

Machine Learning 2 - Features & Neural Networks

A regression example

A regression example

Training data:

x	y	$\text{Loss}(x, y, w) = (\mathbf{w} \cdot \phi(x) - y)^2$
[1, 0]	2	$(w_1 - 2)^2$
[1, 0]	4	$(w_1 - 4)^2$
[0, 1]	-1	$(w_2 - (-1))^2$

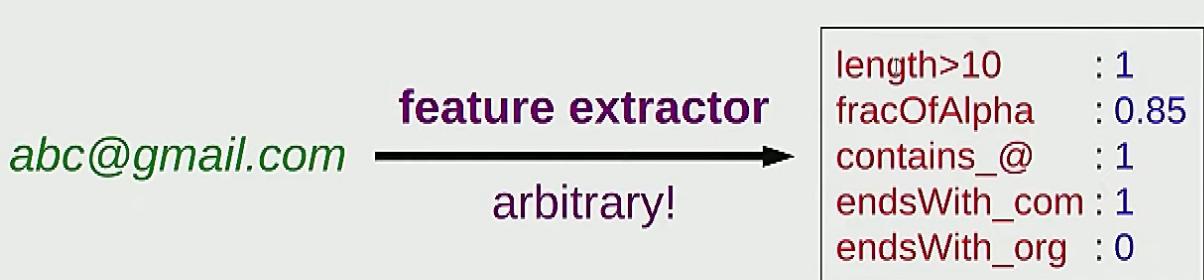
$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{3} ((w_1 - 2)^2 + (w_1 - 4)^2 + (w_2 - (-1))^2)$$

Features

Feature extraction

Organization of features

Task: predict whether a string is an email address



Which features to include? Need an organizational principle...

A feature extractor takes an input and output a set of properties which are useful for prediction

Feature templates

Feature vector representations

```
fracOfAlpha : 0.85
contains_a : 0
...
contains_@ : 1
...
```

Array representation (good for dense features):

[0.85, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

Map representation (good for sparse features):

{"frac0fAlpha": 0.85, "contains_@": 1}

Feature templates

the phi - bottle neck

Hypothesis class

Hypothesis class

Predictor:

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) \quad \text{or} \quad \text{sign}(\mathbf{w} \cdot \phi(\mathbf{x}))$$



Definition: hypothesis class

A **hypothesis class** is the set of possible predictors with a fixed $\phi(\mathbf{x})$ and varying \mathbf{w} :

$$\mathcal{F} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^d\}$$

All possible predictors that we gonna get (With this set of features)

Think of set of functions you can get, by that set of features

Examples

Example: beyond linear functions

Regression: $x \in \mathbb{R}, y \in \mathbb{R}$

Linear functions:

