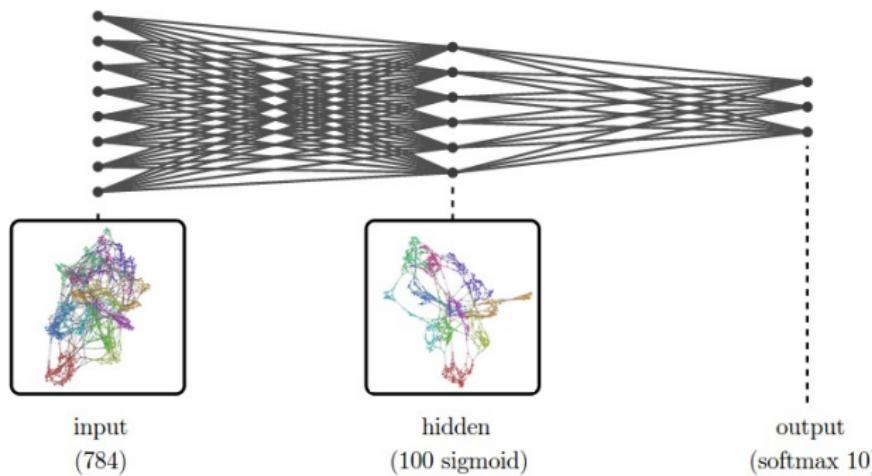
Relationship between Neural Nets and SVMs

Handwrite digit recognition:



Relationship between Neural Nets and SVMs

- At the input layer, the classes are tangled (**NN graphs**).
 - In the next layer, because the model has been trained to distinguish the digit classes, the hidden layer has learned to transform the data into a new representation in which the digit classes are much more separated.

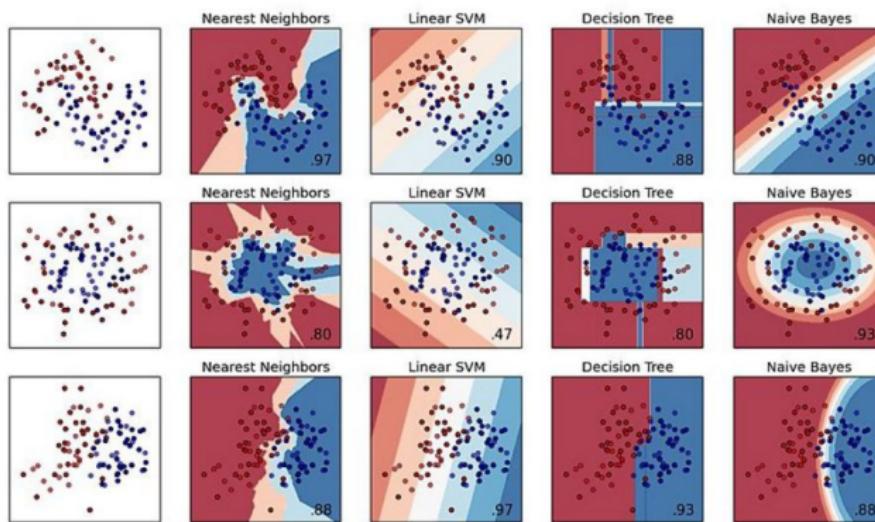


Relationship between Neural Nets and SVMs

Despite the similarities:

- SVMs have been developed in the reverse order to the development of neural networks.
 - The development of neural networks followed a heuristic path, with applications and extensive experimentation preceding theory. In contrast, the development of SVMs involved sound theory first, then implementation and experiments.
- A significant advantage of SVMs is that whilst backpropagation can suffer from multiple local minima, the solution to an SVM is global and unique.
- Unlike neural networks, the computational complexity of SVMs does not depend on the dimensionality of the data.
- The reason that SVMs often outperform neural networks in practice is that they are less prone to overfitting.

Model Selection



Model Selection

There are many hyper-parameters to set.

- Regularization:

- Decision trees: $\operatorname{argmin} E_{in}(T) + \lambda \omega(T)$
- Neural Networks: $\operatorname{argmin} E_{in}(W) + \lambda \|W\|$
- SVMs: $\operatorname{argmin} \frac{1}{2} \|W\|^2 + C \sum \epsilon_i$

Model Selection

There are many hyper-parameters to set.

- Regularization:

- Decision trees: $\operatorname{argmin} E_{in}(T) + \lambda \omega(T)$
- Neural Networks: $\operatorname{argmin} E_{in}(W) + \lambda \|W\|$
- SVMs: $\operatorname{argmin} \frac{1}{2} \|W\|^2 + C \sum \epsilon_i$

- Configuration:

- Neural Networks:

- How many hidden layers?
 - How many hidden units?
 - Momentum?

- SVMs:

- Which kernel to use?

Model Selection

Model selection is to explicitly delimitate the hypothesis space the algorithm will search on.

- Simpler or more complex models?
 - If error minimization is prioritized, then more complex models are preferred.

Model Selection

Model selection is to explicitly delimitate the hypothesis space the algorithm will search on.

- Simpler or more complex models?
 - If error minimization is prioritized, then more complex models are preferred.
 - If the input space is entangled, then more complex representations are needed.
 - More hidden units.
 - Higher polynomial kernels.

Model Selection

Model selection is to find hyper-parameters by validation.

Model Selection

Model selection is to find hyper-parameters by validation.

- Error estimates:
 - E_{in} is a biased estimate.
 - The error estimate is given on the same data in which the model is built.
 - How can we get a less biased estimate?

Model Selection

Model selection is to find hyper-parameters by validation.

- Error estimates:
 - E_{in} is a biased estimate.
 - The error estimate is given on the same data in which the model is built.
 - How can we get a less biased estimate?
 - Separate the training set into two parts T and V .
 - A is used to build the model, and V is used to evaluate the model.
 - The resulting estimate is called E_{val} .

Model Selection

Dilemma.

- We want accurate models and accurate estimates.
 - We want $E_{val} \approx E_{out}$, and we want E_{out} to be low.

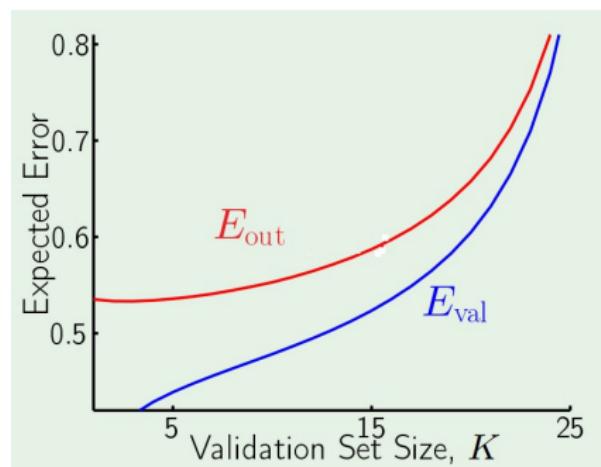
Model Selection

Dilemma.

- We want accurate models and accurate estimates.
 - We want $E_{val} \approx E_{out}$, and we want E_{out} to be low.
 - We want A to be large:
 - More data for training lead to better models.
 - We want V to be large:
 - More data to validate lead to better estimate.
 - The sample is finite.
 - The larger A is, the smaller V will be.

Model Selection

Dilemma.



Model Selection

Leave-one-out.

- $A = n - 1$ examples, $V = \text{one example.}$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)$

Model Selection

Leave-one-out.

- $A = n - 1$ examples, $V = \text{one example.}$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (\textcolor{red}{x_{n-1}, y_{n-1}}), (x_n, y_n)$

Model Selection

Leave-one-out.

- $A = n - 1$ examples, $V = \text{one example.}$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (\textcolor{red}{x_{n-1}, y_{n-1}}), (x_n, y_n)$
- E_{val} is low, but not a good approximation of E_{out} .

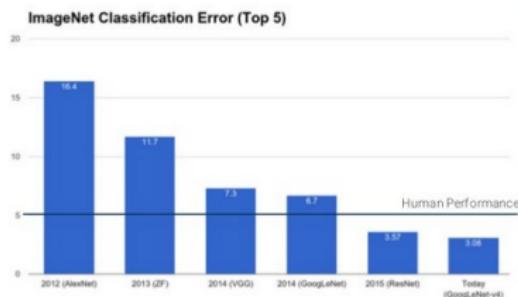
Model Selection

Leave-one-out.

- $A = n - 1$ examples, $V = \text{one example.}$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (x_{n-1}, y_{n-1}), (x_n, y_n)$
 - $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_{n-2}, y_{n-2}), (\textcolor{red}{x_{n-1}, y_{n-1}}), (x_n, y_n)$
- E_{val} is low, but not a good approximation of E_{out} .
- How to improve error estimates?
 - Repeat the process, leaving different examples out at each iteration.
 - Compute the error ϵ_i in each iteration.
 - Cross-validation error $E_{cv} = \frac{1}{k} \sum_i^k \epsilon_i$

Model Selection

Cross-validation.

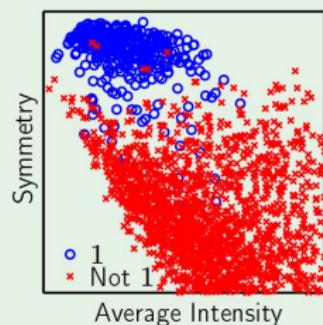


- $E_{cv} = \frac{1}{3} \times (e_1 + e_2 + e_3)$

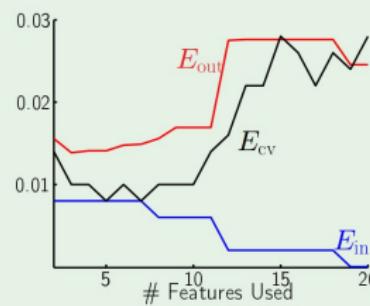
Model Selection

Cross-validation.

Digits classification task



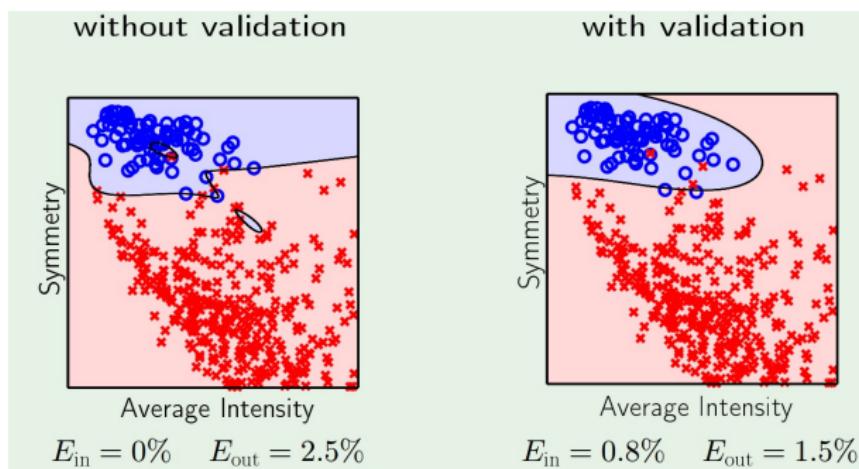
Different errors



$$(1, x_1, x_2) \rightarrow (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2, x_1^3, x_1^2 x_2, \dots, x_1^5, x_1^4 x_2, x_1^3 x_2^2, x_1^2 x_2^3, x_1 x_2^4, x_2^5)$$

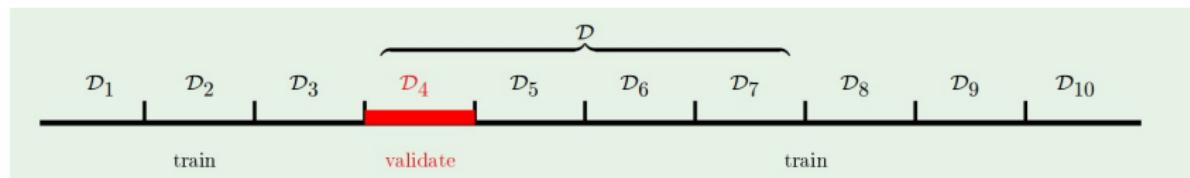
Model Selection

Cross-validation.



Model Selection

Leave more examples out.



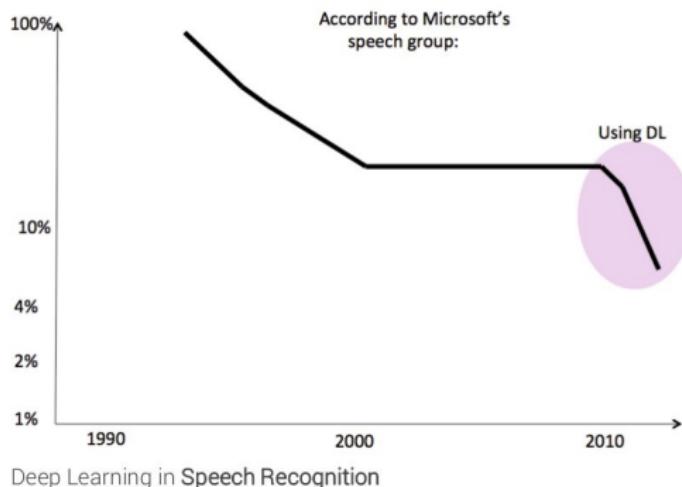
- More examples for validation.
 - k repetitions on $\frac{n}{k}$ examples each.
 - 10-fold cross-validation: $k = \frac{n}{10}$

Networks for Specific Problems

We have looked so far to the fundamentals, ideas and concepts behind neural networks. Now we start looking to applications of neural networks to specific problems.

- Computer Vision.
 - Breakthrough results
- Natural Language Processing.
 - Game Changing

Networks for Specific Problems



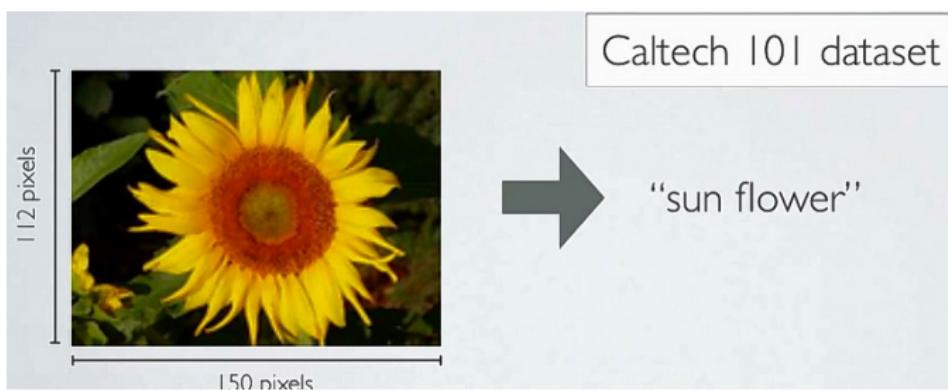
Networks for Specific Problems



Computer Vision

Computer vision.

- The design of computers that can process images and videos in order to accomplish some given task.
 - Object recognition: given some input image, identify which objects it contains.
 - Features: patterns that may help distinguishing different objects in images.



Computer Vision

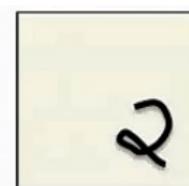
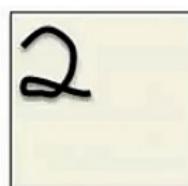
Humans are so good at recognizing objects that it is hard to appreciate how difficult is this task.

- Parts of an object can be hidden behind other objects.
- The same object may have a wide variety of shapes.
- Two dimensional image of three dimensional real scene.
- Multiple objects in the same image.

Computer Vision

Humans are so good at recognizing objects that it is hard to appreciate how difficult is this task.

- Parts of an object can be hidden behind other objects.
- The same object may have a wide variety of shapes.
- Two dimensional image of three dimensional real scene.
- Multiple objects in the same image.
- **Images are high-dimensional inputs.**
 - $112 \times 150 = 16,800$ inputs.
- **Changes in viewpoint cause changes in images.**
 - Image is represented by completely different inputs.



Computer Vision

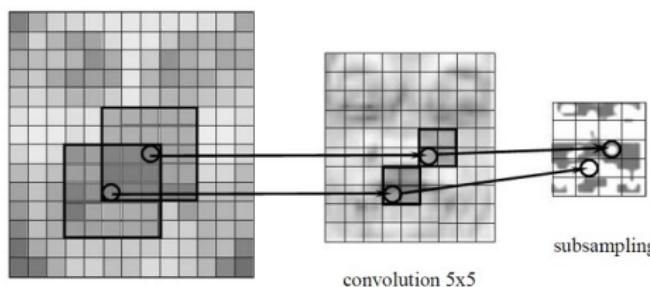
We can exploit the 2-D topology (pixels are organized spatially).

- Put a “box” around the object.
- Assemble different boxes to form new features.
- The process repeats in order to form more abstract features.
- These features are the inputs for a neural network.

Computer Vision

We can exploit the 2-D topology (pixels are organized spatially).

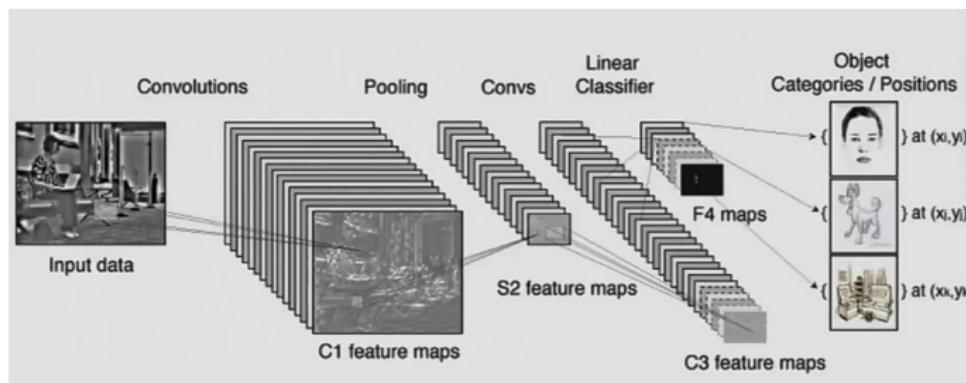
- Put a “box” around the object.
- Assemble different boxes to form new features.
- The process repeats in order to form more abstract features.
- These features are the inputs for a neural network.



- Drastically reduces the dimensionality.
- Provides local invariance (less sensitive to small perturbs).

Computer Vision

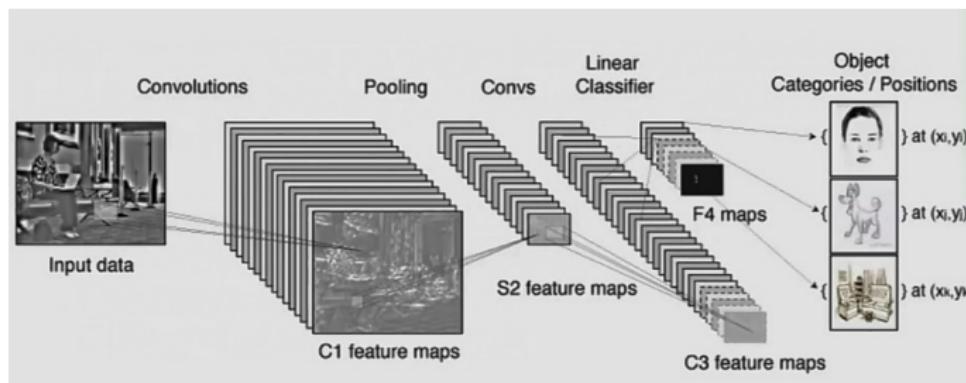
Convolutional neural networks.



- A small box (or filter) passes over all the image
 - Think in it as a search:
 - Where in the image there is something similar to the filter.

Computer Vision

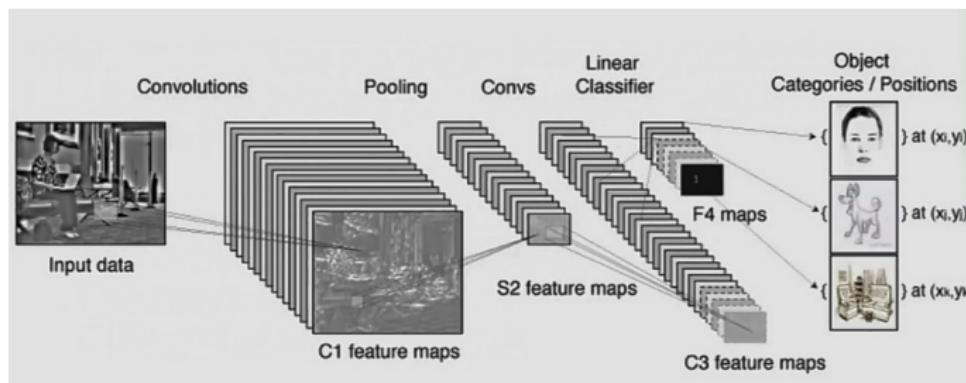
Convolutional neural networks.



- A small box (or filter) passes over all the image
 - Think in it as a search:
 - So, the filter is also an image.

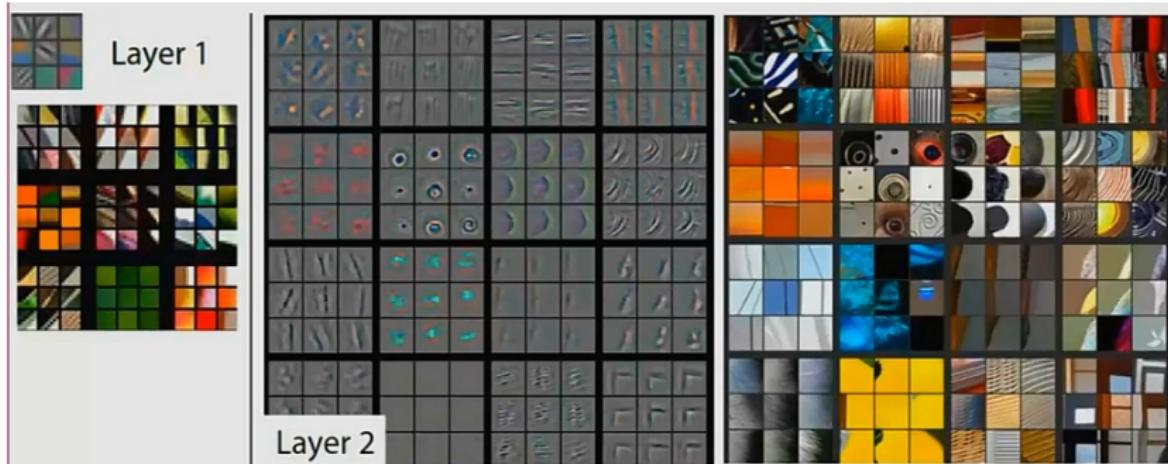
Computer Vision

Convolutional neural networks.



- A small box (or filter) passes over all the image
 - Think in it as a search:
 - Filters are also parameters we want to learn.

Computer Vision



- Layer 1:
 - Simple lines or edges.
- Layer 2:
 - Starts getting some texture and forms.

Computer Vision



- Layer 3:
 - Shapes.
- Layers 4 and 5:
 - Dogs, flowers, people etc.

Computer Vision



Yann LeCun

[Seguir](#)

Director of AI Research at Facebook & Silver Professor at the Courant Institute, New York University

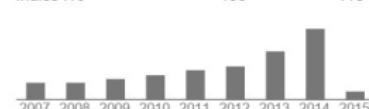
[AI, machine learning, computer vision, robotics, image compression](#)

E-mail confirmado em cs.nyu.edu - [Página inicial](#)

Titulo	1–20	Citado por	Ano
Gradient-based learning applied to document recognition			
Y LeCun, L Bottou, Y Bengio, P Haffner Proceedings of the IEEE 86 (11), 2278-2324		3140	1998
Optimal brain damage			
Y LeCun, JS Denker, SA Solla Advances in neural information processing systems 2, NIPS 1989 2, 598-605		1771	1990

Google Acadêmico

Índices de citações	Todos	Desde 2010
Citações	23845	12419
Índice h	72	51
Índice i10	159	118

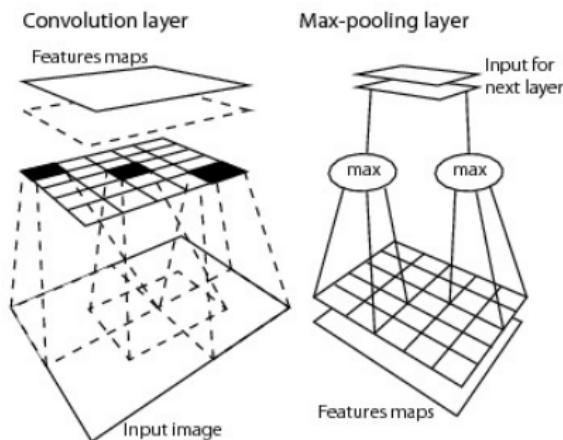


Coautores [Visualizar todos...](#)

Leon Bottou

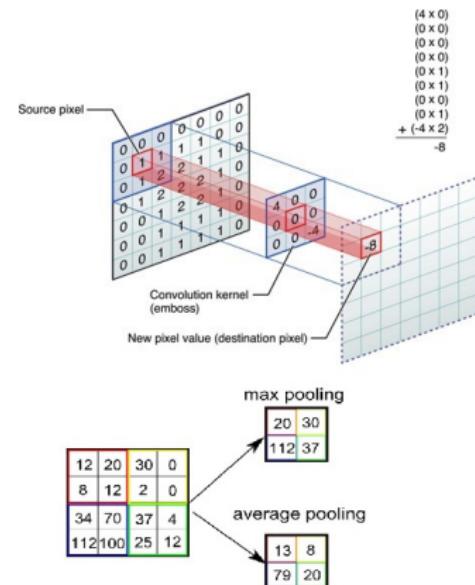
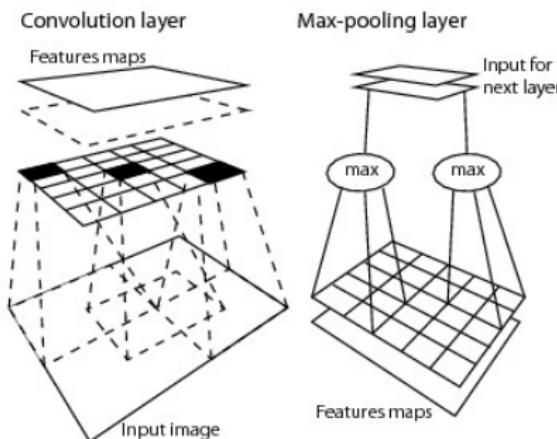
Computer Vision

Convolution and pooling.



Computer Vision

Convolution and pooling.

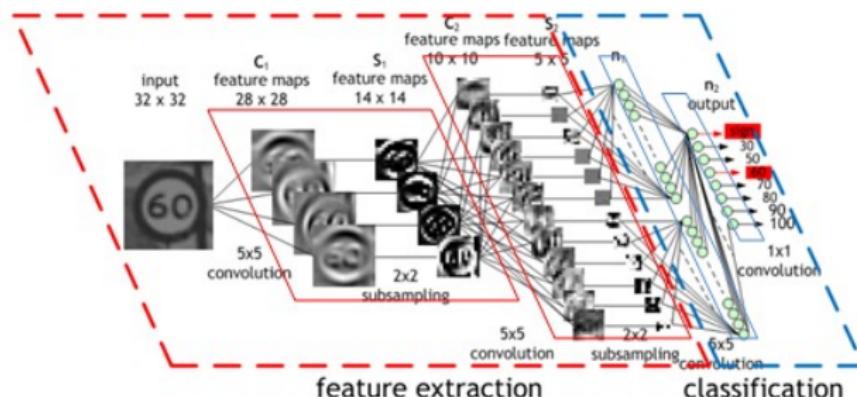


- Convolution matrix = matrix of weights = convolution kernel.

Computer Vision

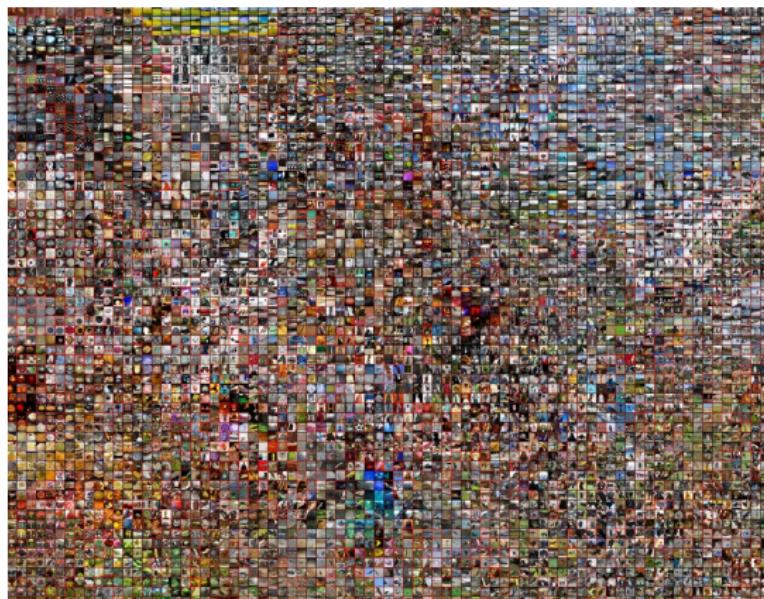
A convolutional network is divided into two parts:

- Feature extraction:
 - Convolution and pooling layers are alternated.
 - Computing intensive.
- Classification:
 - Fully connected network.
 - Parameter intensive.



Computer Vision

Representations.



Computer Vision

Representations.



Computer Vision

Representations.



Computer Vision

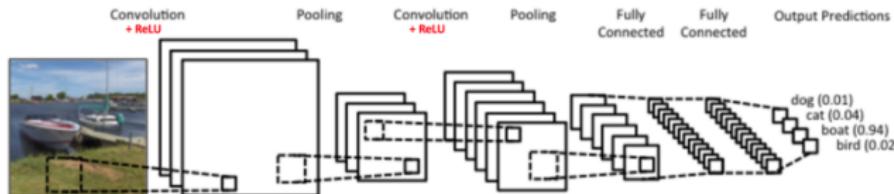
Representations.



Computer Vision

The LeNet architecture.

- LeNet was one of the very first convolutional neural networks.
- There have been several new architectures proposed in the recent years which are improvements over the LeNet, but they all use the main concepts from the LeNet and are relatively easier to understand if you have a clear understanding of the former.



Computer Vision

Convolutional networks are characterized by four operations:

- Convolution.
- Non Linearity (ReLU)
- Pooling or Sub-sampling, or down-sampling
- Classification, or MLP.

These operations are the basic building blocks of every Convolutional Neural Network, so understanding how these work is an important step to developing a sound understanding of ConvNets.

Computer Vision

Every image can be represented as a matrix of pixel values.

- Channel.

- A conventional term used to refer to a certain component of an image. An image from a standard digital camera will have three channels red, green and blue you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

- Grayscale.

- For the purpose of this post, we will only consider grayscale images, so we will have a single 2d matrix representing an image.

Computer Vision

Convolution step.

- The primary purpose of Convolution in case of a ConvNet is to extract features from the input image.
 - Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

1	1	1	0	0				
0	1	1	1	0				
0	0	1	1	1	1	0	1	1
0	0	1	1	0	0	1	0	0
0	1	1	0	0	1	0	1	

Computer Vision

Convolution step.

- In CNN terminology, the 3×3 matrix is called a "filter" or "kernel" or "feature detector" and the matrix formed by sliding the filter over the image and computing the dot product is called the "Convolved Feature" or "Activation Map" or the "Feature Map".
 - It is important to note that filters acts as feature detectors from the original input image.
- Clearly, different values of the filter matrix will produce different Feature Maps for the same input image.



Computer Vision

Convolution step.

- The effects of convolution of the above image with different filters.
 - Different filters can detect different features from an image, for example edges, curves etc.

	Operation	Filter	Convolved Image
Identity		$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
		$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
		$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen		$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (Normalized)		$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)		$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Computer Vision

Convolution step.

- In summary, a filter slides over the input image to produce a feature map.
 - The convolution of another filter over the same image gives a different feature map.
 - It is important to note that the Convolution operation captures the local dependencies in the original image.
 - Also, different filters generate different feature maps from the same original image.
- A CNN learns the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process).

Computer Vision

Convolution step.

- The size of the Feature Map (Convolved Feature) is controlled by three parameters:
 - Depth: Depth corresponds to the number of filters we use for the convolution operation.
 - Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. Having a larger stride will produce smaller feature maps.
 - Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix.

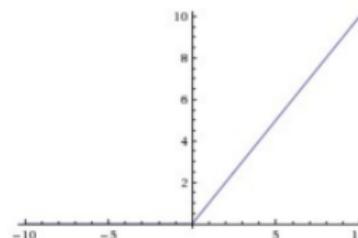


Computer Vision

Non-linearity (ReLU)

- An additional operation called ReLU has been used after every Convolution operation.
 - ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.
 - The purpose of ReLU is to introduce non-linearity in our ConvNet

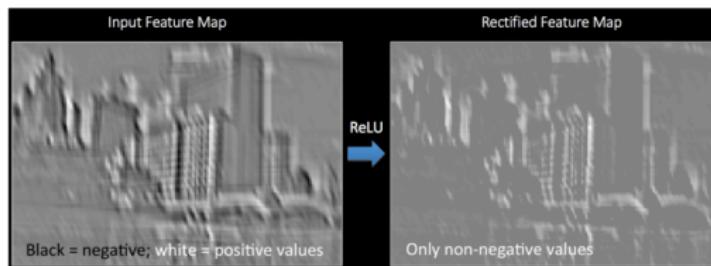
Output = Max(zero, Input)



Computer Vision

Non-linearity (ReLU)

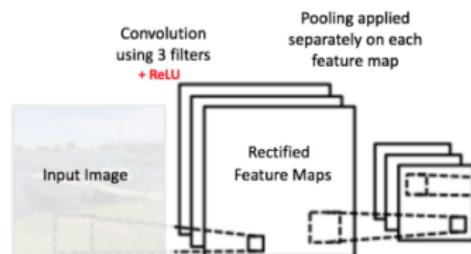
- ReLU is important because it does not saturate, the gradient is always high (equal to 1) if the neuron activates. As long as it is not a dead neuron, successive updates are fairly effective. ReLU is also very quick to evaluate.



Computer Vision

Max-pooling.

- Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information.
 - we define a spatial neighborhood (for example, a 2x2 window) and take the largest element from the rectified feature map within that window.
 - It forms a non-linear down-sampling (or subsampling).



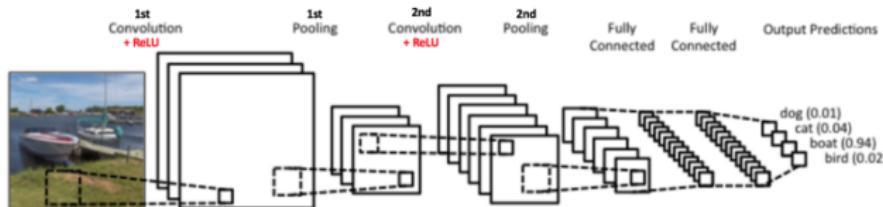
Computer Vision

Pooling.

- The function of Pooling is to progressively reduce the spatial size of the input representation.
 - It makes the input representations (feature dimension) smaller and more manageable
 - It reduces the number of parameters and computations in the network, therefore, controlling overfitting.
 - It makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling since we take the maximum / average value in a local neighborhood).
 - It helps us arrive at an almost scale invariant representation of our image. This is very powerful since we can detect objects in an image no matter where they are located.

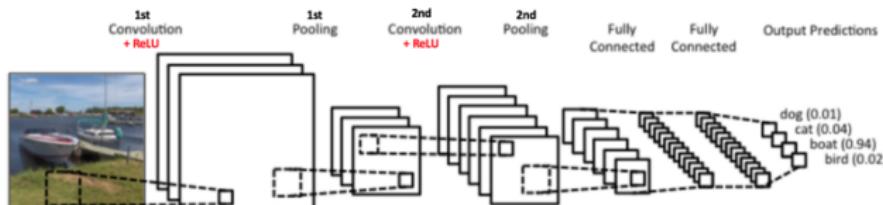
Computer Vision

The architecture of the network.



Computer Vision

The architecture of the network.

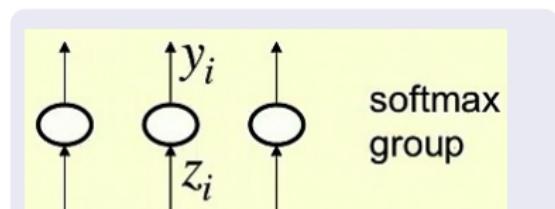


- Together these layers:
 - Extract the useful features from the images.
 - Introduce non-linearity in the network.
 - Reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation

Computer Vision

Classification.

- Multi Layer Perceptron that uses a softmax activation function in the output layer.
 - The output from the convolutional and pooling layers represent high-level features of the input image.
 - The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes.
- Output layer is a softmax non-linear function.
 - This is a way of forcing the outputs of the network to sum up to one so that they represent the probability distribution across mutually-exclusive alternatives.



- $y_i = \frac{e^{z_i}}{\sum_{j \in \text{group}} e^{z_j}}$

Computer Vision

The overall training process of the Convolution Network.

- ① Initialize all filters and parameters / weights with random values.
- ② The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
 - Since weights are randomly assigned for the first training example, output probabilities are also random.
- ③ Calculate the total error at the output layer.
 - Error = $\sum \frac{1}{2} (\text{target prob} - \text{output prob})$

Computer Vision

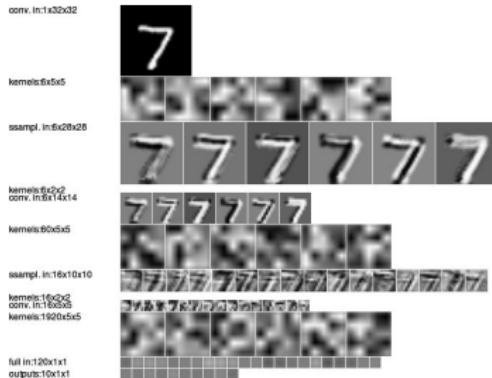
- ➊ Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values and weights and parameter values to minimize the output error.
 - The weights are adjusted in proportion to their contribution to the total error. When the same image is input again, output probabilities might now be closer to the target vector.
 - This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
- ➋ Repeat steps 2-4 with all images in the training set.

Computer Vision



- Initial kernels, and kernels after training the network.

Computer Vision

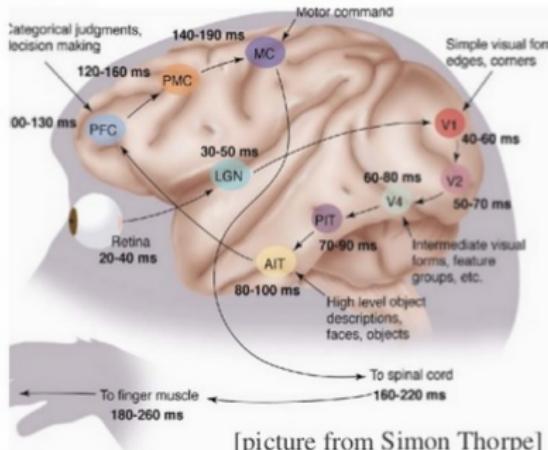
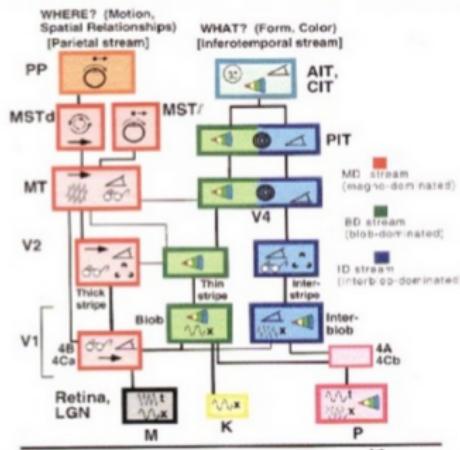


- Initial kernels, and kernels after training the network.

Computer Vision

Biological inspiration.

- The visual cortex is organized in layers that are connected to each other and interpret the signals received by the retina.



Natural Language Processing

NLP.

- The field is concerned with tasks involving language data.
 - Speech processing is also NLP, but we will focus on text data.
 - Ex: we will focus on language models.
- We will discuss neural networks specifically designed for learning language models.
- A language model is a function that captures the statistical characteristics of the distribution of sequences of words.
 - Typically allowing one to make probabilistic predictions of the next word given preceding ones.

Humans are very good in recognizing which words are likely to come next and which are not, because we use our understanding to the **meaning** of the sentence.

Natural Language Processing

There has been a long debate in cognitive science between two rival theories of what is “meaning”:

- The feature theory:
 - Meaning is given as a set of semantic features. This is good for explaining similarities between concepts with similar meanings.
- The structuralist theory:
 - The meaning of a concept lies in its relationship to other concepts (wordnet). So, different meanings are expressed as a relational graph.

Natural Language Processing

There has been a long debate in cognitive science between two rival theories of what is “meaning”:

- The feature theory:
 - Meaning is given as a set of semantic features. This is good for explaining similarities between concepts with similar meanings.
- The structuralist theory:
 - The meaning of a concept lies in its relationship to other concepts (wordnet). So, different meanings are expressed as a relational graph.
- We may use both theories together.
 - We may represent meaning as a vector of semantic features.
 - The features representing the meaning of a concept are calculated based on the relationship to other concepts.

Natural Language Processing

One-hot encoding.

- A basic representation using the word ID.
 - The one-hot vector is filled with 0's, except for a 1 at the position associated with the ID
 - Ex: For vocabulary size $D = 10$, the one-hot vector of word ID $w = 4$ is $e(w) = [0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$
 - A one-hot encoding makes no assumption about word similarity:
 - $\|e(w_1) - e(w_2)\|^2 = 0$ if $w_1 = w_2$
 - $\|e(w_1) - e(w_2)\|^2 = 2$ if $w_1 \neq w_2$
 - All words are equally different from each other.

Natural Language Processing

One-hot encoding.

