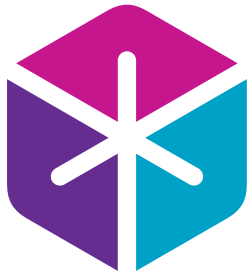
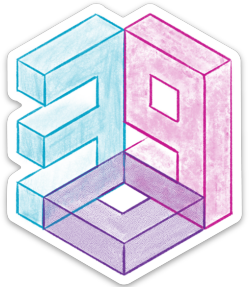




https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 5: Features, Neural Networks I

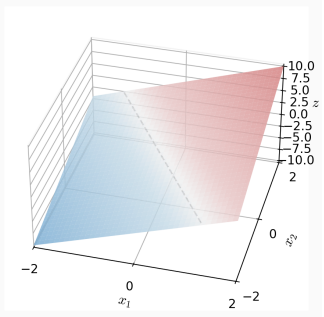
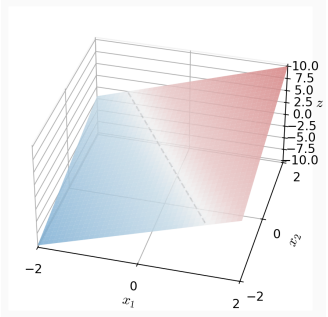
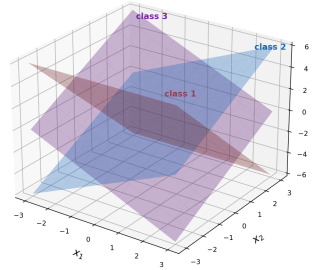
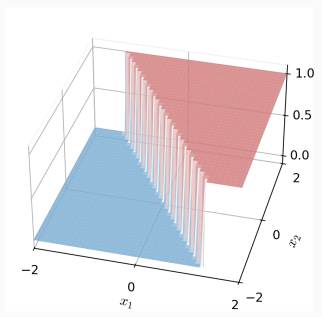
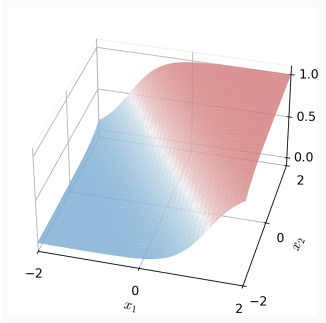
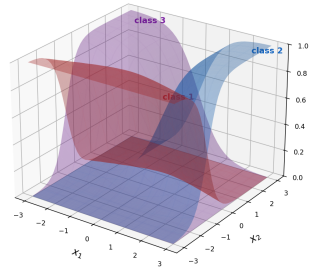
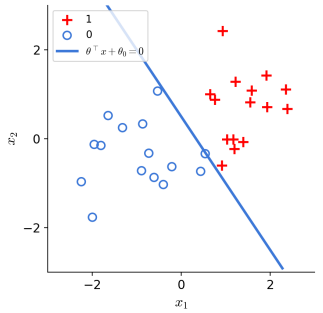
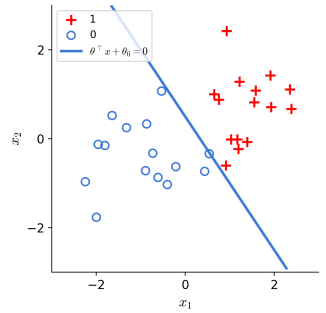
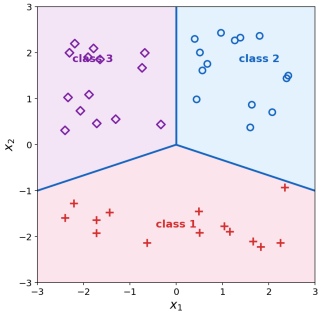
Shen Shen

Mar 2, 2026

3pm, Room 10-250

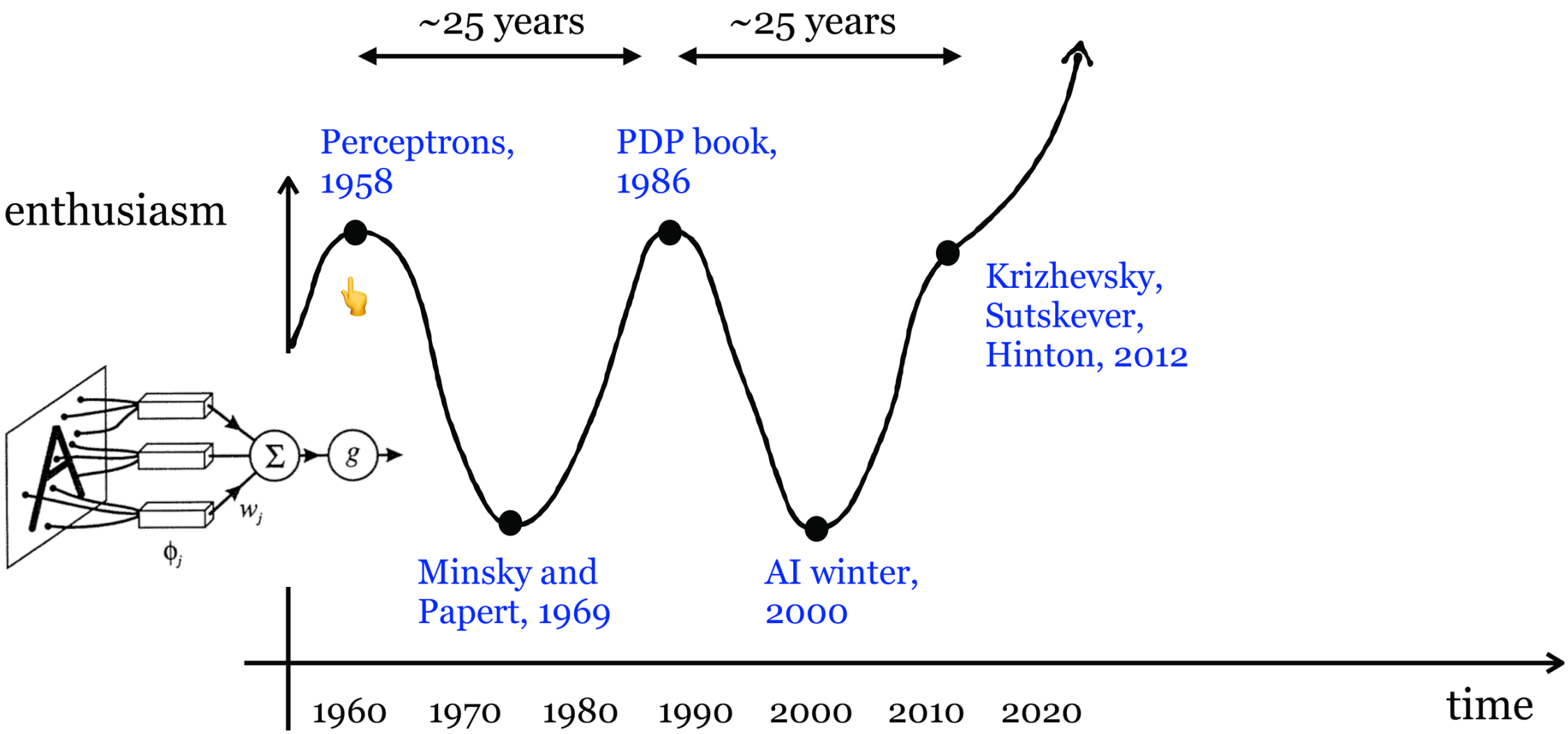
Slides and Lecture Recording

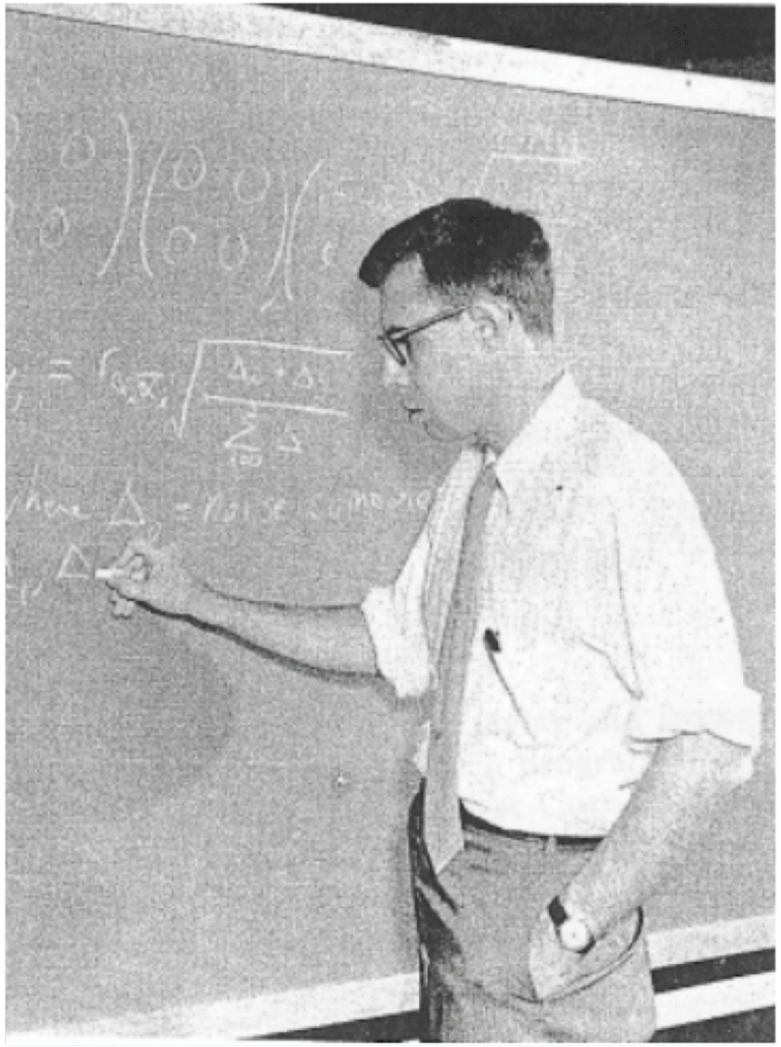
Recall:

step	logistic	multi-class
		
$z = \theta^\top x + \theta_0$	$z = \theta^\top x + \theta_0$	$z_k = \theta_k^\top x + \theta_{0,k}$
		
$g = \text{step}(z)$	$g = \sigma(z)$	$g = \text{softmax}(z_1, \ldots, z_K)$
		

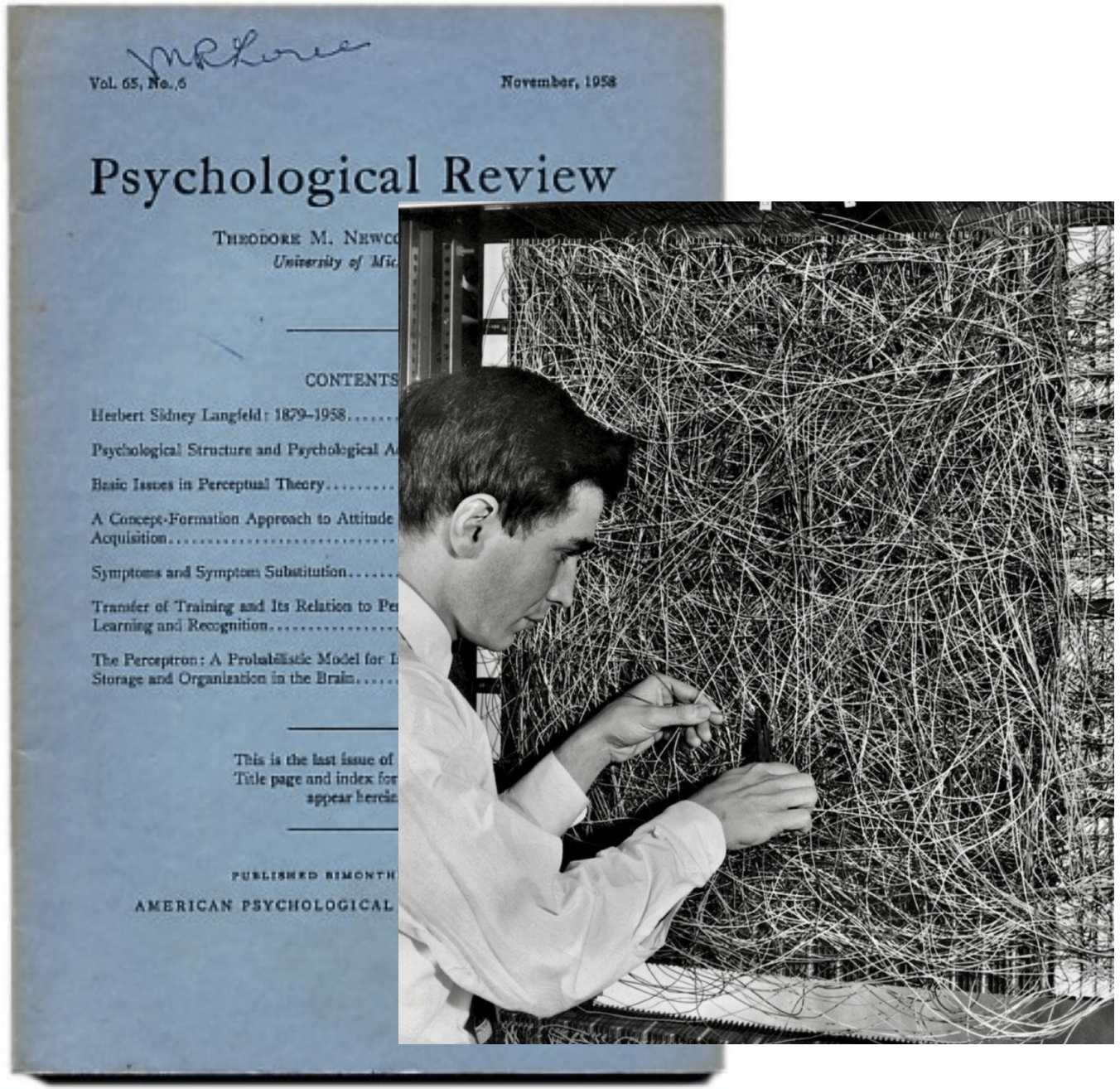
decision boundary is *linear* in feature x

Linear classification played a pivotal role in kicking off the first wave of AI enthusiasm.





http://www.ecse.rpi.edu/homepages/nagy/PDF_chrono/2011_Nagy_Pace_FR.pdf. Photo by George Nagy



http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf

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.

