CS794/CO673: Optimization for Data Science

Lec 00: Introduction

Yaoliang Yu

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS
DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

September 9, 2022

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Office hours: Friday 4-5pm (DC3617) or by email appointment
- TA: Zeou Hu (zeou.hu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/mycourses/794
  slides, notes, videos, assignments, policy, etc.
- Piazza: piazza.com/uwaterloo.ca/fall2022/co673cs794
  announcements, questions, discussions, etc.
- Learn: learn.uwaterloo.ca/d2l/home/825963
  assignments, solutions, grades, etc.

Prerequisites

- Basic linear algebra, calculus, probability, algorithm
- Some relevant books on course website
- Coding

"Coding to programming is like typing to writing."

— Leslie Lamport

Prerequisites

- Basic linear algebra, calculus, probability, algorithm
- Some relevant books on course website
- Coding

"Coding to programming is like typing to writing."

— Leslie Lamport

Prerequisites

- Basic linear algebra, calculus, probability, algorithm
- Some relevant books on course website
- Coding

https://www.python.org/ https://julialang.org/

"Coding to programming is like typing to writing."

— Leslie Lamport

Prerequisites

- Basic linear algebra, calculus, probability, algorithm
- Some relevant books on course website
- Coding

python™ julia

https://www.python.org/ https://julialang.org/

"Coding to programming is like typing to writing."

— Leslie Lamport

Prerequisites

- Basic linear algebra, calculus, probability, algorithm
- Some relevant books on course website
- Coding

https://www.python.org/ https://julialang.org/

"Coding to programming is like typing to writing."

— *Leslie Lamport*

Textbooks

- No required textbook
- Notes, slides, and code will be posted on the course website
- Some fine textbooks for the ambitious ones:

links available on the course website

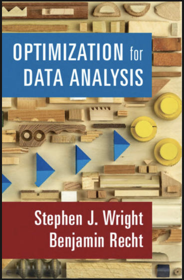
Textbooks

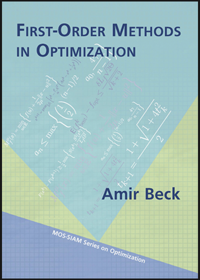
- No required textbook
- Notes, slides, and code will be posted on the course website
- Some fine textbooks for the ambitious ones:

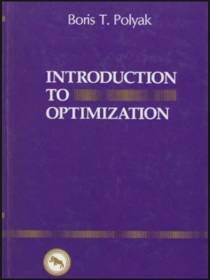
links available on the course website

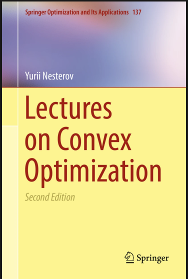
Textbooks

- No required textbook
- Notes, slides, and code will be posted on the course website
- Some fine textbooks for the ambitious ones:









links available on the course website

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Workload

- Roughly 24 lectures, each lasting 80 mins (I hope)
- Expect 5 assignments, approx. 1 bi-weekly
  - 20 points each; total: 100
  - per approval, may substitute 1 assignment with a course project
- Small, constant progress every week
- Submit on LEARN. Submit early and often
  - typeset using LaTeX is recommended

Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ..., notify beforehand
- Appeal within two weeks

Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ..., notify beforehand
- Appeal within two weeks

Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ..., notify beforehand
- Appeal within two weeks

Policy

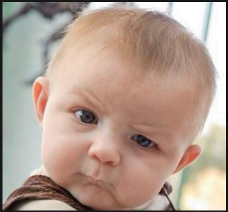
- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ... notify beforehand
- Appeal within two weeks

Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ..., notify beforehand
- Appeal within two weeks

Policy

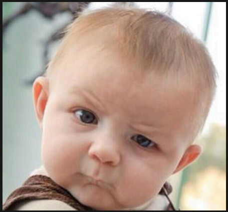
- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion. . .



- NO late submissions!
  - except hospitalization, family urgency, . . . : notify beforehand
- Appeal within two weeks

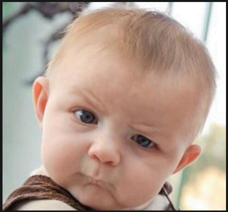
Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ... notify beforehand
- Appeal within two weeks



Policy

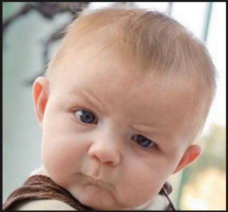
- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion. . .



- NO late submissions!
  - except hospitalization, family urgency, . . . notify beforehand
- Appeal within two weeks

Policy

- Do your work **independently and individually**
  - discussion is fine, but no sharing of text or code
  - explicitly acknowledge any source that helped you
- Ignorance is no excuse
  - good online discussion, more on course website
- Serious offense will result in expulsion...
- NO late submissions!
  - except hospitalization, family urgency, ... notify beforehand
- Appeal within two weeks



Machine Learning is Everywhere

- Everyone uses ML everyday

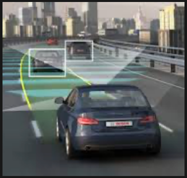
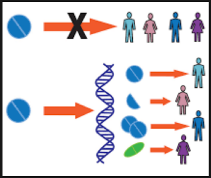
- Lots of cool applications

- Excellent for job-hunting

Machine Learning is Everywhere

- Everyone uses ML everyday

- Lots of cool applications

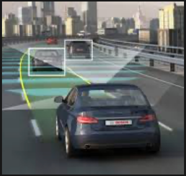



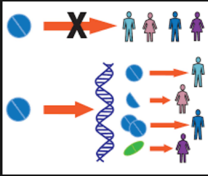
- Excellent for job-hunting

Machine Learning is Everywhere

- Everyone uses ML everyday

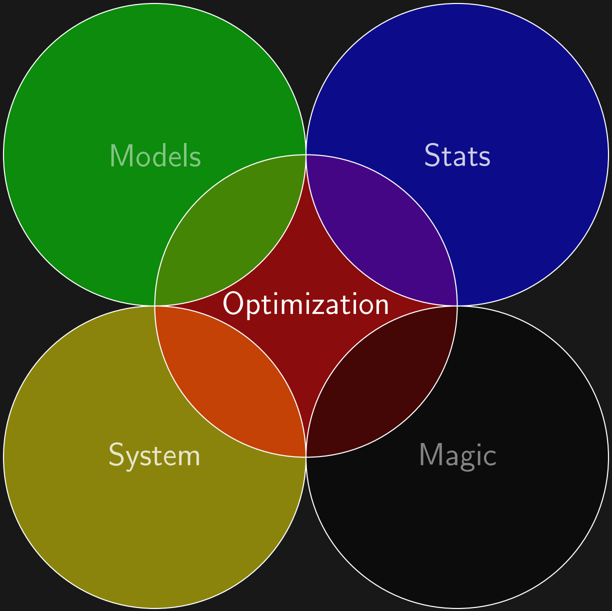
- Lots of cool applications





- Excellent for job-hunting

At the Core is Optimization

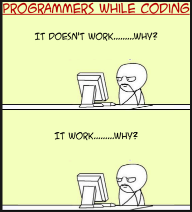


What You Will Learn

- Learn the basic theory and algorithms
- Gain some implementation experience
- Know when to use which algorithm with what guarantees
- Start to formulate problems with algorithms in mind

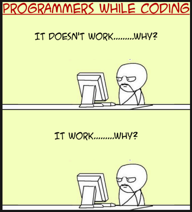
What You Will Learn

- Learn the basic theory and algorithms
- Gain some implementation experience
- Know when to use which algorithm with what guarantees
- Start to formulate problems with algorithms in mind



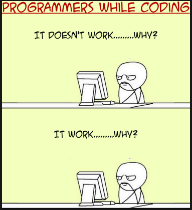
What You Will Learn

- Learn the basic theory and algorithms
- Gain some implementation experience
- Know when to use which algorithm with what guarantees
- Start to formulate problems with algorithms in mind



What You Will Learn

- Learn the basic theory and algorithms
- Gain some implementation experience
- Know when to use which algorithm with what guarantees
- Start to formulate problems with algorithms in mind



	Date	Topic	Slides	Notes	Comments	Applications
00	Sep 09, 2022	Introduction	pdf		pdf	perceptron
01	Sep 09, 2022	Linear System	pdf	pdf		linear regression
02	Sep 16, 2022	Gradient Descent				logistic regression
03	Sep 16, 2022	Projection			hw1	white-box attack
04	Sep 23, 2022	Proximal Gradient				lasso
05	Sep 23, 2022	Subgradient				svm
06	Sep 30, 2022	Conditional Gradient				recommendation
07	Sep 30, 2022	Fictitious Play			hw2	poker
08	Oct 07, 2022	Mirror Descent				sparsity
09	Oct 07, 2022	Metric Gradient				compression
	Oct 14, 2022					reading week
	Oct 14, 2022					reading week
10	Oct 21, 2022	Acceleration				total variation
11	Oct 21, 2022	Smoothing			hw3	robustness
12	Oct 28, 2022	Alternating				expectation-maximization
13	Oct 28, 2022	Coordinate Gradient				covariance estimation
14	Nov 04, 2022	Minimax				adversarial training
15	Nov 04, 2022	Averaging			hw4	GAN
16	Nov 11, 2022	Extragradient				max entropy
17	Nov 11, 2022	Splitting				federated learning
18	Nov 18, 2022	Stochastic Gradient				neural nets
19	Nov 18, 2022	Variance Reduction			hw5	boosting
20	Nov 25, 2022	Randomized Smoothing				certification
21	Nov 25, 2022	Search				black-box attack
22	Dec 02, 2022	Newton				data poisoning
23	Dec 02, 2022	Quasi-Newton				page rank

Let the Journey Begin

What a Dataset Looks Like

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}$	$\mathbf{x}'$
$\mathbb{R}^d \ni$	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
y	$+$	$+$	$-$	$+$	$\cdots$	$-$	?	?!

- each column is a data point: n in total; each has d features
- bottom y is the label vector; binary in this case
- $\mathbf{x}$ and $\mathbf{x}'$ are test samples whose labels need to be predicted

What a Dataset Looks Like

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}$	$\mathbf{x}'$
$\mathbb{R}^d \ni$	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
y	$+$	$+$	$-$	$+$	$\cdots$	$-$	?	?!

- each column is a data point: n in total; each has d features
- bottom y is the label vector; binary in this case
- $\mathbf{x}$ and $\mathbf{x}'$ are test samples whose labels need to be predicted

What a Dataset Looks Like

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}$	$\mathbf{x}'$
$\mathbb{R}^d \ni$	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
y	$+$	$+$	$-$	$+$	$\cdots$	$-$	?	?!

- each column is a data point: n in total; each has d features
- bottom y is the label vector; binary in this case
- $\mathbf{x}$ and $\mathbf{x}'$ are test samples whose labels need to be predicted

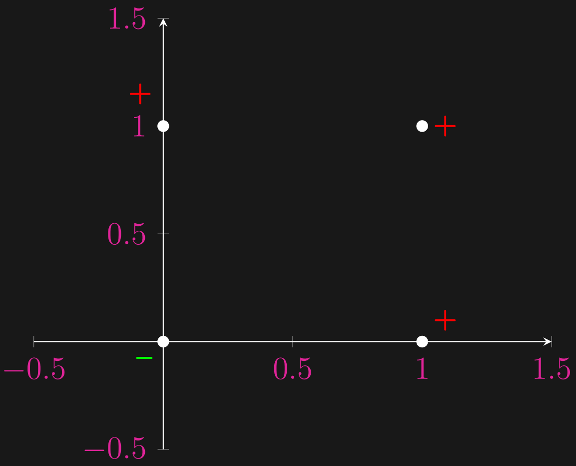
What a Dataset Looks Like

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}$	$\mathbf{x}'$
$\mathbb{R}^d \ni$	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
y	$+$	$+$	$-$	$+$	$\cdots$	$-$	?	?!