- The major problem with the one-hot representation os that it is very high-dimensional.
 - The dimensionality of $e(w)$ is the size of the vocabulary ($\approx 100,000$).
 - A sequence of 10 words would correspond to an input of at least 1,000,000 units.
- Consequences:
 - Vulnerability to overfitting (many parameters/weights to learn)
 - Computationally expensive.

Natural Language Processing

The distributed representation.

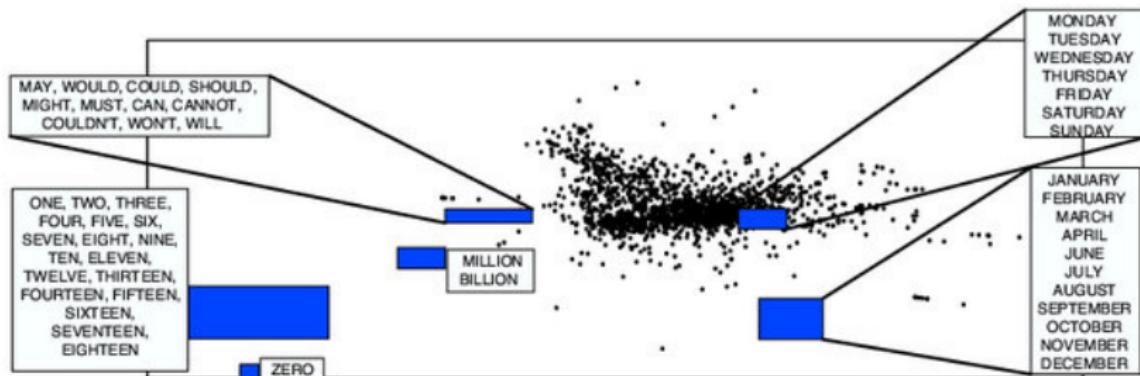
- A distributed representation of a word is a vector of features which characterize the meaning of the word.
 - The basic idea is to learn to associate each word in the text with a continuous-valued vector representation.
 - Each word corresponds to a point in a feature space, so that similar words get to be closer to each other in that space.

Natural Language Processing

The distributed representation.

- A distributed representation of a word is a vector of features which characterize the meaning of the word.
 - The basic idea is to learn to associate each word in the text with a continuous-valued vector representation.
 - Each word corresponds to a point in a feature space, so that similar words get to be closer to each other in that space.
 - A sequence of words can thus be transformed into a sequence of these learned feature vectors.
 - A neural network learns to map that sequence of feature vectors to a prediction of interest, such as the probability distribution over the next word in the sequence.

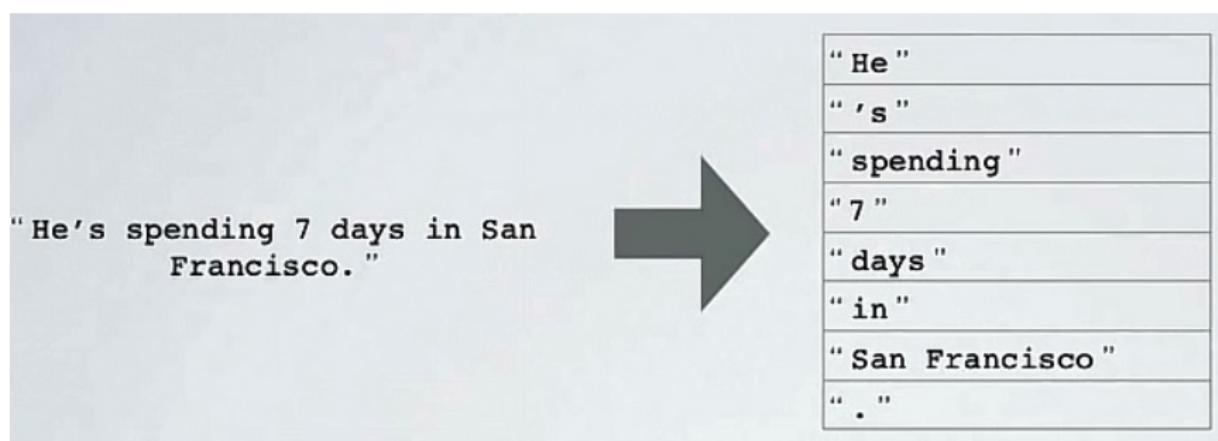
Natural Language Processing



Natural Language Processing

Preprocessing.

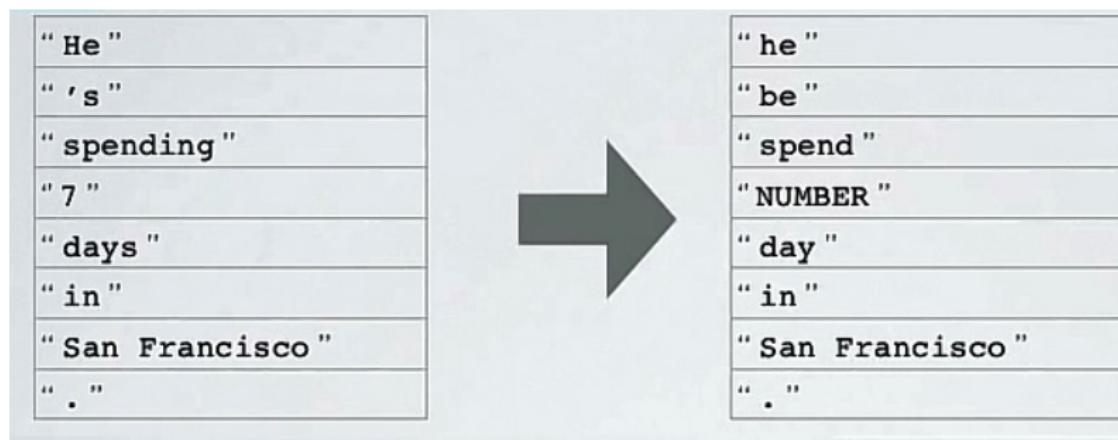
- Take text data in raw form, and put it in another form which is more convenient for neural network processing.
 - Tokenize text (from a long string to a list of token strings)



Natural Language Processing

Preprocessing.

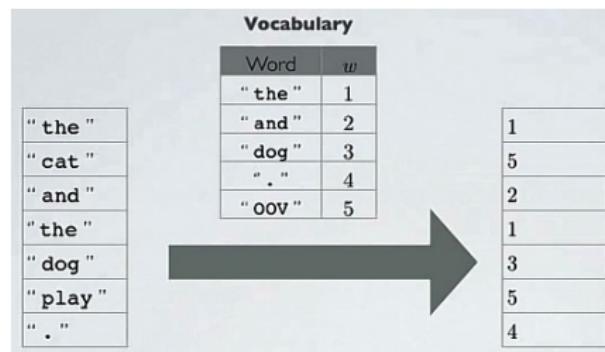
- Take text data in raw form, and put it in another form which is more convenient for neural network processing.
 - Lemmatize tokens (put it into standard form)



Natural Language Processing

Preprocessing.

- Take text data in raw form, and put it in another form which is more convenient for neural network processing.
 - Form a vocabulary of words that maps lemmatized words to a unique ID (the position of the word in the vocabulary).
 - Remove less frequent words.
 - Remove uninformative words from a pre-defined stop-word list.
 - Vocabulary sizes range from 10,000 words to 250,000 words.



Natural Language Processing

Distributed representation.

- Each word w is associated with a real-valued vector $C(w)$
- We would like the distance $\|C(w_1) - C(w_2)\|$ to reflect meaningful similarities between words.

Word	w	$C(w)$
" the "	1	[0.6762, -0.9607, 0.3626, -0.2410, 0.6636]
" a "	2	[0.6859, -0.9266, 0.3777, -0.2140, 0.6711]
" have "	3	[0.1656, -0.1530, 0.0310, -0.3321, -0.1342]
" be "	4	[0.1760, -0.1340, 0.0702, -0.2981, -0.1111]
" cat "	5	[0.5896, 0.9137, 0.0452, 0.7603, -0.6541]
" dog "	6	[0.5965, 0.9143, 0.0899, 0.7702, -0.6392]
" car "	7	[-0.0069, 0.7995, 0.6433, 0.2898, 0.6359]
...

Natural Language Processing

Distributed representation.

- We could use these representations as input to a neural network.
 - To represent a sequence of 10 words $[w_1, w_2, \dots, w_{10}]$, we concatenate the representation of each word:
$$X = [C(w_1)^T, C(w_2)^T, \dots, C(w_{10})^T]^T$$

Natural Language Processing

Distributed representation.

- We could use these representations as input to a neural network.
 - To represent a sequence of 10 words $[w_1, w_2, \dots, w_{10}]$, we concatenate the representation of each word:
$$X = [C(w_1)^T, C(w_2)^T, \dots, C(w_{10})^T]^T$$
- We learn these representations by stochastic gradient descent.
 - Not only the weights are updated.
 - We also update each representation $C(w)$ in the input X with a gradient step.
 - Representations $C(w)$ are initialized randomly.
 - $C(w) \leftarrow C(w) - r \nabla_{C(w)} L$, where L is a **loss function** we want to optimize (typically, negative log likelihood).
 - As the network is trained, it adapts not only the weights of the hidden units, but also the representations.

Natural Language Processing

Language modeling.

- A language model is a probabilistic model that assigns probabilities to any sequence of words.
 - Language modeling is the task of learning a language model that assigns high probabilities to well formed sentences.

Natural Language Processing

Language modeling.

- A language model is a probabilistic model that assigns probabilities to any sequence of words.
 - Language modeling is the task of learning a language model that assigns high probabilities to well formed sentences.
- An assumption that is frequently made is the n^{th} order Markov assumption.
 - $p(w_1, w_2, \dots, w_t) = p(w_1) \times p(w_2|w_1) \times \dots \times p(w_t|w_{t-1}, \dots, w_1)$
 - $p(w_1, w_2, \dots, w_t) = \sum_{i=1}^t p(w_i|w_1, w_2, \dots, w_{i-1})$
 - The t^{th} word was generated from the $n - 1$ previous words.
 - We will refer to $(w_1, w_2, \dots, w_{t-1})$ as the context.

Natural Language Processing

Language modeling.

- A language model is a probabilistic model that assigns probabilities to any sequence of words.
 - Language modeling is the task of learning a language model that assigns high probabilities to well formed sentences.
- An assumption that is frequently made is the n^{th} order Markov assumption.
 - $p(w_1, w_2, \dots, w_t) = p(w_1) \times p(w_2|w_1) \times \dots \times p(w_t|w_{t-1}, \dots, w_1)$
 - $p(w_1, w_2, \dots, w_t) = \sum_{i=1}^t p(w_i|w_1, w_2, \dots, w_{i-1})$
 - The t^{th} word was generated from the $n - 1$ previous words.
 - We will refer to $(w_1, w_2, \dots, w_{t-1})$ as the context.
- Main issue: we want n to be large.
 - However, for large values of n , it is likely that a given n -gram will not have been observed in the training corpus.

Natural Language Processing

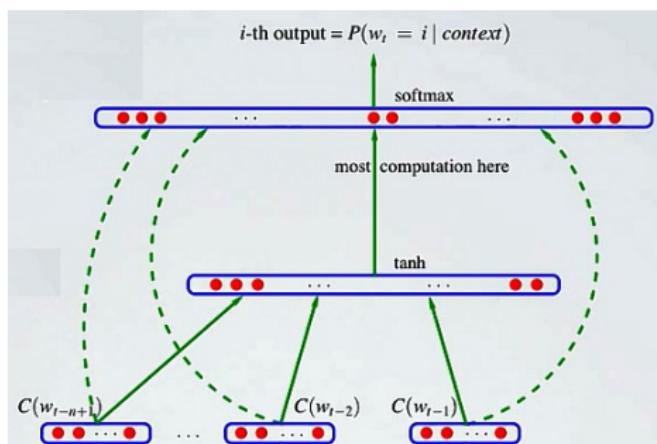
Language modeling.

- The n -gram model is a sequence of n words.
 - unigrams ($n = 1$):
 - "is", "a", "sequence" etc.
 - bigrams ($n = 2$):
 - ["is, "a"], ["a", "sequence"] etc.
 - trigrams ($n = 3$):
 - ["is, "a", "sequence"]", ["a", "sequence", "of"] etc.
- n -gram models estimate the conditional probability from n -gram counts:
 - $p(w_t | w_{t-1}, \dots, w_2, w_1) = \frac{\text{count}(w_t, w_{t-1}, \dots, w_2, w_1)}{\text{count}(w_{t-1}, \dots, w_2, w_1)}$
 - These counts are obtained from a training corpus.

Natural Language Processing

Neural network language model.

- Model the conditional probability $p(w_t | w_1, w_2, \dots, w_{t-1})$ with a neural network.
 - The input is a concatenation of the word representations.
 - The output is the probability of each word to be the next one.



Natural Language Processing

Neural network language model.

- Advantage: can potentially generalize to contexts not seen in the training corpus.
 - $p(\text{"eating"} | \text{"the"}, \text{"cat"}, \text{"is"})$
 - Imagine 4-gram $[\text{"the"}, \text{"cat"}, \text{"is"}, \text{"eating"}]$ is not in the corpus, but $[\text{"the"}, \text{"dog"}, \text{"is"}, \text{"eating"}]$ is.
 - If the word representations for "cat" and "dog" are similar, the neural network would be able to generalize to the case of "cat".

Natural Language Processing

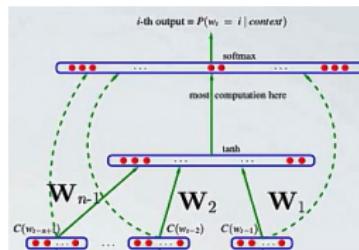
Neural network language model.

- Advantage: can potentially generalize to contexts not seen in the training corpus.
 - $p(\text{"eating"} | \text{"the"}, \text{"cat"}, \text{"is"})$
 - Imagine 4-gram $[\text{"the"}, \text{"cat"}, \text{"is"}, \text{"eating"}]$ is not in the corpus, but $[\text{"the"}, \text{"dog"}, \text{"is"}, \text{"eating"}]$ is.
 - If the word representations for "cat" and "dog" are similar, the neural network would be able to generalize to the case of "cat".
 - Why? Because of other 4-grams that are in the corpus.
 - $[\text{"the"}, \text{"cat"}, \text{"was"}, \text{"sleeping"}]$
 - $[\text{"the"}, \text{"dog"}, \text{"was"}, \text{"sleeping"}]$

Natural Language Processing

Neural network language modeling.

- Gradients to update word representations.
 - We know how to compute the gradient for the hidden layers:
 $\nabla_{a(X)} L$ (i.e., backpropagation)
 - Note the vector W_i , which connects word w_i and the corresponding hidden layer.
 - The gradient with regard to $C(w_i)$ for any word w_i is:
 - $\nabla_{C(w)} L = \sum_{i=1}^n 1_{(w_i=w)} W_i^T \nabla_{a(X)} L$



Natural Language Processing

Neural network language modeling.

- Ex: [“the”, “dog”, “and”, “the”, “cat”]
 - Context: [“the”, “dog”, “and”, “the”]
 - $w_3 = 21$ (“the”) $w_4 = 3$ (“dog”) $w_5 = 14$ (“and”) $w_6 = 21$ (“the”)
- The loss is:
 - $-\log p(\text{“cat”} | \text{“the”, “dog”, “and”, “the”})$
 - $\nabla_{C(3)} L = W_3^T \nabla_{a(X)} L$
 - $\nabla_{C(14)} L = W_2^T \nabla_{a(X)} L$
 - $\nabla_{C(21)} L = W_1^T \nabla_{a(X)} L + W_4^T \nabla_{a(X)} L$
 - $\nabla_{C(w)} L = 0$ for all other w not in the context.
- Finally:
 - $C(w) \Leftarrow C(w) - r \nabla_{C(w)} L$

Natural Language Processing



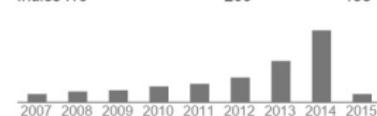
Yoshua Bengio

[Seguir](#) ▾

Professor, U. Montreal, Computer Sc. & Op. Res., member/Fellow of CIFAR, CRM, REPARTI, GRSNC, CIRANO
Machine learning, deep learning, artificial intelligence
 E-mail confirmado em umontreal.ca - [Página inicial](#)

Google Acadêmico

Índices de citações	Todos	Desde 2010
Citações	23202	15948
Índice h	60	52
Índice i10	209	158



Índice	Titúlo	1–20	Citado por	Ano
Gradient-based learning applied to document recognition	Y LeCun, L Bottou, Y Bengio, P Haffner		3140	1998
Proceedings of the IEEE 86 (11), 2278-2324				
Learning deep architectures for AI	Y Bengio		1345	2009
Foundations and trends® in Machine Learning 2 (1), 1-127				

Coautores [Visualizar todos...](#)

Aaron Courville

Natural Language Processing

t-distributed stochastic neighbor embedding (t-SNE).

- t-SNE is a nonlinear dimensionality reduction technique.
 - It is particularly well suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot.
 - It models each high-dimensional vector by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points.



Natural Language Processing

k-Nearest Neighbors and word embeddings.

- Returns the k closest points to a given point.

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Natural Language Processing

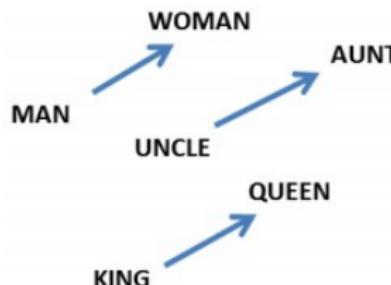
k-Nearest Neighbors and word embeddings.

- Returns the k closest points to a given point.
 - Give me a word like “king”, like “woman”, but unlike “man”:

Natural Language Processing

k-Nearest Neighbors and word embeddings.

- Returns the k closest points to a given point.
 - Give me a word like "king", like "woman", but unlike "man":
 - "queen"
 - Why?
 - $C(\text{"woman"}) - C(\text{"man"}) \approx C(\text{"aunt"}) - C(\text{"uncle"})$
 - $C(\text{"woman"}) - C(\text{"man"}) \approx C(\text{"queen"}) - C(\text{"king"})$
 - Directions in the embedding space seems to have semantic meaning.



Natural Language Processing

k-Nearest Neighbors and word embeddings.

- Returns the k closest points to a given point.
 - What is the capital city of Italy:
 - “Rome”
 - Why?

Natural Language Processing

k-Nearest Neighbors and word embeddings.

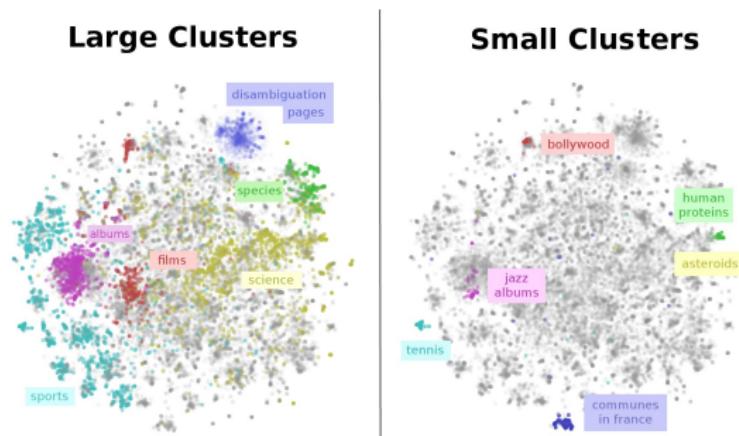
- Returns the k closest points to a given point.
 - What is the capital city of Italy:
 - “Rome”
 - Why?
 - $C(\text{"france"}) - C(\text{"paris"}) \approx C(\text{"italy"}) - C(\text{"paris"})$
 - Difference vectors between words seem to encode analogies

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Natural Language Processing

k-Nearest Neighbors and paragraph embeddings.

- Paragraph vectors represent chunks of text
 - With word embeddings, the network learns vectors for words.
 - With paragraph embeddings, the network learns vectors for paragraphs.
 - A map of Wikipedia.



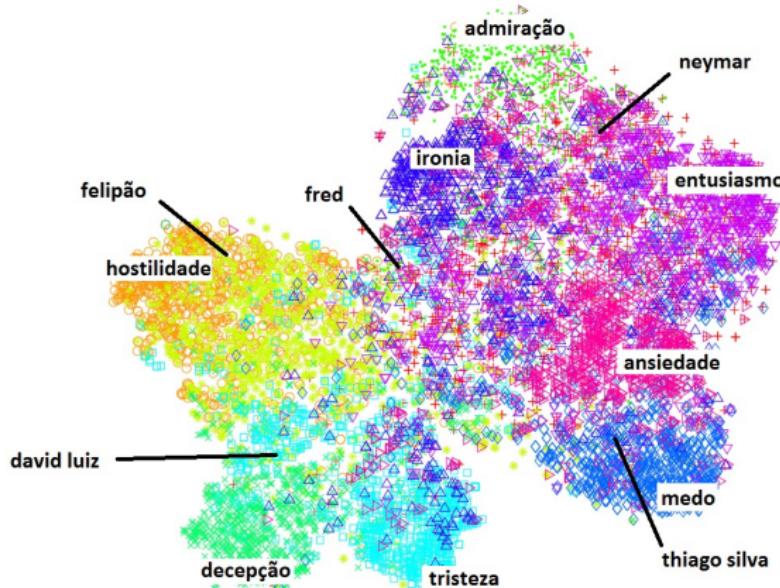
Natural Language Processing

- It is important to appreciate that all of these properties of $C(w)$ are side effects.
 - We did not try to have similar words be close together.
 - We did not try to have analogies encoded with difference vectors.
- All we did was to predict the next word in a sentence.
 - These properties popped out of the network optimization process.
- The use of word representations has become of key importance for the success of many NLP systems.
 - Named entity recognition, part-of-speech tagging, translation, parsing etc.

Example

Information extraction from language data.

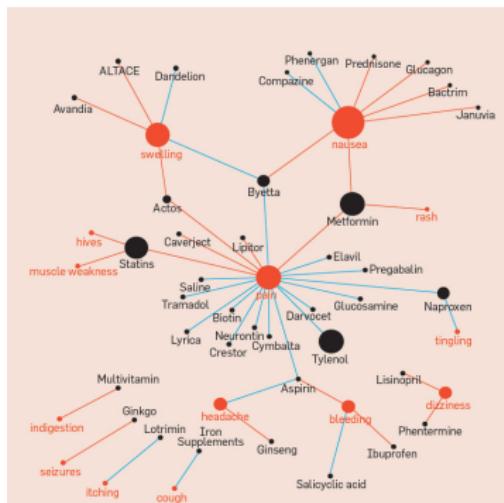
- World cup.



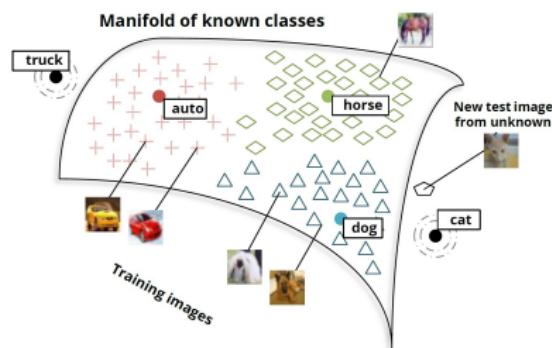
Example

Information extraction from health blogs.

- Diabetes.
 - Therapy which efficiency is close to “Metformin”, and side effects are similar to “Lipitor”.



Natural Language Processing



Unsupervised Feature Learning

So far we have seen neural networks that learn to predict a particular target from a set of inputs.

- Unsupervised feature learning:
 - The input data only contains $X^{(t)}$ for learning
- We will see two neural networks for unsupervised feature learning:
 - Restricted Boltzmann Machines
 - Autoencoders
 - Generative models: rather than predicting a label the objective is to model the input data.
 - Maximize $p(X)$ instead of $p(Y|X)$.
 - The objective is to learn interesting representations of the input data (think in it as a hash).
 - We want a function θ , so that $\theta(X) \approx X$.

Restricted Boltzmann Machine

Modeling binary data.

- Given a training set of binary vectors, fit a model that will assign a probability to every possible binary vector.
 - Goal: make the model to have high probability on training examples.
 - Learn a probability distribution over a set of inputs.
 - It may be useful, for example, for monitoring complex systems to detect unusual behavior.
- RBMs have found many application scenarios:
 - Dimensionality reduction, classification, collaborative filtering, feature learning and topic modelling.

Restricted Boltzmann Machine

Suppose you ask some people to rate a set of movies.

- Put a “1” if she likes the movie, and “0” otherwise.

There are latent factors:

- Movies like “Star Wars” and “Lord of the Rings” might have strong associations with a latent science fiction and fantasy factor, and people who like “Wall-E” and “Toy Story” might have strong associations with a latent Pixar factor.

An RBM is a bipartite undirected graph, composed of:

- A visible layer $X = \{x_1, x_2, \dots, x_n\}$ (e.g., movies).
- A hidden layer $H = \{h_1, h_2, \dots, h_m\}$ (e.g., latent factors).
- A bias unit to adjust for the different inherent popularities of each movie.

Further, each x_i is connected to all h_j 's

- A weight w_{ij} connects x_i to h_j .

Restricted Boltzmann Machine

- Latent variables or hidden variables, as opposed to observable variables, are variables that are not directly observed but are rather inferred from other variables that are observed.
 - Latent variable models: explain observed variables in terms of latent variables.

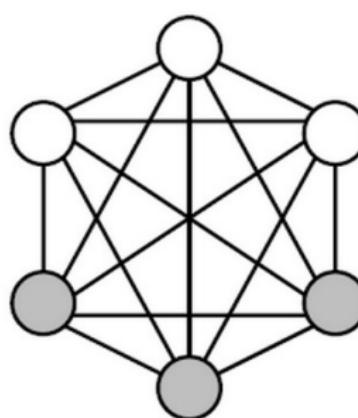
Restricted Boltzmann Machine

- Latent variables or hidden variables, as opposed to observable variables, are variables that are not directly observed but are rather inferred from other variables that are observed.
 - Latent variable models: explain observed variables in terms of latent variables.
- Advantage of using latent variables:
 - It reduces the dimensionality of data.
 - A large number of observable variables can be aggregated in a model to represent an underlying concept, making it easier to understand the data.

Restricted Boltzmann Machine

- Latent variables or hidden variables, as opposed to observable variables, are variables that are not directly observed but are rather inferred from other variables that are observed.
 - Latent variable models: explain observed variables in terms of latent variables.
- Advantage of using latent variables:
 - It reduces the dimensionality of data.
 - A large number of observable variables can be aggregated in a model to represent an underlying concept, making it easier to understand the data.
- Examples of latent variables from economics:
 - Quality of life, business confidence, morale and happiness.
 - These are all variables which cannot be measured directly.

Restricted Boltzmann Machine



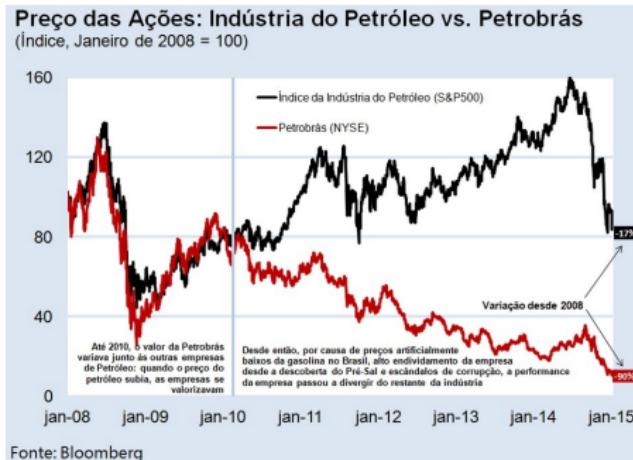
Restricted Boltzmann Machine



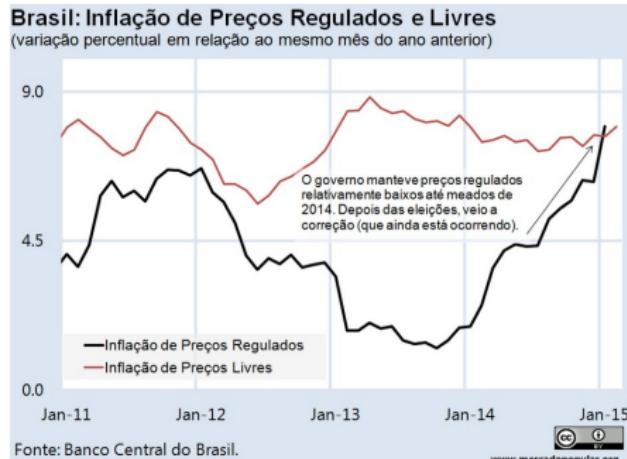
Restricted Boltzmann Machine



Restricted Boltzmann Machine



Restricted Boltzmann Machine



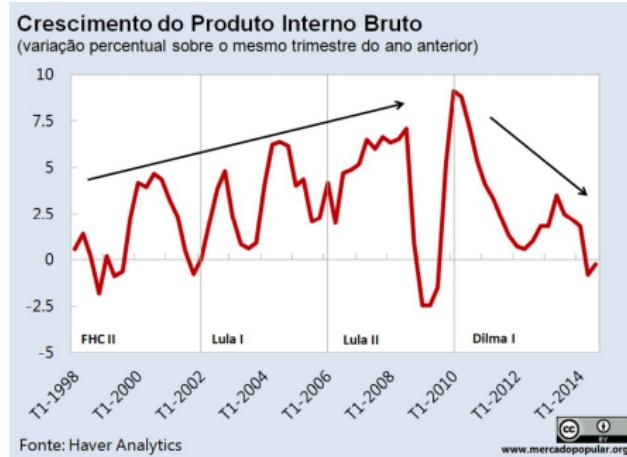
Restricted Boltzmann Machine



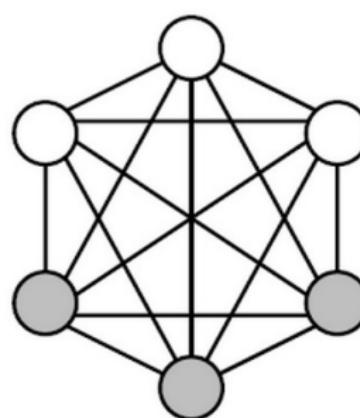
Restricted Boltzmann Machine



Restricted Boltzmann Machine



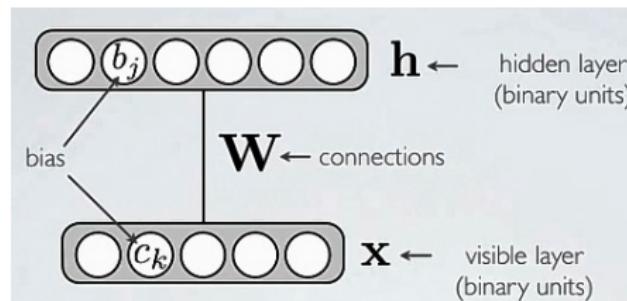
Restricted Boltzmann Machine



Restricted Boltzmann Machine

Energy based model.

- The RBM defines a distribution over X , (i.e., $p(X, H)$).
 - That distribution involves latent variables/factors (h_j 's).



- The probability of generating a visible vector X is computed by summing over all possible hidden units.
 - Each hidden unit h_j is seen as an explanation of X (i.e., a latent factor).

Restricted Boltzmann Machine

Energy based model.

- Inspired by the work of Ludwig Boltzmann who studied the statistical mechanics of gases in thermal equilibrium.
 - Basically, high energies tend to appear in chaotic states.
 - Higher entropies.
- Later, Shannon used these concepts to create the basis for information theory.
 - Boltzmann developed the natural form of entropy, and the rescaled entropy exactly corresponds to Shannon's subsequent information entropy.
 - Boltzmann developed a probability distribution, which is similar to the Gibbs distribution.
- Put simply:
 - Probability density appears in low-entropy/energy states.

Restricted Boltzmann Machine

Energy based model.

- The generation (or reconstruction) of the visible units is defined entirely in terms of the energies of joint configurations of the visible and hidden units.
 - The energies of joint configurations are related to their probabilities.
 - That is, the probability is the chance of finding the network in that joint configuration after we have updated all the units many times.
- Thus, in order to calculate $p(X, H)$ we need to define:
 - Energy: $E(X, H) = \sum_i \sum_j w_{i,j} h_j x_i - \sum_j c_j x_j - \sum_i b_i h_i$
 - Distribution: $p(X, H) = \frac{\exp(-E(X, H))}{Z}$ (**Z is a normalizer**)
 - High energy is associated with low probabilities.

Restricted Boltzmann Machine

For example:

- Suppose we have a set of six movies:
 - “Harry Potter”, “Avatar”, “LOTR 3”, “Gladiator”, “Titanic”, and “Glitter”
- We ask people to tell us which ones they want to watch.
 - We want to learn two latent units underlying movie preferences.
 - Two natural groups in our set of six movies appear to be SF/fantasy (“Harry Potter”, “Avatar”, and “LOTR 3”) and Oscar winners (“LOTR 3”, “Gladiator”, and “Titanic”)
 - We might hope that our latent units will correspond to these categories.

Restricted Boltzmann Machines

Inference with RBMs.

- But, computing $p(X, H)$ is intractable in RBMs:
 - Because of Z , which involves a sum over all possible configurations.
 - There is an exponential number of possible configurations.

Restricted Boltzmann Machines

Inference with RBMs.

- But, computing $p(X, H)$ is intractable in RBMs:
 - Because of Z , which involves a sum over all possible configurations.
 - There is an exponential number of possible configurations.
- There are other types of inference that are tractable:
 - Conditional inference (there are no interactions between x_i 's and h_j 's, they are conditionally independent):
 - $p(h_j = 1|X) = \text{sigmoid}(b_j + w_{j\cdot}X)$
 - $p(x_i = 1|H) = \text{sigmoid}(c_i + H^T w_{i\cdot})$
 - These inferences are tractable because the RBM has the shape of a bipartite graph, with no intra-layer connections.
 - The hidden units are mutually independent given the visible units and conversely, the visible units are mutually independent given the hidden units.

Restricted Boltzmann Machine

Loss function.

- An energy-based model can be learnt by performing stochastic gradient descent on the empirical negative log-likelihood (or log-probability) of the training set.
 - Loss function: $L(W) = \frac{1}{T} \sum_t -\log p(X^{(t)})$
- Generative model: maximize $p(X)$ instead of $p(Y|X)$.
- Stochastic gradient descent:
 - The gradient is:
$$\frac{\partial -\log p(X^{(t)})}{\partial W} = E_H \left[\frac{\partial E(X^{(t)}, H)}{\partial W} | X^{(t)} \right] - E_{X, H} \left[\frac{\partial E(X, H)}{\partial W} \right]$$
 - Positive and negative phases.
 - Positive phase is a sum over all h_j 's

Restricted Boltzmann Machine

Loss function.

- An energy-based model can be learnt by performing stochastic gradient descent on the empirical negative log-likelihood (or log-probability) of the training set.
 - Loss function: $L(W) = \frac{1}{T} \sum_t -\log p(X^{(t)})$
- Generative model: maximize $p(X)$ instead of $p(Y|X)$.
- Stochastic gradient descent:
 - The gradient is:
$$\frac{\partial -\log p(X^{(t)})}{\partial W} = E_H \left[\frac{\partial E(X^{(t)}, H)}{\partial W} | X^{(t)} \right] - E_{X, H} \left[\frac{\partial E(X, H)}{\partial W} \right]$$
 - Positive and negative phases.
 - Positive phase is a sum over all h_j 's
 - But, negative phase is an exponential sum.
 - In order to proceed with SGD, we must approximate the gradient (i.e., negative phase is intractable).

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.
- ① Take a training example X , compute the probabilities $p(h_j = 1|X) \forall h_j$ s.

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.
- ① Take a training example X , compute the probabilities $p(h_j = 1|X) \forall h_j$ s.
- ② Set h_j to 1 with probability $p(h_j = 1|X)$, and to 0 with $1 - p(h_j = 1|X)$.

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.
- ① Take a training example X , compute the probabilities $p(h_j = 1|X) \forall h_j$ s.
- ② Set h_j to 1 with probability $p(h_j = 1|X)$, and to 0 with $1 - p(h_j = 1|X)$.
- ③ For each edge e_{ij} compute the **positive gradient** $P(e_{ij}) = x_i \times h_j$ (i.e., measure whether both units are on).

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.
- ① Take a training example X , compute the probabilities $p(h_j = 1|X) \forall h_j's$.
- ② Set h_j to 1 with probability $p(h_j = 1|X)$, and to 0 with $1 - p(h_j = 1|X)$.
- ③ For each edge e_{ij} compute the **positive gradient** $P(e_{ij}) = x_i \times h_j$ (i.e., measure whether both units are on).
- ④ Now, reconstruct the visible units in a similar manner. Compute the probabilities $p(x_i = 1|H) \forall x_i's$.

Restricted Boltzmann Machine

Training RBMs.

- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.

 - ① Take a training example X , compute the probabilities $p(h_j = 1|X)\forall h'_j s$.
 - ② Set h_j to 1 with probability $p(h_j = 1|X)$, and to 0 with $1 - p(h_j = 1|X)$.
 - ③ For each edge e_{ij} compute the **positive gradient** $P(e_{ij}) = x_i \times h_j$ (i.e., measure whether both units are on).
 - ④ Now, reconstruct the visible units in a similar manner. Compute the probabilities $p(x_i = 1|H)\forall x'_i s$. Set x_i to 1 with probability $p(x_i = 1|H)$, and to 0 with $1 - p(x_i = 1|H)$. Then reconstruct the hidden units.
 - ⑤ For each edge e_{ij} compute the **negative gradient** $N(e_{ij}) = x_i \times h_j$.

Restricted Boltzmann Machine

Training RBMs.

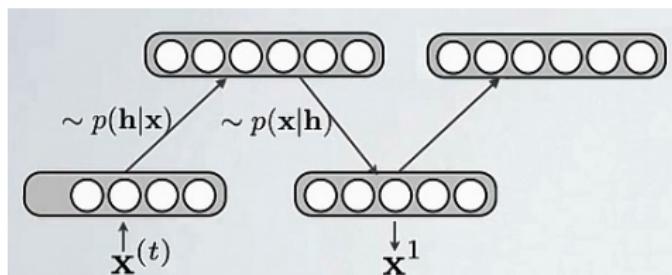
- Contrastive divergence algorithm (optimize W).
 - It relies on an approximation of the gradient (a good direction of change for the parameters) of the log-likelihood (the basic criterion optimized by probabilistic learning algorithms).
 - The algorithm performs Gibbs sampling (i.e., sample X then H iteratively) and uses SGD.
- ① Take a training example X , compute the probabilities $p(h_j = 1|X)\forall h'_j s$.
- ② Set h_j to 1 with probability $p(h_j = 1|X)$, and to 0 with $1 - p(h_j = 1|X)$.
- ③ For each edge e_{ij} compute the **positive gradient** $P(e_{ij}) = x_i \times h_j$ (i.e., measure whether both units are on).
- ④ Now, reconstruct the visible units in a similar manner. Compute the probabilities $p(x_i = 1|H)\forall x'_i s$. Set x_i to 1 with probability $p(x_i = 1|H)$, and to 0 with $1 - p(x_i = 1|H)$. Then reconstruct the hidden units.
- ⑤ For each edge e_{ij} compute the **negative gradient** $N(e_{ij}) = x_i \times h_j$.
- ⑥ $w_{ij} \leftarrow w_{ij} + r \times (P(e_{ij}) - N(e_{ij}))$

Restricted Boltzmann Machine

Why does this update rule make sense?

- Note that:
 - In the positive phase, $P(e_{ij})$ measures the association between the x_i and h_j that we want the network to learn from our training examples (i.e., is coherent with the data).
 - In the negative (or reconstruction) phase, where the RBM generates the states of visible units based on its hypotheses about the hidden units alone, $N(e_{ij})$ measures the association that the network itself generates (“daydreams”) when no units are fixed to the training set (i.e., is coherent with the model).
- So, by adding $P(e_{ij}) - N(e_{ij})$ to each weight w_{ij} , we are helping the networks daydreams better match the reality of the training examples.

Restricted Boltzmann Machine



Vector form:

- ① For each training example $X^{(t)}$
 - ① Generate a negative sample X' using k steps of Gibbs sampling, starting at $X^{(t)}$
 - ② Update weights:
 - $W \leftarrow W + r \times (H \cdot X^{(t)} - H' \cdot X')$
 - $b \leftarrow b + r \times (H - H')$
 - $c \leftarrow c + r \times (X^{(t)} - X')$
- ② Pick the next example until convergence is achieved

Restricted Boltzmann Machine

Contrastive divergence.

- CD- k : contrastive divergence with k iterations of Gibbs sampling.
 - In general, the bigger k is, the less biased the estimate of the gradient will be.
 - In practice, however, $k = 1$ works surprisingly well.

Restricted Boltzmann Machine

Example:

- Suppose the following training examples:

Table: Training set

	Harry Potter	Avatar	LOTR 3	Gladiator	Titanic	Glitter
$X^{(1)}$	1	1	1	0	0	0
$X^{(2)}$	1	0	1	0	0	0
$X^{(3)}$	1	1	1	0	0	0
$X^{(4)}$	0	0	1	1	1	0
$X^{(5)}$	0	0	1	1	1	0
$X^{(6)}$	0	0	1	1	1	0

Restricted Boltzmann Machine

- $X^{(1)}$ is a big SF/fantasy fan.
- $X^{(2)}$ is a SF/fantasy fan, but does not like “Avatar”.
- $X^{(3)}$ is a big SF/fantasy fan.
- $X^{(4)}$ is a big Oscar winners fan.
- $X^{(5)}$ is an Oscar winners fan, except for “Titanic”.
- $X^{(6)}$ is a big Oscar winners fan.

Restricted Boltzmann Machine

Table: Weights

	Bias (b'_i s)	h_1	h_2
Bias (c'_j s)		-0.19041546	1.57007782
Harry Potter	-0.82602559	-7.08986885	4.96606654
Avatar	-1.84023877	-5.18354129	2.27197472
LOTR 3	3.92321075	2.51720193	4.11061382
Gladiator	0.10316995	6.74833901	-4.00505343
Titanic	-0.97646029	3.25474524	-5.59606865
Glitter	-4.44685751	-2.81563804	-2.91540988

- What is h_1 ? And h_2 ?

