Neural networks

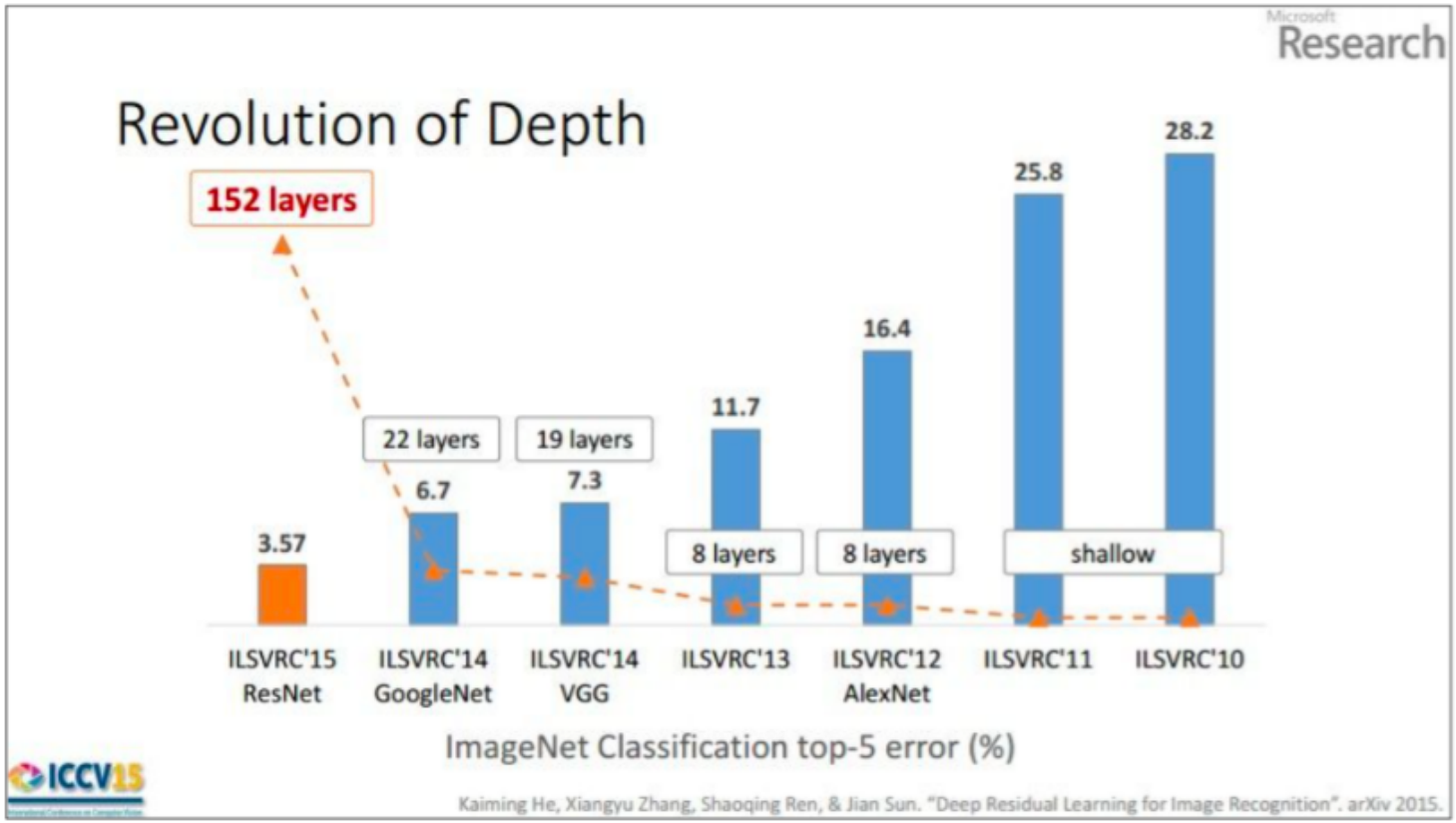
Mladen Kolar (mkolar@chicagobooth.edu)

Example: ImageNet

`http://cs.stanford.edu/people/karpathy/ilsvrc/`

Imagenet Classification

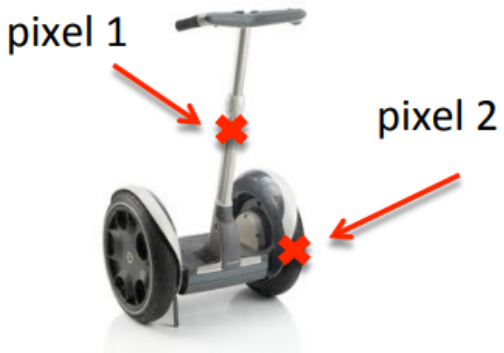
1000 kinds of objects.



(slide from Kaiming He's recent presentation)

2016 error rate is 3.0%

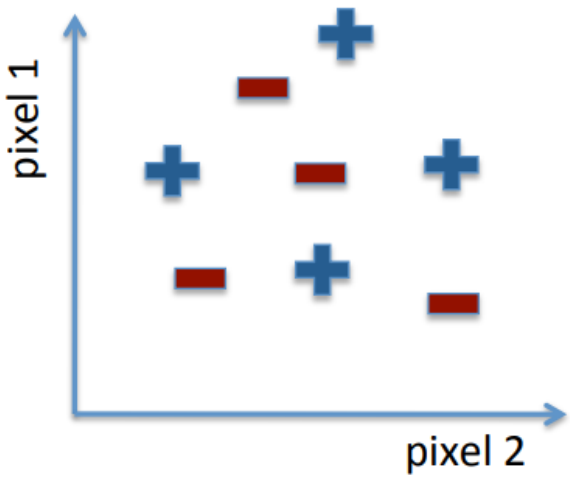
Naive Machine Learning Approach



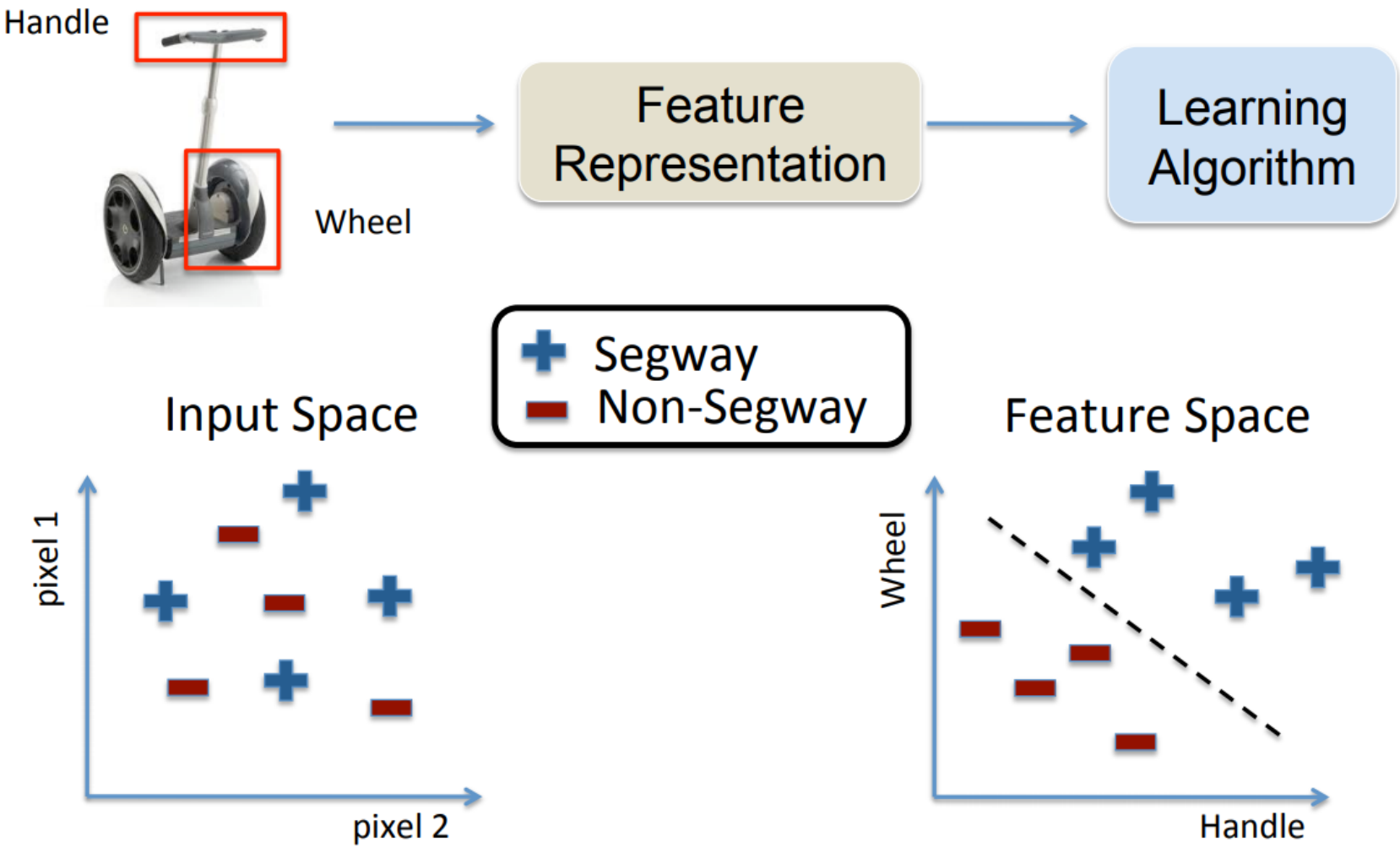
Learning Algorithm

Segway
Non-Segway

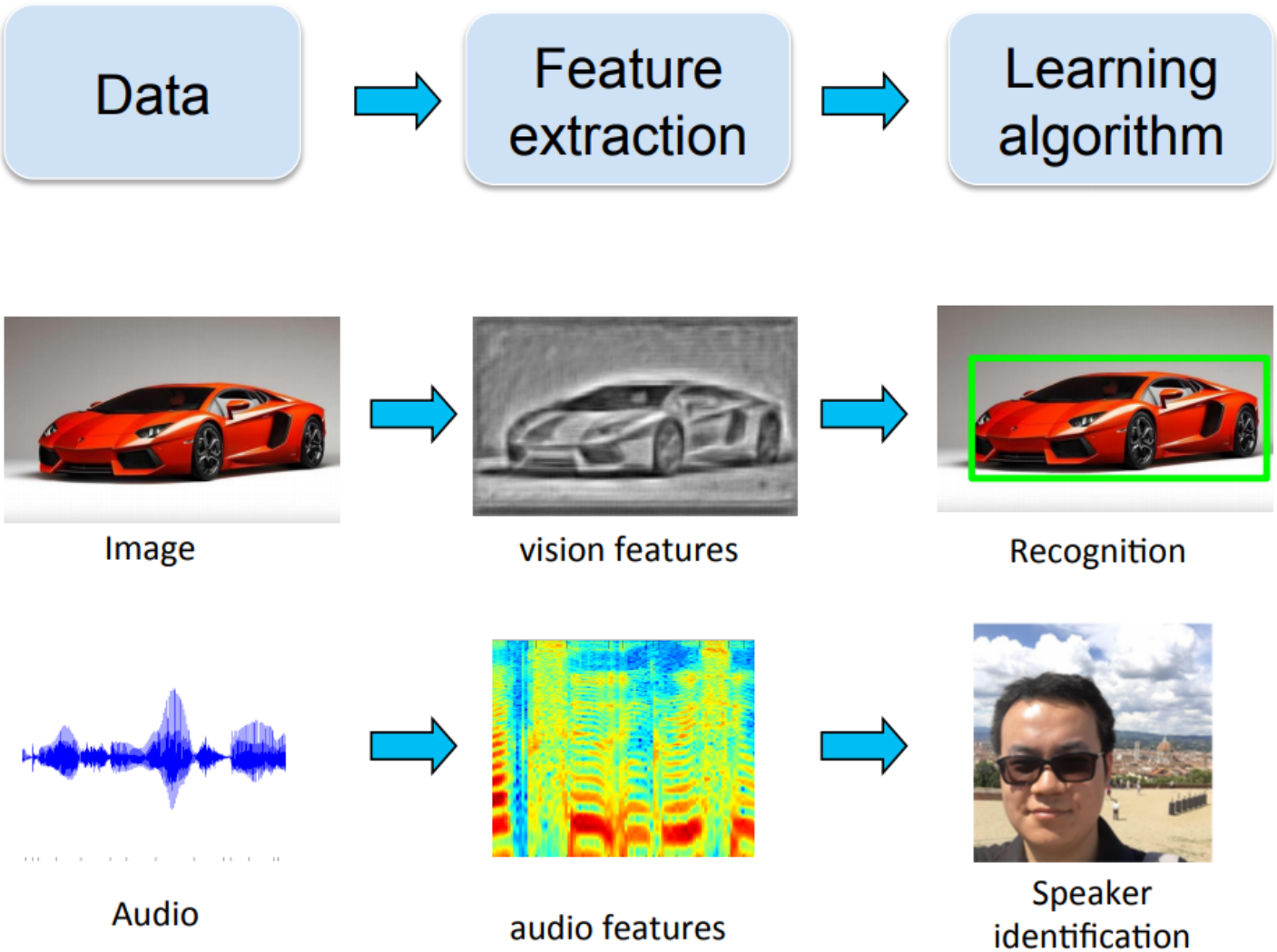
Input Space



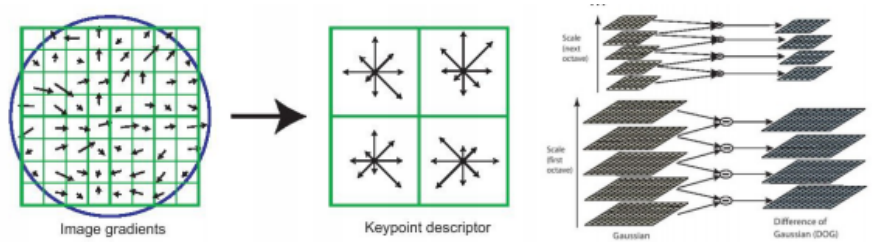
Machine Learning with Engineered Features



