

<https://introml.mit.edu>

6.390 Intro to Machine Learning

Lecture 5: Features and Neural Networks I

Shen Shen

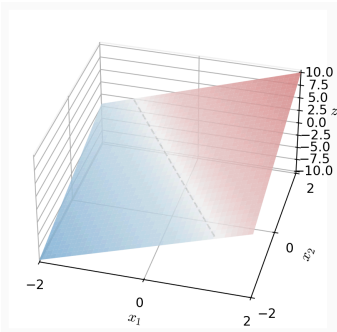
Oct 5, 2026

2:30pm, Room 10-250

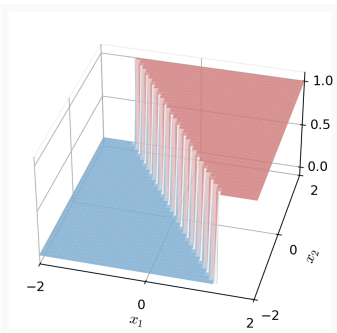
[Slides and Lecture Recording](#)

Recall:

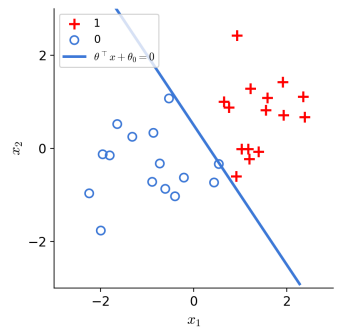
step



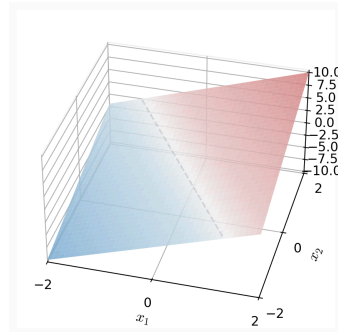
$$z = \theta^\top x + \theta_0$$



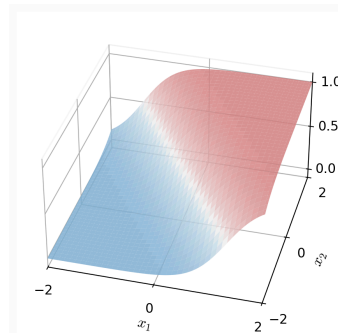
$$g = \text{step}(z)$$



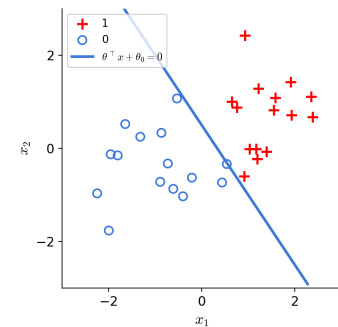
logistic



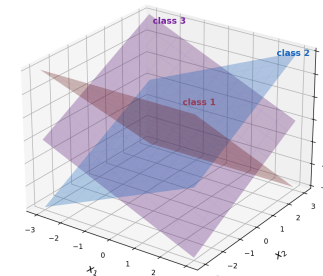
$$z = \theta^\top x + \theta_0$$



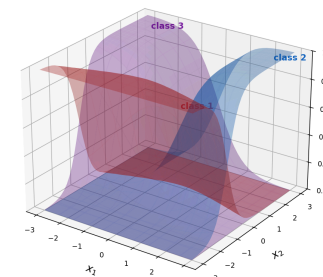
$$g = \sigma(z)$$



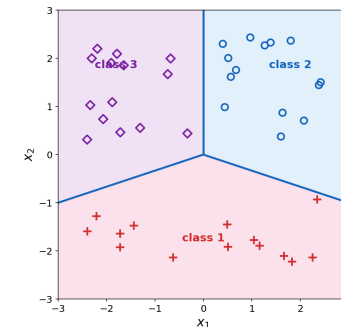
multi-class



$$z_k = \theta_k^\top x + \theta_{0,k}$$



$$g = \text{softmax}(z_1, \dots, z_K)$$



decision boundary is *linear* in x

[mock-up post generated by Claude Opus 5.5, not real]



The New York Times
@nytimes · 2h

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.

1.2K

4.8K

21K

2.3M

[real front-page of NY 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 be-

ings, 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 s
huma

reme
tion
nary
what
cards

La
to re
their
late
speed
lang

Mr
ciple
build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

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.

ic computer
1958 New York Times...

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

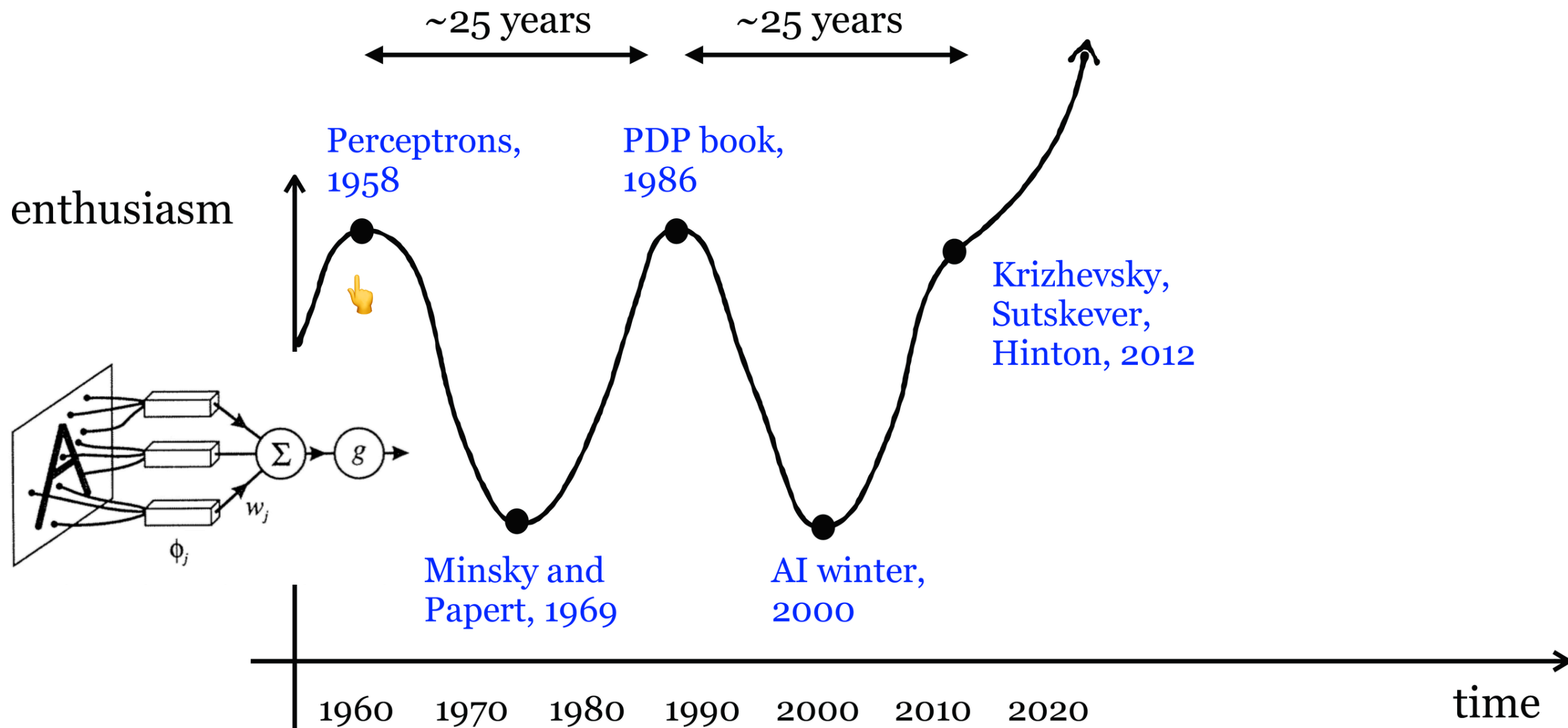
In the first fifty trials, the machine made no distinction be-

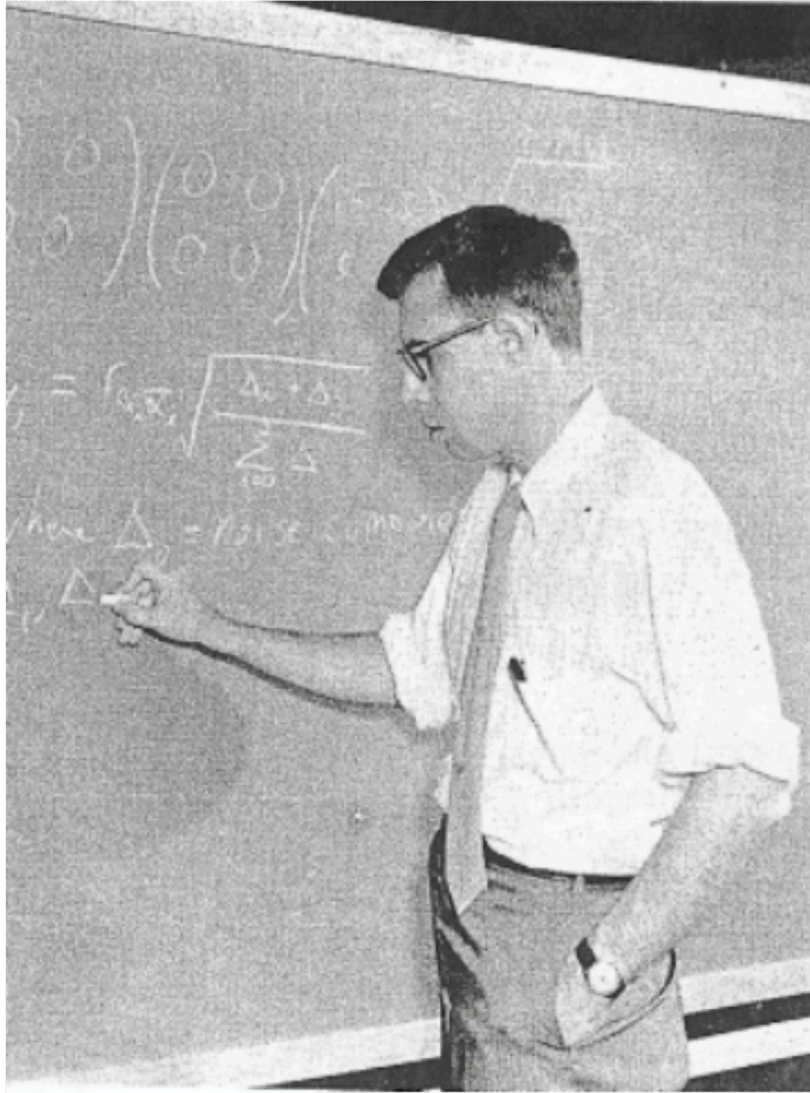
d
it
at

d
e
al
r
d
ll
ic
g
e-
0
in

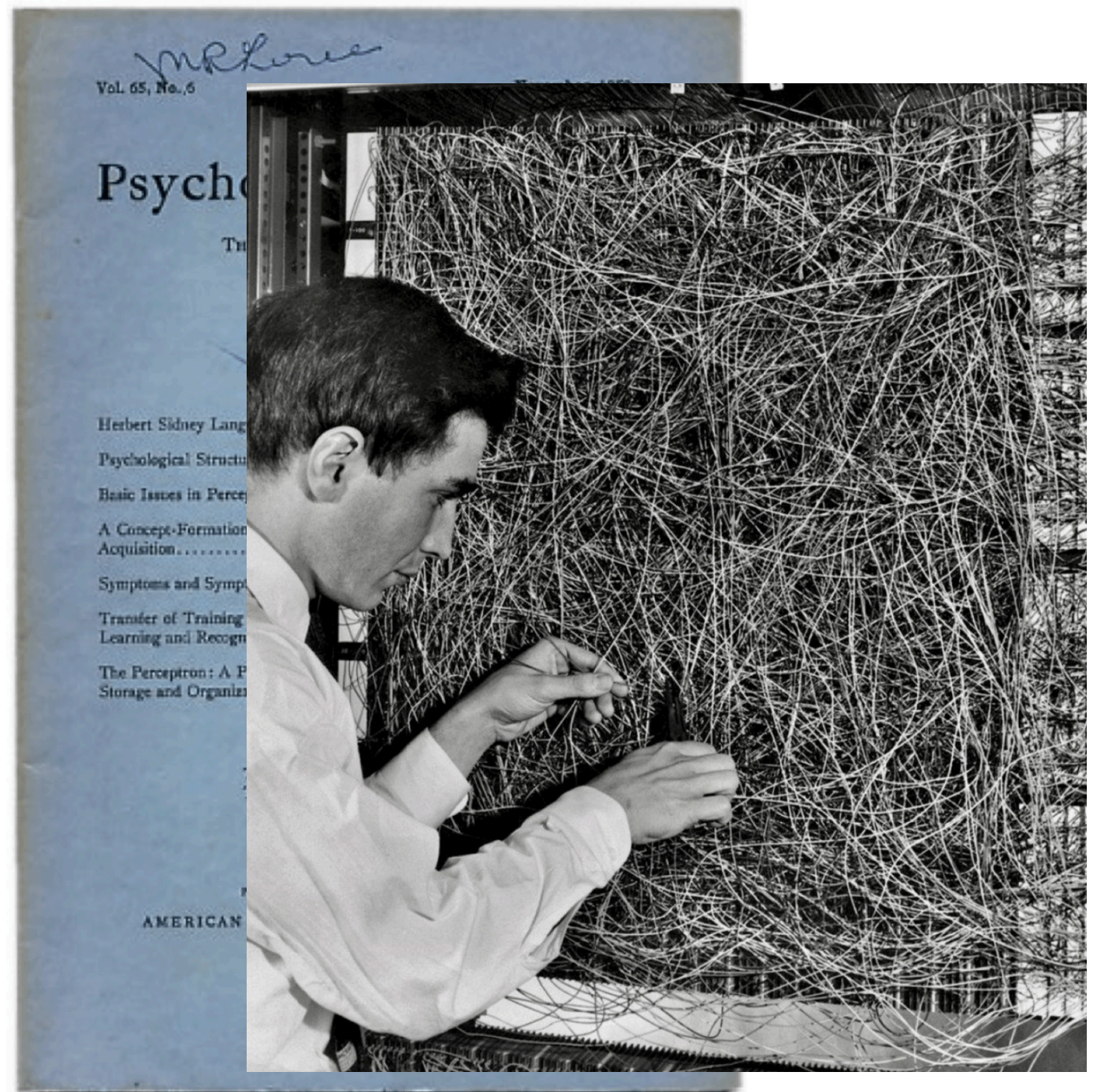
has 10,000,000 responsive cells, including 100,000,000 connections with the eyes.

Linear classification helped kick off the first wave of AI enthusiasm.



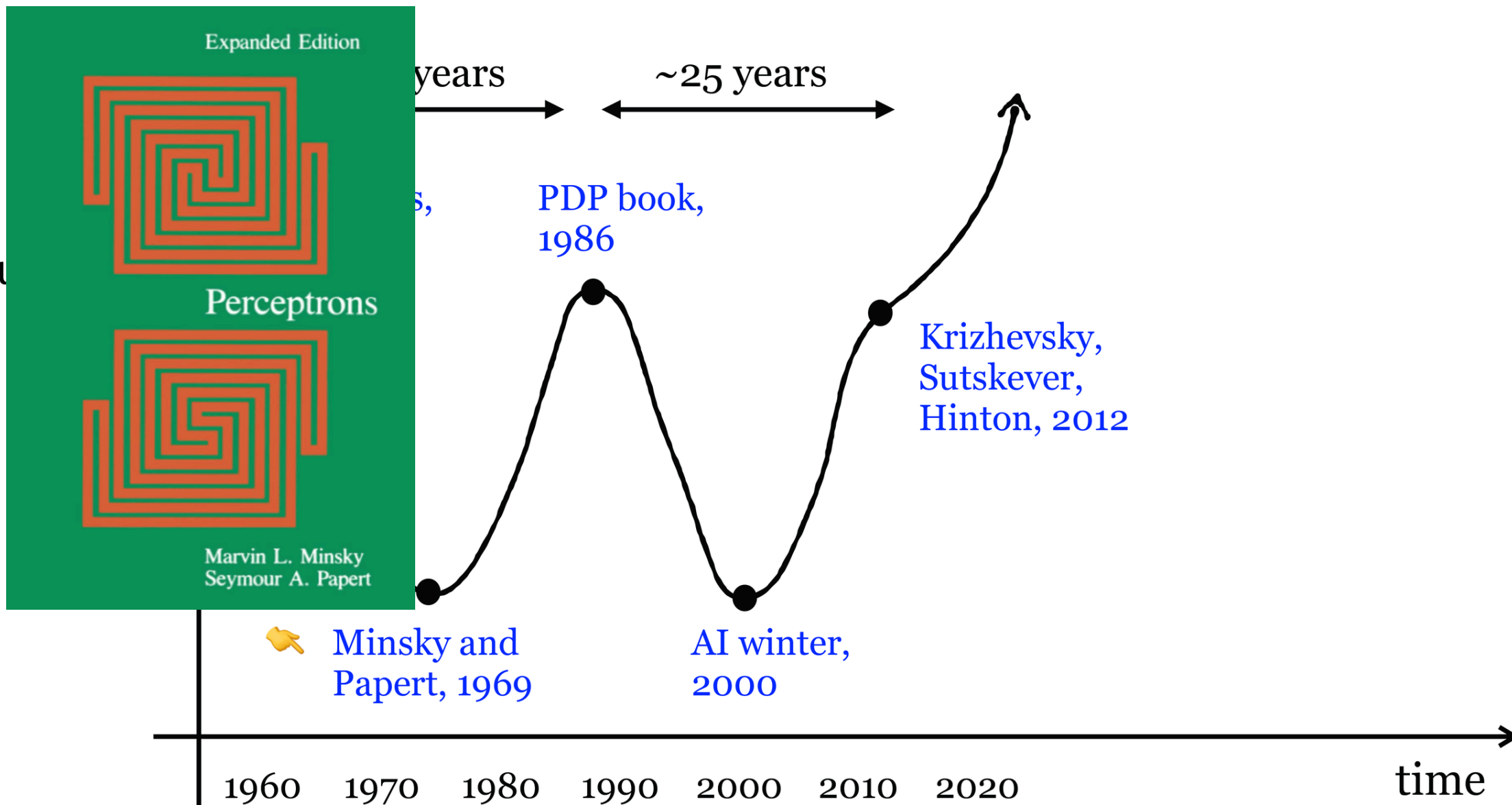


http://www.ecse.rpi.edu/homepages/nagy/PDF_chrono/2011_Nagy_Pace_FR.pdf. Photo by George Nagy