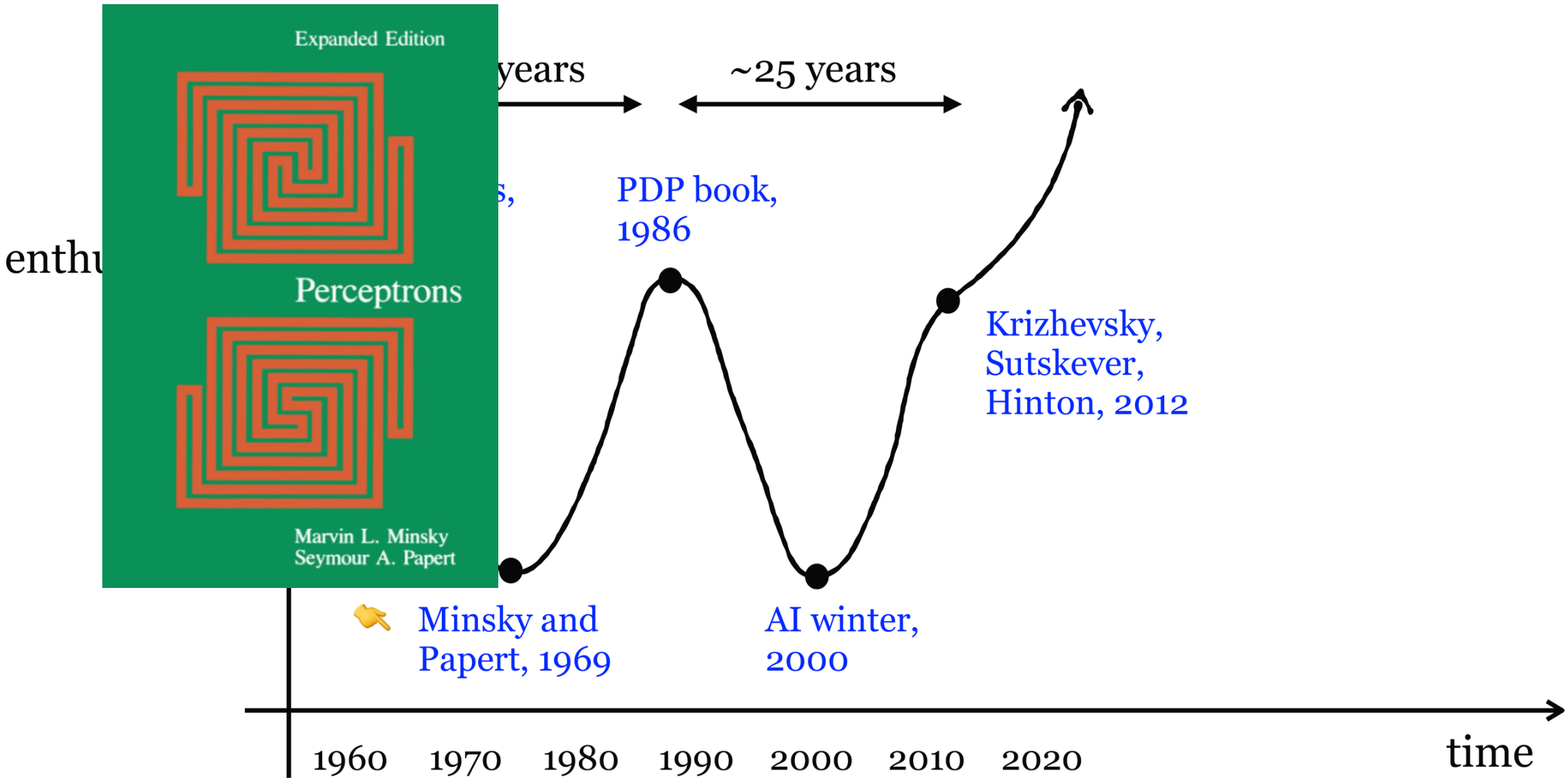
In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

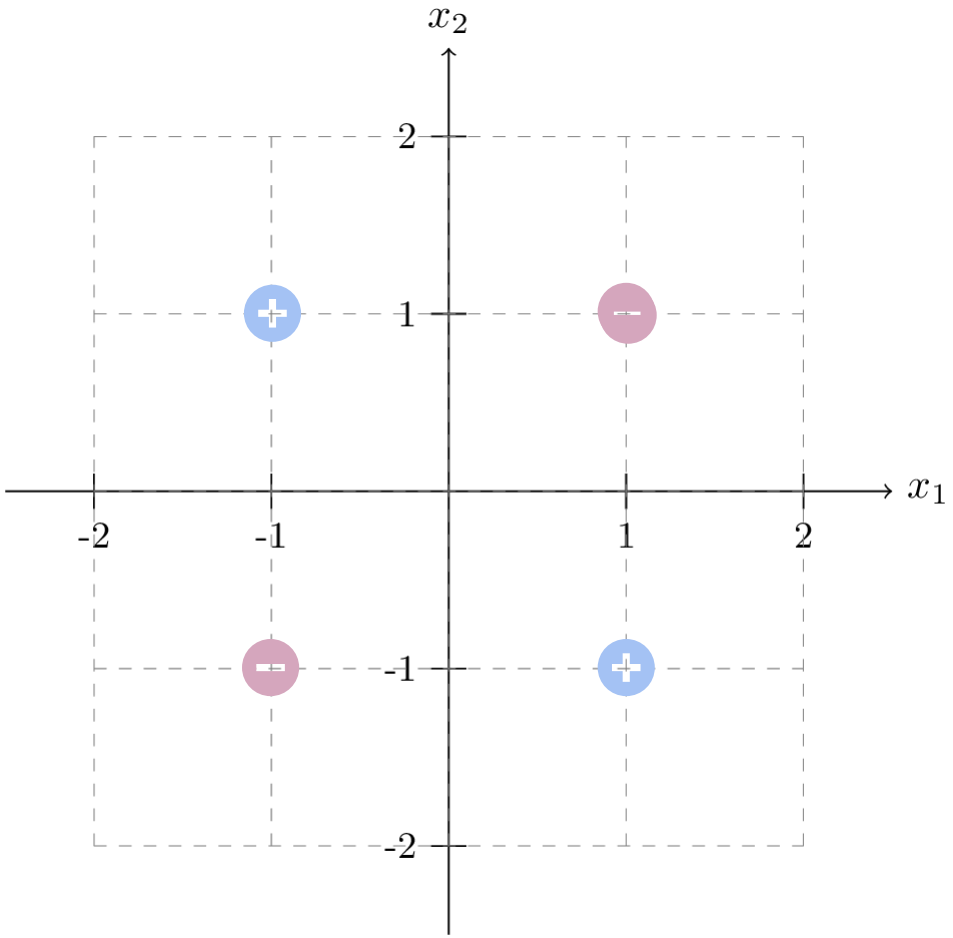
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

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.


Expanded Edition
Perceptrons
Marvin L. Minsky
Seymour A. Papert
enthu
years
~25 years
s,
PDP book,
1986
Krizhevsky,
Sutskever,
Hinton, 2012
Minsky and
Papert, 1969
AI winter,
2000
1960
1970
1980
1990
2000
2010
2020
time

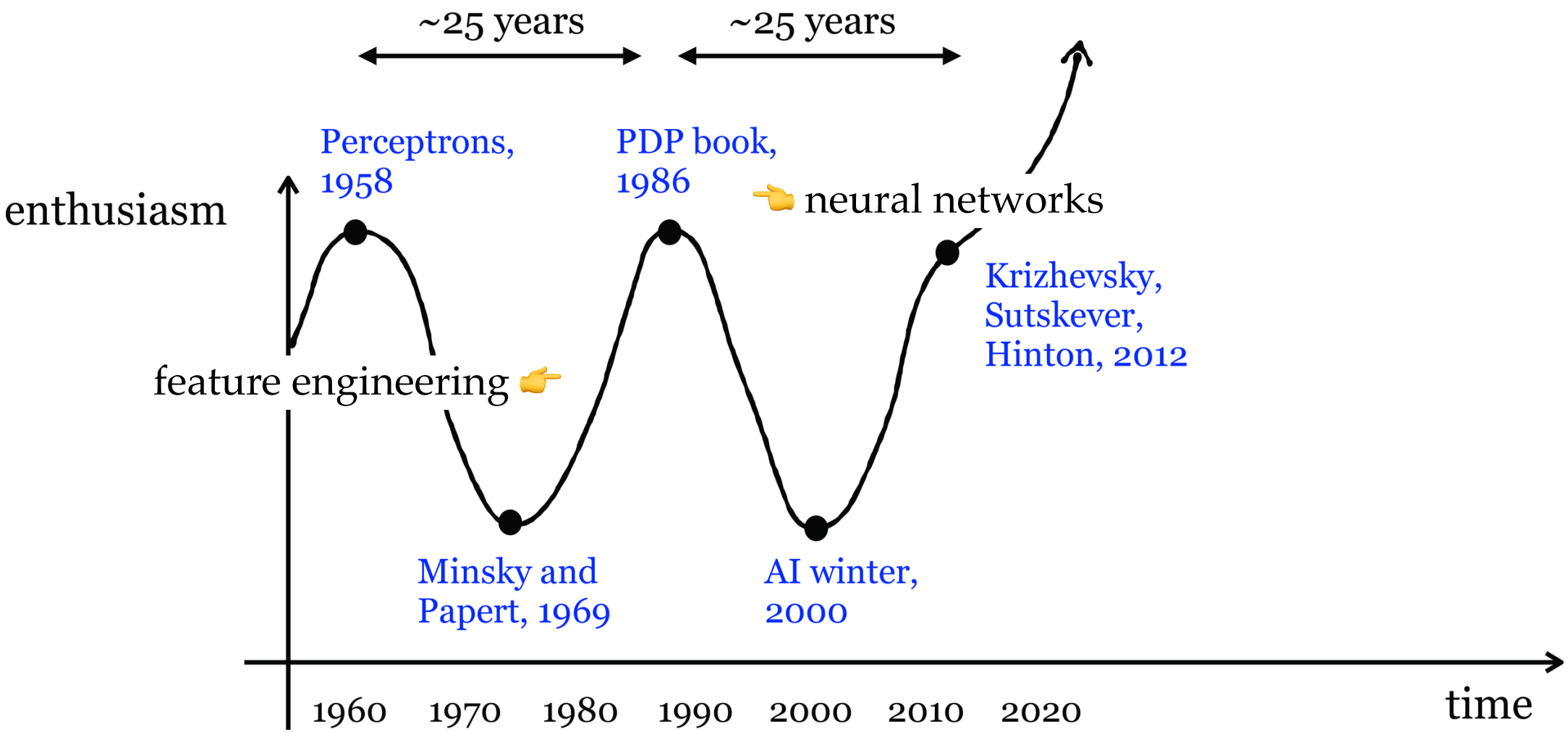
XOR dataset



Not **linearly** separable.

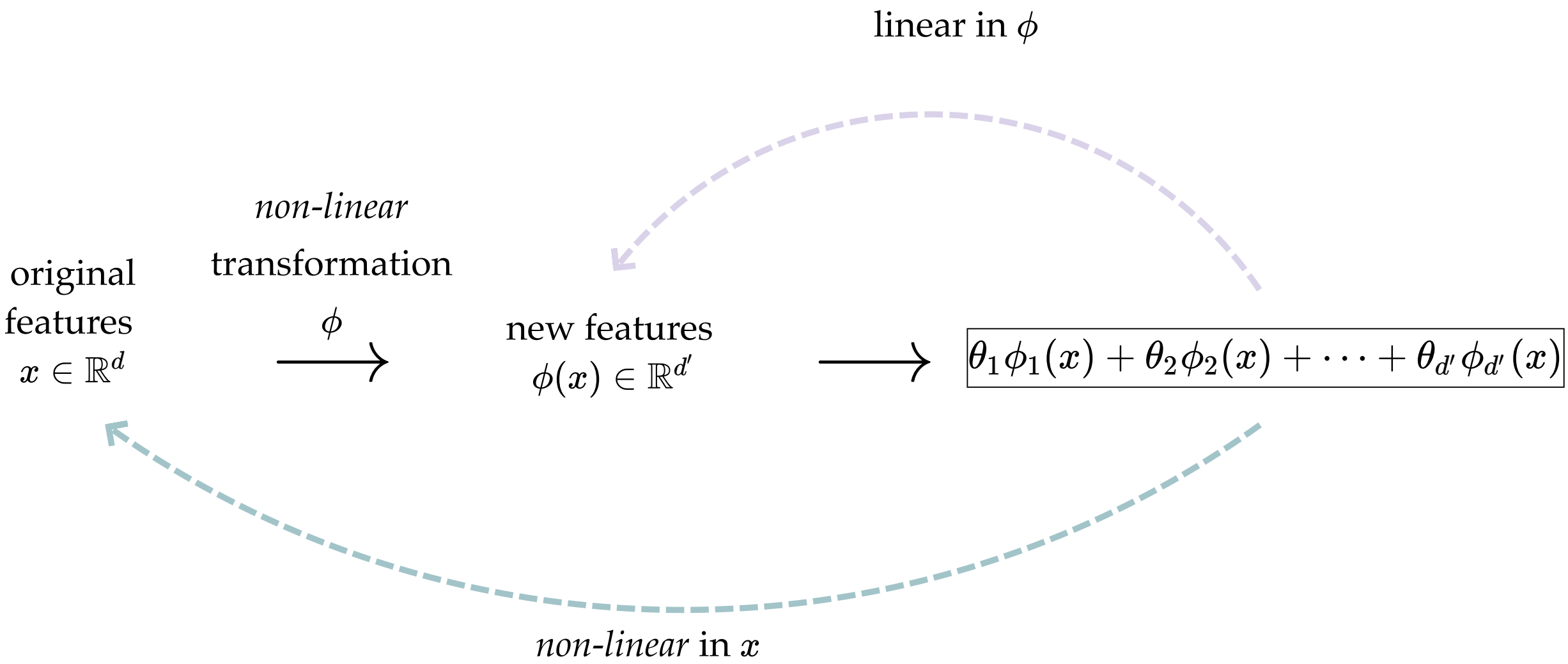
~~Linear tools cannot solve interesting tasks.~~

Linear tools cannot, *by themselves,* solve interesting tasks.

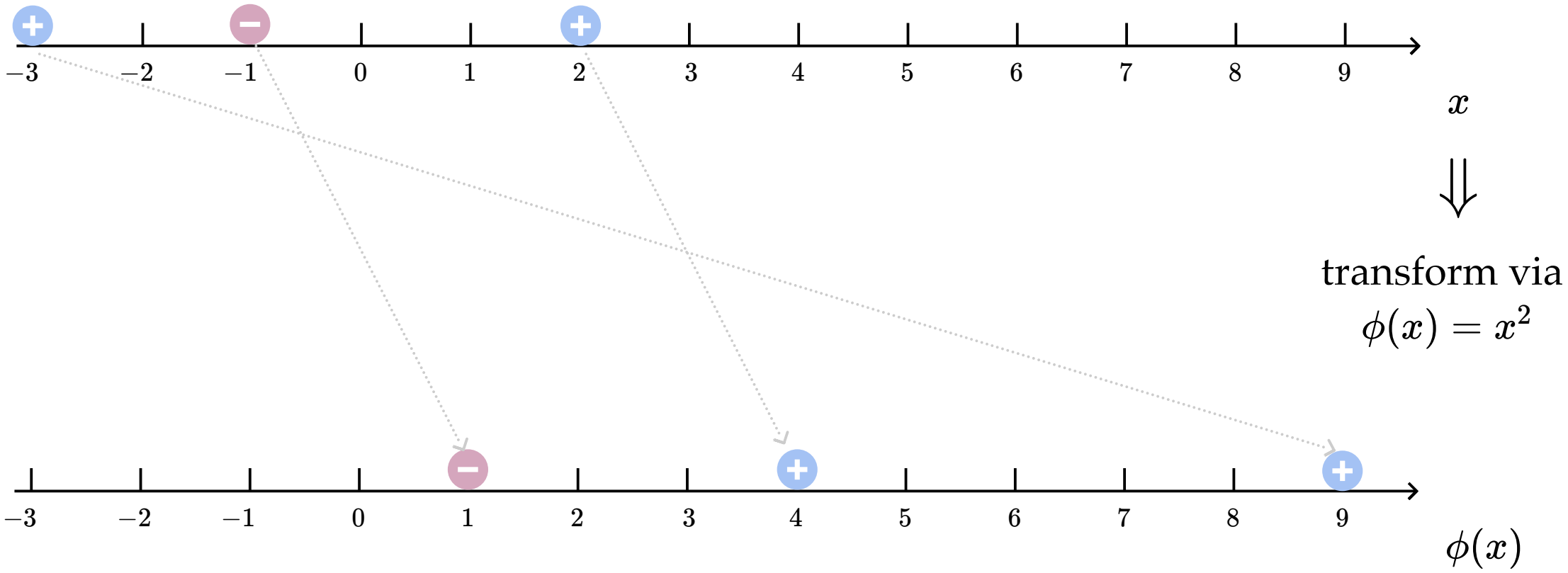

~25 years
~25 years
Perceptrons, 1958
PDP book, 1986
neural networks
enthusiasm
Krizhevsky, Sutskever, Hinton, 2012
feature engineering
Minsky and Papert, 1969
AI winter, 2000
1960
1970
1980
1990
2000
2010
2020
time

Outline

- Systematic feature transformations
  - Engineered features
  - Polynomial features
  - Expressive power
- Neural networks

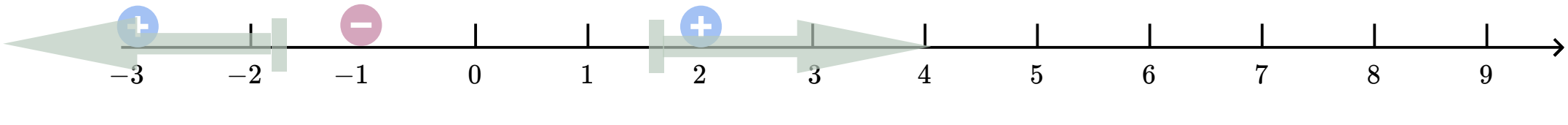

linear in ϕ
non-linear
transformation
ϕ
original
features
$x \in \mathbb{R}^d$
new features
$\phi(x) \in \mathbb{R}^{d'}$
$\theta_1\phi_1(x) + \theta_2\phi_2(x) + \cdots + \theta_{d'}\phi_{d'}(x)$
non-linear in x