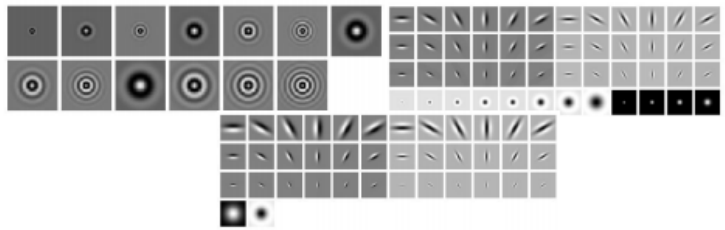
Traditional Machine Learning



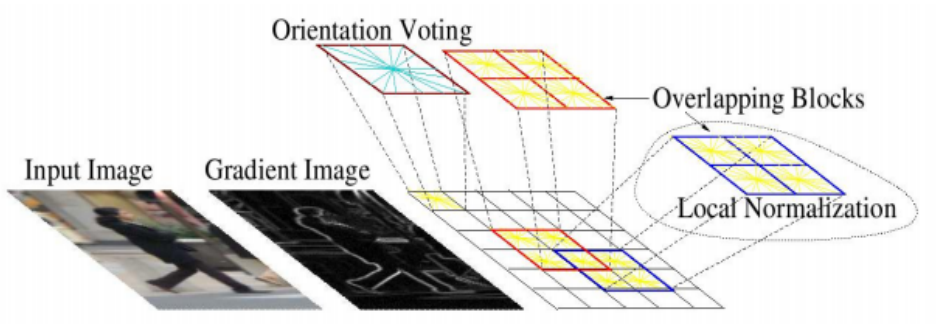
Computer Vision Features



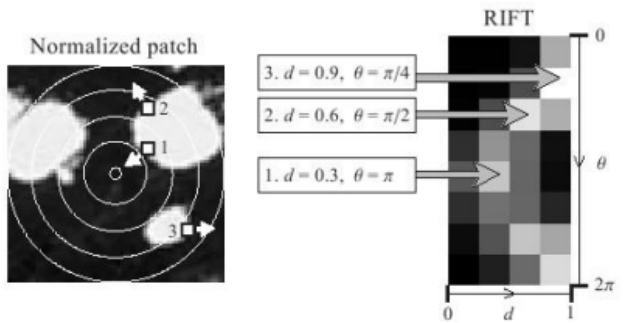
SIFT



Textons

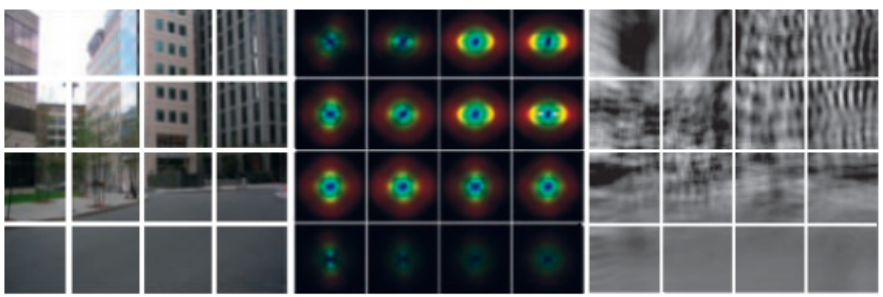


HoG

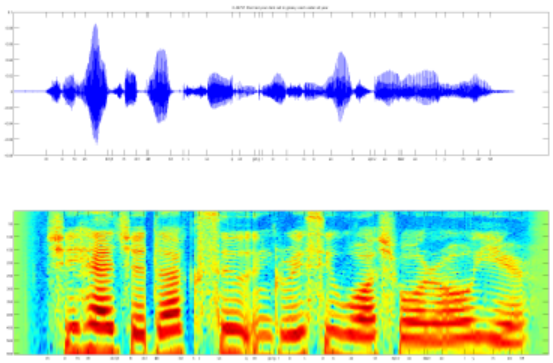


RIFT

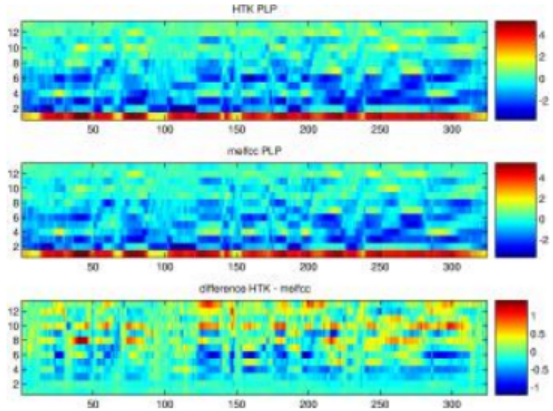
GIST



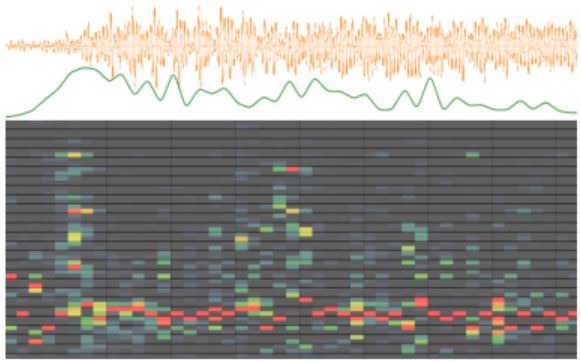
Audio Features



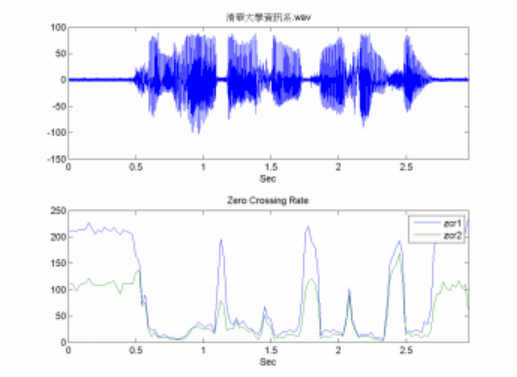
Spectrogram



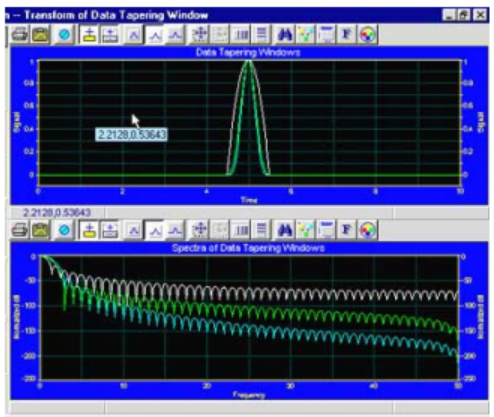
MFCC



Flux



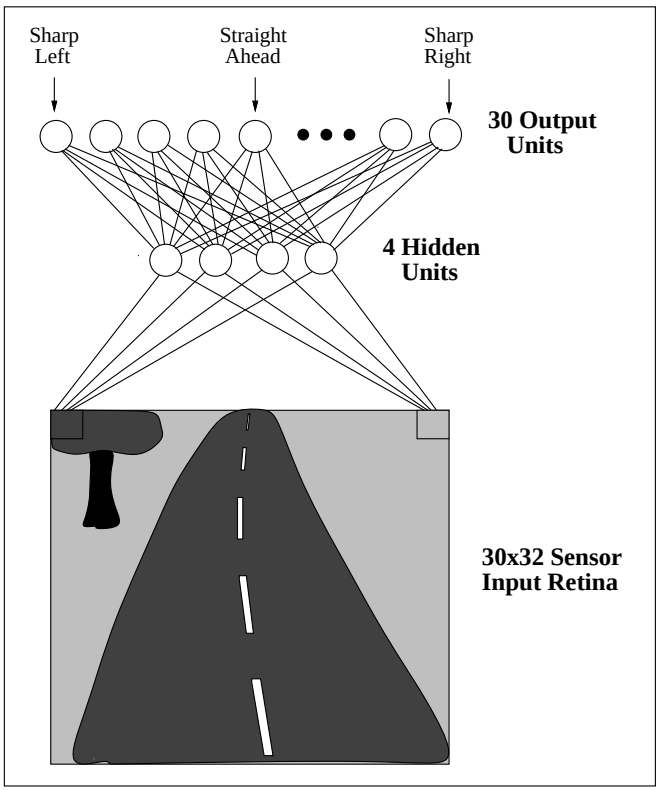
ZCR

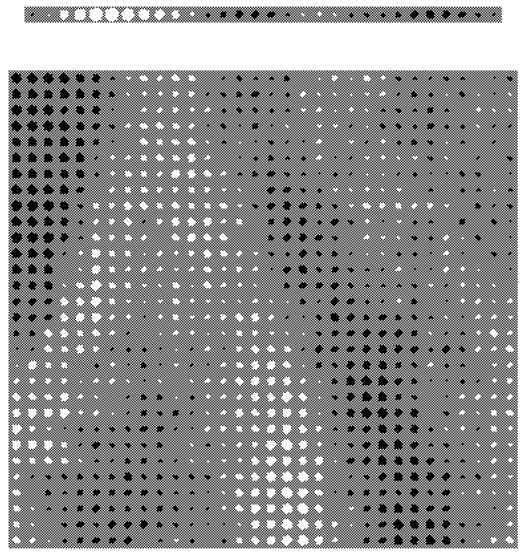


Rolloff

Example: ALVINN

Autonomous Land Vehicle In a Neural Network

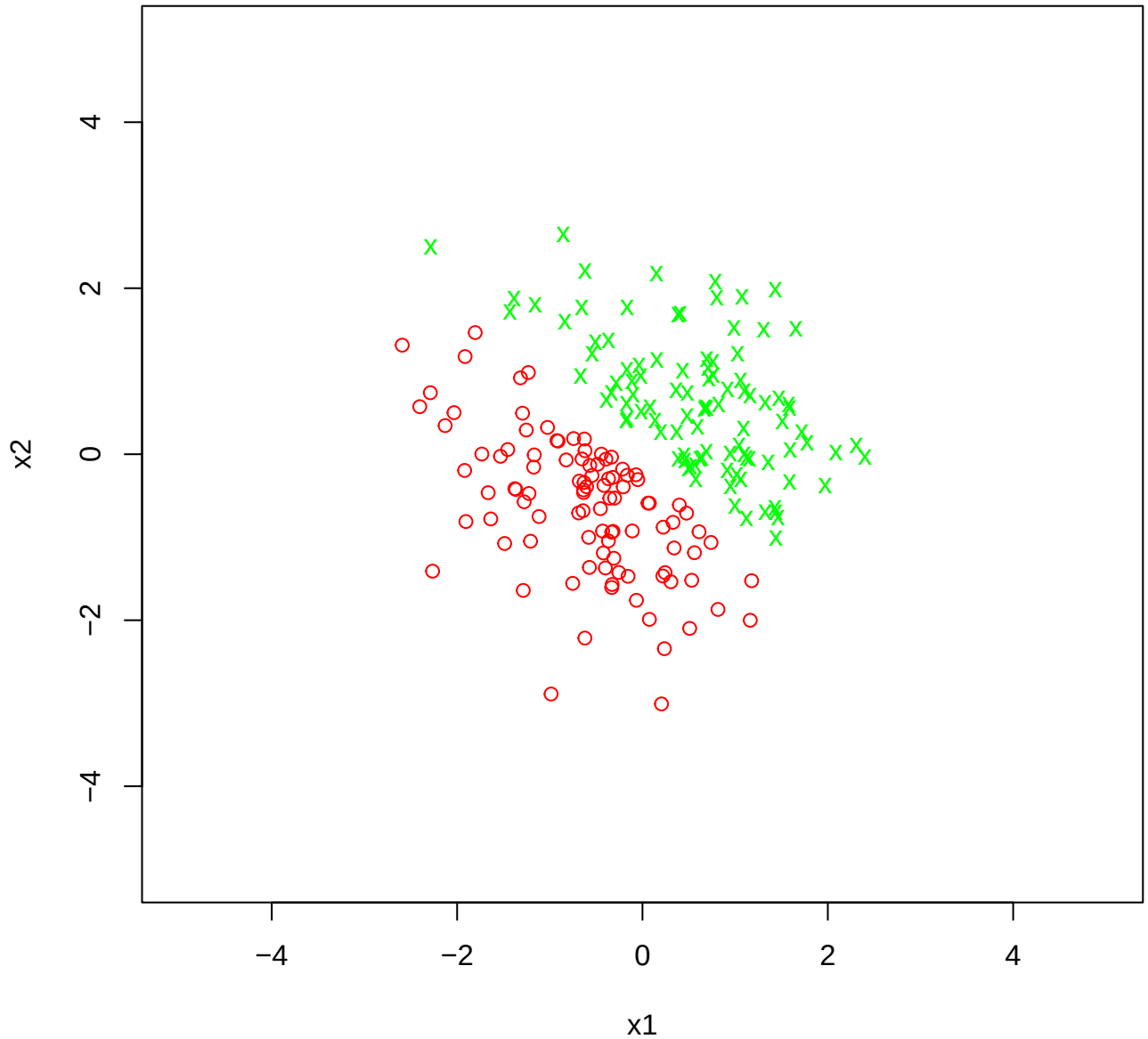




Video: `http://watson.latech.edu/book/intelligence/intelligenceOverview5b4.html`

Perceptron

Linear (Hyperplane) Decision Boundaries



Learning a Linear Classifier

We want to learn a mapping $f : X \mapsto Y$

- X are features (input variables)
- Y is the target class ($+1$ or -1)

Decision rule: Is the new observation x_f on one side of the hyperplane or the other

$$\hat{\beta}_0 + \hat{\beta} \cdot x_f \geq 0?$$

$$\hat{y}_0 = \text{sign}(\hat{\beta}_0 + \hat{\beta} \cdot x_f)$$

Perceptron

1958 New York Times...

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.