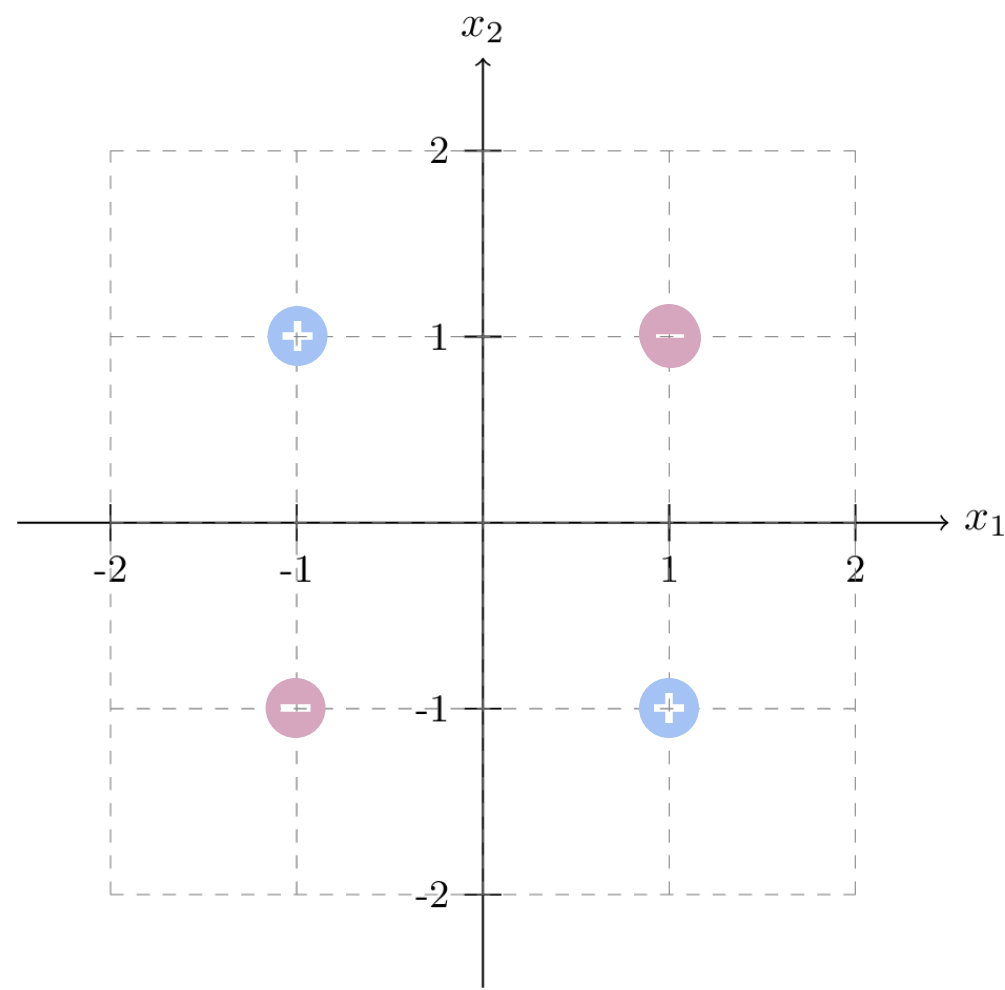


<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>

enthu



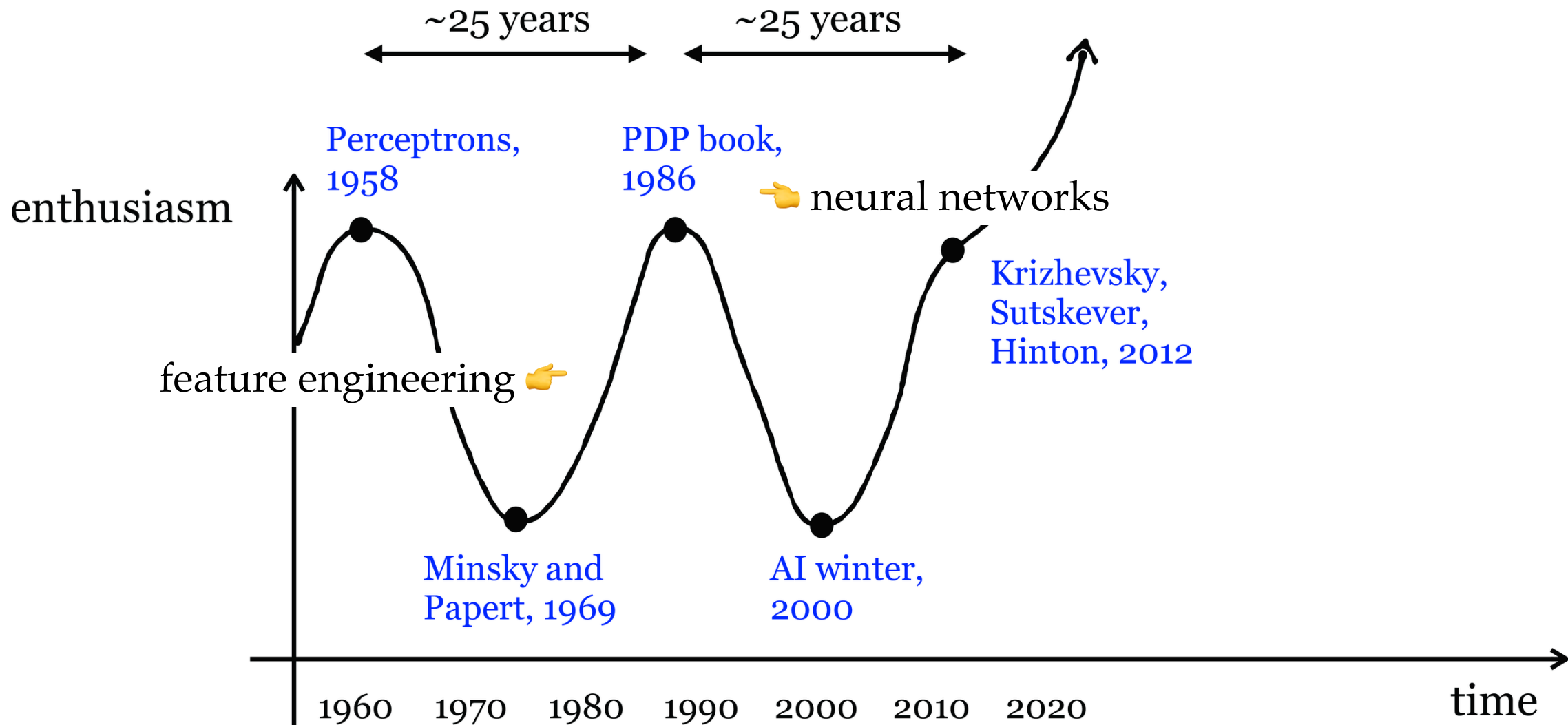
XOR dataset



Not **linearly** separable.

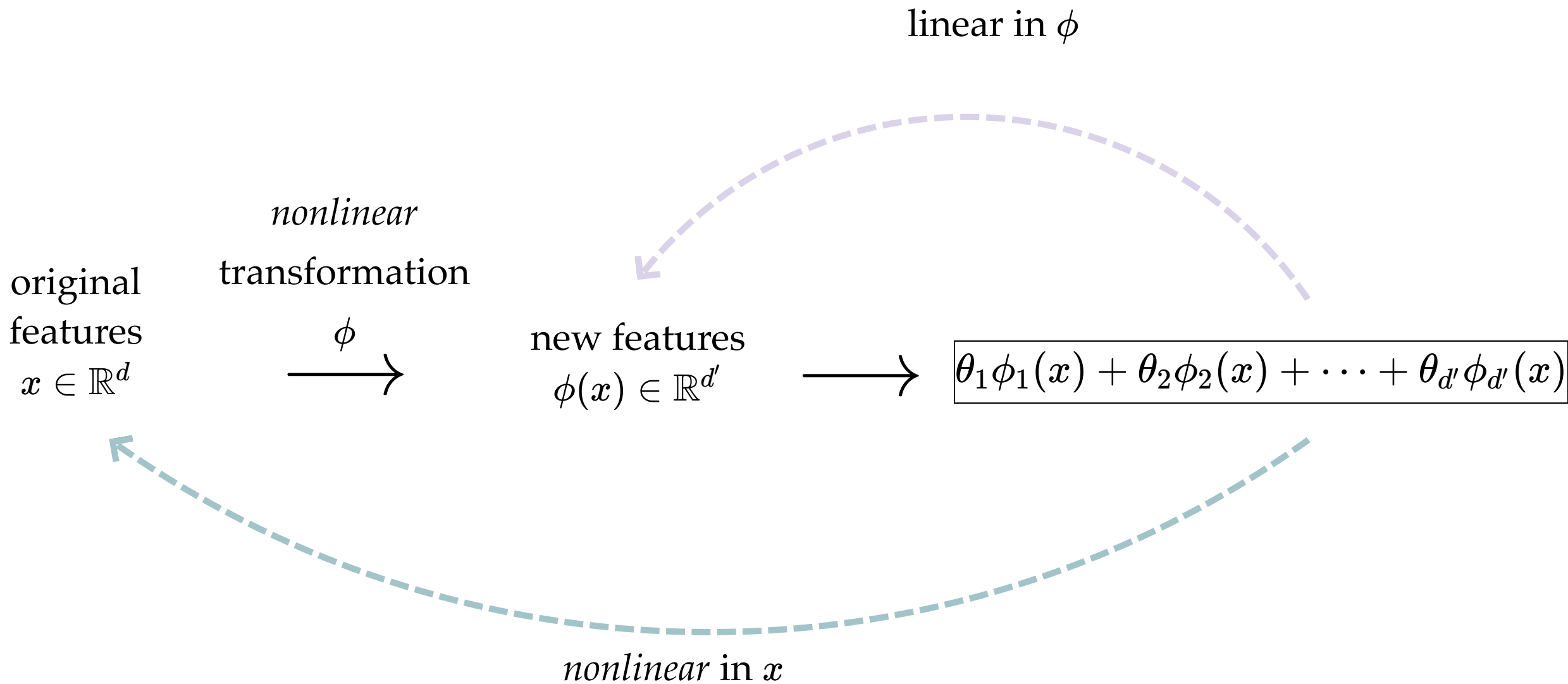
~~Linear tools cannot solve interesting tasks.~~

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

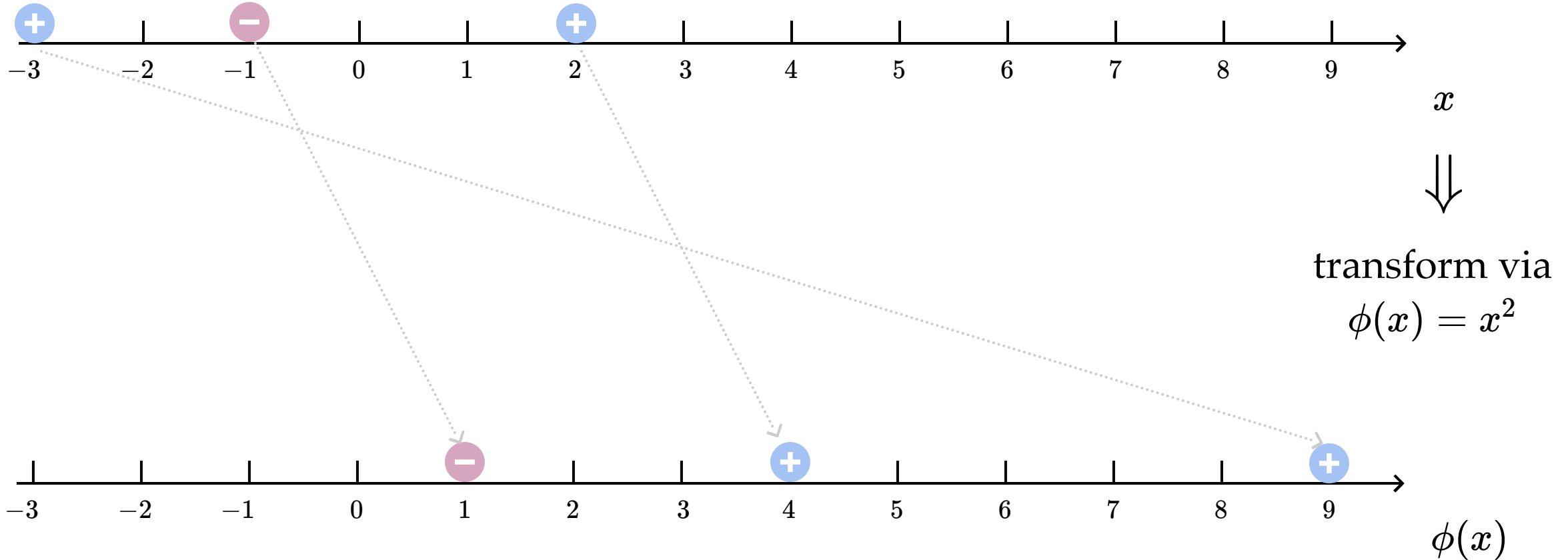


Outline

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

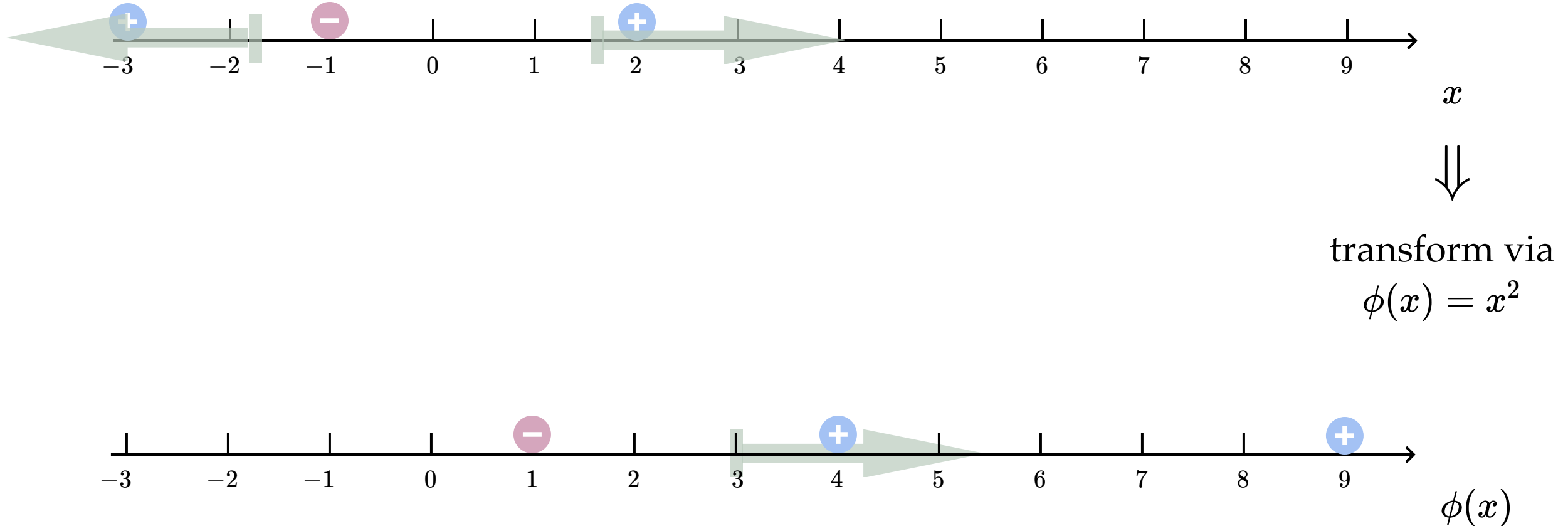


Not linearly separable in x space



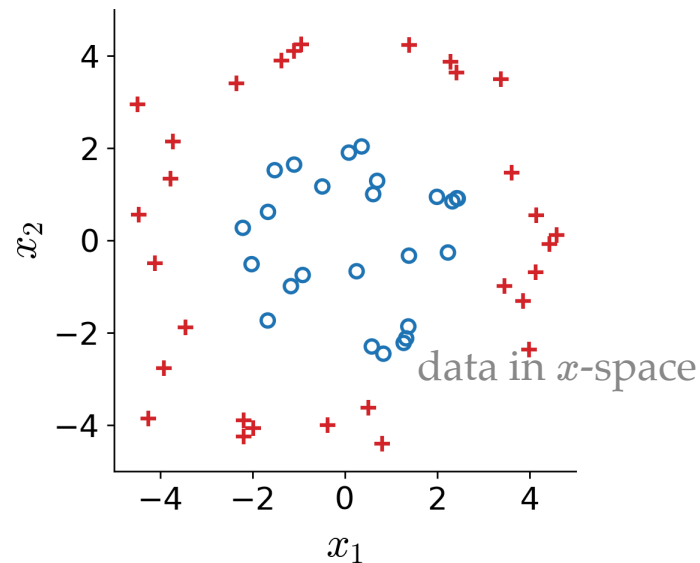
Linearly separable in $\phi(x) = x^2$ space

Separable by a *nonlinear* rule in x space, e.g. predict positive if $x^2 \geq 3$

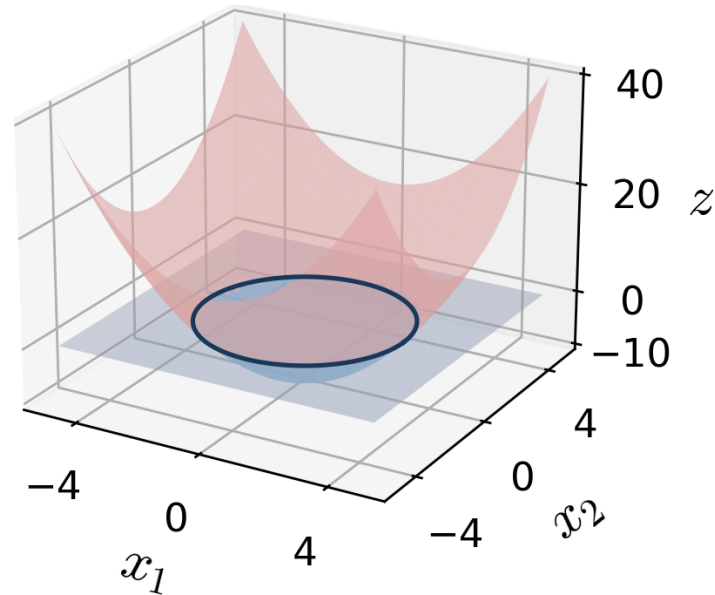


Linearly separable in $\phi(x) = x^2$ space, e.g. predict positive if $\phi \geq 3$