Not linearly separable in x space

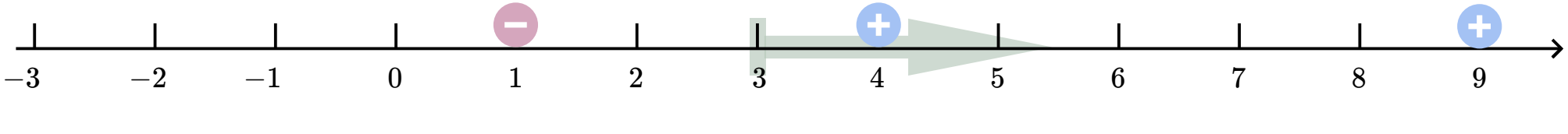


Linearly separable in $\phi(x) = x^2$ space

Non-linearly separated in x space, e.g. predict positive if $x^2 \geq 3$

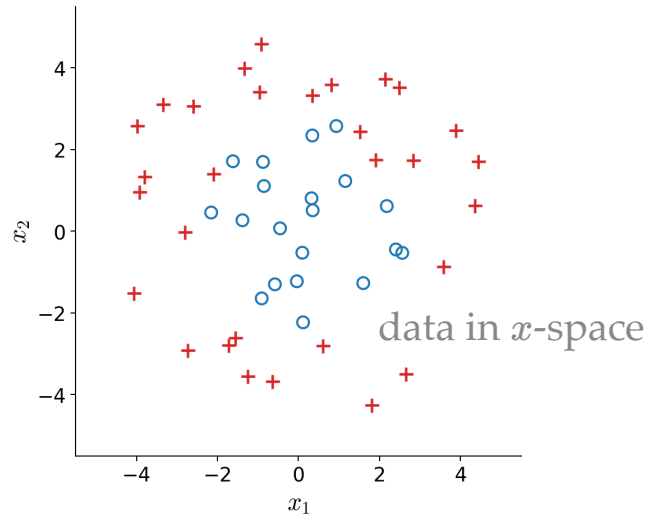


transform via
$\phi(x) = x^2$

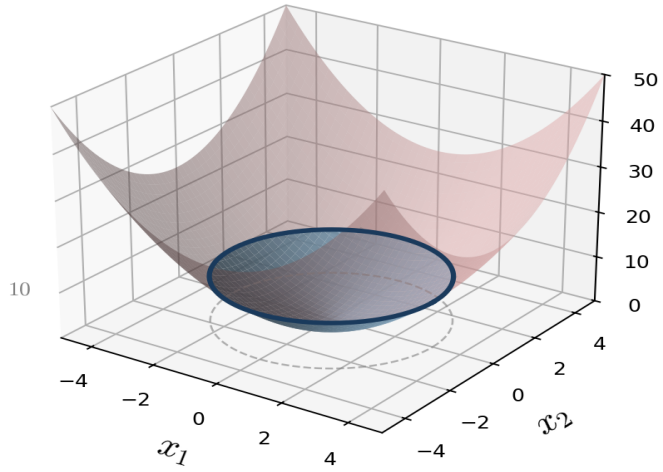


Linearly separable in $\phi(x) = x^2$ space, e.g. predict positive if $\phi \geq 3$

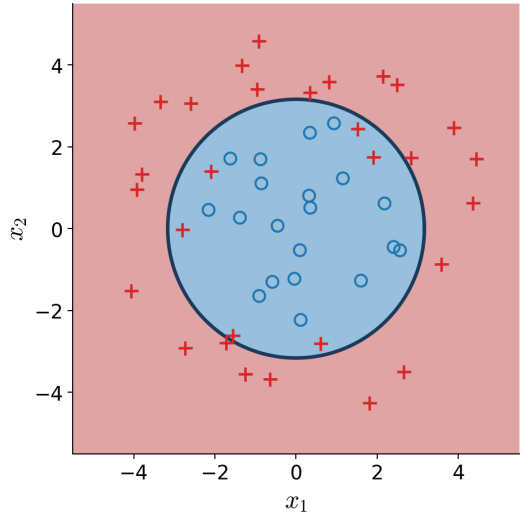
non-linear classification



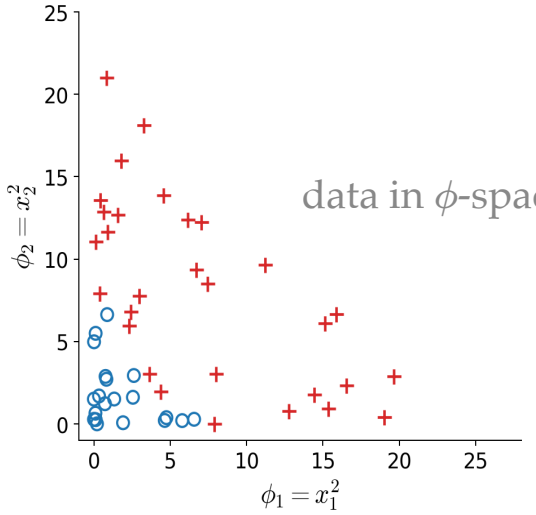
$z = x_1^2 + x_2^2$, threshold at $z = 10$



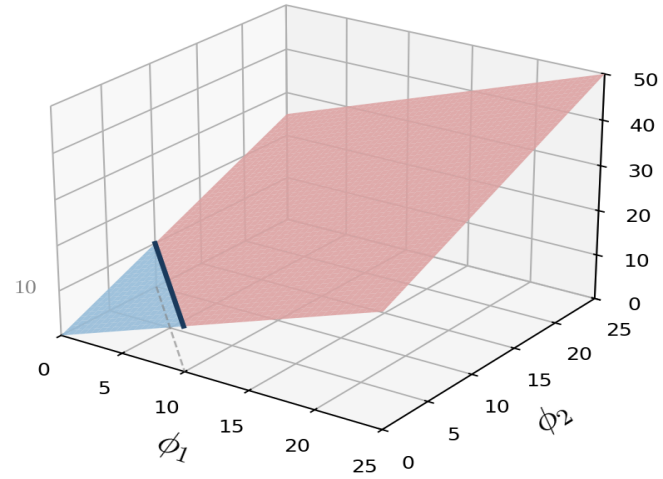
$x_1^2 + x_2^2 = 10$: *non-linear* in x



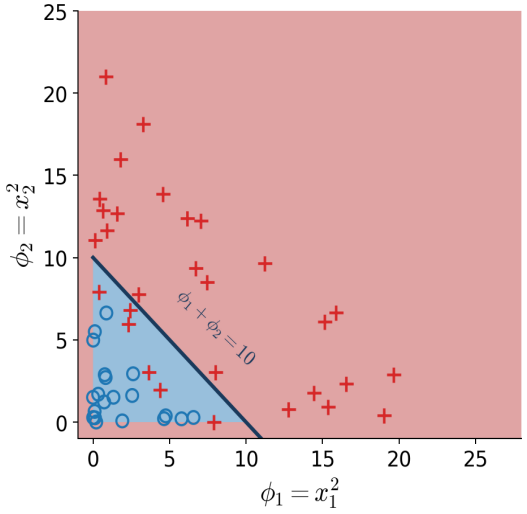
transform via $\phi(x) = (x_1^2, x_2^2)$



$z = \phi_1 + \phi_2$, threshold at $z = 10$

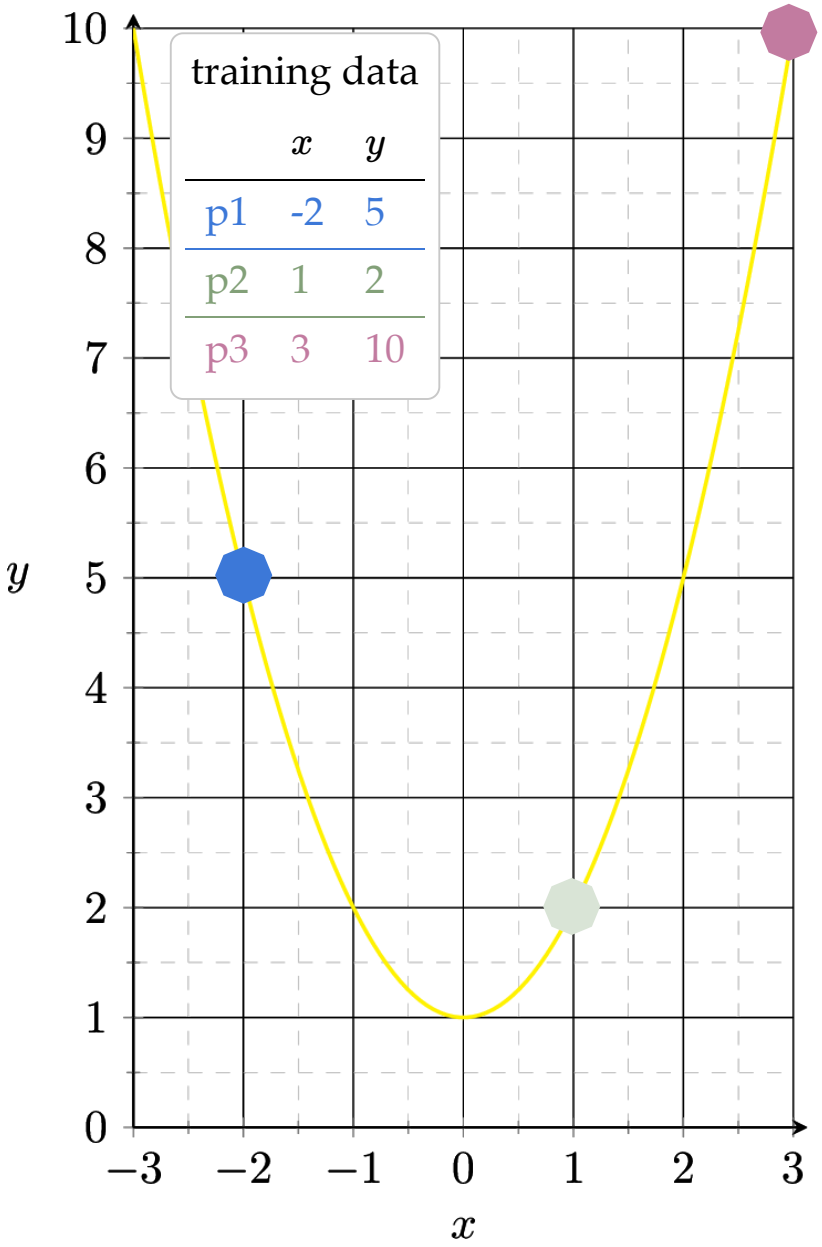


$\phi_1 + \phi_2 = 10$: *linear* in ϕ



decision boundary is *linear* in ϕ, *nonlinear* in x

non-linear regression



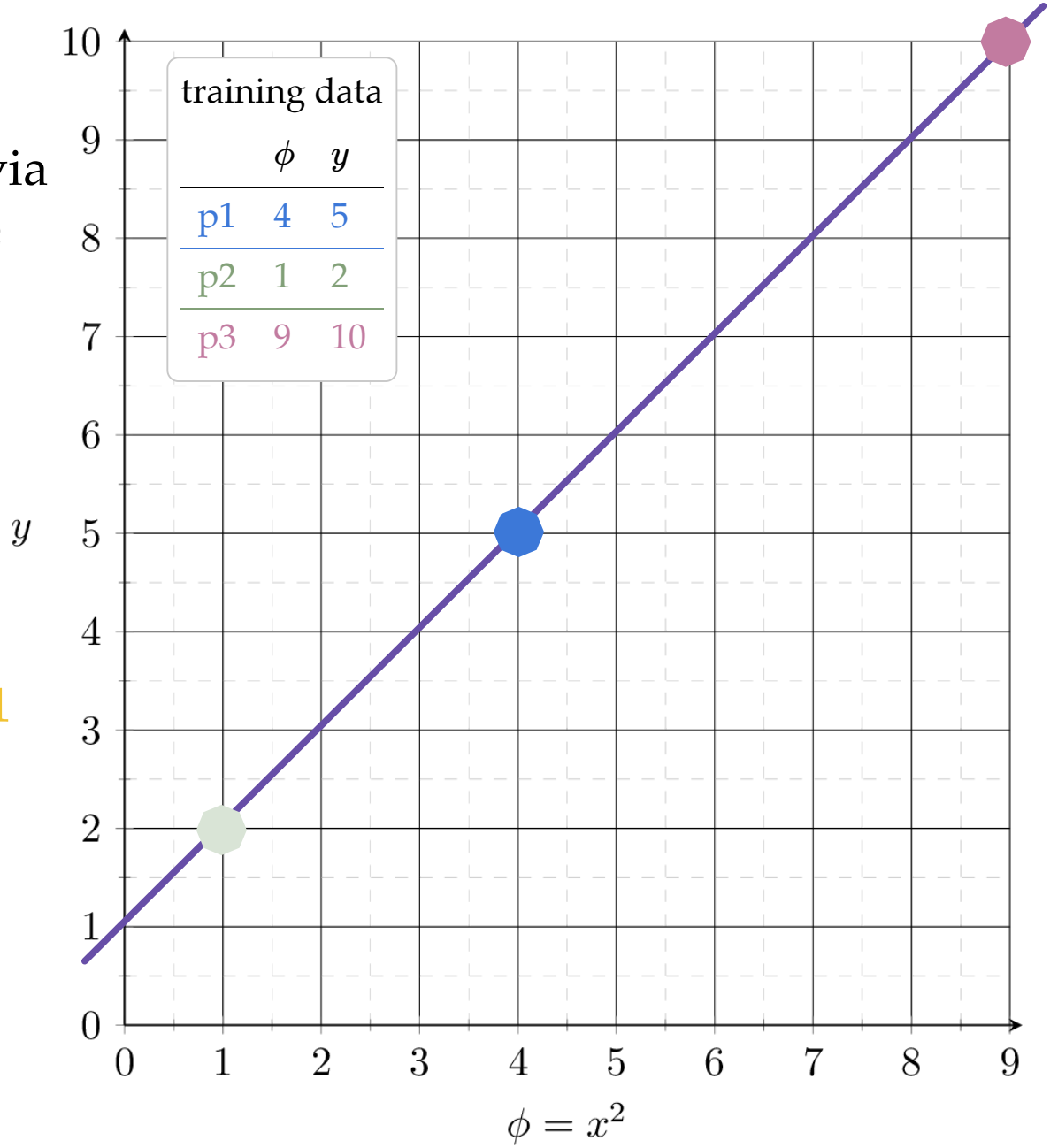
transform via

$$\phi(x) = x^2$$

$$\Rightarrow$$

$$g = \phi + 1$$

$$g = x^2 + 1$$



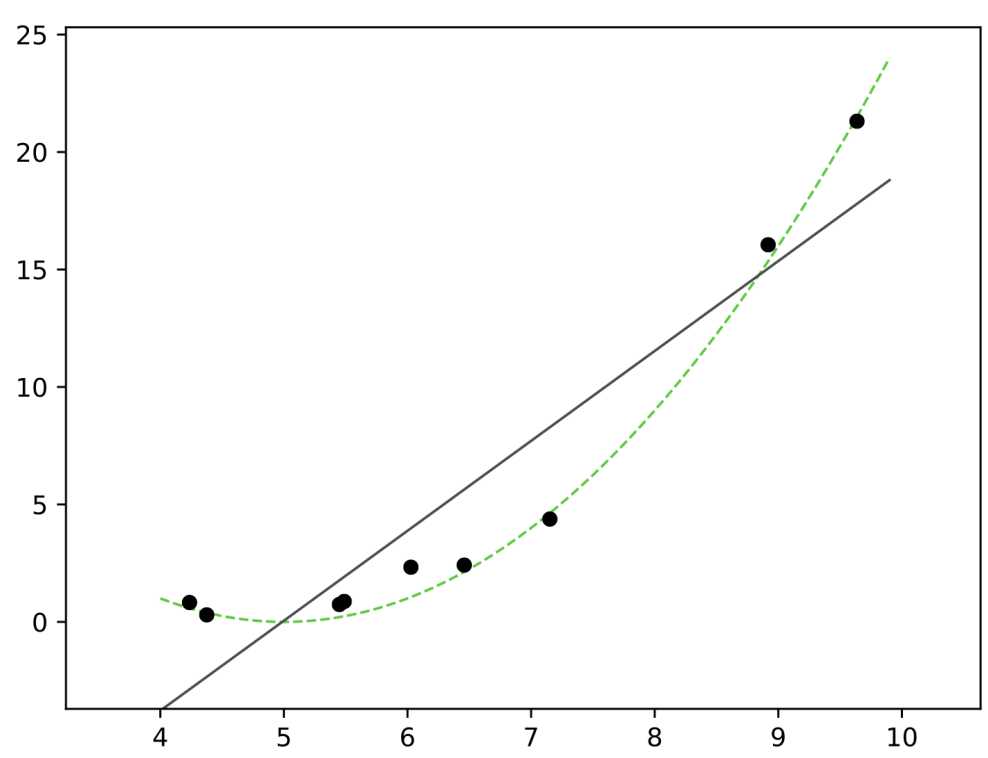
systematic polynomial features construction

	$d = 1$, features: x_1	$d = 2$, features: x_1, x_2
$k = 0$	1	1
$k = 1$	1, x_1	1, $x_1,\ x_2$
$k = 2$	1, x_1, x_1^2	1, $x_1,\ x_2$, $x_1^2,\ x_1x_2,\ x_2^2$
$k = 3$	1, x_1, x_1^2, x_1^3	1, $x_1,\ x_2$, $x_1^2,\ x_1x_2,\ x_2^2$, $x_1^3,\ x_1^2x_2,\ x_1x_2^2,\ x_2^3$
$\vdots$	$\vdots$	$\vdots$

- Elements in the basis are the monomials of original features raised up to power k
- With a given d and a fixed k, the basis is **fixed**.