- each column is a data point: n in total; each has d features
- bottom y is the label vector; binary in this case
- $\mathbf{x}$ and $\mathbf{x}'$ are test samples whose labels need to be predicted

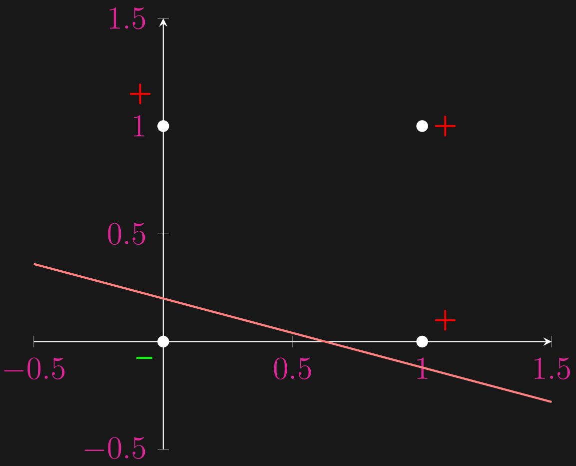
OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



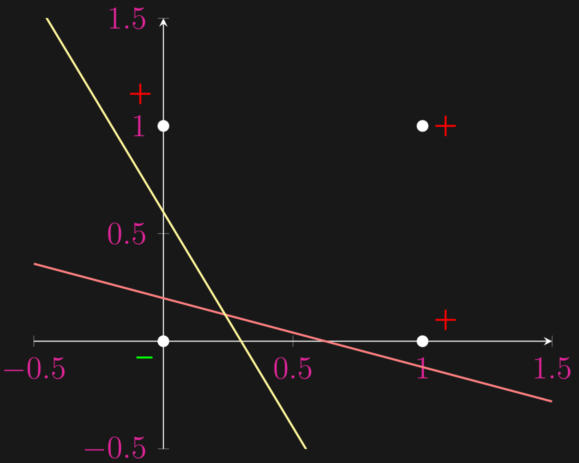
OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



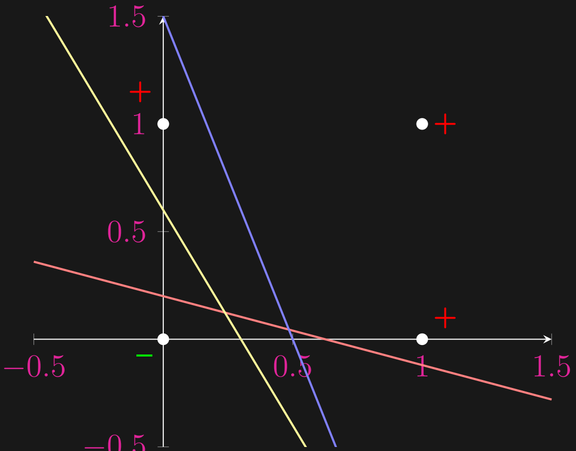
OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



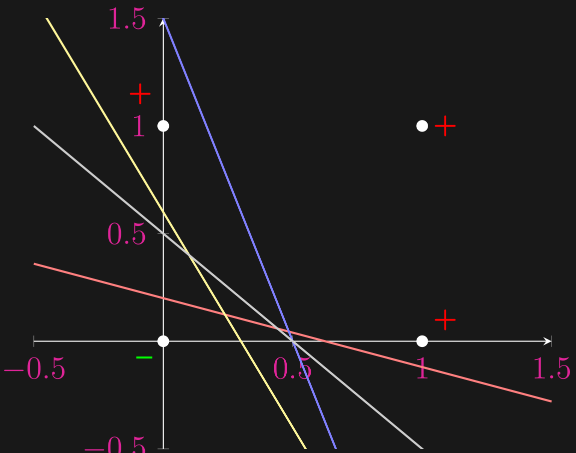
OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



The Early Hype in AI...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI)—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.

The embryo—the Weather Bureau's \$2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of \$100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

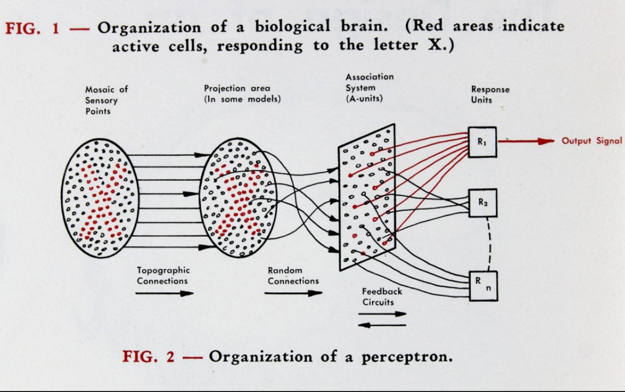
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

New York Times, 1958

...due to Perceptron



Frank Rosenblatt
(1928 – 1971)

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \ \text{ such that } \ \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists

- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \text{ such that } \forall i, \; \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \text{ such that } \forall i,\ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \;\text{ such that }\; \forall i,\; \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \text{ such that } \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \ \text{ such that } \ \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$ partially

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

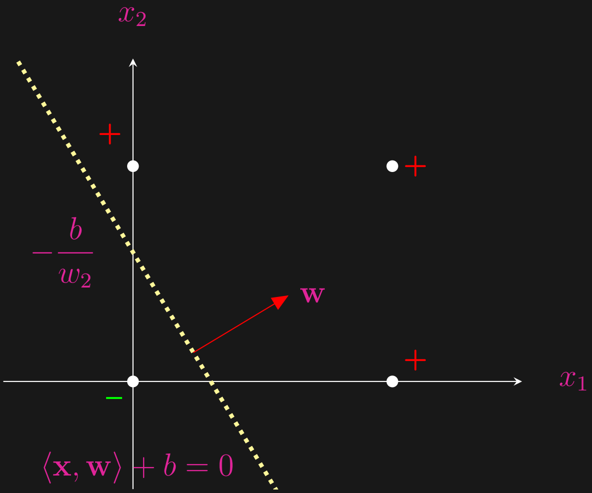
$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \ \text{ such that } \ \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

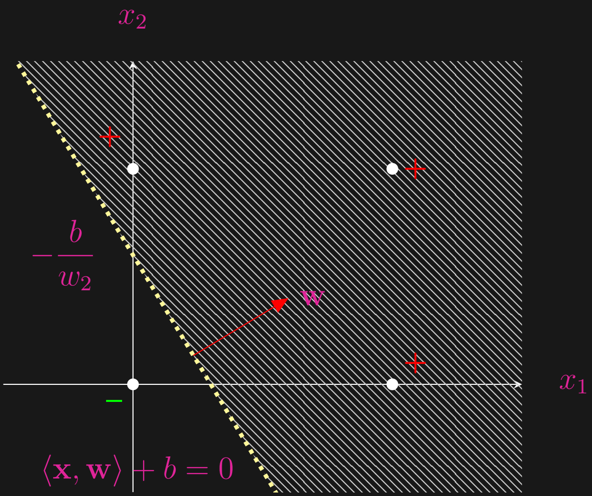
$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$ partially

Geometrically



Geometrically



Algorithm 1: Perceptron

Input: Dataset $\mathcal{D} = \{\!\!\{(\mathbf{x}_i, \mathsf{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\}\!\!\}$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$

Output: approximate solution $\mathbf{w}$ and b

for $t = 1, 2, \ldots$ **do**
- receive index $I_t \in \{1, \ldots, n\}$ `// I_t can be random`
- **if** $\mathsf{y}_{I_t}(\langle \mathbf{x}_{I_t}, \mathbf{w} \rangle + b) \leq \delta$ **then**
  - $\mathbf{w} \leftarrow \mathbf{w} + \mathsf{y}_{I_t} \mathbf{x}_{I_t}$ `// update after a "mistake"`
  - $b \leftarrow b + \mathsf{y}_{I_t}$

- Typically $\delta = 0$ and $\mathbf{w}_0 = \mathbf{0}$, $b = 0$
  - $\mathsf{y}\hat{y} > 0$ vs. $\mathsf{y}\hat{y} < 0$ vs. $\mathsf{y}\hat{y} = 0$, where $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle + b$
- Lazy update: "if it ain't broke, don't fix it"

F. Rosenblatt (1958). "The perceptron: A probabilistic model for information storage and organization in the brain". *Psychological Review*, vol. 65, no. 6, pp. 386–408.

Algorithm 2: Perceptron

Input: Dataset $\mathcal{D} = \Lbag(\mathbf{x}_i, \mathsf{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\Rbag$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$

Output: approximate solution $\mathbf{w}$ and b

```
for t = 1, 2, ... do
    receive index I_t ∈ {1, ..., n}           // I_t can be random
    if y_{I_t}(⟨x_{I_t}, w⟩ + b) ≤ δ then
        w ← w + y_{I_t} x_{I_t}               // update after a "mistake"
        b ← b + y_{I_t}
```

- Typically $\delta = 0$ and $\mathbf{w}_0 = \mathbf{0}$, $b = 0$
  - $\mathsf{y}\hat{y} > 0$ vs. $\mathsf{y}\hat{y} < 0$ vs. $\mathsf{y}\hat{y} = 0$, where $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle + b$
- Lazy update: "if it ain't broke, don't fix it"

F. Rosenblatt (1958). "The perceptron: A probabilistic model for information storage and organization in the brain". *Psychological Review*, vol. 65, no. 6, pp. 386–408.

Algorithm 3: Perceptron

Input: Dataset $\mathcal{D} = \wr(\mathbf{x}_i, \mathsf{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n \int$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$

Output: approximate solution $\mathbf{w}$ and b

for $t = 1, 2, \ldots$ **do**

  receive index $I_t \in \{1, \ldots, n\}$ `// It can be random`

  if $\mathsf{y}_{I_t}(\langle \mathbf{x}_{I_t}, \mathbf{w} \rangle + b) \leq \delta$ **then**

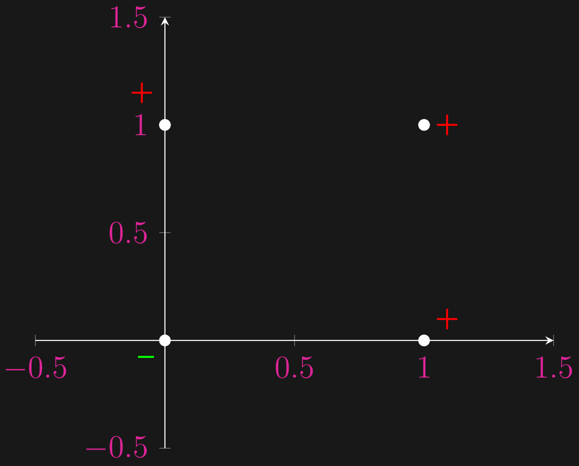
    $\mathbf{w} \leftarrow \mathbf{w} + \mathsf{y}_{I_t}\mathbf{x}_{I_t}$ `// update after a “mistake”`

    $b \leftarrow b + \mathsf{y}_{I_t}$

- Typically $\delta = 0$ and $\mathbf{w}_0 = \mathbf{0}$, $b = 0$
  - $\mathsf{y}\hat{y} > 0$ vs. $\mathsf{y}\hat{y} < 0$ vs. $\mathsf{y}\hat{y} = 0$, where $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle + b$
- Lazy update: “if it ain’t broke, don’t fix it”

F. Rosenblatt (1958). “The perceptron: A probabilistic model for information storage and organization in the brain”. *Psychological Review*, vol. 65, no. 6, pp. 386–408.