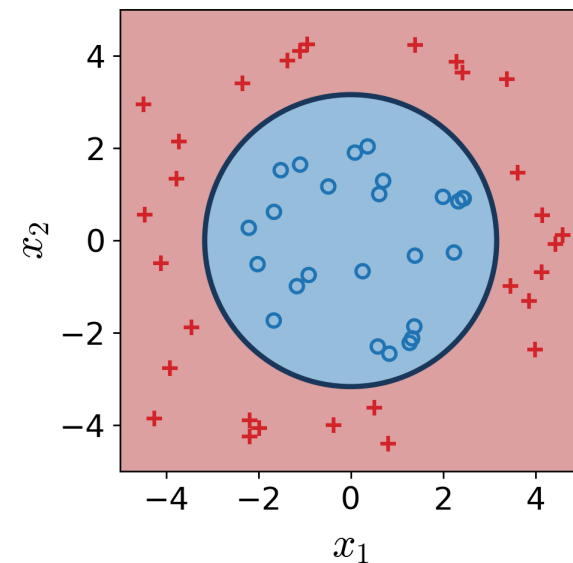
data and classifier in x space



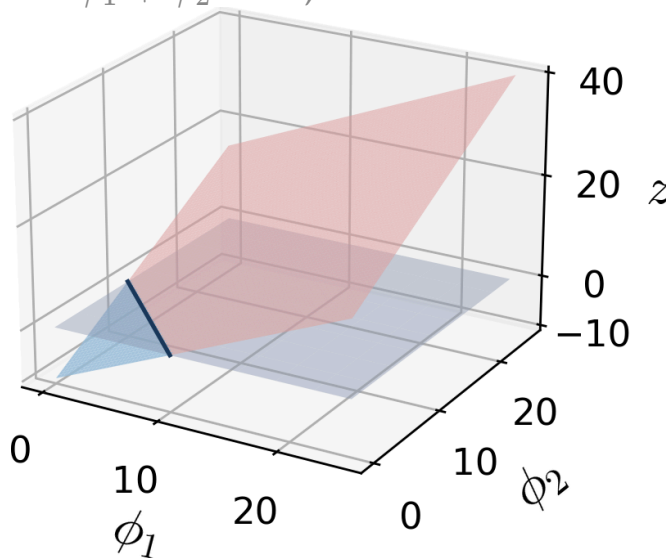
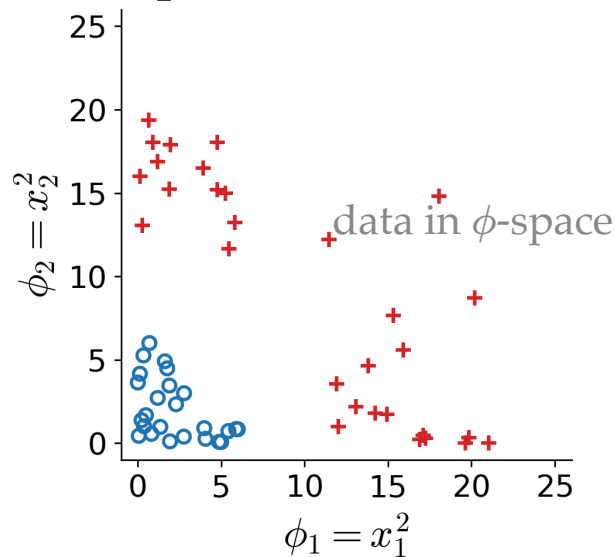
$$z = x_1^2 + x_2^2 - 10, \text{ threshold at } z=0$$



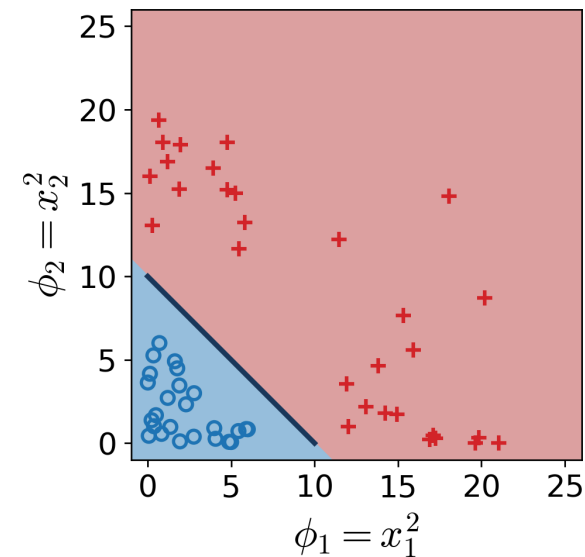
$$x_1^2 + x_2^2 = 10: \text{nonlinear in } x$$



... in ϕ space, where $\phi(x) = (x_1^2, x_2^2)$ $z = \phi_1 + \phi_2 - 10$, threshold at $z=0$

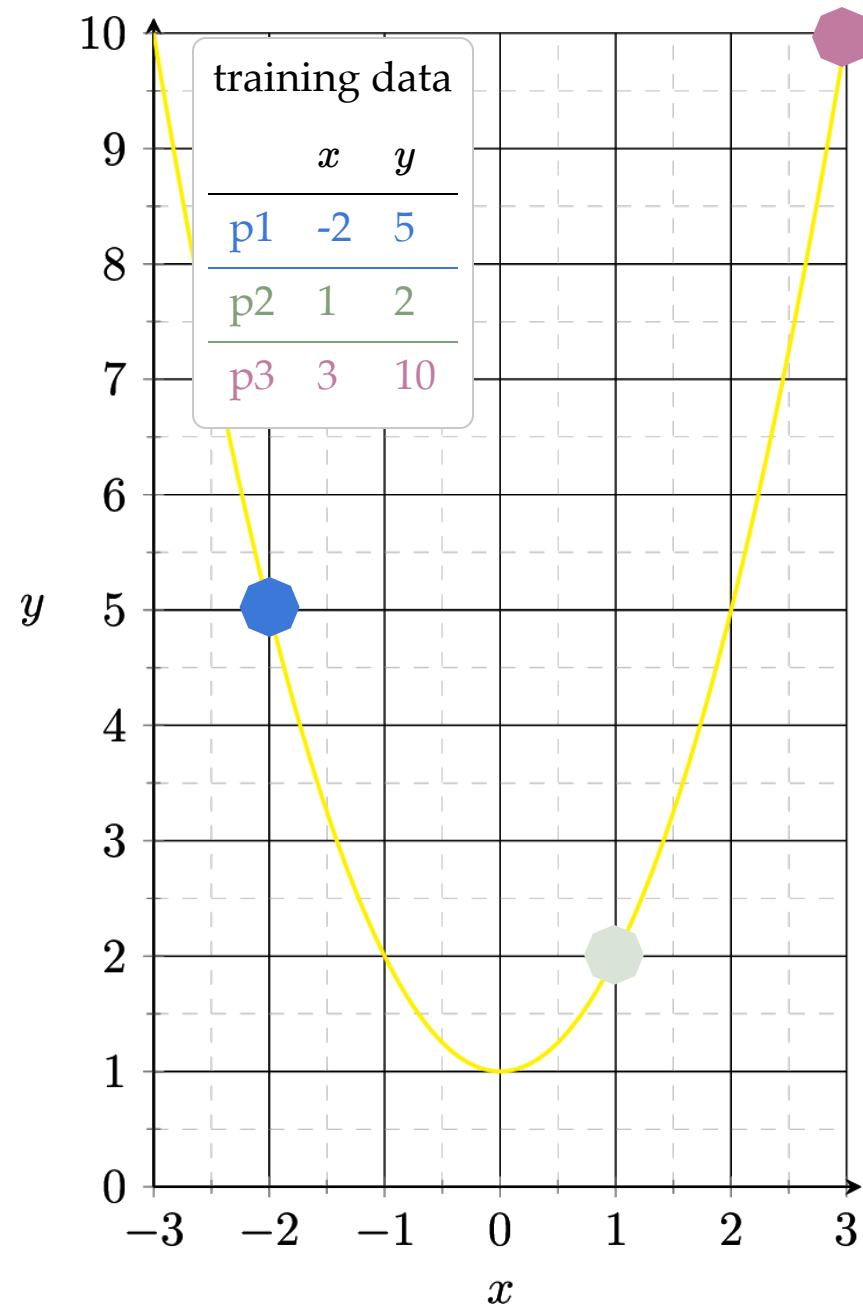


$$\phi_1 + \phi_2 = 10: \text{linear in } \phi$$



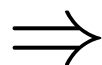
decision boundary is *linear* in ϕ , *nonlinear* in x

nonlinear regression

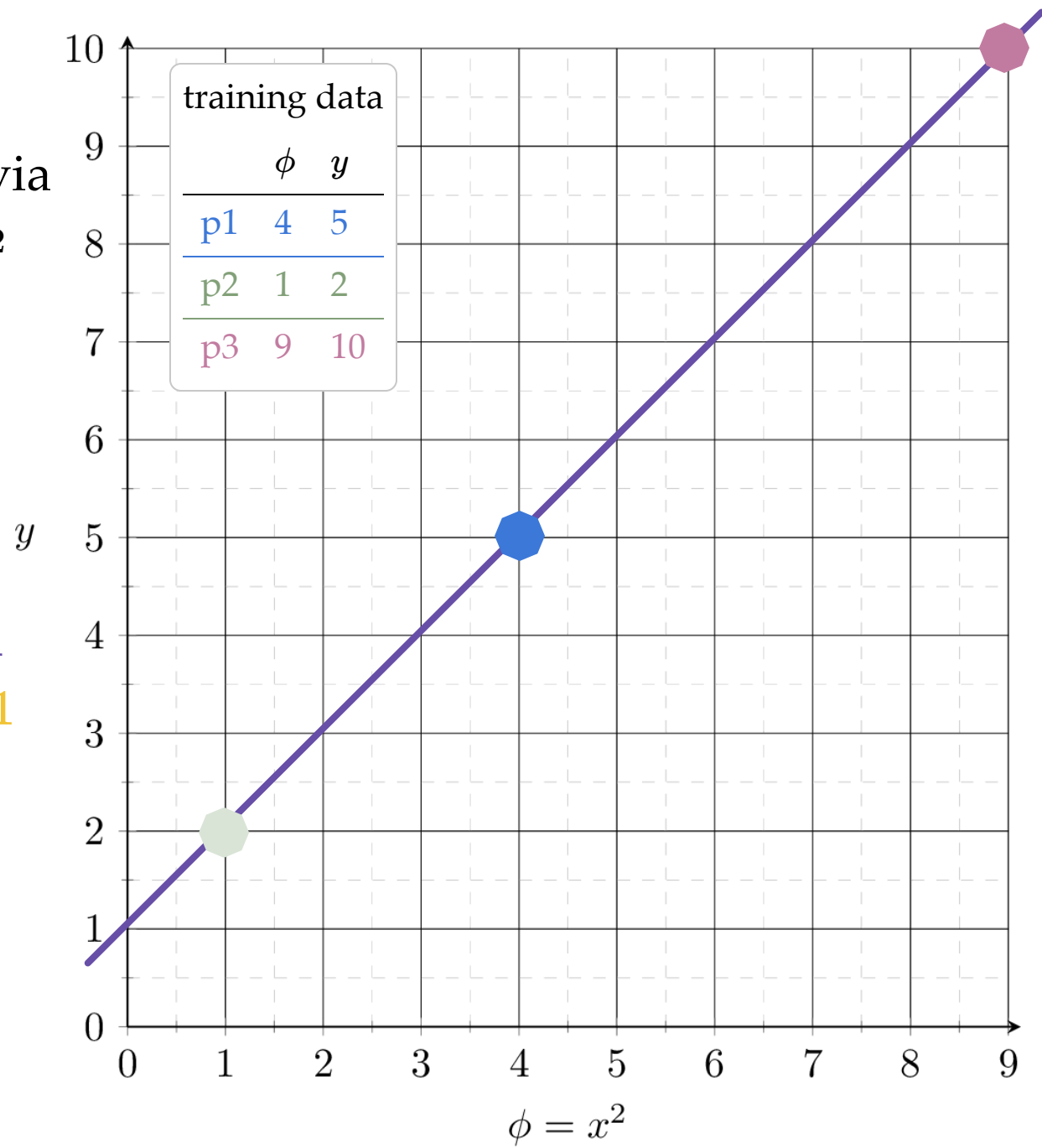


transform via

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



$$g = \phi + 1$$
$$g = x^2 + 1$$



systematic construction of polynomial features

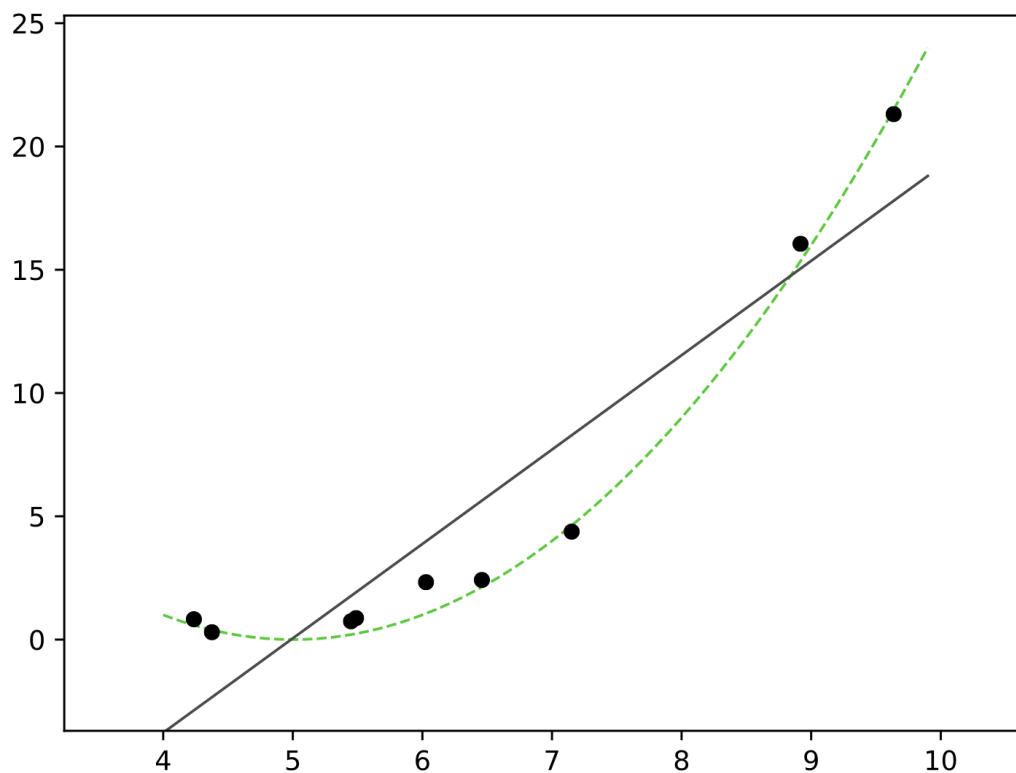
	$d = 1, \text{ features: } x_1$	$d = 2, \text{ features: } x_1, x_2$
$k = 0$	1	1
$k = 1$	1, x_1	1, x_1, x_2
$k = 2$	1, x_1 , x_1^2	1, x_1, x_2 , x_1^2, x_1x_2, x_2^2
$k = 3$	1, x_1 , x_1^2 , x_1^3	1, x_1, x_2 , x_1^2, x_1x_2, x_2^2 , $x_1^3, x_1^2x_2, x_1x_2^2, x_2^3$
$\vdots$	$\vdots$	$\vdots$

number of features $d' = \binom{d+k}{k}$; $d = 2, k = 3$ gives 10

- The basis holds every monomial of the original features with total degree up to k
- Fixing d and k **fixes** the basis.