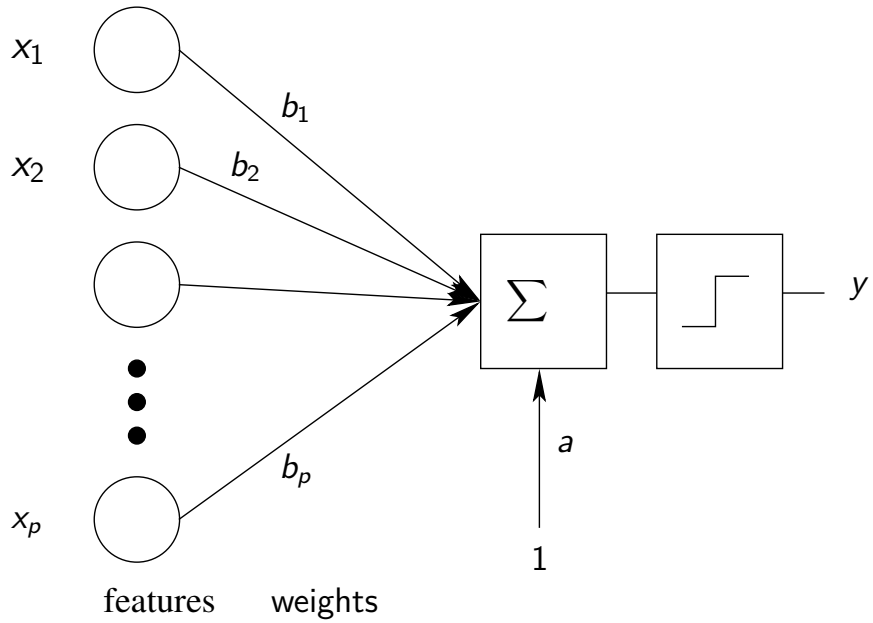
Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

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.

. . . first device to think as the human brain. . . Perceptron will make mistakes at first, but will grow wiser as it gains experience.

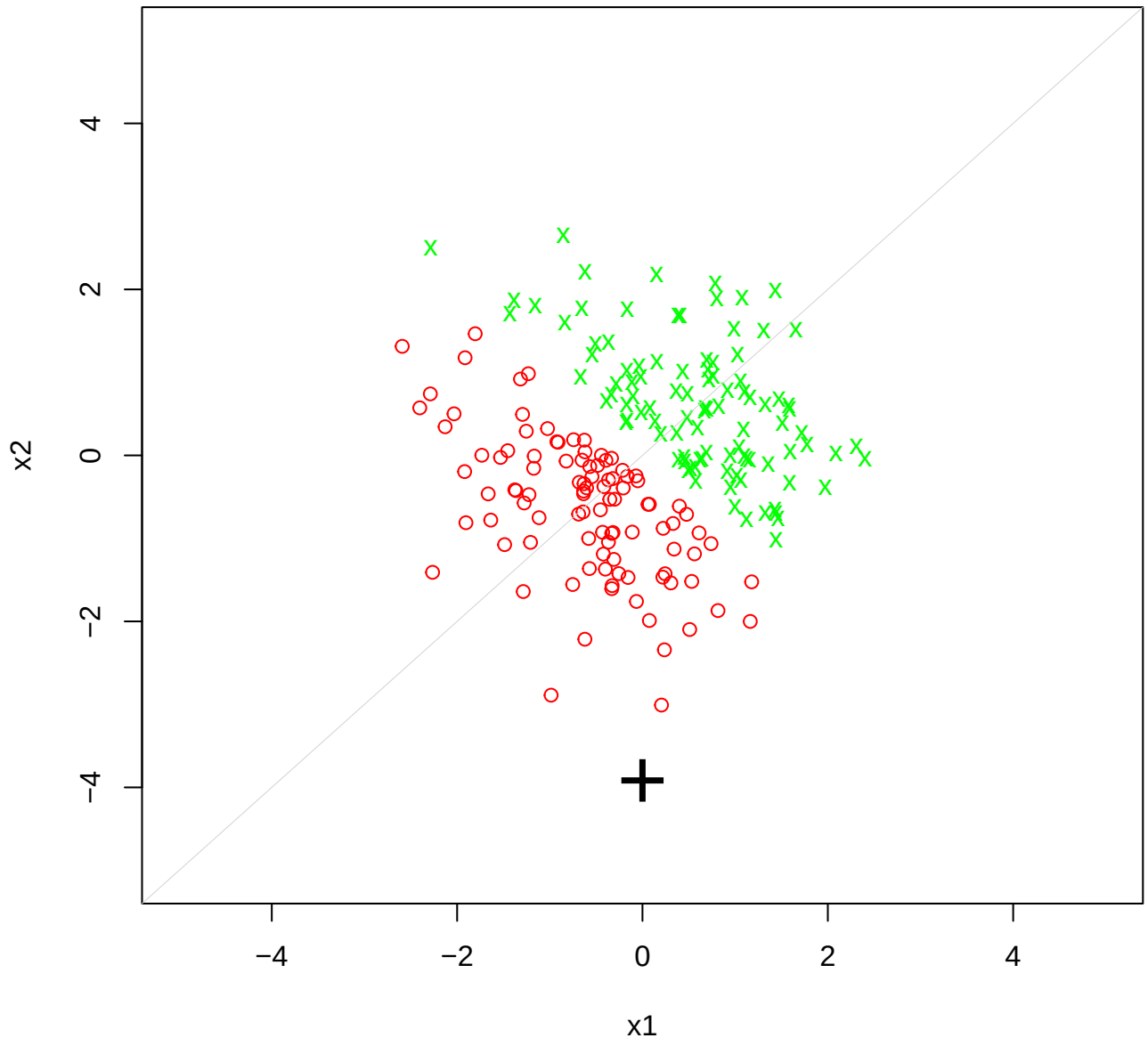
Frank Rosenblatt

Represenation with perceptron

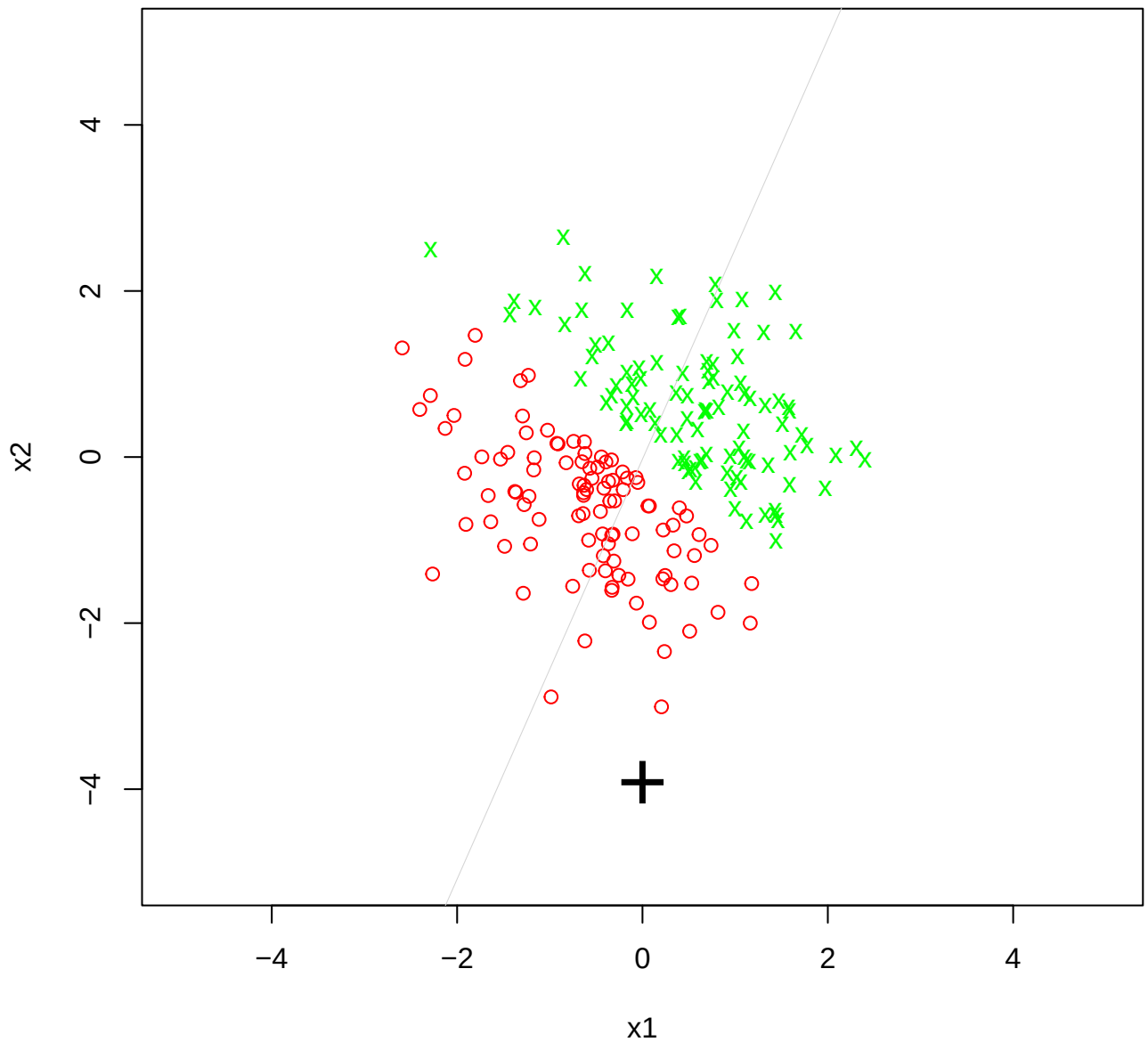


$$\text{Predict} \begin{cases} y = 1 & \text{if } a + \sum_{j=1}^{p} b_j x_j \geq 0 \\ y = 0 & \text{if } a + \sum_{j=1}^{p} b_j x_j < 0 \end{cases}$$

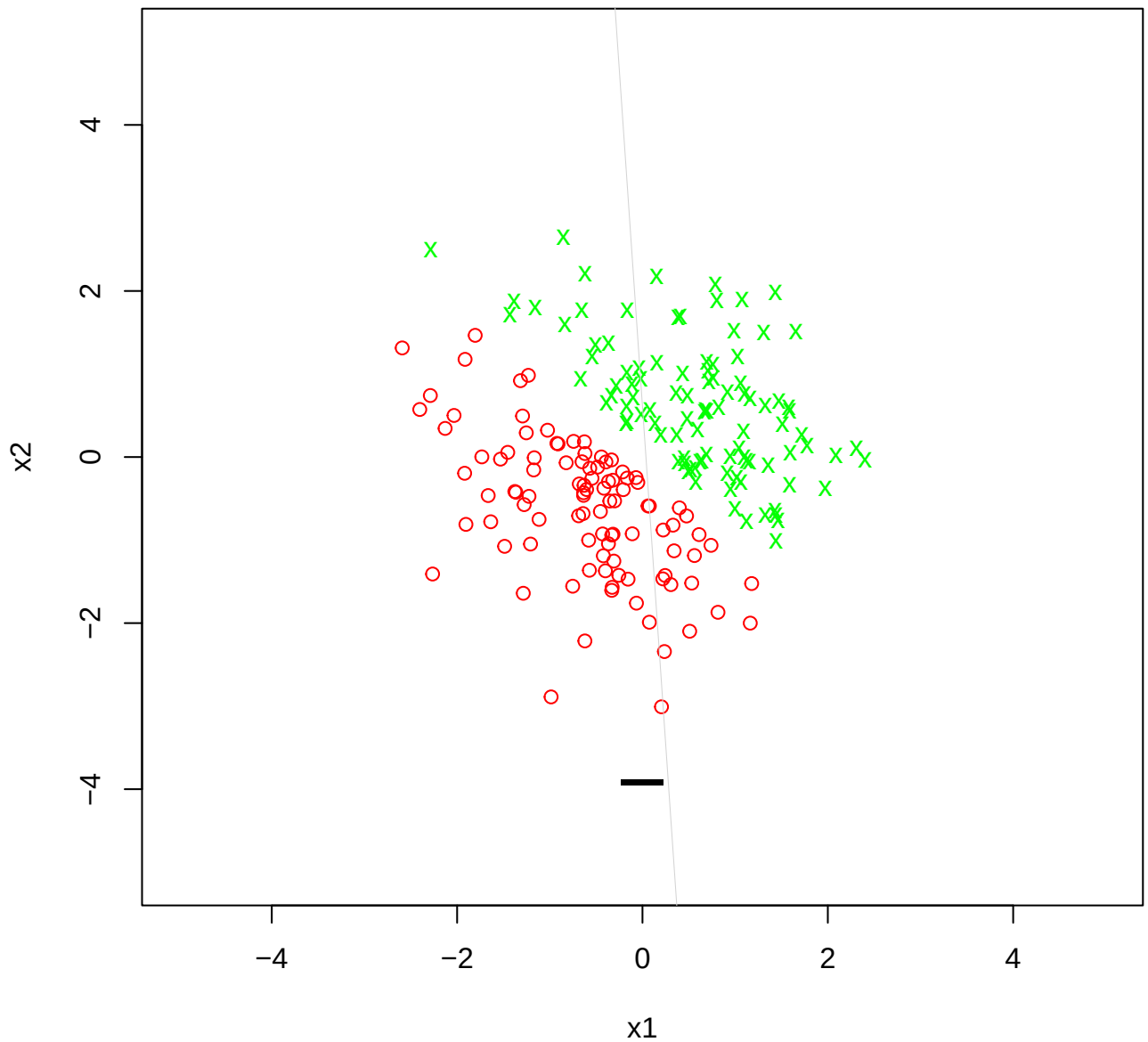
Model $f(x) = \text{sign}(a + \sum_{j=1}^{p} b_j x_j)$