$$\phi(x) = x$$

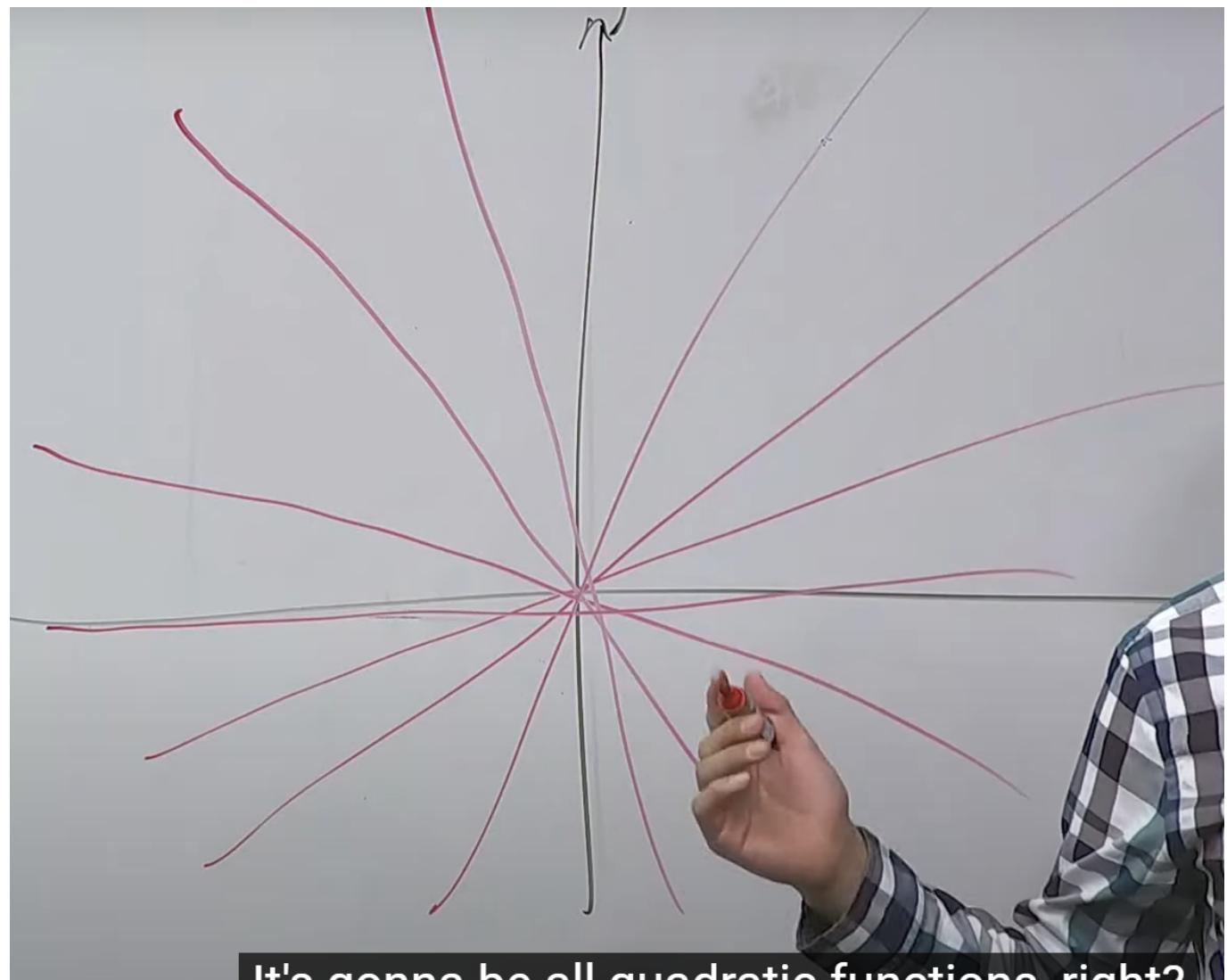
$$\mathcal{F}_1 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

Quadratic functions:

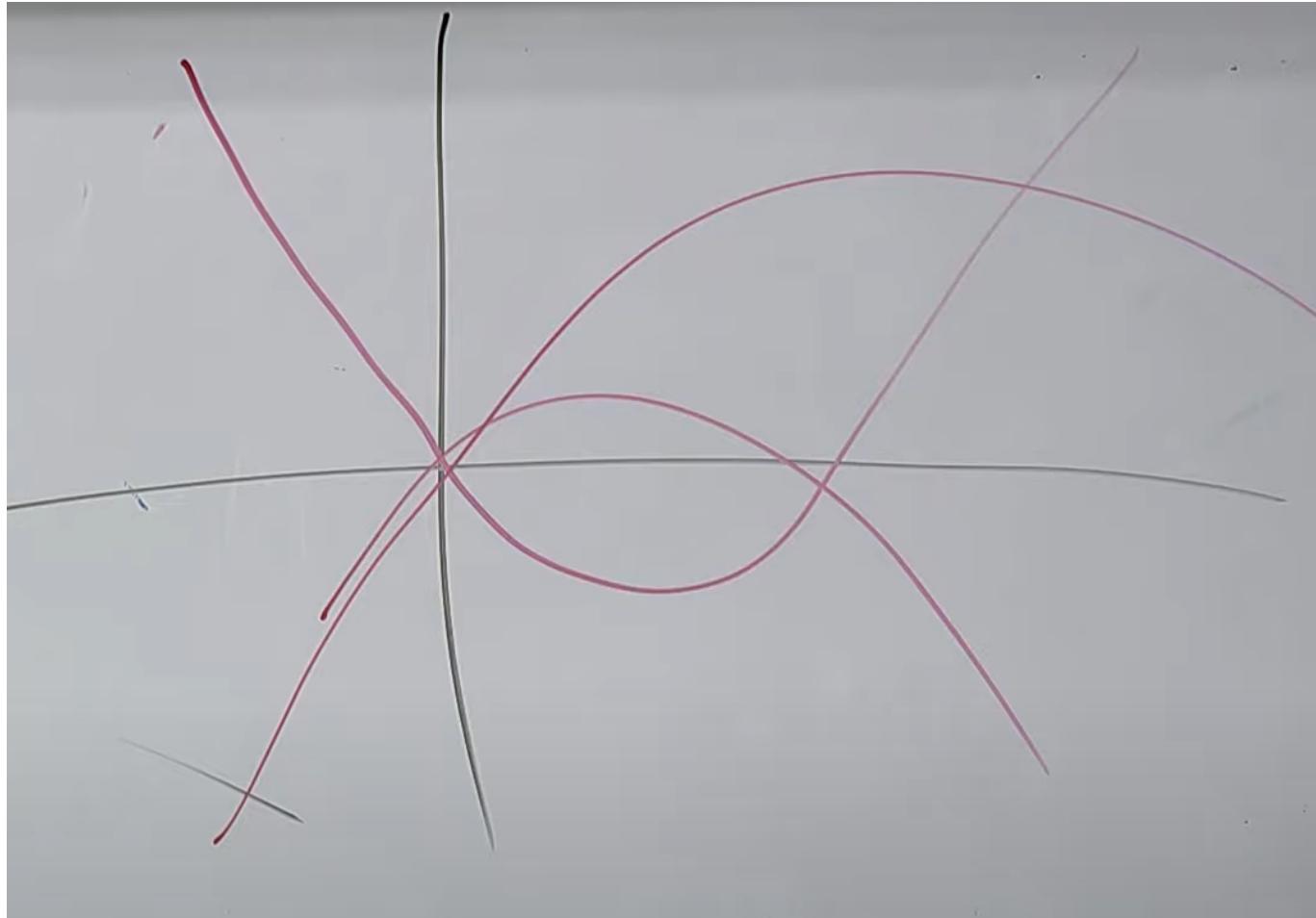
$$\phi(x) = [x, x^2]$$

$$\mathcal{F}_2 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

[whiteboard]

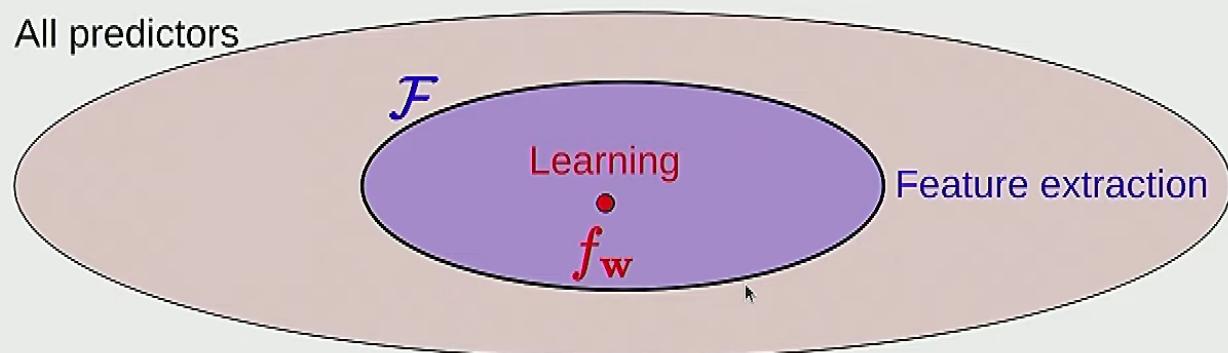


It's gonna be all quadratic functions, right?