Restricted Boltzmann Machine

Table: Weights

	Bias (b'_i s)	h_1	h_2
Bias (c'_j s)		-0.19041546	1.57007782
Harry Potter	-0.82602559	-7.08986885	4.96606654
Avatar	-1.84023877	-5.18354129	2.27197472
LOTR 3	3.92321075	2.51720193	4.11061382
Gladiator	0.10316995	6.74833901	-4.00505343
Titanic	-0.97646029	3.25474524	-5.59606865
Glitter	-4.44685751	-2.81563804	-2.91540988

- What is h_1 ? And h_2 ?
 - h_1 is Oscar winners.
 - h_2 is SF/fantasy.

Restricted Boltzmann Machine

Table: Weights

	Bias (b_i' s)	h_1	h_2
Bias (c_j' s)		-0.19041546	1.57007782
Harry Potter	-0.82602559	-7.08986885	4.96606654
Avatar	-1.84023877	-5.18354129	2.27197472
LOTR 3	3.92321075	2.51720193	4.11061382
Gladiator	0.10316995	6.74833901	-4.00505343
Titanic	-0.97646029	3.25474524	-5.59606865
Glitter	-4.44685751	-2.81563804	-2.91540988

- What happens if we give the RBM a new person who has $[0, 0, 0, 1, 1, 0]$ as his preferences?

Restricted Boltzmann Machine

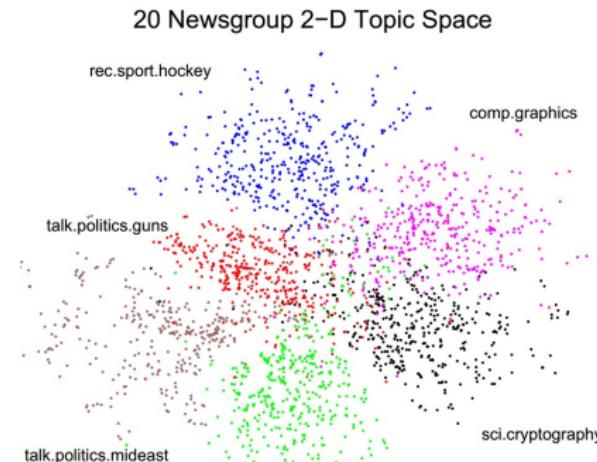
Table: Weights

	Bias (b_i' s)	h_1	h_2
Bias (c_j' s)		-0.19041546	1.57007782
Harry Potter	-0.82602559	-7.08986885	4.96606654
Avatar	-1.84023877	-5.18354129	2.27197472
LOTR 3	3.92321075	2.51720193	4.11061382
Gladiator	0.10316995	6.74833901	-4.00505343
Titanic	-0.97646029	3.25474524	-5.59606865
Glitter	-4.44685751	-2.81563804	-2.91540988

- What happens if we give the RBM a new person who has $[0, 0, 0, 1, 1, 0]$ as his preferences?
 - It turns h_1 on, but not h_2 .

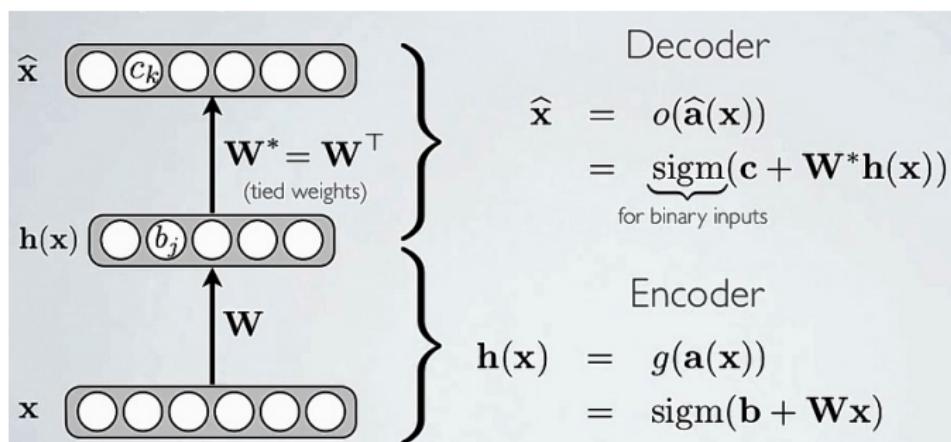
Restricted Boltzmann Machine

- Visible units:
 - Word occurrence in the document.
- Hidden units:
 - Topic detectors.



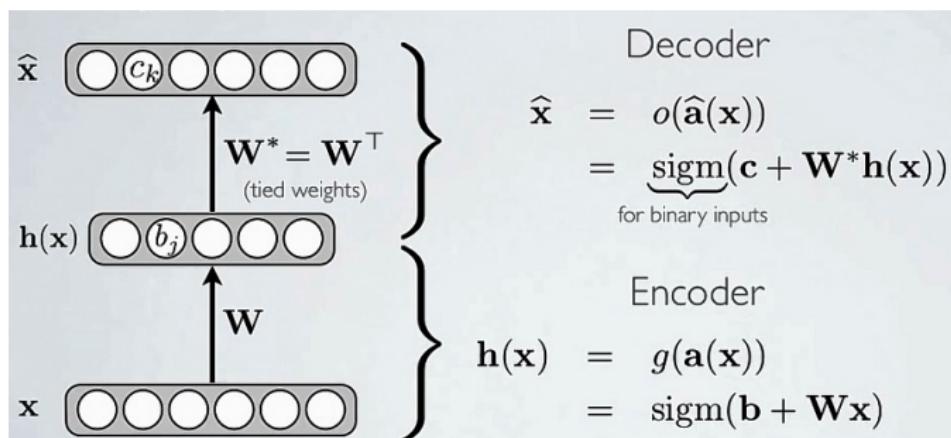
Autoencoder

- Feedforward network:
 - Supervised learning algorithm to do unsupervised feature learning
 - It is trained to reproduce its inputs in the output layer.
 - The output layer has the same size of the input layer.



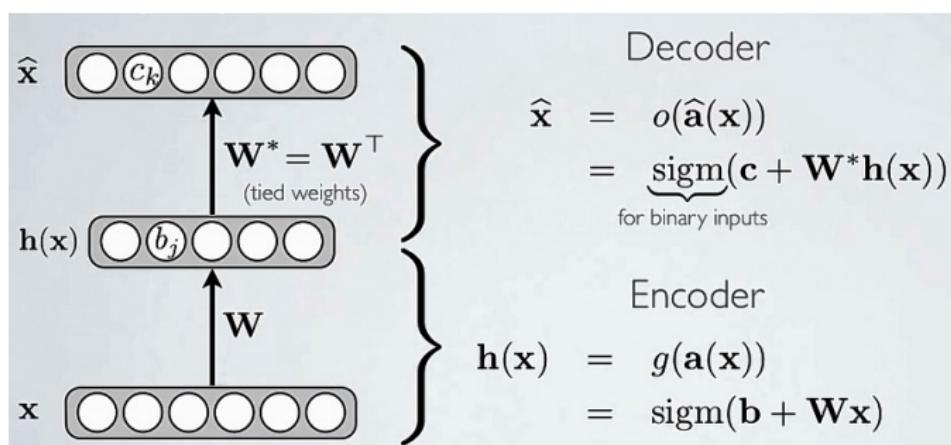
Autoencoder

- The objective is to learn $h(x)$, which is the latent representation for the inputs.
 - Encoder (W) and decoder (W^*) are tied.
 - This means that $W^* = W^T$.



Autoencoder

- The main motivation is to make $h(x)$ to maintain all the information about the input.
 - If the hidden layer is smaller than the input layer, then the autoencoder is going to compress the inputs.



Autoencoder

Loss function.

- We need something to optimize.

Autoencoder

Loss function.

- We need something to optimize.
 - How well the network reproduces the inputs.

Autoencoder

Loss function.

- We need something to optimize.
 - How well the network reproduces the inputs.
 - Stochastic optimization with gradient descent.
- Loss for binary inputs:
 - $L(W) = - \sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k))$
 - Cross-entropy: it minimizes if $x_i = \hat{x}_i$.
- Loss for real-valued inputs:
 - $L(W) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$
 - Sum of squared differences (squared Euclidean distance).
- For both cases, the gradient $\nabla_W L$ has a very simple form:
 - $\nabla_W L = \hat{x}^{(t)} - x^{(t)}$

Autoencoder

Loss function.

- We need something to optimize.
 - How well the network reproduces the inputs.
 - Stochastic optimization with gradient descent.
- Loss for binary inputs:
 - $L(W) = - \sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k))$
 - Cross-entropy: it minimizes if $x_i = \hat{x}_i$.
- Loss for real-valued inputs:
 - $L(W) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$
 - Sum of squared differences (squared Euclidean distance).
- For both cases, the gradient $\nabla_W L$ has a very simple form:
 - $\nabla_W L = \hat{x}^{(t)} - x^{(t)}$
 - Weights are updated by backpropagation.

Autoencoder

Undercomplete and overcomplete hidden layers.

- Autoencoders are learned to create compressed representation of information that is coming from inputs.
 - Compression here is considered in sense of “compressed with lost of information”, that is, autoencoders extract the most important information and convert it to new compressed, less dimensional form (i.e. representation).

Autoencoder

Undercomplete and overcomplete hidden layers.

- Autoencoders are learned to create compressed representation of information that is coming from inputs.
 - Compression here is considered in sense of “compressed with lost of information”, that is, autoencoders extract the most important information and convert it to new compressed, less dimensional form (i.e. representation).
- What is the impact of the size of the hidden layer?
 - Impact in terms of compression, that is, the types of features the autoencoder will learn.

Autoencoder

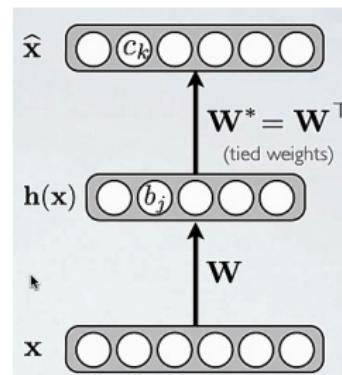
Undercomplete representation.

- Hidden layer is smaller than the input layer.

Autoencoder

Undercomplete representation.

- Hidden layer is smaller than the input layer.
 - In this case, hidden layer compresses the input
 - It is trained to compress well only inputs that are generated following the training distribution
 - The representation works well for objects like the ones in the training set



Autoencoder

Overcomplete representation.

- Hidden layer is larger than the input layer.

Autoencoder

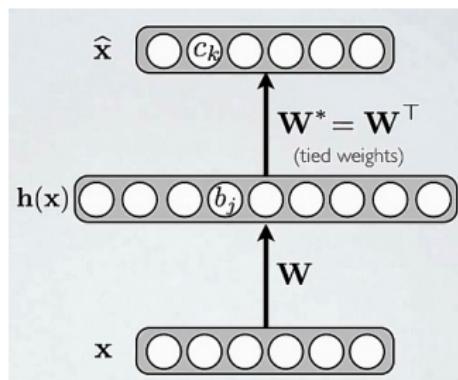
Overcomplete representation.

- Hidden layer is larger than the input layer.
 - In this case, hidden layer does not compress the input
 - No guarantee that the hidden units will extract a meaningful structure from the inputs.

Autoencoder

Overcomplete representation.

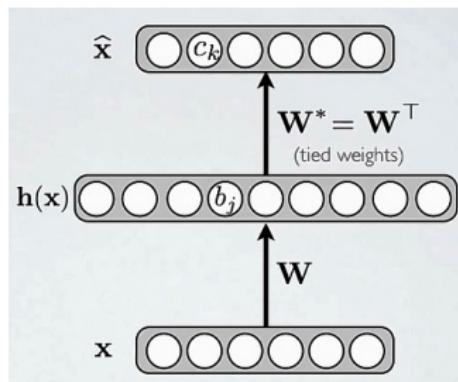
- Hidden layer is larger than the input layer.
 - In this case, hidden layer does not compress the input
 - No guarantee that the hidden units will extract a meaningful structure from the inputs.
 - Each hidden unit tends to simply copy a different input component.



Autoencoder

Denoising autoencoder.

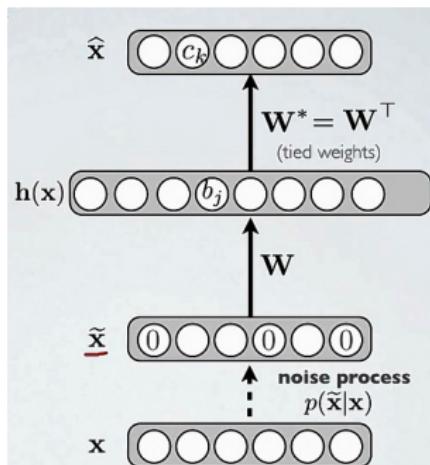
- Prevents the (overcomplete) autoencoder to copy the input.
 - Idea: representation should be robust to the injection of noise.
 - Random assignment of subset of inputs to 0, with probability v .
 - Pepper and salt noise, flip a coin.
 - Gaussian additive noise.



Autoencoder

Denoising autoencoder.

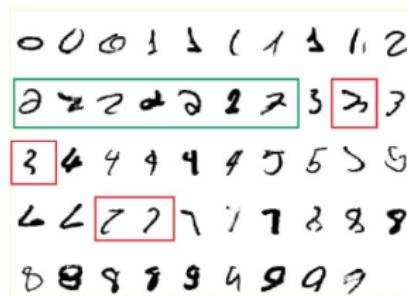
- Prevents the (overcomplete) autoencoder to copy the input.
 - Reconstruction of \hat{x} is computed from the corrupted input \tilde{x}
 - But, loss function compares \hat{x} to the noiseless/original input x .



Autoencoder

Denoising autoencoder.

- Inject noise in the digits. How?



Autoencoder

Denoising autoencoder.

- Inject noise in the digits. How?



- Rotate the digit, shift it, re-scale it.
- Random rotate, random shift, random re-scale.
 - What will happen?

Autoencoder

Denoising autoencoder.

- Inject noise in the digits. How?



- Rotate the digit, shift it, re-scale it.
- Random rotate, random shift, random re-scale.
 - What will happen?
 - The autoencoder takes as input the rotated digit, but reconstruct the original.
 - The autoencoder is robust to rotations, shifts and scale.

Autoencoder

Contractive autoencoder.

- Alternative approach to avoid uninteresting representations.
- Add an explicit term in the loss that penalizes these solutions.
 - Autoencoder reconstruction:
$$-\sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k))$$
This encourages the network to keep good information.
 - Penalty: similarity of hidden and input layers.
 - If the similarity is high, then the penalty is also high.
 - This encourages the network to throw away information.
- Minimize reconstruction error + penalty.
 - Intuition: an interesting solution has low reconstruction error and is not similar to the inputs.

Autoencoder

Image retrieval.

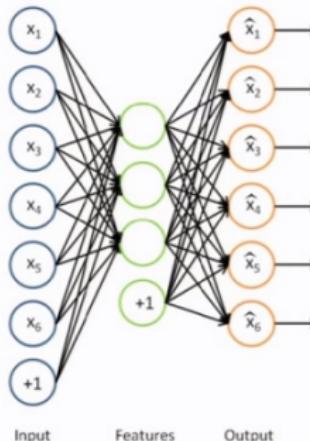
- How autoencoders could be used to image retrieval?
 - Is it possible to do in an unsupervised way?



Autoencoder

Image retrieval.

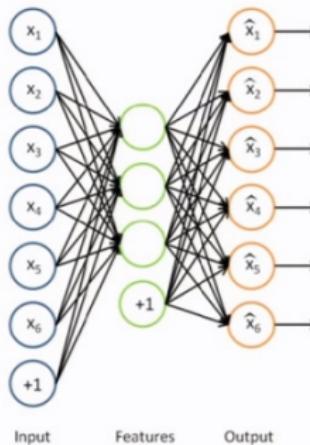
- The input is an image patch.
 - Each input unit is a pixel.
 - Six pixels in the input.
- Train the network to reconstruct the image in the output.



Autoencoder

Image retrieval.

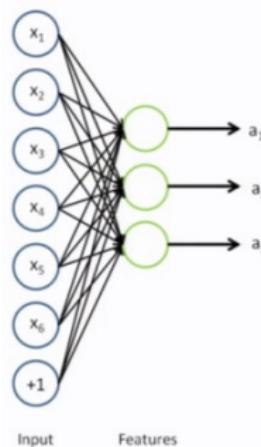
- Our network is able to take images of 6 pixels and encode them using only three (mental) pixels.
 - Also, in order to decode properly, the three values must encode all the necessary information.



Autoencoder

Image retrieval.

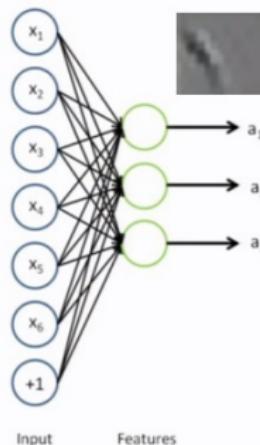
- Once the network is trained (i.e., after many images are shown), we can finally remove the output layer.
 - An arbitrary image is presented to the network, which then outputs a representation for it.



Autoencoder

Image retrieval.

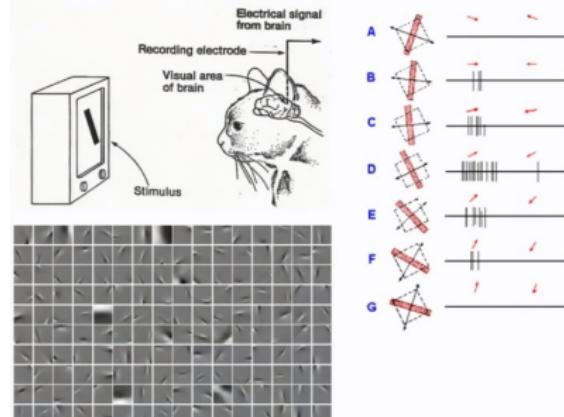
- The weights coming on each neuron form a filter.
 - Which has the same dimension of the image.
 - Each neuron then is able to detect the pattern within its filter (i.e., a feature).



Autoencoder

Image retrieval.

- The network outputs a representation for arbitrary images.
 - Now we can simply hash images, and return those that collide.
 - Notice that certain neurons will output high values, while others will output low values, when presented to specific images (where something similar to its corresponding filter is presented, as with Hubel's famous experiment).



Autoencoder

Distributed representation.

nature

Vol 435 | 23 June 2005 | doi:10.1038/nature03687

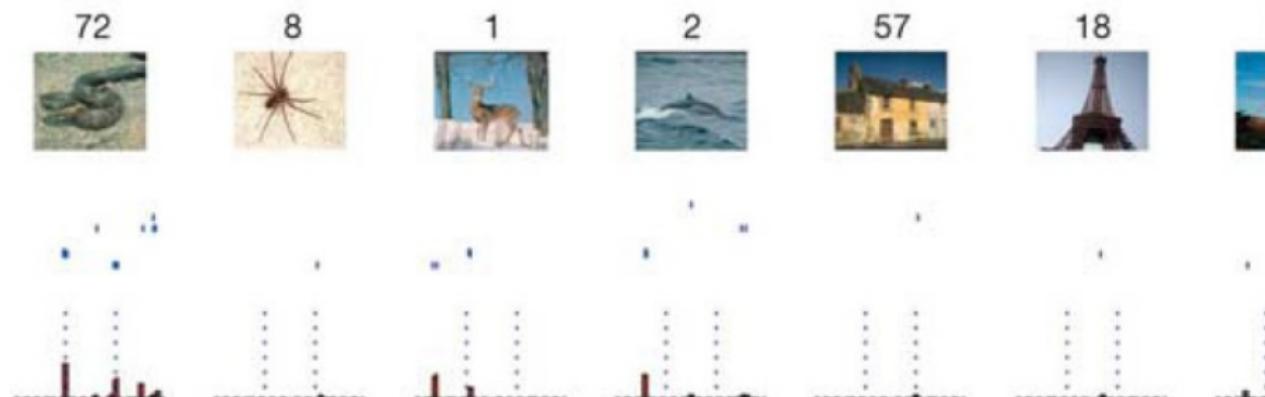
LETTERS

Invariant visual representation by single neurons in the human brain

R. Quian Quiroga^{1,2,†}, L. Reddy¹, G. Kreiman³, C. Koch¹ & I. Fried^{2,4}

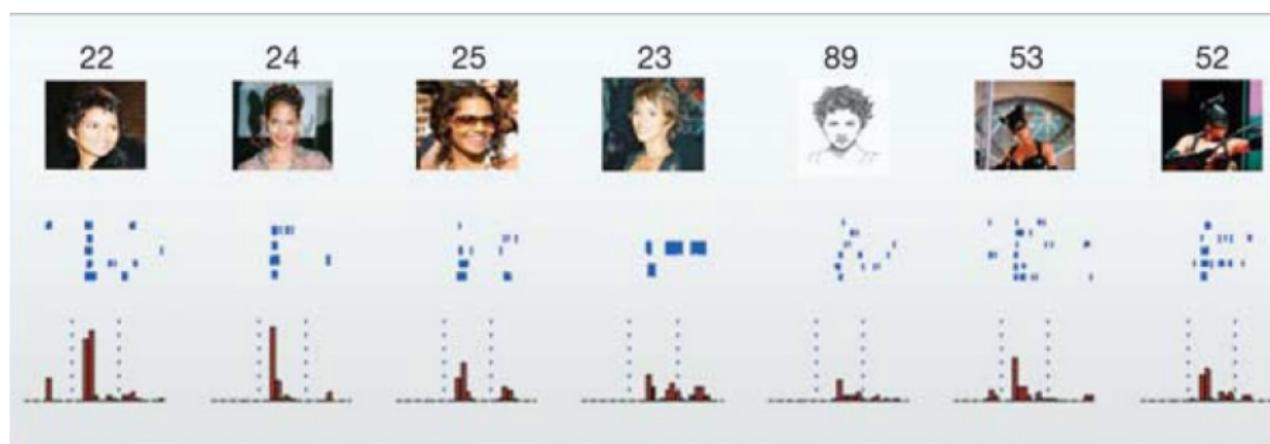
Autoencoder

Distributed representation.



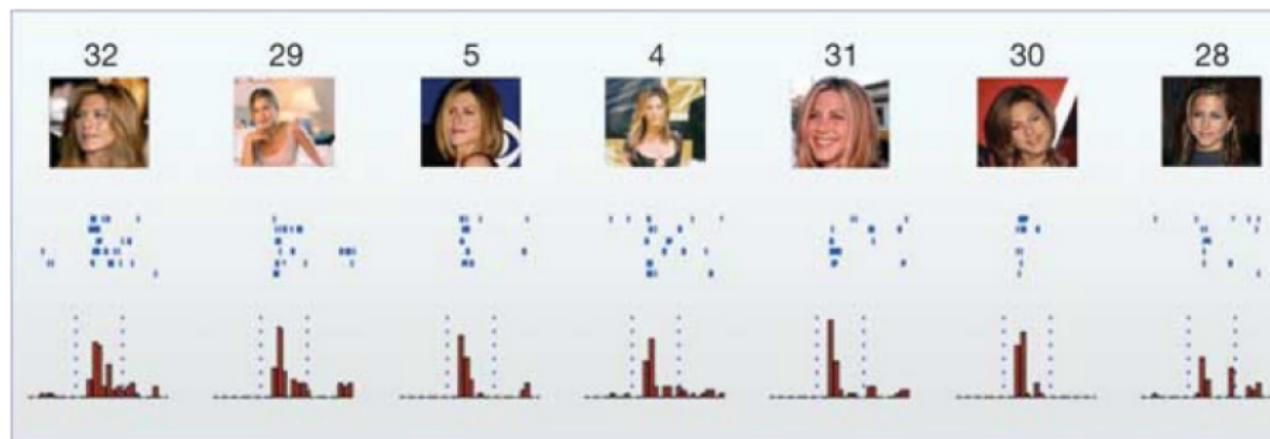
Autoencoder

Distributed representation.



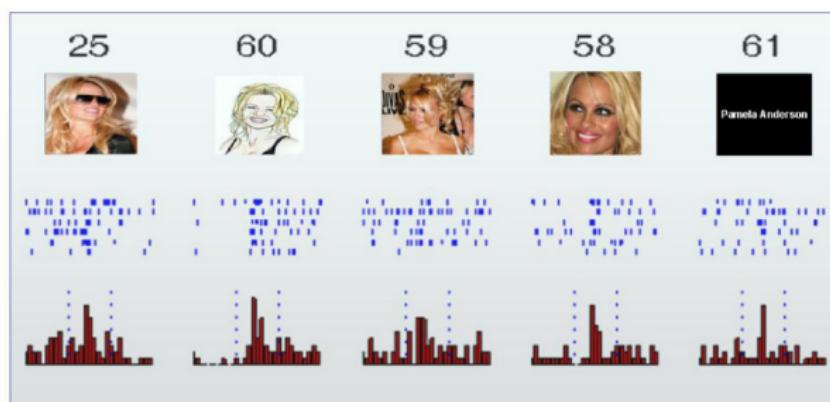
Autoencoder

Distributed representation.



Autoencoder

Distributed representation.



Autoencoder

Training the autoencoder.

- First, we need to ensure low reconstruction error:
 - $\min_{W,b,c} \frac{1}{m} \sum_{i=1}^m (\sigma(W^T \sigma(WX^{(i)} + b) + c) - X^{(i)})^2$
- Regularization?

Autoencoder

Training the autoencoder.

- First, we need to ensure low reconstruction error:

$$\min_{W,b,c} \frac{1}{m} \sum_{i=1}^m (\sigma(W^T \sigma(WX^{(i)} + b) + c) - X^{(i)})^2$$

- Regularization? (sparse autoencoder)

- Recall Hubel's experiment:

- Only part of the neurons fire when a specific pattern is presented to them.

- So, given a specific pattern we want to ensure that only few neurons will fire.

Autoencoder

Training the autoencoder.

- First, we need to ensure low reconstruction error:

- $$\min_{W,b,c} \frac{1}{m} \sum_{i=1}^m (\sigma(W^T \sigma(WX^{(i)} + b) + c) - X^{(i)})^2$$

- Regularization? (sparse autoencoder)

- Recall Hubel's experiment:

- Only part of the neurons fire when a specific pattern is presented to them.**

- So, given a specific pattern we want to ensure that only few neurons will fire.

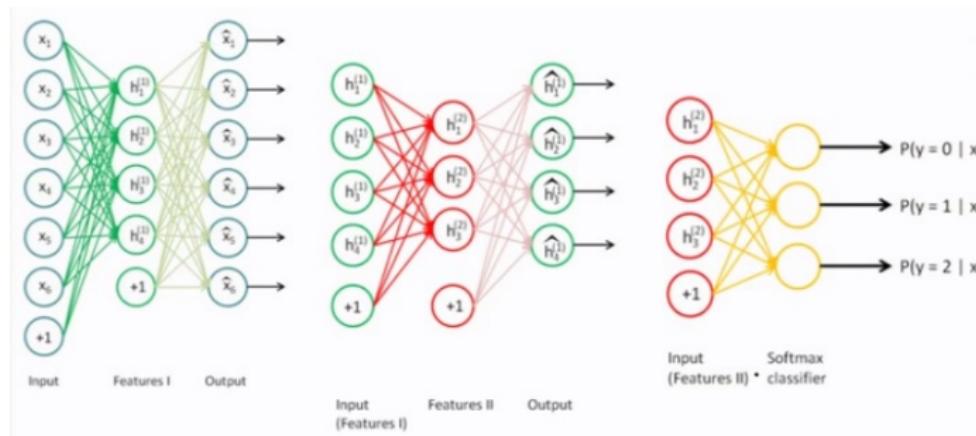
- $$\lambda \times \left(\sum_{i=1}^m \sum_{j=1}^k |W_j \cdot X^{(i)}| + \sum_{j=1}^k \sum_{i=1}^m |W_j \cdot X^{(i)}| \right)$$

- For a particular image $X^{(i)}$, only part of the neurons will fire.
- A particular neuron W_j will only fire for part of the images.

Deep Learning

Greedy layer-wise training.

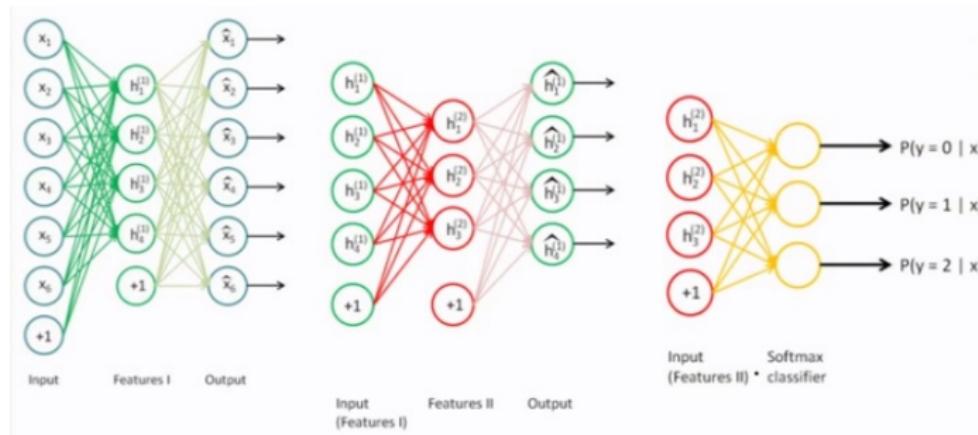
- Training a deep network is hard.
 - The gradient quickly vanishes during backpropagation.



Deep Learning

Greedy layer-wise training.

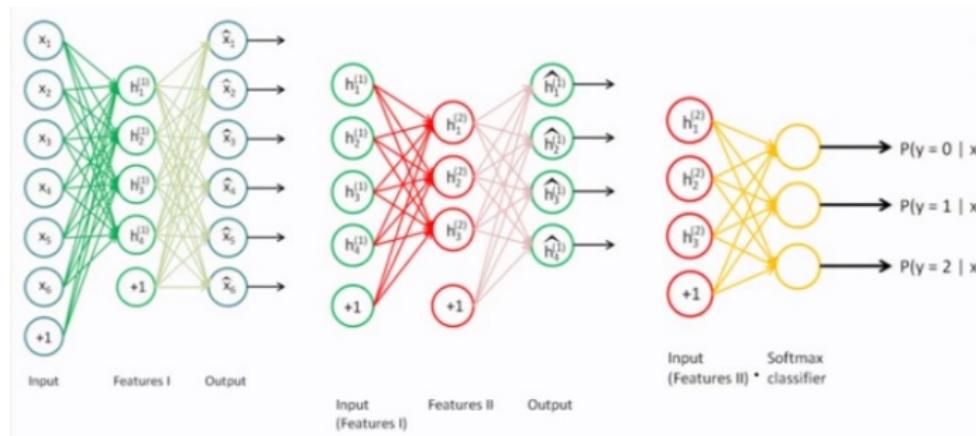
- Pick the data (images, text etc.)
 - And train an autoencoder from it.



Deep Learning

Greedy layer-wise training.

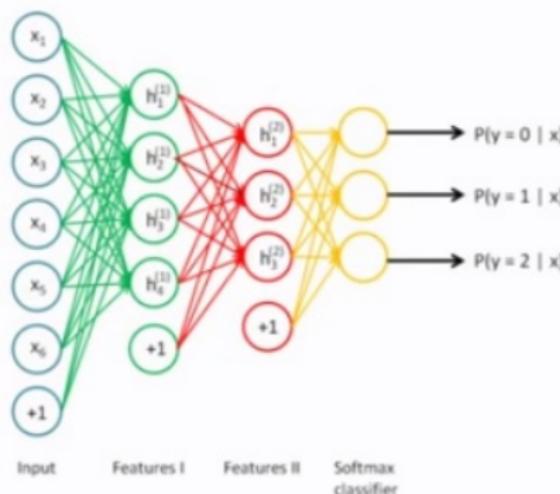
- Then, remove the output layer.
 - And use the hidden layer as input for the next autoencoder.



Deep Learning

Greedy layer-wise training.

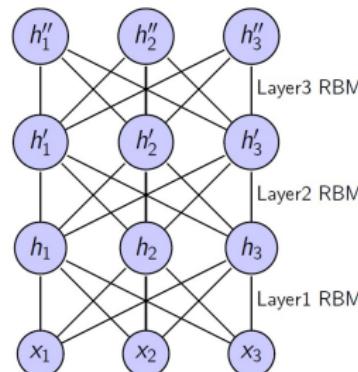
- This process is unsupervised, and is called pre-training.
 - Training the output layer using supervision (1-layer network).
 - Perform backpropagation to refine the weights.



Deep Learning

Greedy layer-wise training.

- Deep belief networks.
 - Pre-training with RBMs ($f : x \rightarrow y$, then $f : x \rightarrow h \rightarrow y$).
- Generative model:
 - $p(X) = \sum_{h,h',h''} p(x|h)p(h|h')p(h'|h'')$
 - Top two layers are an RBM, and lower layers are sigmoids.



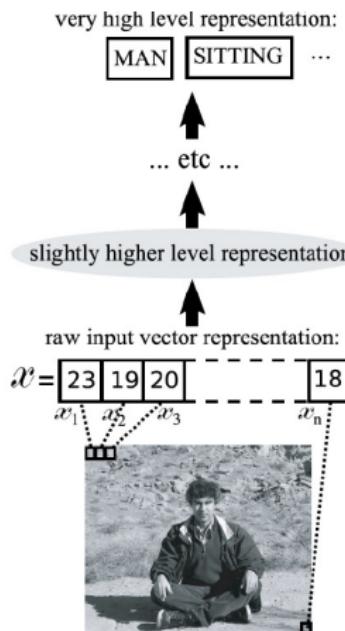
Deep Learning

Why the interest in learning deep neural networks?

- The promise:
 - Deep architectures allows the network to learn high-level representations of the raw data.
 - The raw representation of an image may be its pixel intensities.
 - But learning requires high-level abstractions, modeled by highly non-linear functions.
- Need to find a way to go from raw data to very high level representations.
 - Deep architecture is one way to achieve this: each intermediate layer is a successively higher level of abstraction.

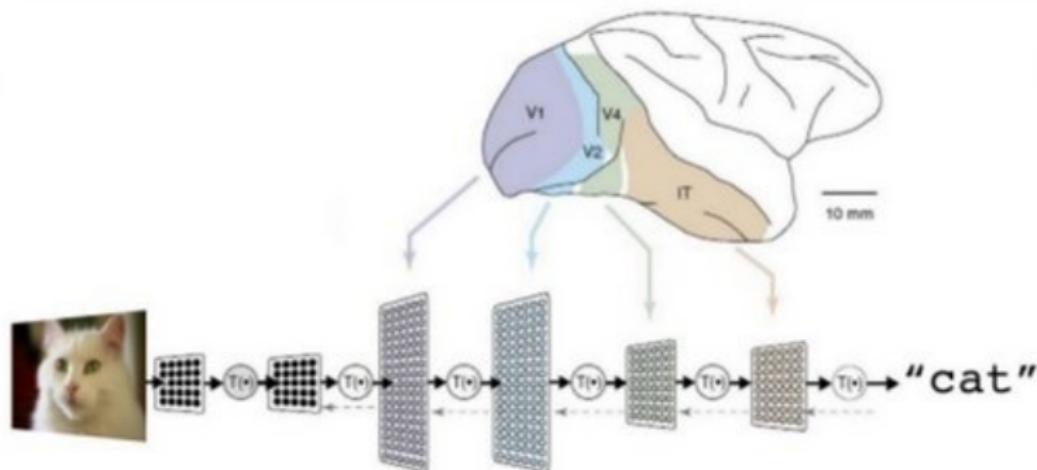
Deep Learning

Why the interest in learning deep neural networks?



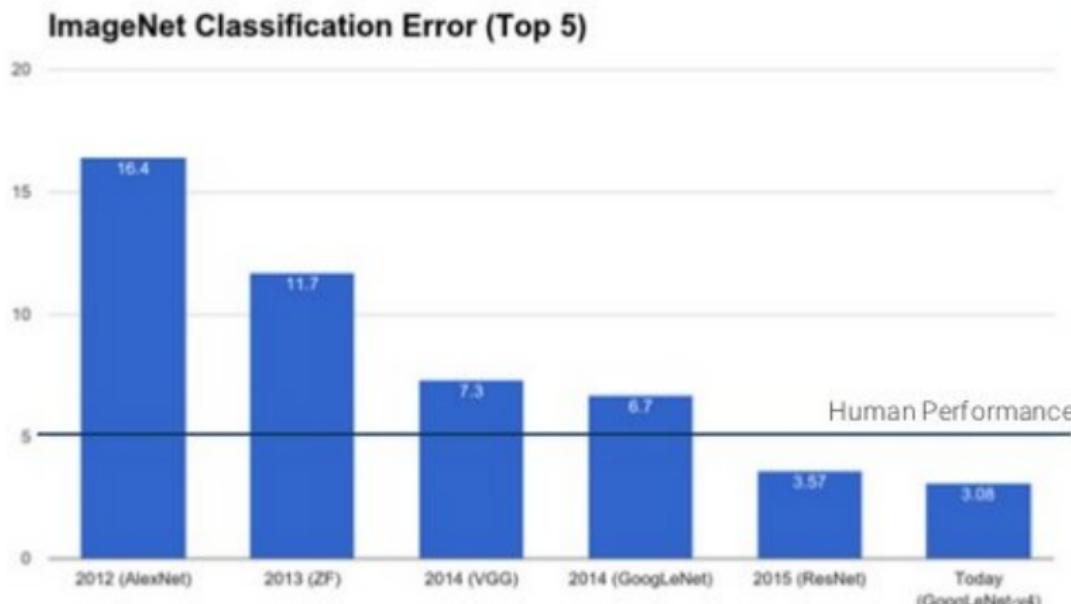
Deep Learning

Why the interest in learning deep neural networks?

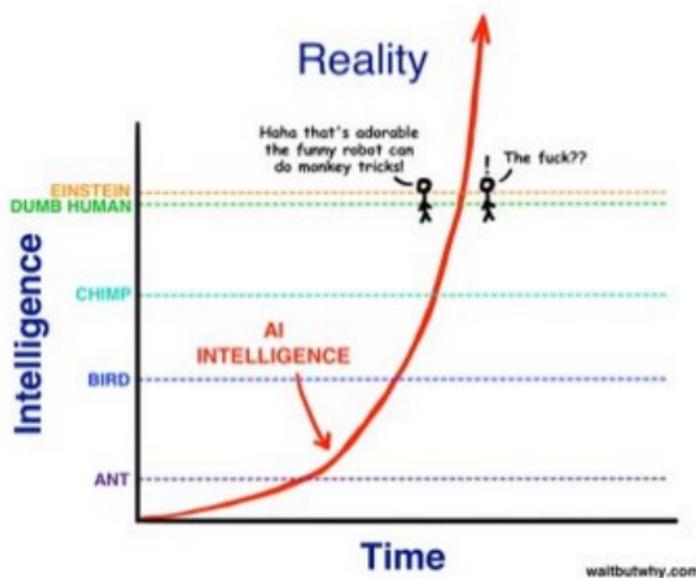


Deep Learning

Why the interest in learning deep neural networks?



Deep Learning



Deep Learning

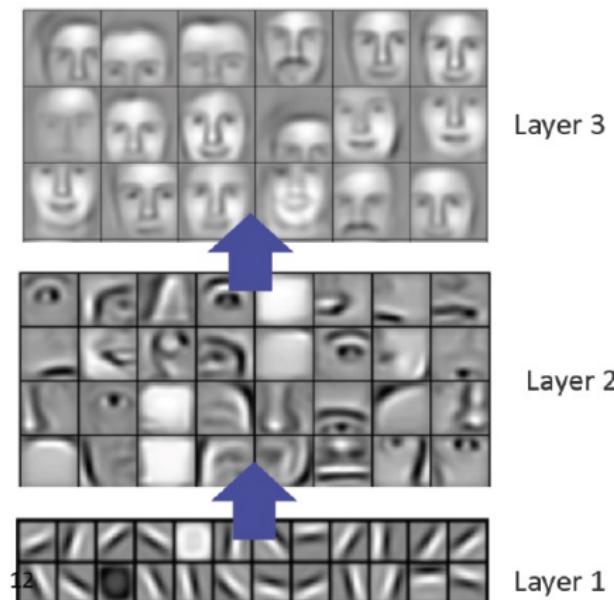
Why the interest in learning deep neural networks?

- Google experiment:
 - 3-layer network, 1 billion parameters
 - 10 million 200×200 images from 10 million YouTube videos
 - 1,000 machines (16,000 cores) \times 1 week
 - Lots of tricks for data/model parallelization on GPUs.



Deep Learning

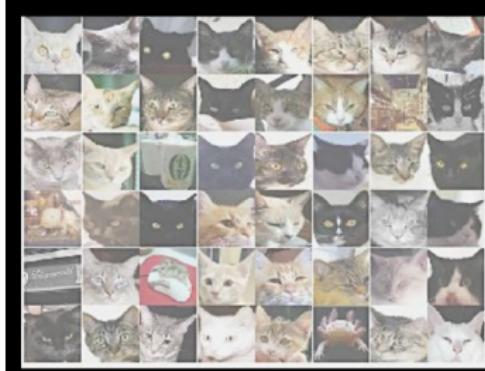
Why the interest in learning deep neural networks?