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

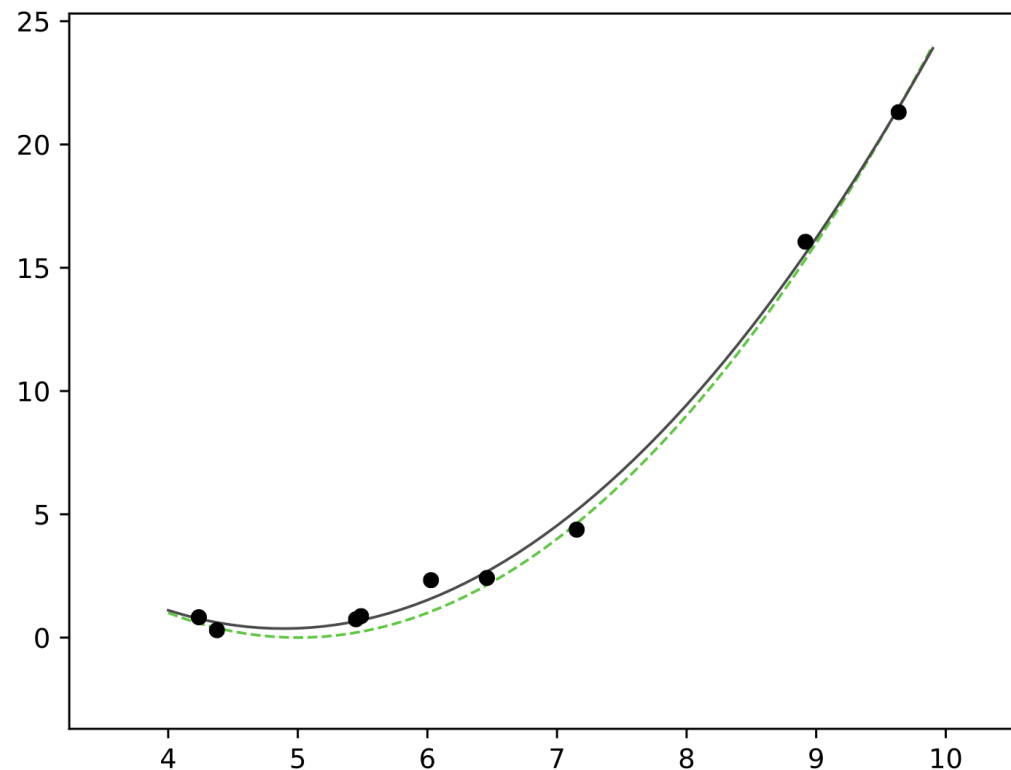
$k = 1$



$$h = \theta_0 + \theta_1 x$$

Learn 2 parameters — degree-1

$k = 2$

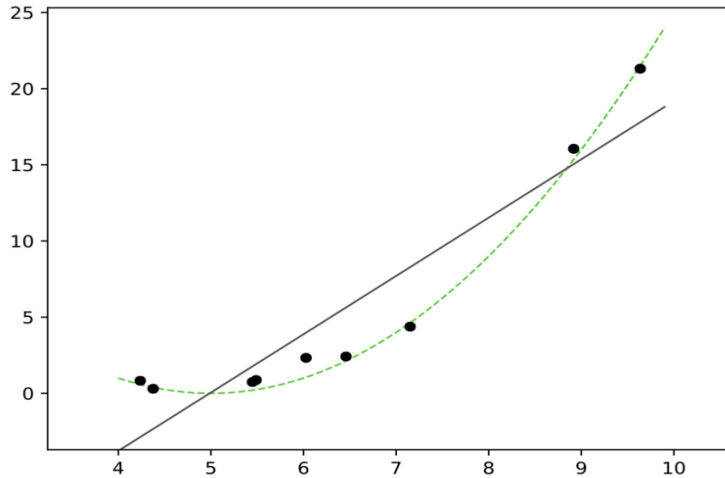


$$h = \theta_0 + \theta_1 x + \theta_2 x^2$$

Learn 3 parameters — degree-2

Underfitting

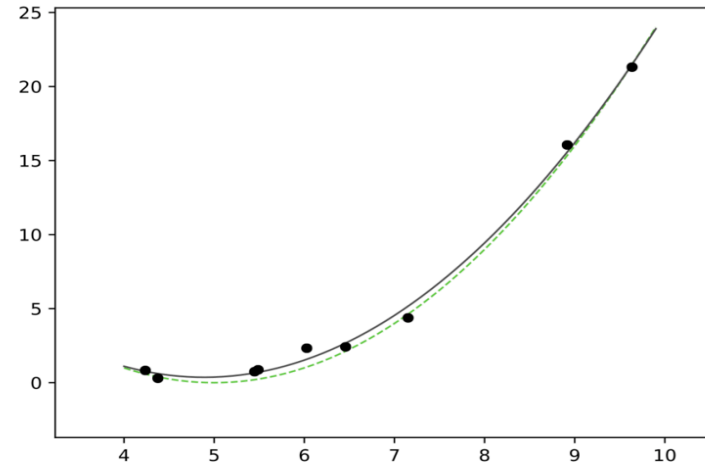
$k = 1$



high error on train set
high error on test set

Appropriate model

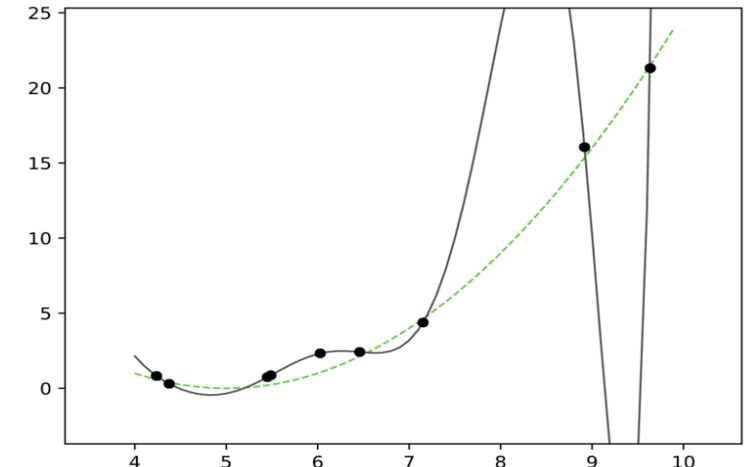
$k = 2$



low error on train set
low error on test set

Overfitting

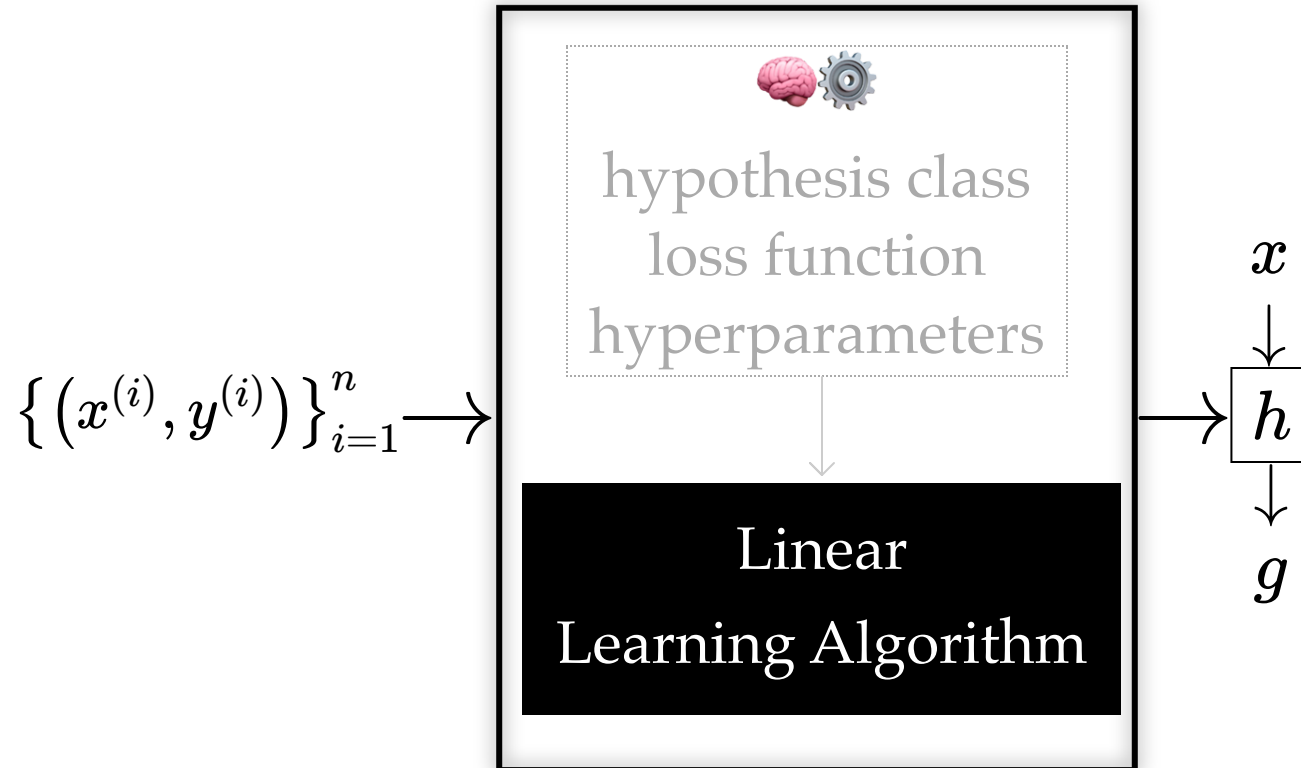
$k = 10$



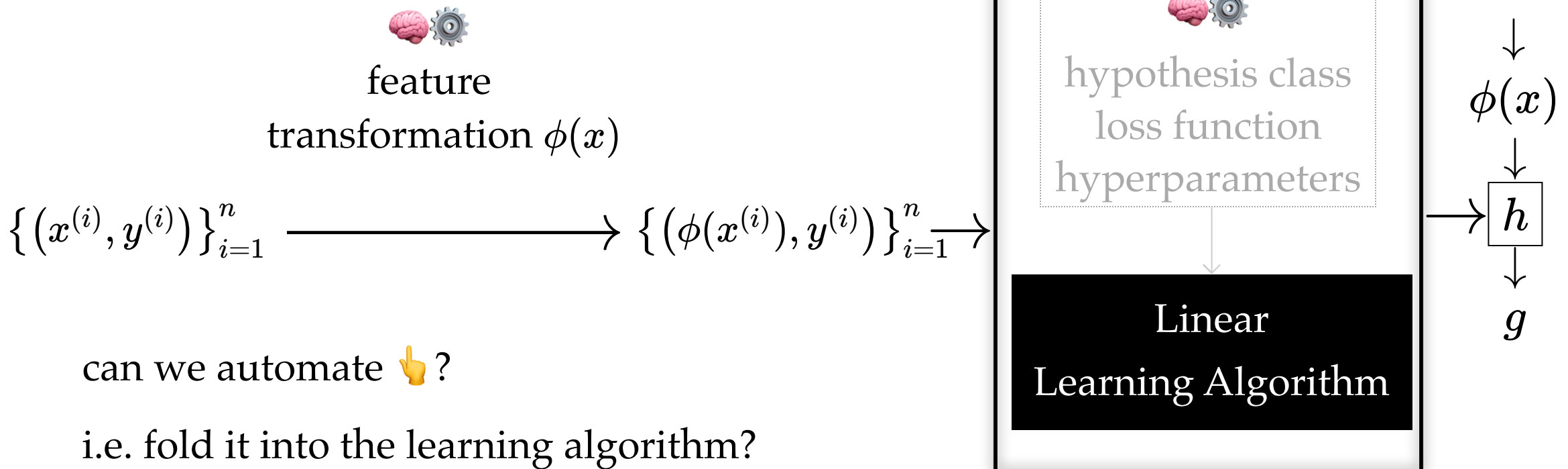
low error on train set
high error on test set

- 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.

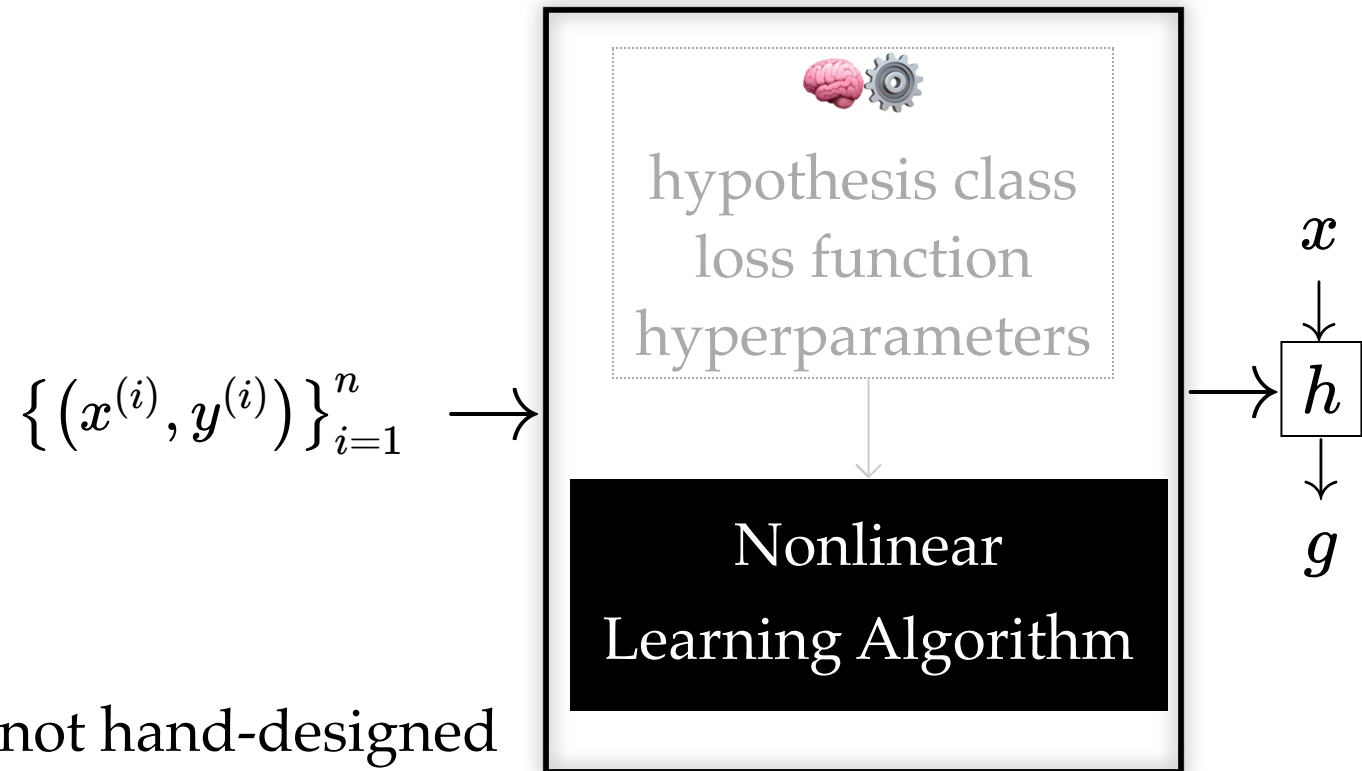
Previously:



today, so far:

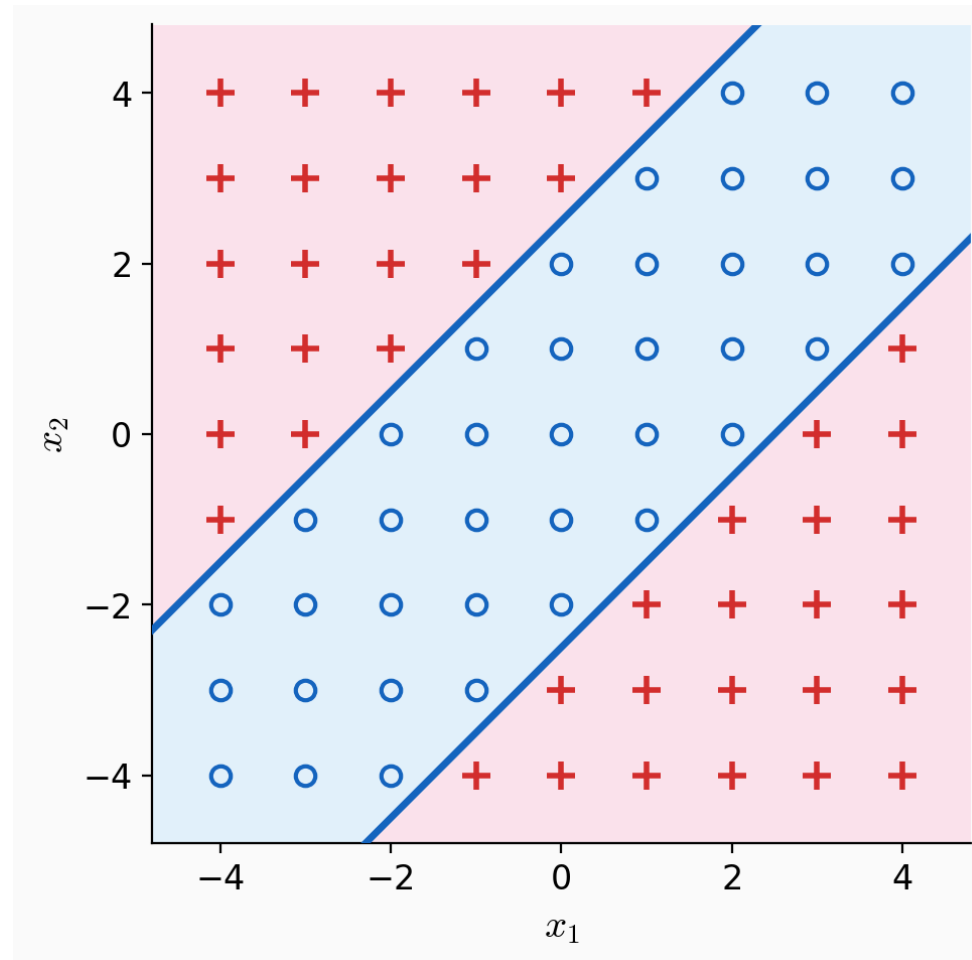


neural networks:



the nonlinearity ϕ is learned, not hand-designed

Something to think about:



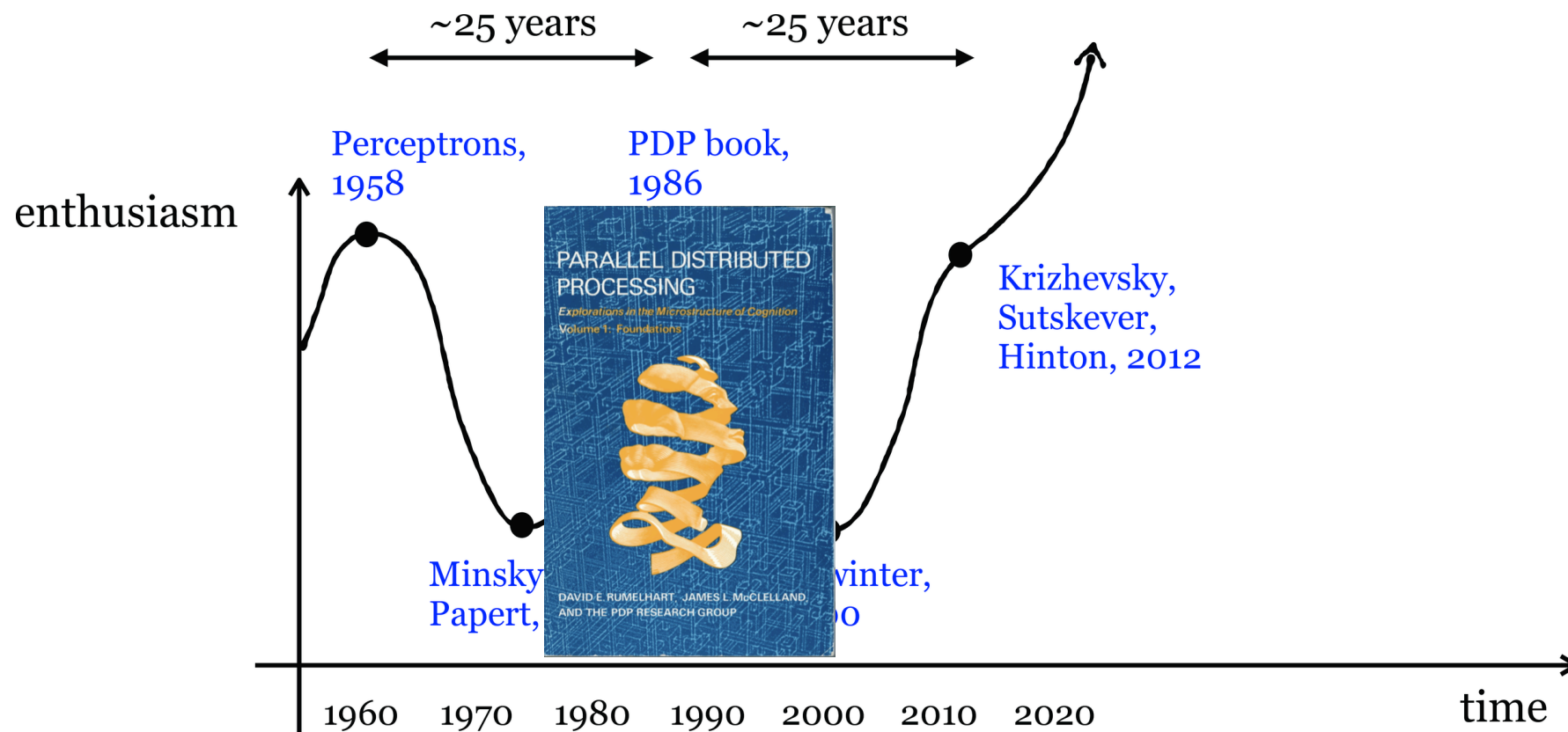
What fixed transformation ϕ can make this decision boundary in x space?