Full pipeline of Machine learning

Feature extraction + learning



- Feature extraction: set \mathcal{F} based on domain knowledge
- Learning: set $f_w \in \mathcal{F}$ based on data
- The hypothesis class is the set of all predictors

- When you go and define a feature map, that is gonna carve out a much smaller set in the hypothesis class
- What is learning doing is choosing a particular function of that family (two bigger sets), based on the data

This means:

- You first define a set of functions that u interested in
- And now you say, okay based on data, let me go and search through that, and find the one is best for me

An example task

An example task



Example: detecting responses

Input x :

two consecutive messages in a chat

Output $y \in \{+1, -1\}$:

whether the second message is a response to the first

Recall: feature extractor ϕ should pick out properties of x that might be useful for prediction of y

Feature template:



Question

What feature templates would you use for predicting whether the second message is a response to the first?

time elapsed

time elapsed is between ____ and ____

first message contains ____

second message contains ____

two messages both contain ____

two messages have ____ common words

The **time elapsed** is a single number, it can be really small or large, which means, if it is really large, the contribution to the score gonna be really large, it going to affect the model too much, which you might not want to see. (too sensitive)

In this case, you can split it into pieces/intervals eg **time elapsed between ... and ...**

Neural Networks

Motivation



Example: predicting car collision

Input: position of two oncoming cars $x = [x_1, x_2]$

Output: whether safe ($y = +1$) or collide ($y = -1$)

True function: safe if cars sufficiently far

$$y = \text{sign}(|x_1 - x_2| - 1)$$

Examples:

x	y
[1, 3]	+1
[3, 1]	+1
[1, 0.5]	-1

Stanford

The true function: Check if the distance of two cars are larger than 1

Decomposing the problem

Test if car 1 is far right of car 2:

$$h_1 = \mathbf{1}[x_1 - x_2 \geq 1]$$

Test if car 2 is far right of car 1:

$$h_2 = \mathbf{1}[x_2 - x_1 \geq 1]$$

Safe if at least one is true:

$$y = \text{sign}(h_1 + h_2)$$

x	h_1	h_2	y
[1, 3]	0	1	+1
[3, 1]	1	0	+1
[1, 0.5]	0	0	-1

Break down the problem

But in the real, we don't know what are these functions Can we kind of learn those functions?

Well now let's make it abstract:

Learning strategy

Define: $\phi(x) = [1, x_1, x_2]$

Intermediate hidden subproblems:

$$h_1 = \mathbf{1}[\mathbf{v}_1 \cdot \phi(x) \geq 0] \quad \mathbf{v}_1 = [-1, +1, -1]$$

$$h_2 = \mathbf{1}[\mathbf{v}_2 \cdot \phi(x) \geq 0] \quad \mathbf{v}_2 = [-1, -1, +1]$$

Final prediction:

$$f_{\mathbf{V}, \mathbf{w}}(x) = \text{sign}(\mathbf{w}_1 h_1 + \mathbf{w}_2 h_2) \quad \mathbf{w} = [1, 1]$$



Key idea: joint learning

Goal: learn both hidden subproblems $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2)$ and combination weights $\mathbf{w} = [w_1, w_2]$