Deep Learning

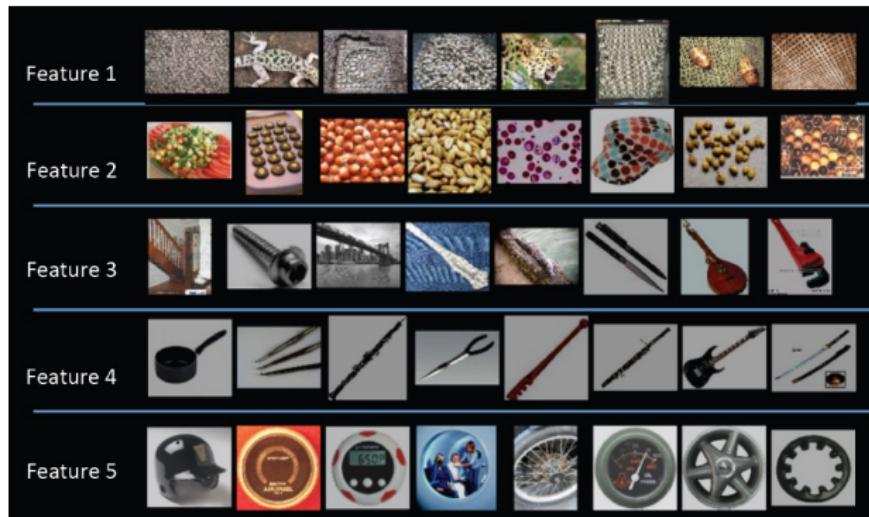
Why the interest in learning deep neural networks?

- Google experiment:
 - 3-layer network, 1 billion parameters
 - 10 million 200×200 images from 10 million YouTube videos
 - 1,000 machines (16,000 cores) \times 1 week
 - Lots of tricks for data/model parallelization on GPUs.



Deep Learning

Why the interest in learning deep neural networks?



Deep Learning

Why the interest in learning deep neural networks?



Deep Learning

Why the interest in learning deep neural networks?



Deep Learning

Why the interest in learning deep neural networks?

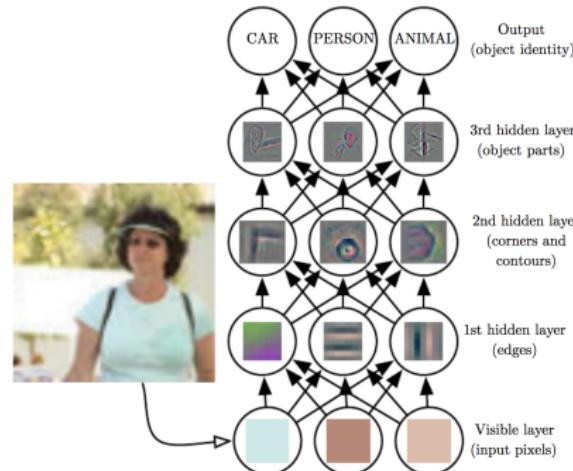
- Train the last layer in a supervised way.



Deep Learning

Why the interest in learning deep neural networks?

- Train the last layer in a supervised way.



Deep Learning

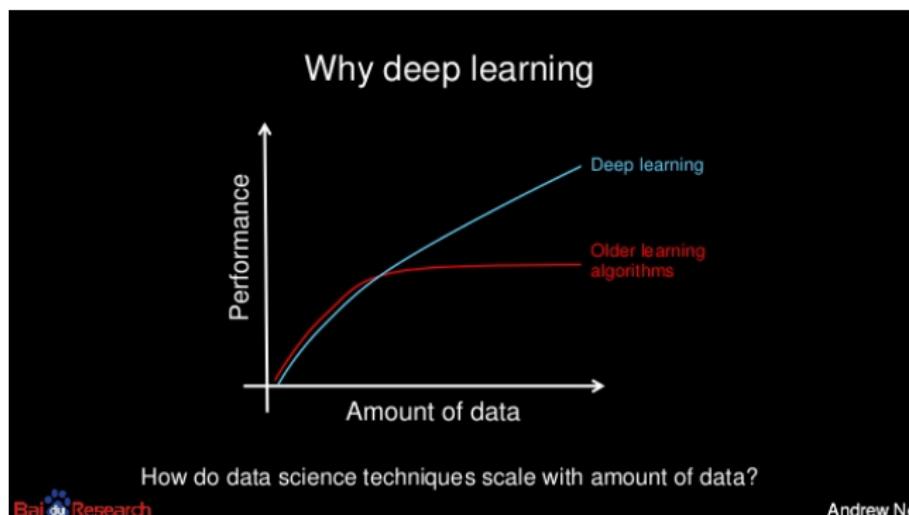
Why the interest in learning deep neural networks?

Table: ImageNet Performance

Method	Accuracy
Random	0.005%
Previous State-of-the-art	9.300%
Google without pre-training	13.600%
Google with pre-training	15.800%

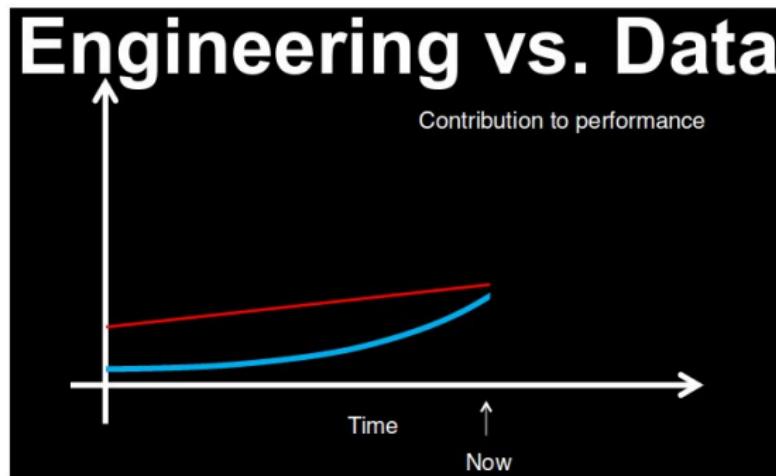
Deep Learning

Pre-training + fine-tuning.



Deep Learning

Feature Learning.



Deep Learning

Why the interest in learning deep neural networks?

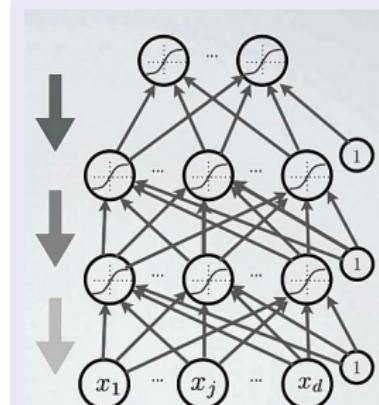
- Best results reported using a single hidden layer.
 - But, intuitively, better results would come with more layers.

Deep Learning

Why the interest in learning deep neural networks?

- Best results reported using a single hidden layer.
 - But, intuitively, better results would come with more layers.
- Backpropagation is prone to gradient vanishing:
 - As errors propagate from layer to layer, they shrink exponentially with the number of layers.

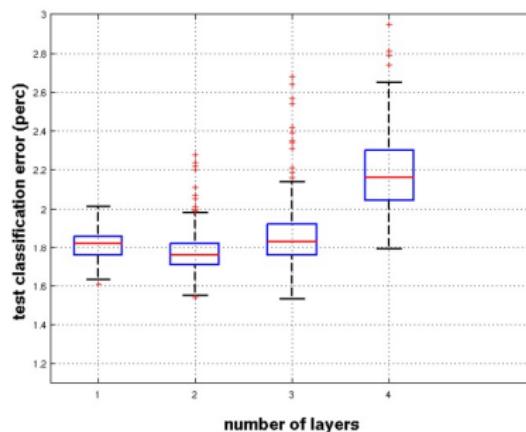
- Multiplying the activation by the partial derivative tends to 0.
 - When the activation gets closer to its minimum or maximum (i.e., saturation point).



Deep Learning

Poor performance of backpropagation on deep neural networks.

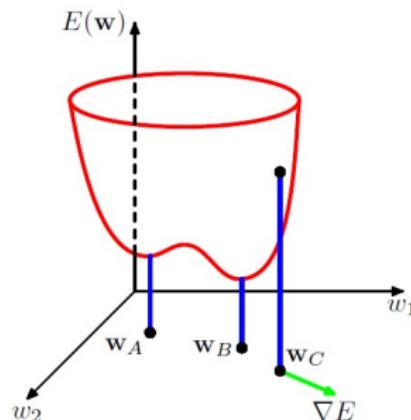
- Depends severely on how the weights are initialized.
- Although using $L + 1$ layers makes the network more expressive, in practice it is worse than using only L layers.



Deep Learning

Poor performance of backpropagation on deep neural networks.

- As more layers are employed, the loss function becomes less and less convex.
- This means we have more and more local minima. Achieving a local minimum depends mainly on the initial weights.
 - For deep networks, backpropagation is apparently getting a local minimum that does not generalize well.



Deep Learning

Why the interest in learning deep neural networks?

- Shallow architectures (neural nets with one layer, SVMs, boosting) are universal approximators.
 - But may require exponentially more neurons than corresponding deep architectures.
 - Thus, deep neural networks are statistically more efficient/compact to learn than fat shallow architectures.

Journal of Machine Learning Research 1 (2009) 1-40

Submitted 12/07; Revised 9/08; Published 1/09

Exploring Strategies for Training Deep Neural Networks

Hugo Larochelle

Yoshua Bengio

Jérôme Louradour

Pascal Lamblin

Département d'informatique et de recherche opérationnelle

Université de Montréal

2920, chemin de la Tour

Montréal, Québec, Canada, H3T 1J8

LAROCHEH@IRO.UMONTREAL.CA

BENGOY@IRO.UMONTREAL.CA

LOURADOJ@IRO.UMONTREAL.CA

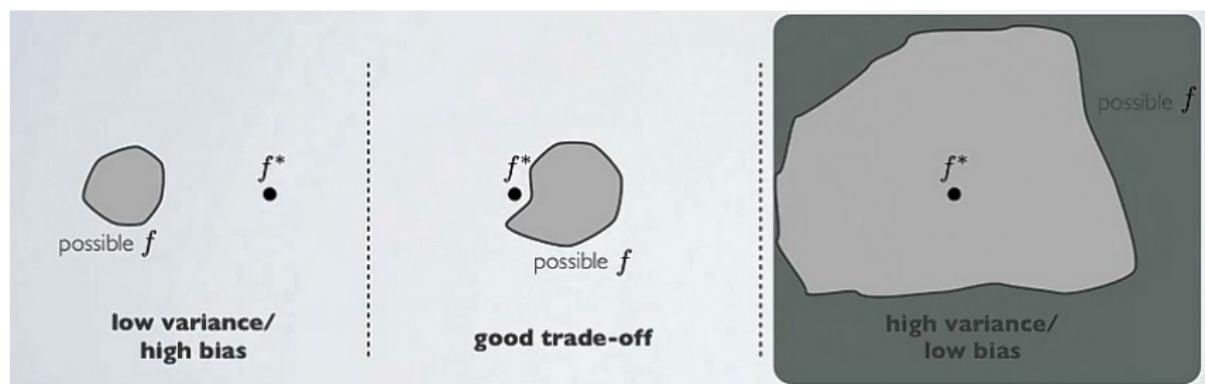
LAMBLINP@IRO.UMONTREAL.CA

Deep Learning

Why the interest in learning deep neural networks?

- A Boolean circuit is a sort of feed-forward network where hidden units are logic gates (i.e., AND, OR or NOT)
 - Any Boolean function can be represented by a single-layer Boolean circuit.
 - However, it may require an exponential number of units.
 - It can be shown that there are Boolean functions which:
 - Require an exponential number of units in the 1-layer case.
 - Require a polynomial number of hidden units if we can adapt the number of layers.
- Multilayer networks usually have a large number of weights/parameters that have to be learned.
 - Lots of possible network configurations that we can model.
 - Thus, the space of possible functions is humongous.
 - This puts the class of models induced by multilayer neural networks in the high-variance/low-bias situation.

Deep Learning



Deep Learning

Solution:

- Focus on modeling the input $P(X)$ better with each successive layer.
 - Pre-training focuses on optimizing likelihood on the examples, not the target label.
 - This enables us to initialize the deep network with good weights!
 - Get away from local minima.
 - Extra advantage: can exploit possibly large amounts of unlabeled data.
- Worry about optimizing the task $P(Y|X)$ latter.

Deep Learning

Learning models with multiple layers of representation.

- Unsupervised pre-training:
 - Deep Belief networks (stacked RBMs)
 - **Stacked autoencoders**
 - Stacked denoising autoencoders
 - Multilayer neural network
- Regularization:
 - Stochastic dropout.
- Robust gradients:
 - Rectified linear units.
- Each layer corresponds to a distributed representation.
 - Biology: a specific stimulus is coded by its unique pattern of activity over a group of neurons.
 - Each unit/neuron is seen a separate feature of the input, and are not mutually exclusive.

Deep Learning



Success stories.

- Microsoft Research, Baidu: speech recognition.
- Google: computer vision.
- Facebook: NLP.

Deep Learning

Unsupervised pre-training.

- Initialize hidden layers using autoencoders or RBMs.
- Force the network to represent the latent structure of the inputs.
 - Prevents local minima: in non-convex optimization solution depends mostly on where we start.



character image



random image



Deep Learning

Unsupervised pre-training.

- ① Train one layer at a time, from first to last (autoencoder or RBM).

Deep Learning

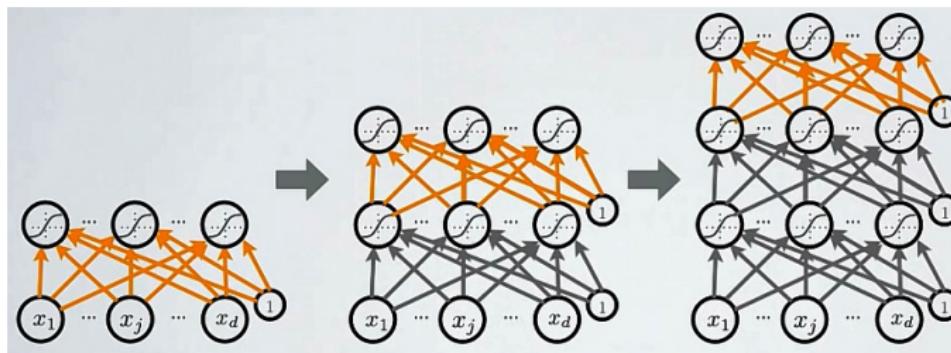
Unsupervised pre-training.

- ① Train one layer at a time, from first to last (autoencoder or RBM).
- ② Keep/freeze the weights within each layer.

Deep Learning

Unsupervised pre-training.

- ① Train one layer at a time, from first to last (autoencoder or RBM).
- ② Keep/freeze the weights within each layer.
- ③ Previous layers viewed as feature extraction (like kernels).



Deep Learning

Fine tuning.

- Once all layers are pre-trained:
 - Add output layer.
 - Train the entire network using supervised learning (with the appropriate loss function).
 - Supervised learning using backpropagation.
- All weights are tuned for the supervised task.
 - Representation is adjusted to become more discriminative.
 - Convergence is much faster.
- The process remembers **semi-supervised learning**.
 - It is a mix of semi-supervised learning, and online kernel adaptation.

Deep Learning

Impact of unsupervised pre-training.

Journal of Machine Learning Research 11 (2010) 625-660

Submitted 8/09; Published 2/10

Why Does Unsupervised Pre-training Help Deep Learning?

Dumitru Erhan*

Yoshua Bengio

Aaron Courville

Pierre-Antoine Manzagol

Pascal Vincent

Département d'informatique et de recherche opérationnelle

Université de Montréal

2920, chemin de la Tour

Montréal, Québec, H3T 1J8, Canada

DUMITRU.ERHAN@UMONTREAL.CA

YOSHUA.BENGIO@UMONTREAL.CA

AARON.COURVILLE@UMONTREAL.CA

PIERRE-ANTOINE.MANZAGOL@UMONTREAL.CA

PASCAL.VINCENT@UMONTREAL.CA

Samy Bengio

Google Research

1600 Amphitheatre Parkway

Mountain View, CA, 94043, USA

BENGIO@GOOGLE.COM

Editor: Léon Bottou

Deep Learning

Impact of pre-training.

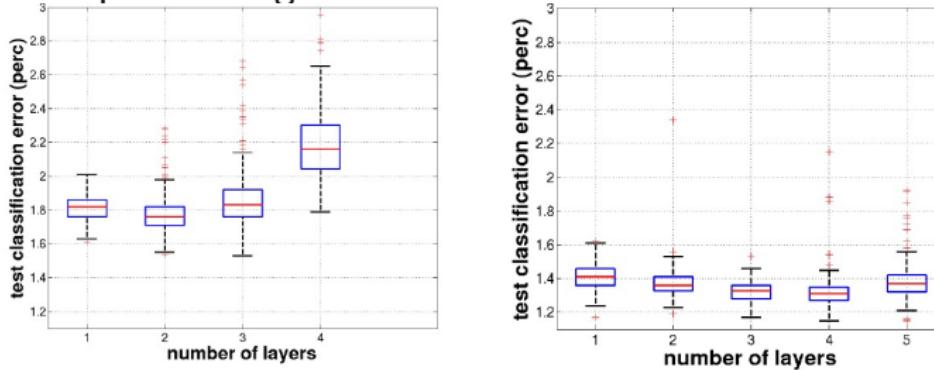


Figure 1: Effect of depth on performance for a model trained (**left**) without unsupervised pre-training and (**right**) with unsupervised pre-training, for 1 to 5 hidden layers (networks with 5 layers failed to converge to a solution, without the use of unsupervised pre-training). Experiments on MNIST. Box plots show the distribution of errors associated with 400 different initialization seeds (top and bottom quartiles in box, plus outliers beyond top and bottom quartiles). Other hyperparameters are optimized away (on the validation set). *Increasing depth seems to increase the probability of finding poor apparent local minima.*

Deep Learning

Impact of pre-training.

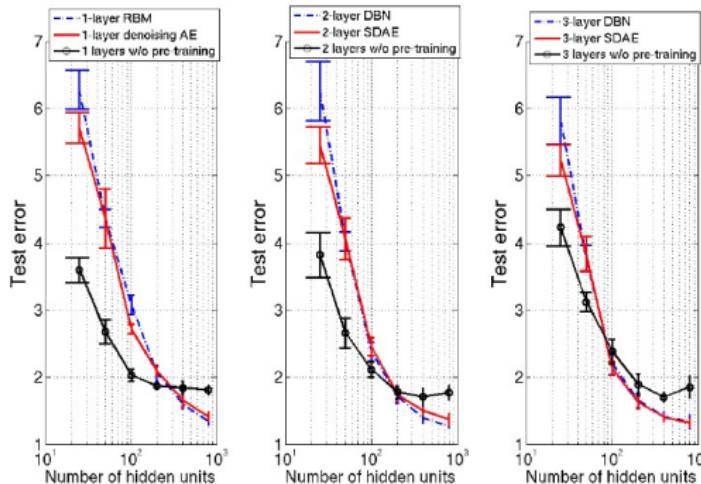


Figure 9: Effect of layer size on the changes brought by unsupervised pre-training, for networks with 1, 2 or 3 hidden layers. Experiments on MNIST. Error bars have a height of two standard deviations (over initialization seed). Pre-training hurts for smaller layer sizes and shallower networks, but it helps for all depths for larger networks.

Deep Learning

Impact of pre-training.

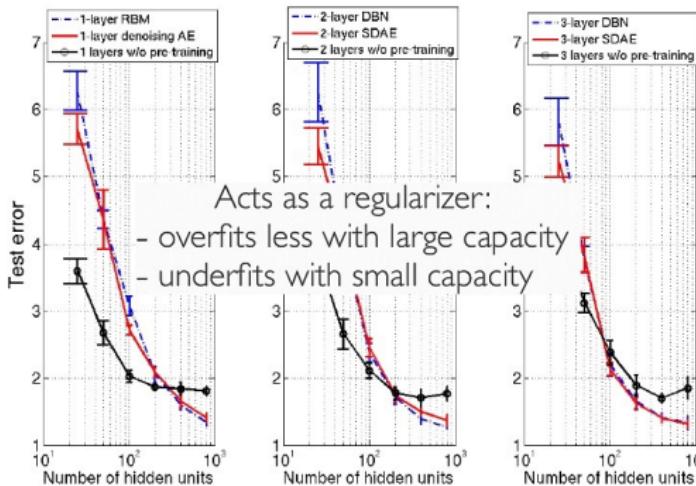
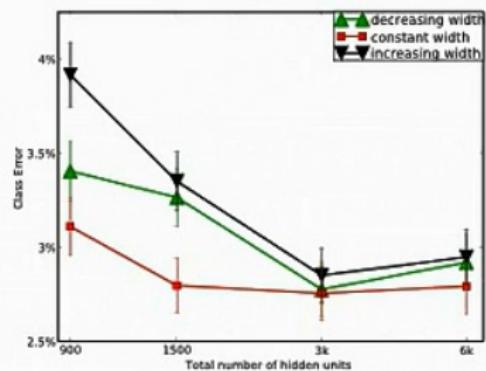


Figure 9: Effect of layer size on the changes brought by unsupervised pre-training, for networks with 1, 2 or 3 hidden layers. Experiments on MNIST. Error bars have a height of two standard deviations (over initialization seed). Pre-training hurts for smaller layer sizes and shallower networks, but it helps for all depths for larger networks.

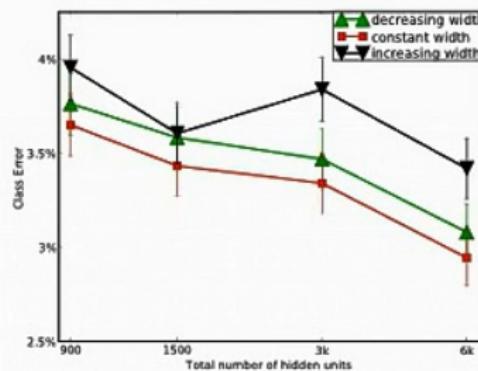
Deep Learning

Impact of pre-training.

RBM



Autoencoder



Deep Learning

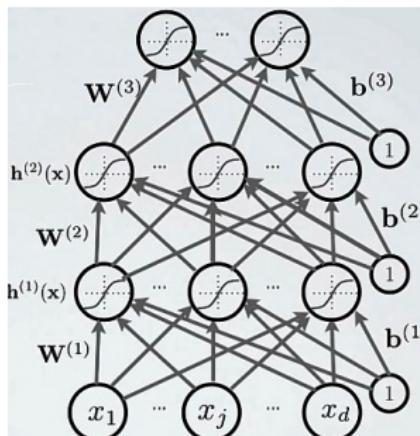
Impact of pre-training.

	Stacked Autoencoders	Stacked RBMS	Stacked Denoising Autoencoders
Dataset	SVM _{rbf}	SAA-3	DBN-3
<i>basic</i>	3.03±0.15	3.46±0.16	3.11±0.15
<i>rot</i>	11.11±0.28	10.30±0.27	10.30±0.27
<i>bg-rand</i>	14.58±0.31	11.28±0.28	6.73±0.22
<i>bg-img</i>	22.61±0.37	23.00±0.37	16.31±0.32
<i>rot-bg-img</i>	55.18±0.44	51.93±0.44	47.39±0.44
<i>rect</i>	2.15±0.13	2.41±0.13	2.60±0.14
<i>rect-img</i>	24.04±0.37	24.05±0.37	22.50±0.37
<i>convex</i>	19.13±0.34	18.41±0.34	18.63±0.34
			19.06±0.34 (10%)

Deep Learning

Dropout during fine-tuning.

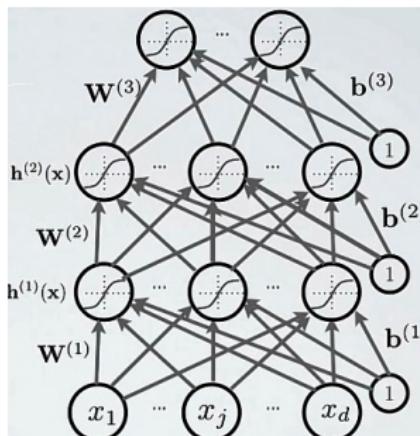
- Combining different models can be very useful (i.e., Kaggle)
 - Voting, bagging, boosting etc.



Deep Learning

Dropout during fine-tuning.

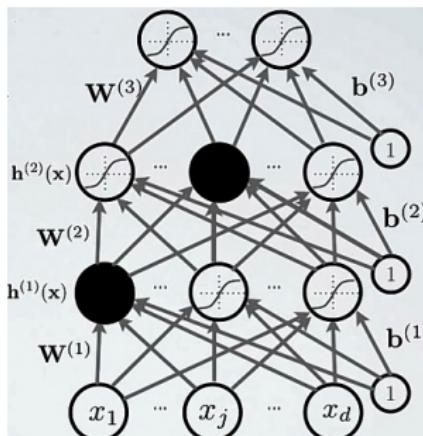
- Similar to Random Forests.
 - Training many different networks, however, is time consuming.



Deep Learning

Dropout during fine-tuning.

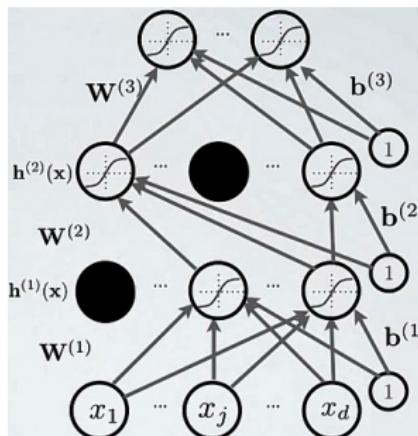
- Solution:
 - Set the output of each hidden neuron to 0, with probability v .



Deep Learning

Dropout during fine-tuning.

- Solution:
 - Set the output of each hidden neuron to 0, with probability v .



Deep Learning

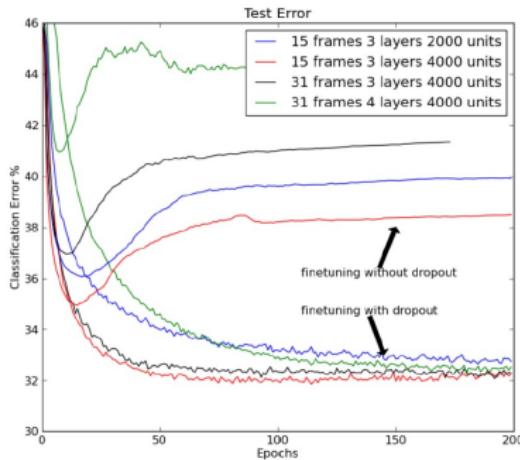
Dropout during fine-tuning.

- The neurons which are “dropped out” do not contribute to the forward pass and do not participate in backpropagation.
 - Every time an input is presented, the network builds a different architecture, but all these architectures share weights.
 - This is like sampling from 2^h different architectures.
- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of other neurons.
 - It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
 - Without dropout, our network exhibits substantial overfitting.
 - Dropout roughly doubles the number of iterations required to converge.

Deep Learning

Dropout during fine-tuning.

- Phone recognition.
 - TIMIT is a corpus of phonemically and lexically transcribed speech of English speakers of different genres and dialects.



Deep Learning

Large scale distributed training.

- Large scale data enables more complex models.
 - We can increase the capacity of the models as more data is available for training

Deep Learning

Large scale distributed training.

- Large scale data enables more complex models.
 - We can increase the capacity of the models as more data is available for training
- Learning from large scale data demands high performance computing.



Deep Learning

How To Build a GPU System for Deep Learning

- GPU:
 - GTX 580 (no money); GTX 980 (best performance); GTX Titan (if you need memory)
- CPU:
 - Two cores per GPU; > 2GHz; cache does not matter;
- RAM:
 - Use asynchronous mini-batch allocation; clock rate and timings do not matter; buy at least as much CPU RAM as you have GPU RAM;

Deep Learning

How To Build a GPU System for Deep Learning

- Cooling:
 - Set coolbits flag in your config if you run a single GPU; otherwise flashing BIOS for increased fan speeds is easiest and cheapest; use water cooling for multiple GPUs and/or when you need to keep down the noise
- Motherboard:
 - Get PCIe 3.0 and as many slots as you need for your (future) GPUs (one GPU takes two slots; max 4 GPUs per system)
- Monitors:
 - If you want to upgrade your system to be more productive, it might make more sense to buy an additional monitor rather than upgrading your GPU

Deep Learning

Advantages of deep learning:

- Do not need to hand engineer features for non-linear learning problems
 - Save time and scalable to the future, since hand engineering is seen by some as a short-term band-aid.
 - The learnt features are sometimes better than the best hand-engineered features, and can be so complex (computer vision - e.g. face-like features) that it would take way too much human time to engineer.

Deep Learning

Advantages of deep learning:

- Do not need to hand engineer features for non-linear learning problems
 - Save time and scalable to the future, since hand engineering is seen by some as a short-term band-aid.
 - The learnt features are sometimes better than the best hand-engineered features, and can be so complex (computer vision - e.g. face-like features) that it would take way too much human time to engineer.
- Deep Belief Nets can sample from the learned distributions
 - Generate synthetic data corresponding to the learned classes/clusters.

Deep Learning

Advantages of deep learning:

- Can use unlabeled data to pre-train the network.
- Suppose we have 1,000,000 unlabeled images and 1,000 labeled images.
 - We can now drastically improve a supervised learning algorithm by pre-training on the 1,000,000 unlabeled images with deep learning.
 - In addition, in some domains we have so much unlabeled data but labeled data is hard to find. An algorithm that can use this unlabeled data to improve classification is valuable.

Deep Learning

Advantages of deep learning:

- Can use unlabeled data to pre-train the network.
- Suppose we have 1,000,000 unlabeled images and 1,000 labeled images.
 - We can now drastically improve a supervised learning algorithm by pre-training on the 1,000,000 unlabeled images with deep learning.
 - In addition, in some domains we have so much unlabeled data but labeled data is hard to find. An algorithm that can use this unlabeled data to improve classification is valuable.
- Empirically, smashed many benchmarks
 - Were only seeing incremental improvements until the introduction of deep learning methods.

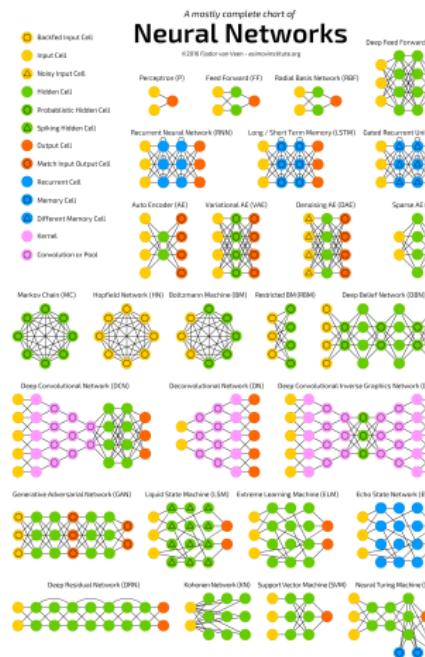
Deep Learning

Advantages of deep learning:

- Can use unlabeled data to pre-train the network.
- Suppose we have 1,000,000 unlabeled images and 1,000 labeled images.
 - We can now drastically improve a supervised learning algorithm by pre-training on the 1,000,000 unlabeled images with deep learning.
 - In addition, in some domains we have so much unlabeled data but labeled data is hard to find. An algorithm that can use this unlabeled data to improve classification is valuable.
- Empirically, smashed many benchmarks
 - Were only seeing incremental improvements until the introduction of deep learning methods.
- Same algorithm works in multiple areas with raw (perhaps with minor pre-processing) inputs.

Deep Learning

Deep Learning Zoo.



Deep Learning

We do not start thinking from scratch every time.

- Our thoughts have persistence.
- Traditional neural networks cannot do this.
 - It seems like a major shortcoming, since most of learning is based on this assumption.

Deep Learning

We do not start thinking from scratch every time.

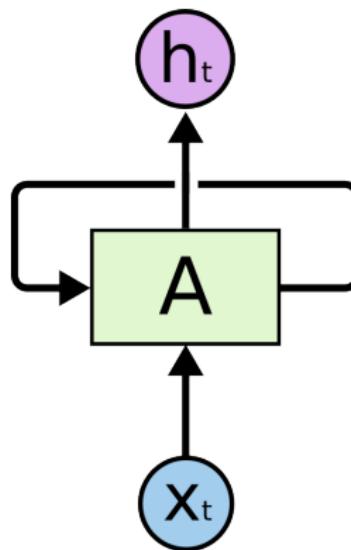
- Our thoughts have persistence.
- Traditional neural networks cannot do this.
 - It seems like a major shortcoming, since most of learning is based on this assumption.
- Try to think the alphabet from A to Z, and from Z to A.

Recurrent neural networks address this issue.

Deep Learning

Recurrent Networks.

- Networks with loops in them, allowing information to persist.

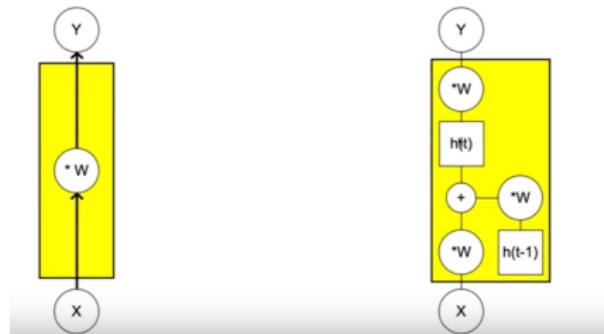


Deep Learning

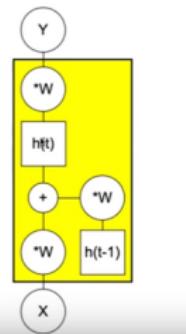
Recurrent Networks.

- The network has a state.
- The state is given in terms of the input and the previous state.

Feedforward neural net



Recurrent neural net

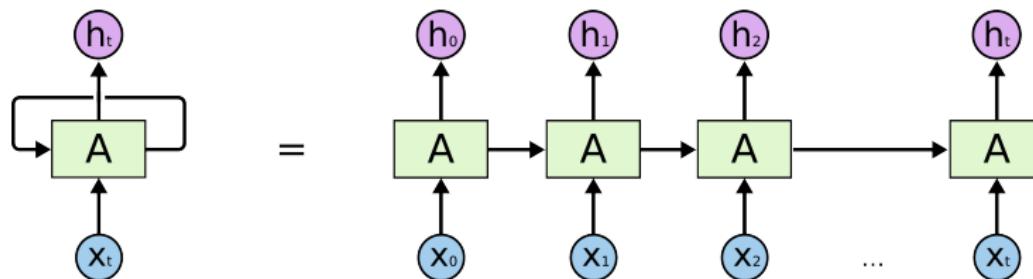


$$\mathbf{h}_t = \phi(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}),$$

Deep Learning

Recurrent Networks.

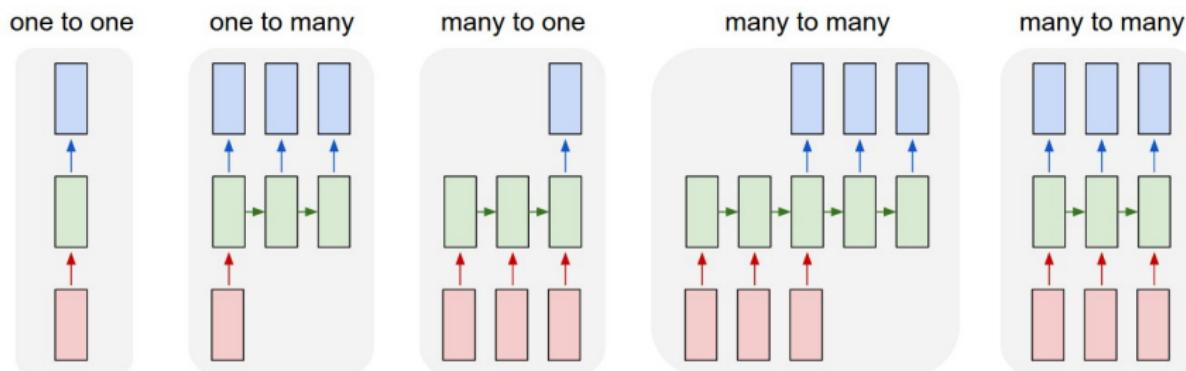
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.



Deep Learning

Recurrent Networks.

- Natural architecture of neural network to use for sequential data.



Deep Learning

Recurrent Networks.

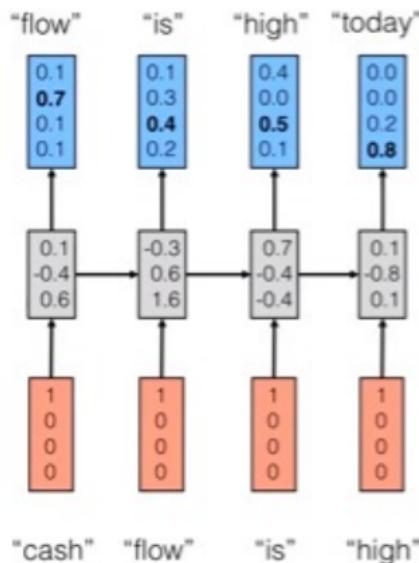
- Language models.



Deep Learning

Recurrent Networks.

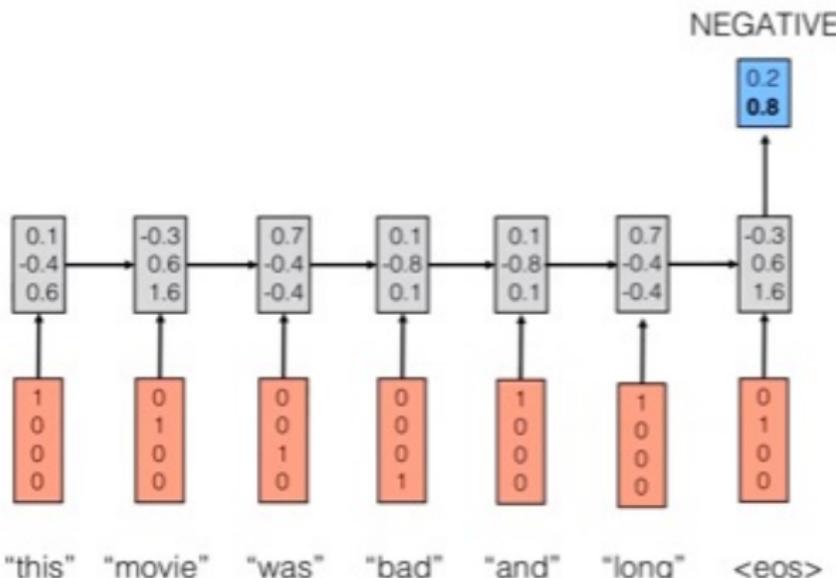
- Language models.



Deep Learning

Recurrent Networks.

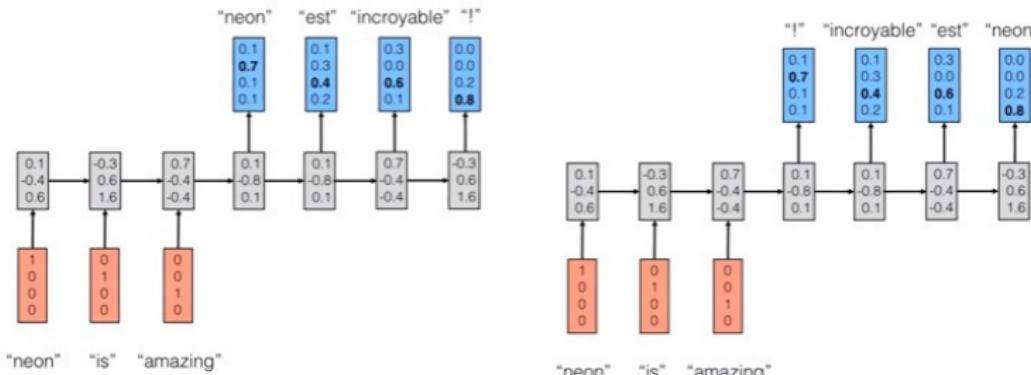
- Sentiment Analysis.



Deep Learning

Recurrent Networks.

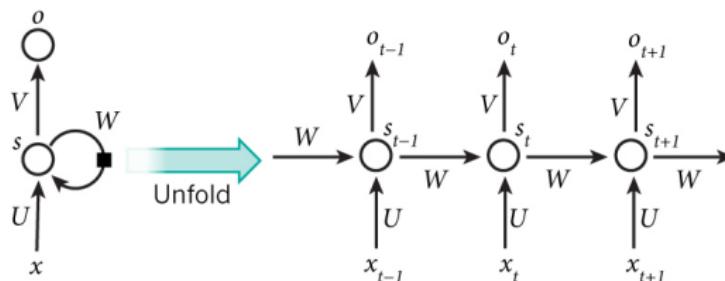
- Translation (Q&A, chatbots etc).



Deep Learning

How to train recurrent networks?

- Cross-Entropy loss.
 - $E_t(o_t, \hat{y}_t) = -o_t \times \log \hat{y}_t \rightarrow E(o, \hat{y}) = \sum_t E_t(o_t, \hat{y}_t)$
 - $E(o, \hat{y}) = - \sum_t o_t \times \log \hat{y}_t$
- o_t is correct output at time step t , and \hat{y}_t is the prediction.
 - We treat the full sequence as one training example, so the total error is just the sum of the errors at each time step.



Deep Learning

How to train recurrent networks?

- Goal:
 - To calculate the gradients of the error with respect to parameters U, V and W.
- Just like we sum up the errors, we also sum up the gradients at each time step for one training example:
 - $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$
 - To calculate these gradients we use the backpropagation algorithm when applied backwards starting from the error.

Deep Learning

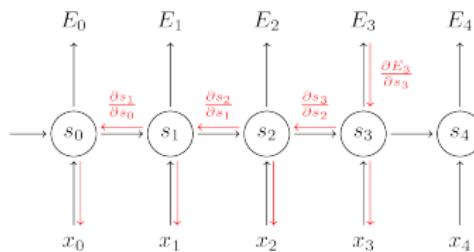
How to train recurrent networks?

- Goal:
 - To calculate the gradients of the error with respect to parameters U , V and W .
- Just like we sum up the errors, we also sum up the gradients at each time step for one training example:
 - $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$
 - To calculate these gradients we use the backpropagation algorithm when applied backwards starting from the error.
 - Notice that gradients with regard to V are easy to compute, since they only depend on the values at the current time step. The history is different for W .

Deep Learning

Back-propagation through time.