Outline

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

👉 heads-up:

all neural network diagrams focus on a single data point

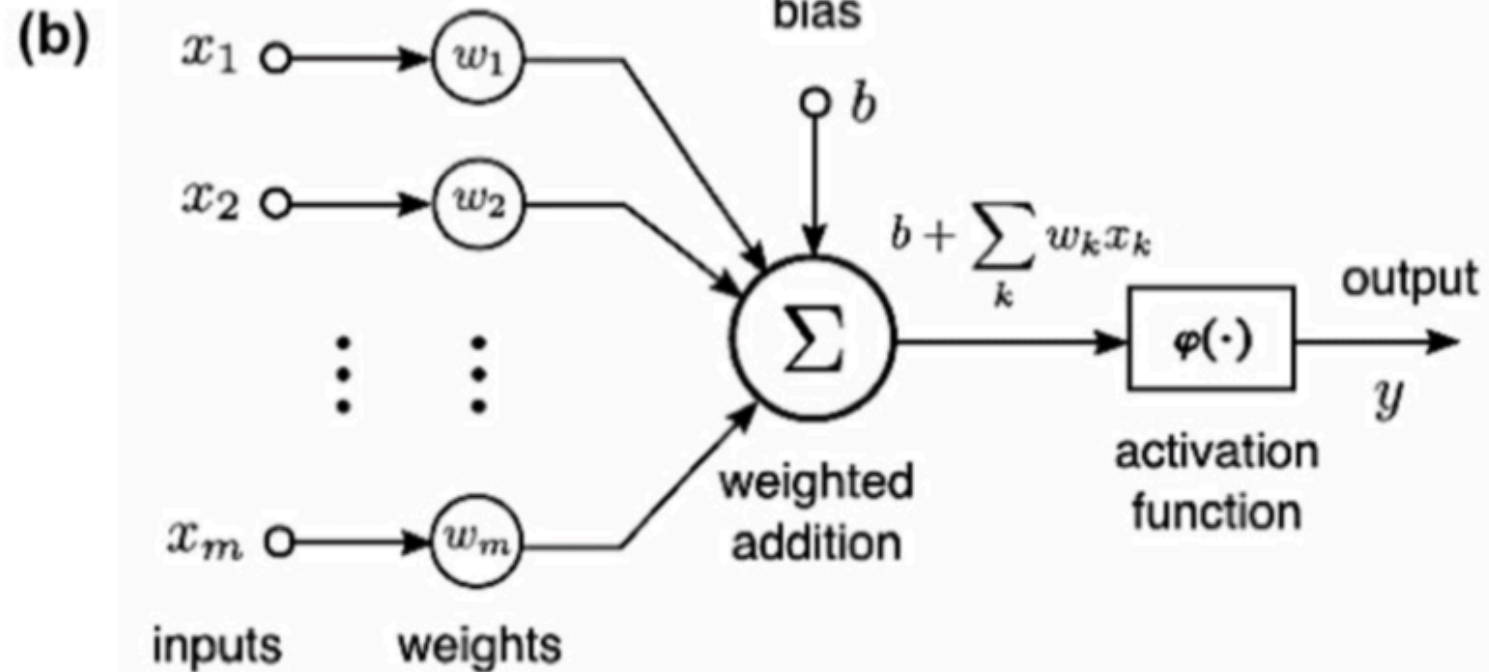
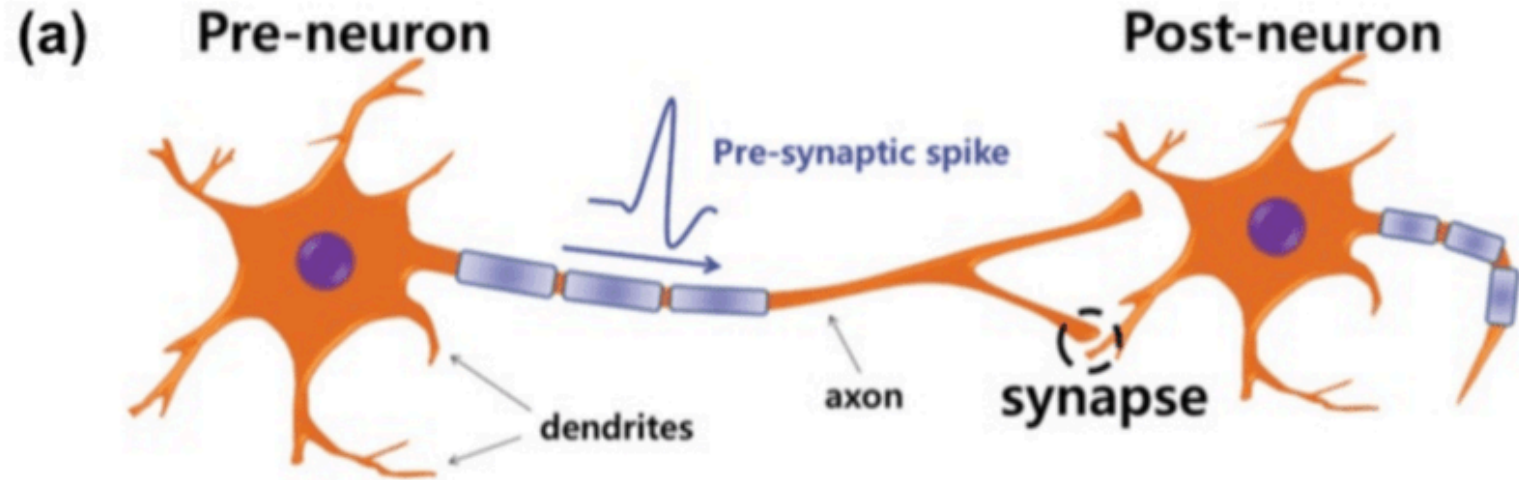


Outlined the fundamental concepts of neural networks:

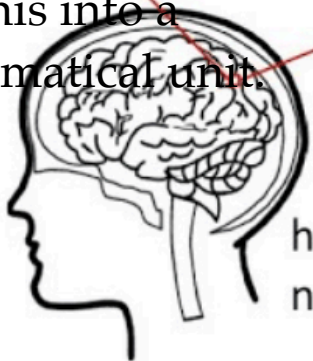
- Nonlinear feature transformation
 - Composing simple nonlinearities amplifies this effect
 - Backpropagation
- } expressiveness
- efficient learning

layered
structure {

Neurons and the brain

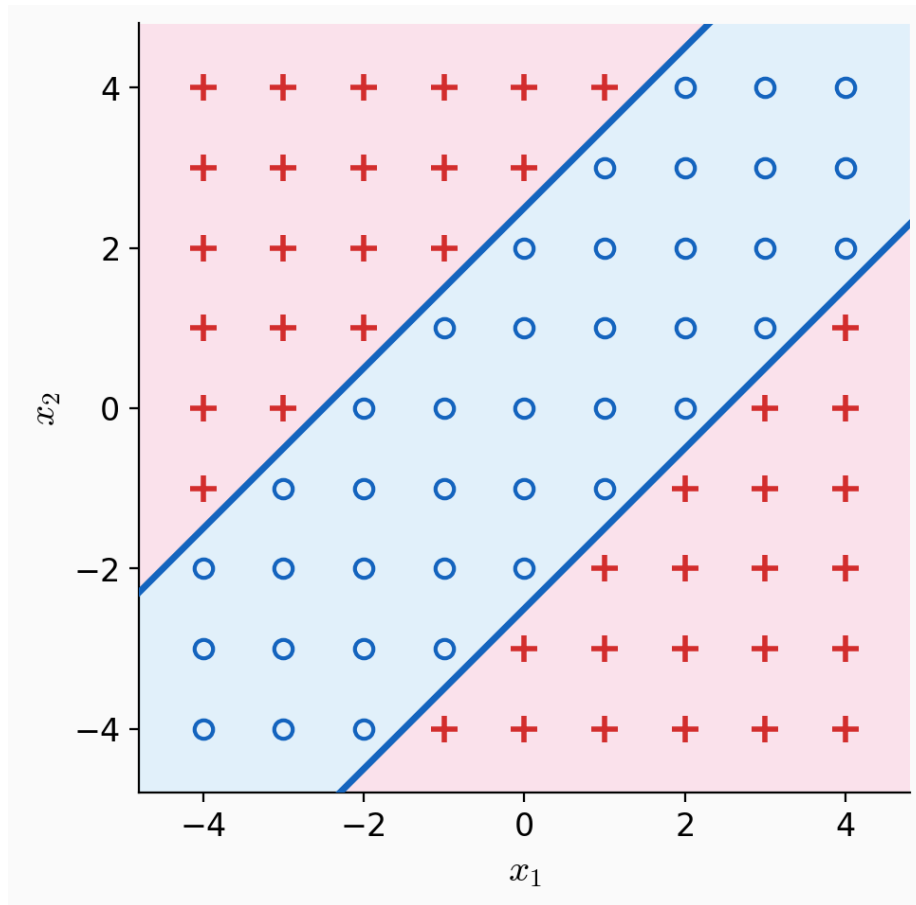


We abstract this into a simple mathematical unit.

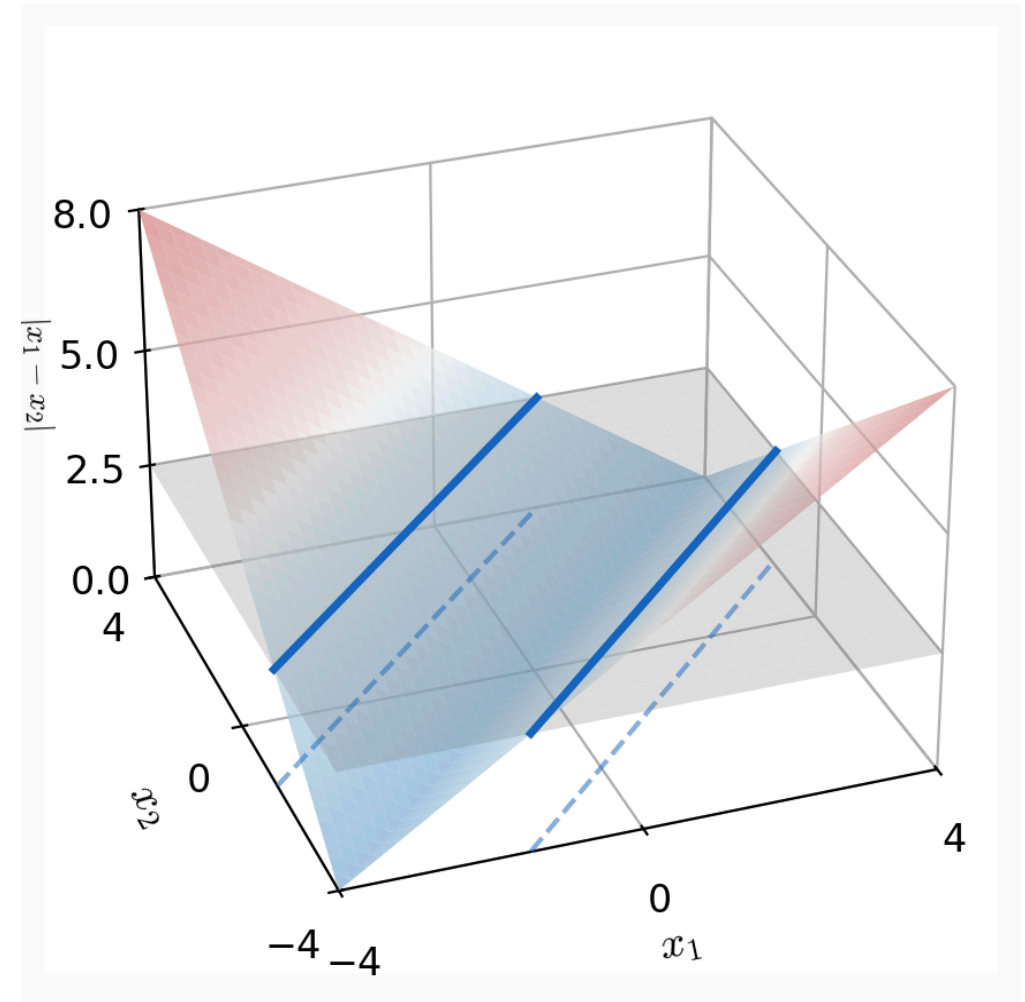


- Nonlinear feature transformation

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

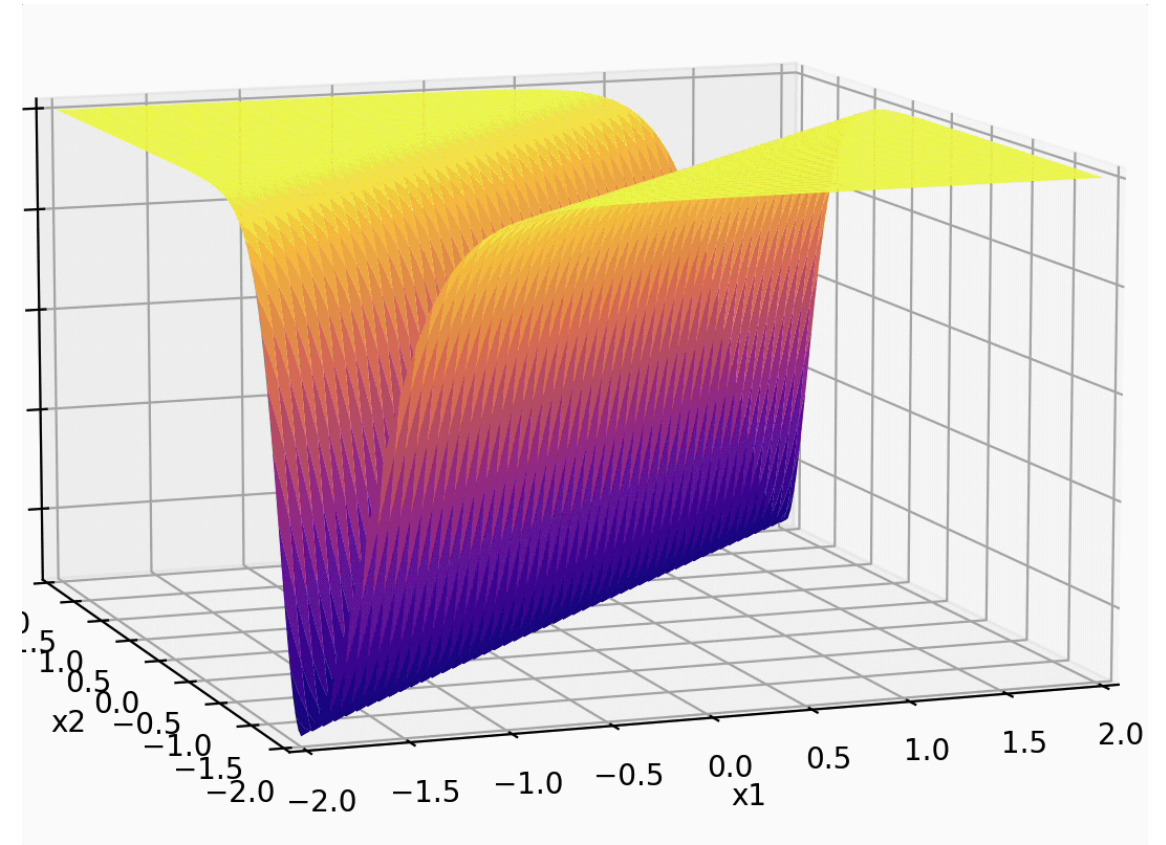
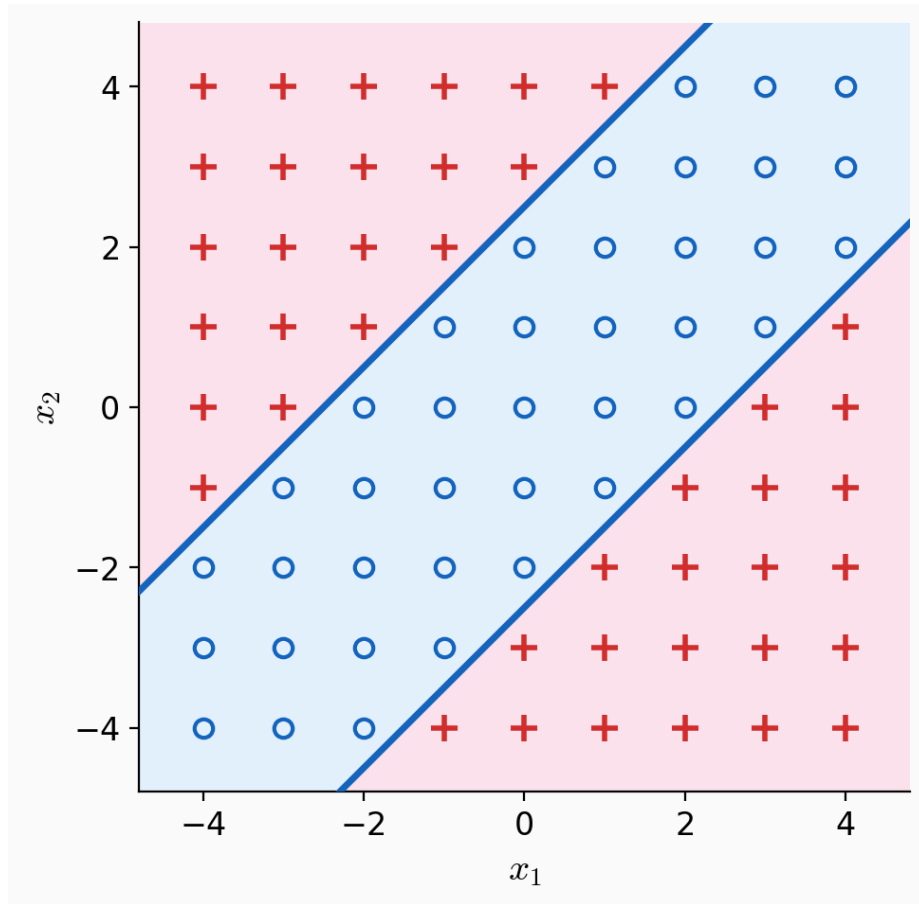


$$\phi([x_1; x_2]) = [|x_1 - x_2|] = 2.5$$



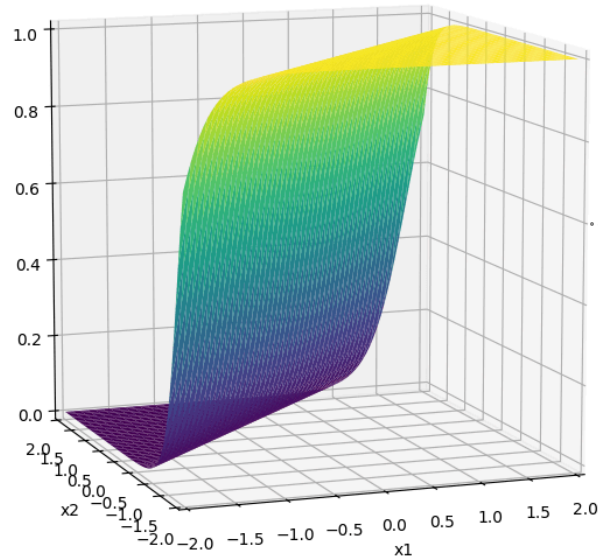
importantly, linear in ϕ , nonlinear in x