Stanford

$$h_1 = -1 + x_1 - x_2 \geq 0$$

=>

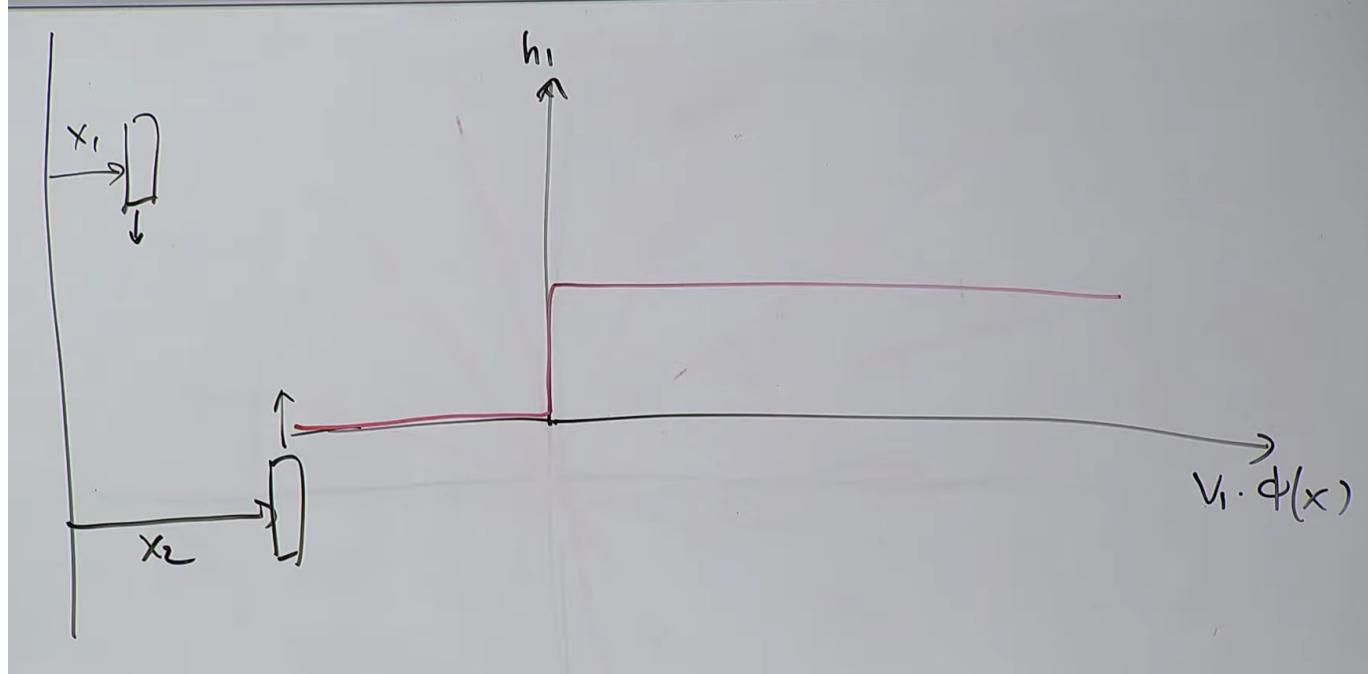
$$x_1 - x_2 \geq 1$$

That are the same, just another forms of representation

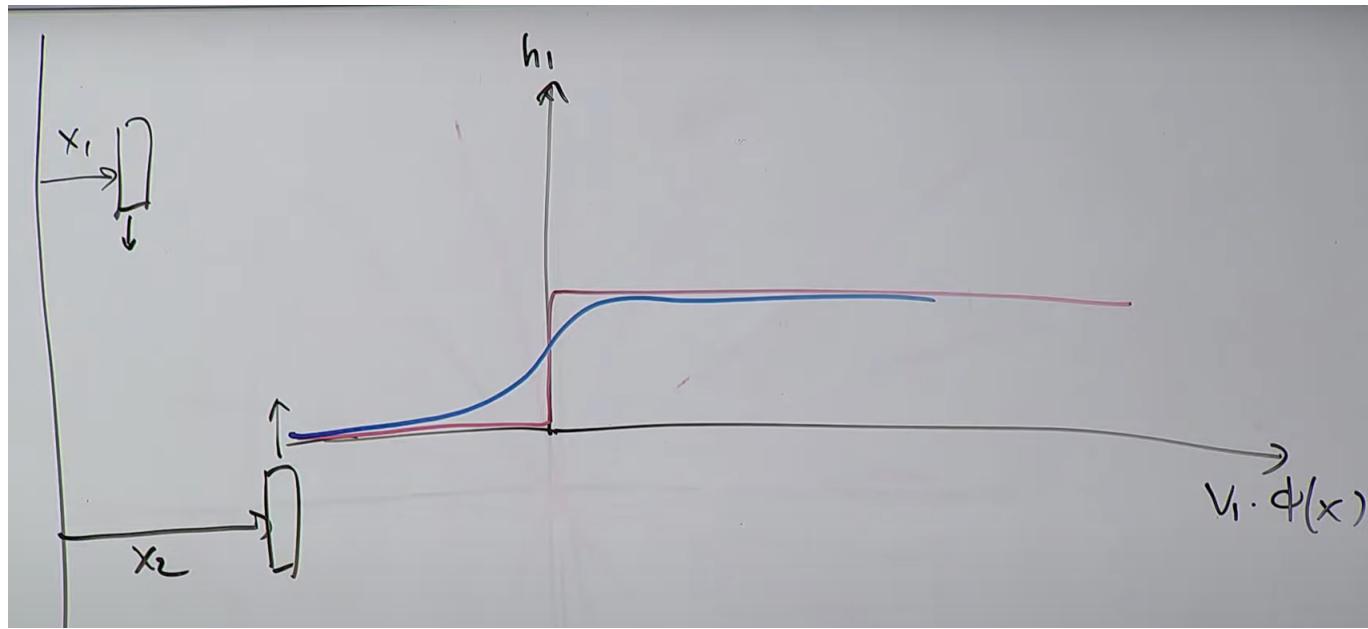
In the neural networks, we will leave v_1, v_2, w as unknown, and try to fit them through training