- $n = 9$ data points, each with feature $x \in \mathbb{R}$ and label $y \in \mathbb{R}$
- data generated from green dashed curve

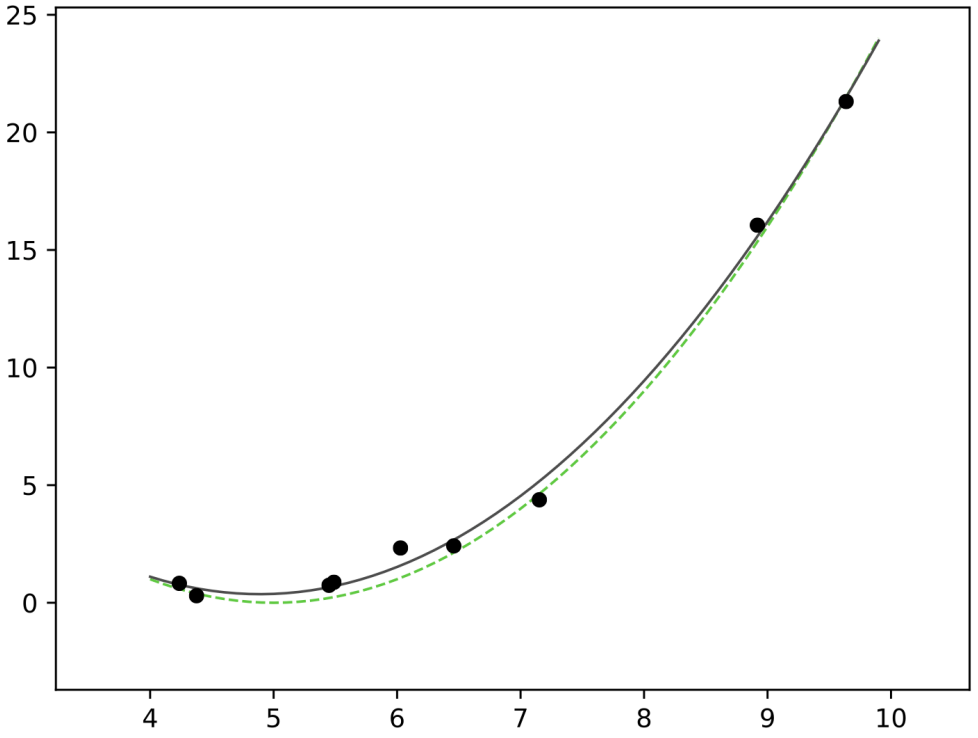
$k = 1$



$h = \theta_0 + \theta_1 x$

Learn 2 parameters — degree-1

$k = 2$

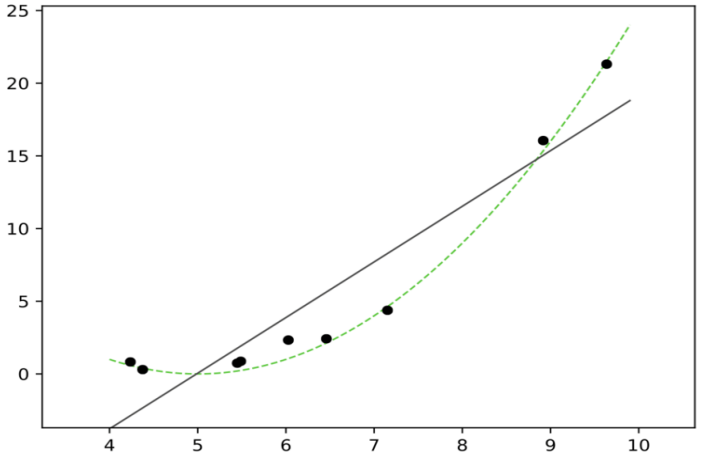


$h = \theta_0 + \theta_1 x + \theta_2 x^2$

Learn 3 parameters — degree-2

Underfitting

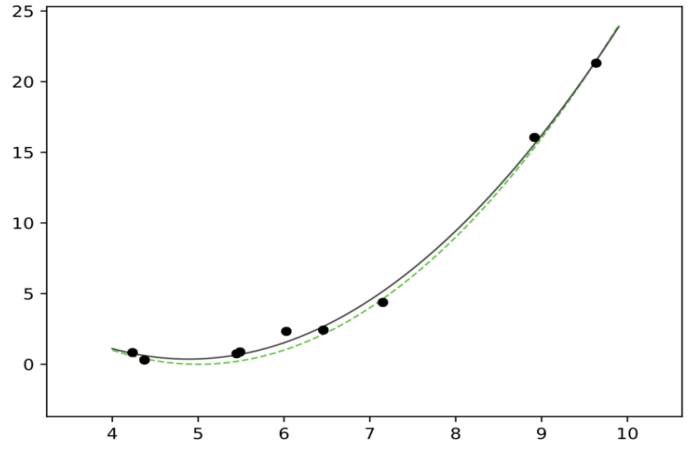
$k = 1$



high error on train set
high error on test set

Appropriate model

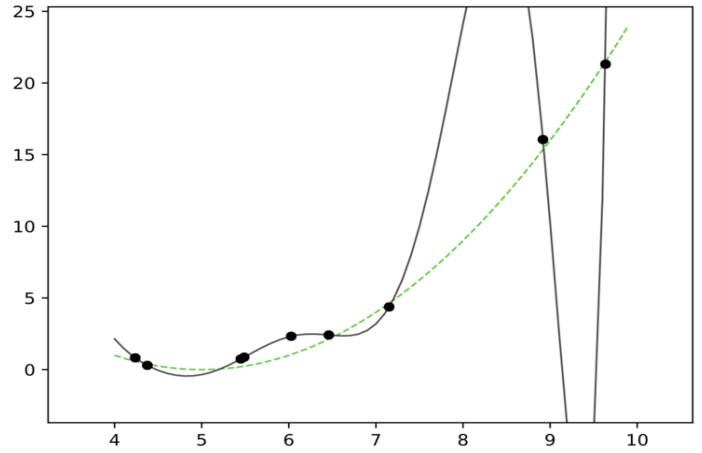
$k = 2$



low error on train set
low error on test set

Overfitting

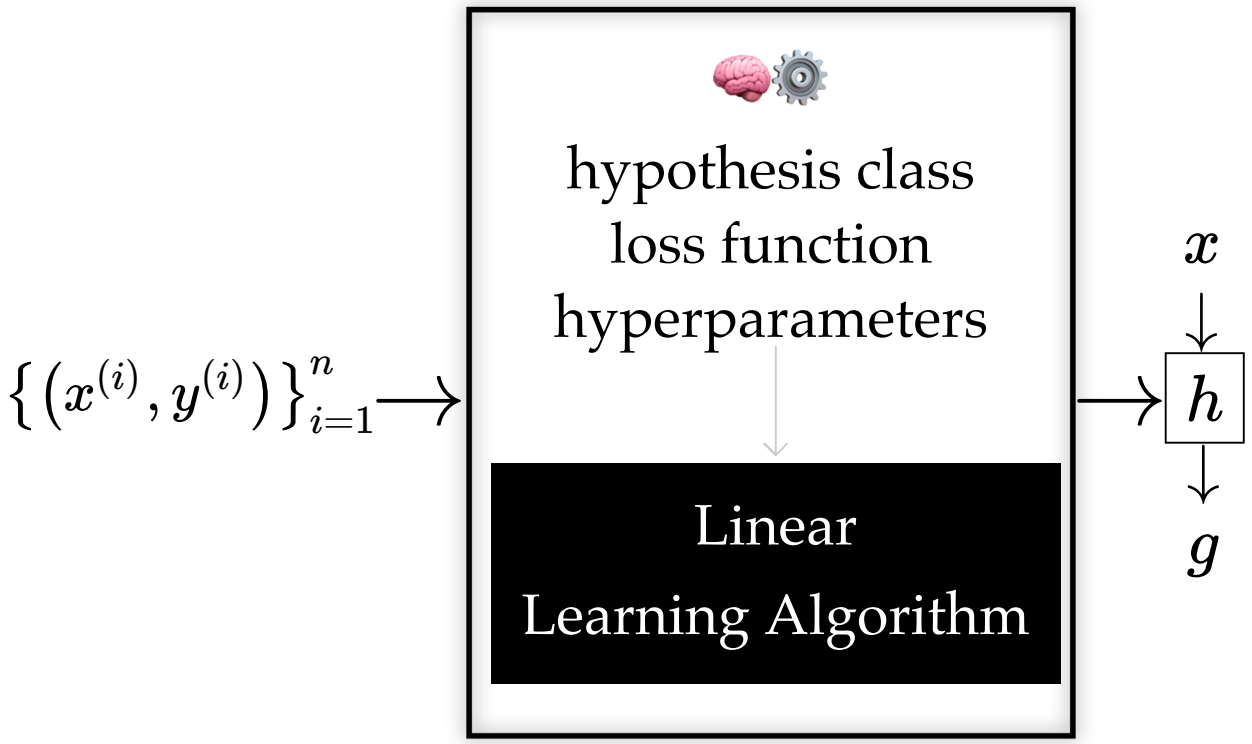
$k = 10$



low error on train set
high error on test set

- k : a hyperparameter that determines the capacity (expressiveness) of the hypothesis class.
- Models with many rich features and free parameters tend to have high capacity but also greater risk of overfitting.
- How to choose k? Validation/cross-validation.

Previously:



- Linear models are convenient but lack expressiveness for real-world tasks.
- We applied **fixed** nonlinear feature transformations, then used linear methods.
- The essence of ML was **feature engineering** — manually designing useful representations.

today, so far:

🧠⚙️

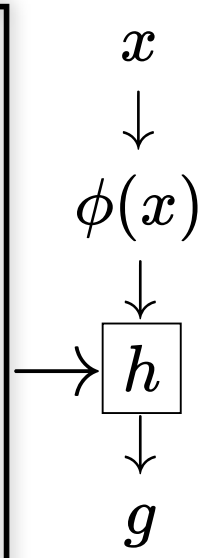
feature
transformation $\phi(x)$

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{n} \longrightarrow \{(\phi(x^{(i)}), y^{(i)})\}_{i=1}^{n} \longrightarrow$$

🧠⚙️

hypothesis class
loss function
hyperparameters

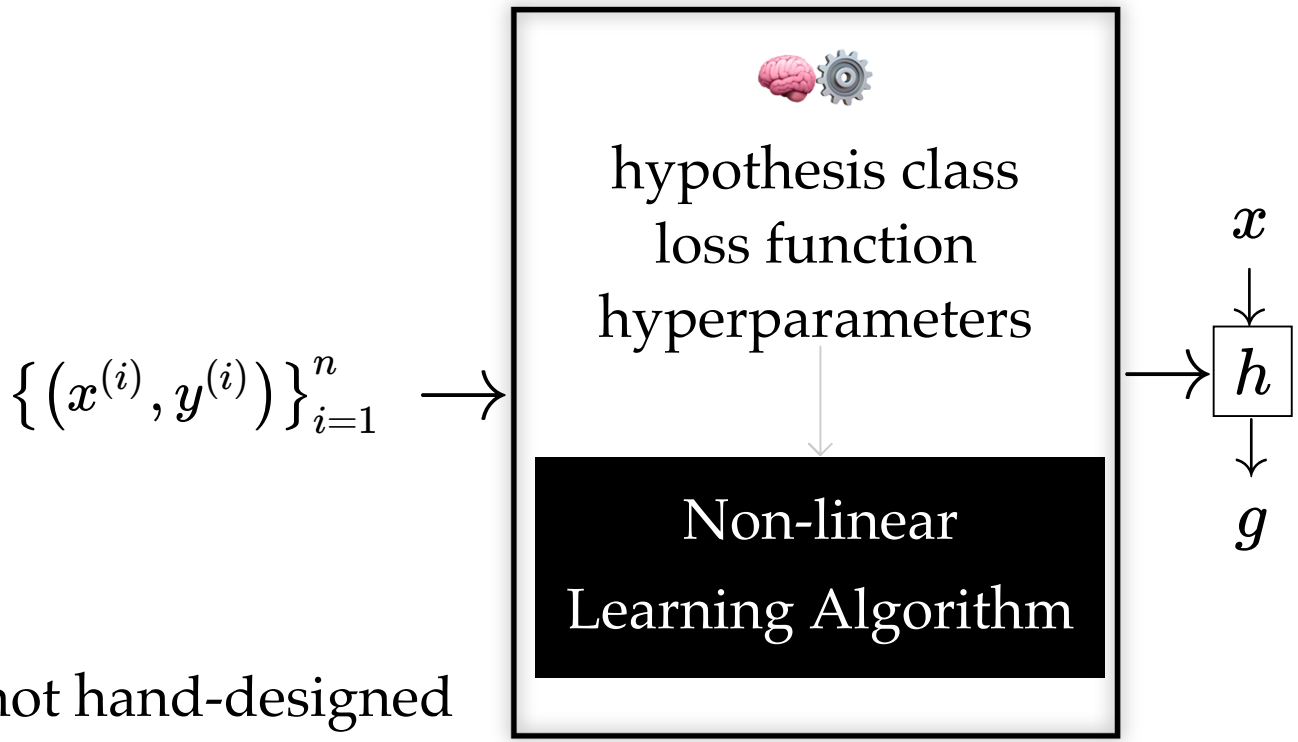
Linear
Learning Algorithm



can we automate 👆?

i.e. fold it into the learning algorithm?

neural networks:



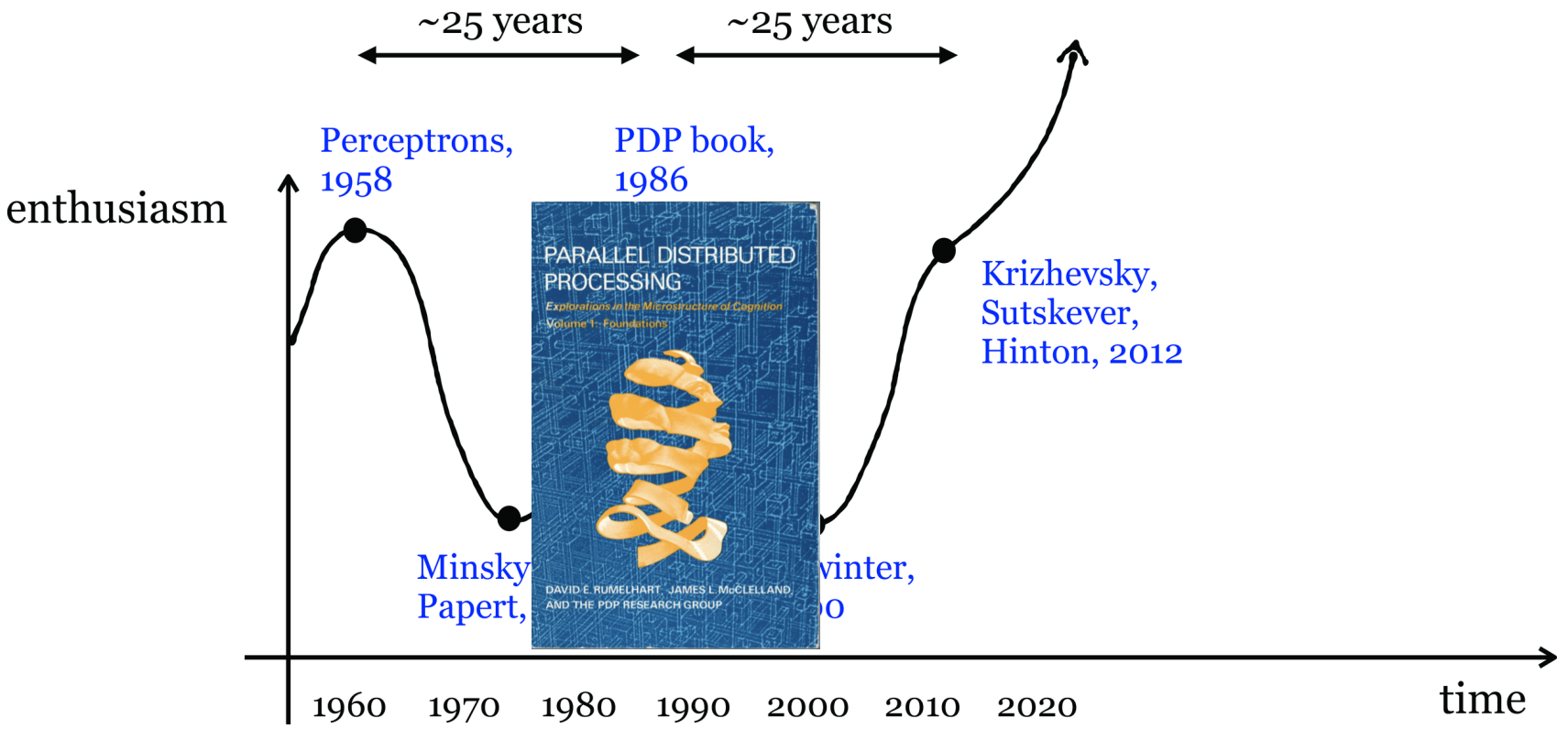
the nonlinearity ϕ is learned not hand-designed

Outline

- Systematic feature transformations
- Neural networks
  - Terminology
  - Design choices

👋 heads-up:

all neural network diagrams focus on a single data point



Outlined the fundamental concepts of neural networks:

- Nonlinear feature transformation
- Composing simple nonlinearities amplifies this effect

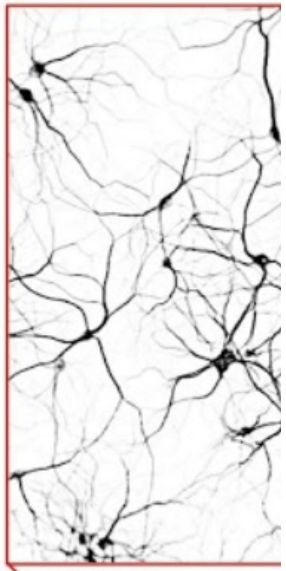
} expressiveness

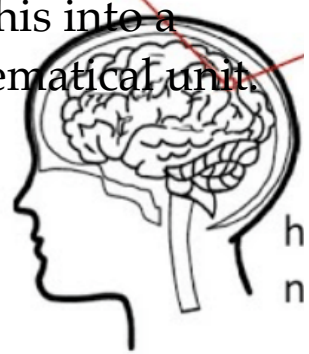
layered structure {

- Backpropagation

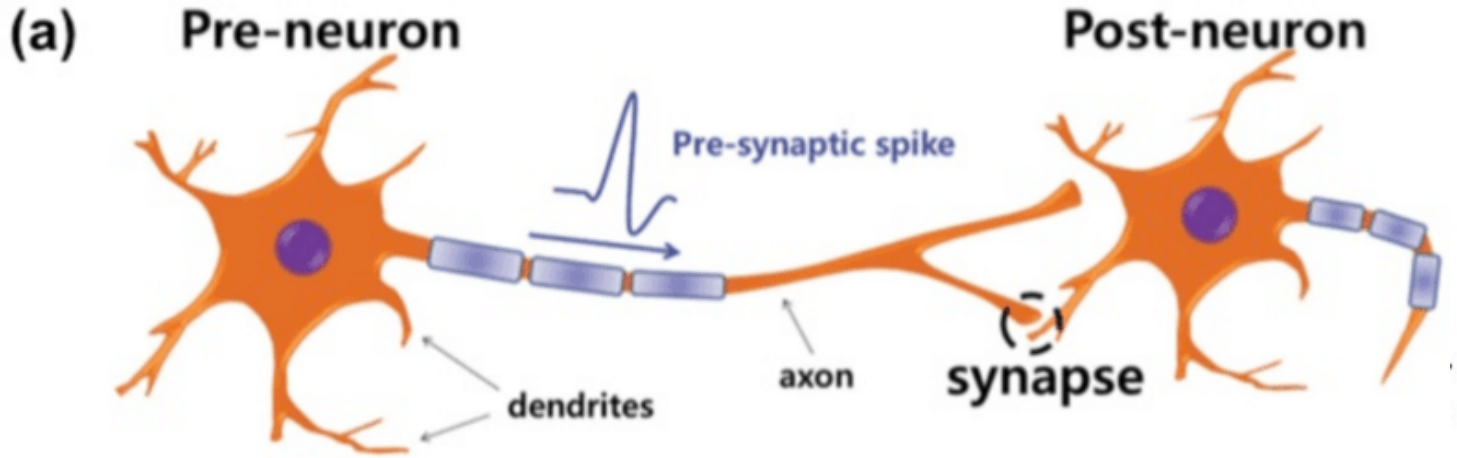
efficient learning

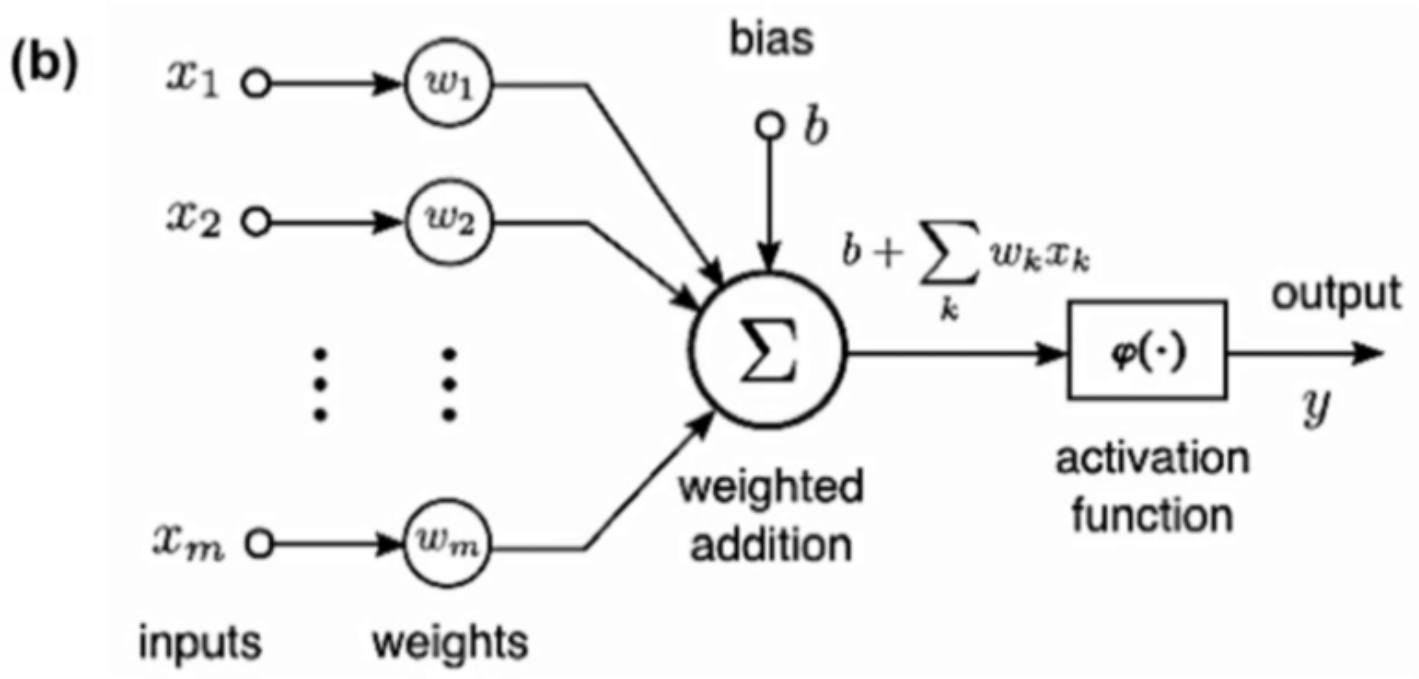
Neurons and the brain





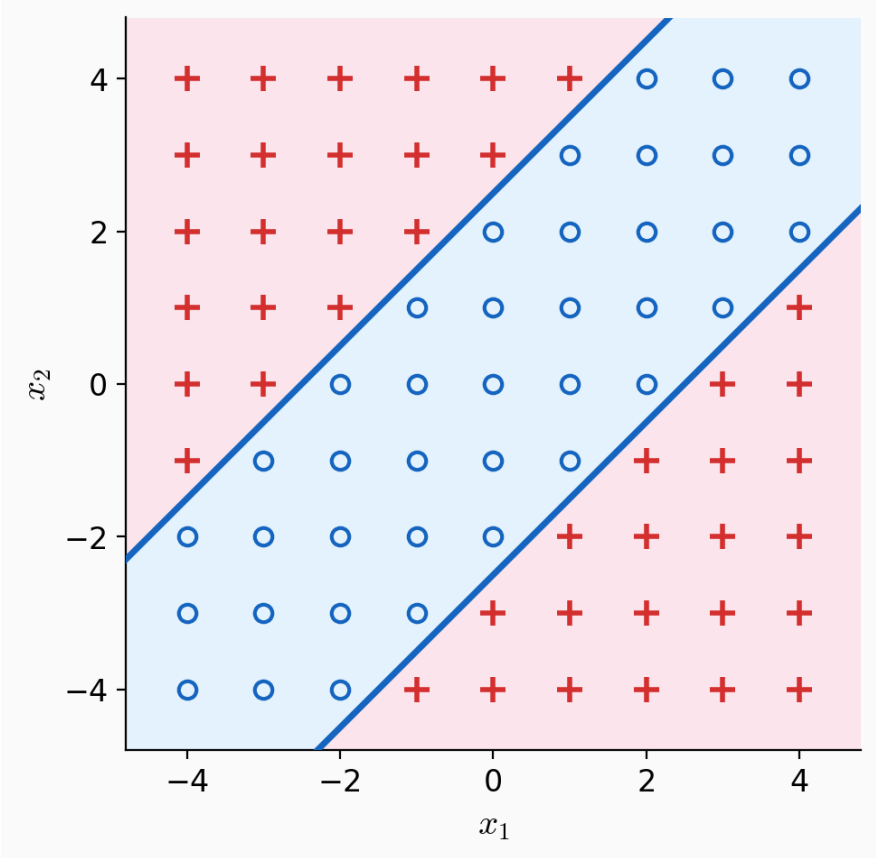
We abstract this into a simple mathematical unit.