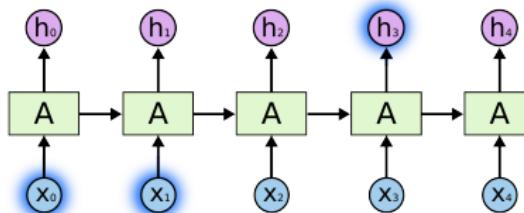
- Similar to back-propagation, but with some constraints on the weights.
 - Since W is used in every step up to the output, we need to backpropagate gradients from $t = 3$ through the network all the way to $t = 0$.
 - Sequences can be quite long, and thus you need to back-propagate through many layers. In practice we truncate the backpropagation to a few steps.



Deep Learning

The Problem of Long-Term Dependencies.

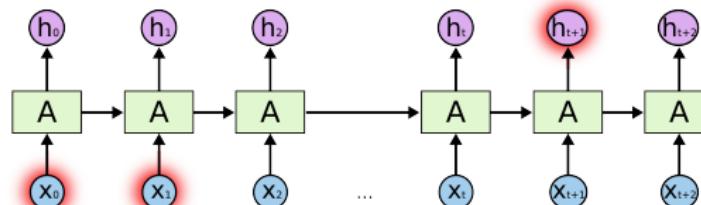
- Sometimes, we only need to look at recent information to perform a prediction. For example, if we are trying to predict the last word in “the clouds are in the sky,” we do not need any further context — it is obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that its needed is small, RNNs can learn to use the past information.



Deep Learning

The Problem of Long-Term Dependencies.

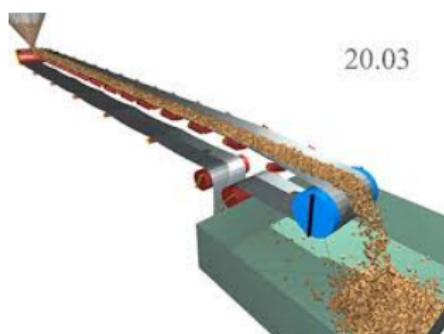
- But there are also cases where we need more context.
Consider trying to predict the last word in the text “I grew up in France... I speak fluent French.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



Deep Learning

Long Short-Term Memory (LSTM).

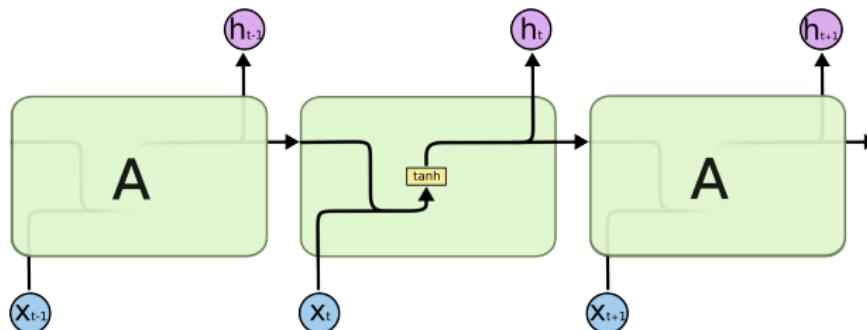
- A special kind of RNN, capable of learning long-term dependencies.
 - Variable length input.



Deep Learning

Long Short-Term Memory (LSTM).

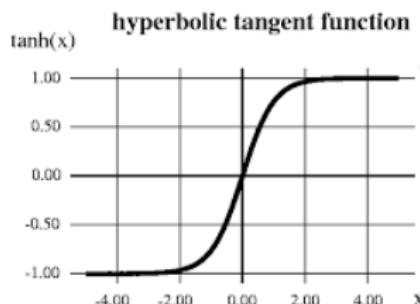
- All recurrent neural networks have the form of a chain of repeating modules.
 - In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.



Deep Learning

Long Short-Term Memory (LSTM).

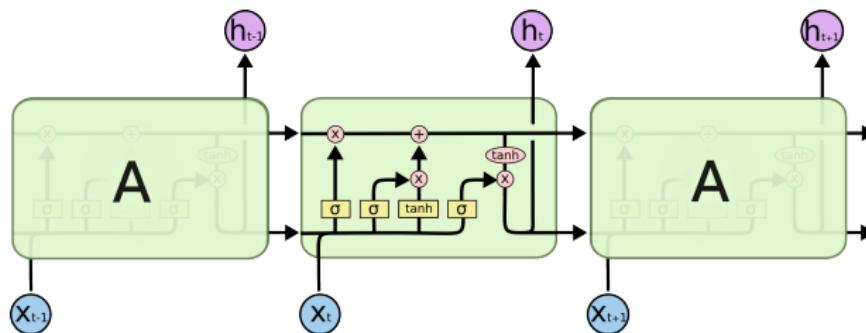
- Hyperbolic Tangent.
 - Ranges from -1 to +1.



Deep Learning

Long Short-Term Memory (LSTM).

- LSTMs also have this chain like structure.
 - But the repeating module has a different structure.
 - A single layer has four components, that we will discuss.

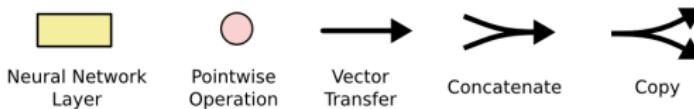


Deep Learning

Long Short-Term Memory (LSTM).

- Notation.

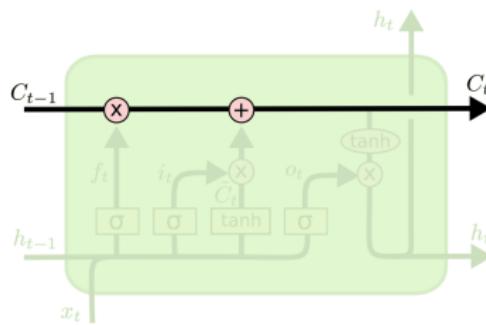
- Each line carries an entire vector.
- The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers.
- Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.



Deep Learning

Long Short-Term Memory (LSTM).

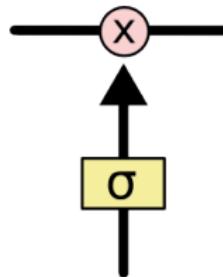
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
 - It runs straight down the entire chain, with only some minor linear interactions.



Deep Learning

Long Short-Term Memory (LSTM).

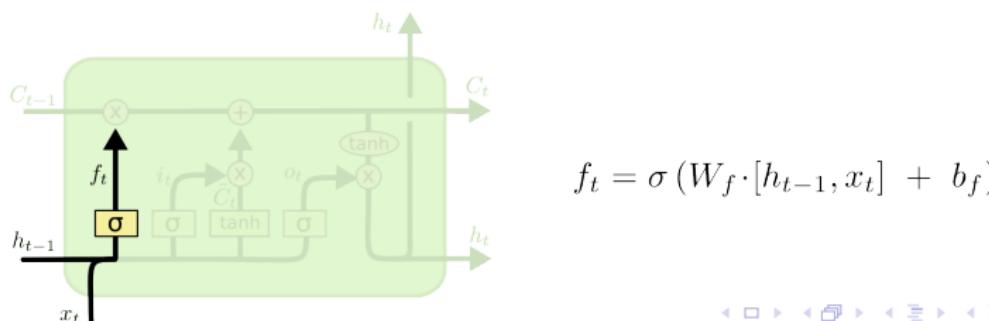
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.
 - An LSTM has three of these gates, to protect and control the cell state.



Deep Learning

Long Short-Term Memory (LSTM).

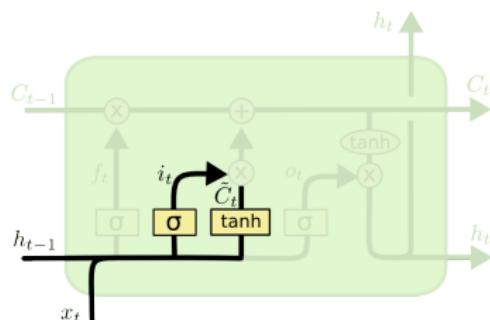
- The first step in our LSTM is to decide what information to throw away from the cell state.
 - This decision is made by a sigmoid layer called the “forget gate layer.”
 - It looks at h_{t-1} and x_t , and outputs a number between 0 and 1 for each number in the cell state C_{t-1} . A 1 represents “completely keep this” while a 0 represents “completely get rid of this.”



Deep Learning

Long Short-Term Memory (LSTM).

- What new information to store in the cell state?
 - First, a sigmoid layer called the “input gate layer” decides which values to update.
 - Next, a tanh layer creates a vector of new candidate values, \hat{C}_t , that could be added to the state.



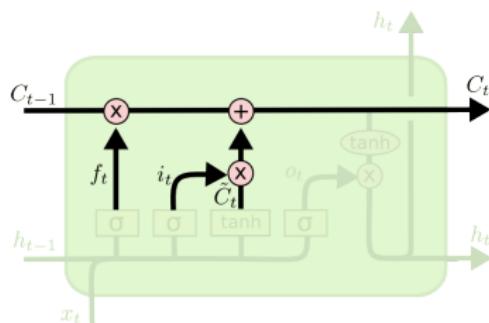
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Deep Learning

Long Short-Term Memory (LSTM).

- Now it is time to update the old cell state, C_{t-1} , into the new cell state C_t .
 - We multiply the old state by f_t , forgetting the things we decided to forget earlier.
 - Then we add $i_t \times \hat{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

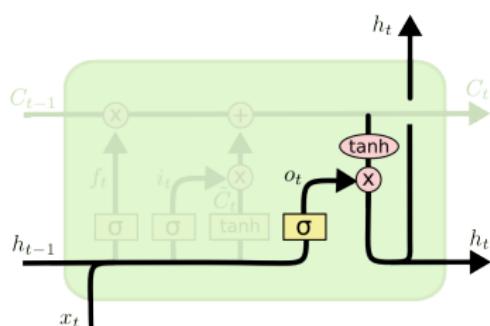


$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t$$

Deep Learning

Long Short-Term Memory (LSTM).

- The output will be based on our cell state.
 - First, we run a sigmoid layer which decides what parts of the cell state to output.
 - Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate.



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

Deep Learning

Long Short-Term Memory (LSTM).

- That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent.
- The central plus sign in both diagrams is essentially the secret of LSTMs. Stupidly simple as it may seem, this basic change helps them preserve a constant error when it must be backpropagated at depth. Instead of determining the subsequent cell state by multiplying its current state with new input, they add the two, and that quite literally makes the difference.

Deep Learning

Discriminative vs. Generative models

- A discriminative model learns a function that maps the input data (x) to some desired output class label (y). In probabilistic terms, they directly learn the conditional distribution $P(y|x)$.
- A generative model tries to learn the joint probability of the input data and labels simultaneously, i.e. $P(x, y)$. The generative ability could be used for creating likely new (x, y) samples.

Both types of models are useful, but generative models have one interesting advantage over discriminative models: they have the potential to understand and explain the underlying structure of the input data even when there are no labels.

Deep Learning

Generative Adversarial Networks.



Deep Learning

Generative Adversarial Networks.



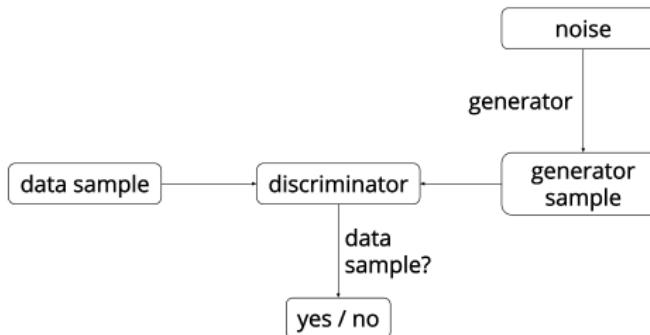
Deep Learning

Generative Adversarial Networks (GANs).

- GANs are a very promising family of generative models. GANs are formulated as a zero-sum game between two networks.
 - A two-player non-cooperative game between the discriminator and the generator.
 - A game is minimax iff it has 2 players and in all states the reward of player 1 is the negative of reward of player 2.
 - Nash Equilibrium: each player does not want to change their actions, given the other players actions.
 - The game is at a Nash equilibrium when neither player can improve its payoff by changing their parameters slightly.

Deep Learning

Generative Adversarial Networks.



- A continuous game, where the generator is learning to produce more realistic samples, and the discriminator is learning to get better at distinguishing generated data from real data. The hope is that the competition will drive the generated samples to be indistinguishable from real data.

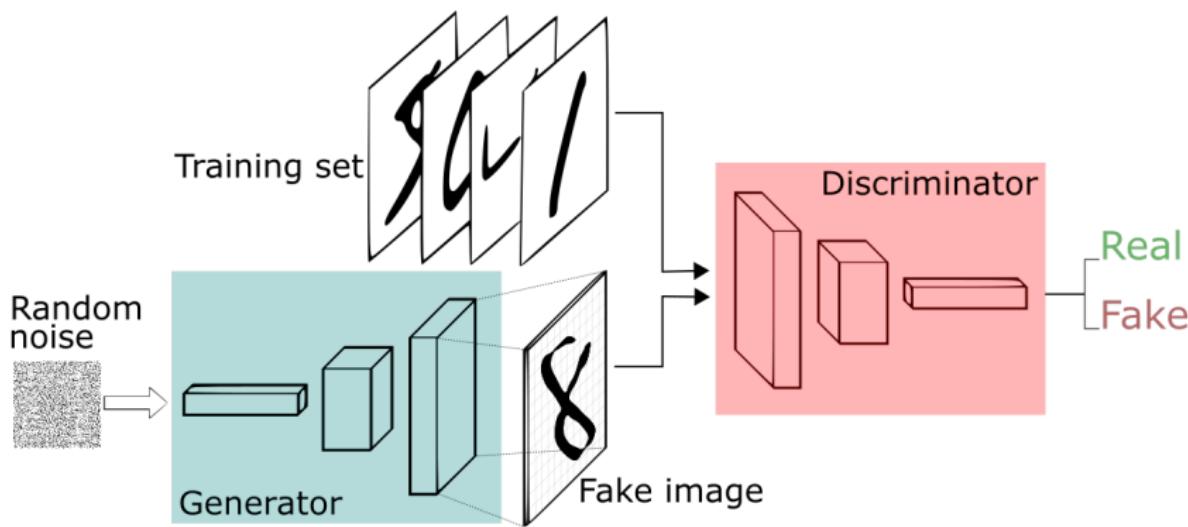
Deep Learning

Generative Adversarial Networks.



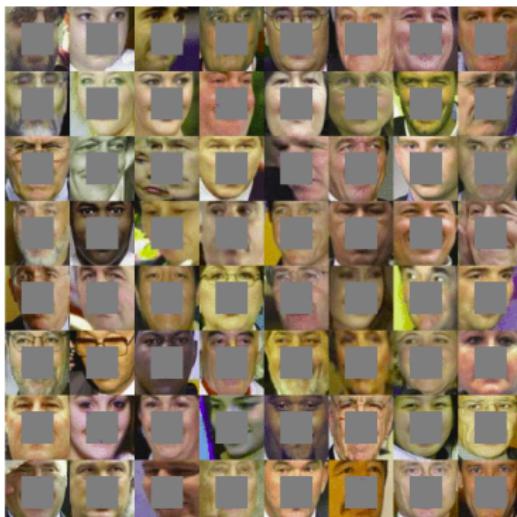
Deep Learning

Generative Adversarial Networks.



Deep Learning

Generative Adversarial Networks.



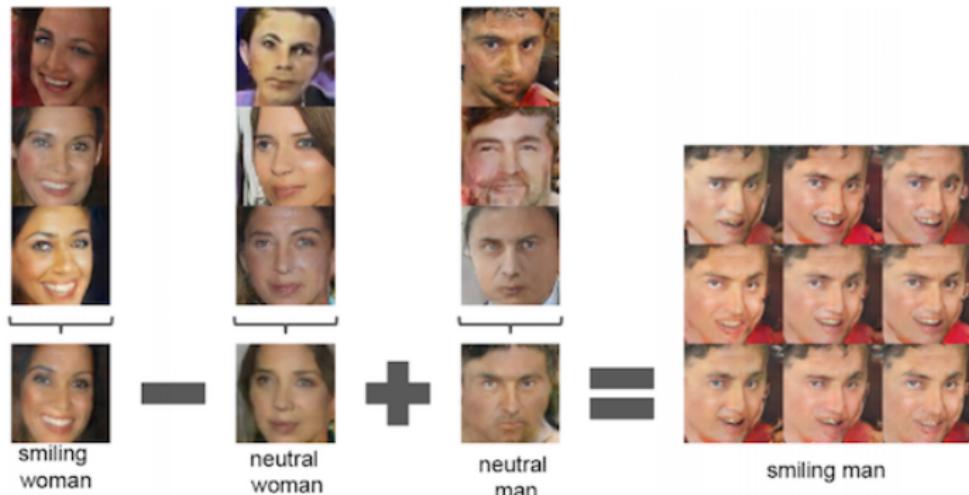
Deep Learning

Generative Adversarial Networks.



Deep Learning

Generative Adversarial Networks.



Deep Learning

Generative Adversarial Networks.



Deep Learning

Generative Adversarial Networks.

- GANs are hard to train.
 - Each side of the GAN can overpower the other. If the discriminator is too good, it will return values so close to 0 or 1 that the generator will struggle to read the gradient. If the generator is too good, it will persistently exploit weaknesses in the discriminator that lead to false negatives. This may be mitigated by the nets respective learning rates.
- When you train the discriminator, hold the generator values constant; and when you train the generator, hold the discriminator constant. Each should train against a static adversary.

Naive Bayes

Simple approach to classification tasks.

- Given:
 - $D = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), \dots, (X^{(n)}, y^{(n)})\}$
 - $X^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_d^{(i)})$ (d -dimensional space)
 - $y^{(i)} \in Y$
- Algorithm:
 - ① Estimate a distribution p_θ from D
 - ② For a given X , compute $\hat{y} = \text{argmax}(p_\theta(y|X))$

Naive Bayes

Probabilistic model to estimate p_θ .

- Conditional probability: $p(y|X) = p(y|x_1, x_2, \dots, x_d) \forall y \in Y$
 - To get $p(y|X)$, we need to count over all possible feature combinations.

Problem:

Naive Bayes

Probabilistic model to estimate p_θ .

- Conditional probability: $p(y|X) = p(y|x_1, x_2, \dots, x_d) \forall y \in Y$
 - To get $p(y|X)$, we need to count over all possible feature combinations.

Problem:

- d may be large.
- Each x_i can take on a large number of values.
 - Building a probability table based on all possible combinations is infeasible.

Naive Bayes

Solution: Reformulate the model using Bayes' theorem.

- The conditional probability $p(y|X)$ can be decomposed as:
 - $p(y|X) = \frac{p(y) \times p(X|y)}{p(X)}$

Naive Bayes

Solution: Reformulate the model using Bayes' theorem.

- The conditional probability $p(y|X)$ can be decomposed as:
 - $p(y|X) = \frac{p(y) \times p(X|y)}{p(X)}$
 - posterior = $\frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$ (Bayes' rule)
- Bayes' rule is used whenever there are uncertainty.
 - Bad news: you tested positive for a serious disease, and that the test is 99% accurate (i.e., the probability of testing positive given that you have the disease is 0.99, as is the probability of testing negative given that you do not have the disease).
 - Good news: this is a rare disease, that affects only 1 in 10,000 people.
- What are the chances that you actually have the disease?

Naive Bayes

- The test is 99% accurate:
 - $p(T = 1|D = 1) = 0.99$
 - $p(T = 0|D = 0) = 0.99$
 - Where T denotes test and D denotes disease.
- The disease affects 1 in 10000:
 - $p(D = 1) = 0.0001$

Naive Bayes

- The test is 99% accurate:
 - $p(T = 1|D = 1) = 0.99$
 - $p(T = 0|D = 0) = 0.99$
 - Where T denotes test and D denotes disease.
- The disease affects 1 in 10000:
 - $p(D = 1) = 0.0001$
- $p(D = 1|T = 1) = \frac{p(T=1|D=1) \times p(D=1)}{p(T=1|D=0) \times p(D=0) + p(T=1|D=1) \times p(D=1)} = 0.0098$

Naive Bayes

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

- In practice, we are only interested in the numerator.
 - The denominator is **constant**, since X is fixed.
 - The numerator is called **joint probability**:
 - $p(y, x_1, x_2, \dots, x_d)$

Naive Bayes

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

- In practice, we are only interested in the numerator.
 - The denominator is **constant**, since X is fixed.
 - The numerator is called **joint probability**:
 - $p(y, x_1, x_2, \dots, x_d)$
 - ... which can be rewritten using the **chain rule**:
 - $$\begin{aligned} p(y, X) &= p(y) \times p(x_1, x_2, \dots, x_d) \\ &= p(y) \times p(x_1|y) \times p(x_2, \dots, x_d|y, x_1) \\ &= p(y) \times p(x_1|y) \times p(x_2|y, x_1) \times p(x_3, \dots, x_d|y, x_1, x_2) \\ &= p(y) \times p(x_1|y) \times p(x_2|y, x_1) \times \dots \times p(x_d|y, x_1, x_2, \dots, x_{d-1}) \end{aligned}$$

Naive Bayes

Assumption: conditional independence.

- Assume that features are conditionally independent.
- Each x_i is seen as a coin flip.
 - $p(x_i|y, x_j) = p(x_i|y)$
 - $p(x_i|y, x_j, x_k) = p(x_i|y)$
 - $p(x_i|y, x_j, x_k, x_l) = p(x_i|y)$

Naive Bayes

Assumption: conditional independence.

- Assume that features are conditionally independent.
- Each x_i is seen as a coin flip.
 - $p(x_i|y, x_j) = p(x_i|y)$
 - $p(x_i|y, x_j, x_k) = p(x_i|y)$
 - $p(x_i|y, x_j, x_k, x_l) = p(x_i|y)$
- Now we can estimate p_θ
 - $$\begin{aligned} p(y|x_1, x_2, \dots, x_d) &\approx p(y, x_1, x_2, \dots, x_d) \\ &\approx p(y) \times p(x_1|y) \times p(x_2|y) \times \dots \times p(x_d|y) \end{aligned}$$
 - $$\approx p(y) \times \prod_{i=1}^d p(x_i|y)$$

Naive Bayes

The log-sum-exp trick.

- Multiplying probabilities may quickly lead to underflow.
- $\prod_{i=1}^d p(x_i|y) \rightarrow 0$ as d increases
- Compute probabilities in the log domain.
 - Multiplying probabilities is equivalent to adding them in the log domain.
 - $p(A) \times p(B) = \log(\exp(A) + \exp(B))$
 - $= \log(\exp(A - m) + \exp(B - m)) + m$, where $m = \max(A, B)$

Naive Bayes

Example:

Table: Naive Bayes.

chills	runny nose	headache	fever	flu?
Y	N	Mild	Y	N
Y	Y	No	N	Y
Y	N	Strong	Y	Y
N	Y	Mild	Y	Y
N	N	No	N	N
N	Y	Strong	Y	Y
N	Y	Strong	N	N
Y	Y	Mild	Y	Y

Naive Bayes

We need to compute only individual probabilities $p(x_i|y)$.

Table: Joint probability distribution $p\theta(y|X)$.

$p(\text{flu}=Y)$	0.625	$p(\text{flu}=N)$	0.375
$p(\text{chills}=Y \text{flu}=Y)$	0.600	$p(\text{chills}=Y \text{Flu}=N)$	0.333
$p(\text{chills}=N \text{flu}=Y)$	0.400	$p(\text{chills}=N \text{Flu}=N)$	0.667
$p(\text{runny nose}=Y \text{flu}=Y)$	0.800	$p(\text{runny nose}=Y \text{Flu}=N)$	0.333
$p(\text{runny nose}=N \text{flu}=Y)$	0.200	$p(\text{runny nose}=N \text{Flu}=N)$	0.667
$p(\text{headache}=\text{Mild} \text{flu}=Y)$	0.400	$p(\text{headache}=\text{Mild} \text{Flu}=N)$	0.333
$p(\text{headache}=\text{No} \text{flu}=Y)$	0.200	$p(\text{headache}=\text{No} \text{Flu}=N)$	0.333
$p(\text{headache}=\text{Strong} \text{flu}=Y)$	0.400	$p(\text{headache}=\text{Strong} \text{Flu}=N)$	0.333
$p(\text{fever}=Y \text{flu}=Y)$	0.800	$p(\text{fever}=Y \text{Flu}=N)$	0.333
$p(\text{fever}=N \text{flu}=Y)$	0.200	$p(\text{fever}=N \text{Flu}=N)$	0.667

Naive Bayes

Prediction.

Table: Naive Bayes.

chills	runny nose	headache	fever	flu?
Y	N	Mild	N	?

Naive Bayes

Prediction.

Table: Naive Bayes.

chills	runny nose	headache	fever	flu?
Y	N	Mild	N	?

- Return the argmax:

- $p(\text{flu} = Y) \times p(\text{chills} = Y|\text{flu} = Y) \times p(\text{nose} = N|\text{flu} = Y) \times p(\text{headache} = \text{Mild}|\text{flu} = Y) \times p(\text{fever} = N|\text{flu} = Y) = 0.0060$
- $p(\text{flu} = N) \times p(\text{chills} = Y|\text{flu} = N) \times p(\text{nose} = N|\text{flu} = N) \times p(\text{headache} = \text{Mild}|\text{flu} = N) \times p(\text{fever} = N|\text{flu} = N) = 0.0185$

Naive Bayes

Continuous data.

- Attribute x_i assumes continuous values according to a **Gaussian distribution**.
 - First, segment the data by the class.
 - Then, compute the mean and the variance of x_i .
- Let:
 - μ_y be the mean of the values in x_i associated with class y .
 - σ_y^2 be the variance of the values in x_i associated with class y .
- Then:
 - $p(x_i = v|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(v-\mu_y)^2}{2\sigma_y^2}}$

Naive Bayes

Continuous data.

- Attribute x_i assumes continuous values according to a **Gaussian distribution**.
 - First, segment the data by the class.
 - Then, compute the mean and the variance of x_i .
- Let:
 - μ_y be the mean of the values in x_i associated with class y .
 - σ_y^2 be the variance of the values in x_i associated with class y .
- Then:
 - $p(x_i = v|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(v-\mu_y)^2}{2\sigma_y^2}}$
- Alternative: discretize the data.
 - Throw away discriminative information (whenever continuous data is discretized, there is always some amount of discretization error).

Naive Bayes

Example.

Table: Naive Bayes.

height	weight	foot size	sex?
6.00	180	12	M
5.92	190	11	M
5.58	170	12	M
5.92	165	10	M
5.00	100	6	F
5.50	150	8	F
5.42	130	7	F
7.75	150	9	F

Naive Bayes

Table: Naive Bayes.

sex	$\mu(\text{height})$	$\sigma(\text{height})$	$\mu(\text{weight})$	$\sigma(\text{weight})$	$\mu(\text{foot})$	$\sigma(\text{foot})$
M	5.855	3.503e-02	176.25	1.229e+02	11.25	9.166e-01
F	5.417	9.722e-02	132.50	5.583e+02	7.50	1.666e+00

Naive Bayes

Table: Naive Bayes.

height	weight	foot size	sex?
6	130	8	?

- $p(y = M) = 0.50$.
- $p(\text{height} = 6|y = M) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(6-\mu)^2}{2\sigma^2}} \approx 1.5789$
- $p(\text{weight} = 130|y = M) = 5.9881 \times 10^{-6}$
- $p(\text{footsize} = 8|y = M) = 1.3112 \times 10^{-3}$
 - Joint probability (male): 6.1984×10^{-9}

Naive Bayes

Table: Naive Bayes.

height	weight	foot size	sex?
6	130	8	?

- $p(y = M) = 0.50$.
- $p(\text{height} = 6|y = M) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(6-\mu)^2}{2\sigma^2}} \approx 1.5789$
- $p(\text{weight} = 130|y = M) = 5.9881 \times 10^{-6}$
- $p(\text{footsize} = 8|y = M) = 1.3112 \times 10^{-3}$
 - Joint probability (male): 6.1984×10^{-9}
 - Joint probability (female): 5.3778×10^{-4}

Naive Bayes

Histogram data (bag of words).

- Multinomial distribution.
 - Inputs represent frequencies: $X = \{p_1, p_2, \dots, p_d\}$.
 - X is a histogram.
 - For example, $X^{(i)}$ is a document, which is represented by the number of times each word has happened in it.
- The likelihood of a word w are:
 - $p(w|y) = \frac{\text{count}(w,c)+1}{\text{count}(c)+|V|}$

Naive Bayes

Table: Naive Bayes.

	Doc	Words	Class?
Training	1	chinese beijing chinese	c
	2	chinese chinese shanghai	c
	3	chinese macao	c
	4	tokyo japan chinese	j
Test	5	chinese chinese chinese tokyo japan	?

Naive Bayes

- $p(chinese|c) = \frac{5+1}{8+6} = \frac{6}{14} = \frac{3}{7}$
- $p(tokyo|c) = \frac{1}{14}$
- $p(japan|c) = \frac{1}{14}$
- $p(chinese|j) = \frac{1+1}{3+6} = \frac{2}{9}$
- $p(tokyo|j) = \frac{2}{9}$
- $p(japan|j) = \frac{2}{9}$
 - $p(c|d_5) = \frac{3}{4} \times (\frac{2}{9})^3 \times \frac{2}{9} \times \frac{2}{9} = 0.0003$
 - $p(j|d_5) = \frac{1}{4} \times (\frac{2}{9})^3 \times \frac{2}{9} \times \frac{2}{9} = 0.0001$

Naive Bayes

Advantages of assuming conditional independence.

- We can estimate p_θ very fast.

Naive Bayes

Advantages of assuming conditional independence.

- We can estimate p_θ very fast.
- We can estimate p_θ more accurately with less data.
 - Data containing potentially “all” feature combinations.
 - From statistics:
 - a “wrong” (approximate) but simple model is usually statistically better than a “correct” but complicated one.
 - Curse of dimensionality: The distribution can be independently estimated from one-dimensional distributions. This helps alleviate problems stemming from the curse of dimensionality, such as the need for data sets that scale exponentially with the number of features.

Boosting



Boosting

One of the most powerful ideas in machine learning.

- Until now we talk about using only one model to do something.

Boosting

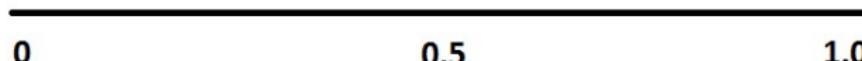
One of the most powerful ideas in machine learning.

- Until now we talk about using only one model to do something.
 - Does a crowd is smarter than the individuals in the crowd?
 - Build a strong classifier from weak ones.
 - Simple to implement.

Boosting

Lets assume **binary classification**.

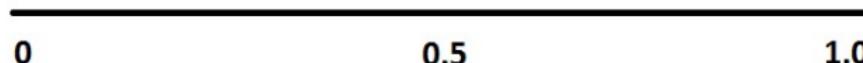
- $h(X) \rightarrow \{-1, +1\}$



Boosting

Lets assume **binary classification**.

- $h(X) \rightarrow \{-1, +1\}$

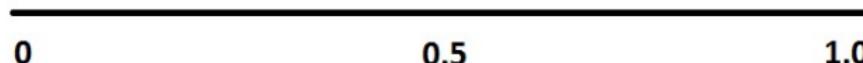


- What is a strong classifier?

Boosting

Lets assume **binary classification**.

- $h(X) \rightarrow \{-1, +1\}$

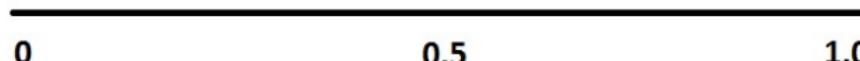


- What is a strong classifier?
 - Near the extremes.
- What is a weak classifier?

Boosting

Lets assume **binary classification**.

- $h(X) \rightarrow \{-1, +1\}$



- What is a strong classifier?
 - Near the extremes.
- What is a weak classifier?
 - Little better than flipping a coin.

Boosting

Can we achieve a strong classifier by combining weak classifiers and letting them vote?

- $h^*(X) = \text{sign}(h^1(X) + h^2(X) + h^3(X))$

Boosting

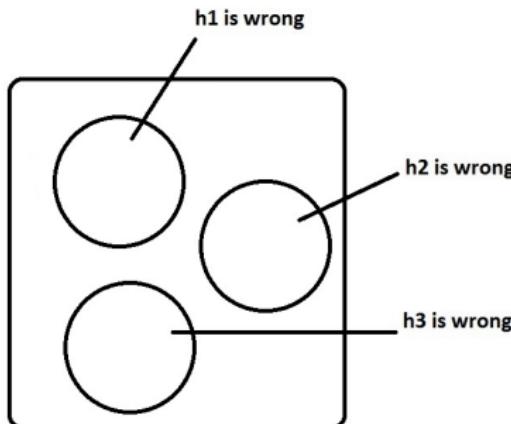
Can we achieve a strong classifier by combining weak classifiers and letting them vote?

- $h^*(X) = \text{sign}(h^1(X) + h^2(X) + h^3(X))$
 - One classifier can be wrong, as long the other two are right.

Boosting

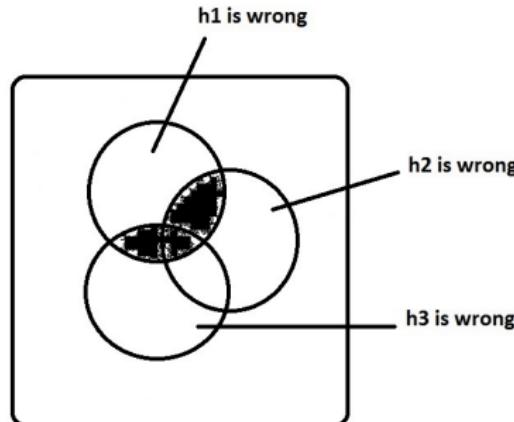
Can we achieve a strong classifier by combining weak classifiers and letting them vote?

- $h^*(X) = \text{sign}(h^1(X) + h^2(X) + h^3(X))$
 - One classifier can be wrong, as long the other two are right.
- What happens if h^1, h^2, h^3 are independent?



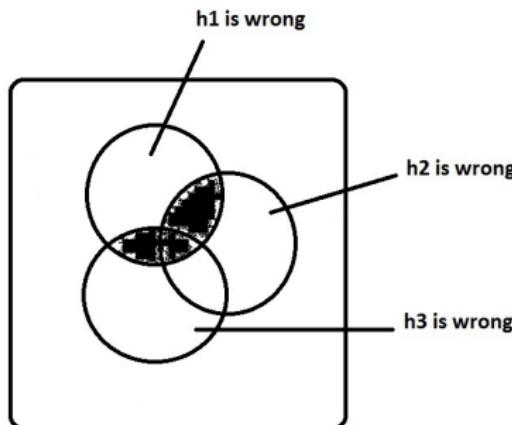
Boosting

- But, classifiers are dependent.



Boosting

- But, classifiers are dependent.

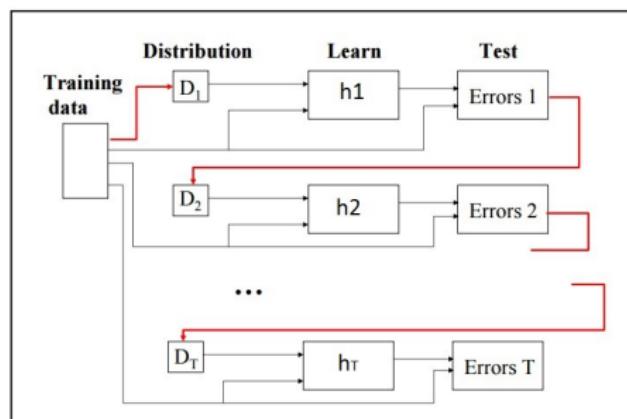


- How to minimize the dependency between the classifiers?

Boosting

Weighting training examples.

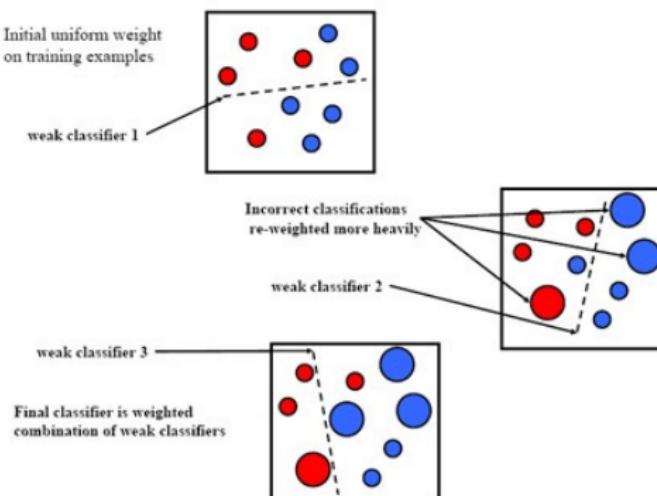
- Focus on difficult examples.



Boosting

Weighting training examples.

- Examples that have been misclassified by the previous weak classifier.

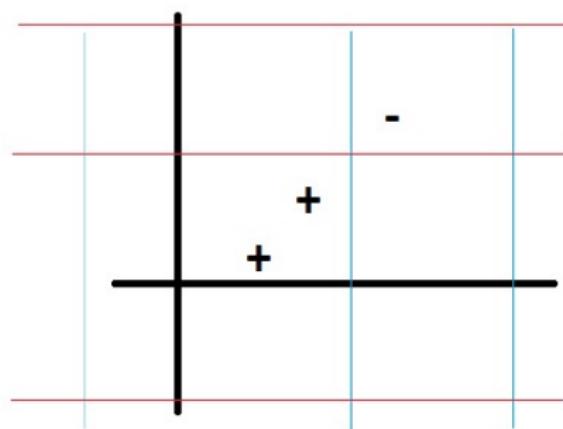


Boosting

Weak classifiers must be simple.

- Perceptron.
- Linear SVM.
- **Decision stumps.**
 - Fast.
 - Simple and easy to program.
 - Few parameters.
 - Weak classifiers too complex leads to overfitting.

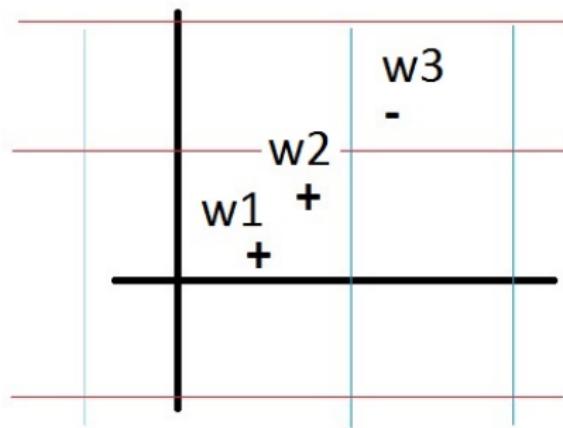
Boosting



Boosting

A weight w_i is assigned to each example.

- Initially, all examples have the same importance (the same weight $\frac{1}{n}$).
- $\sum w_i = 1$



Boosting

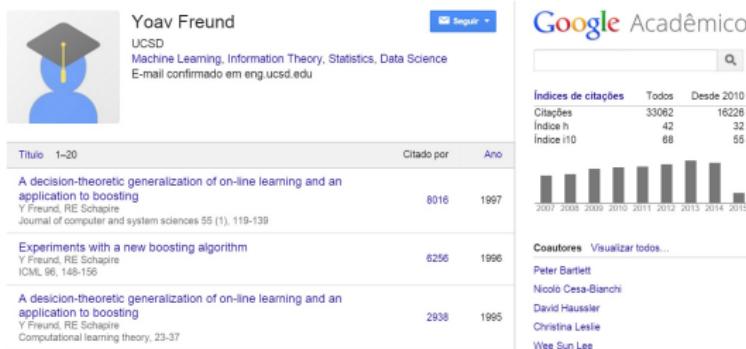
Adaboost (adaptive boosting).

- How to update the weights?
- How to calculate α
- How to choose the weak classifier?

Boosting

Adaboost (adaptive boosting).

- How to update the weights?
- How to calculate α
- How to choose the weak classifier?



Boosting

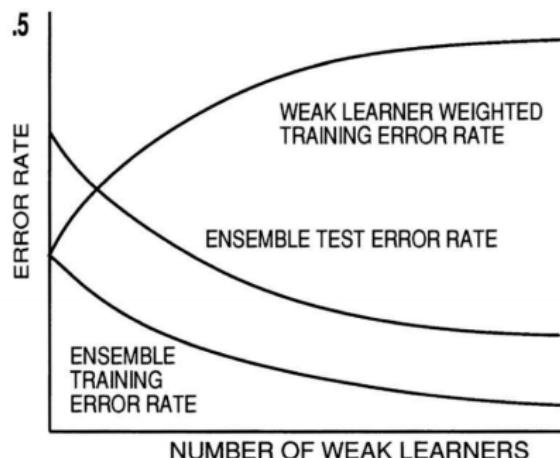
Adaboost (adaptive boosting).

- $h^*(X) = \alpha^1 \times h^1(X) + \alpha^2 \times h^2(X) + \alpha^3 \times h^3(X) + \dots$
- Some classifiers are more important than others.

Boosting

Adaboost (adaptive boosting).

- $h^*(X) = \alpha^1 \times h^1(X) + \alpha^2 \times h^2(X) + \alpha^3 \times h^3(X) + \dots$
- Some classifiers are more important than others.



Boosting

- ➊ Let $w_i^1 = \frac{1}{n}$
- ➋ Loop:
 - ➌ Pick h_t that minimizes the empirical error at iteration t .
 - ➍ Calculate α^t
 - ➎ Calculate w_i^{t+1}
- ➏ Until the empirical error is near 0.

Boosting

Weight update.

- w_i^t and α^t are associated.
- $w_i^{t+1} = \frac{w_i^t}{z} \times e^{-\alpha^t \times h^t(X) \times y(X)}$
 - Where $y(X)$ returns the correct output $\{-1, +1\}$.
 - z is a normalizer factor that ensures that $\sum w_i^{t+1} = 1$.
- If $h^t(X) \neq y(X)$ then $w_i^{t+1} > w_i^t$
- $\alpha^t = \frac{1}{2} \log \frac{1-\epsilon^t}{\epsilon^t}$
 - Where ϵ^t is the empirical error at iteration t .