- "Composing" simple transformations

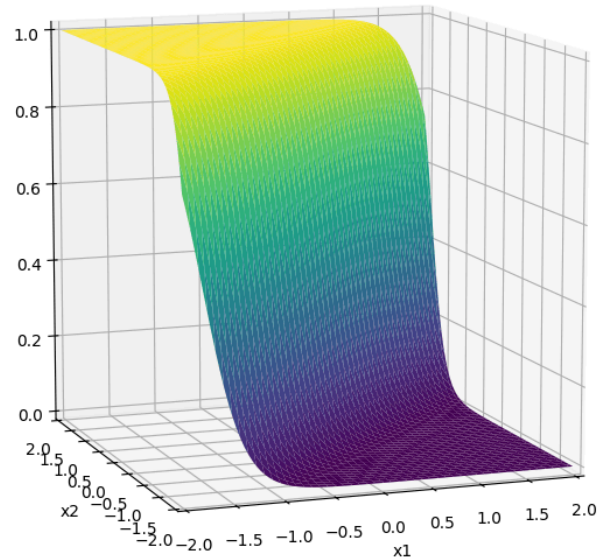


$$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$$

- "Composing" simple transformations

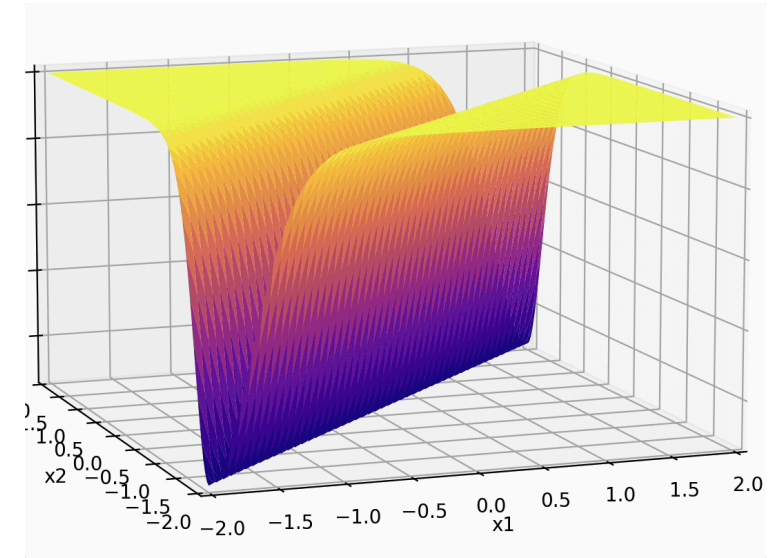


$$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$$



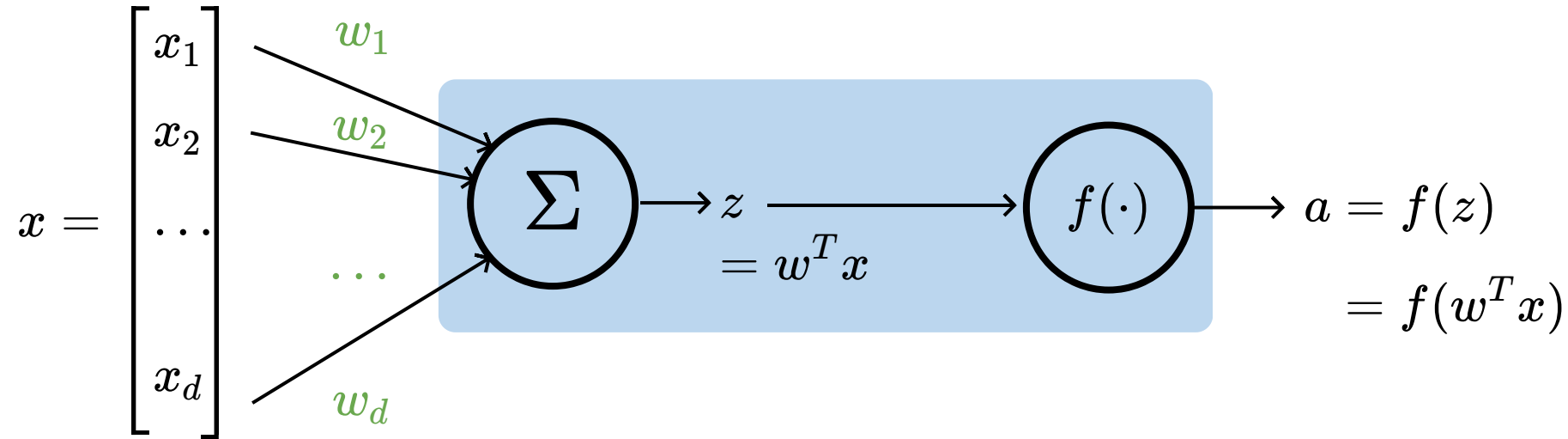
some appropriately
weighted sum

+ **=**



<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

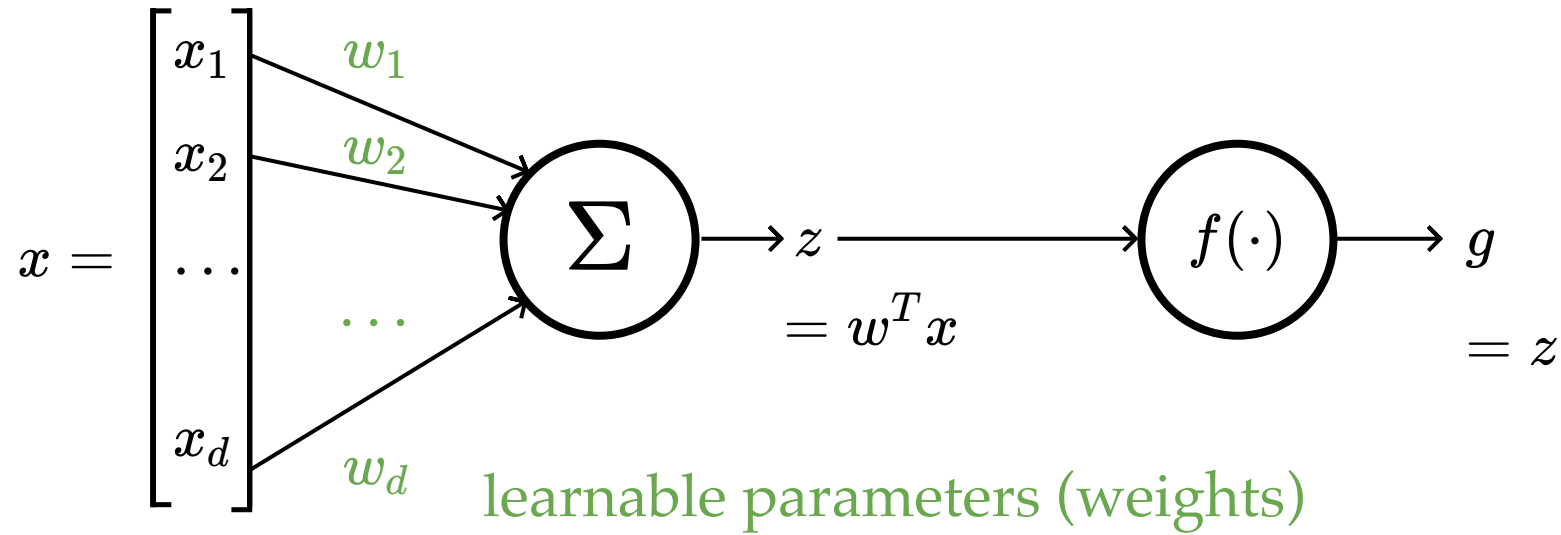
↓

f : what we engineers choose

↓

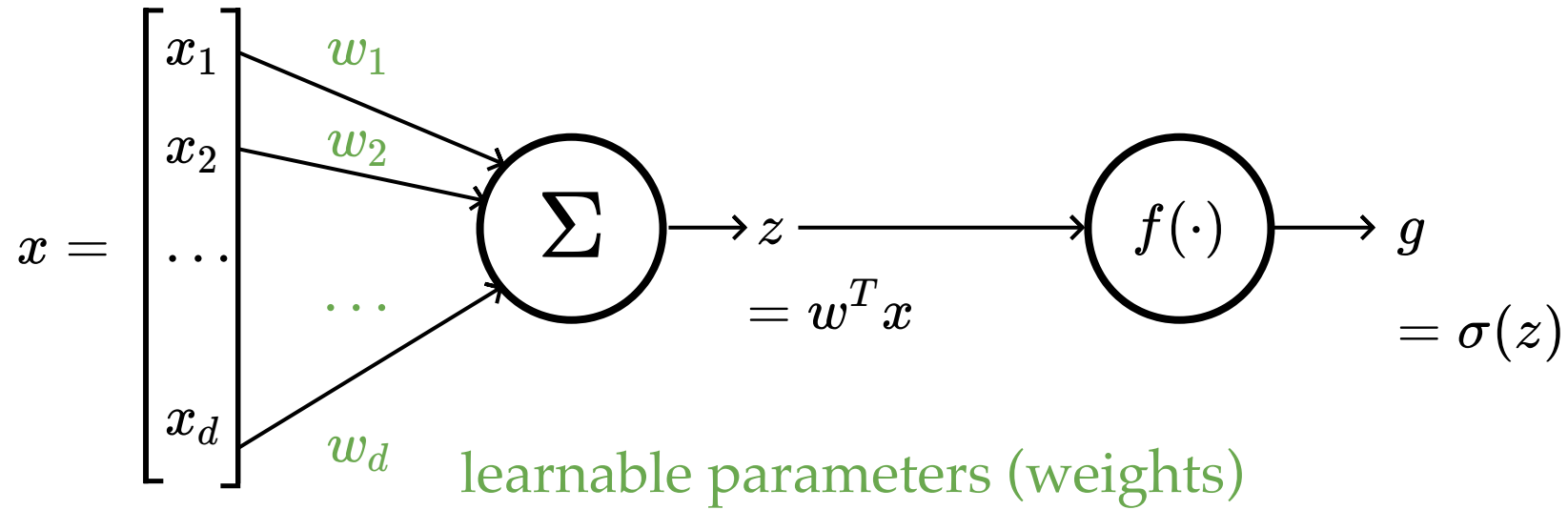
a : scalar

e.g. linear regressor represented as a computation graph



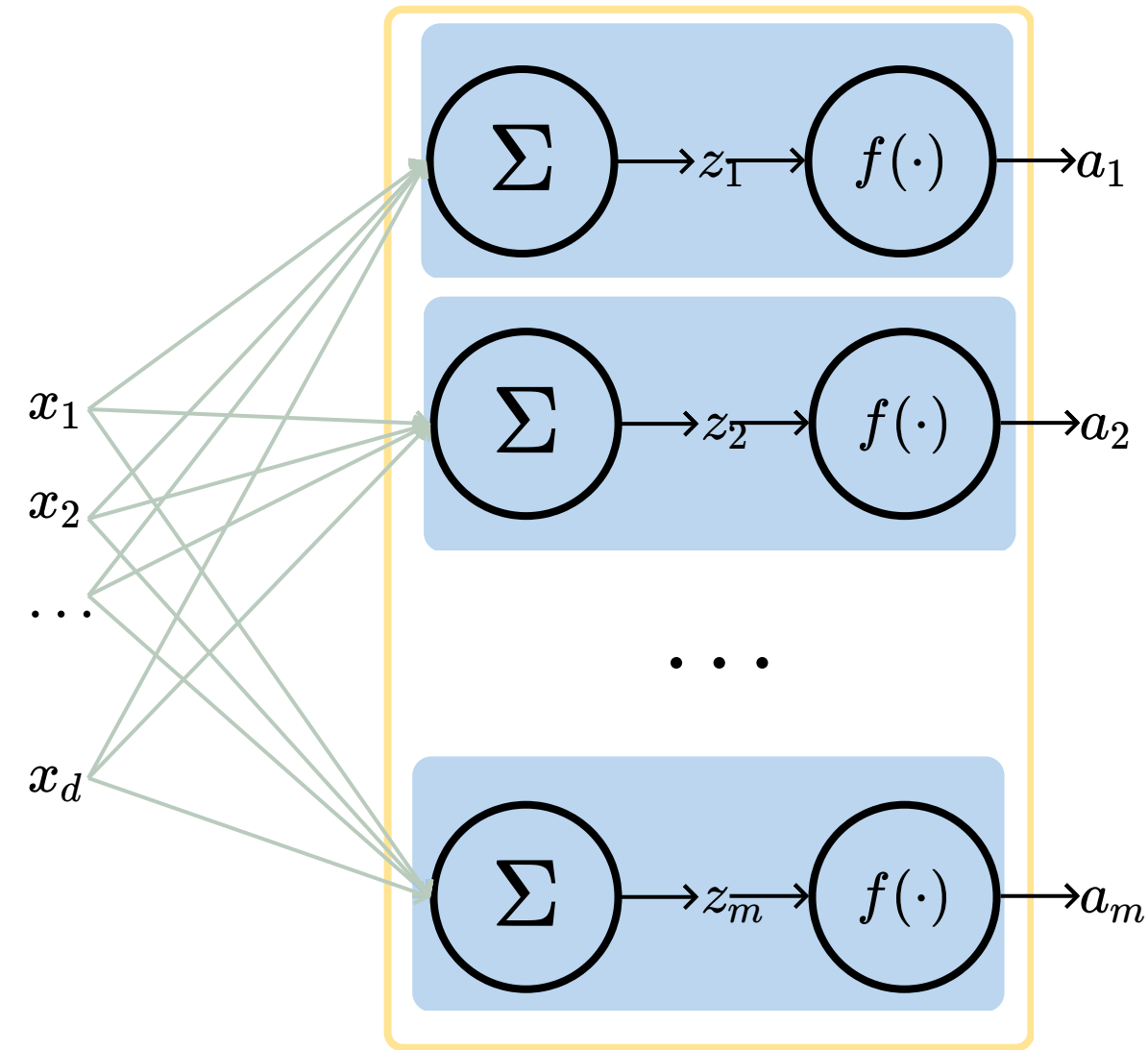
Choose activation $f(z) = z$

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



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

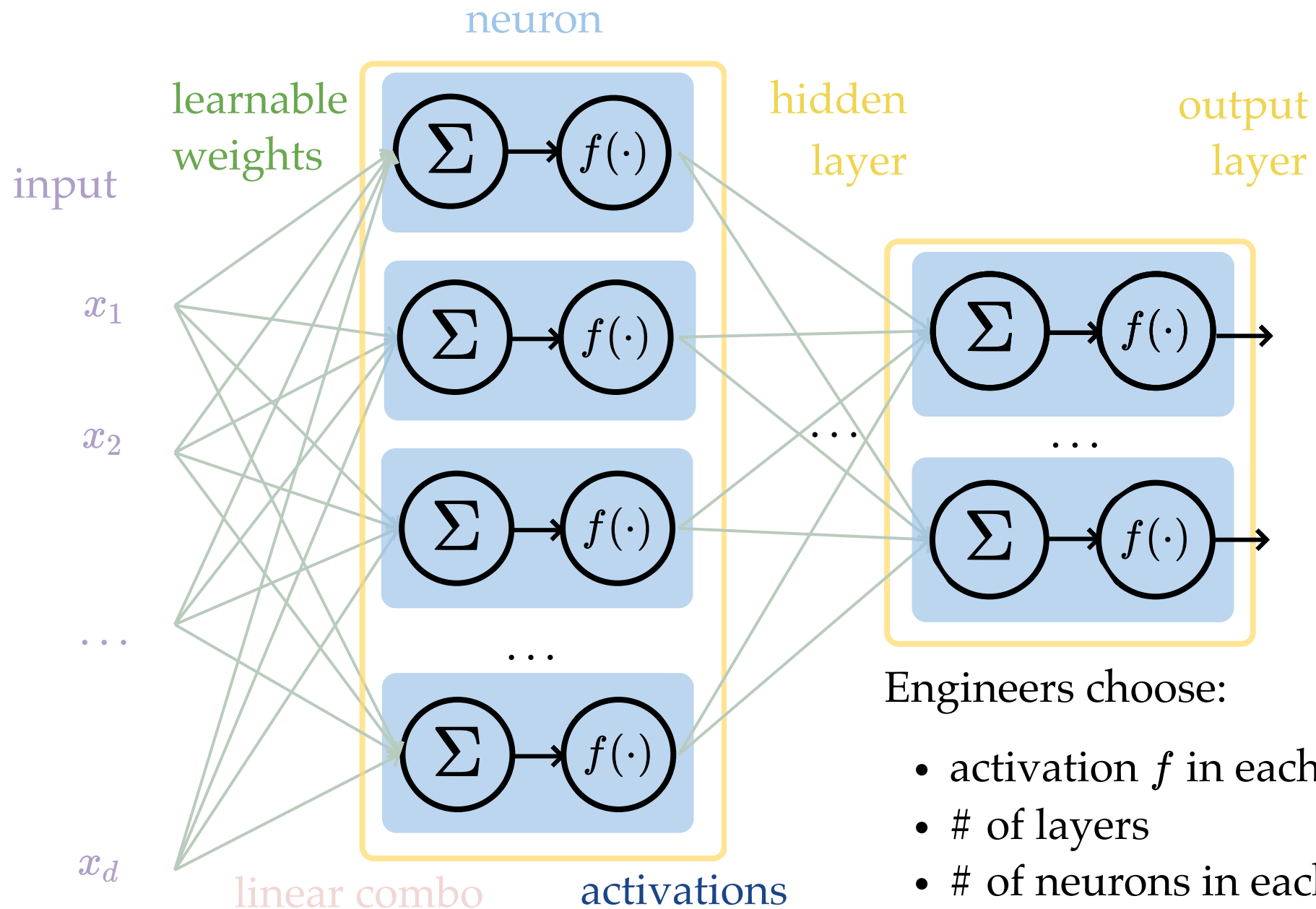
A layer:



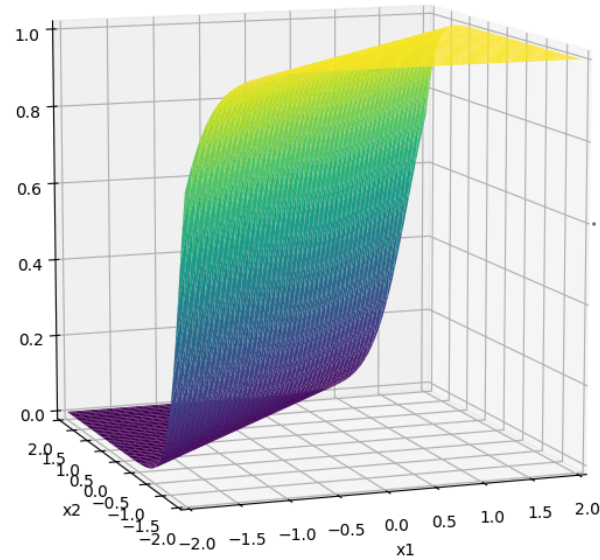
learnable weights

- (# of neurons) = layer's output dimension.
- activations apply element-wise, so z_1 doesn't affect a_2 (softmax in the output layer is the 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:

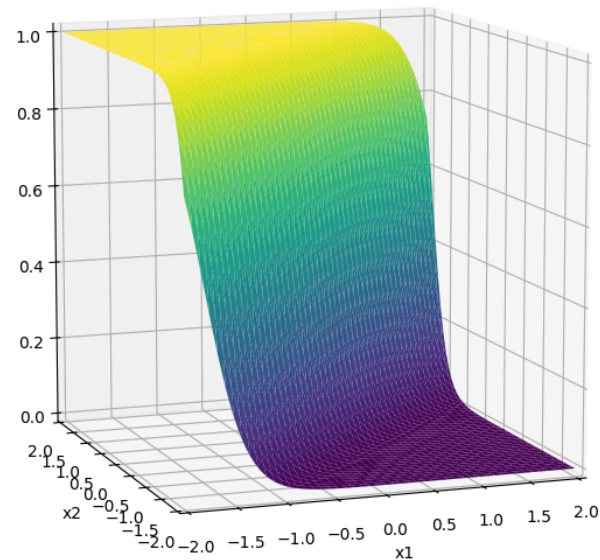


$$\sigma_1 = \sigma(5x_1 - 5x_2 + 1)$$



recall this example

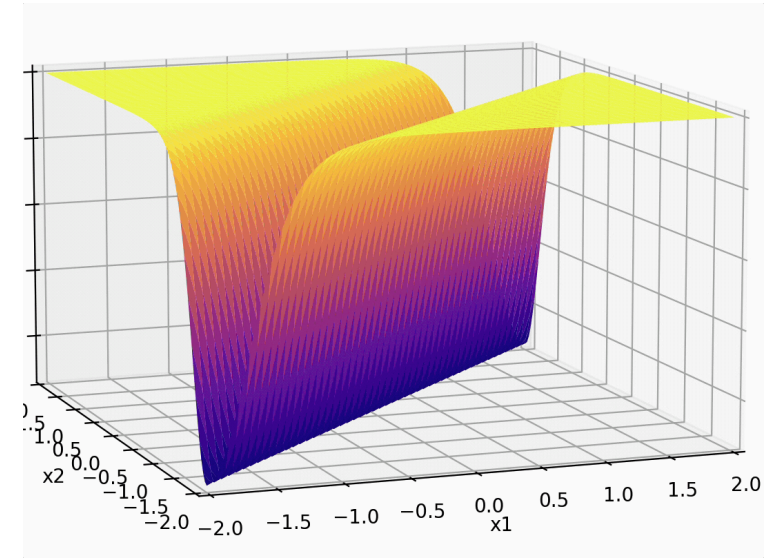
$$\sigma_2 = \sigma(-5x_1 + 5x_2 + 1)$$



some appropriately
weighted sum

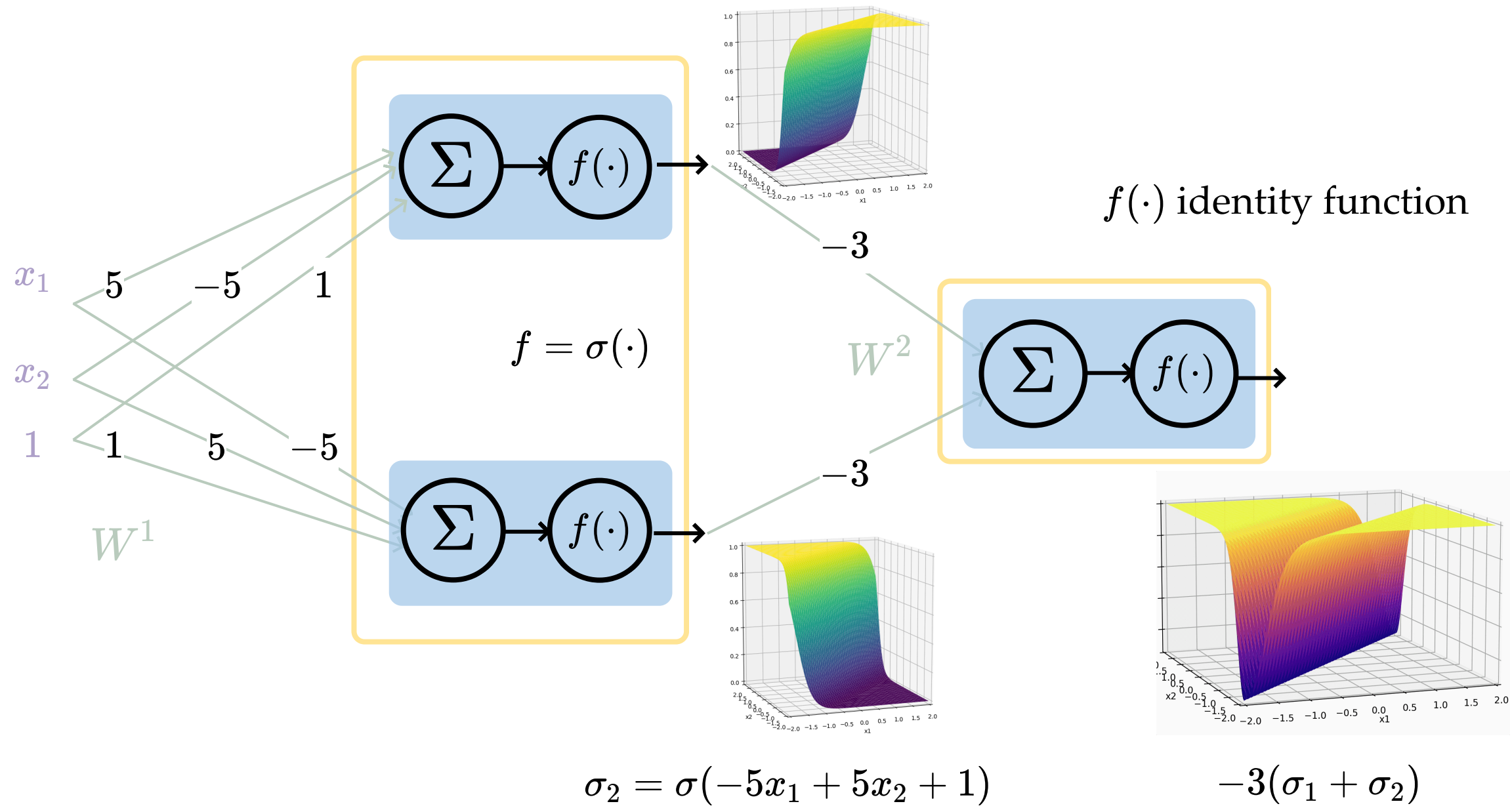


=

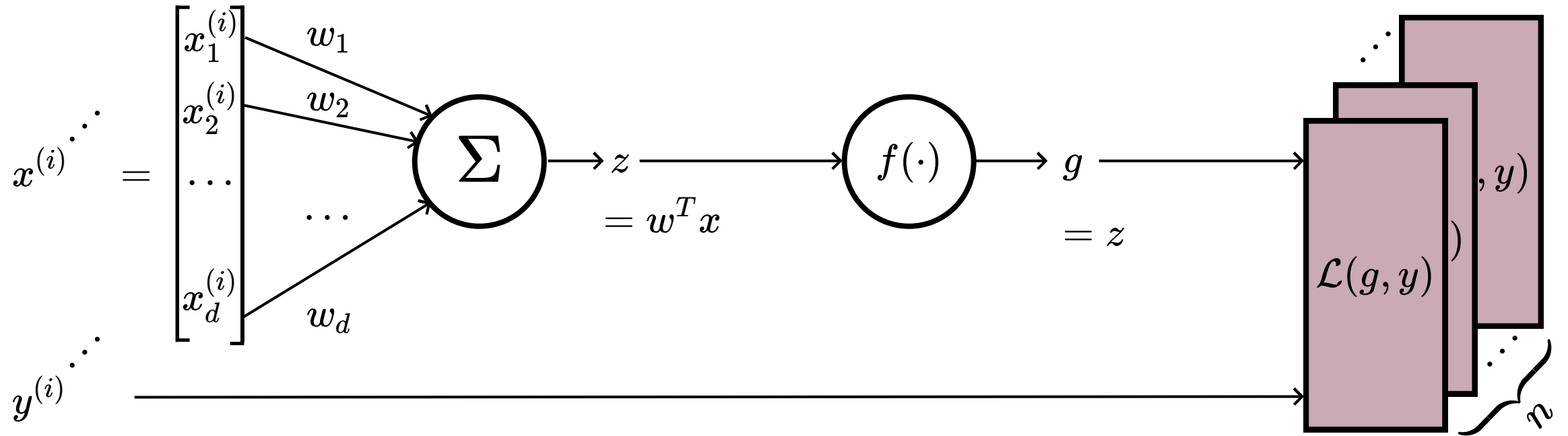


can be represented as:

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

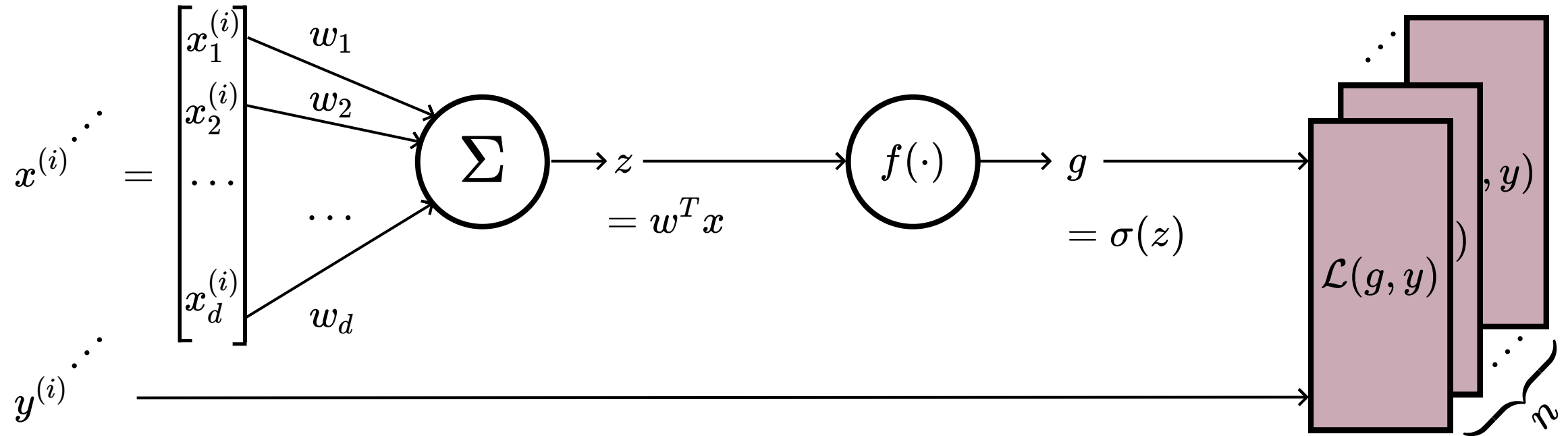


e.g. forward pass of a linear regressor



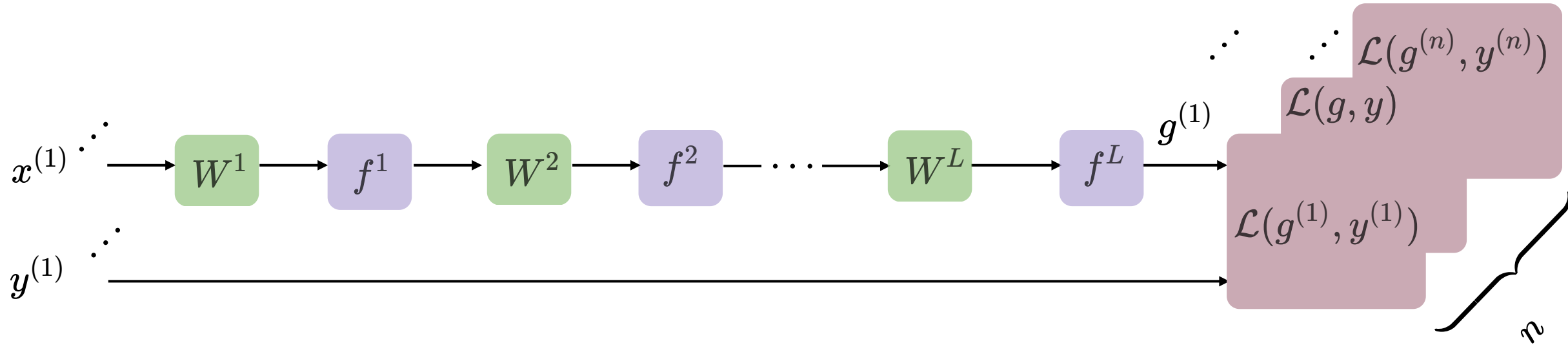
- Activation f is chosen as the identity function
- Evaluate the loss $\mathcal{L} = (g^{(i)} - y^{(i)})^2$
- Repeat for each data point, average the n individual losses

e.g. forward pass of a linear logistic classifier



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

Forward pass: evaluate, *given* the current parameters



- the model outputs $g^{(i)} = f^L \left(\dots f^2 \left(f^1(\mathbf{x}^{(i)}; \mathbf{W}^1); \mathbf{W}^2 \right) ; \dots \mathbf{W}^L \right)$
- 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

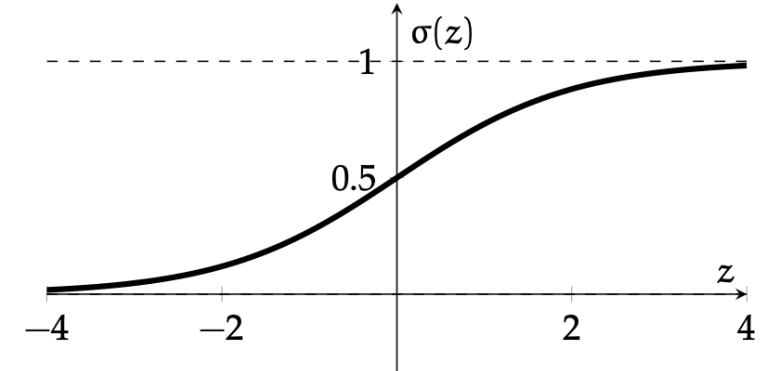
- Systematic feature transformations
- Neural networks
 - Terminology
 - Design choices

Choosing the hidden-layer activation f

σ used to be the most popular

firing rate of a neuron

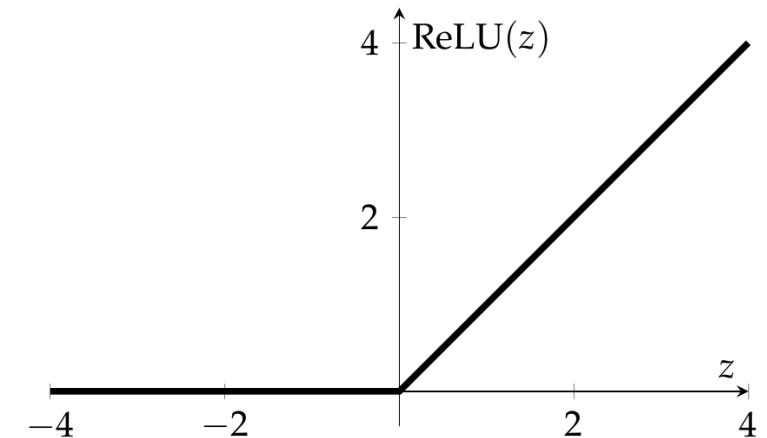
elegant gradient $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$



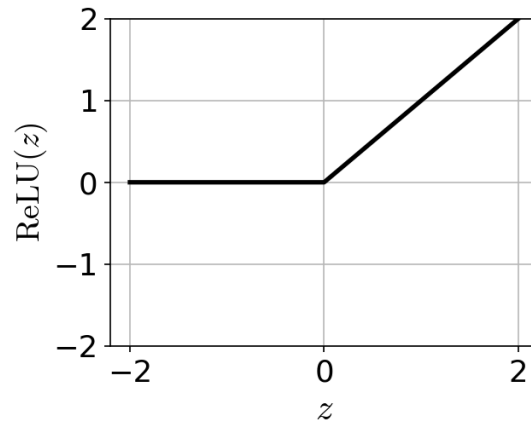
nowadays, the default choice:

$$\text{ReLU}(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases} = \max(0, z)$$