How do we learn it

There is a problem, h_1 with respect to $v_1 \cdot \phi(x)$ is 0, and we don't like it, cuz SGD won't work



So we can find a function smooth it out with logistic fuction



Gradients

Problem: gradient of h_1 with respect to \mathbf{v}_1 is 0

$$h_1 = \mathbf{1}[\mathbf{v}_1 \cdot \phi(x) \geq 0]$$

[whiteboard]



Definition: logistic function

The logistic function maps $(-\infty, \infty)$ to $[0, 1]$:

$$\sigma(z) = (1 + e^{-z})^{-1}$$

Derivative:

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

Solution:

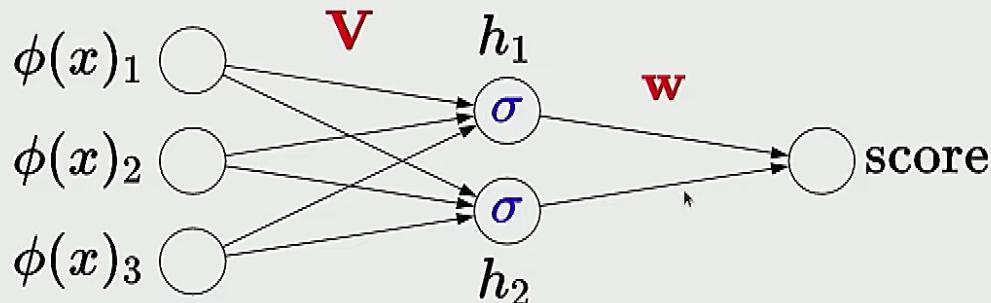
$$h_1 = \sigma(\mathbf{v}_1 \cdot \phi(x))$$

Stanfor

Define nural nets

Neural networks

Neural network (one hidden layer):



Intermediate hidden units:

$$h_j = \sigma(\mathbf{v}_j \cdot \phi(x)) \quad \sigma(z) = (1 + e^{-z})^{-1}$$

Output:

$$\text{score} = \mathbf{w} \cdot \mathbf{h}$$

The neural nets are breaking a problem into subproblems, have the result of intermediate computations, and they become feature of next layer

It's linear classifiers packaged up. The outcome of one set of classifiers become features to the next layer, and so on.

For deeper networks, as you proceed, you can derive more abstract features

Do the neural learning

How calculate the gradient

Approach

Mathematically: just grind through the chain rule

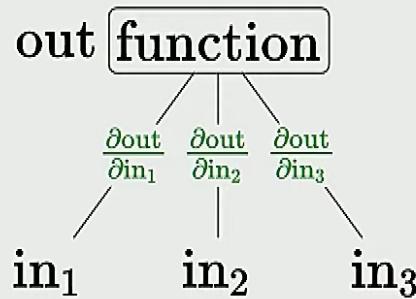
Next: visualize the computation using a **computation graph**

Advantages:

- Avoid long equations
- Reveal structure of computations (modularity, efficiency, dependencies) — TensorFlow/PyTorch are built on this

