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	$(w_2 - (-1))^2$

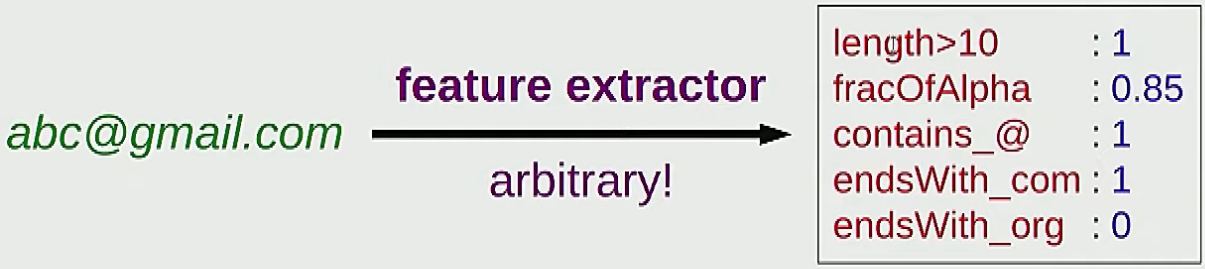
$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{3}\left((w_1 - 2)^2 + (w_1 - 4)^2 + (w_2 - (-1))^2\right)$$

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 extrator 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):

```
{"fracOfAlpha": 0.85, "contains_@": 1}
```

Feature templates

the phi - bottle neck

Hypothesis class

Hypothesis class

Predictor:

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

Definition: hypothesis class

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

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

All possiable 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:

$$\phi(x) = x$$

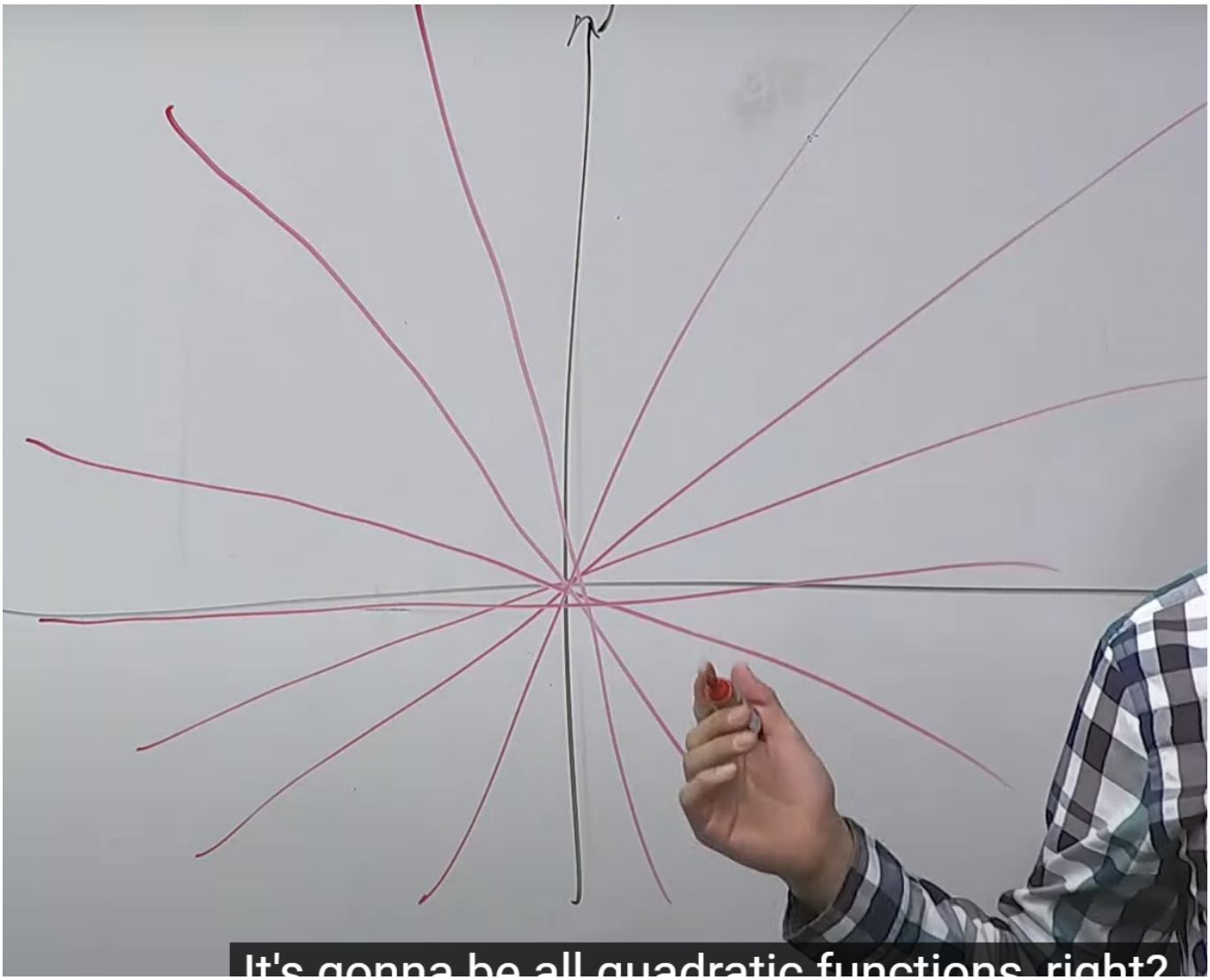
$$\mathcal{F}_1 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