x2
4
2
0
−2
−4
−4
−2
0
2
4
x1

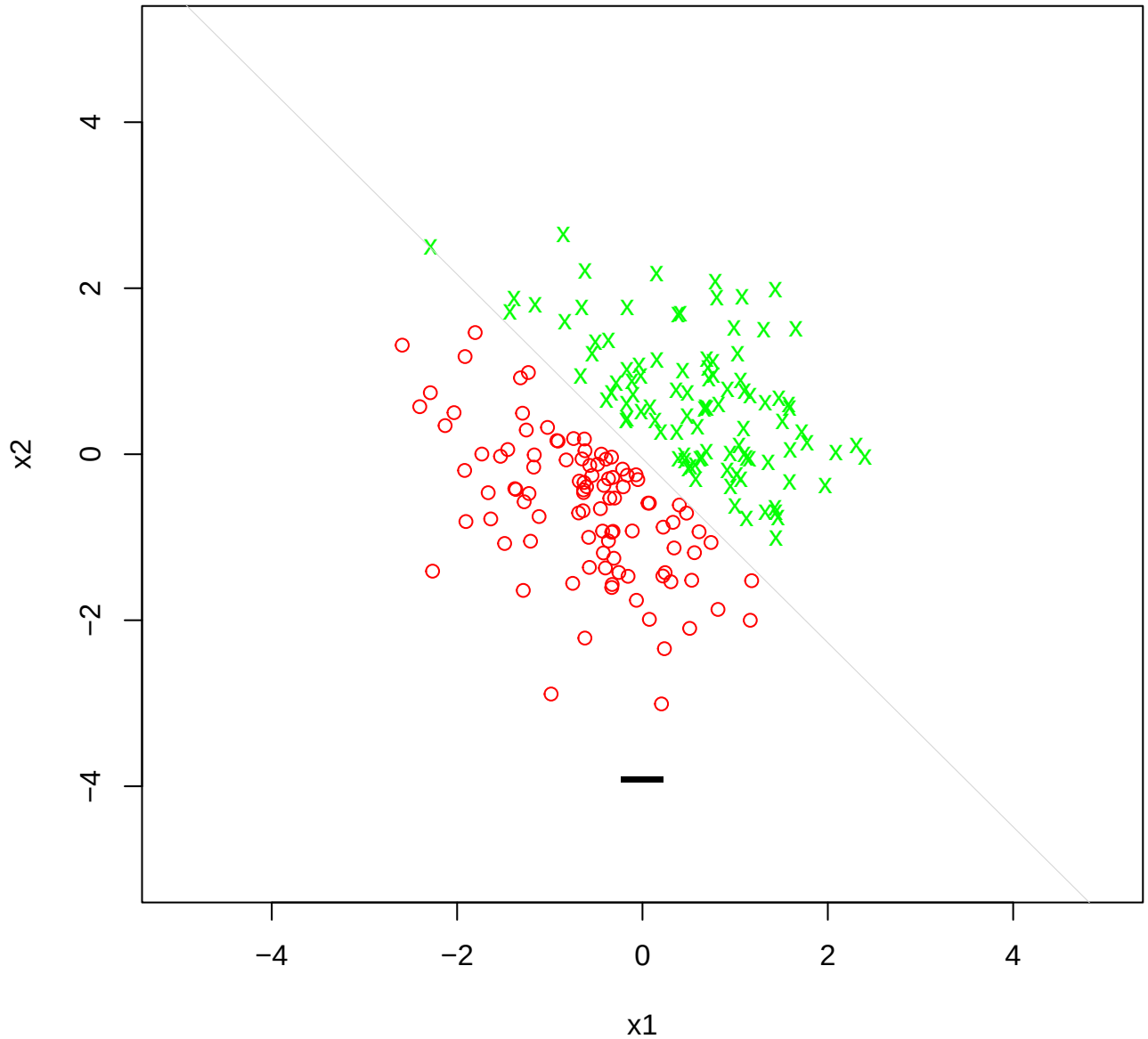
Iteration 1



Iteration 2



Iteration 10



Perceptron Algorithm

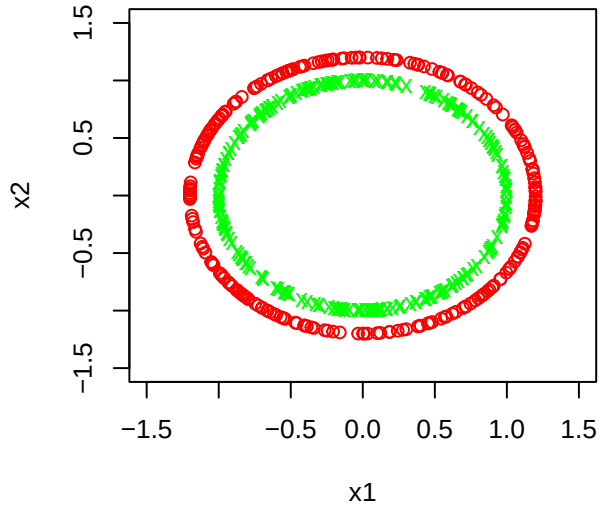
Fit a linear classifier β_0, β to the training set using **stochastic gradient descent**.

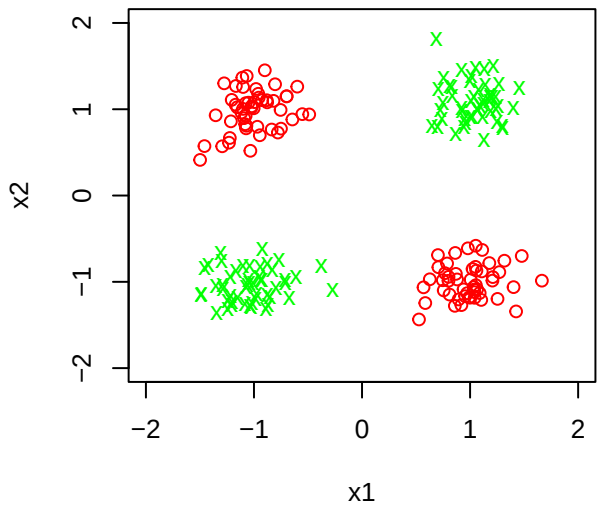
- update β_0, β based on just one data point (x, y) at a time
- If $y(\beta_0 + \beta \cdot x) > 0$: zero loss, no update
- If $y(\beta_0 + \beta \cdot x) < 0$: loss is $-y(\beta_0 + \beta \cdot x)$

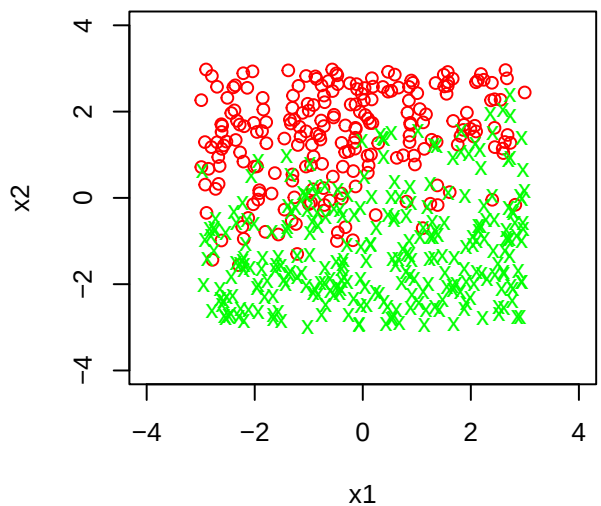
Algorithm

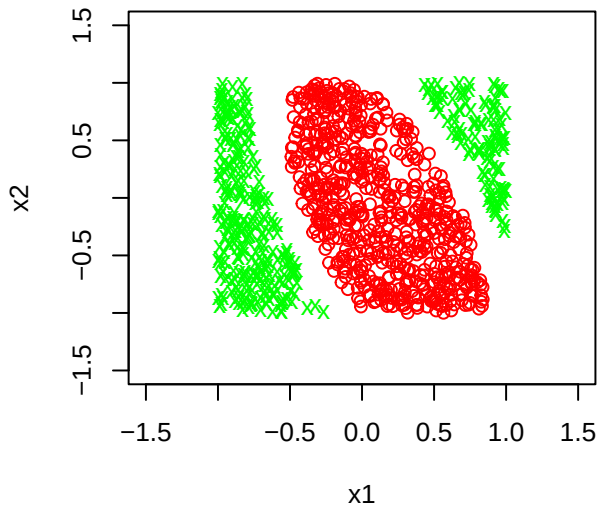
- Initialize $\beta_0 = 0$ and $\beta = 0$
- Keep cycling through the training data (x, y):
  - If $y(\beta_0 + \beta \cdot x) < 0$ (that is, point is incorrectly classified)
    - $\beta = \beta + yx$
    - $\beta_0 = \beta_0 + y$

Data not-linearly separable





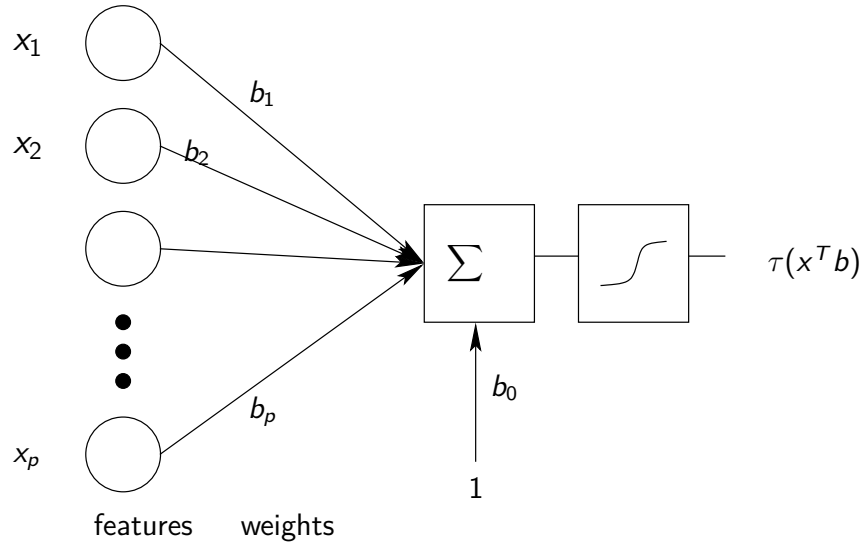




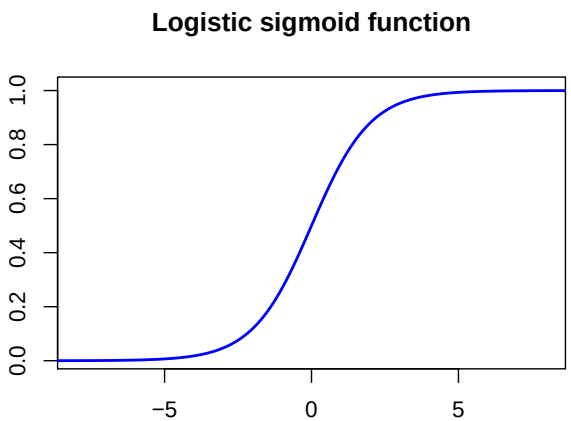
Perceptron properties

- Works when data is linearly separable.
- We can add non-linear features to make the problem separable
- Can be extended to multiple class classification
- Does not assign probability to our prediction