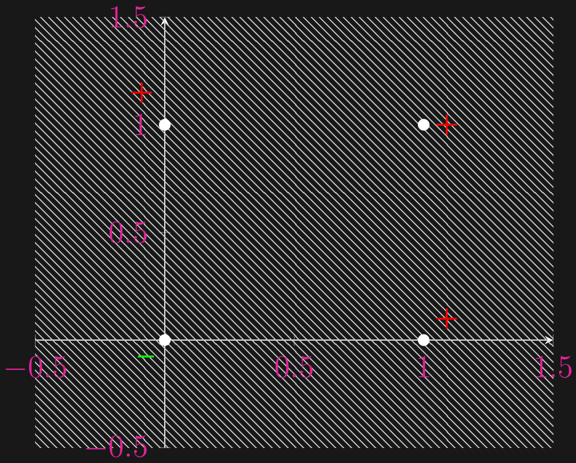
Does it work? ☞ code



$$\hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

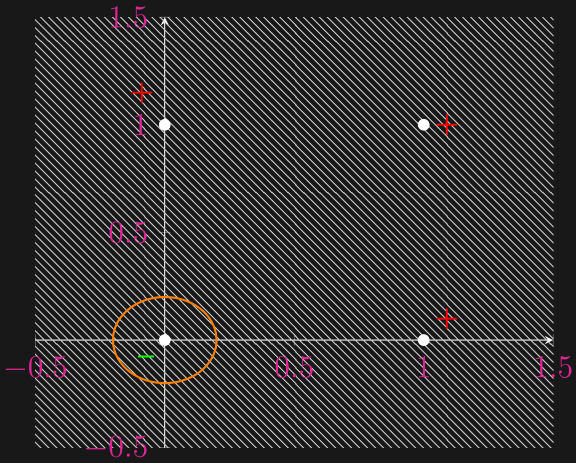
Does it work? ☞ code



$$\mathbf{w} = [0, 0],\ b = 0,\ \ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

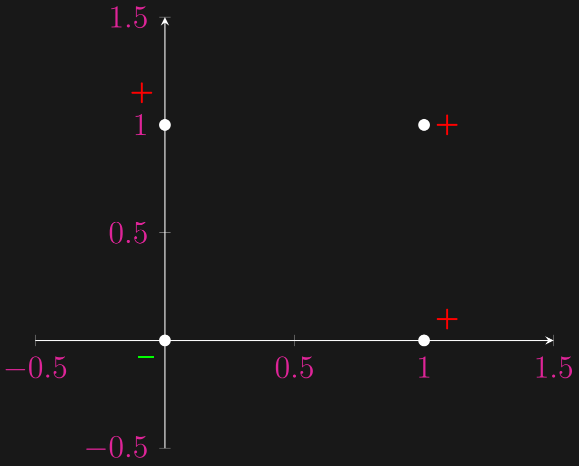
Does it work? ☞ code



$$\mathbf{w} = [0, 0],\ b = 0,\ \ \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w}\rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

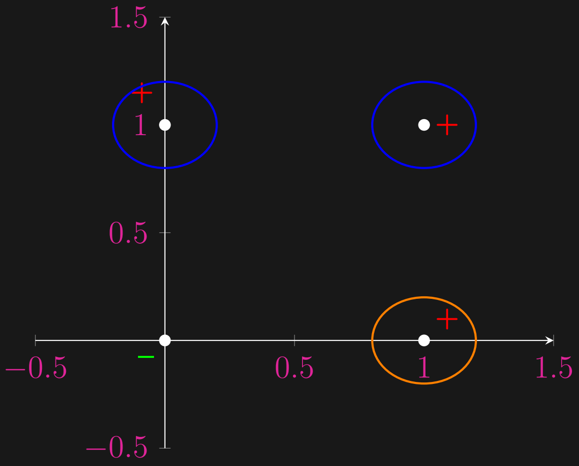


$$\mathbf{w} = [0, 0],\ \ b = -1, \quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

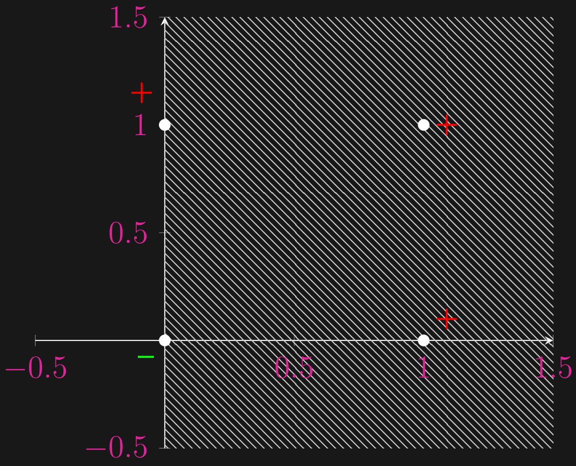
☞ code



$$\mathbf{w} = [0, 0],\ \ b = -1,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

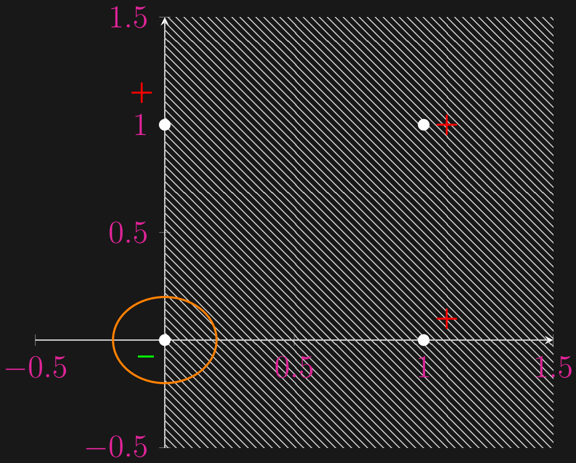


$$\mathbf{w} = [1, 0],\ b = 0,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

☞ code

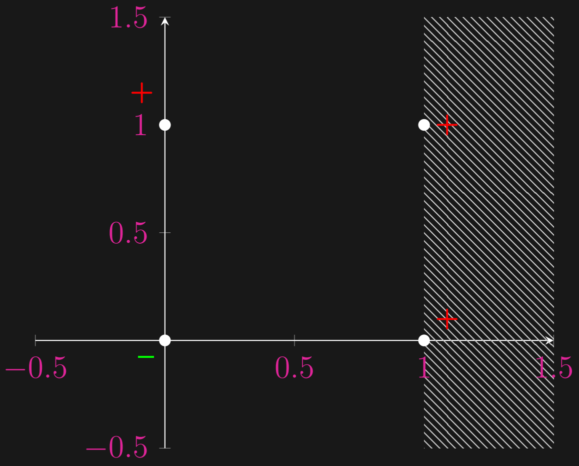


$$\mathbf{w} = [1, 0],\ b = 0,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

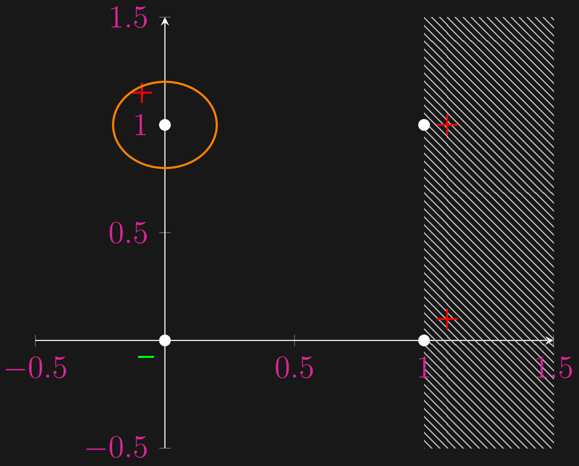
☞ code



$$\mathbf{w} = [1, 0],\ \ b = -1, \quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

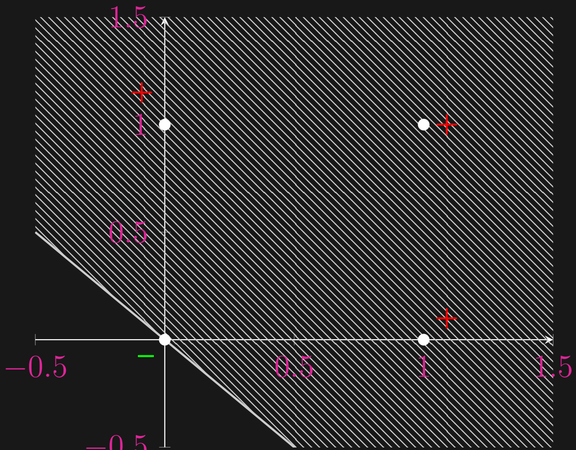


$$\mathbf{w} = [1, 0],\ b = -1,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

☞ code

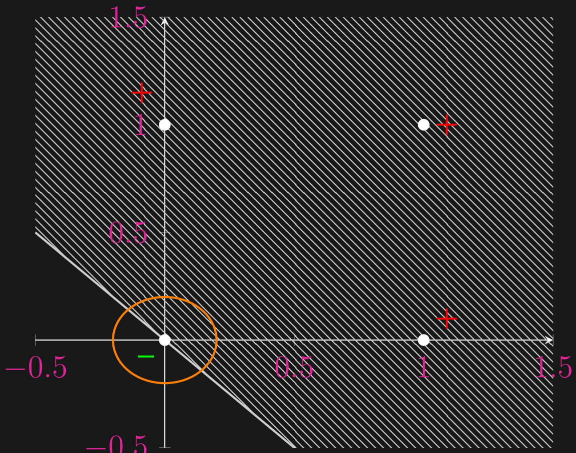


$$\mathbf{w} = [1, 1],\ b = 0,\quad \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

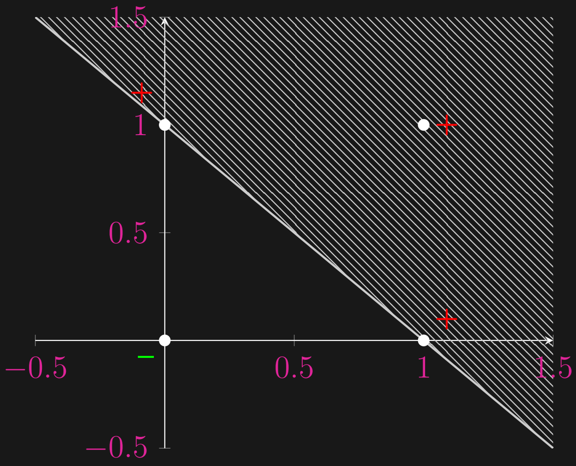
☞ code



$$\mathbf{w} = [1, 1],\ b = 0,\ \ \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

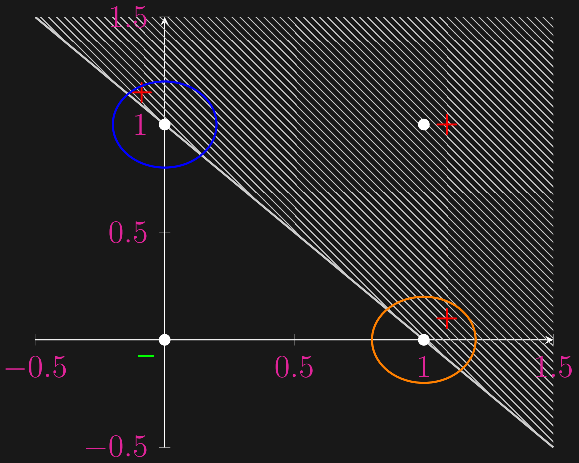


$$\mathbf{w} = [1, 1],\ \ b = -1,\ \ \ \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

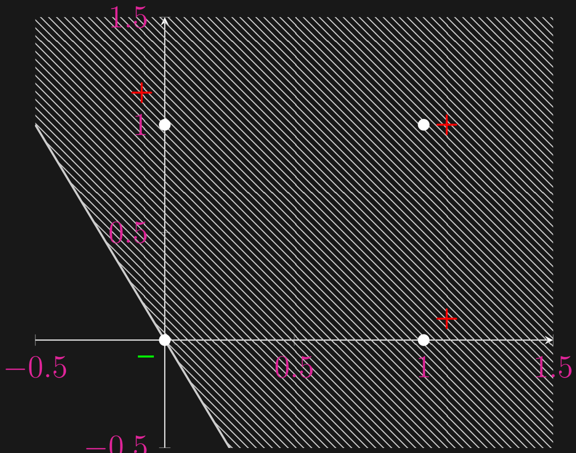
☞ code



$$\mathbf{w} = [1,1],\ b = -1,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w}\rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

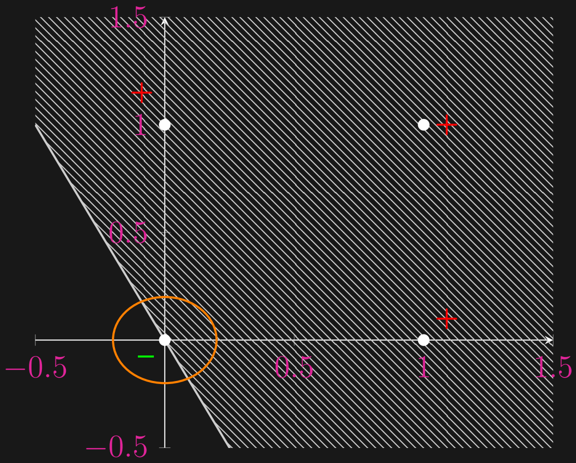
Does it work? ☞ code



$$\mathbf{w} = [2, 1],\ b = 0,\ \ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

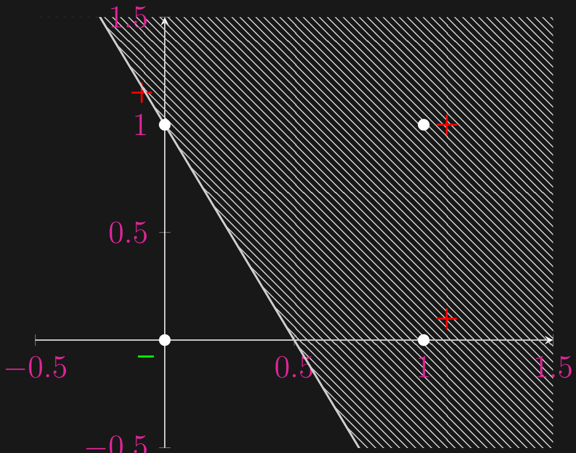
Does it work? ☞ code



$$\mathbf{w} = [2, 1],\ b = 0,\quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