Boosting

Table: Boosting.

id	Vampire	Evil	Transforms	Sparkly
1	Y	N	Y	Y
2	Y	Y	Y	Y
3	Y	Y	N	N
4	Y	Y	N	Y
5	Y	Y	N	Y
6	Y	N	Y	Y
7	Y	N	Y	Y
8	N	N	Y	N
9	N	Y	N	N
10	N	N	N	Y

Boosting

Table: Stumps.

Stump	Test	Value	Misclassified
A	Evil	Y	1,6,7,9
B	Transforms	Y	3,4,5,8
C	Sparkly	Y	3,10
D	TRUE	-	8,9,10
E	Evil	N	2,3,4,5,8,10
F	Transforms	N	1,2,6,7,9,10
G	Sparkly	N	1,2,4,5,6,7,8,9
H	FALSE	-	1,2,3,4,5,6,7

Boosting

	Round 1	Round 2	Round 3
w_1			
w_2			
w_3			
w_4			
w_5			
w_6			
w_7			
w_8			
w_9			
w_{10}			
Stump			
ϵ			
α			

Boosting

	Round 1	Round 2	Round 3
w_1	$\frac{1}{10}$		
w_2	$\frac{1}{10}$		
w_3	$\frac{1}{10}$		
w_4	$\frac{1}{10}$		
w_5	$\frac{1}{10}$		
w_6	$\frac{1}{10}$		
w_7	$\frac{1}{10}$		
w_8	$\frac{1}{10}$		
w_9	$\frac{1}{10}$		
w_{10}	$\frac{1}{10}$		
Stump		Sparkly=Y	
ϵ		$\frac{1}{5}$	
α		$\frac{1}{2} \log 4$	

Boosting

	Round 1	Round 2	Round 3
w_1	$\frac{1}{10}$	$\frac{1}{16}$	
w_2	$\frac{1}{10}$	$\frac{1}{16}$	
w_3	$\frac{1}{10}$	$\frac{1}{16}$	
w_4	$\frac{1}{10}$	$\frac{1}{16}$	
w_5	$\frac{1}{10}$	$\frac{1}{16}$	
w_6	$\frac{1}{10}$	$\frac{1}{16}$	
w_7	$\frac{1}{10}$	$\frac{1}{16}$	
w_8	$\frac{1}{10}$	$\frac{1}{16}$	
w_9	$\frac{1}{10}$	$\frac{1}{16}$	
w_{10}	$\frac{1}{10}$	$\frac{1}{16}$	
Stump	Sparkly=Y	Evil=Y	
ϵ	$\frac{1}{5}$	$\frac{4}{16}$	
α	$\frac{1}{2} \log 4$	$\frac{1}{2} \log 3$	

Boosting

	Round 1	Round 2	Round 3
w_1	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w_2	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w_3	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{4}{24}$
w_4	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w_5	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w_6	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w_7	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w_8	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{1}{24}$
w_9	$\frac{1}{10}$	$\frac{1}{16}$	$\frac{3}{24}$
w_{10}	$\frac{1}{10}$	$\frac{4}{16}$	$\frac{4}{24}$
Stump	Sparkly=Y	Evil=Y	TRUE
ϵ	$\frac{1}{5}$	$\frac{4}{16}$	$\frac{7}{24}$
α	$\frac{1}{2} \log 4$	$\frac{1}{2} \log 3$	$\frac{1}{2} \log 2.43$

Boosting

Final classifier.

- $h^*(X) = \text{sign}(\frac{1}{2} \log 4 \times [\text{Sparkly} = Y] + \frac{1}{2} \log 3 \times [\text{Evil} = Y] + \frac{1}{2} \log 2.43 \times [\text{TRUE}])$
- What is the final error?

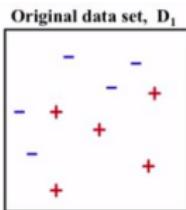
Boosting

Final classifier.

- $h^*(X) = \text{sign}(\frac{1}{2} \log 4 \times [\text{Sparkly} = Y] + \frac{1}{2} \log 3 \times [\text{Evil} = Y] + \frac{1}{2} \log 2.43 \times [\text{TRUE}])$
- What is the final error?
- Overfitting?
- Noise?

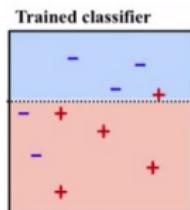
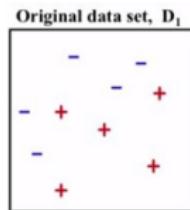
Boosting

Classifying with decision stumps.



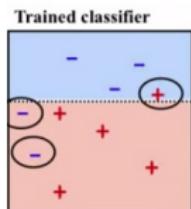
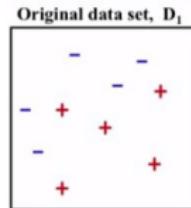
Boosting

Classifying with decision stumps.



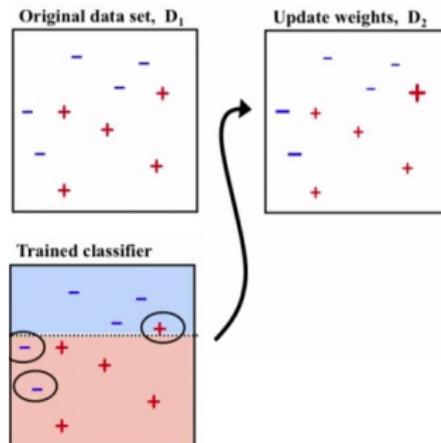
Boosting

Classifying with decision stumps.



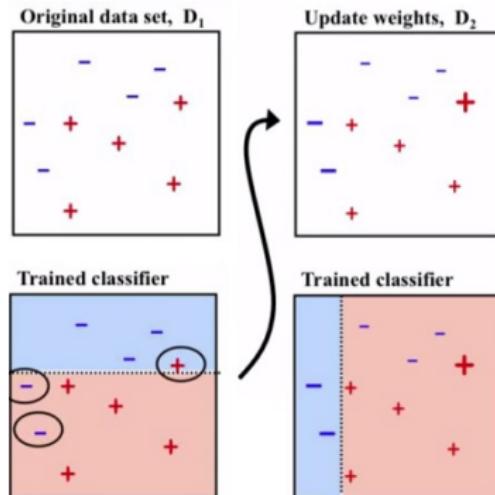
Boosting

Classifying with decision stumps.



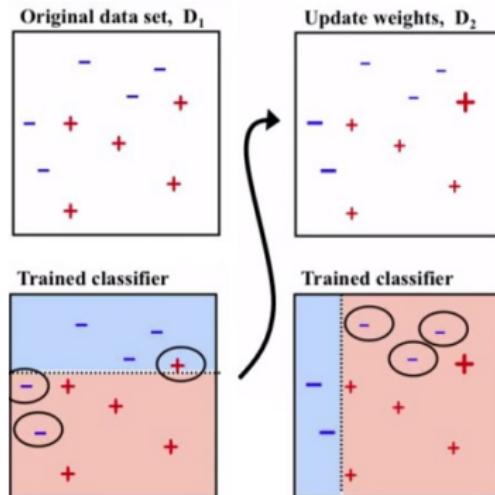
Boosting

Classifying with decision stumps.



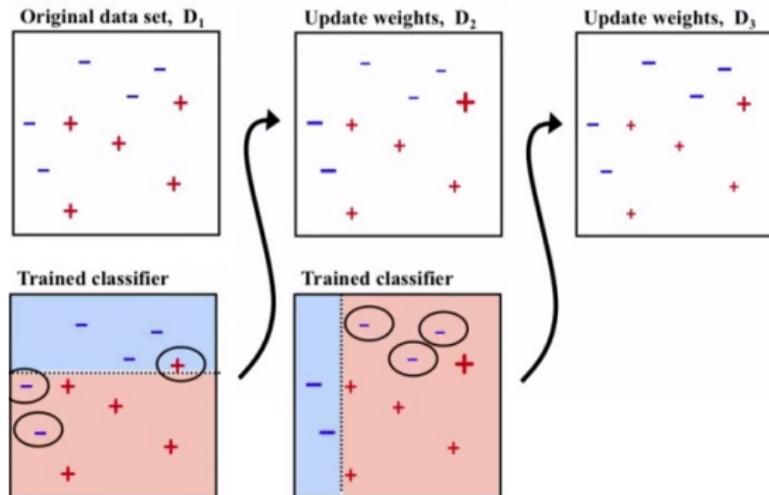
Boosting

Classifying with decision stumps.



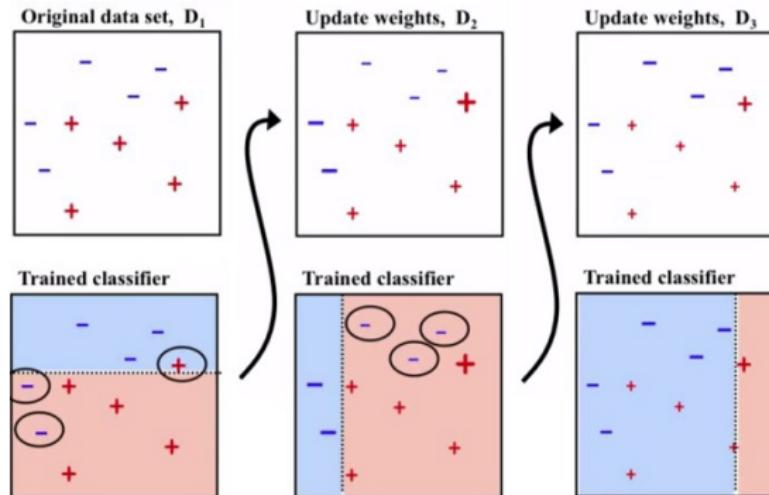
Boosting

Classifying with decision stumps.



Boosting

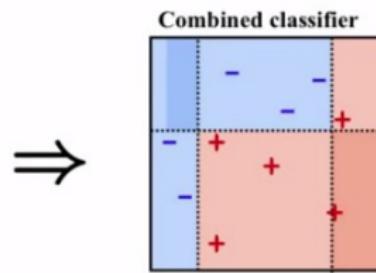
Classifying with decision stumps.



Boosting

Weight each classifier and combine them:

$$.33 * \boxed{\text{blue} \mid \text{orange}} + .57 * \boxed{\text{orange} \mid \text{orange}} + .42 * \boxed{\text{blue} \mid \text{orange}} \wedge \vee 0$$



Bagging

Bootstrap Aggregation.

- Given a training set D
 - Generate m new training sets D_i by sampling uniformly with replacement.
 - Some examples may be duplicated in each D_i .
- This kind of sampling approach is known as **bootstrap**.

Bagging

Bootstrap Aggregation.

- Given training sets D_1, D_2, \dots, D_m
 - Learn a model M_i from each D_i
 - Each model M_i gives an unbiased estimation.
 - Average all models into a final model M .

Bagging

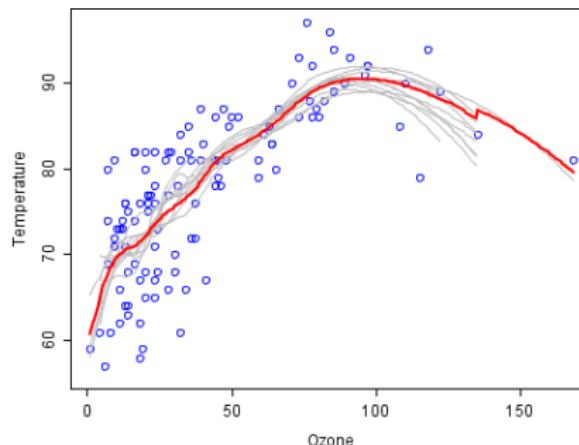
Bootstrap Aggregation.

- Given training sets D_1, D_2, \dots, D_m
 - Learn a model M_i from each D_i
 - Each model M_i gives an unbiased estimation.
 - Average all models into a final model M .
 - Majority voting, averaging probabilities, averaging estimates etc.
- This is known as **model aggregation**.

Bagging

Bagging makes the model less prone to overfitting.

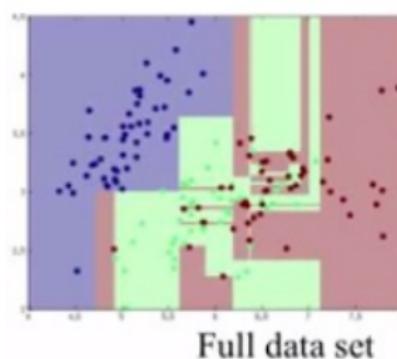
- The intuition is that averaging models makes the final model more robust to variances in the data.
 - It is impossible to memorize D (no M_i sees the entire data).



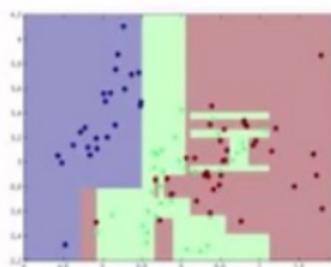
Bagging

It is very common to perform bagging with decision trees.

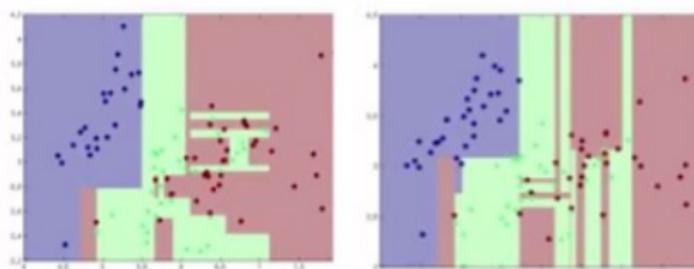
- A single decision tree is not likely to be the best choice.
 - But combining many of them leads to improved performance.



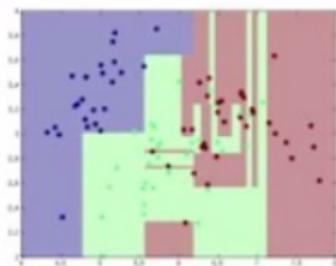
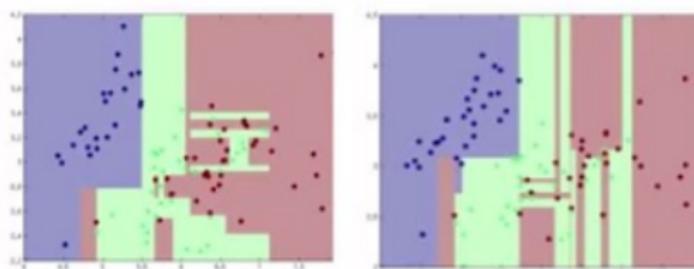
Bagging



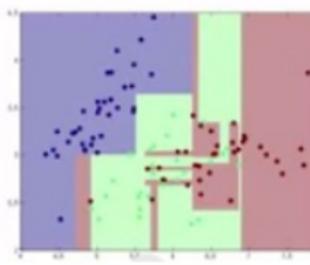
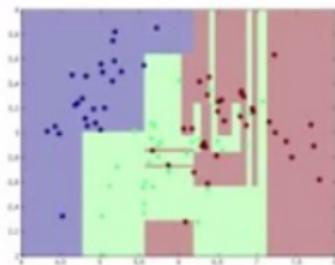
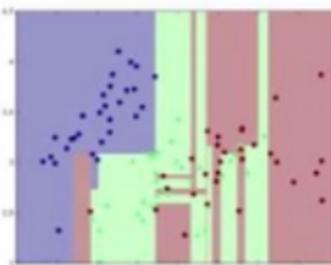
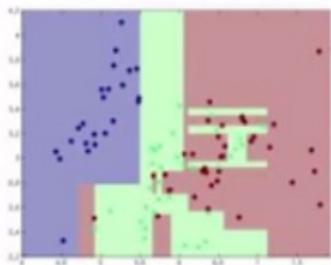
Bagging



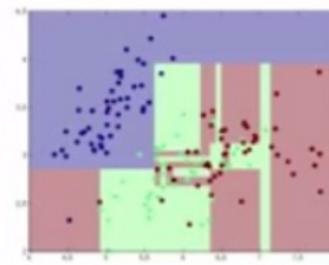
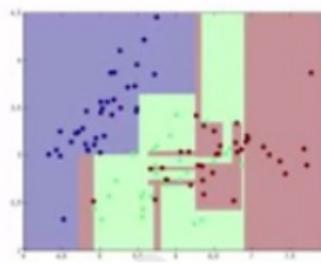
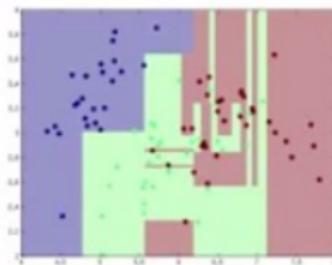
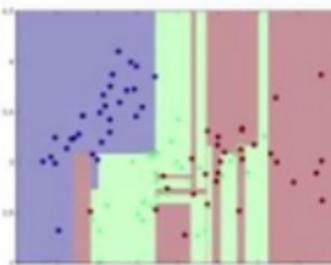
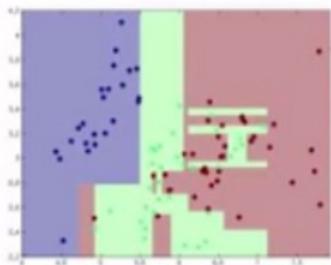
Bagging



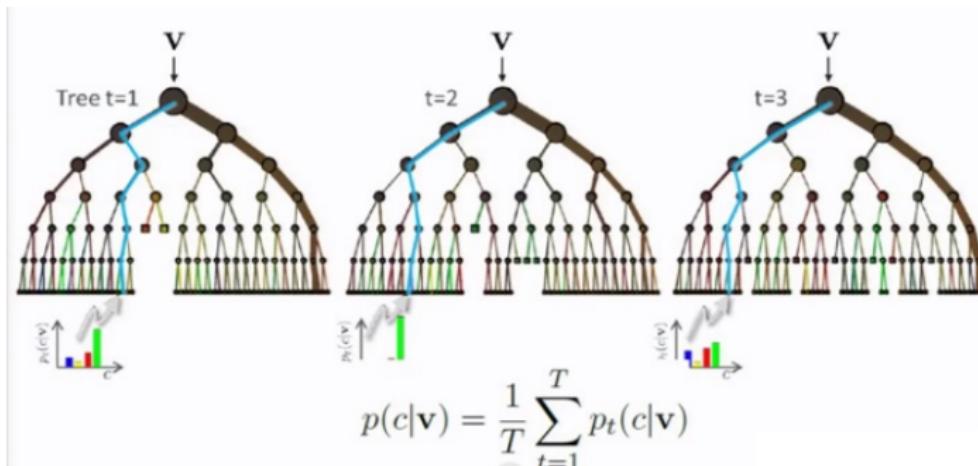
Bagging



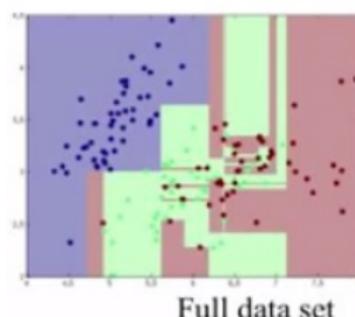
Bagging



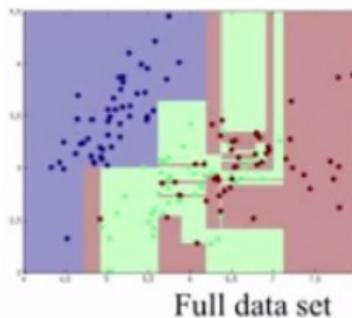
Bagging



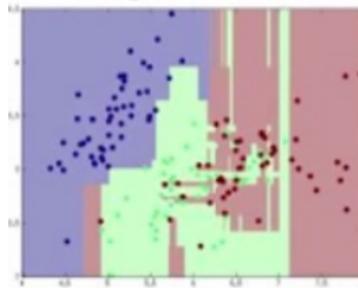
Bagging



Bagging



Avg of 5 trees



Bagging

Random Forests.

- Exploits two sources of randomness
 - Bootstrap sampling.

Bagging

Random Forests.

- Exploits two sources of randomness
 - Bootstrap sampling.
 - Random projection of features.
 - Randomly select \sqrt{n} features.

Bagging

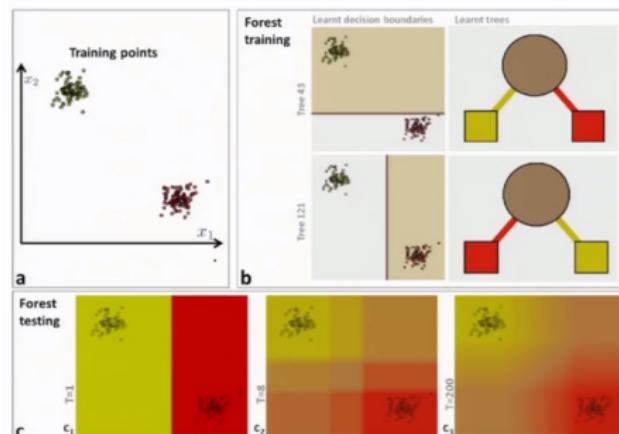
Random Forests.

- Exploits two sources of randomness
 - Bootstrap sampling.
 - Random projection of features.
 - Randomly select \sqrt{n} features.
 - **Trees must be uncorrelated.**
- Basically, random forests correct for decision tree's habit of overfitting.

Bagging

Random Forests.

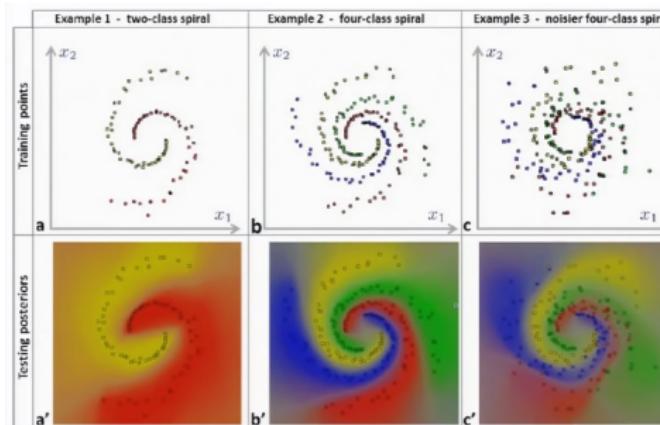
- How many trees should be averaged?



Bagging

Random Forests.

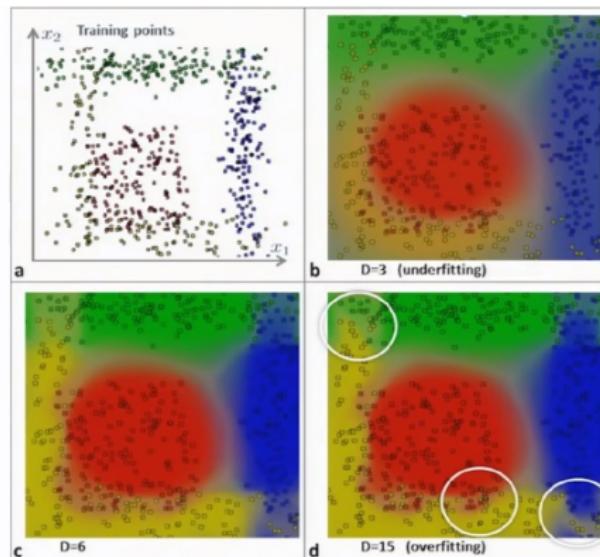
- What is the effect of the number of classes?
- What is the impact of adding noise?



Bagging

Random Forests.

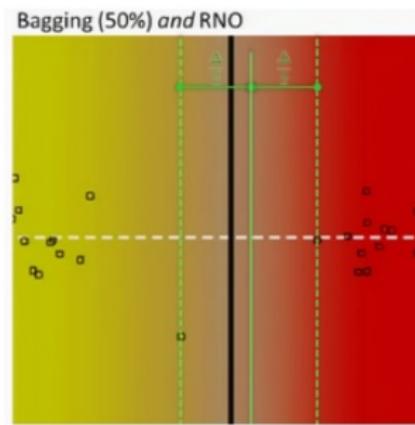
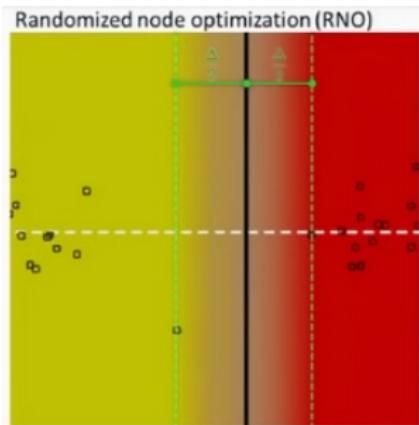
- What is the effect tree complexity (depth)?



Bagging

Random Forests.

- What is the effect of bagging?



Bagging

Comparison of algorithms.

MODEL	COVT	ADULT	MEDIS	SLAC	HS	MG	CALHOUS	COD	BACT	MEAN
BAG-DT	.878	.944*	.762	.898*	.856	.898	.948	.856	.926	.887*
RF	.876	.946*	.785	.912*	.871	.891*	.941	.874	.824	.884
ANN	.764	.884	.791*	.881	.932*	.859	.923	.667	.882	.854
BST-DT	.874	.842	.523	.807	.860	.785	.933	.835	.858	.828
SVM	.696	.819	.600	.859	.788	.776	.833	.864	.763	.781
BST-STMP	.605	.865	.624	.779	.683	.799	.817	.581	.906*	.710
DT	.652	.872	.449	.769	.609	.829	.831	.389	.899*	.708
NB	.552	.843	.011	.714	-.654	.655	.759	.636	.688	.481

Unsupervised Learning

Supervised learning is related to **function approximation**.

- The problem is well defined.

Unsupervised Learning

Supervised learning is related to **function approximation**.

- The problem is well defined.

Unsupervised learning is related to **data description**.

- Much less well defined than supervised learning

Unsupervised Learning

Supervised learning is related to **function approximation**.

- The problem is well defined.

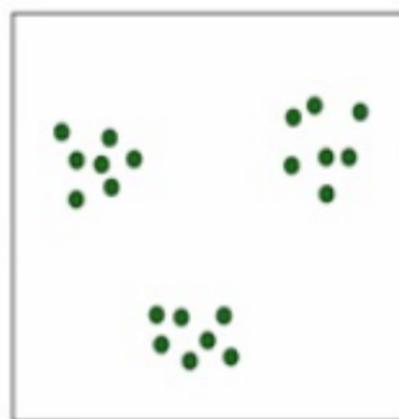
Unsupervised learning is related to **data description**.

- Much less well defined than supervised learning
- Input: $\{x_1, x_2, \dots, x_n\}$
 - Goal: to find patterns that describe the data
 - Clustering: patterns are groups.

Unsupervised Learning

Clustering.

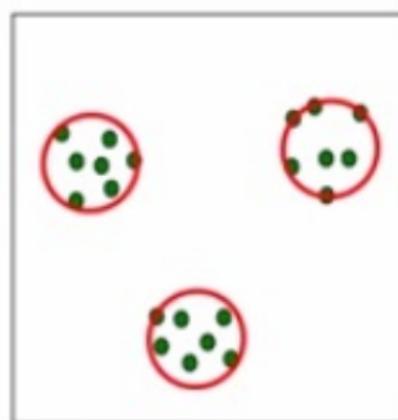
- The meaning of “groups” may vary by data



Unsupervised Learning

Clustering.

- Location



Unsupervised Learning

Clustering.

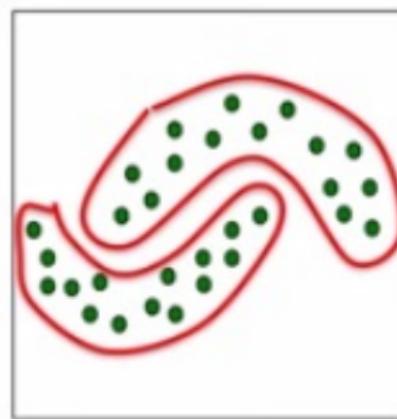
- The meaning of “groups” may vary by data



Unsupervised Learning

Clustering.

- Shape



Unsupervised Learning

Clustering.

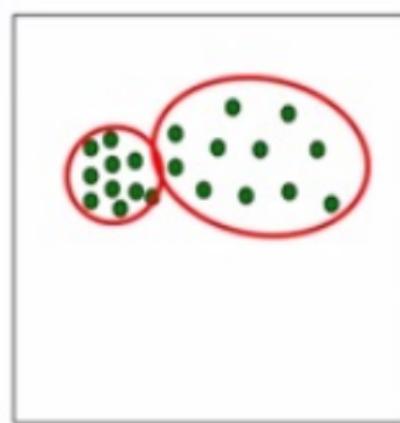
- The meaning of “groups” may vary by data



Unsupervised Learning

Clustering.

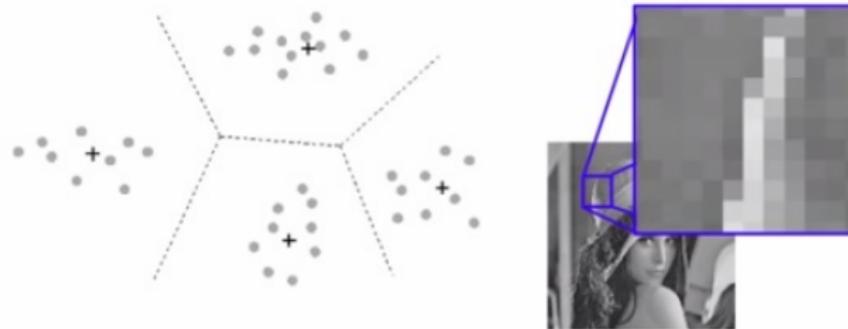
- Density



Unsupervised Learning

Clustering and data compression.

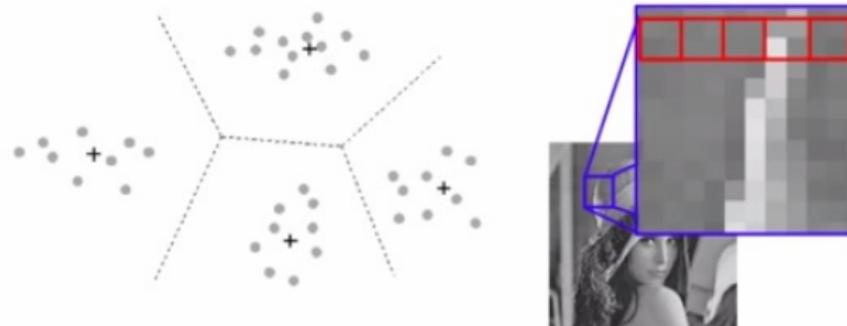
- Clustering is related to vector quantization.
 - Each group has a centroid, which is used as a dictionary of vectors.



Unsupervised Learning

Clustering and data compression.

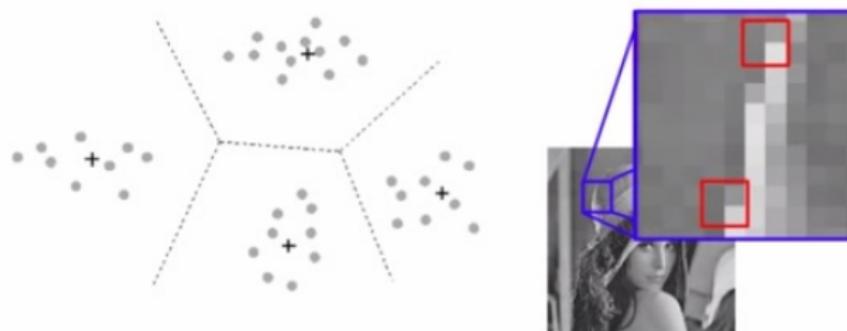
- Clustering is related to vector quantization.
 - Each group has a centroid, which is used as a dictionary of vectors.



Unsupervised Learning

Clustering and data compression.

- Clustering is related to vector quantization.
 - Each group has a centroid, which is used as a dictionary of vectors.



Unsupervised Learning

Hierarchical agglomerative clustering.

- Define a distance function between clusters.

Unsupervised Learning

Hierarchical agglomerative clustering.

- Define a distance function between clusters.
- Initialize:
 - Every example is a cluster.
- Iterate:
 - Compute distance between all pairs of clusters.
 - Merge the two closest clusters.

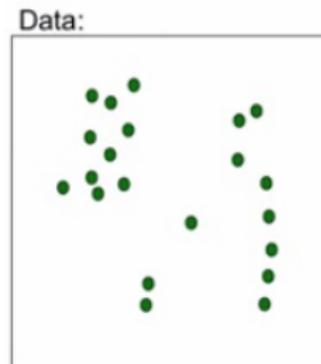
Unsupervised Learning

Hierarchical agglomerative clustering.

- Define a distance function between clusters.
- Initialize:
 - Every example is a cluster.
- Iterate:
 - Compute distance between all pairs of clusters.
 - Merge the two closest clusters.
- This forms a **dendrogram**.

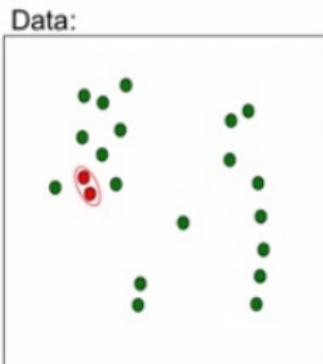
Unsupervised Learning

Hierarchical agglomerative clustering.

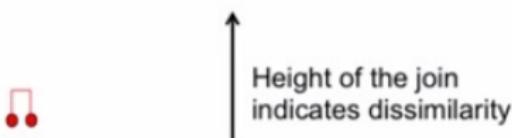


Unsupervised Learning

Hierarchical agglomerative clustering.

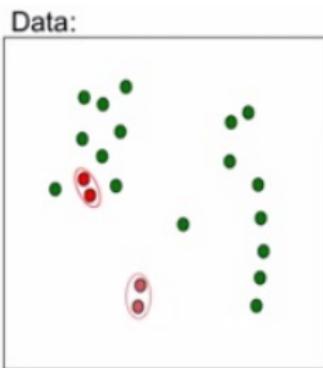


Dendrogram:



Unsupervised Learning

Hierarchical agglomerative clustering.



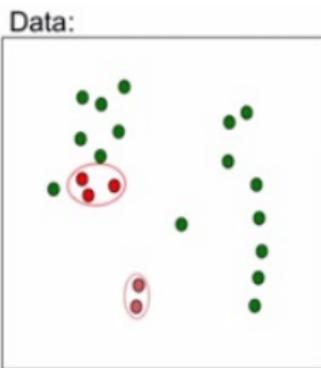
Dendrogram:



Height of the join
indicates dissimilarity

Unsupervised Learning

Hierarchical agglomerative clustering.

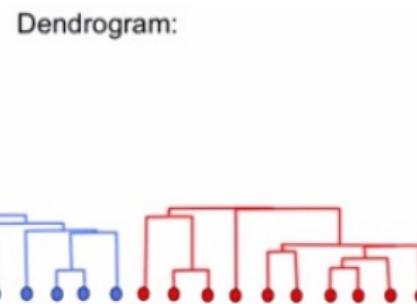
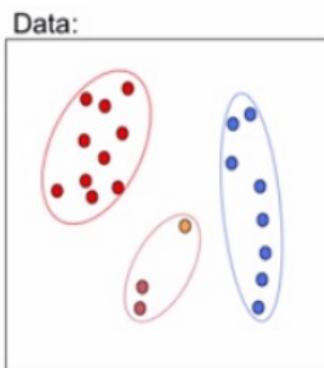


Dendrogram:



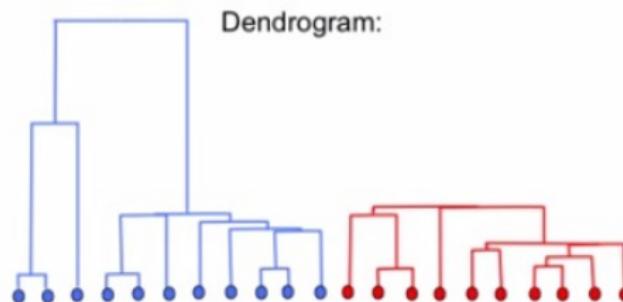
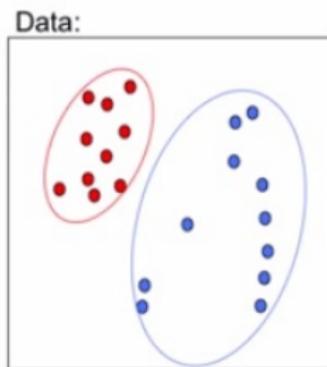
Unsupervised Learning

Hierarchical agglomerative clustering.



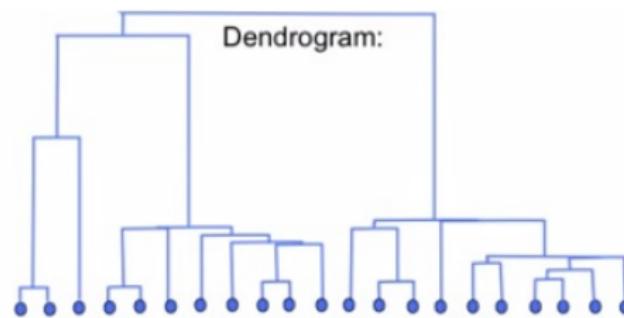
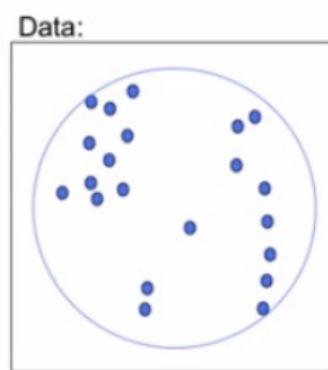
Unsupervised Learning

Hierarchical agglomerative clustering.



Unsupervised Learning

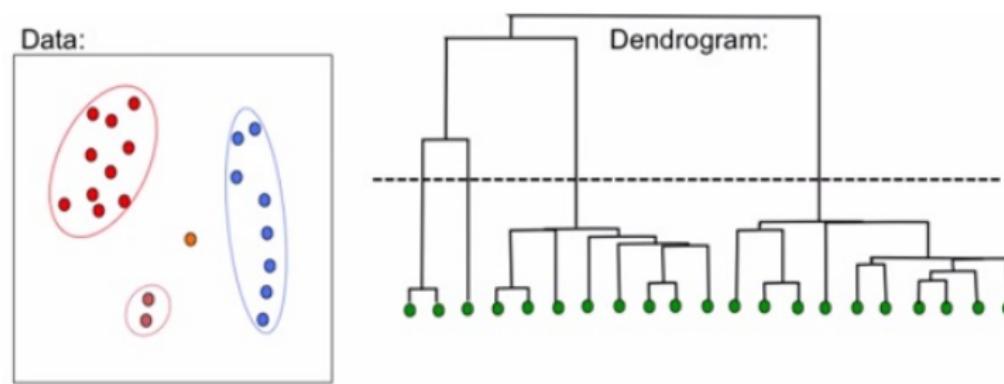
Hierarchical agglomerative clustering.



- The dendrogram illustrates the trace of the merging procedure

Unsupervised Learning

Hierarchical agglomerative clustering.

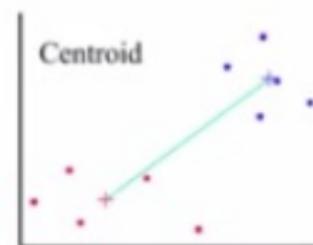
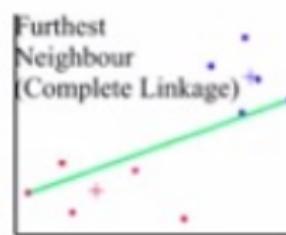
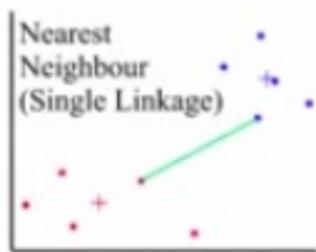


- A cut across the dendrogram corresponds to a similarity threshold

Unsupervised Learning

Distances:

- $D_{min}(C_i, C_j) = \min ||x - y||^2$
- $D_{max}(C_i, C_j) = \max ||x - y||^2$
- $D_{mean}(C_i, C_j) = ||\mu_i - \mu_j||^2$

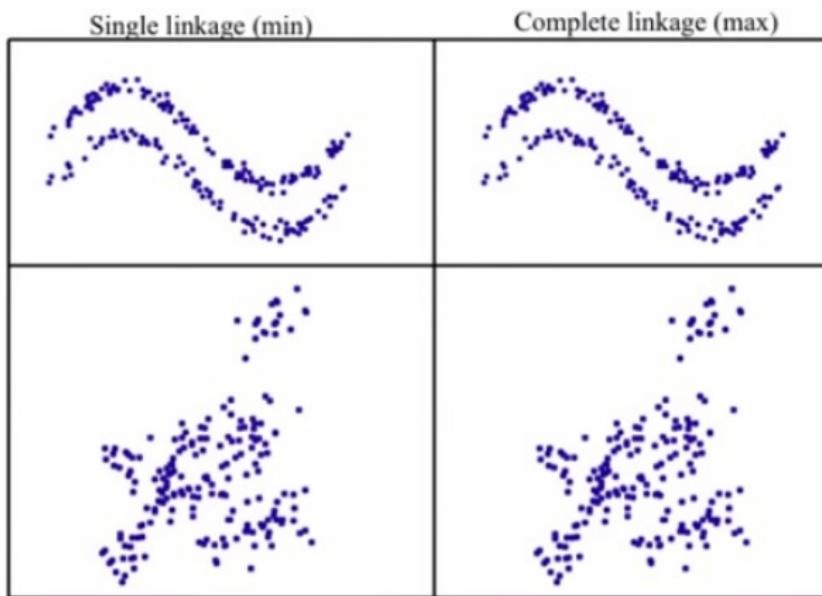


Unsupervised Learning

- Different distances result in different clusterings:

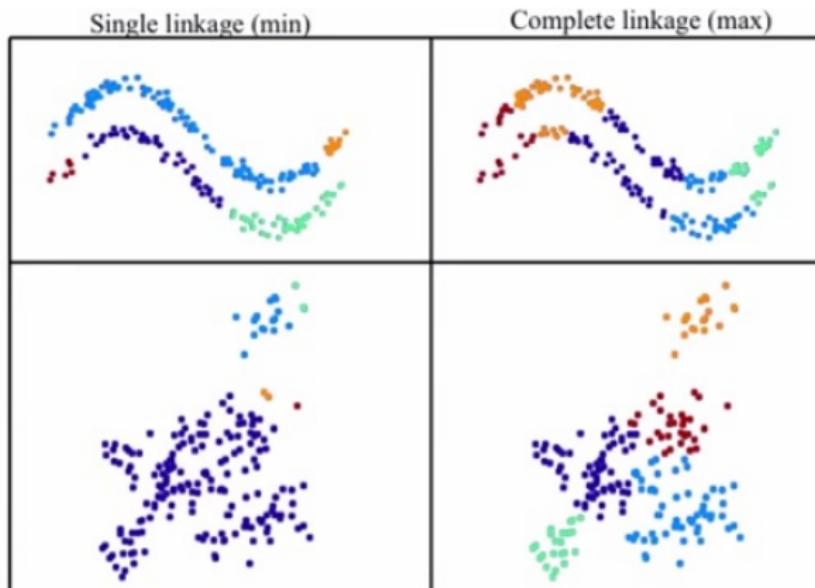
Unsupervised Learning

- Different distances result in different clusterings:



Unsupervised Learning

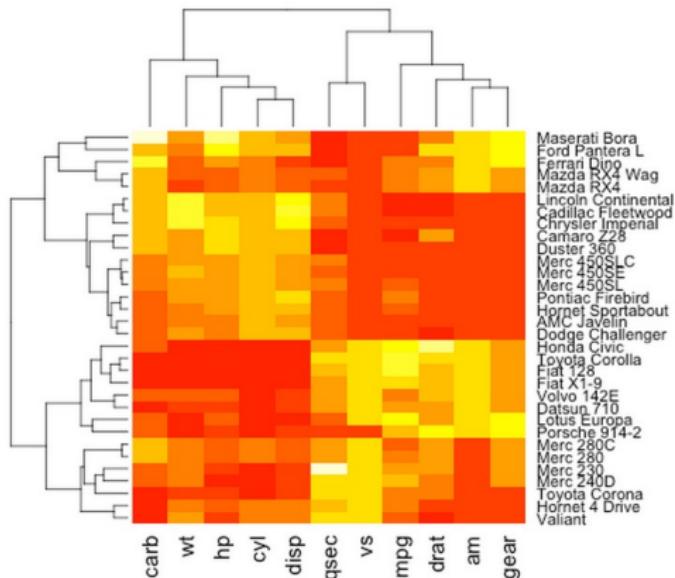
- **max** avoids elongated (or highly separated) clusters.