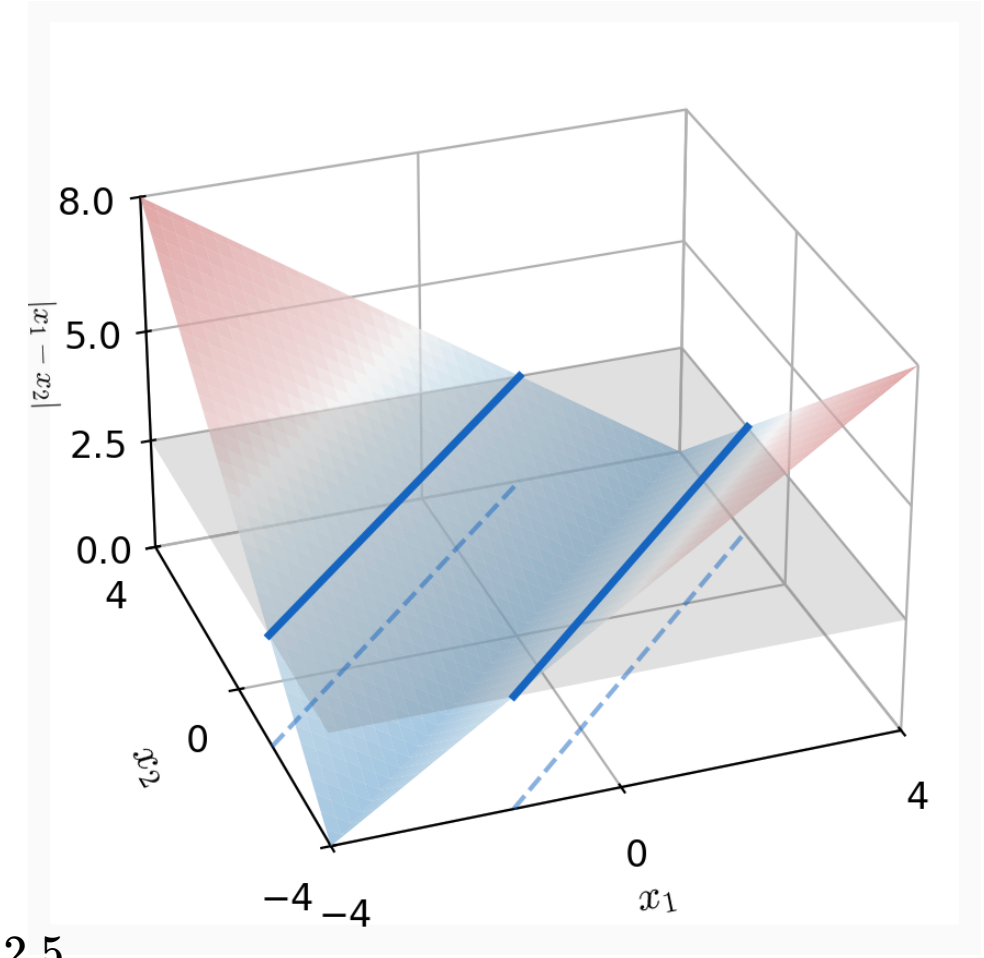




- Nonlinear feature transformation

transform via $\phi([x_1; x_2]) = [|x_1 - x_2|]$

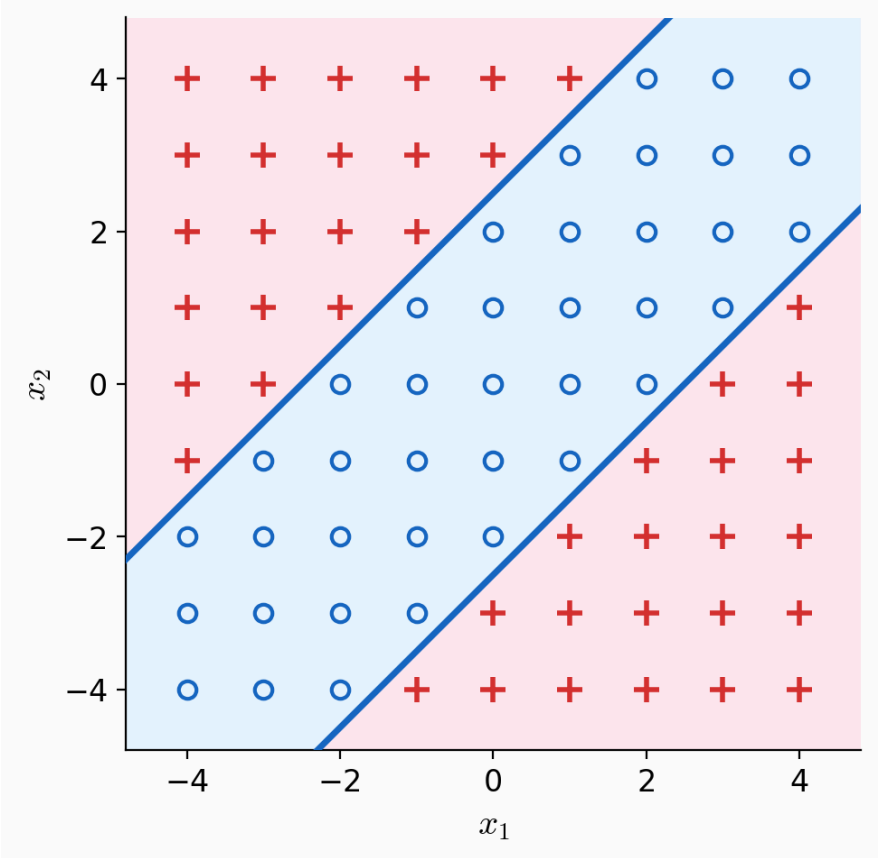


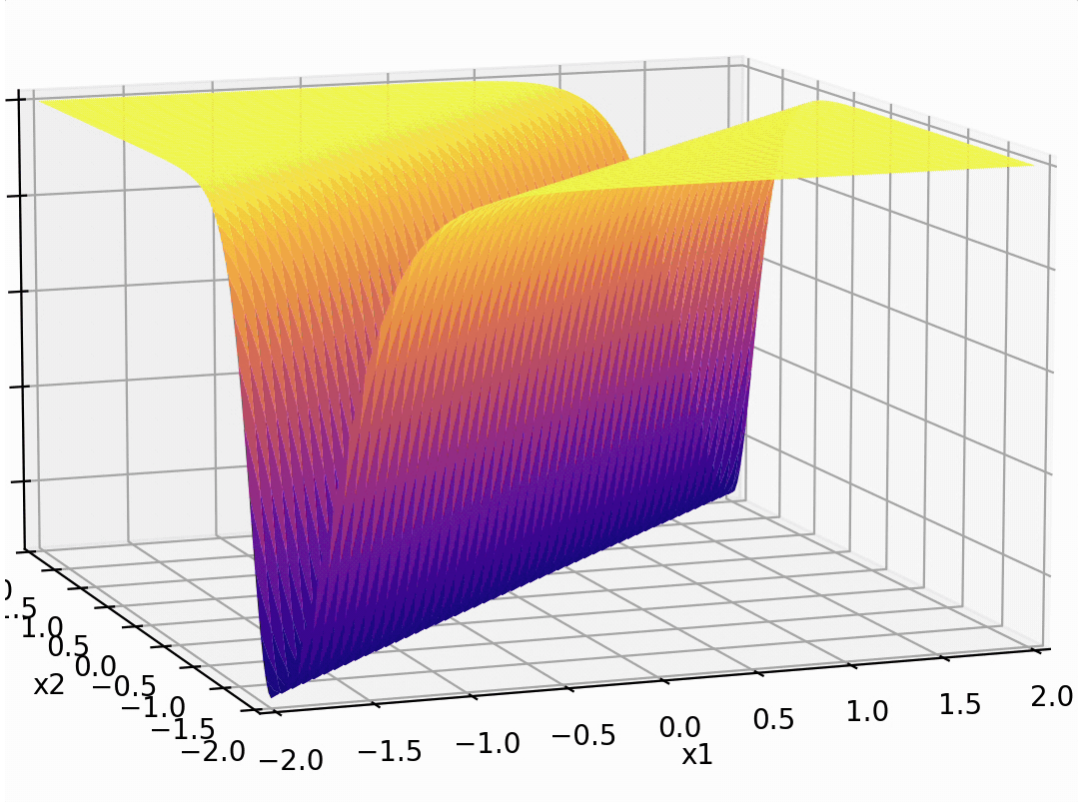


$\phi([x_1; x_2]) = [|x_1 - x_2|] = 2.5$

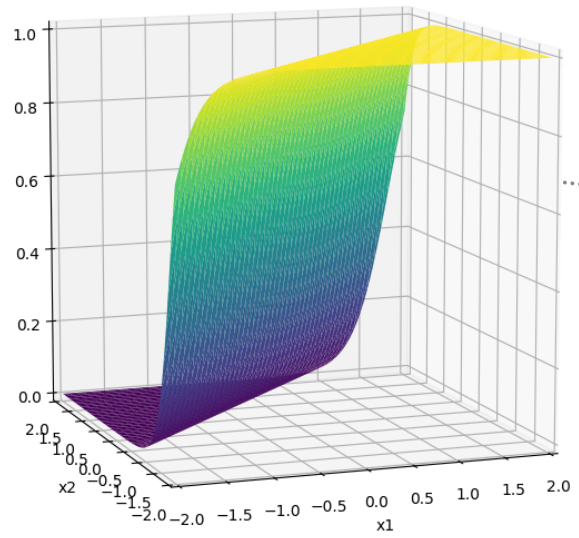
importantly, linear in ϕ, non-linear in x

- "Composing" simple transformations





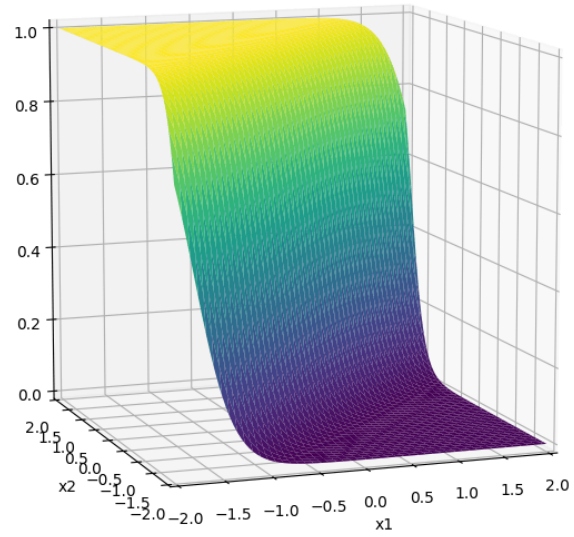
$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$



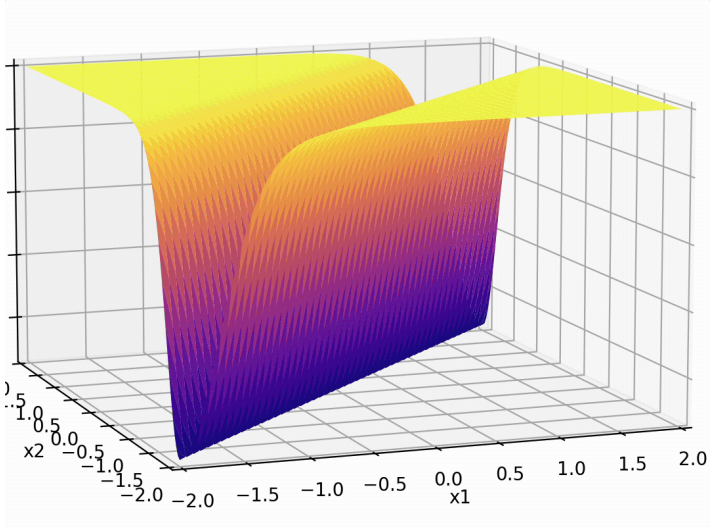
- "Composing" simple transformations

some appropriately weighted sum

$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$

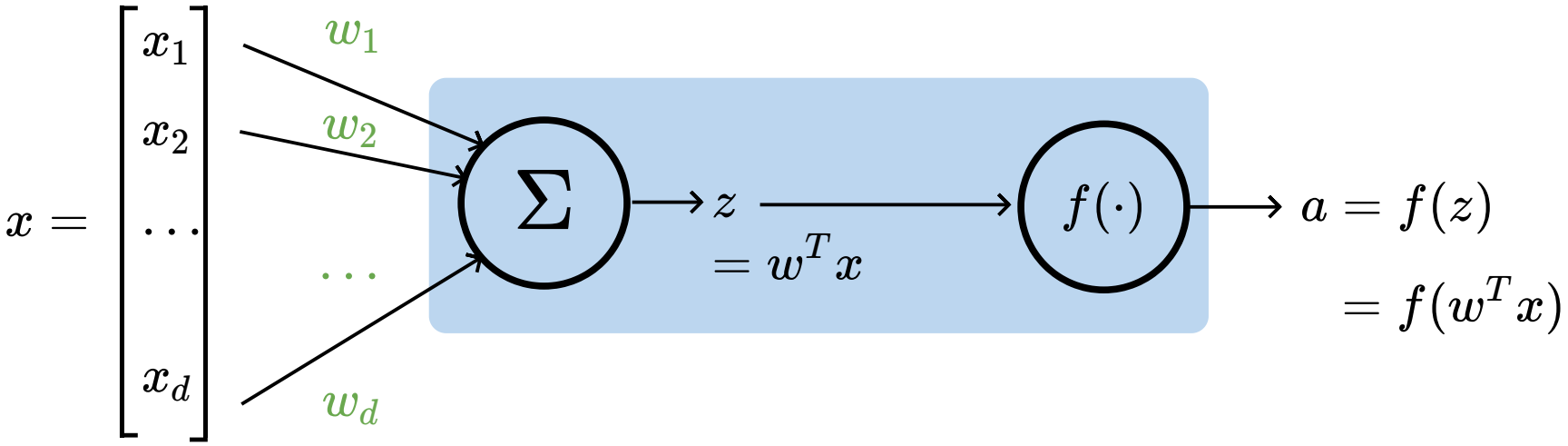


=



https://shenshen.mit.edu/demos/2layers.html?embed=sigmoid

A neuron:

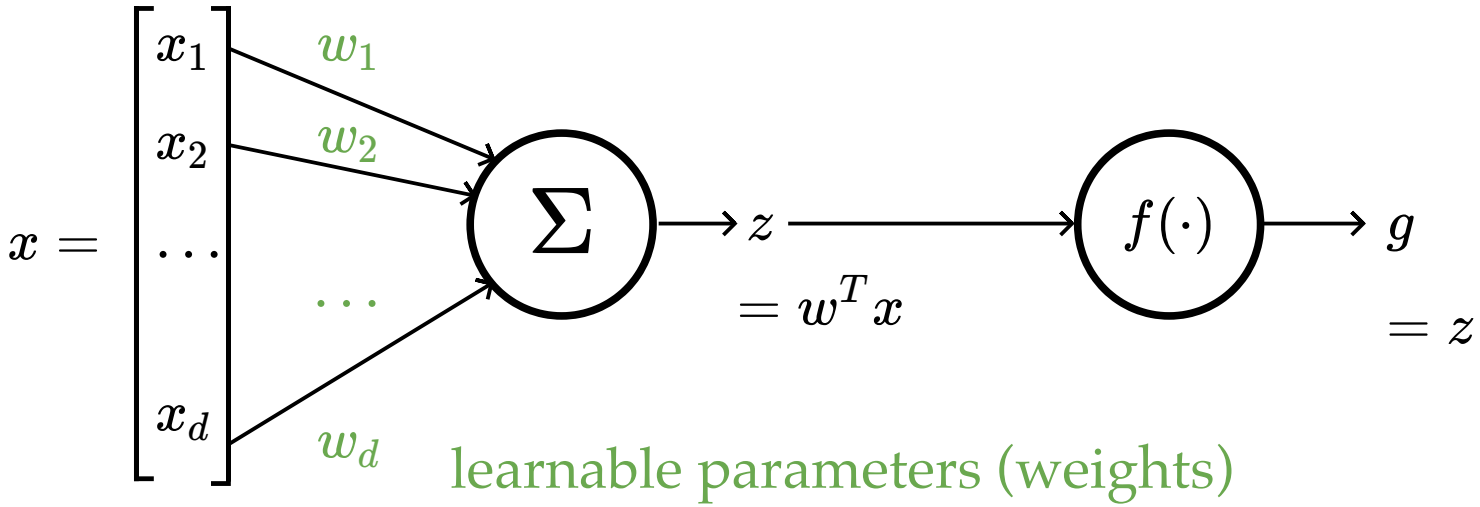


- x: d-dimensional input
- w: weights (i.e. parameters)
- z: pre-activation output
- f: activation function
- a: post-activation output

w: what the algorithm learns

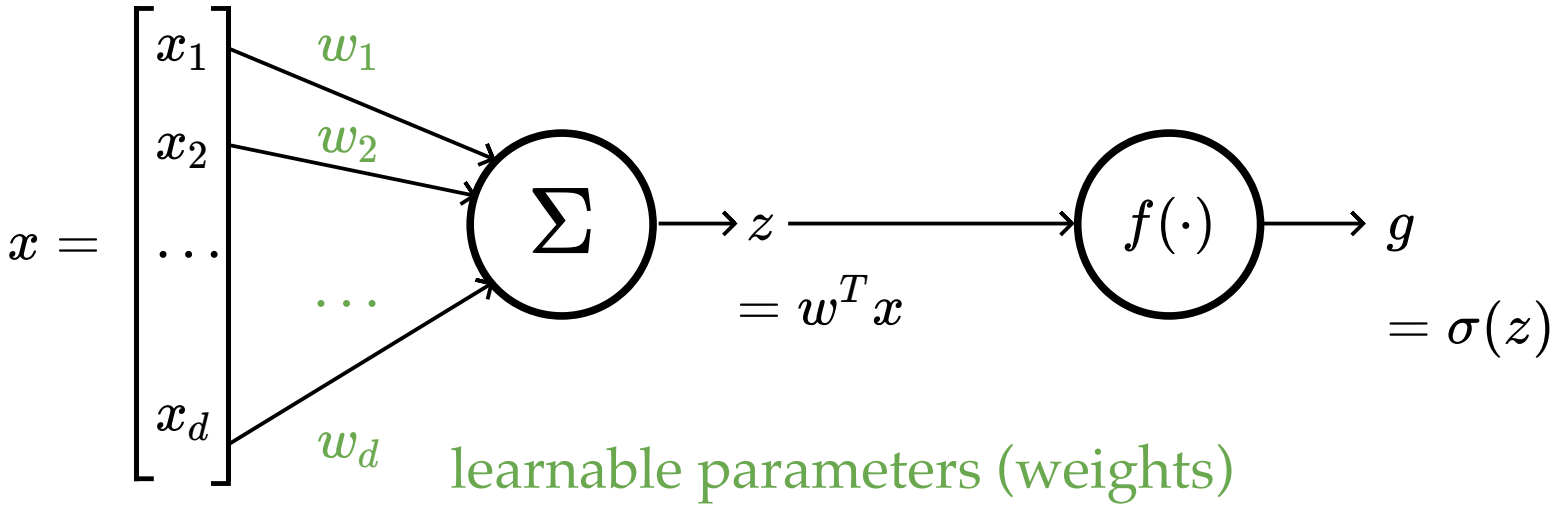
z: scalar
↓
f: what we engineers choose
↓
a: scalar