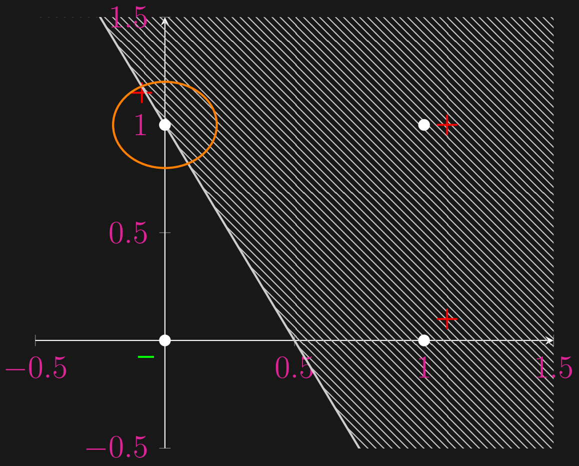


$$\mathbf{w} = [2, 1],\ b = -1,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w}\rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

☞ code

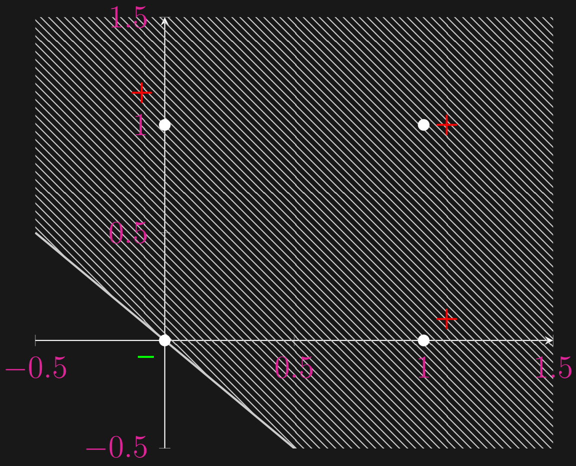


$$\mathbf{w} = [2, 1],\ b = -1,\quad \hat{\mathbf{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w}\rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

☞ code

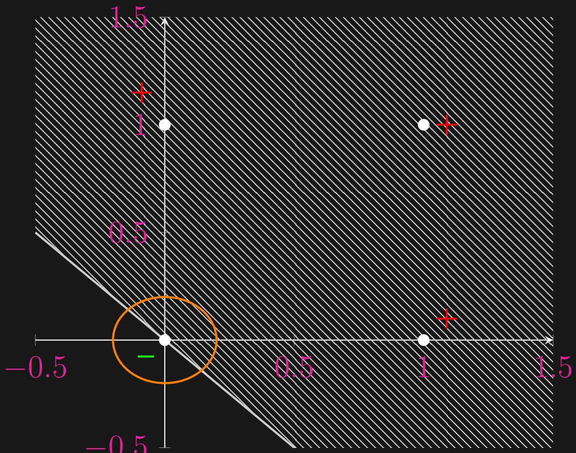


$$\mathbf{w} = [2, 2],\ \ b = 0,\ \ \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work?

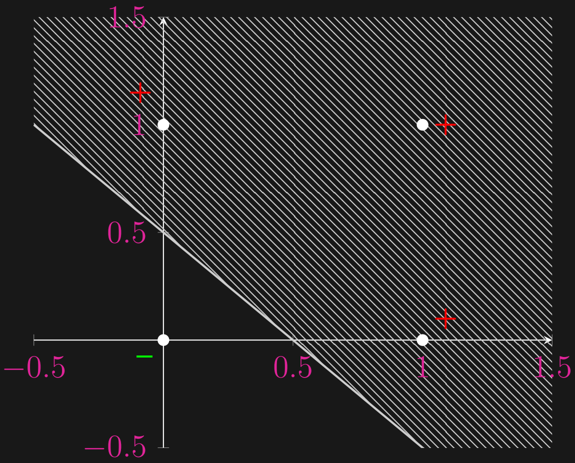
☞ code



$$\mathbf{w} = [2, 2],\ b = 0,\ \ \hat{y} = \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\operatorname{sign}(0)$ is undefined (i.e., always counted as a mistake).

Does it work? ☞ code

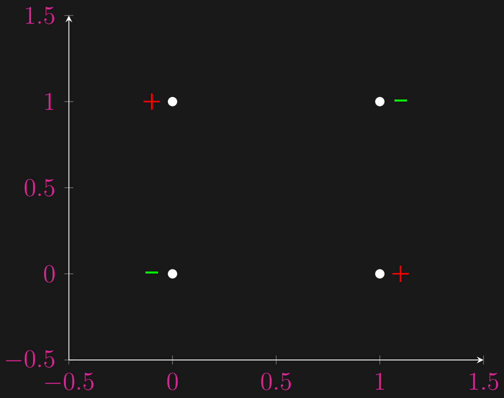


$$\mathbf{w} = [2, 2],\ \ b = -1, \quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (i.e., always counted as a mistake).

XOR Dataset

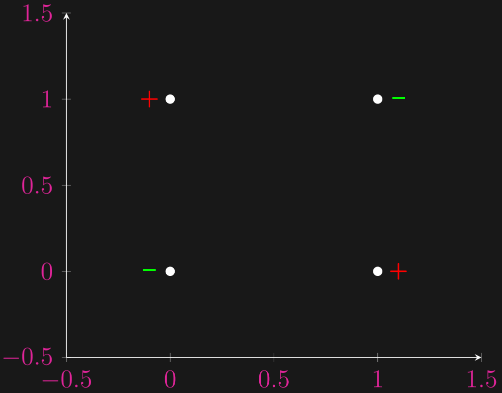
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
$\mathbf{y}$	−	+	+	−



- Prove that no line can separate + from −
- What happens if we run Perceptron regardless?

XOR Dataset

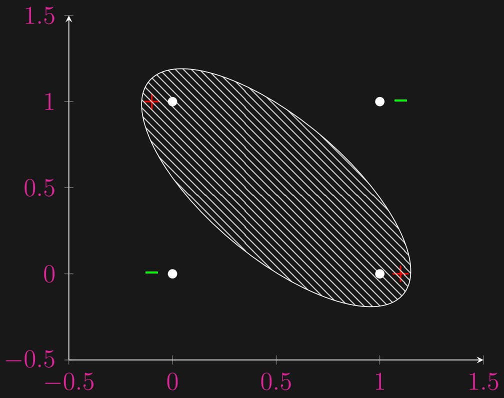
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
$\mathbf{y}$	−	+	+	−



- Prove that no line can separate + from −
- What happens if we run Perceptron regardless?

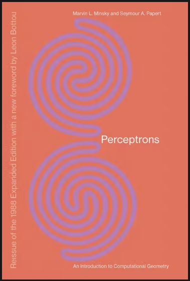
XOR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
$\mathbf{y}$	$-$	$+$	$+$	$-$

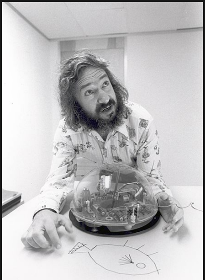


- Prove that no line can separate $+$ from $-$
- What happens if we run Perceptron regardless?

Perceptron and the 1^{st} AI Winter



Marvin Minsky
(1927 – 2016)



Seymour Papert
(1928 – 2016)

M. L. Minsky and S. A. Papert (1969). "Perceptron". MIT press.

Projection Algorithms

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \;\; \text{such that} \;\; \forall i, \;\; \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0$$

$$\text{find } \mathsf{w} = [\mathbf{w}; b] \in \mathbb{R}^{d+1} \;\; \text{such that} \;\; \forall i, \;\; \langle \mathsf{a}_i, \mathsf{w} \rangle \leq c_i, \;\; \mathsf{a}_i = -\mathsf{y}_i[\mathbf{x}_i; 1]$$

$$\text{find } \mathsf{w} \in \mathbb{R}^p \;\; \text{such that} \;\; \mathsf{A}^\top \mathsf{w} \leq \mathbf{c}$$

Algorithm 4: Projection Algorithm for Linear Inequalities

Input: $\mathsf{A} \in \mathbb{R}^{p \times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathsf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$

for $t = 1, 2, \ldots$ **do**

  select index $I_t \in \{1, \ldots, n\}$ `// index` I_t `can be random`

  $\mathsf{w} \leftarrow (1 - \eta)\mathsf{w} + \eta \left[\mathsf{w} - \frac{(\langle \mathsf{a}_{I_t}, \mathsf{w} \rangle - c_{I_t})^+}{\|\mathsf{a}_{I_t}\|_2} \cdot \frac{\mathsf{a}_{I_t}}{\|\mathsf{a}_{I_t}\|_2} \right]$

T. S. Motzkin and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404; S. Agmon (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.

Projection Algorithms

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \ \text{ such that } \ \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0$$

$$\text{find } \mathbf{w} = [\mathbf{w}; b] \in \mathbb{R}^{d+1} \ \text{ such that } \ \forall i, \ \langle \mathbf{a}_i, \mathbf{w} \rangle \leq c_i, \ \mathbf{a}_i = -\mathsf{y}_i[\mathbf{x}_i; 1]$$

$$\text{find } \mathbf{w} \in \mathbb{R}^p \ \text{ such that } \ \mathbf{A}^\top \mathbf{w} \leq \mathbf{c}$$

Algorithm 5: Projection Algorithm for Linear Inequalities

Input: $\mathbf{A} \in \mathbb{R}^{p \times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathbf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$

```
for t = 1, 2, ... do
    select index I_t ∈ {1, ..., n}        // index I_t can be random
    w ← (1 − η)w + η [ w − ((⟨a_{I_t}, w⟩ − c_{I_t})^+ / ||a_{I_t}||_2) · (a_{I_t} / ||a_{I_t}||_2) ]
```

$$\mathbf{w} \leftarrow (1-\eta)\mathbf{w} + \eta \left[\mathbf{w} - \frac{(\langle \mathbf{a}_{I_t}, \mathbf{w} \rangle - c_{I_t})^+}{\|\mathbf{a}_{I_t}\|_2} \cdot \frac{\mathbf{a}_{I_t}}{\|\mathbf{a}_{I_t}\|_2}\right]$$

T. S. Motzkin and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404; S. Agmon (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.

Projection Algorithms

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \ \text{ such that } \ \forall i,\ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0$$

$$\text{find } \mathsf{w} = [\mathbf{w}; b] \in \mathbb{R}^{d+1} \ \text{ such that } \ \forall i,\ \langle \mathsf{a}_i, \mathsf{w} \rangle \le c_i,\ \mathsf{a}_i = -\mathsf{y}_i[\mathbf{x}_i; 1]$$

$$\text{find } \mathsf{w} \in \mathbb{R}^p \ \text{ such that } \ \mathsf{A}^\top \mathsf{w} \le \mathbf{c}$$

Algorithm 6: Projection Algorithm for Linear Inequalities

Input: $\mathsf{A} \in \mathbb{R}^{p \times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathsf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$

```
for t = 1, 2, ... do
    select index I_t ∈ {1, ..., n}          // index I_t can be random
    w ← (1 − η)w + η [ w − ((⟨a_{I_t}, w⟩ − c_{I_t})^+ / ‖a_{I_t}‖_2) · (a_{I_t} / ‖a_{I_t}‖_2) ]
```

$$\mathsf{w} \leftarrow (1-\eta)\mathsf{w} + \eta \left[\mathsf{w} - \frac{(\langle \mathsf{a}_{I_t}, \mathsf{w} \rangle - c_{I_t})^+}{\|\mathsf{a}_{I_t}\|_2} \cdot \frac{\mathsf{a}_{I_t}}{\|\mathsf{a}_{I_t}\|_2} \right]$$

T. S. Motzkin and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404; S. Agmon (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.

Projection Algorithms

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \;\text{ such that }\; \forall i,\; \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w}\rangle + b) > 0$$

$$\text{find } \mathbf{w} = [\mathbf{w}; b] \in \mathbb{R}^{d+1} \;\text{ such that }\; \forall i,\; \langle \mathbf{a}_i, \mathbf{w}\rangle \le c_i,\; \mathbf{a}_i = -\mathsf{y}_i[\mathbf{x}_i; 1]$$

$$\text{find } \mathbf{w} \in \mathbb{R}^p \;\text{ such that }\; \mathbf{A}^\top \mathbf{w} \le \mathbf{c}$$

Algorithm 7: Projection Algorithm for Linear Inequalities

Input: $\mathbf{A} \in \mathbb{R}^{p\times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathbf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$

for $t = 1, 2, \ldots$ **do**

  select index $I_t \in \{1, \ldots, n\}$ `// index` I_t `can be random`

  $\mathbf{w} \leftarrow (1-\eta)\mathbf{w} + \eta \left[\mathbf{w} - \frac{(\langle \mathbf{a}_{I_t}, \mathbf{w}\rangle - c_{I_t})^+}{\|\mathbf{a}_{I_t}\|_2} \cdot \frac{\mathbf{a}_{I_t}}{\|\mathbf{a}_{I_t}\|_2}\right]$

T. S. Motzkin and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404; S. Agmon (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.