Unsupervised Learning

Clustergrams.

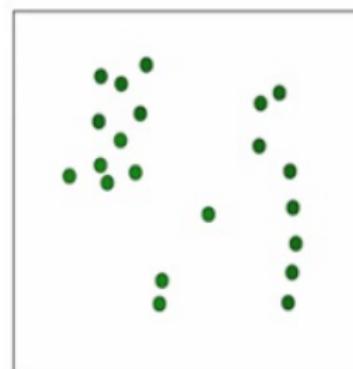
- Sort dimensions according to the clustering order.



Unsupervised Learning

K-Means.

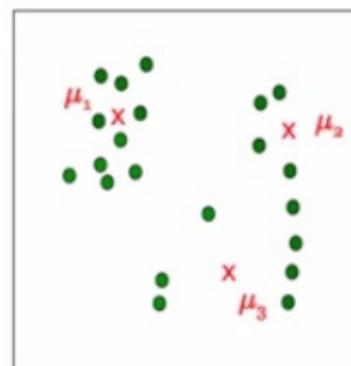
- Iterate between:
 - Updating the assignment of points to clusters.
 - Updating the clusters.
- Assume k clusters.



Unsupervised Learning

K-Means.

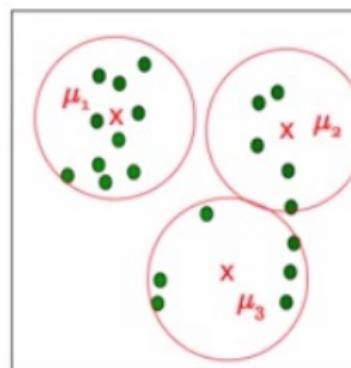
- Iterate between:
 - Updating the assignment of points to clusters.
 - Updating the clusters.
- Each cluster c is given as μ_c .



Unsupervised Learning

K-Means.

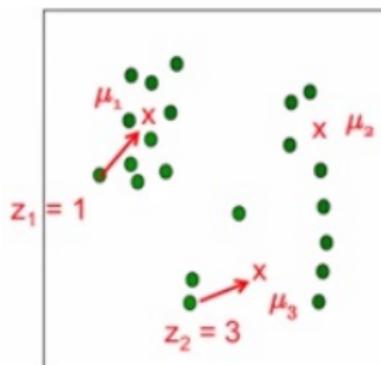
- Iterate between:
 - Updating the assignment of points to clusters.
 - Updating the clusters.
- Each cluster will claim a set of nearby points.



Unsupervised Learning

K-Means.

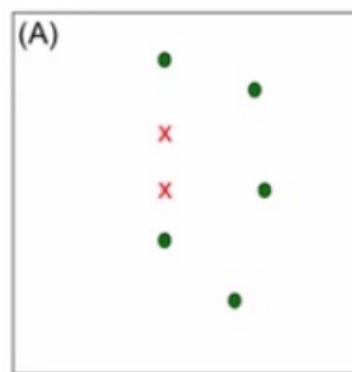
- Iterate between:
 - Updating the assignment of points to clusters.
 - Updating the clusters.
- Each cluster receives a name z_i .



Unsupervised Learning

K-Means.

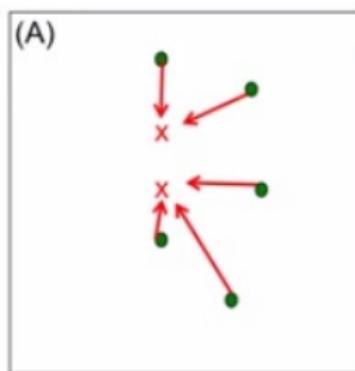
- z_i takes the value c that minimizes distance between point x_i and the centroid μ_c .
 - $z_i = \text{argmin} ||x_i - \mu_c||^2 \forall i$



Unsupervised Learning

K-Means.

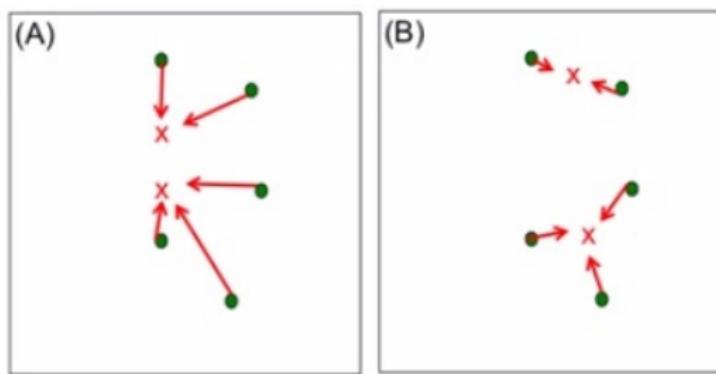
- z_i takes the value c that minimizes distance between point x_i and the centroid μ_c .
 - Perform cluster assignments and update centroids μ_c .



Unsupervised Learning

K-Means.

- z_i takes the value c that minimizes distance between point x_i and the centroid μ_c .
 - Perform cluster assignments and update centroids μ_c .

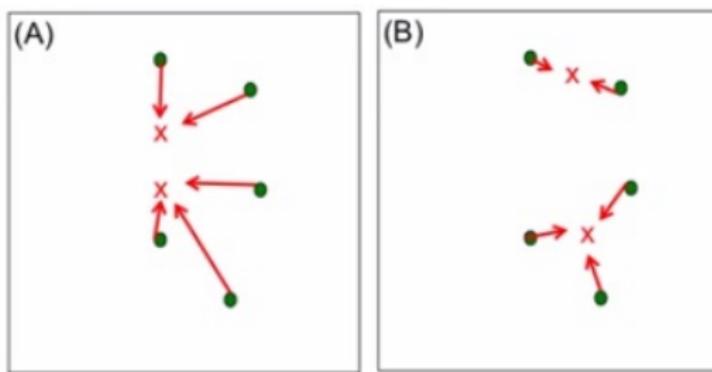


Unsupervised Learning

K-Means.

- Can be viewed as optimizing a loss function:

- $$L = \sum_i ||x_i - \mu_{z_i}||^2$$



Unsupervised Learning

K-Means.

- Can be viewed as optimizing a loss function:
 - $L = \sum_i ||x_i - \mu_{z_i}||^2$
- Guaranteed to converge.
 - If new means lead to the same assignments
 - Same assignments lead to the same means.
 - Same means lead to the same assignments.
 - But there are multiple local minima
 - It depends on the initialization.

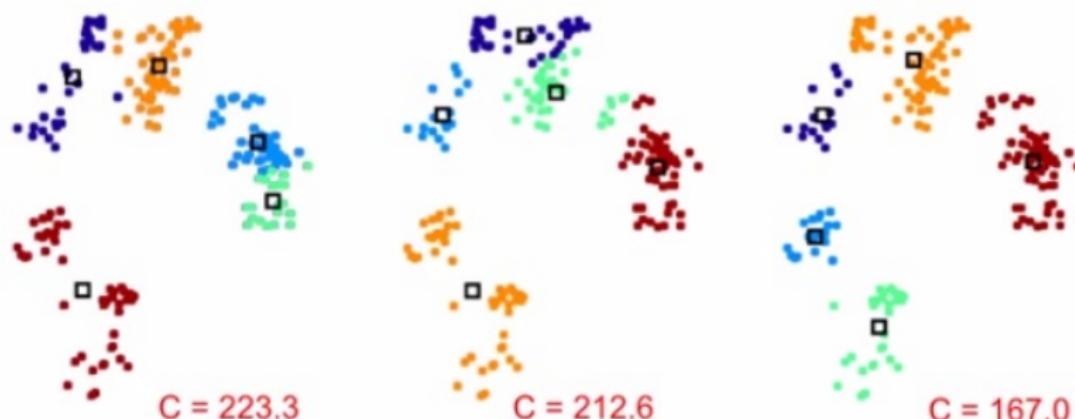
Unsupervised Learning

K-Means.

- Can be viewed as optimizing a loss function:
 - $L = \sum_i ||x_i - \mu_{z_i}||^2$
- Guaranteed to converge.
 - If new means lead to the same assignments
 - Same assignments lead to the same means.
 - Same means lead to the same assignments.
 - But there are multiple local minima
 - It depends on the initialization.
 - Try different (randomized) initializations.
 - Can use loss function C to decide which initialization.

Unsupervised Learning

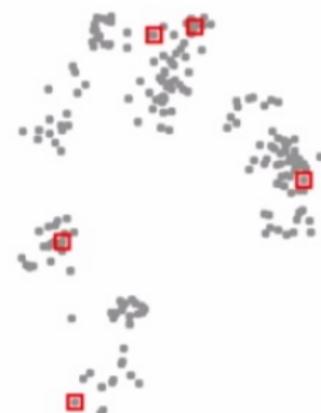
K-Means.



Unsupervised Learning

K-Means.

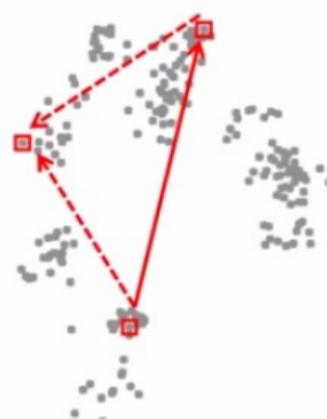
- Initialization methods
 - Random:
 - May choose nearby points.



Unsupervised Learning

K-Means.

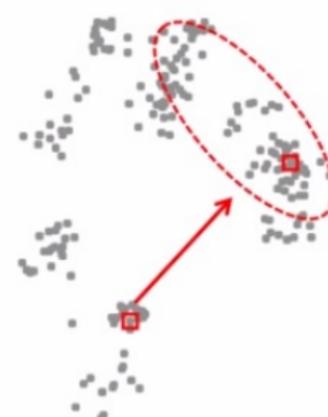
- Initialization methods
 - Distance-based:
 - May choose outliers, and does not provide many opportunities to optimization (not enough randomness).



Unsupervised Learning

K-Means.

- Initialization methods
 - Random + Distance-based (K-Means++):
 - Choose next points far, but randomly.



Unsupervised Learning

K-Means

- How to choose the number of clusters?

Unsupervised Learning

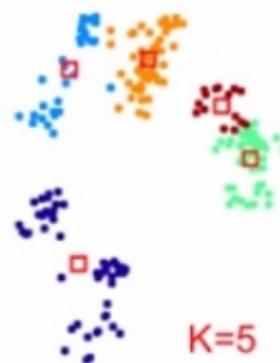
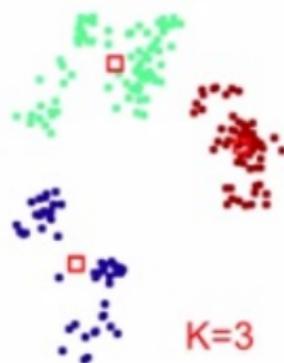
K-Means

- How to choose the number of clusters?
 - Use the loss function:
$$L = \sum_i ||x_i - \mu_{z_i}||^2$$
 - What will happen?

Unsupervised Learning

K-Means

- How to choose the number of clusters?
 - Use the loss function:
$$L = \sum_i ||x_i - \mu_{z_i}||^2$$
 - What will happen?
 - Obviously, more clusters minimize L .



Unsupervised Learning

K-Means

- A model complexity issue
 - If k is too small:
 - Any new point will be placed in the same group (there is no useful clustering).
 - If k is too large:
 - Any point is a group itself.

Unsupervised Learning

K-Means

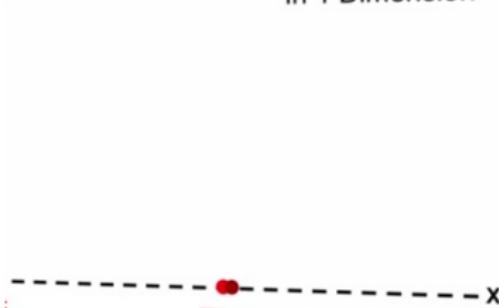
- A model complexity issue
 - If k is too small:
 - Any new point will be placed in the same group (there is no useful clustering).
 - If k is too large:
 - Any point is a group itself.
- Solution: penalize for complexity
 - More clusters increase the loss, if they do not help "enough"
 - Bayesian Information Criterion:
 - $C = \log\left(\frac{1}{n \times d} \sum_i \|x_i - \mu_{z_i}\|^2\right) + k \times \frac{\log n}{n}$

Unsupervised Learning

K-Means.

- Dimensionality plays a fundamental role also.
 - Distance and dimensionality.

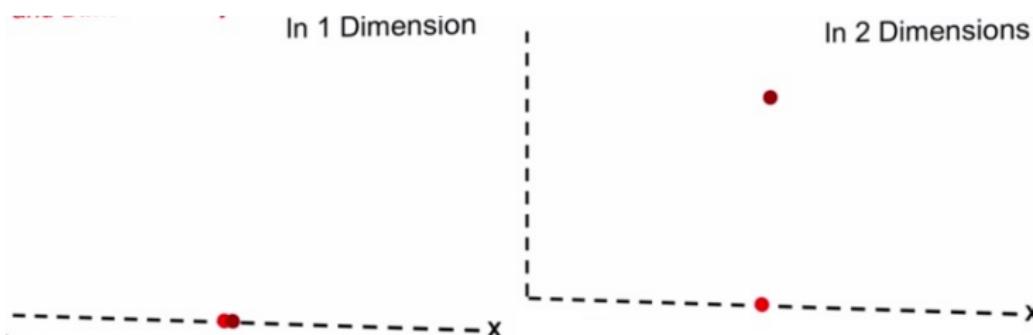
In 1 Dimension



Unsupervised Learning

K-Means.

- Dimensionality plays a fundamental role also.
 - Distance and dimensionality.



- Can we reduce dimensionality?
 - PCA, SVD etc.

Unsupervised Learning

K-Means.

- Problems with massive data
 - All distances have to be computed at every iteration

Unsupervised Learning

K-Means.

- Problems with massive data
 - All distances have to be computed at every iteration
 - Pre-clustering may save costly (or redundant) computations:
 - Canopy clustering.

Unsupervised Learning

Mixture models (density probability estimation).

- Hard clustering:
 - Clusters do not overlap (K-Means).
- Soft clustering:
 - Clusters may overlap.
 - There is a probability associated with the assignment.
 - Each cluster is a probability distribution:
 - Gaussian or multinomial.

Unsupervised Learning

Mixture models.

- Assume one dimension only.
- Assume two Gaussians only.

- Just calculate μ and σ^2 .

- $\mu_a = \frac{x_1 + x_2 + \dots + x_{n_a}}{n_a}$

- $\sigma^2 = \frac{(x_1 - \mu_a)^2 + (x_2 - \mu_a)^2 + \dots + (x_{n_a} - \mu_a)^2}{n_a}$

Unsupervised Learning

Mixture models.

- Assume one dimension only.
- Assume two Gaussians only.

- Just calculate μ and σ^2 .

- $\mu_a = \frac{x_1 + x_2 + \dots + x_{n_a}}{n_a}$

- $\sigma^2 = \frac{(x_1 - \mu_a)^2 + (x_2 - \mu_a)^2 + \dots + (x_{n_a} - \mu_a)^2}{n_a}$

- $p(x_i | a) = \frac{1}{2\pi\sigma_a^2} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$



Unsupervised Learning

Mixture models.

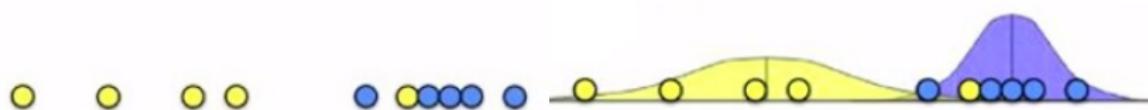
- Assume one dimension only.
- Assume two Gaussians only.

- Just calculate μ and σ^2 .

- $\mu_a = \frac{x_1 + x_2 + \dots + x_{n_a}}{n_a}$

- $\sigma^2 = \frac{(x_1 - \mu_a)^2 + (x_2 - \mu_a)^2 + \dots + (x_{n_a} - \mu_a)^2}{n_a}$

- $p(x_i | a) = \frac{1}{2\pi\sigma_a^2} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$



Unsupervised Learning

Mixture models.

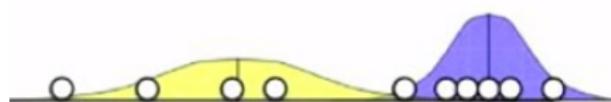
- What if we do not know the sources (blue or yellow)?
 - Can we fit the models now?



Unsupervised Learning

Mixture models.

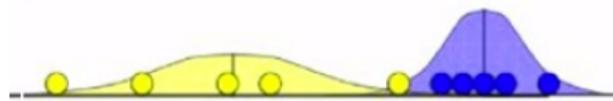
- What if we do not know the sources (blue or yellow)?
 - Only if we know the parameters of the Gaussians (μ and σ^2).



Unsupervised Learning

Mixture models.

- What if we do not know the sources (blue or yellow)?
 - We can (probabilistically) guess the color of the points.
 - $p(x_i|a) = \frac{1}{2\pi\sigma_a^2} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$
 - $p(a|x_i) = \frac{p(x_i|a) \times p(a)}{p(x_i|a) \times p(a) + p(x_i|b) \times p(b)}$



Unsupervised Learning

Mixture models.

- Chicken and egg problem.
 - Need (μ_a, σ_a^2) and (μ_b, σ_b^2) to guess the source of points.
 - Need to know the sources to estimate (μ_a, σ_a^2) and (μ_b, σ_b^2) .
- Expectation-Maximization algorithm:
 - Start with randomly placed Gaussians.
 - Iterate:
 - Probabilistically guess the source for each x_i .
 - Adjust the Gaussians to fit points assigned to them.

Unsupervised Learning

Mixture models.

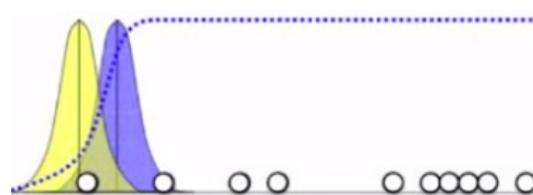


Unsupervised Learning

Mixture models.



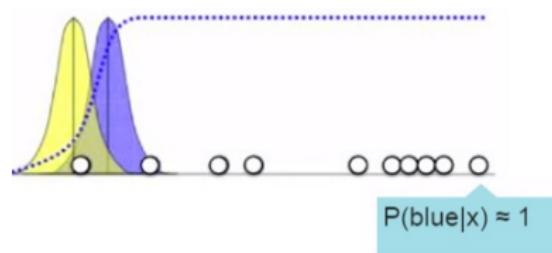
- Randomly place the Gaussians.



Unsupervised Learning

Mixture models.

- Guess the color of each point.
 - $p(x_i|a) = \frac{1}{2\pi\sigma_a^2} \exp\left(-\frac{(x_i - \mu_a)^2}{2\sigma_a^2}\right)$
 - $p(a|x_i) = \frac{p(x_i|a) \times p(a)}{p(x_i|a) \times p(a) + p(x_i|b) \times p(b)}$

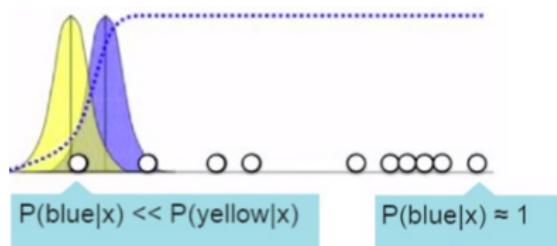


- Small $p(x_i|a)$ and small $p(x_i|b)$, but high $p(a|x_i)$.
 - It is not likely to be either blue or yellow, but it is much more unlikely to be yellow (Bayes rule).

Unsupervised Learning

Mixture models.

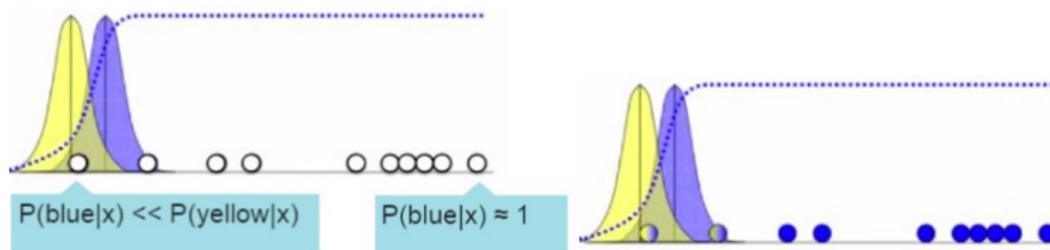
- Guess the color of each point.



Unsupervised Learning

Mixture models.

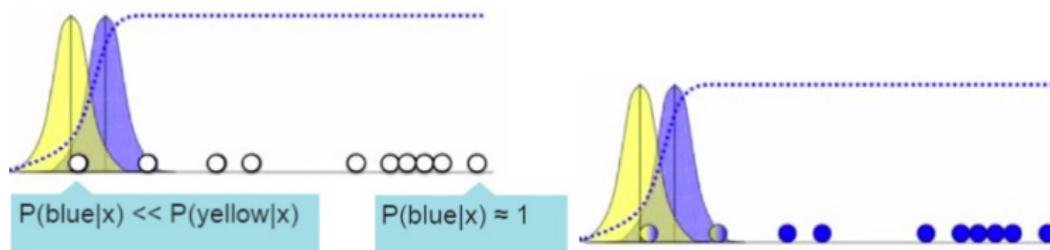
- Guess the color of each point.



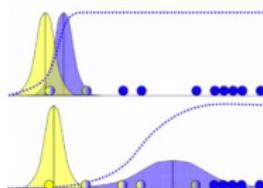
Unsupervised Learning

Mixture models.

- Guess the color of each point.



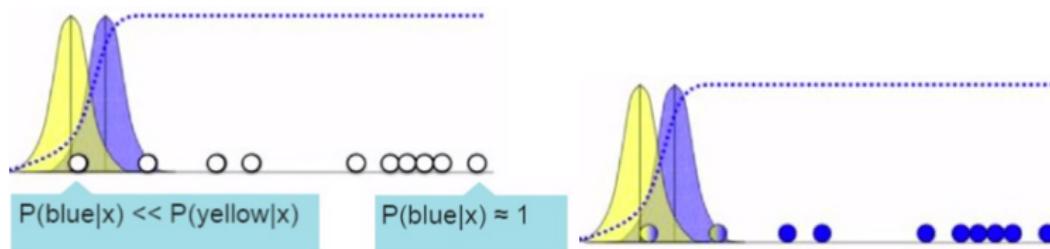
- Now, re-estimate.



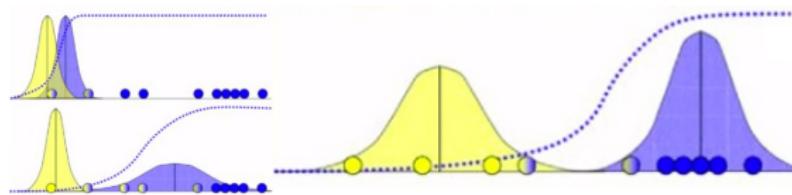
Unsupervised Learning

Mixture models.

- Guess the color of each point.

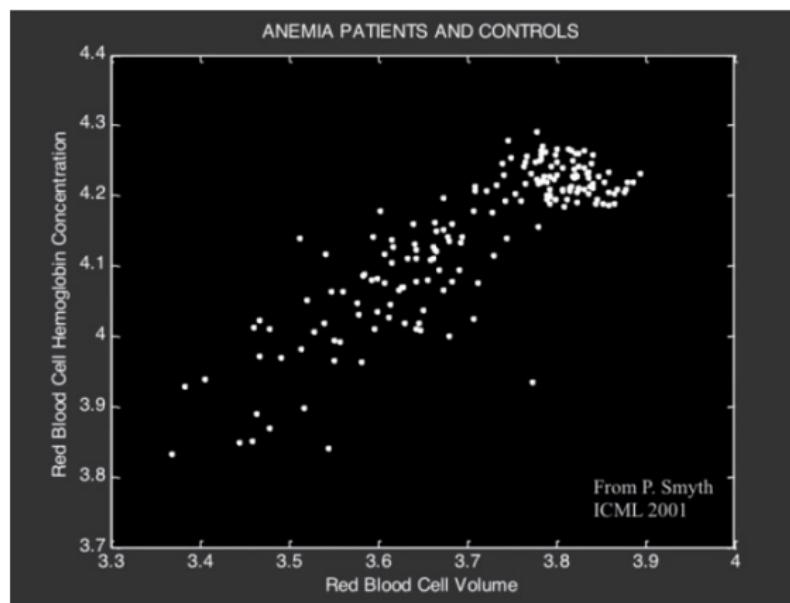


- Now, re-estimate.



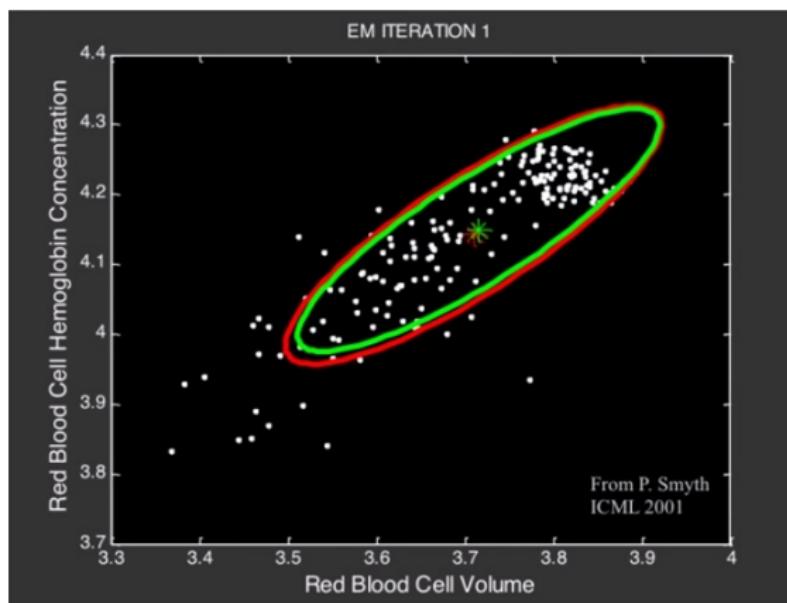
Unsupervised Learning

Mixture models.



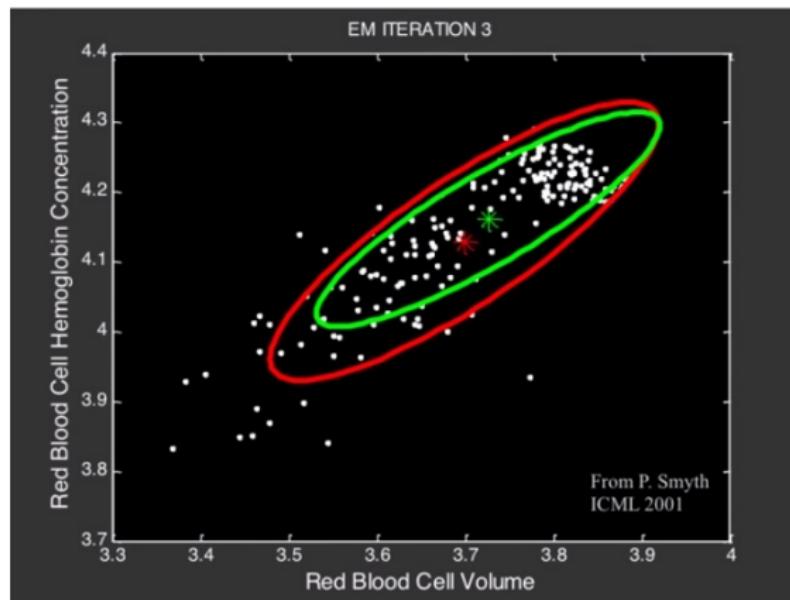
Unsupervised Learning

Mixture models.



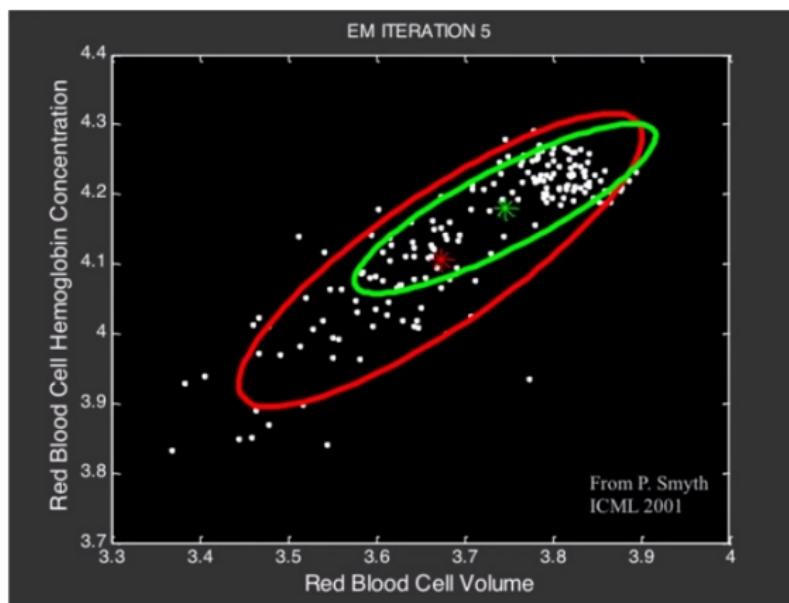
Unsupervised Learning

Mixture models.



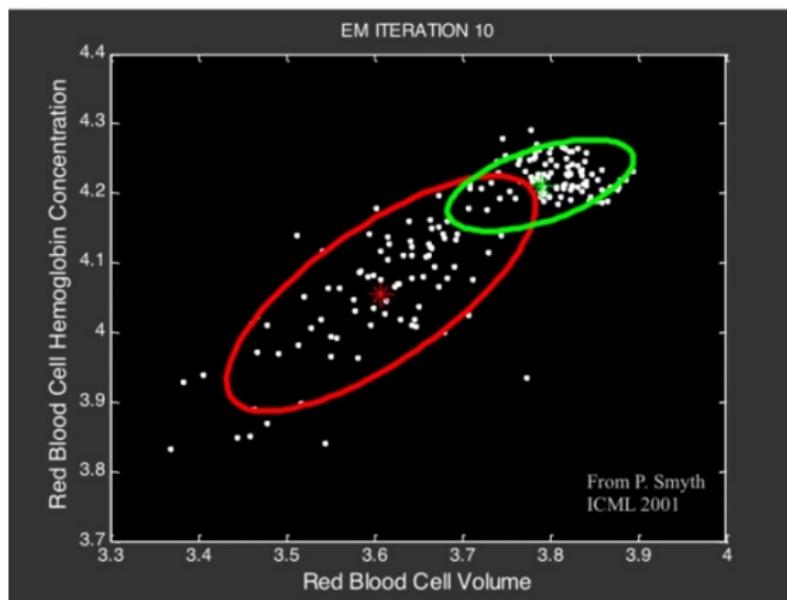
Unsupervised Learning

Mixture models.



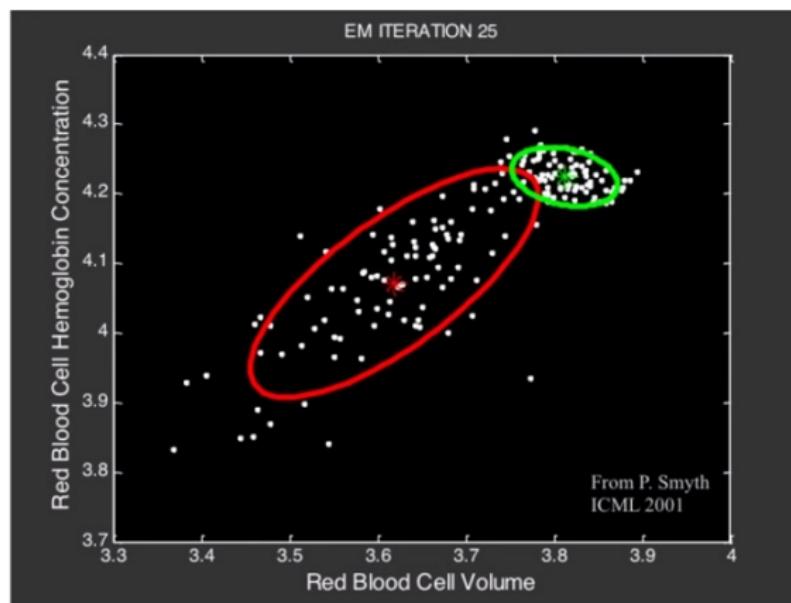
Unsupervised Learning

Mixture models.



Unsupervised Learning

Mixture models.



Semi-supervised Learning

We saw supervised and unsupervised algorithms.

- Semi-supervised learning:

Semi-supervised Learning

We saw supervised and unsupervised algorithms.

- Semi-supervised learning:
 - Class of supervised learning.
 - Must make use of both labeled and unlabeled data.
 - Small amounts of labeled data.
 - Large amounts of unlabeled data.

Semi-supervised Learning

We saw supervised and unsupervised algorithms.

- Semi-supervised learning:
 - Class of supervised learning.
 - Must make use of both labeled and unlabeled data.
 - Small amounts of labeled data.
 - Large amounts of unlabeled data.
 - $\{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k), x_{k+1}, x_{k+2}, \dots, x_m\}$
 - Considerable gain in effectiveness
 - Labeling cost vs effectiveness.

Semi-supervised Learning

Can one hope to have a more accurate predictive model by taking into account the unlabeled data?

- The distribution of examples, which the unlabeled data will help elucidate, be relevant for the classification problem.
 - Knowledge on $p(x)$ that one gains through the unlabeled data has to carry information that is useful in the inference of $p(y|x)$.
 - It might even happen that using unlabeled data degrades the prediction accuracy by misguiding the inference.

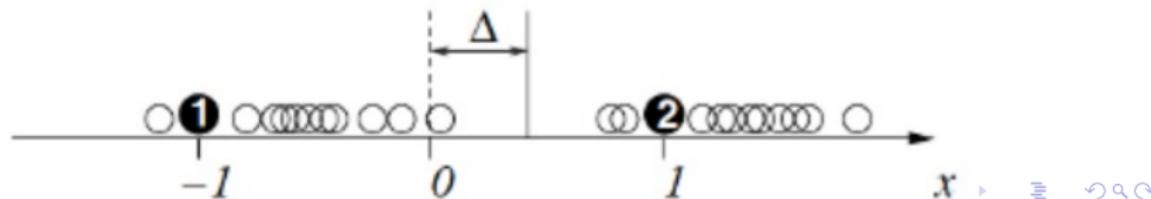
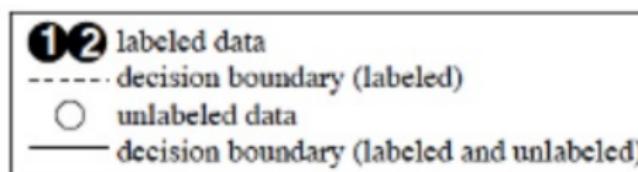
Semi-supervised Learning

Can one hope to have a more accurate predictive model by taking into account the unlabeled data?

- The distribution of examples, which the unlabeled data will help elucidate, be relevant for the classification problem.
 - Knowledge on $p(x)$ that one gains through the unlabeled data has to carry information that is useful in the inference of $p(y|x)$.
 - It might even happen that using unlabeled data degrades the prediction accuracy by misguiding the inference.
- Main assumption of semi-supervised learning:
 - If two inputs x_1 and x_2 are close in the common space, then so should be the corresponding outputs y_1 and y_2 .
 - Important variation: if two inputs x_1 and x_2 in a high-density region of the space are close, then so should be the corresponding outputs y_1 and y_2 .

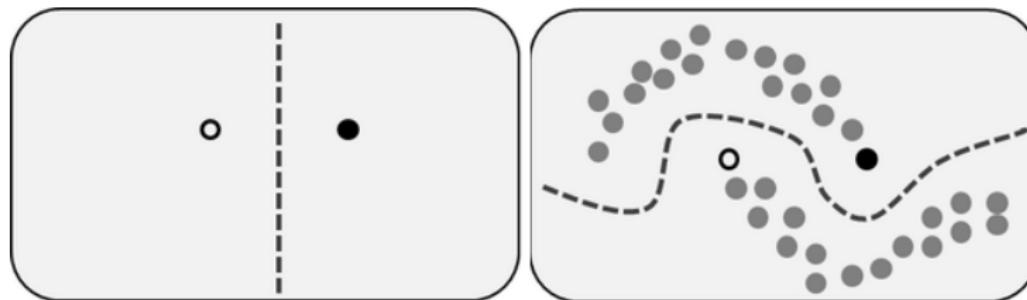
Semi-supervised Learning

- Cluster assumption:
 - If points are in the same cluster, they are likely to be of the same class.
 - The decision boundary should lie in a low-density region.
- These assumptions inspired different algorithms for semi-supervised learning.

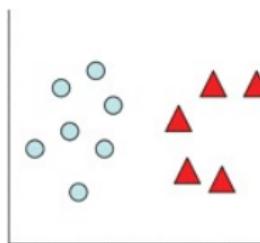


Semi-supervised Learning

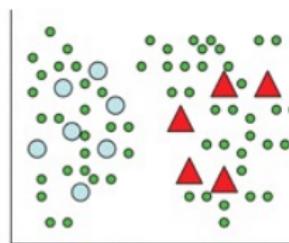
- Cluster assumption:
 - If points are in the same cluster, they are likely to be of the same class.
 - The decision boundary should lie in a low-density region.
- These assumptions inspired different algorithms for semi-supervised learning.



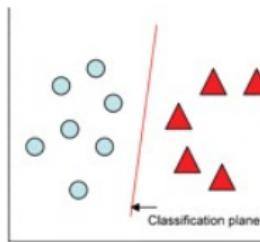
Semi-supervised Learning



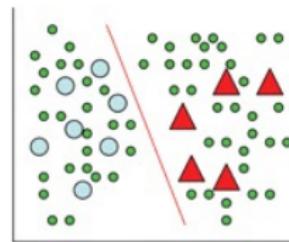
Labeled Data
(a)



Labeled and Unlabeled Data
(b)

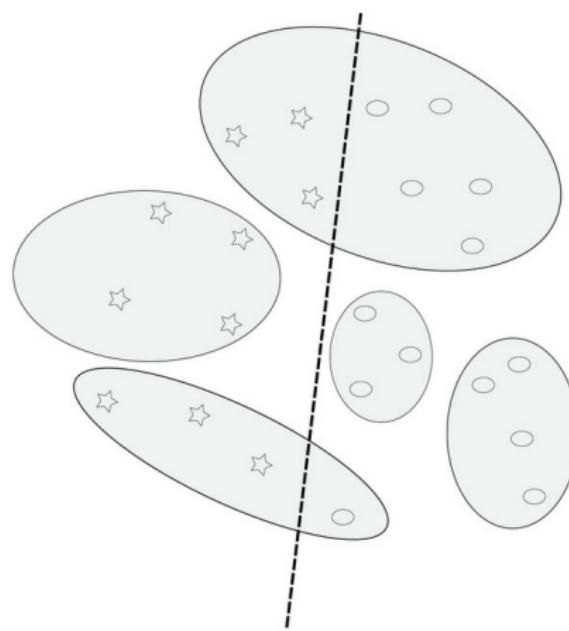


Supervised Learning
(c)



Semi-Supervised Learning
(d)

Semi-supervised Learning



Semi-supervised Learning

A very simple algorithm.

- Use Naive Bayes on labeled data and then apply

Semi-supervised Learning

A very simple algorithm.