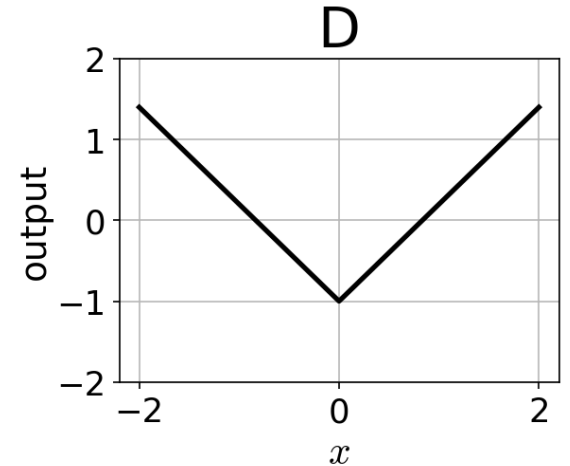
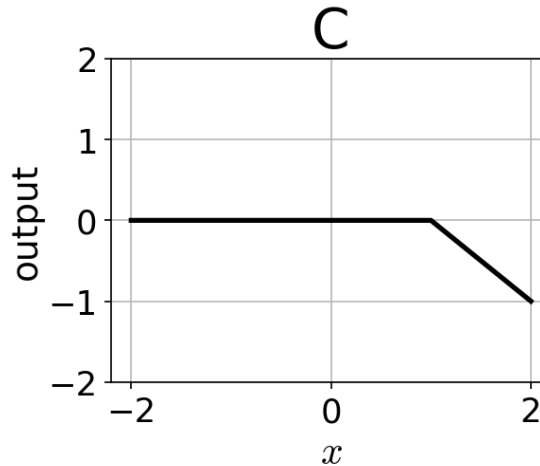
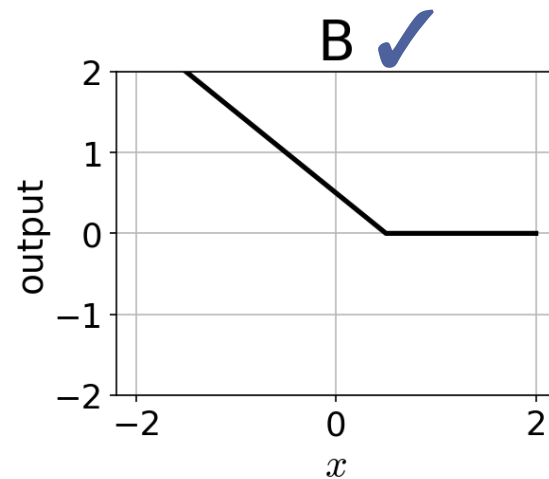
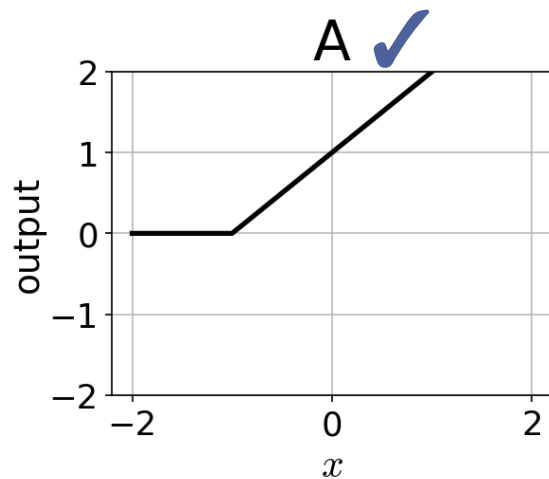
very simple form, and so is its gradient.



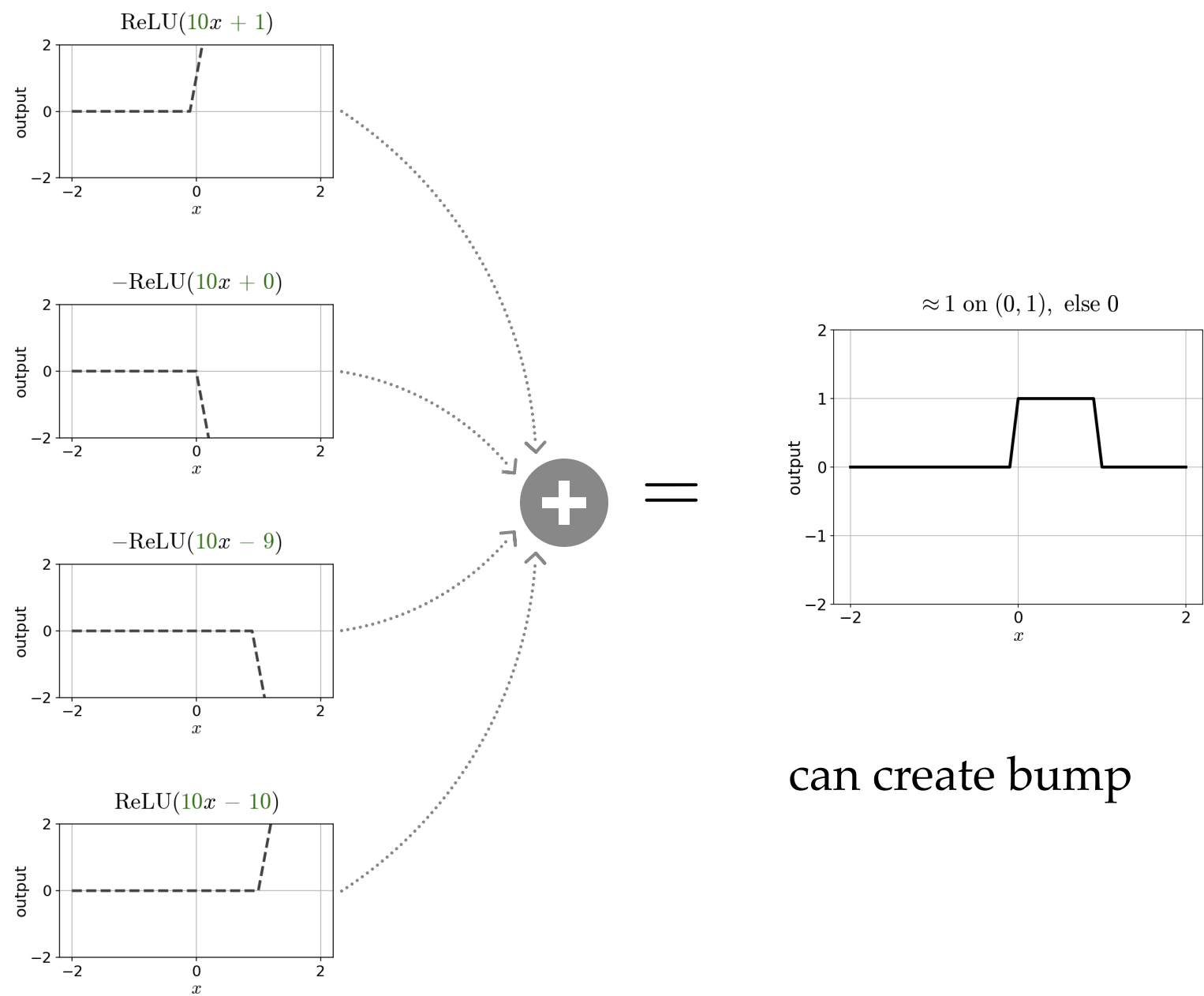
Plotted against z , $\text{ReLU}(z)$ looks like this:



With learnable w and w_0 , which of these can $\text{ReLU}(w x + w_0)$ draw?

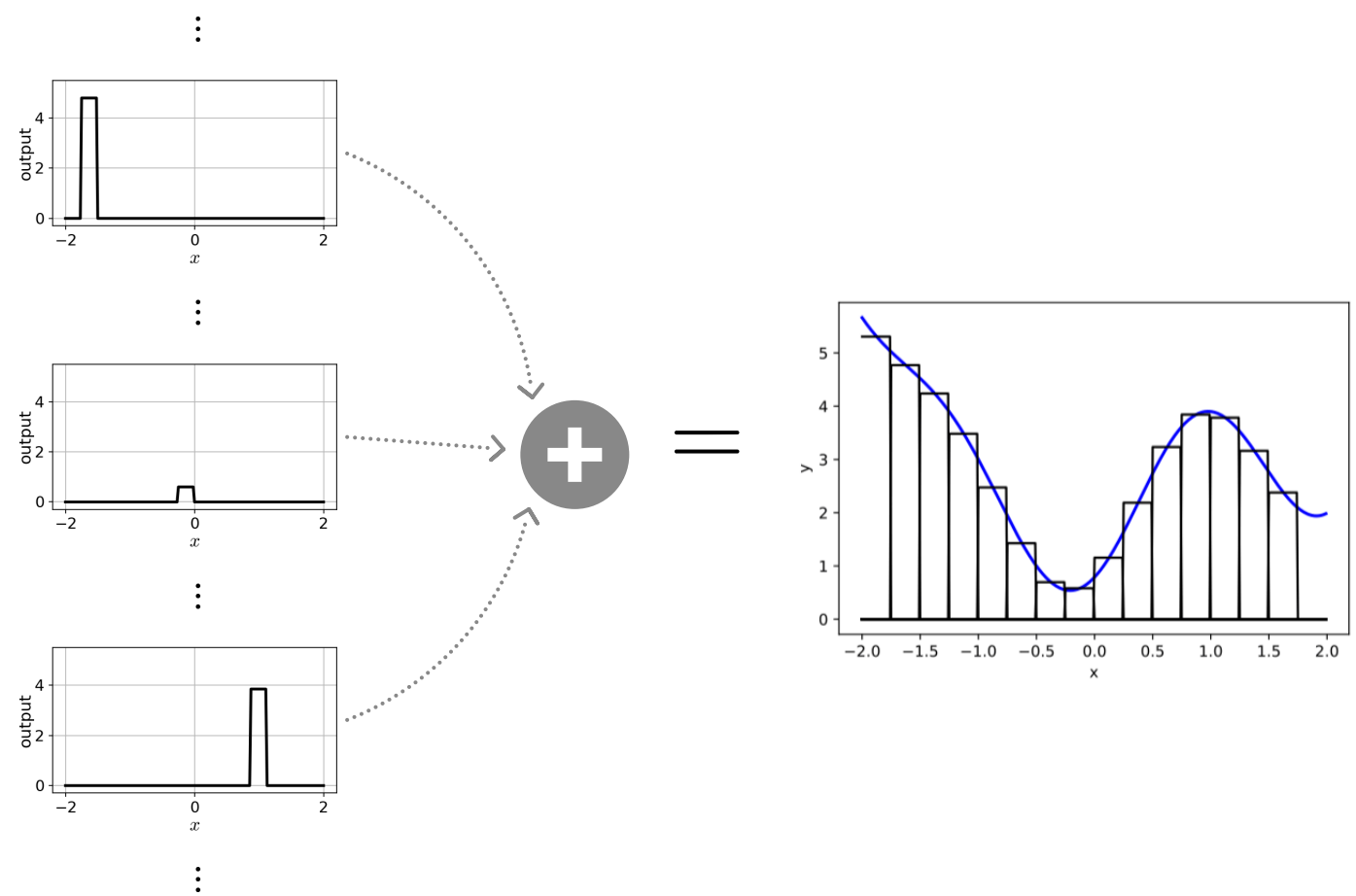


compositions of ReLU(s) can be quite expressive, e.g.,



can create bump

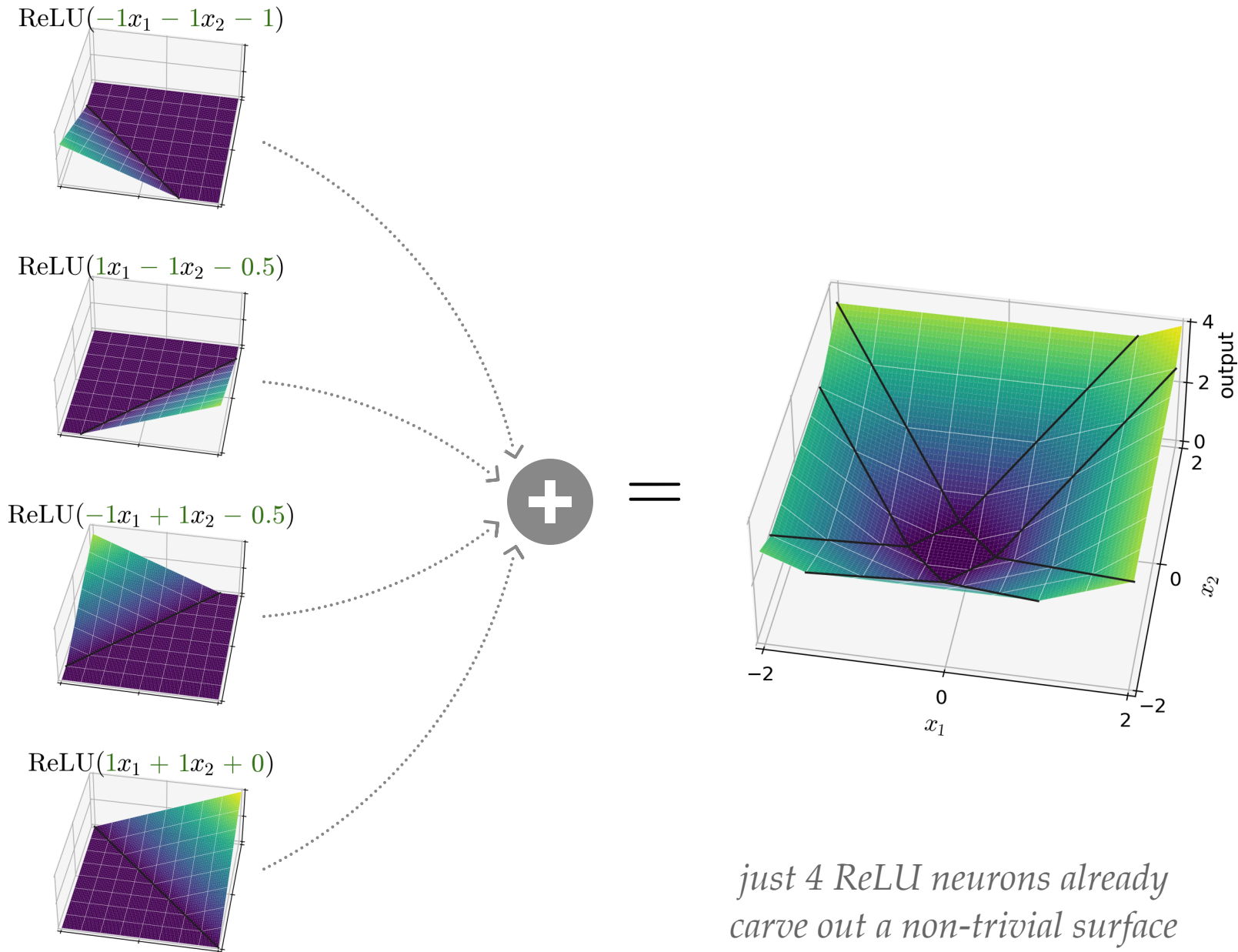
with enough of these bumps appropriately scaled and shifted:



asymptotically, can approximate any continuous function arbitrarily well (for regression), therefore can also approximate arbitrary decision boundaries (for classification)

asymptotically: as the number of neurons grows. With enough neurons, even one hidden layer gets within any error of any continuous function on a bounded domain.

the same holds for higher dimensions



<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 nonlinear activation) makes the model more expressive, but it also increases the risk of overfitting.

For neural networks, though, how size relates to overfitting is still open research; see the double-descent curve and scaling laws.

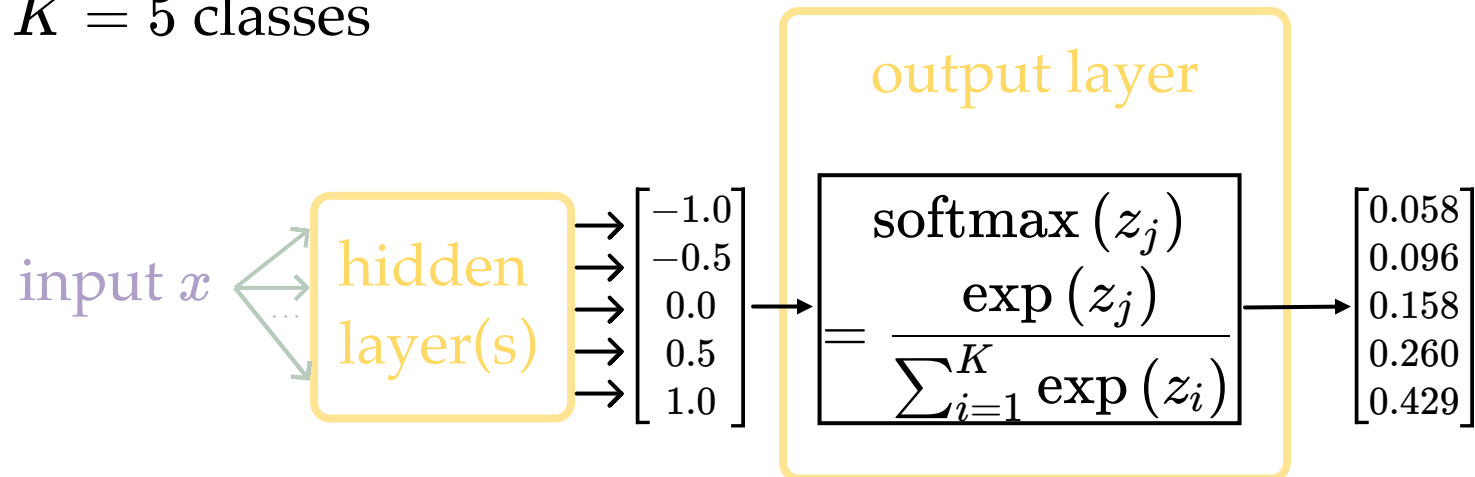
(The demo doesn't show in the PDF; the link below works.)

<https://shenshen.mit.edu/demos/nn/playground/?embed>

output layer design choices

- # neurons, activation, and loss depend on the high-level goal.
- Typically straightforward.
- Multi-class setup: to predict *exactly one* of K classes, the output layer has K neurons, softmax activation, and cross-entropy loss

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



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

Summary

feature transformations feature transformation polynomial features feature engineering

neural networks neuron activation function layer hidden layer forward pass

design choices ReLU width depth

- Linear models are convenient but lack expressiveness for most real-world tasks.
- A **fixed** nonlinear 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 nonlinear 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)