Interpreting Perceptron

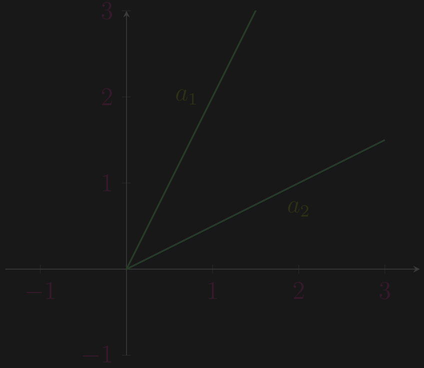
Theorem:

$$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$$

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$

$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$

$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

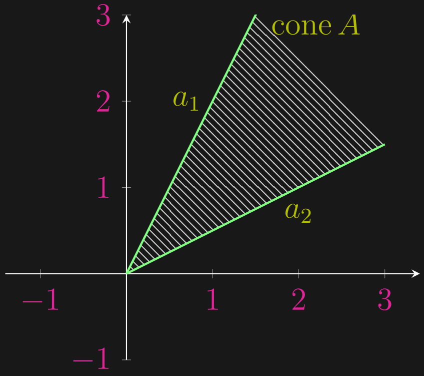


Interpreting Perceptron

Theorem:

$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$

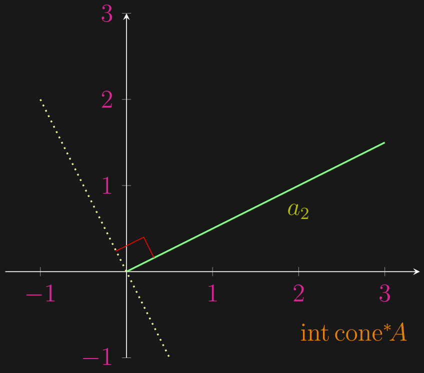
$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$



Interpreting Perceptron

Theorem:
$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset$.

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

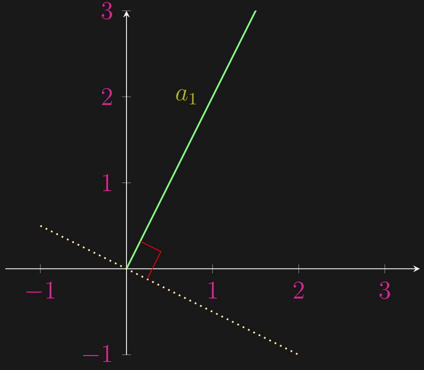


Interpreting Perceptron

Theorem:

$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset$.

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

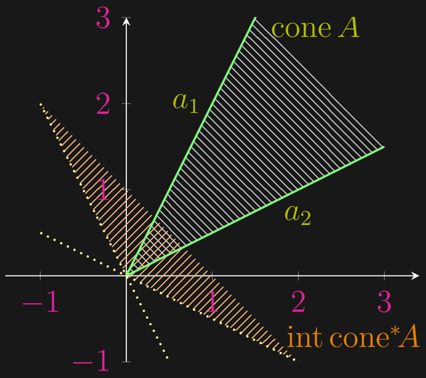


Interpreting Perceptron

Theorem:

$$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$$

$$\begin{aligned} \operatorname{cone} A &:= \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\} \\ \operatorname{cone}^* A &:= \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\} \\ \operatorname{int}\operatorname{cone}^* A &:= \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\} \end{aligned}$$

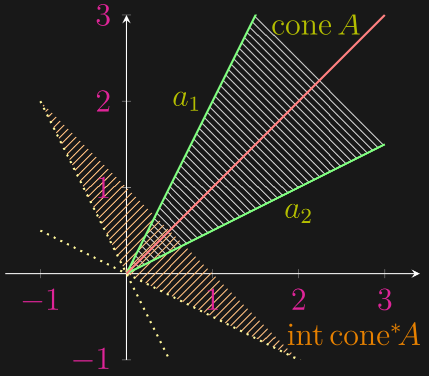


Interpreting Perceptron

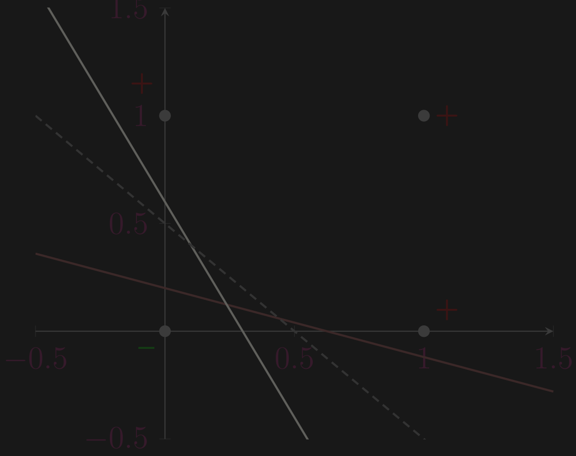
Theorem:

$\text{int cone}^* A \neq \emptyset \iff \text{int cone}^* A \cap \text{cone} A \neq \emptyset$.

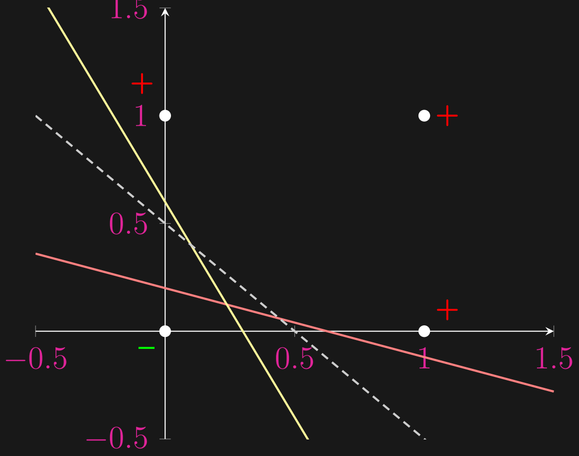
$$\text{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\text{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\text{int cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$



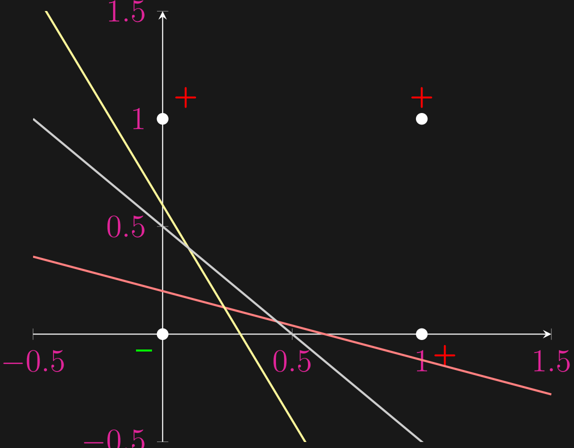
Is Perceptron Unique?



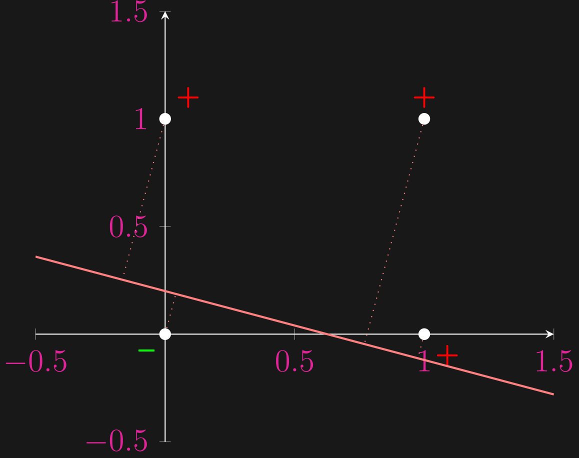
Is Perceptron Unique?