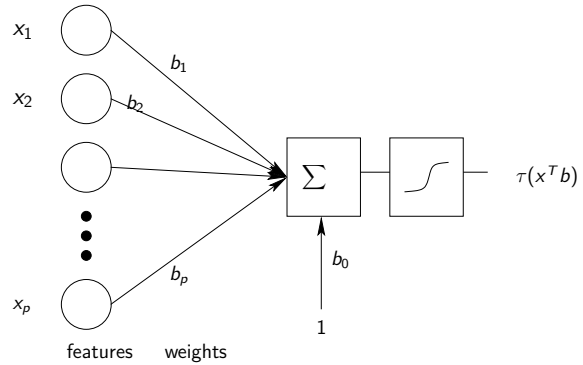
Model of a neuron



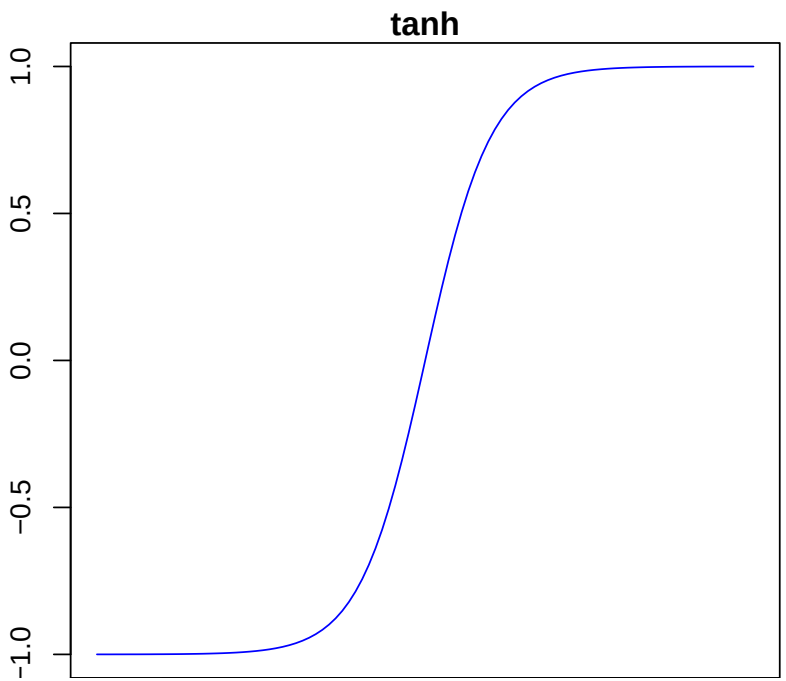
A close cousin to perceptron. Think about putting a rug over the threshold.

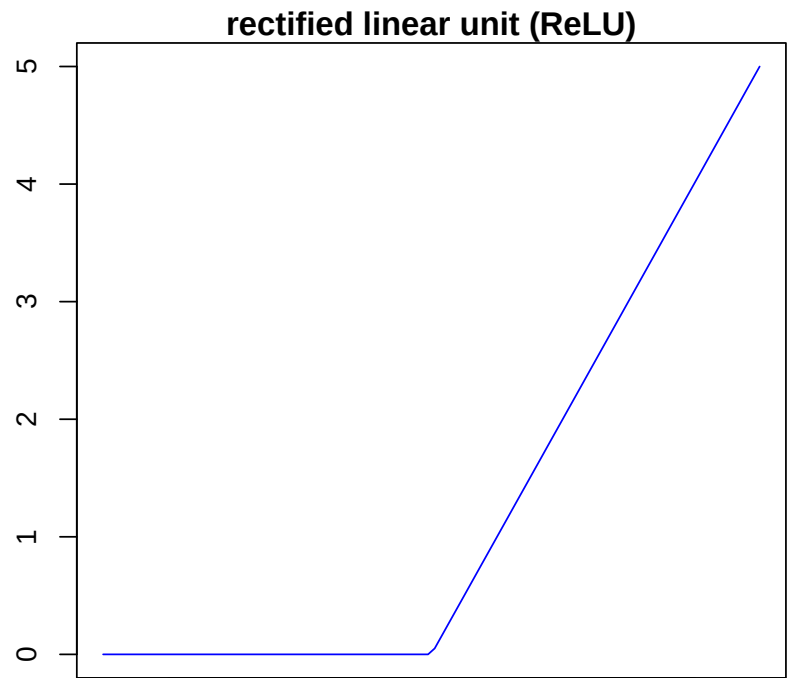


Model of a neuron

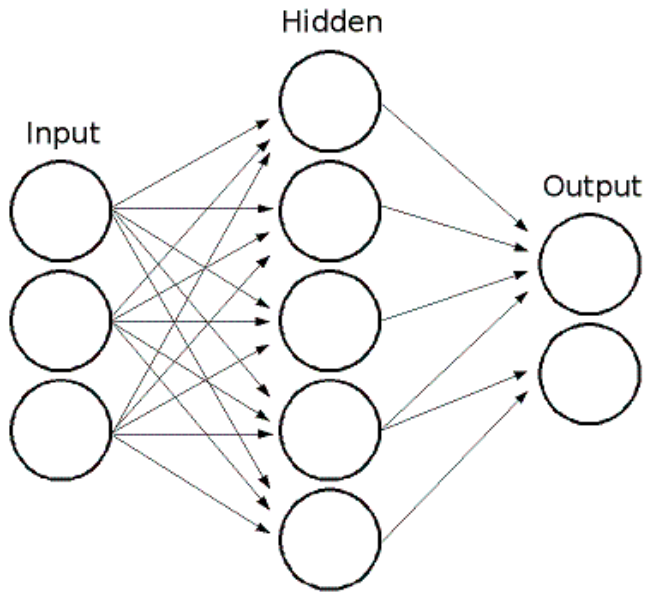


Other nonlinear activations





Multilayer Perceptron



2 Layers of Neurons

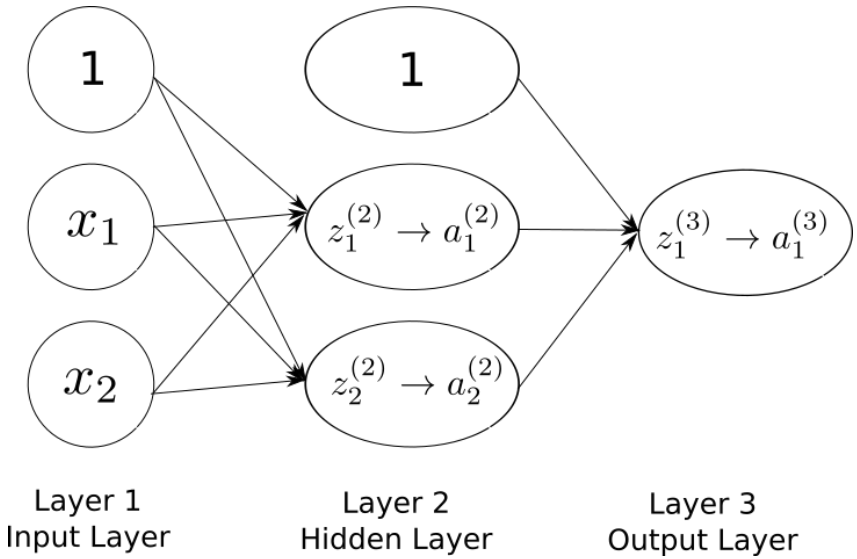
- 1st layer takes input x
- 2nd layer takes output of 1st layer
- The last layer is the output

Multilayer Perceptron

The activities of the neurons in each layer are a non-linear function of the activities in the layer below.
Can approximate arbitrary functions

- Provided hidden layer is large enough
- "fat" 2-layer network

1 hidden layer details



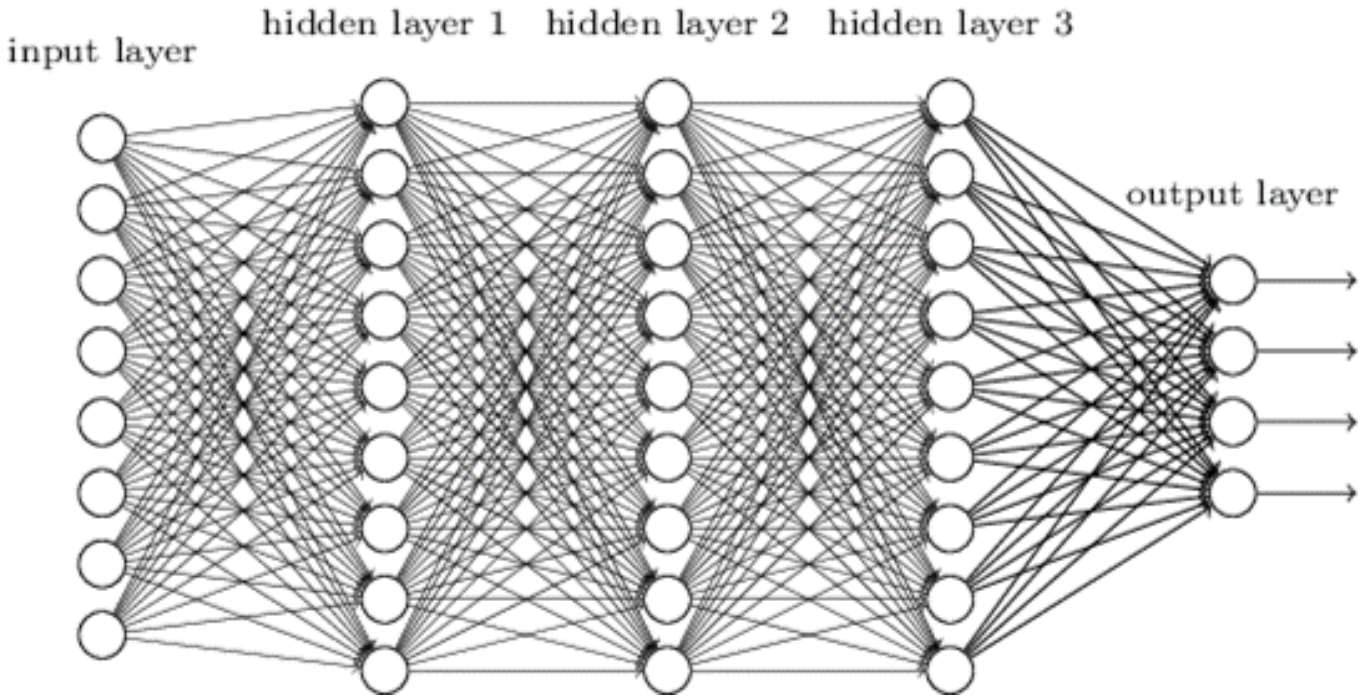
$$z_1^{(2)} = b_{10}^{(1)} + b_{11}^{(1)} x_1 + b_{12}^{(1)} x_2 \quad \longrightarrow \quad a_1^{(2)} = g(z_1^{(2)})$$
$$z_2^{(2)} = b_{20}^{(1)} + b_{21}^{(1)} x_1 + b_{22}^{(1)} x_2 \quad \longrightarrow \quad a_2^{(2)} = g(z_2^{(2)})$$
$$z_1^{(3)} = b_{10}^{(2)} + b_{11}^{(2)} a_1^{(2)} + b_{12}^{(2)} a_2^{(2)} \quad \longrightarrow \quad a_1^{(3)} = g(z_1^{(3)})$$

Example

Simulated XOR

- See *xor.Rmd*

Deep neural network



If there is more than one hidden layer, networks are called “deep” neural networks.

Optimization — Detour

Optimization in Machine Learning

Often we have some conceptual problem that we want to solve.