Computation graph

Functions as boxes



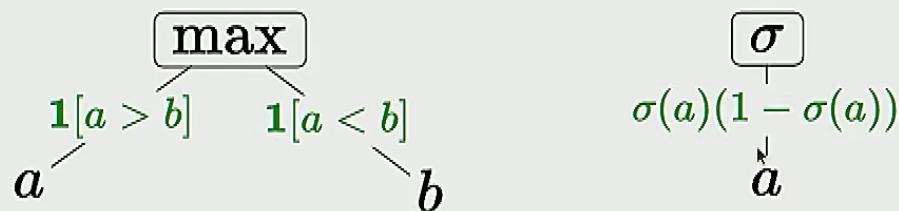
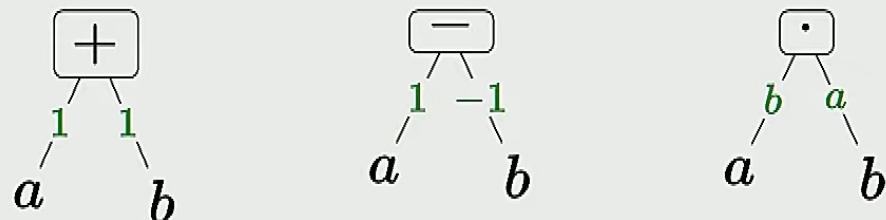
Partial derivatives (gradients): how much does the output change if an input changes?

Example:

$$2(\text{in}_1 + \epsilon) + \text{in}_2 \text{in}_3 = \text{out} + 2\epsilon$$



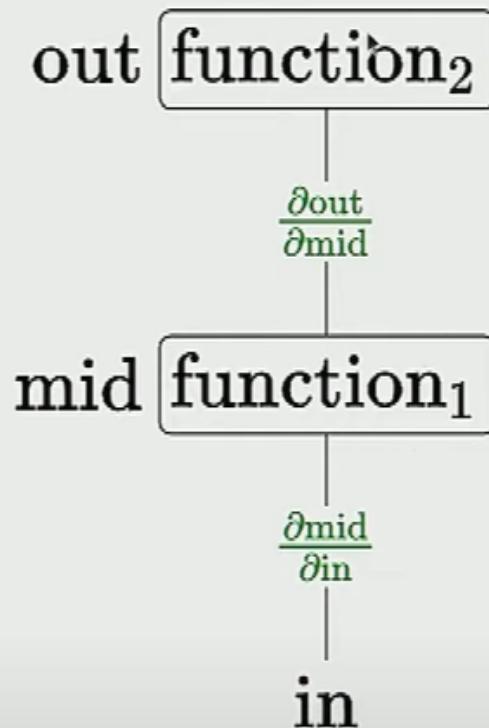
Basic building blocks



Think what if we change the input a little bit

The chain rule

Composing functions



rule:

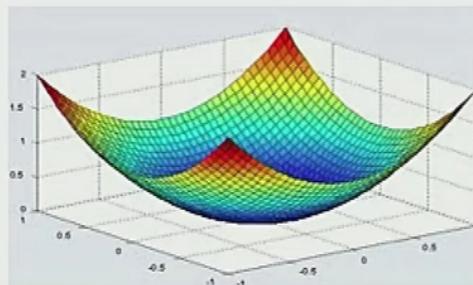
$$\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$$

Note on optimization for Neural nets

Note on optimization

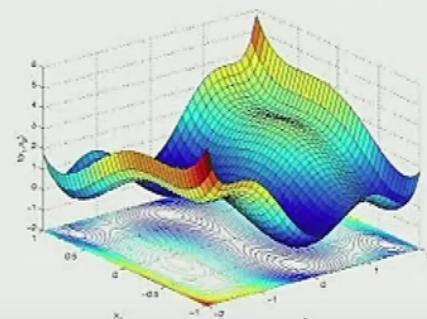
$\text{TrainLoss}(\mathbf{w})$

Linear functions



(convex loss)

Neural networks



(non-convex)

Optimization of neural networks is generally **hard**

The convex functions are those you can hold in hands 😊

However, the train loss for Neural nets turns out to be non-convex. Meaning using gradients to find optimum point can be hard, sometimes you end up in local optimum.