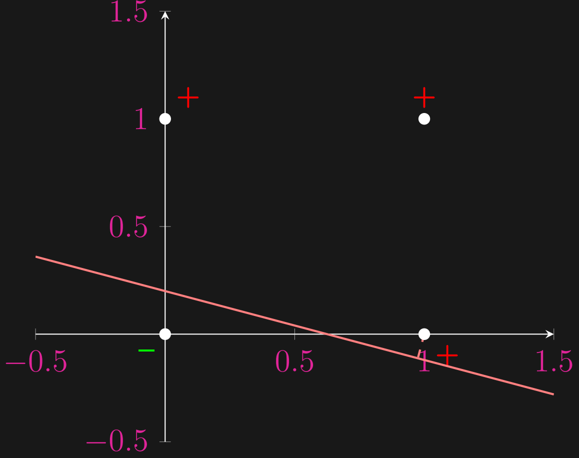
Support Vector Machines: Primal



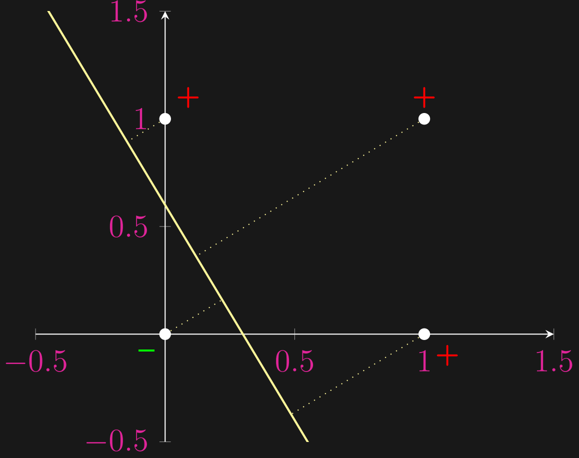
Support Vector Machines: Primal



Support Vector Machines: Primal

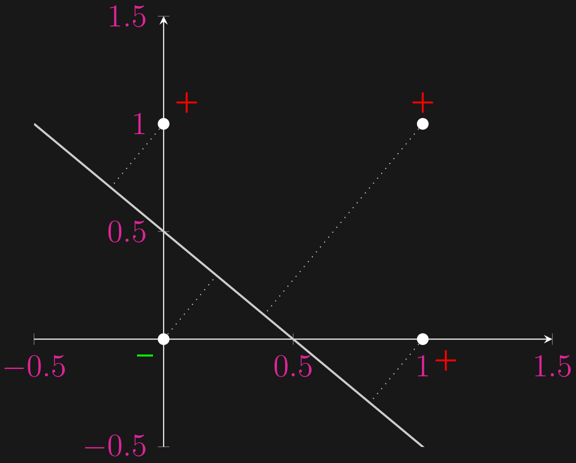


Support Vector Machines: Primal

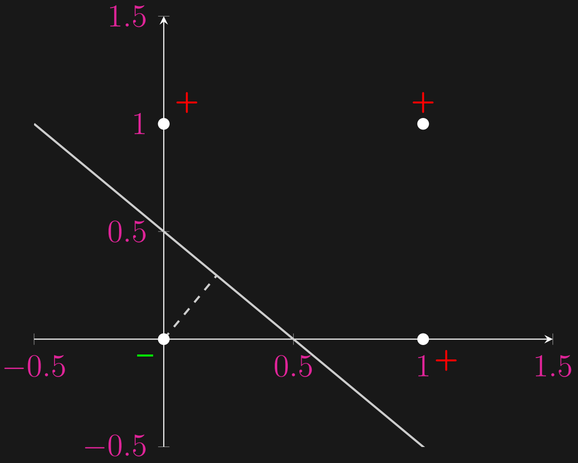


Support Vector Machines: Primal

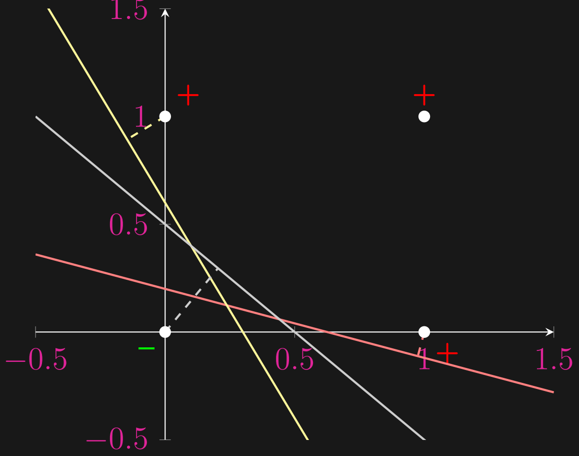
Support Vector Machines: Primal



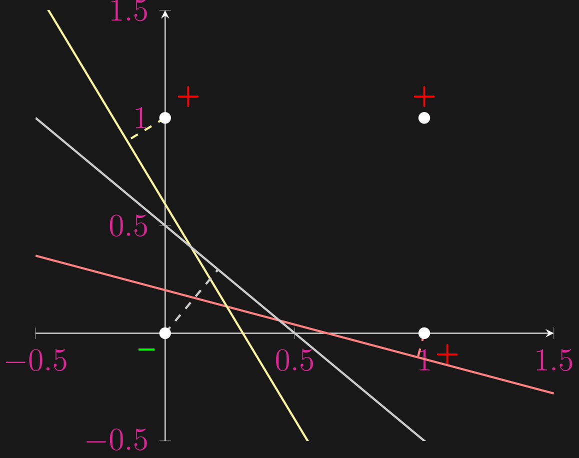
Support Vector Machines: Primal



Support Vector Machines: Primal

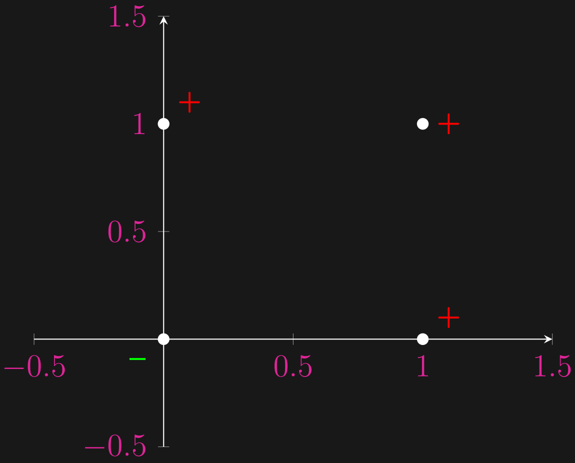


Support Vector Machines: Primal

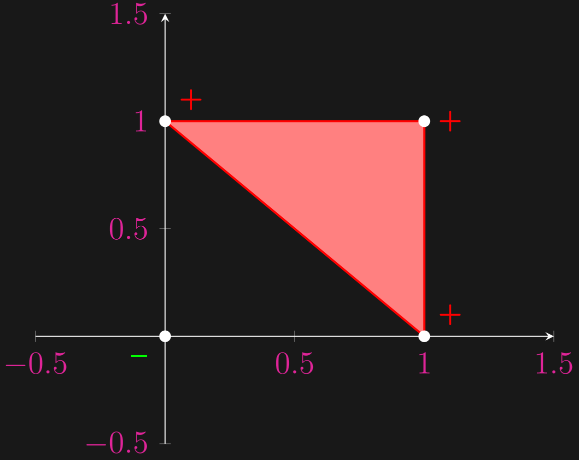


$$\max_{\mathbf{w}: \forall i, \hat{y}_i \mathsf{y}_i > 0} \; \min_{i=1,\ldots,n} \frac{\hat{y}_i \mathsf{y}_i}{\|\mathbf{w}\|}, \quad \text{where} \quad \hat{y}_i := \langle \mathbf{x}_i, \mathbf{w} \rangle + b$$

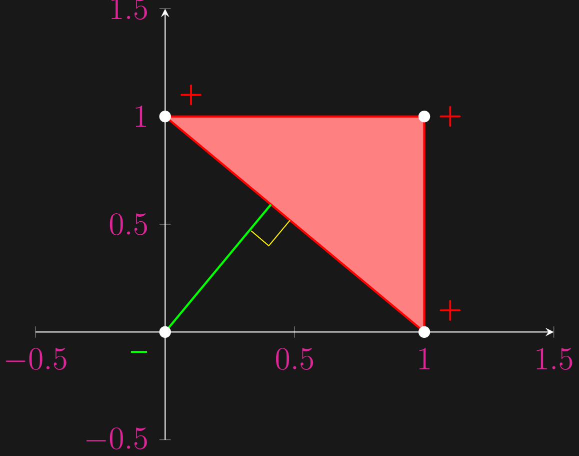
Support Vector Machines: Dual



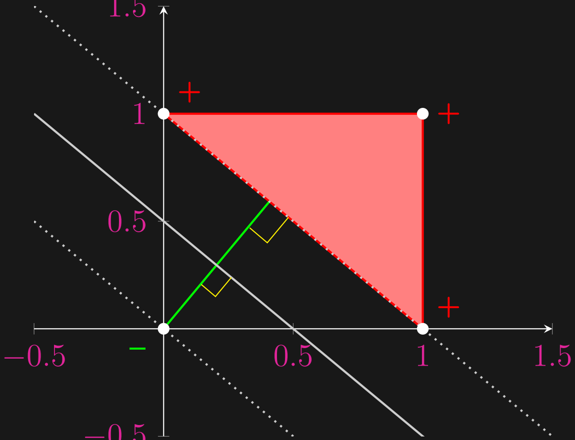
Support Vector Machines: Dual



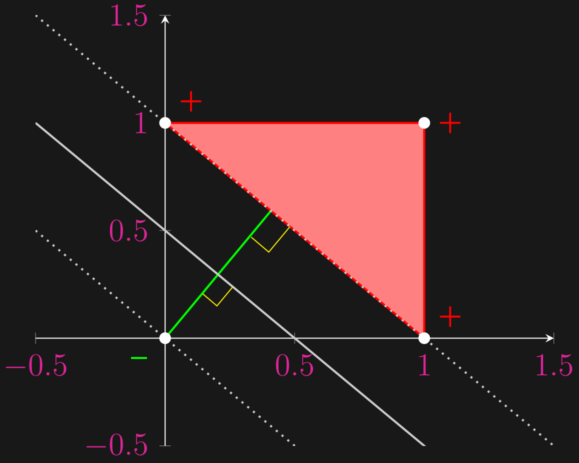
Support Vector Machines: Dual



Support Vector Machines: Dual

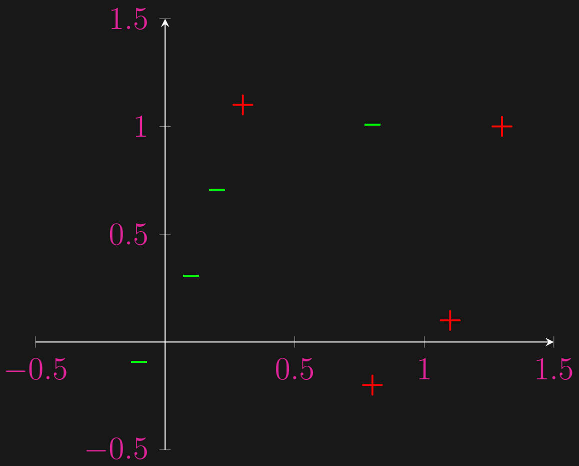


Support Vector Machines: Dual

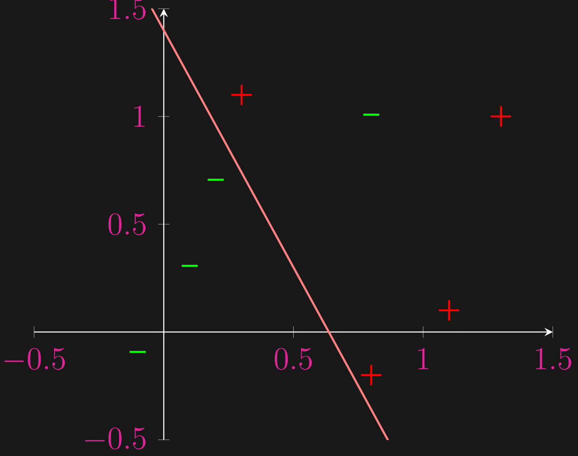


$$\min_{\boldsymbol{\mu} \in \Delta_+} \min_{\boldsymbol{\nu} \in \Delta_-} \left\| \sum_{i:\mathsf{y}_i=+} \mu_i \mathbf{x}_i - \sum_{j:\mathsf{y}_j=-} \nu_j \mathbf{x}_j \right\|$$

Beyond Separability

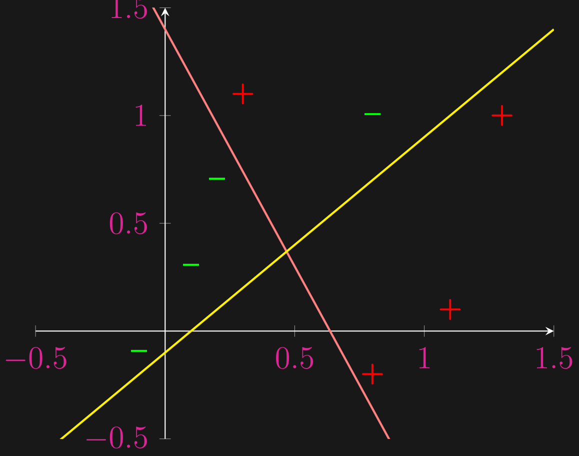


Beyond Separability



Beyond Separability

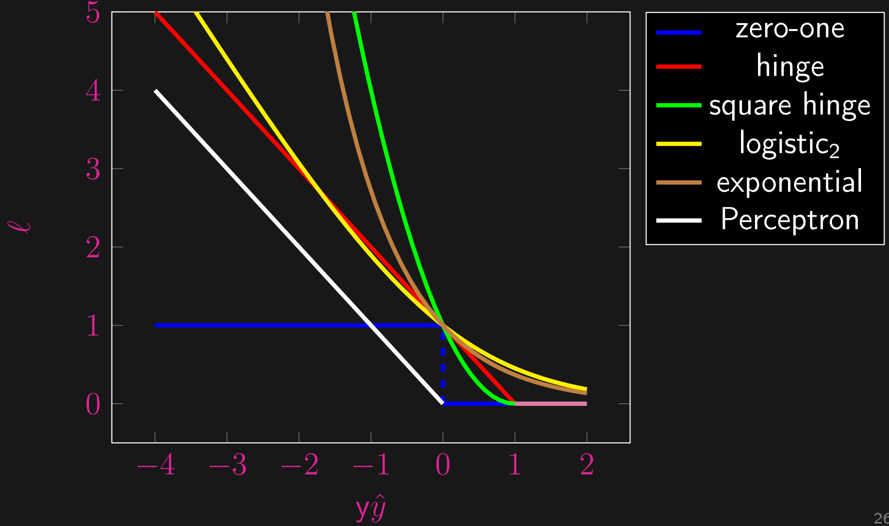
Beyond Separability



$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathsf{y}\hat{y}) + \mathrm{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$

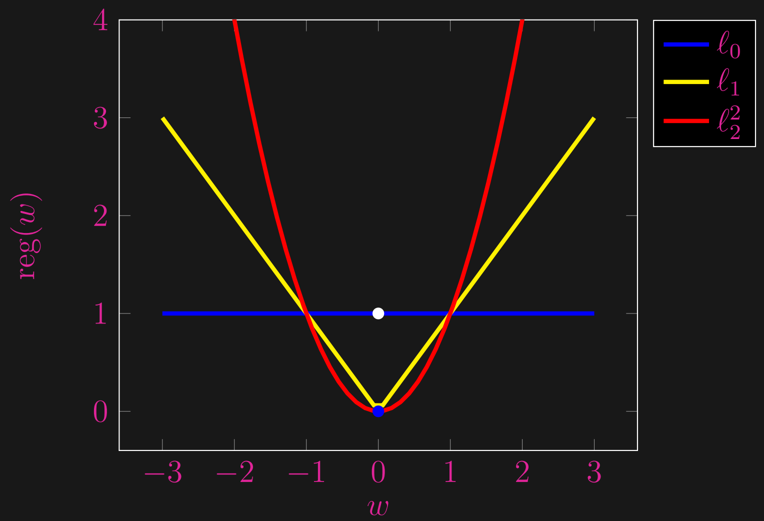
Empirical Risk Minimization

$$\min_{\mathbf{w}} \ \hat{\mathrm{E}}\ell(\mathrm{y}\hat{y}) \qquad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$