We translate this problem into an optimization problem of the form

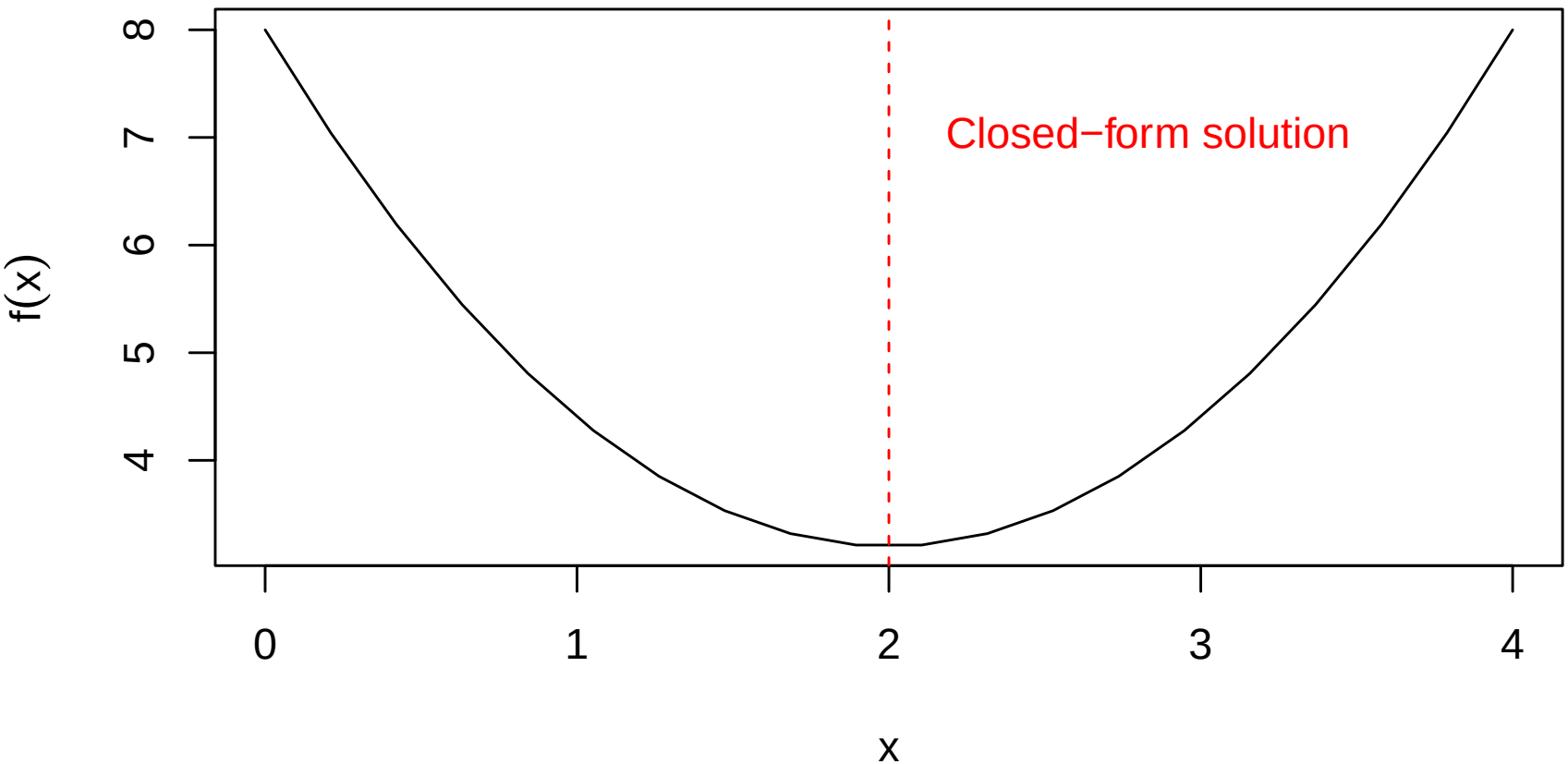
$$\min_{x \in D} \; f(x)$$

What are examples from our lectures?

Gradient descent

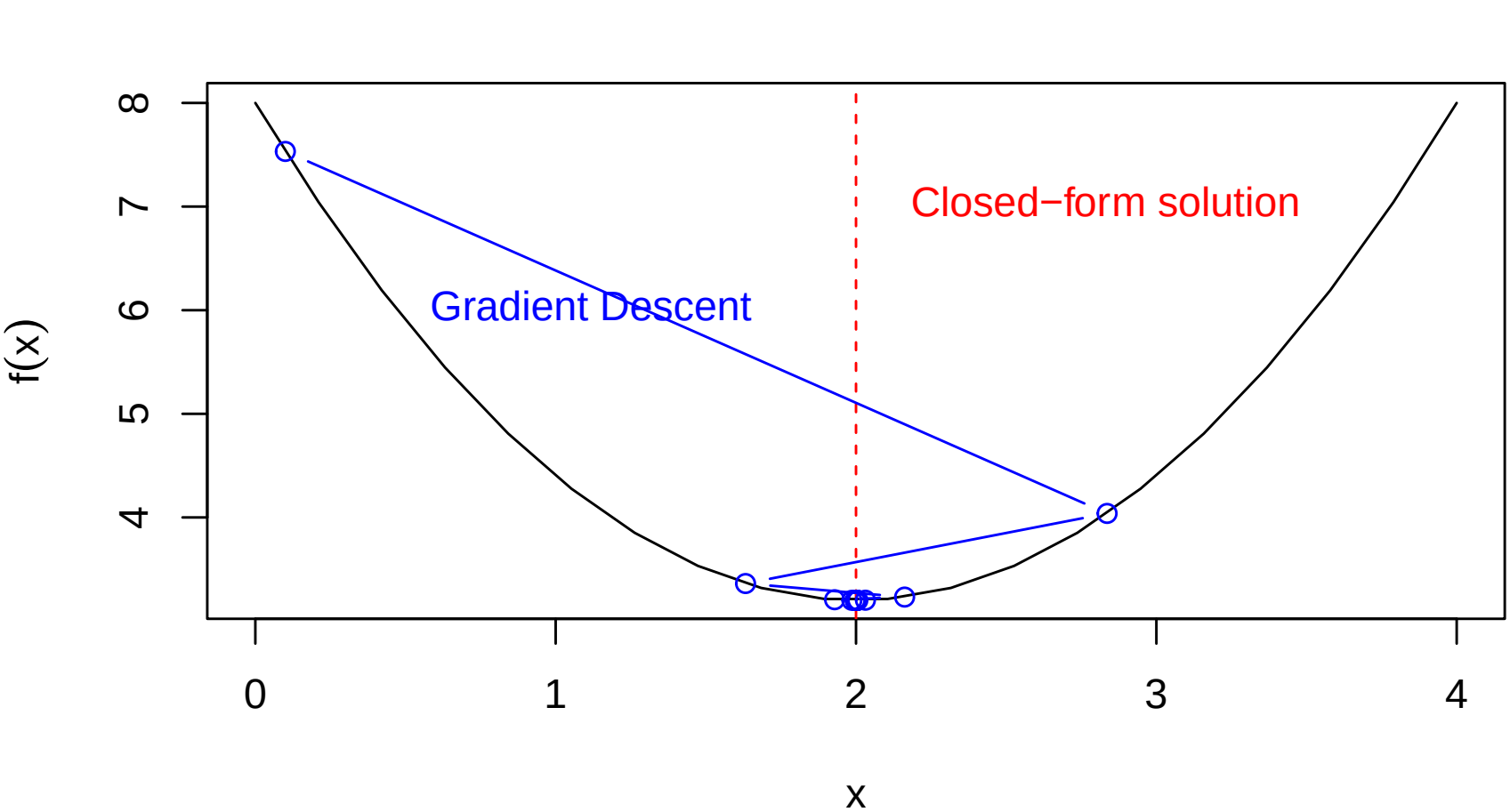
Suppose that we want to find x that minimizes the following function

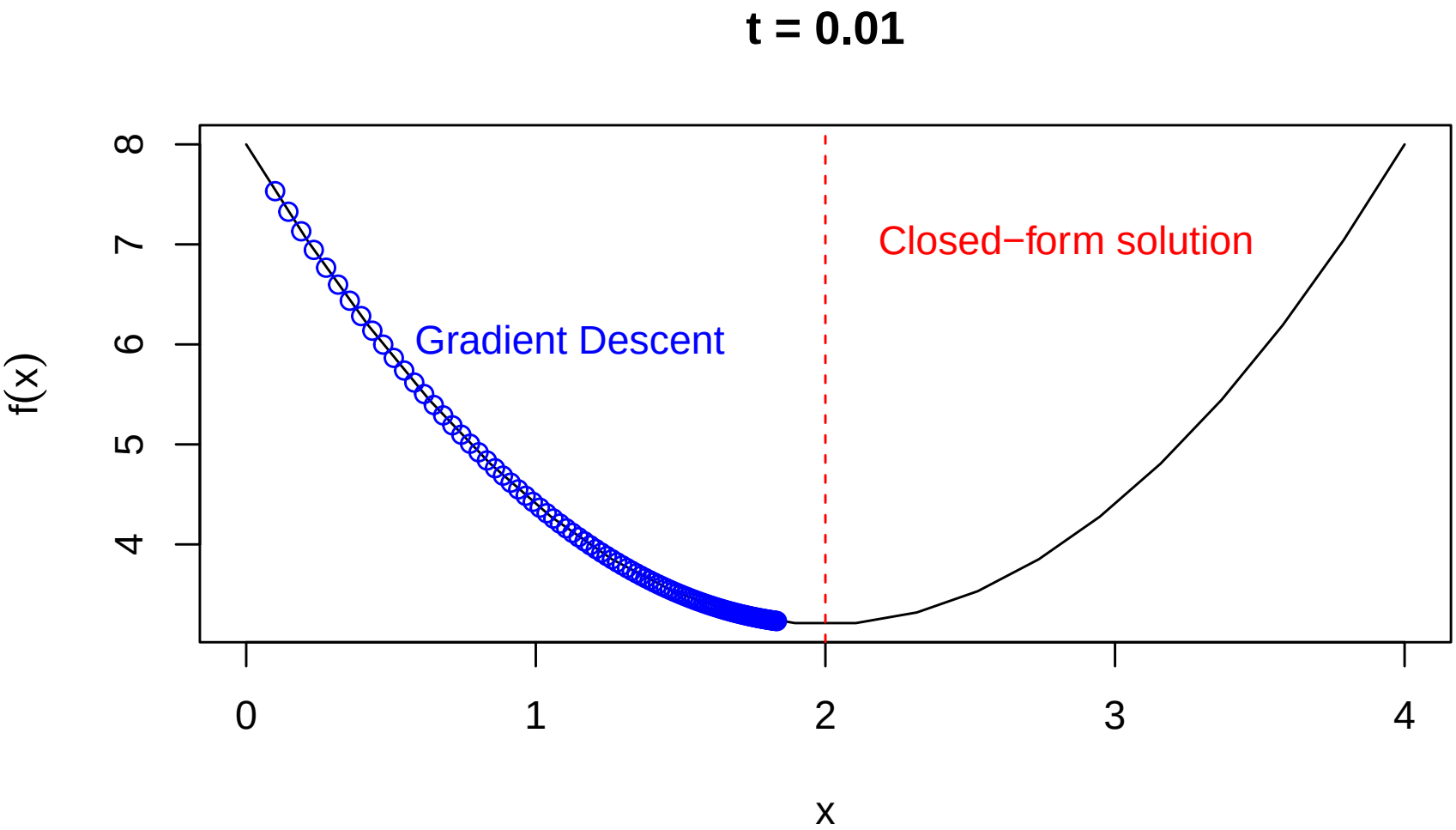
$$f(x) = 1.2(x-2)^2 + 3.2$$

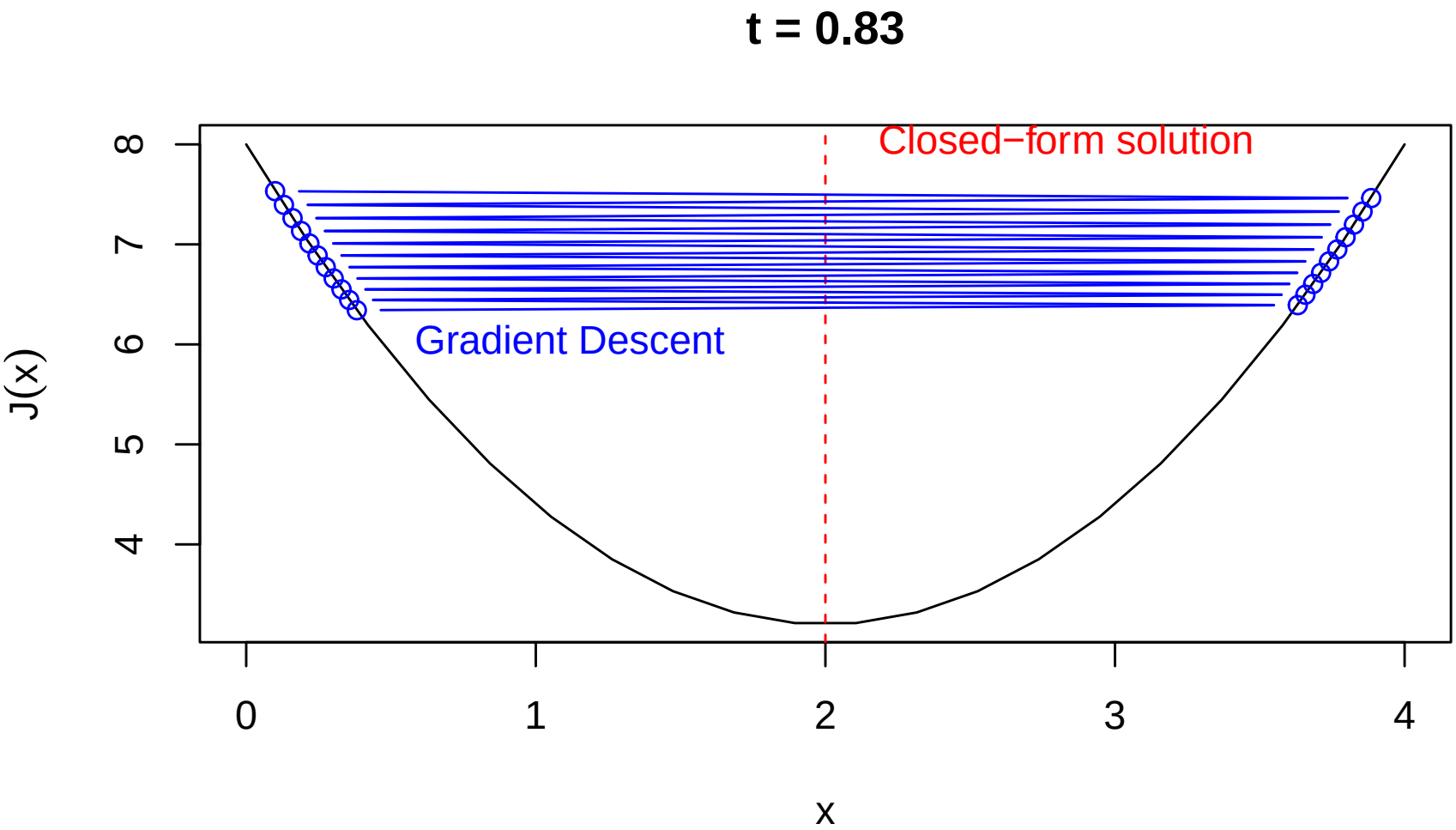


In general, we cannot find a closed form solution, but can compute $\partial f(x)$.