- Use Naive Bayes on labeled data and then apply **Expectation-Maximization**.
use
 - ① Build Naive Bayes model on labeled data.
 - ② Label unlabeled data based on class probabilities (Expectation step).
 - ③ Build new model using Naive Bayes (Maximization step).
 - ④ Repeat 2 and 3.

Semi-supervised Learning

Another simple algorithm.

- Self-training.



Semi-supervised Learning

Another simple algorithm.

- Self-training.



Semi-supervised Learning

Positive and unlabeled classification.

- Only positive labels are shown.

Semi-supervised Learning

Positive and unlabeled classification.

- Only positive labels are shown.
 - 1 Treat unlabeled data as negative.
 - 2 Build a classifier.
 - 3 Relabel the data.
 - 4 Repeat until convergence.

Active Learning

Active learning is not a new teaching idea.

- It goes back at least as far as Socrates.
- Learning is naturally an active process.
 - It involves putting students in situations which compel them to read, speak, listen, think deeply, and write.
 - Active learning puts the responsibility of organizing what is to be learned in the hands of the learners themselves.

Active Learning

Active learning is not a new teaching idea.

- It goes back at least as far as Socrates.
- Learning is naturally an active process.
 - It involves putting students in situations which compel them to read, speak, listen, think deeply, and write.
 - Active learning puts the responsibility of organizing what is to be learned in the hands of the learners themselves.
- The most common active learning strategy is to let students ask questions.

Active Learning

It is a special case of supervised learning.

- Unlabeled data is abundant but manually labeling is expensive.
 - A learning algorithm is able to interactively query the training source to obtain the desired outputs at new examples.
 - Iteratively query for informative labels.
 - Since the learning algorithm chooses the examples, the number of examples to learn a concept can often be much lower than the number required in normal supervised learning.
- In statistics literature, active learning is also known as optimal experimental design.

Active Learning

Active learning setting.

- Loop:
 - D_k : labeled examples.
 - D_u : unlabeled data.
 - D_c : a subset of D_u that is chosen to be labeled.

Active Learning

Active learning setting.

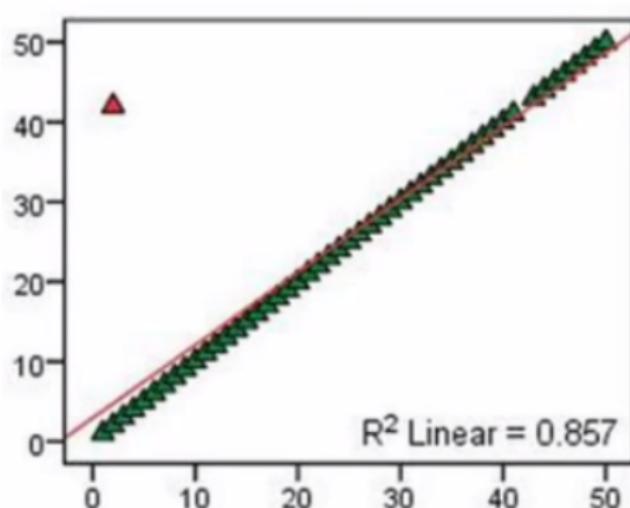
- Loop:
 - D_k : labeled examples.
 - D_u : unlabeled data.
 - D_c : a subset of D_u that is chosen to be labeled.
- Active learning algorithm differ on how they choose D_c .

Active Learning

Efficient search through hypothesis space.

- Simple case:
 - Points lie on the real line, and hypotheses are thresholds.
 - How many labels are needed to find a hypothesis h whose error is at most ϵ ?
 - PAC: we need approximately $\frac{1}{1/\epsilon}$ random labeled examples
 - Active learning: from $\frac{1}{\epsilon}$ to $\log \frac{1}{\epsilon}$.
 - For instance, if supervised learning requires a million labels, active learning requires just $\log 1,000,000 \approx 20$!
- Thus active learning is a technique to create (optimal) training sets.

Active Learning



Active Learning

Query strategies.

- Which examples should be selected?

Active Learning

Query strategies.

- Which examples should be selected?
 - Uncertainty sampling:
 - Label the examples for which the current model is least certain as to what the correct output should be.

Active Learning

Query strategies.

- Which examples should be selected?
 - Uncertainty sampling:
 - Label the examples for which the current model is least certain as to what the correct output should be.
 - Query by committee:
 - A variety of models are trained on the current labeled data, and vote on the output for unlabeled data
 - Label the examples for which the committee disagrees the most

Active Learning

Query strategies.

- Which examples should be selected?
 - Uncertainty sampling:
 - Label the examples for which the current model is least certain as to what the correct output should be.
 - Query by committee:
 - A variety of models are trained on the current labeled data, and vote on the output for unlabeled data
 - Label the examples for which the committee disagrees the most
 - Expected model change:
 - Label the examples that would change most the current model.

Active Learning

Query strategies.

- Which examples should be selected?
 - Uncertainty sampling:
 - Label the examples for which the current model is least certain as to what the correct output should be.
 - Query by committee:
 - A variety of models are trained on the current labeled data, and vote on the output for unlabeled data
 - Label the examples for which the committee disagrees the most
 - Expected model change:
 - Label the examples that would change most the current model.
 - Expected error reduction:
 - Label the examples that would reduce most the model's generalization error

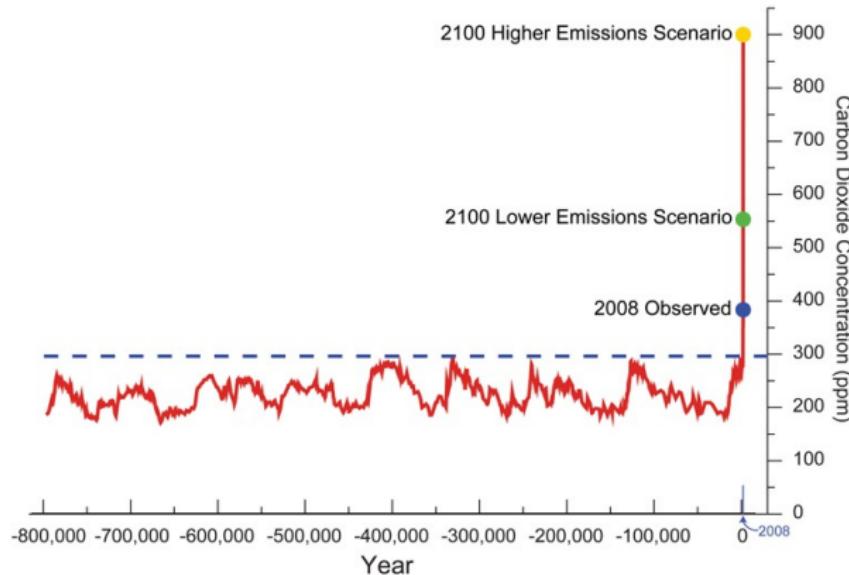
Hidden Markov Models

Canonical probabilistic model for temporal or sequential data.

- Markov assumption:
 - The future is independent of the past, given the present.
 - Everything you need to know is already encoded in the current state.
 - Sequential data:
 - Language (Mark V. Shaney)
 - Music
 - Weather
 - Finance

Hidden Markov Models

- Model sequential data with randomness + patterns.



Hidden Markov Models

- $D = \{x_1, x_2, \dots, x_n\}$ such that $t(x_k) < t(x_{k+1})$
 - Probabilistic model:
 - Independent and identically distributed?

Hidden Markov Models

- $D = \{x_1, x_2, \dots, x_n\}$ such that $t(x_k) < t(x_{k+1})$
 - Probabilistic model:
 - Independent and identically distributed? No!
 - Everything depends on everything else?

Hidden Markov Models

- $D = \{x_1, x_2, \dots, x_n\}$ such that $t(x_k) < t(x_{k+1})$
 - Probabilistic model:
 - Independent and identically distributed? No!
 - Everything depends on everything else? Untractable.

Hidden Markov Models

- $D = \{x_1, x_2, \dots, x_n\}$ such that $t(x_k) < t(x_{k+1})$
 - Probabilistic model:
 - Independent and identically distributed? No!
 - Everything depends on everything else? Untractable.
 - The recent past tells more than the distant past.
 - x_t depends on $x_{t-1}, x_{t-2}, \dots, x_{t-m}$
 - Simple case: $m = 1$
 - Markov property:
 - Each time step only depends on the previous

Hidden Markov Models

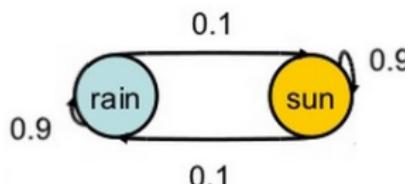
Markov chains.

- $X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_n$
- Simplifying assumptions:
 - Discrete time
 - Discrete space
- A (first-order) Markov chain respects
 - $p(x_t|x_1, x_2, \dots, x_{t-1}) = p(x_t|x_{t-1})$
- Value of x_t is called the **state**

Hidden Markov Models

Example of Markov chains.

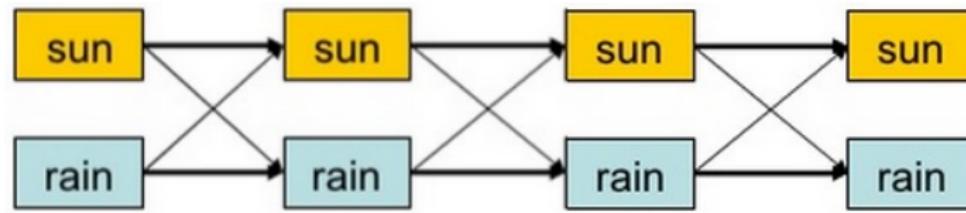
- Weather:
 - States: $X = \{\text{rain}, \text{sun}\}$
 - Transitions



- Initial distribution: sun=1.0
 - What is the probability distribution after one step?
 - $p(x_2 = \text{sun}) = p(x_2 = \text{sun}|x_1 = \text{sun}) \times p(x_1 = \text{sun}) + p(x_2 = \text{sun}|x_1 = \text{rain}) \times p(x_1 = \text{rain}) = 0.9 \times 1.0 + 0.1 \times 0.0 = 0.9$

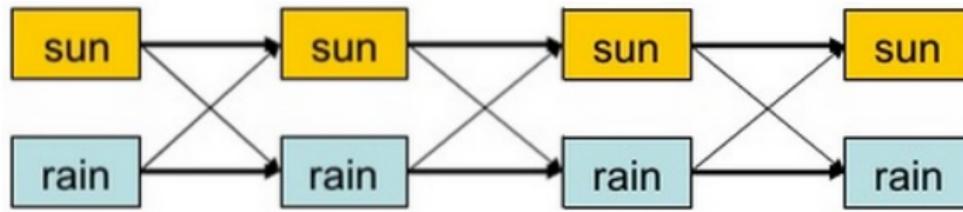
Hidden Markov Models

What is $p(x)$ on some day t ?



Hidden Markov Models

What is $p(x)$ on some day t ?



Mini-forward algorithm.

- $p(x_t) = \sum_{x_{t-1}} p(x_t | x_{t-1}) \times p(x_{t-1})$
- $p(x_1)$ is known.

Hidden Markov Models

Initial observation of sun or rain.

$$\begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix} \quad \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix} \quad \begin{pmatrix} 0.82 \\ 0.18 \end{pmatrix} \xrightarrow{\hspace{1cm}} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\text{P}(X_1) \quad \text{P}(X_2) \quad \text{P}(X_3) \quad \text{P}(X_\infty)$$

$$\begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix} \quad \begin{pmatrix} 0.1 \\ 0.9 \end{pmatrix} \quad \begin{pmatrix} 0.18 \\ 0.82 \end{pmatrix} \xrightarrow{\hspace{1cm}} \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}$$

$$\text{P}(X_1) \quad \text{P}(X_2) \quad \text{P}(X_3) \quad \text{P}(X_\infty)$$

- What happens if we simulate the chain long enough?

Hidden Markov Models

Initial observation of sun or rain.

$$\begin{array}{c} \left\langle \begin{array}{c} 1.0 \\ 0.0 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.9 \\ 0.1 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.82 \\ 0.18 \end{array} \right\rangle \quad \xrightarrow{\hspace{1cm}} \quad \left\langle \begin{array}{c} 0.5 \\ 0.5 \end{array} \right\rangle \\ P(X_1) \qquad P(X_2) \qquad P(X_3) \qquad P(X_\infty) \end{array}$$

$$\begin{array}{c} \left\langle \begin{array}{c} 0.0 \\ 1.0 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.1 \\ 0.9 \end{array} \right\rangle \quad \left\langle \begin{array}{c} 0.18 \\ 0.82 \end{array} \right\rangle \quad \xrightarrow{\hspace{1cm}} \quad \left\langle \begin{array}{c} 0.5 \\ 0.5 \end{array} \right\rangle \\ P(X_1) \qquad P(X_2) \qquad P(X_3) \qquad P(X_\infty) \end{array}$$

- What happens if we simulate the chain long enough?
 - Uncertainty accumulates
 - Eventually, we have no idea what the state is.
 - Usually, a chain can only predict a short time out.

Hidden Markov Models

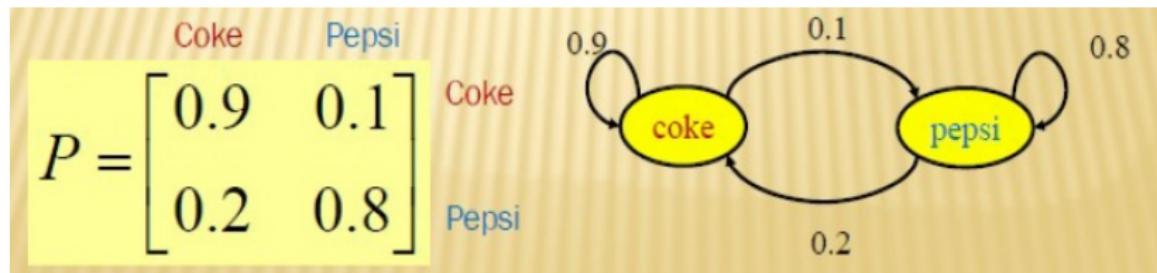
Coke vs. Pepsi.

- Given that the last purchase was coke, there is 90% chance of the next purchase to be coke.
- Given that the last purchase was pepsi, there is 80% chance of the next purchase to be pepsi.

Hidden Markov Models

Coke vs. Pepsi.

- Given that the last purchase was coke, there is 90% chance of the next purchase to be coke.
- Given that the last purchase was pepsi, there is 80% chance of the next purchase to be pepsi.



Hidden Markov Models

Coke vs. Pepsi.

- Given that you currently buy pepsi:
 - What are the odds of buying coke after the next two purchases?
- States:
 - C_n and P_n : %buyers of coke and pepsi.
- Model:
 - $C_{n+1} = 0.9 \times C_n + 0.2 \times P_n$
 - $P_{n+1} = 0.1 \times C_n + 0.8 \times P_n$
- Goal:
 - P_2 and C_2
- Solution:
 - $C_1 = 0.9 \times C_0 + 0.2 \times P_0 = 0.20,$
 $P_1 = 0.1 \times C_0 + 0.8 \times P_0 = 0.80$
 - $C_2 = 0.9 \times C_1 + 0.2 \times P_1 = 0.34,$
 $P_2 = 0.1 \times C_1 + 0.8 \times P_1 = 0.66$

Hidden Markov Models

$$P^3 = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix} \begin{bmatrix} 0.83 & 0.17 \\ 0.34 & 0.66 \end{bmatrix} = \begin{bmatrix} 0.781 & 0.219 \\ 0.438 & 0.562 \end{bmatrix}$$

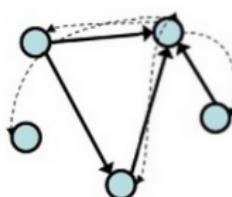
Hidden Markov Models

Stationary distribution.

- Stochastic process:
 - Possibilities follow a probability distribution
 - Given the same start, there are many possibilities of ending.
 - But some are more likely than others.
- For most chains, the distribution we end up is independent of the initial distribution.
 - Called the **stationary distribution**

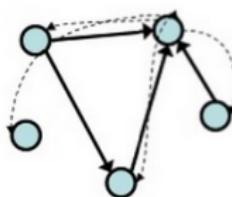
Hidden Markov Models

PageRank.



Hidden Markov Models

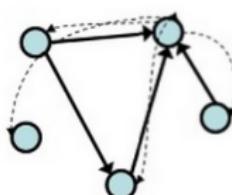
PageRank.



- Each web page is a state.
- Initial distribution: uniform over the pages
- Transitions:
 - With probability c , jump to a random page (dotted lines)
 - With probability $1 - c$, follow an outlink (solid lines)

Hidden Markov Models

PageRank.

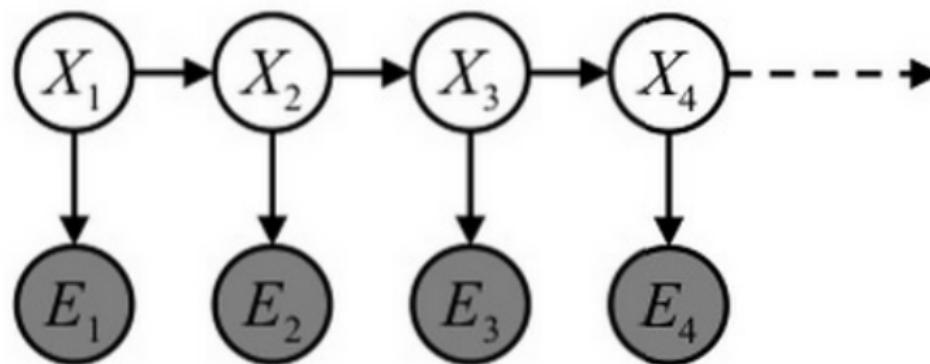


- Each web page is a state.
- Initial distribution: uniform over the pages
- Transitions:
 - With probability c , jump to a random page (dotted lines)
 - With probability $1 - c$, follow an outlink (solid lines)
- Stationary distribution:
 - Will spend more time on highly reachable pages
 - Google 1.0

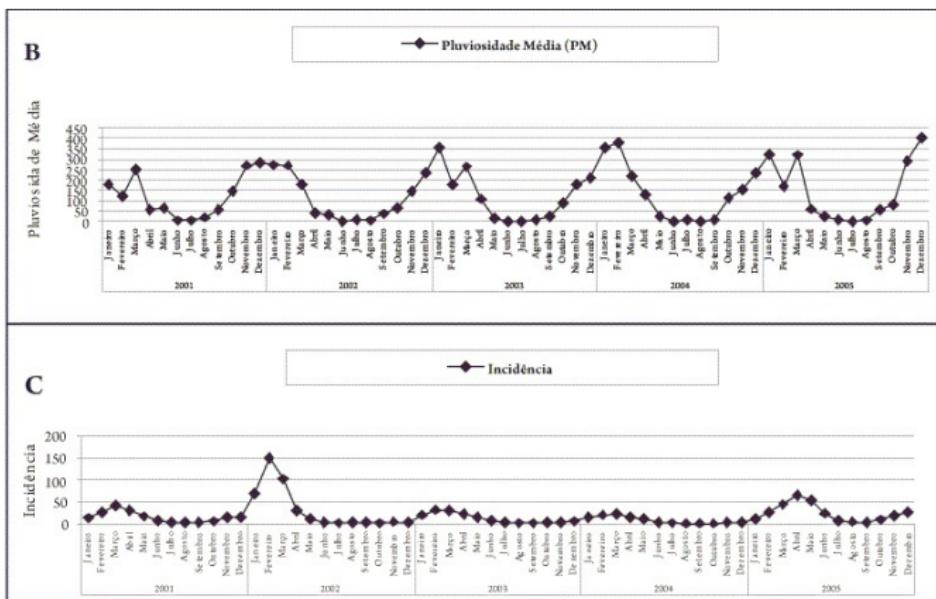
Hidden Markov Models

Cannot expect to perfectly observe the complete state of the system.

- There is hidden information.
 - There are hidden states we cannot observe.
 - Model hidden states with hidden/latent variables.
- This model is called Hidden Markov Models.



Hidden Markov Models



Hidden Markov Models

Example.

- Two persons:
 - Bob and Alice
 - Live in distant cities.
 - Alice would like to guess if today is raining or sunny on Bob's city.
 - Alice knows that, historically, there is a 60% chance of raining in Bob's city.
 - Also Bob only does three activities:
 - Walk on the park
 - Buy food
 - Clean his house.

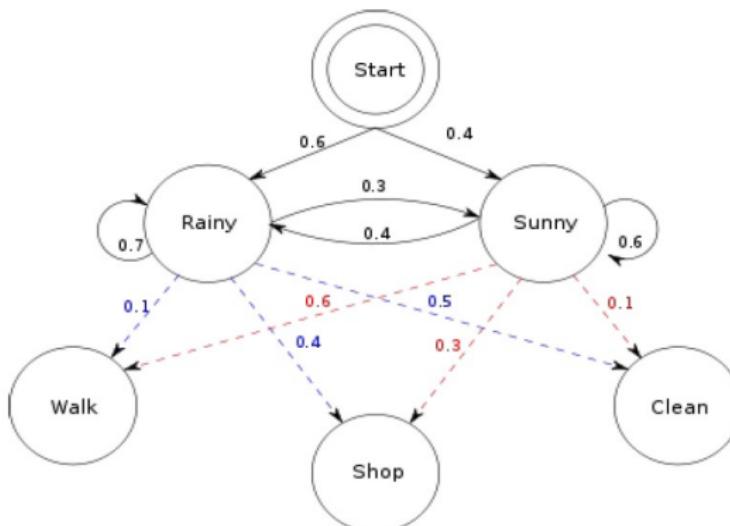
Hidden Markov Models

Example.

- Bob told Alice by phone which was the activity today.
 - But, does not tell anything about the weather.
- Alice knows the climate trends in Bob's city:
 - Initial probabilities
 - Transition probabilities
- Also, Alice knows the likelihood of each activity given the weather:
 - Rain \rightarrow 50% chance cleaning the house
 - Sun \rightarrow 60% chance of going walk

Hidden Markov Models

- Parameters of a HMM:
 - Initial probabilities
 - Transition probabilities
 - Emission probabilities



Hidden Markov Models

Example.

- Bob told Alice that:
 - First day: walk on the park
 - Second day: buy food
 - Third day: clean his house
- Alice wants to know which is the most likely sequence of rain/sun.

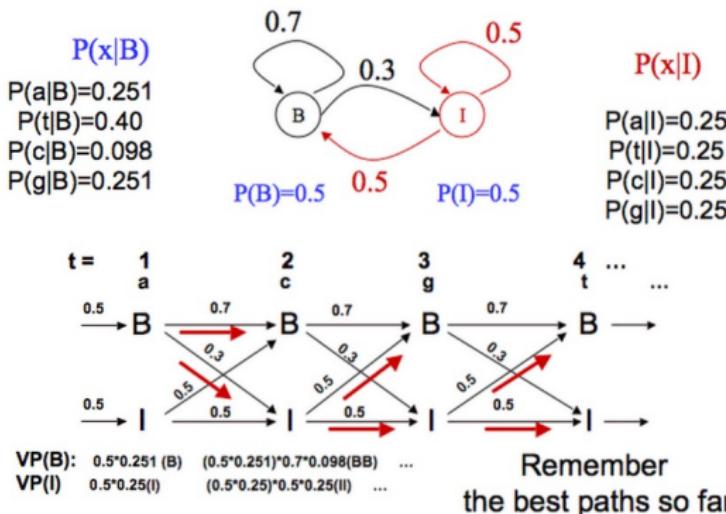
Hidden Markov Models

Example.

- Bob told Alice that:
 - First day: walk on the park
 - Second day: buy food
 - Third day: clean his house
- Alice wants to know which is the most likely sequence of rain/sun.
 - Evaluate all possible sequences
 - Choose the most likely.
 - This is exponential.

Hidden Markov Models

Viterbi algorithm.



Hidden Markov Models

Inference using HMMs.

- Forward/Backward algorithm
 - Dynamic programming approach.
 - Compute $p(e_k | X)$
 - Forward:
 - Compute $p(e_k, x_1, x_2, \dots, x_k)$
 - Backward:
 - Compute $p(x_{k+1}, x_{k+2}, \dots, x_n | e_k)$
 - $p(e_k | X) \approx p(x_{k+1:n} | e_k, x_{1:k}) \times p(e_k, x_{1:k})$
- Useful for sampling e 's given x 's

Hidden Markov Models

Forward algorithm.

- Compute $p(e_k, x_{1:k})$
- $$p(e_k, x_{1:k}) = \sum p(e_k, e_{k-1}, x_{1:k}) = \\ \sum p(x_k|e_k, e_{k-1}, x_{1:k-1}) \times p(e_k|e_{k-1}, x_{1:k-1}) \times p(z_{k-1}, x_{1:k-1})$$

Hidden Markov Models

Forward algorithm.

- Compute $p(e_k, x_{1:k})$
 - $p(e_k, x_{1:k}) = \sum p(e_k, e_{k-1}, x_{1:k}) =$
 $\sum p(x_k|e_k, e_{k-1}, x_{1:k-1}) \times p(e_k|e_{k-1}, x_{1:k-1}) \times p(z_{k-1}, x_{1:k-1})$
 - $\sum p(x_k|e_k) \times p(e_k|e_{k-1}) \times p(z_{k-1}, x_{1:k-1})$
 - The last term is solved by variable elimination using dynamic programming (reuse earlier computations).

Hidden Markov Models

Backward algorithm.

- Compute $p(x_{k+1:n}|e_k)$
 - $p(x_{k+1:n}|e_k) = \sum p(x_{k+1:n}, e_{k+1}|e_k) =$
 $\sum p(x_{k+2:n}|e_{k+1}, e_k, x_{k+1}) \times p(x_{k+1}|e_{k+1}, e_k) \times p(e_{k+1}|e_k)$

Hidden Markov Models

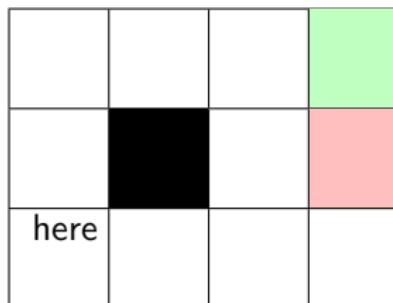
Backward algorithm.

- Compute $p(x_{k+1:n}|e_k)$
 - $p(x_{k+1:n}|e_k) = \sum p(x_{k+1:n}, e_{k+1}|e_k) =$
 $\sum p(x_{k+2:n}|e_{k+1}, e_k, x_{k+1}) \times p(x_{k+1}|e_{k+1}, e_k) \times p(e_{k+1}|e_k)$
 - $\sum p(x_{k+2:n}|e_{k+1}) \times p(x_{k+1}|e_{k+1}) \times p(e_{k+1}|e_k)$
 - The first term is solved by variable elimination using dynamic programming (reuse earlier computations).

Decision Making and Reinforcement Learning

Markov Decision Process.

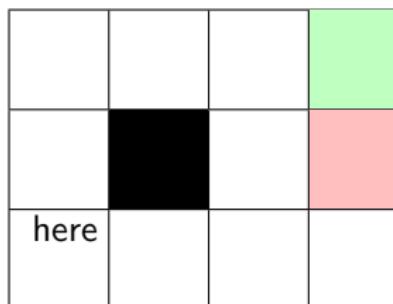
- Assume a grid world.
 - Each cell is called a state
 - Actions: you can go up, down, left and right



Decision Making and Reinforcement Learning

Markov Decision Process.

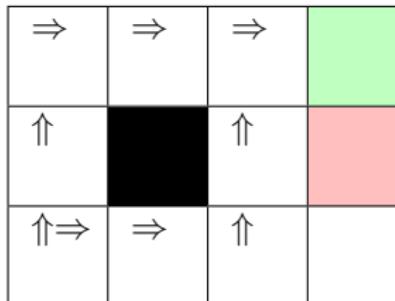
- Assume a grid world.
 - Each cell is called a state
 - Actions: you can go up, down, left and right



- What is the shortest sequence getting you from the start to the goal?

Decision Making and Reinforcement Learning

- Usually we have multiple answers, multiple minima.

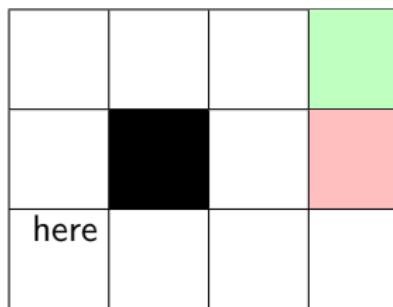


Decision Making and Reinforcement Learning

- Every time you took an action it did exactly what was expected.
 - Let's include some uncertainty in the process (i.e., stochasticity)

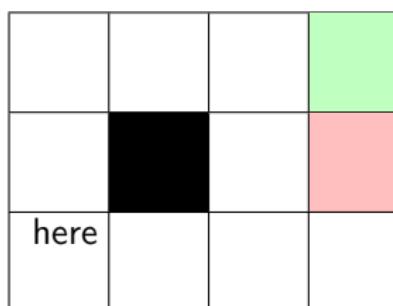
Decision Making and Reinforcement Learning

- Every time you took an action it did exactly the expected.
 - Lets include some uncertainty in the process (i.e., stochasticity)
- Action executes with a probability:
 - Correctly: 0.8
 - Move at right angle: 0.1 and 0.1.



Decision Making and Reinforcement Learning

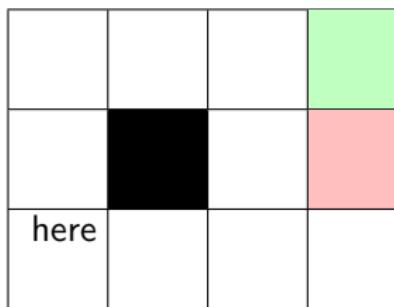
- Every time you took an action it did exactly the expected.
 - Lets include some uncertainty in the process (i.e., stochasticity)
- Action executes with a probability:
 - Correctly: 0.8
 - Move at right angle: 0.1 and 0.1.



- What is the reliability of UURRR?

Decision Making and Reinforcement Learning

- Every time you took an action it did exactly the expected.
 - Lets include some uncertainty in the process (i.e., stochasticity)
- Action executes with a probability:
 - Correctly: 0.8
 - Move at right angle: 0.1 and 0.1.



- What is the reliability of UURRR? $0.8^5 + 0.1^4 \times 0.8 = 0.32776$

Decision Making and Reinforcement Learning

A MDP is defined as:

- States: S
 - Our world has 12 states.
- Model: $T(s, a, s')$

Decision Making and Reinforcement Learning

A MDP is defined as:

- States: S
 - Our world has 12 states.
- Model: $T(s, a, s') \rightarrow p(s'|s, a)$
 - Markov property: only the present matters.
 - The current state encodes information from previous states.
 - There is uncertainty in the process but the probabilities are fixed.
- Actions: $A(s)$
 - up, down, right, left (all you can do in a particular state)
- Reward: $R(s)$

Decision Making and Reinforcement Learning

A MDP is defined as:

- States: S
 - Our world has 12 states.
- Model: $T(s, a, s') \rightarrow p(s'|s, a)$
 - Markov property: only the present matters.
 - The current state encodes information from previous states.
 - There is uncertainty in the process but the probabilities are fixed.
- Actions: $A(s)$
 - up, down, right, left (all you can do in a particular state)
- Reward: $R(s)$ (something you get by being in a state)

Decision Making and Reinforcement Learning

A MDP is defined as:

- States: S
 - Our world has 12 states.
- Model: $T(s, a, s') \rightarrow p(s'|s, a)$
 - Markov property: only the present matters.
 - The current state encodes information from previous states.
 - There is uncertainty in the process but the probabilities are fixed.
- Actions: $A(s)$
 - up, down, right, left (all you can do in a particular state)
- Reward: $R(s)$ (something you get by being in a state)
- Policy: $\pi(s) \rightarrow a$
 - π^* is the optimal policy.

Decision Making and Reinforcement Learning

Policy.

- $\pi(s) \rightarrow a$
 - A function that takes a state and returns an action.
 - It is a command, $\pi(\text{start}) \rightarrow \text{up}$.
 - You always know your state and the reward.
- π^* is the optimal policy
 - The best sequence of actions

Decision Making and Reinforcement Learning

Policy.

- $\pi(s) \rightarrow a$
 - A function that takes a state and returns an action.
 - It is a command, $\pi(\text{start}) \rightarrow \text{up}$.
 - You always know your state and the reward.
- π^* is the optimal policy
 - The best sequence of actions
 - The sequence of actions that maximizes the long term expected reward.

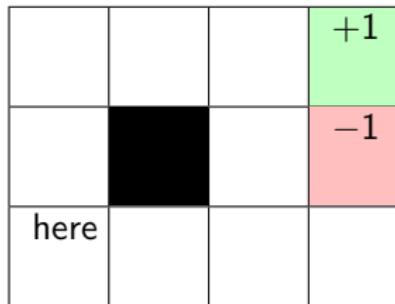
Decision Making and Reinforcement Learning

Policy.

- $\pi(s) \rightarrow a$
 - A function that takes a state and returns an action.
 - It is a command, $\pi(\text{start}) \rightarrow \text{up}$.
 - You always know your state and the reward.
- π^* is the optimal policy
 - The best sequence of actions
 - The sequence of actions that maximizes the long term expected reward.
- Supervised learning:
 - Examples: $(s_1, a_1), (s_2, a_2), \dots, (s_n, a_n)$
 - a is the correct action to take.
- Reinforcement learning:
 - Sequence: $(s_1, a_1, r_1), (s_2, a_2, r_2), \dots, (s_n, a_n, r_n)$
 - Being in s and taking a you receive r .

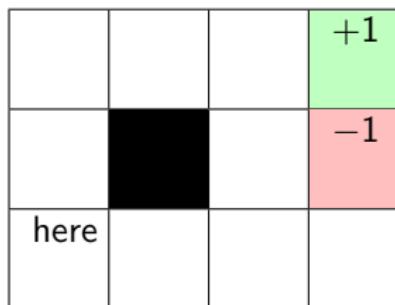
Decision Making and Reinforcement Learning

Rewards.



- Delayed reward.
 - Sequence of actions and minor changes matter a lot.
 - This is the difference from supervised learning.
 - Playing chess.
 - An action that seems to be good now may be not good in the long term.
 - This is called the **temporal credit assignment** problem.

Decision Making and Reinforcement Learning



- $R(s) = -0.04$, except from green and red states.
 - Walking on a beach with hot sand.
 - Ocean is the green state.
 - Jellyfish is the red state.

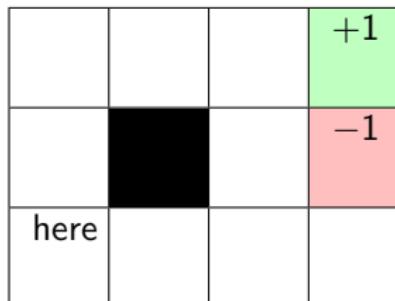
Decision Making and Reinforcement Learning

\Rightarrow	\Rightarrow	\Rightarrow	+1
\uparrow		\uparrow	-1
\uparrow	\Leftarrow	\Leftarrow	\Leftarrow

- $R(s) = -0.04$, except from green and red states.
 - Walking on a beach with hot sand.
 - Ocean is the green state.
 - Jellyfish is the red state.

Decision Making and Reinforcement Learning

- $R(s) = -2$



Decision Making and Reinforcement Learning

- $R(s) = -2$

\Rightarrow	\Rightarrow	\Rightarrow	+1
\uparrow		\Rightarrow	-1
\Rightarrow	\Rightarrow	\uparrow	\uparrow

Decision Making and Reinforcement Learning

- Infinite horizons.

\Rightarrow	\Rightarrow	\Rightarrow	+1
\uparrow		\uparrow	-1
\uparrow	\Leftarrow	\Leftarrow	\Leftarrow

- Finite horizon:

- The action given a state may change depending on the horizon.

Decision Making and Reinforcement Learning

- Utility of sequences.
 - Assuming an infinite horizon.

$$\begin{aligned} & \text{if } U(s_0, s_1, s_2, \dots) > U(s'_0, s'_1, s'_2, \dots) \\ & \text{then } U(s_1, s_2, \dots) > U(s'_1, s'_2, \dots) \end{aligned} \quad (3)$$

- $U(s_0, s_1, s_2, \dots) = \sum_{t=0}^{\infty} R(s_t)$
 - This does not work!
 - $+1 \rightarrow +1 \rightarrow +1 \rightarrow +1 \rightarrow +1 \rightarrow +1 \rightarrow \dots$
 - $+1 \rightarrow +1 \rightarrow +2 \rightarrow +1 \rightarrow +1 \rightarrow +2 \rightarrow \dots$

Decision Making and Reinforcement Learning

- Discounted cumulative reward.

$$\begin{aligned} U(s_0, s_1, s_2, \dots) &= \sum_{t=0}^{\infty} \gamma^t \times R(s_t), \text{ with } 0 \leq \gamma < 1 \\ &\leq \sum_{t=0}^{\infty} \gamma^t \times R_{max} = \frac{R_{max}}{1 - \gamma} \end{aligned}$$

Decision Making and Reinforcement Learning

$$\begin{aligned}\sum \gamma^t R_{max} &= (\sum \gamma^t) R_{max} \\ x &= (\gamma^0 + \gamma^1 + \gamma^2 + \dots) \\ &= \gamma^0 + \gamma \times x \\ x - \gamma \times x &= 1 \\ x &= \frac{1}{1 - \gamma}\end{aligned}$$

Decision Making and Reinforcement Learning

$$\begin{aligned}\sum \gamma^t R_{max} &= (\sum \gamma^t) R_{max} \\ x &= (\gamma^0 + \gamma^1 + \gamma^2 + \dots) \\ &= \gamma^0 + \gamma \times x \\ x - \gamma \times x &= 1 \\ x &= \frac{1}{1 - \gamma} \\ x &= \frac{1}{1 - \gamma} \times R_{max}\end{aligned}$$

Decision Making and Reinforcement Learning

Optimal policy.

- $\pi^* = \operatorname{argmax}_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi\right]$

Decision Making and Reinforcement Learning

Optimal policy.

- $\pi^* = \operatorname{argmax}_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi\right]$
- $U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \times R_{max} | \pi, s_0 = s\right]$
 - Notice that utility is different from reward.
 - Reward gives the immediate feedback.
 - Utility gives the long term feedback (**delayed reward**).

Decision Making and Reinforcement Learning

Optimal policy.

- $\pi^* = \operatorname{argmax}_\pi E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi\right]$
- $U^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \times R_{max} | \pi, s_0 = s\right]$
 - Notice that utility is different from reward.
 - Reward gives the immediate feedback.
 - Utility gives the long term feedback (**delayed reward**).
- $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} T(s, a, s') \times U(s')$
 - The optimal policy is the one that for every state returns the action that maximizes the expected utility
- $U(s) = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')]$

Decision Making and Reinforcement Learning

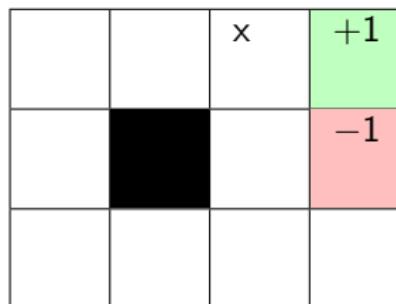
- $$U(s) = R(s) + \gamma \times \max_a \left[\sum_{s'} T(s, a, s') \times U(s') \right]$$
 - n equations and n unknowns.
 - But they are non-linear (max function).

Decision Making and Reinforcement Learning

- $U(s) = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')]$
 - n equations and n unknowns.
 - But they are non-linear (max function).
- Value iteration algorithm:
 - Start with arbitrary utilities
 - Update utilities based on neighbors (states you can reach)
 - Repeat until convergence
 - $U(s)_{t+1} = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')_t]$

Decision Making and Reinforcement Learning

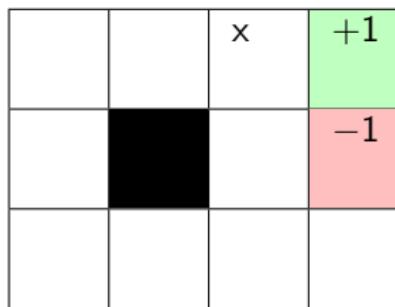
- $U(s)_{t+1} = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')_t]$
 - $\gamma = 0.5, R(s) = -0.04, U(s)_0 = 0$



- $U(x)_1 =$

Decision Making and Reinforcement Learning

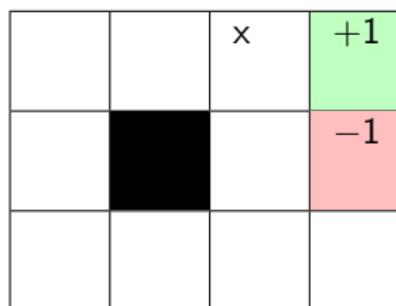
- $U(s)_{t+1} = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')_t]$
 - $\gamma = 0.5, R(s) = -0.04, U(s)_0 = 0$



- $U(x)_1 = -0.04 + 0.5 \times [0 + 0 + 0.8] = 0.36$
- $U(x)_2 =$

Decision Making and Reinforcement Learning

- $U(s)_{t+1} = R(s) + \gamma \times \max_a [\sum_{s'} T(s, a, s') \times U(s')_t]$
 - $\gamma = 0.5, R(s) = -0.04, U(s)_0 = 0$



- $U(x)_1 = -0.04 + 0.5 \times [0 + 0 + 0.8] = 0.36$
- $U(x)_2 = -0.04 + 0.5 \times [0.036 - 0.04 + 0.8] = 0.376$

Decision Making and Reinforcement Learning

Finding the optimal policy.

- ① Start with $\pi_0 = \text{guess.}$
 - ② Evaluate: given π_t calculate U_t
 - ③ Improve: $\pi_{t+1} = \operatorname{argmax}_a \sum T(s, a, s') \times U_t(s')$
-
- Policy iteration: we need to plug policies into utilities.
 - $$U(s)_t = R(s) + \gamma \times \sum_{s'} T(s, \pi(s)_t, s') \times U(s')_t$$
 - n equations and n unknowns.

Thank You!

adrianov@dcc.ufmg.br