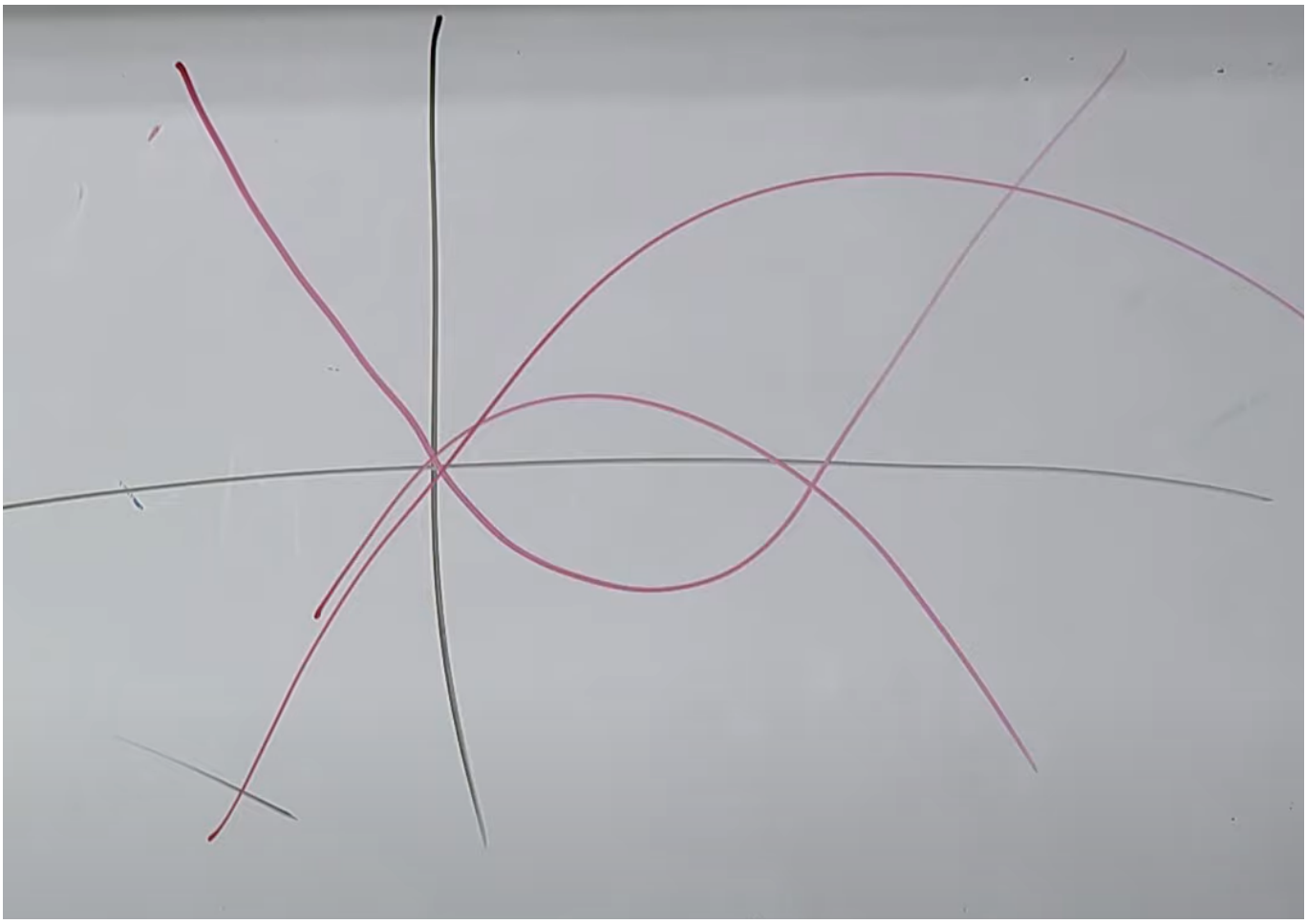
Quadratic functions:

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

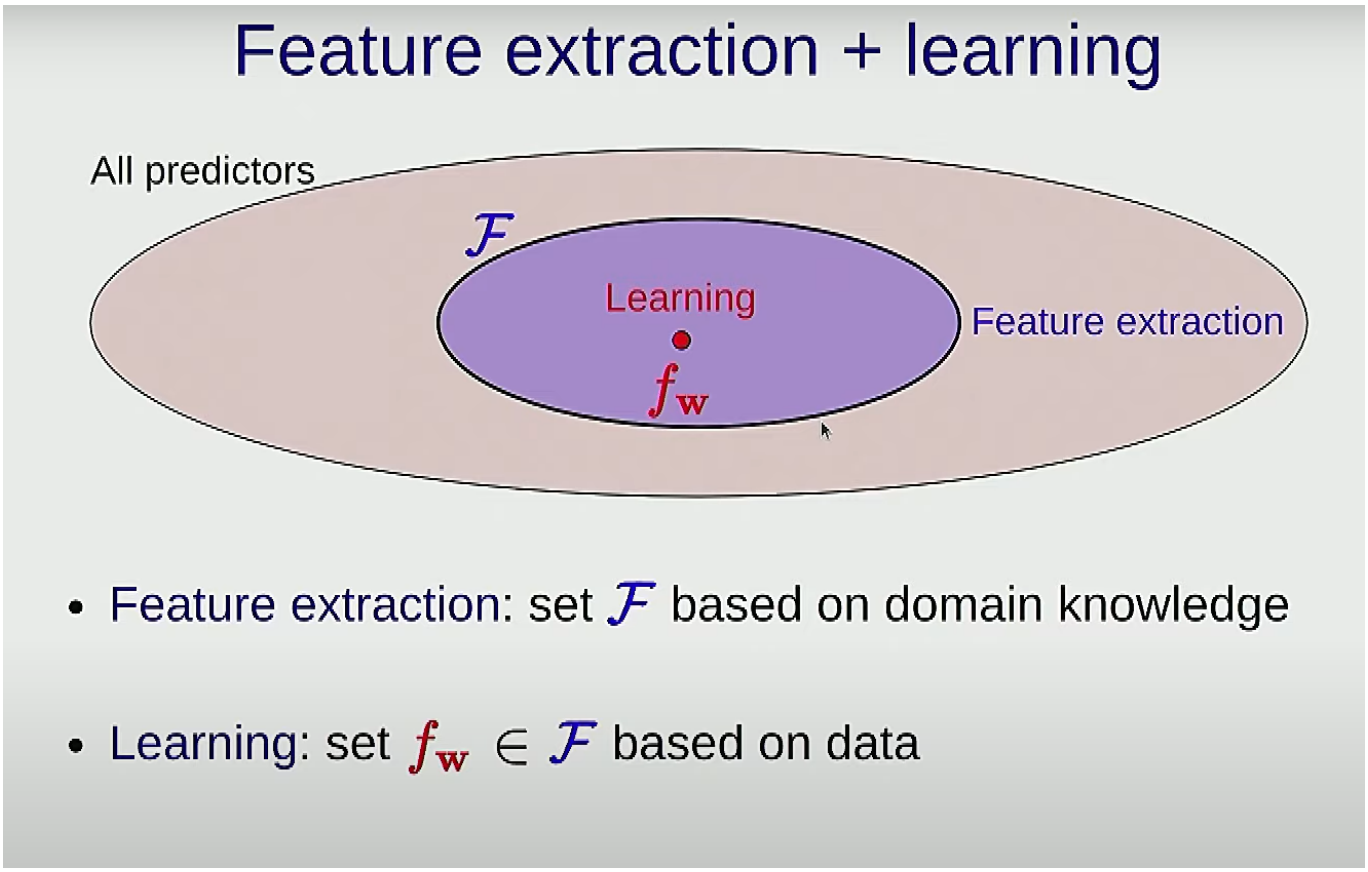
$$\mathcal{F}_2 = \{x \mapsto w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