Set $x^0 = 0$. Iteratively update $x^{(k+1)} = x^{(k)} - t \cdot \nabla f(x^{(k)})$.
t is the step size. Also known as the learning rate.




t = 0.01
f(x)
8
7
6
5
4
0
1
2
3
4
x
Gradient Descent
Closed−form solution


t = 0.83
Closed−form solution
Gradient Descent
J(x)
4
5
6
7
8
0
1
2
3
4
x

Gradient descent

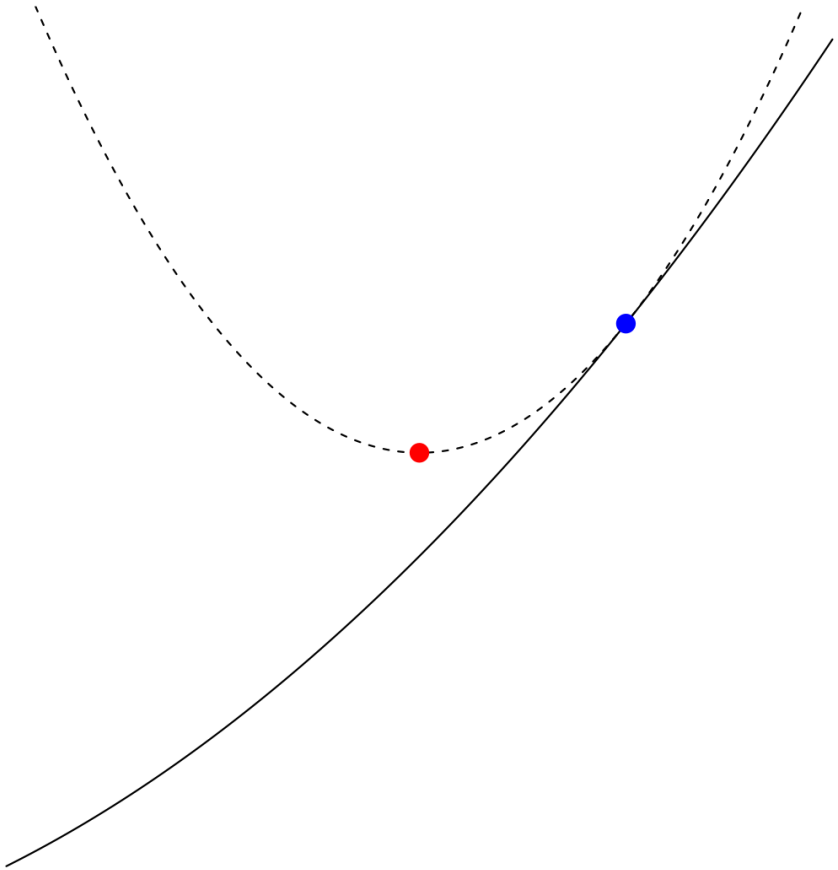
Consider unconstrained, smooth convex optimization

$$\min_x \; f(x)$$

Gradient descent: choose initial point $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f\left(x^{(k)}\right), \quad k = 1, 2, 3, \ldots$$

Stop at some point



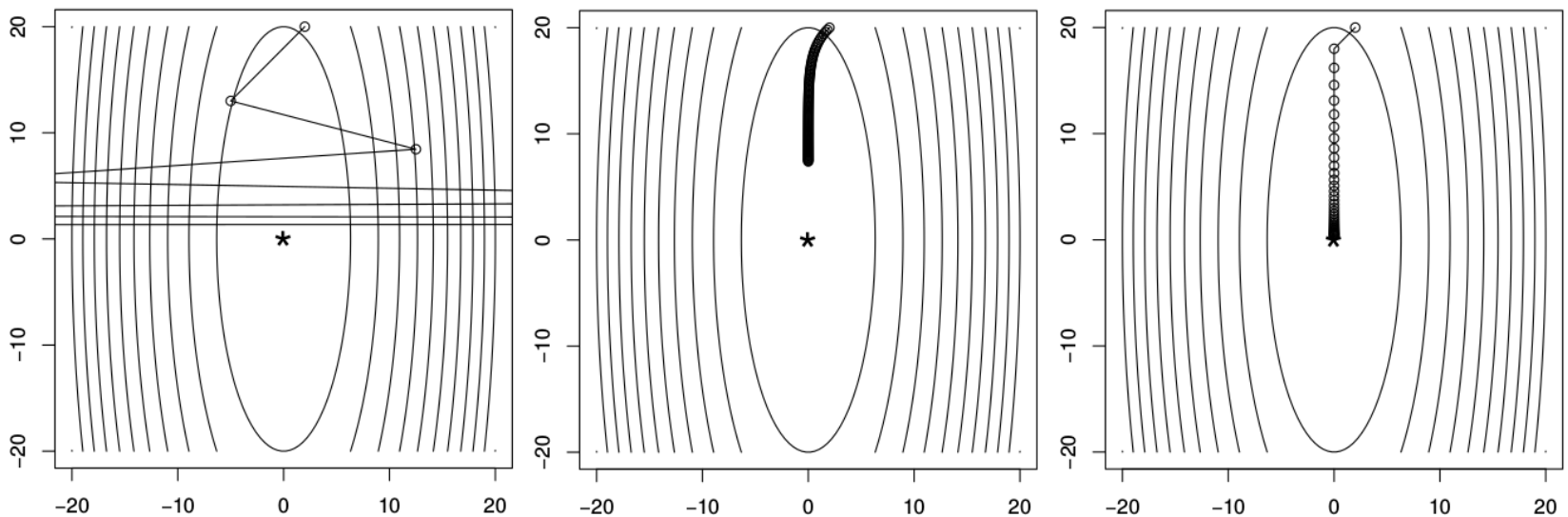
Blue point is x, red point is

$$x^+ = \underset{y}{\operatorname{argmin}} \; f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} \|y - x\|_2^2$$

Choosing an appropriate step size is extremely important!

Fixed step size

- our iterates can diverge if the step is too big
- if the step size is too small, very slow convergence



Backtracking line search DETAILS!

One way to adaptively choose the step size is to use backtracking line search:

- First fix parameters $0 < \beta < 1$ and $0 < \alpha \leq 1/2$
- At each iteration, start with $t = 1$, and while

$$f(x - t\nabla f(x)) > f(x) - \alpha t \|\nabla f(x)\|_2^2$$

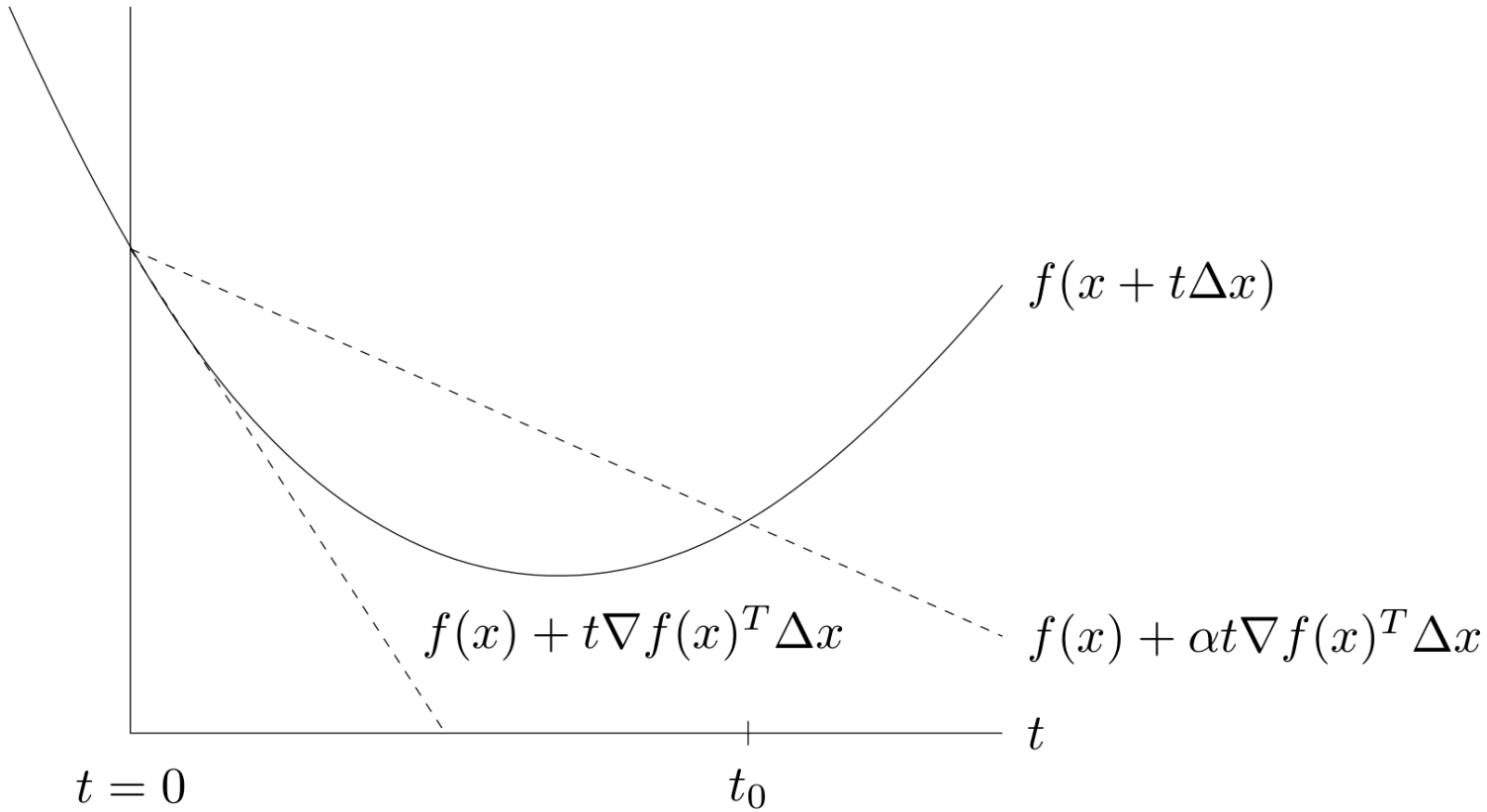
shrink $t = \beta t$. Else perform gradient descent update

$$x^+ = x - t\nabla f(x)$$

Simple and tends to work well in practice (further simplification: just take $\alpha = 1/2$)

Backtracking line search

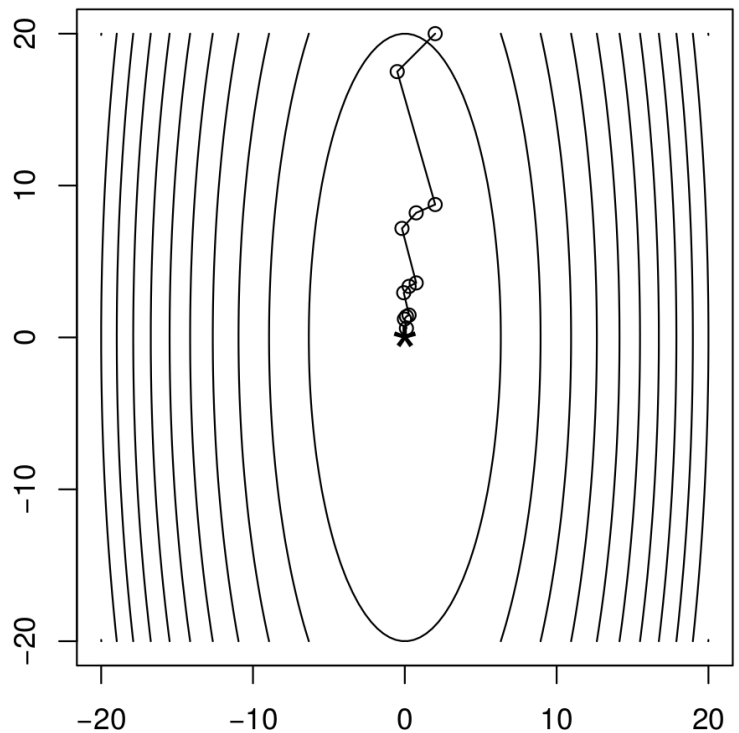
DETAILS!



For us $\Delta x = -\nabla f(x)$

Backtracking line search

Backtracking picks up roughly the right step size (12 outer steps, 40 steps total):



Here $\alpha = \beta = 0.5$

Example: gradient descent for logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$ for $i = 1, \ldots, n$, consider the logistic regression loss:

$$f(\beta) = \sum_{i=1}^{n} \underbrace{\left(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta))\right)}_{f_i(\beta)} = \sum_{i=1}^{n} f_i(\beta)$$

The gradient is

$$\nabla f(\beta) = \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i$$

where

$$\begin{aligned} p_i(\beta) &= P(Y = 1 \mid x_i, \beta) \\ &= \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta)), \quad i = 1, \ldots, n. \end{aligned}$$

Example: gradient descent for logistic regression

Initialize $\beta = (0, \ldots, 0)$
repeat
  Let $g = (0, \ldots, 0)$ be the gradient vector
  for $i = 1{:}n$ **do**
    $p_i = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$
    $\text{error}_i = p_i - y_i$
    $g = g + \text{error}_i \cdot x_i$
  end
  $\beta = \beta - \eta \cdot g/n$
until *until convergence*;

Note that algorithm uses all observations to compute the gradient.
This approach is called batch gradient descent.