Nearest Neighbours

Nearest neighbors



Algorithm: nearest neighbors

Training: just store $\mathcal{D}_{\text{train}}$

Predictor $f(x')$:

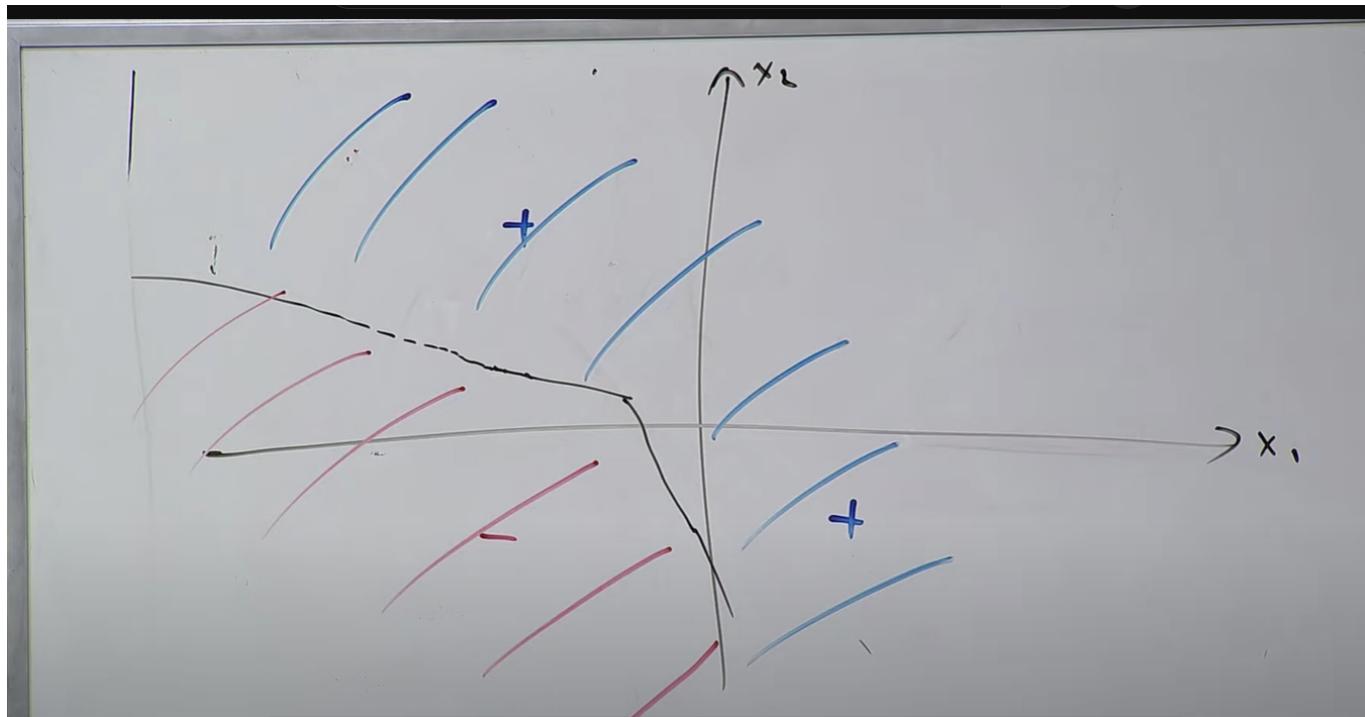
- Find $(x, y) \in \mathcal{D}_{\text{train}}$ where $\|\phi(x) - \phi(x')\|$ is smallest
- Return y



Key idea: similarity

Similar examples tend to have similar outputs.

离得越近 越有可能是一类



Can be computational expensive because u have store entire training set



Summary of learners

- Linear predictors: combine raw features
prediction is **fast**, **easy** to learn, **weak** use of features
- Neural networks: combine learned features
prediction is **fast**, **hard** to learn, **powerful** use of features
- Nearest neighbors: predict according to similar examples
prediction is **slow**, **easy** to learn, **powerful** use of features