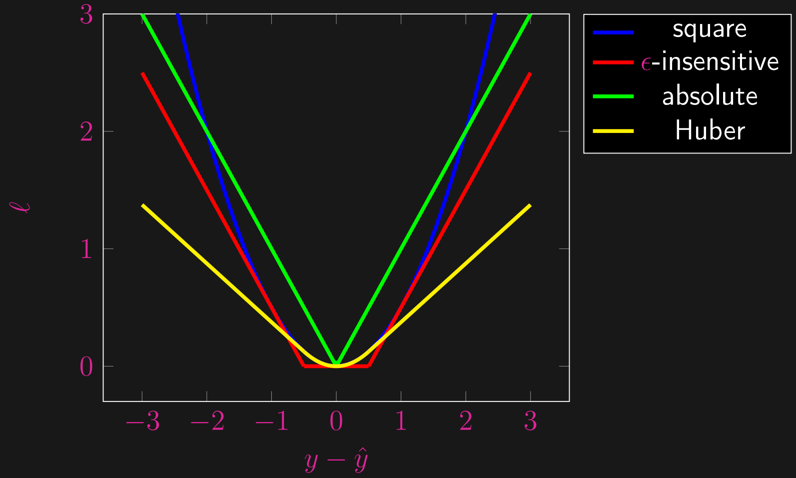
Regularization

$$\min_{\mathbf{w}} \quad \mathrm{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$



Regression

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(y-\hat{y}) + \mathrm{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w}\rangle + b$$



Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Plan I: Basic

- Lec04: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec05: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec10: Acceleration: optimal algorithm under smoothness
- Lec11: Smoothing: nonsmooth —> smooth
- Lec12: Alternating: divide and conquer
- Lec13: Coordinate Gradient: large model
- Lec18: Stochastic Gradient: large dataset
- Lec22: Newton: even faster under smoothness
- Lec23: Quasi-Newton: Newton made economical

Denoising

$$\min_{\mathbf{z}} \underbrace{\frac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2}_{\text{fidelity}} + \underbrace{\lambda \cdot \|\mathbf{z}\|_{\text{tv}}}_{\text{regularization}}$$

- λ controls the trade-off
- regularization encodes prior knowledge
- crucial to not over-smooth

Denoising

$$\min_{\mathbf{z}} \underbrace{\tfrac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2}_{\text{fidelity}} + \underbrace{\lambda \cdot \|\mathbf{z}\|_{\text{tv}}}_{\text{regularization}}$$

- λ controls the trade-off
- regularization encodes prior knowledge
- crucial to not over-smooth

Denoising

$$\min_{\mathbf{z}} \underbrace{\tfrac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2}_{\text{fidelity}} + \underbrace{\lambda \cdot \|\mathbf{z}\|_{\text{tv}}}_{\text{regularization}}$$

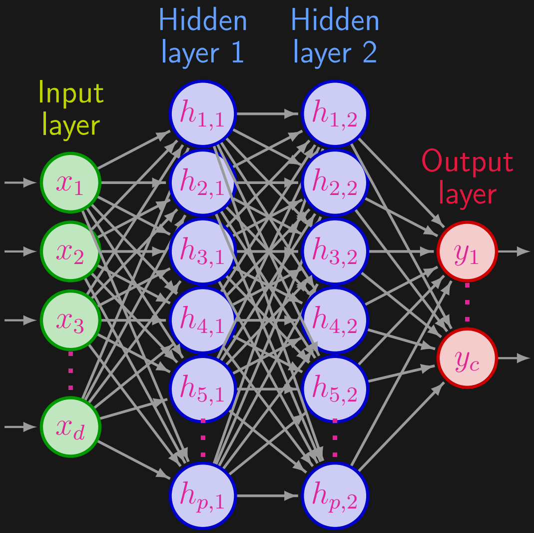
- λ controls the trade-off
- regularization encodes prior knowledge
- crucial to not over-smooth

Denoising

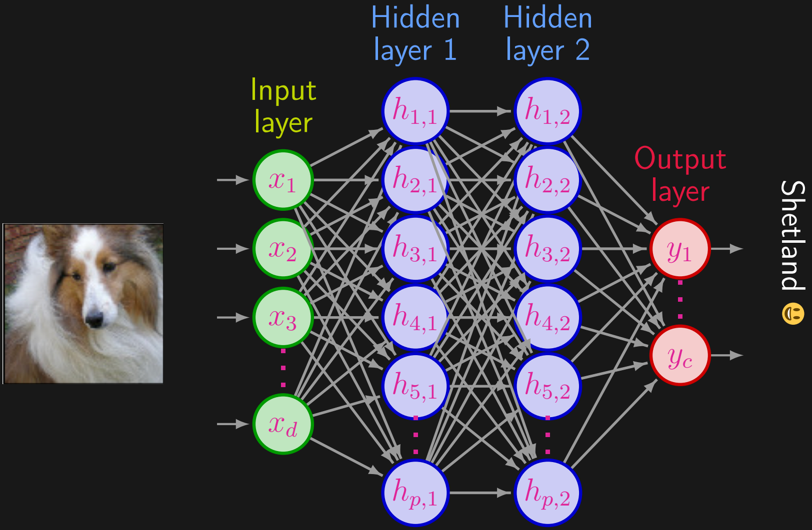
$$\min_{\mathbf{z}} \; \underbrace{\tfrac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2}_{\text{fidelity}} \; + \; \underbrace{\lambda \cdot \|\mathbf{z}\|_{\text{tv}}}_{\text{regularization}}$$

- λ controls the trade-off
- regularization encodes prior knowledge
- crucial to not over-smooth

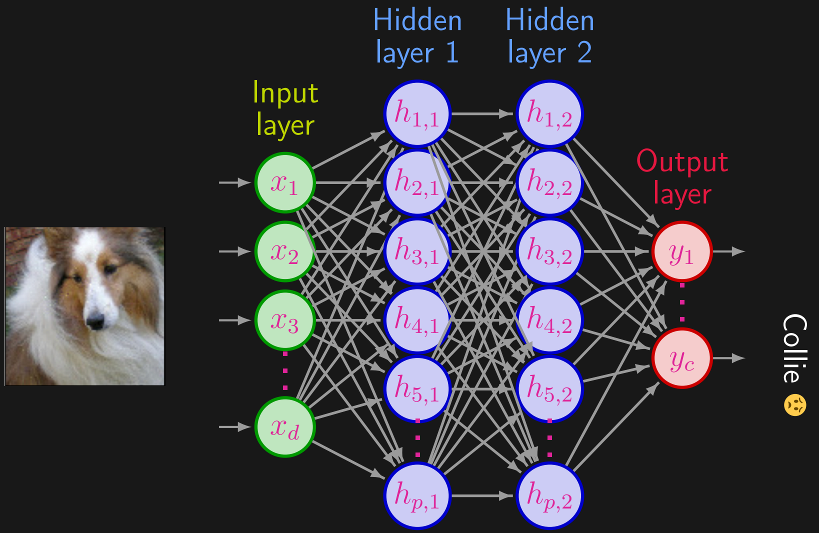
Adversarial Examples



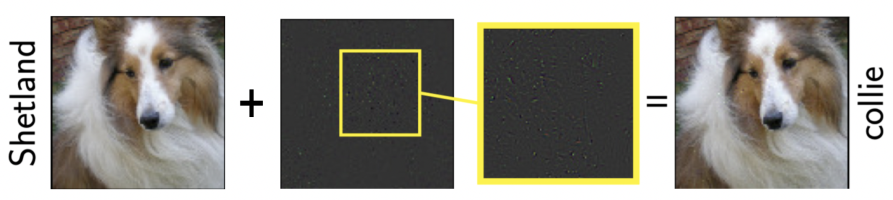
Adversarial Examples



Adversarial Examples



Adversarial Attacks

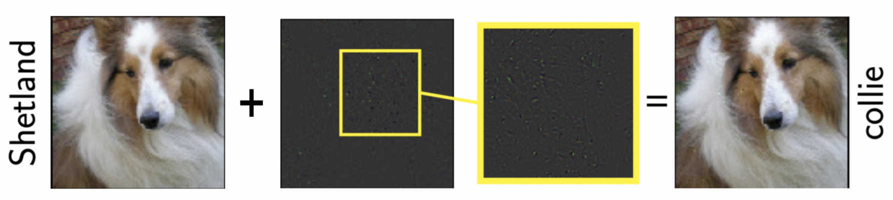


- Mathematically, a neural network is a function $f(\mathbf{w};\mathbf{x})$
- Typically, input $\mathbf{x}$ is given and network weights $\mathbf{w}$ optimized
- Could also freeze weights $\mathbf{w}$ and optimize $\mathbf{x}$, adversarially!

$$\min_{\boldsymbol{\delta}} \text{size}(\boldsymbol{\delta}) \quad \text{s.t.} \quad \text{pred}[f(\mathbf{w};\mathbf{x}+\boldsymbol{\delta})] \neq y$$

- More generally: $\max_{\boldsymbol{\delta}} \ell(\mathbf{w};\mathbf{x}+\boldsymbol{\delta},y) \quad \text{s.t.} \quad \text{size}(\boldsymbol{\delta}) \leq \epsilon$

Adversarial Attacks

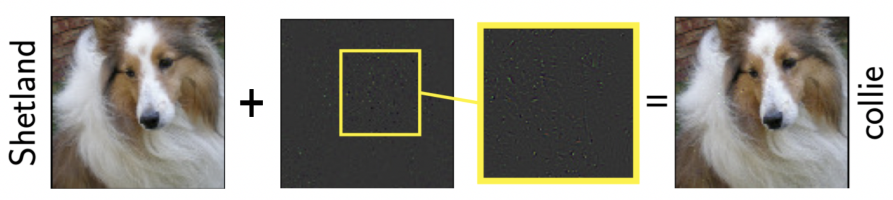


- Mathematically, a neural network is a function $f(\mathbf{w}; \mathbf{x})$
- Typically, input $\mathbf{x}$ is given and network weights $\mathbf{w}$ optimized
- Could also freeze weights $\mathbf{w}$ and optimize $\mathbf{x}$, adversarially!

$$\min_{\boldsymbol{\delta}} \ \text{size}(\boldsymbol{\delta}) \quad \text{s.t.} \quad \text{pred}[f(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta})] \neq \mathrm{y}$$

- More generally: $\max_{\boldsymbol{\delta}} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, \mathrm{y}) \quad \text{s.t.} \quad \text{size}(\boldsymbol{\delta}) \leq \epsilon$

Adversarial Attacks

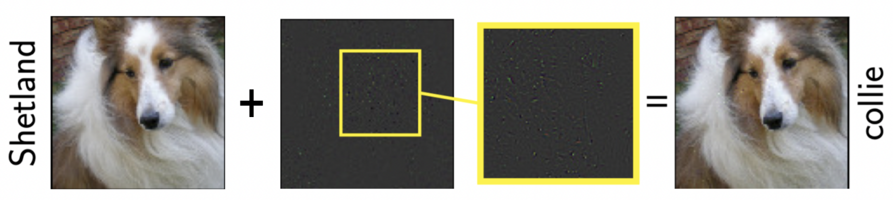


- Mathematically, a neural network is a function $f(\mathbf{w}; \mathbf{x})$
- Typically, input $\mathbf{x}$ is given and network weights $\mathbf{w}$ optimized
- Could also freeze weights $\mathbf{w}$ and optimize $\mathbf{x}$, adversarially!

$$\min_{\boldsymbol{\delta}} \ \text{size}(\boldsymbol{\delta}) \quad \text{s.t.} \quad \text{pred}[f(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta})] \neq \mathbf{y}$$

- More generally: $\max_{\boldsymbol{\delta}} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, \mathbf{y}) \quad \text{s.t.} \quad \text{size}(\boldsymbol{\delta}) \leq \epsilon$

Adversarial Attacks

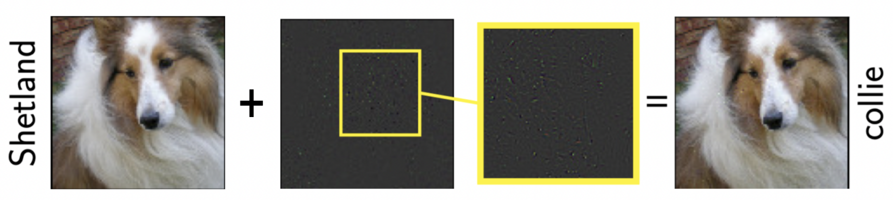


- Mathematically, a neural network is a function $f(\mathbf{w}; \mathbf{x})$
- Typically, input $\mathbf{x}$ is given and network weights $\mathbf{w}$ optimized
- Could also freeze weights $\mathbf{w}$ and optimize $\mathbf{x}$, adversarially!