The gradient computation $\nabla f(\beta) = \sum_{i=1}^{n}(y_i - p_i(\beta))x_i$ is doable when n is moderate, but not when $n \approx 500$ million.

- One batch update costs $O(np)$
- One stochastic update costs $O(p)$

So clearly, for example, 10K stochastic steps are much more affordable.

Stochastic gradient descent for logistic regression

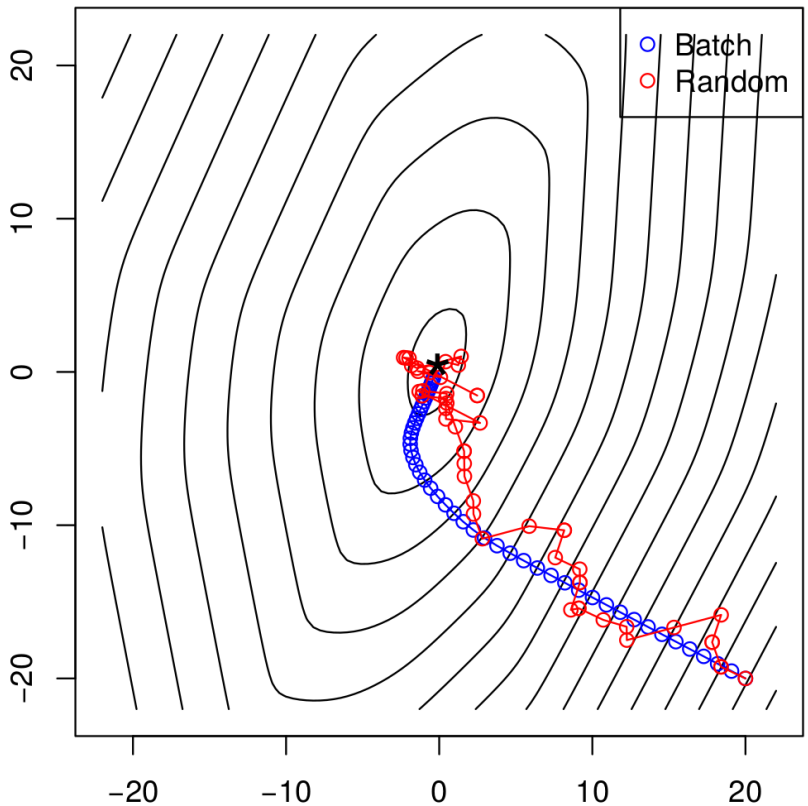
Initialize $\beta = (0, \ldots, 0)$
repeat
  Pick observation i
  $p_i = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$
  $\text{error}_i = p_i - y_i$
  $\beta = \beta - \eta(\text{error}_i \cdot x_i)$
until *until convergence*;

Coefficient β is updated after each observation.

In practice, mini batch is often used.

- Compute gradient based on a small subset of observations
- Make update to coefficient vector

The "classic picture":



Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

Avoiding overfitting using early stopping

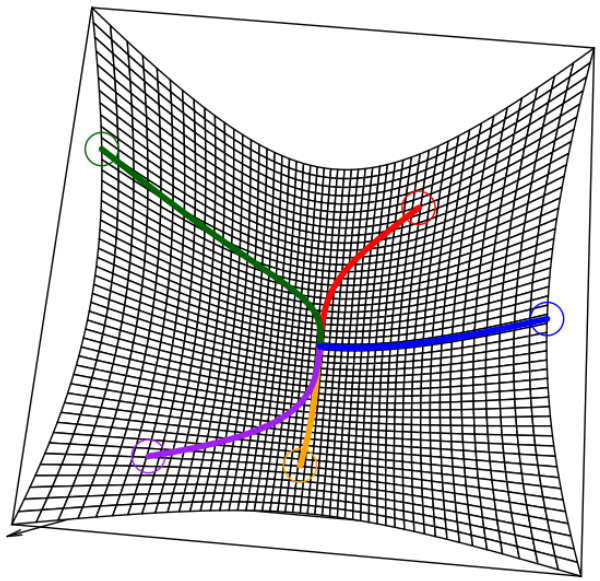
Common practice is to stop gradient descent before it reaches optimum

One monitors performance of the current solution on the validation data

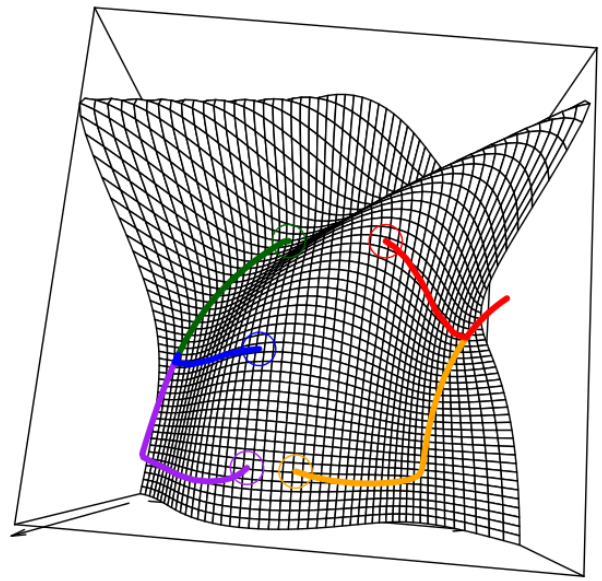
If the performance starts to deteriorate, stop the descent

Why should this work?

Fitting neural networks



Convex



Nonconvex

Fitting neural networks

Gradient descent + chain rule + lot of tricks

- We will not provide details
- The procedure is called backpropagation
- Difficult to train because there are many local minima

Avoiding over-fitting

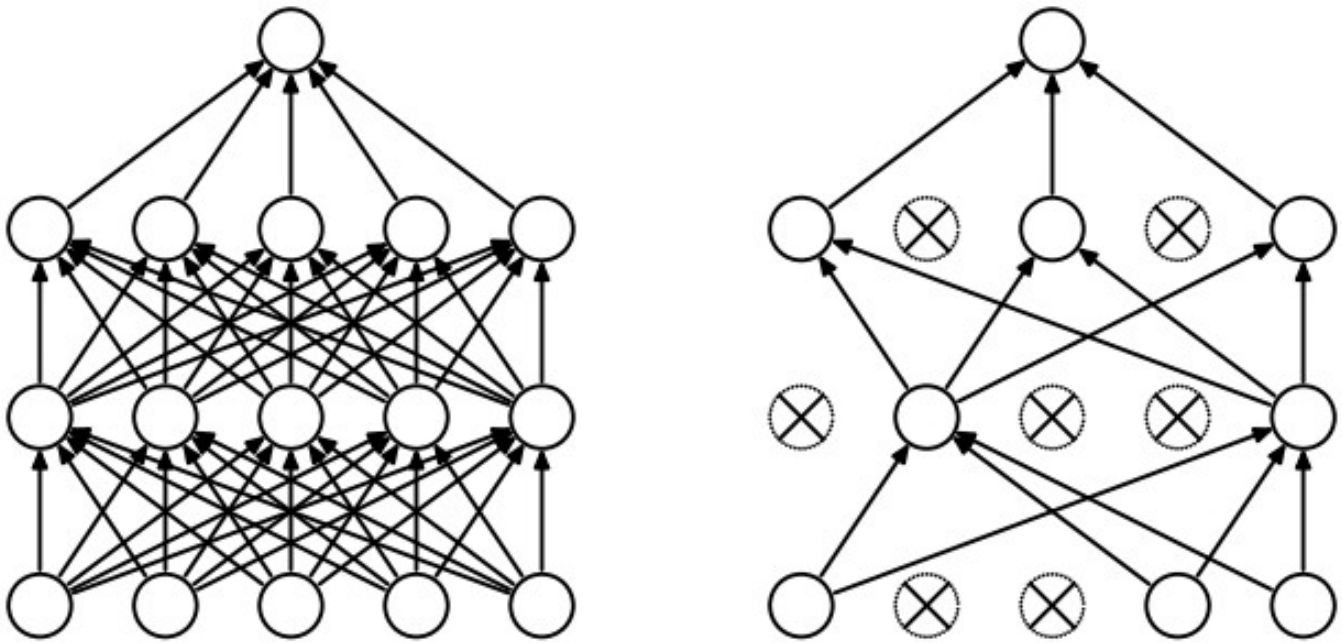
- Regularization
  - L1 penalty on the parameters
  - L2 penalty on the parameters (weight decay parameter)
  - Early stopping
- Dropout

See H2O booklet:
`http://docs.h2o.ai/h2o/latest-stable/index.html`

- `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/booklets/DeepLearningBooklet.pdf`

Fitting neural networks

Dropout:

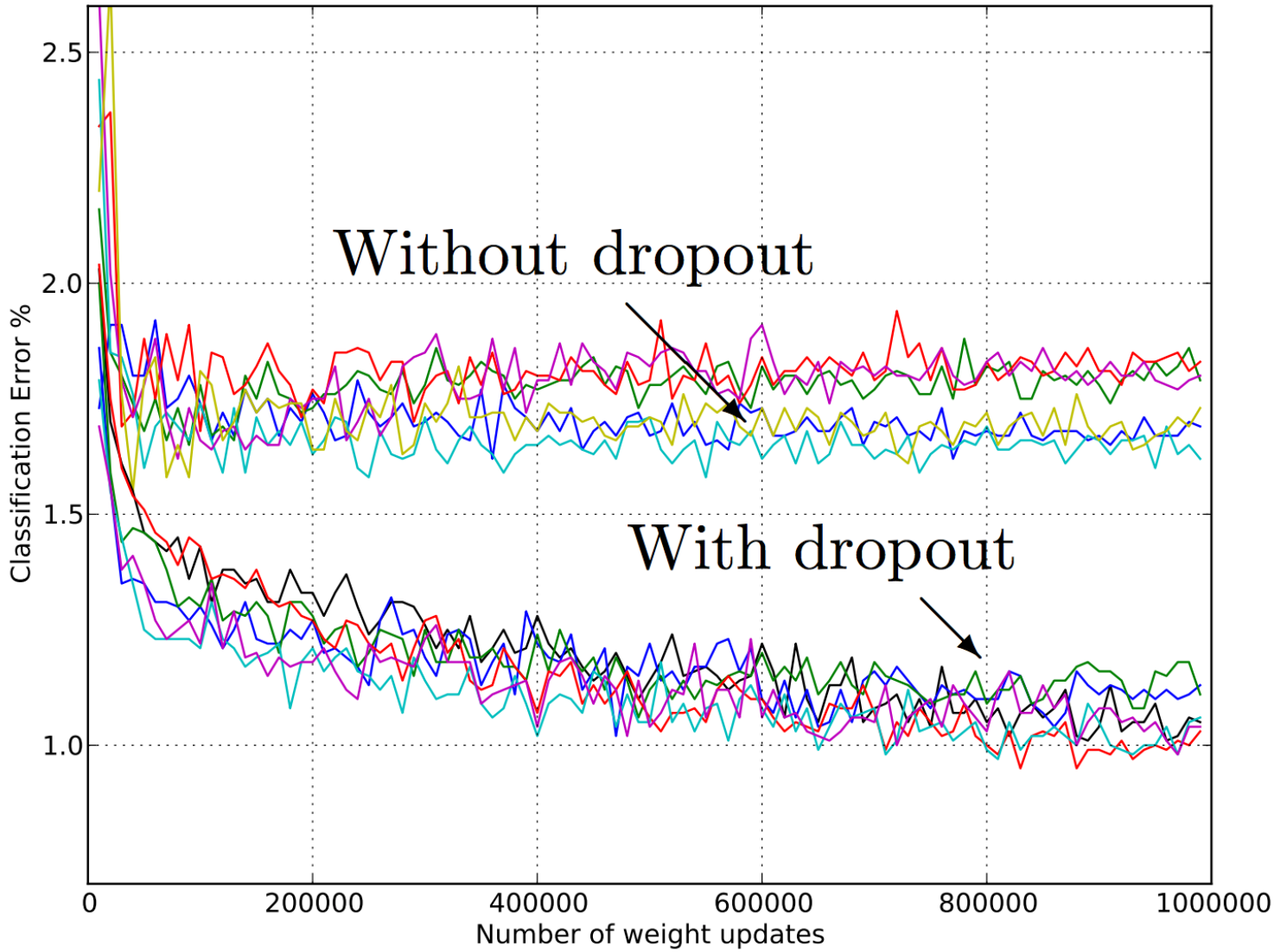


(a) Standard Neural Net

(b) After applying dropout.

https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Fitting neural networks



https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Additional resources

- Deep Learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville `http://www.deeplearningbook.org/`
- Free online book by Michael Nielsen
  `http://neuralnetworksanddeeplearning.com/`
  (explains backpropagation well)
- Yann LeCun's Deep Learning Course
  `https://cds.nyu.edu/deep-learning/`