e.g. linear regressor represented as a computation graph



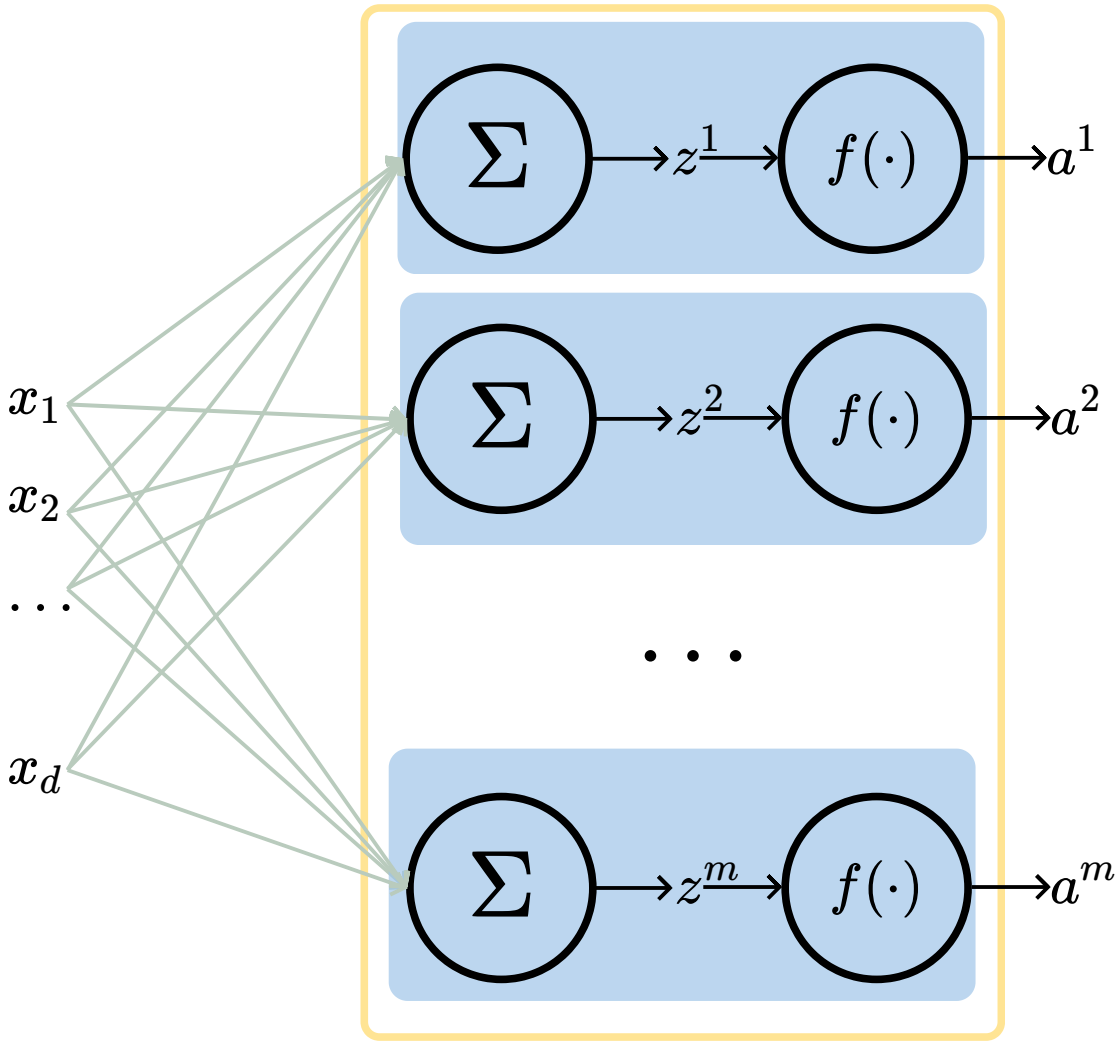
Choose activation $f(z) = z$

e.g. linear logistic classifier represented as a computation graph



Choose activation $f(z) = \sigma(z)$

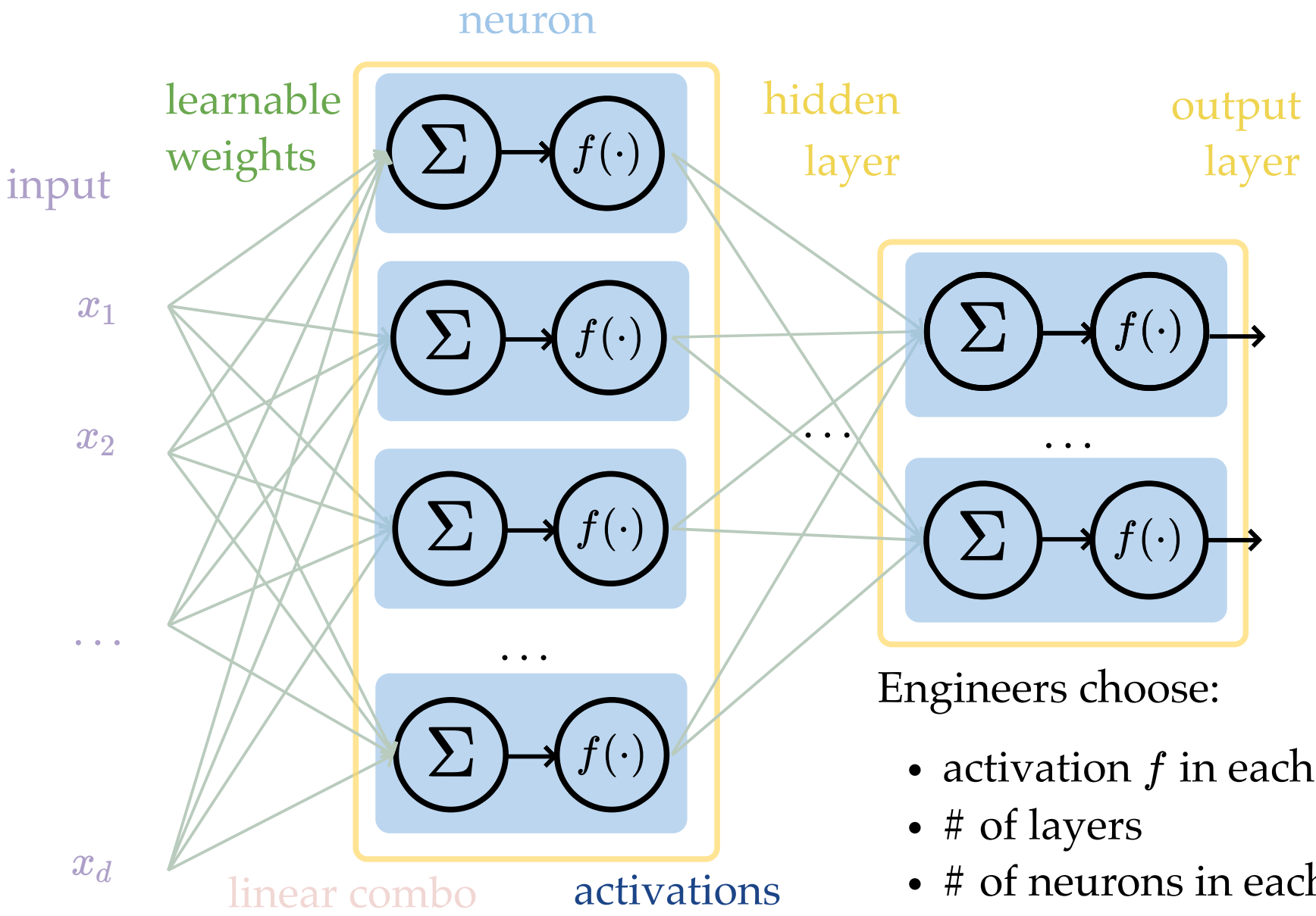
A layer:



learnable weights

- (# of neurons) = layer's output dimension.
- activations applied element-wise, e.g. z^1 won't affect a^2. (softmax in the output layer is a common exception)
- all neurons in a layer typically share the same f (mixed f is possible but complicates the math).
- layers are typically fully connected, each input x_i influences every a^j .

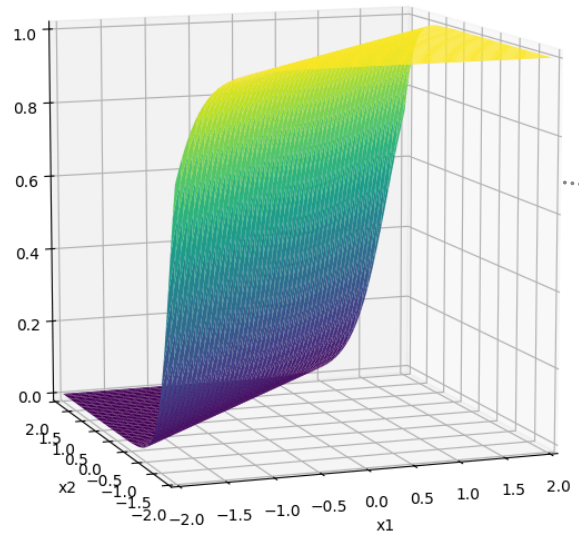
A (fully-connected, feed-forward) neural network:



Engineers choose:

- activation f in each layer
- # of layers
- # of neurons in each layer

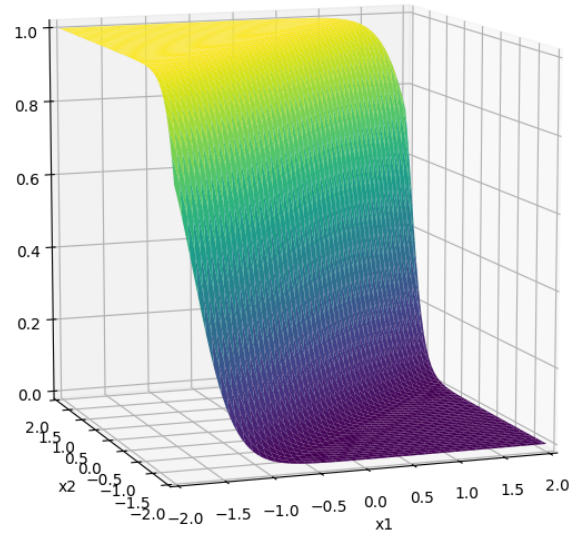
$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$

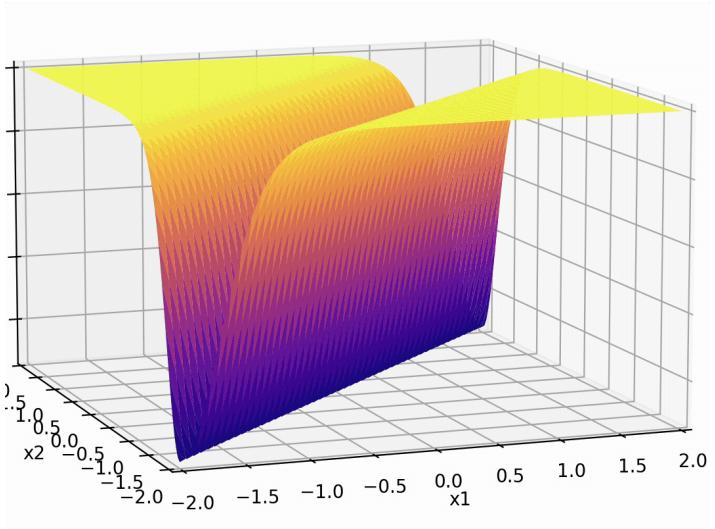


recall this example

some appropriately weighted sum

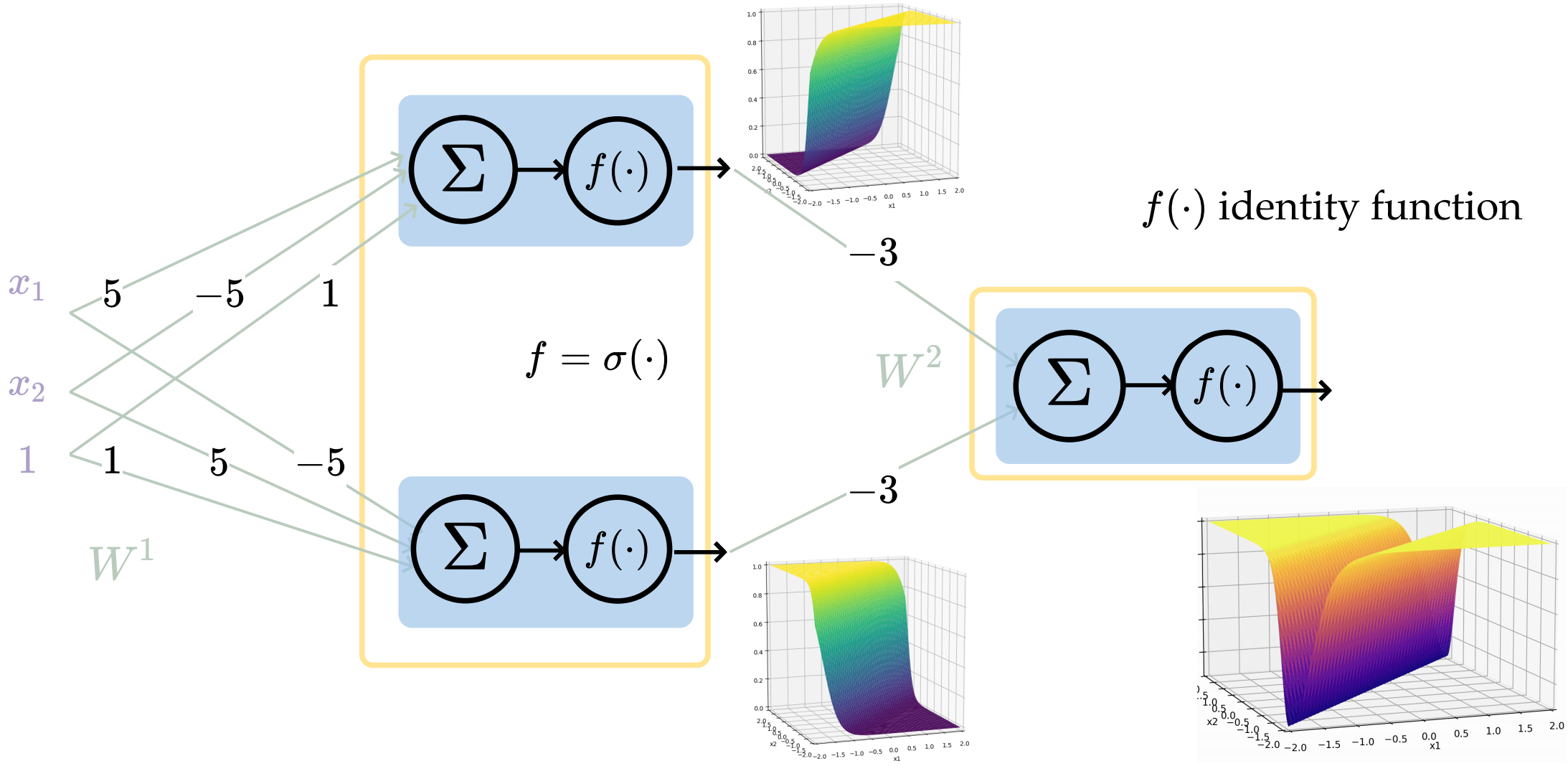
$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$





can be represented as:

$$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$$

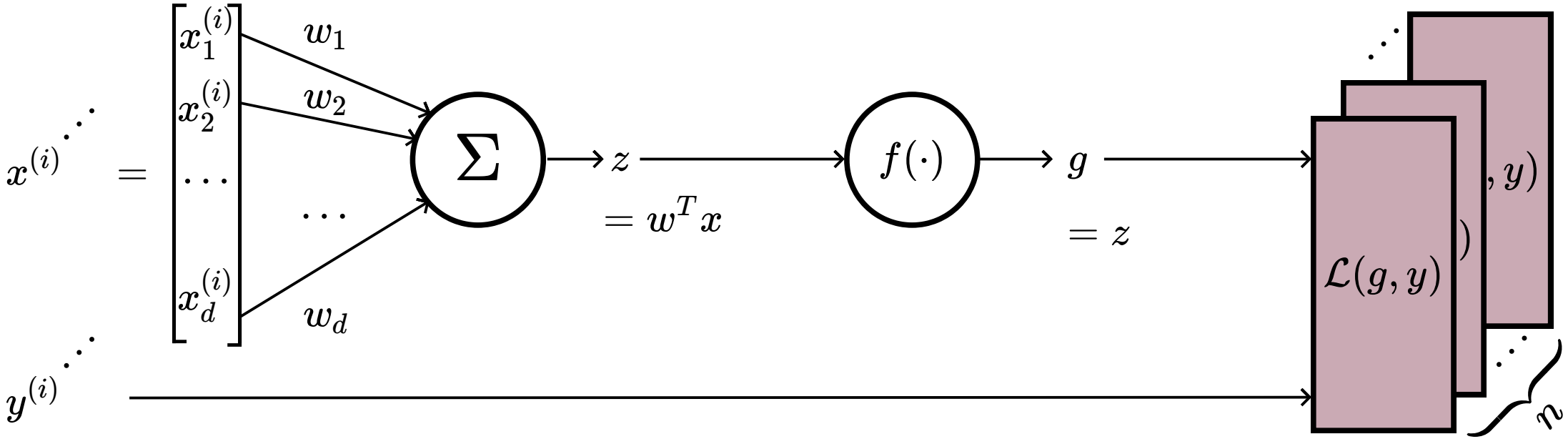


$f(\cdot)$ identity function

$$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$$

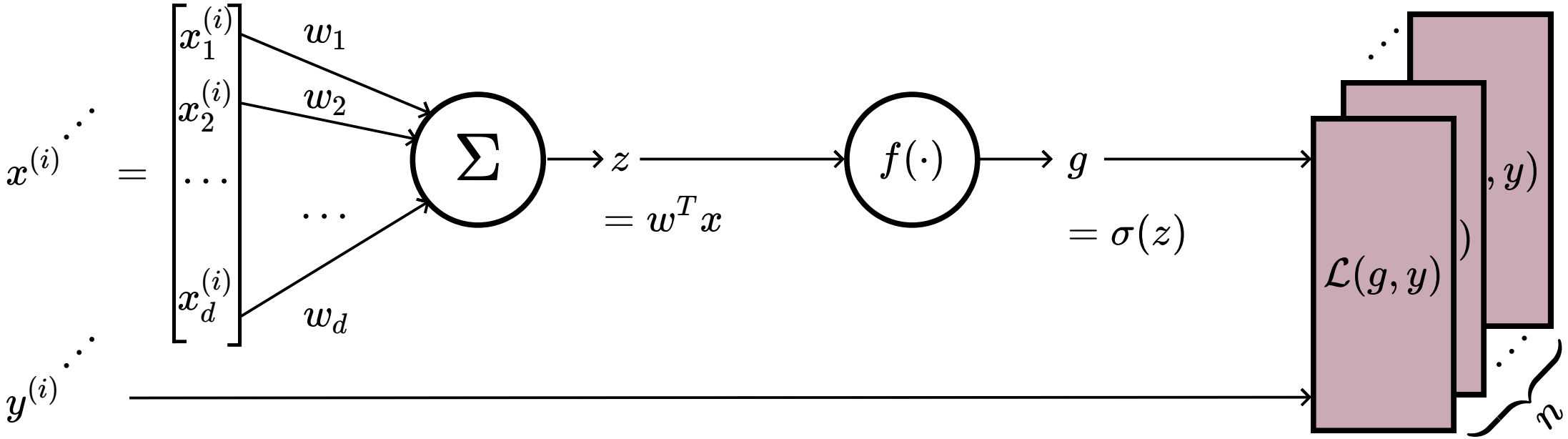
$$-3(\sigma_1 + \sigma_2)$$

e.g. forward-pass of a linear regressor



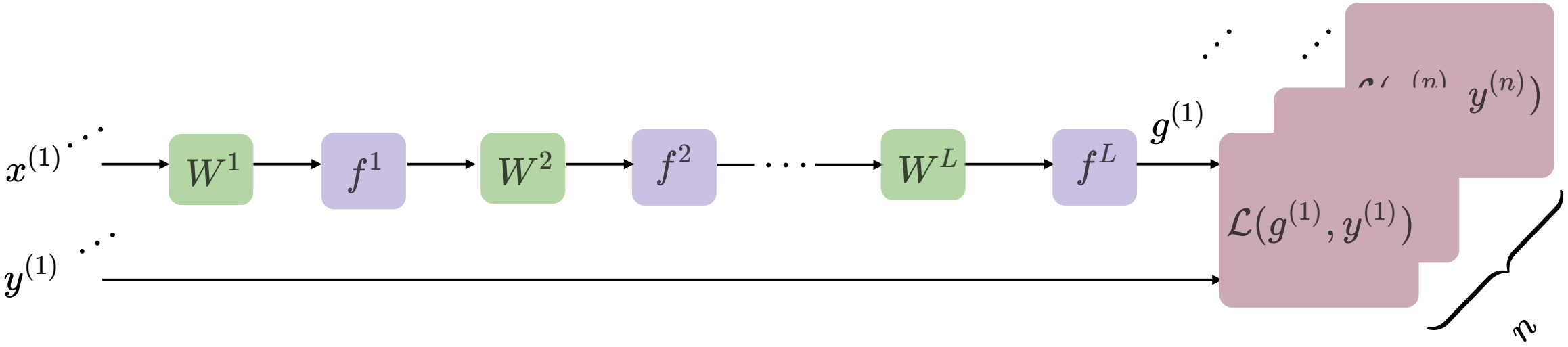
- Activation f is chosen as the identity function
- Evaluate the loss $\mathcal{L} = (g^{(i)} - y^{(i)})^2$
- Repeat for each data point, average the sum of n individual losses

e.g. forward-pass of a linear logistic classifier



- Activation f is chosen as the sigmoid function
- Evaluate the loss $\mathcal{L} = -[y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log \left(1 - g^{(i)}\right)]$
- Repeat for each data point, average the sum of n individual losses

Forward pass: evaluate, *given* the current parameters



- the model outputs $g^{(i)} = f^L\left(\ldots f^2\left(\ f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2\right)\ ; \ldots \mathbf{W}^L\right)$
- the loss incurred on the current data $\mathcal{L}(g^{(i)}, y^{(i)})$
- the training error $J = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(g^{(i)}, y^{(i)})$

linear combination

(nonlinear) activation

loss function

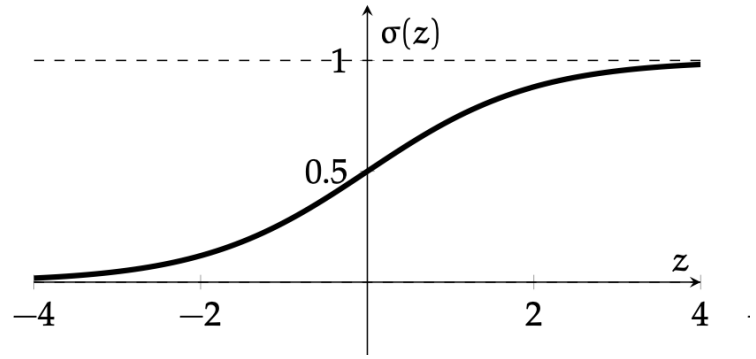
Outline

- Systematic feature transformations
- Neural networks
  - Terminology
  - Design choices

Hidden layer activation function f choices

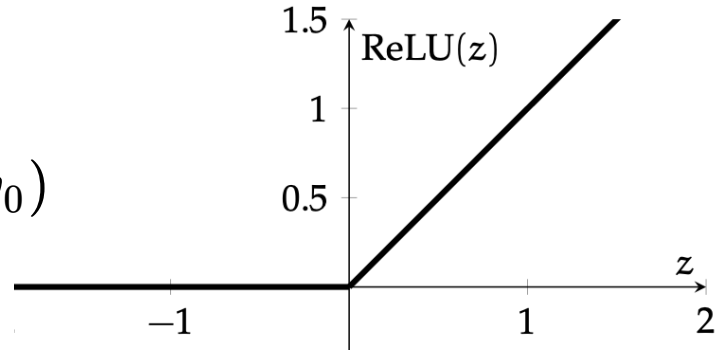
σ used to be the most popular

- firing rate of a neuron
- elegant gradient $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$



nowadays, the default choice:

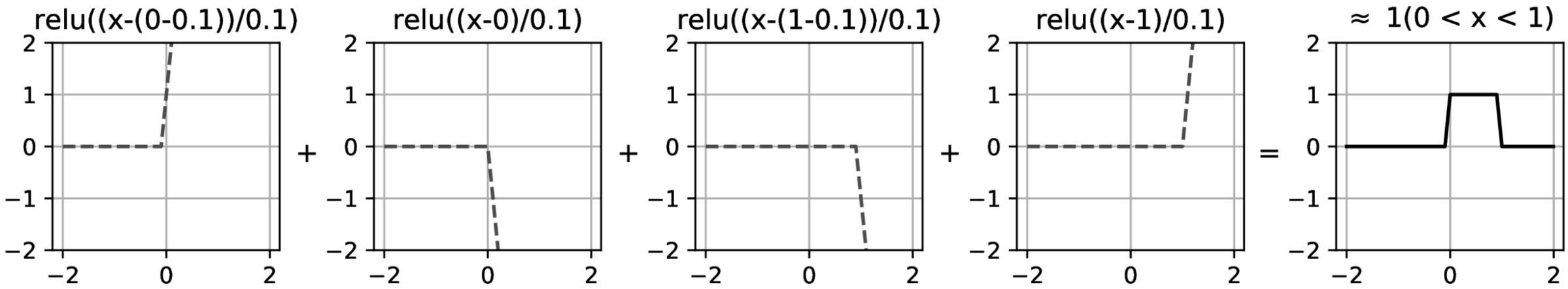
$$\mathrm{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z) \ = \max(0, w^T x + w_0)$$



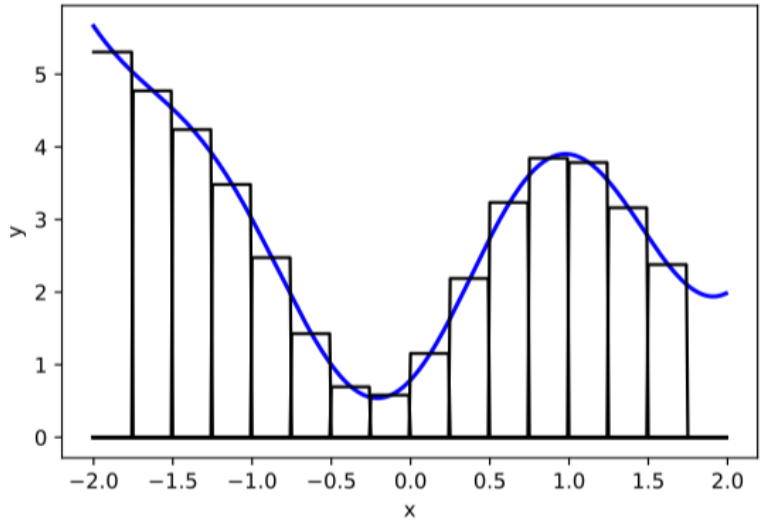
very simple function form (so is the gradient).

$$\frac{\partial \mathrm{ReLU}(z)}{\partial z} = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{otherwise} \end{cases}$$

compositions of ReLU(s) can be quite expressive

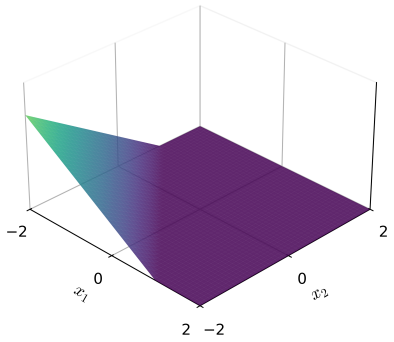


asymptotically, can approximate any continuous function arbitrarily well (for regression)

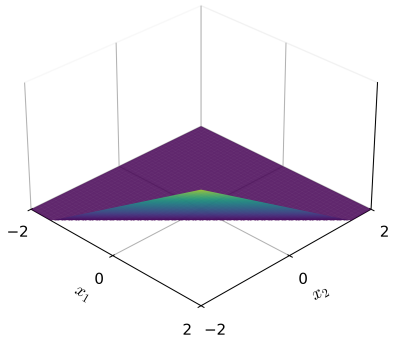


therefore can also approximate arbitrary decision boundaries (for classification)

compositions of ReLU(s) can be quite expressive

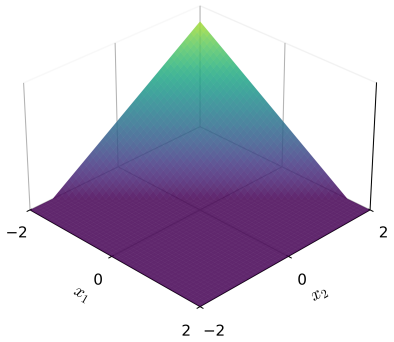


$\mathrm{ReLU}(-x_1 - x_2 - 1)$

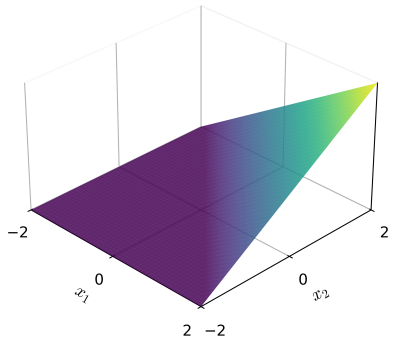


$\mathrm{ReLU}(x_1 - x_2 - 0.5)$

+

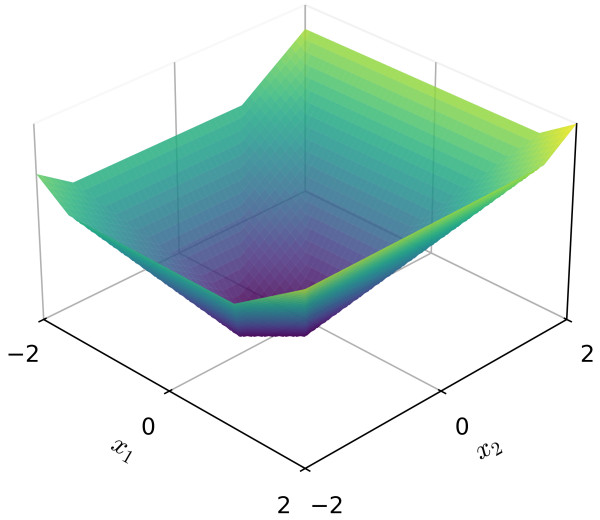


$\mathrm{ReLU}(-x_1 + x_2 - 0.5)$



$\mathrm{ReLU}(x_1 + x_2)$

=



just 4 ReLU neurons already carve out a non-trivial surface

https://shenshen.mit.edu/demos/2layers.html?embed=relu

- Width: # of neurons in layers
- Depth: # of layers

- Typically, increasing either the width or depth (with non-linear activation) makes the model more expressive, but it also increases the risk of overfitting.

However, in the realm of neural networks, the precise nature of this relationship remains an active area of research—for example, phenomena like the double-descent curve and scaling laws

(The demo won't embed in PDF. But the direct link below works.)

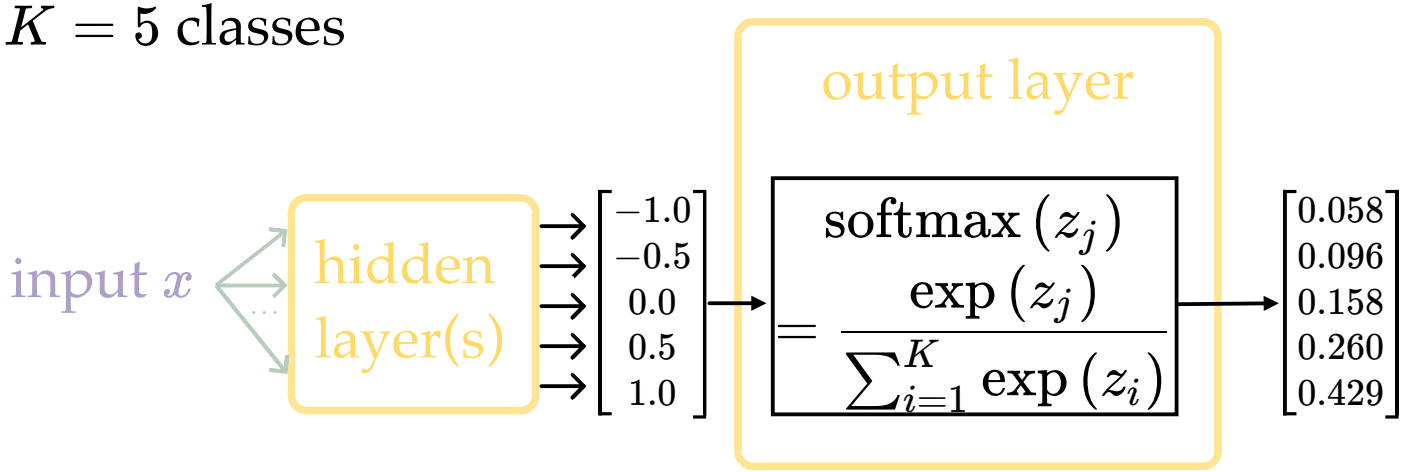
https://shenshen.mit.edu/demos/nn/playground/

output layer design choices

- # neurons, activation, and loss depend on the high-level goal.
- typically straightforward.
- Multi-class setup: if predict *one and only one* class out of K possibilities, then

  output layer: K neurons, softmax activation, cross-entropy loss

  e.g., say $K = 5$ classes



- For other multi-class settings, see hw/lab.

Summary

- Linear models are convenient but lack expressiveness for most real-world tasks.
- A **fixed** non-linear feature transformation lets us use linear methods on complex problems, but designing features by hand gets tedious.
- Neural networks automate feature learning: layers alternate parameterized linear maps with non-linear activations (typically ReLU).
- For classification outputs, we apply sigmoid (binary) or softmax (multi-class).

Next time: How do we *train* neural networks? (gradient descent + backpropagation)