$$\min_{\boldsymbol{\delta}} \ \mathrm{size}(\boldsymbol{\delta}) \quad \text{s.t.} \quad \mathrm{pred}[f(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta})] \neq \mathrm{y}$$

- More generally: $\max_{\boldsymbol{\delta}} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, \mathrm{y}) \quad \text{s.t.} \quad \mathrm{size}(\boldsymbol{\delta}) \leq \epsilon$

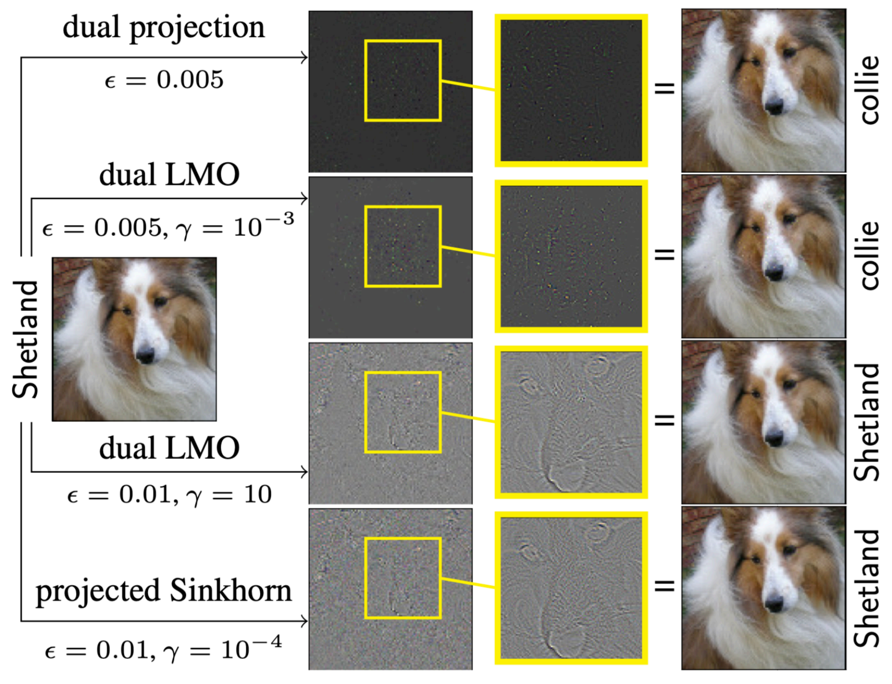
Adversarial Attacks



- Mathematically, a neural network is a function $f(\mathbf{w}; \mathbf{x})$
- Typically, input $\mathbf{x}$ is given and network weights $\mathbf{w}$ optimized
- Could also freeze weights $\mathbf{w}$ and optimize $\mathbf{x}$, adversarially!

$$\min_{\boldsymbol{\delta}} \ \mathrm{size}(\boldsymbol{\delta}) \quad \text{s.t.} \quad \mathrm{pred}[f(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta})] \neq \mathrm{y}$$

- More generally: $\max_{\boldsymbol{\delta}} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, \mathrm{y}) \quad \text{s.t.} \quad \mathrm{size}(\boldsymbol{\delta}) \leq \epsilon$



dual projection
$\epsilon = 0.005$
dual LMO
$\epsilon = 0.005, \gamma = 10^{-3}$
Shetland
dual LMO
$\epsilon = 0.01, \gamma = 10$
projected Sinkhorn
$\epsilon = 0.01, \gamma = 10^{-4}$
=
collie
=
collie
=
Shetland
=
Shetland

Robustness as Optimization

- Empirical risk minimization recalled:

$$\min_{\mathbf{w}} \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x}, y)$$

- Adversarial attack perturbs $(\mathbf{x}, y)$ while fixing $\mathbf{w}$:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- Robustness by anticipating the worst-case:

$$\min_{\mathbf{w}} \hat{\mathsf{E}} \max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- The game continues by anticipating the anticipation:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{leader}$$

$$\min_{\mathbf{w}} \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{follower}$$

Robustness as Optimization

- Empirical risk minimization recalled:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x}, y)$$

- Adversarial attack perturbs $(\mathbf{x}, y)$ while fixing $\mathbf{w}$:

$$\max_{\text{size}(\boldsymbol{\delta}) \le \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- Robustness by anticipating the worst-case:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}} \max_{\text{size}(\boldsymbol{\delta}) \le \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- The game continues by anticipating the anticipation:

$$\max_{\text{size}(\boldsymbol{\delta}) \le \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{leader}$$

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{follower}$$

Robustness as Optimization

- Empirical risk minimization recalled:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x}, y)$$

- Adversarial attack perturbs $(\mathbf{x}, y)$ while fixing $\mathbf{w}$:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- Robustness by anticipating the worst-case:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}} \max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- The game continues by anticipating the anticipation:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \le \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{leader}$$

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{follower}$$

Robustness as Optimization

- Empirical risk minimization recalled:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x}, y)$$

- Adversarial attack perturbs $(\mathbf{x}, y)$ while fixing $\mathbf{w}$:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \leq \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- Robustness by anticipating the worst-case:

$$\min_{\mathbf{w}} \ \hat{\mathsf{E}} \max_{\mathrm{size}(\boldsymbol{\delta}) \leq \epsilon} \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y)$$

- The game continues by anticipating the anticipation:

$$\max_{\mathrm{size}(\boldsymbol{\delta}) \leq \epsilon} \ \ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{leader}$$

$$\min_{\mathbf{w}} \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{x} + \boldsymbol{\delta}, y) \qquad \text{follower}$$

Plan II: Game-theoretic

- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Plan II: Game-theoretic

- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Plan II: Game-theoretic

- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Plan II: Game-theoretic

- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Plan II: Game-theoretic

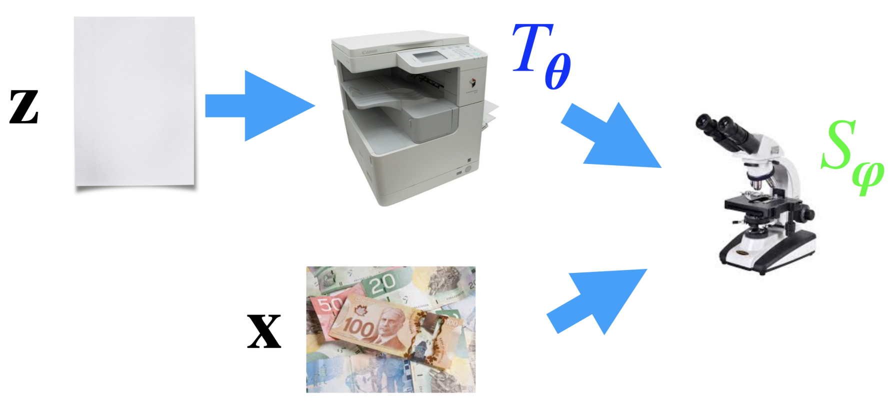
- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Plan II: Game-theoretic

- Lec07: Fictitious Play: playing against oneself
- Lec14: Minimax: understanding duality
- Lec15: Averaging: projected gradient descent ascent
- Lec16: Extragradient: faster under smoothness
- Lec17: Splitting: exploiting structure
- Lec20: Randomized Smoothing: simulating gradient

Generative Adversarial Networks

$$\min_{\theta} \max_{\varphi} \hat{\mathsf{E}} \log S_{\varphi}(\mathbf{x}) + \hat{\mathsf{E}} \log(1 - S_{\varphi} \circ T_{\theta}(\mathbf{z}))$$



I. Goodfellow et al. (2014). "Generative Adversarial Nets". In: *NIPS*.

Generative Adversarial Networks

$$\min_{\theta} \max_{\varphi} \hat{\mathsf{E}} \log S_{\varphi}(\mathbf{x}) + \hat{\mathsf{E}} \log(1 - S_{\varphi} \circ T_{\theta}(\mathbf{z}))$$

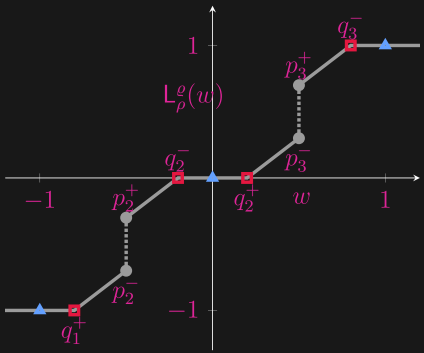
I. Goodfellow et al. (2014). "Generative Adversarial Nets". In: *NIPS*.


Parti


TheJasper
Whisperer.
com

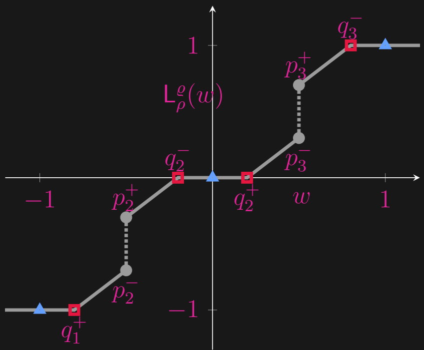
Parti

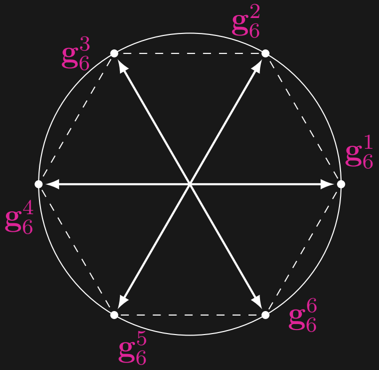
Plan III: Exotic



- Lec06: Conditional Gradient: model weights quantization
- Lec09: Metric Gradient: model gradient quantization
- Lec08: Mirror Descent: gradient under non-Euclidean geometry

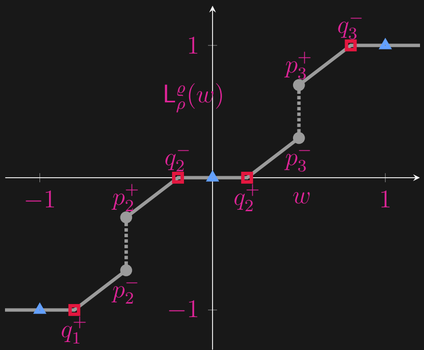
Plan III: Exotic

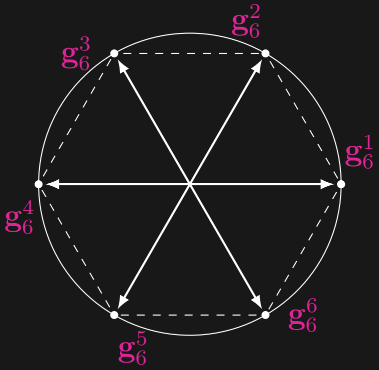




- Lec06: Conditional Gradient: model weights quantization
- Lec09: Metric Gradient: model gradient quantization
- Lec08: Mirror Descent: gradient under non-Euclidean geometry

Plan III: Exotic





- Lec06: Conditional Gradient: model weights quantization
- Lec09: Metric Gradient: model gradient quantization
- Lec08: Mirror Descent: gradient under non-Euclidean geometry

History Goes A Long Way Back

"Nothing in the world takes place without optimization, and there is no doubt that all aspects of the world that have a rational basis can be explained by optimization methods."
— *Leonhard Euler, 1744*

"Every year I meet Ph.D. students of different specializations who ask me for advice on reasonable numerical schemes for their optimization models. And very often they seem to have come too late. In my experience, if an optimization model is created without taking into account the abilities of numerical schemes, the chances that it will be possible to find an acceptable numerical solution are close to zero. In any field of human activity, if we create something, we know in advance why we are doing so and what we are going to do with the result." — *Yurii Nesterov*

History Goes A Long Way Back

"Nothing in the world takes place without optimization, and there is no doubt that all aspects of the world that have a rational basis can be explained by optimization methods."
— *Leonhard Euler, 1744*

"Every year I meet Ph.D. students of different specializations who ask me for advice on reasonable numerical schemes for their optimization models. And very often they seem to have come too late. In my experience, if an optimization model is created without taking into account the abilities of numerical schemes, the chances that it will be possible to find an acceptable numerical solution are close to zero. In any field of human activity, if we create something, we know in advance why we are doing so and what we are going to do with the result." — *Yurii Nesterov*

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid answers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no stupid questions, only stupid students."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no inferior algorithms, only inferior engineers."

[1] D. H. Wolpert and W. G. Macready (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82.

[2] K. G. Murty and S. N. Kabadi (1987). "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2, pp. 117–129.