[whiteboard]





Full pipeline of Machine learning



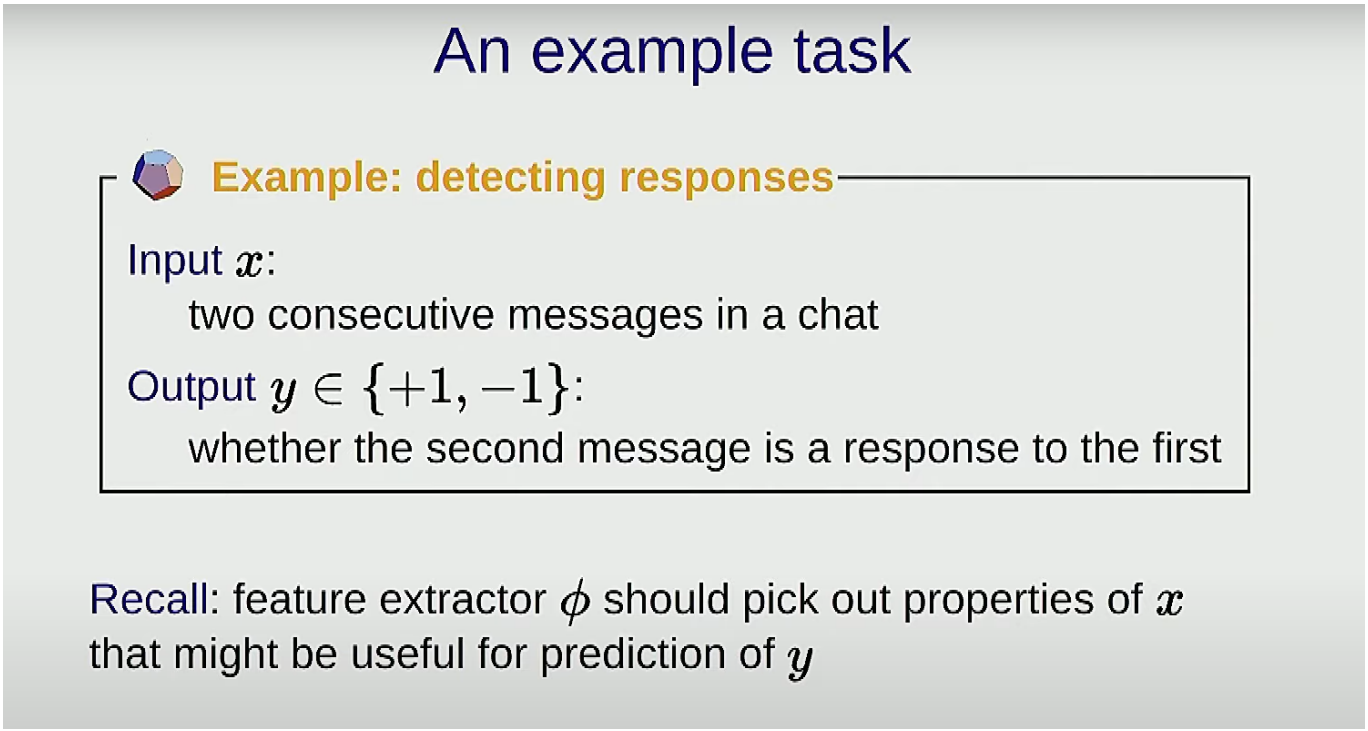
- The hypothesis class is the set of all predictors

- 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

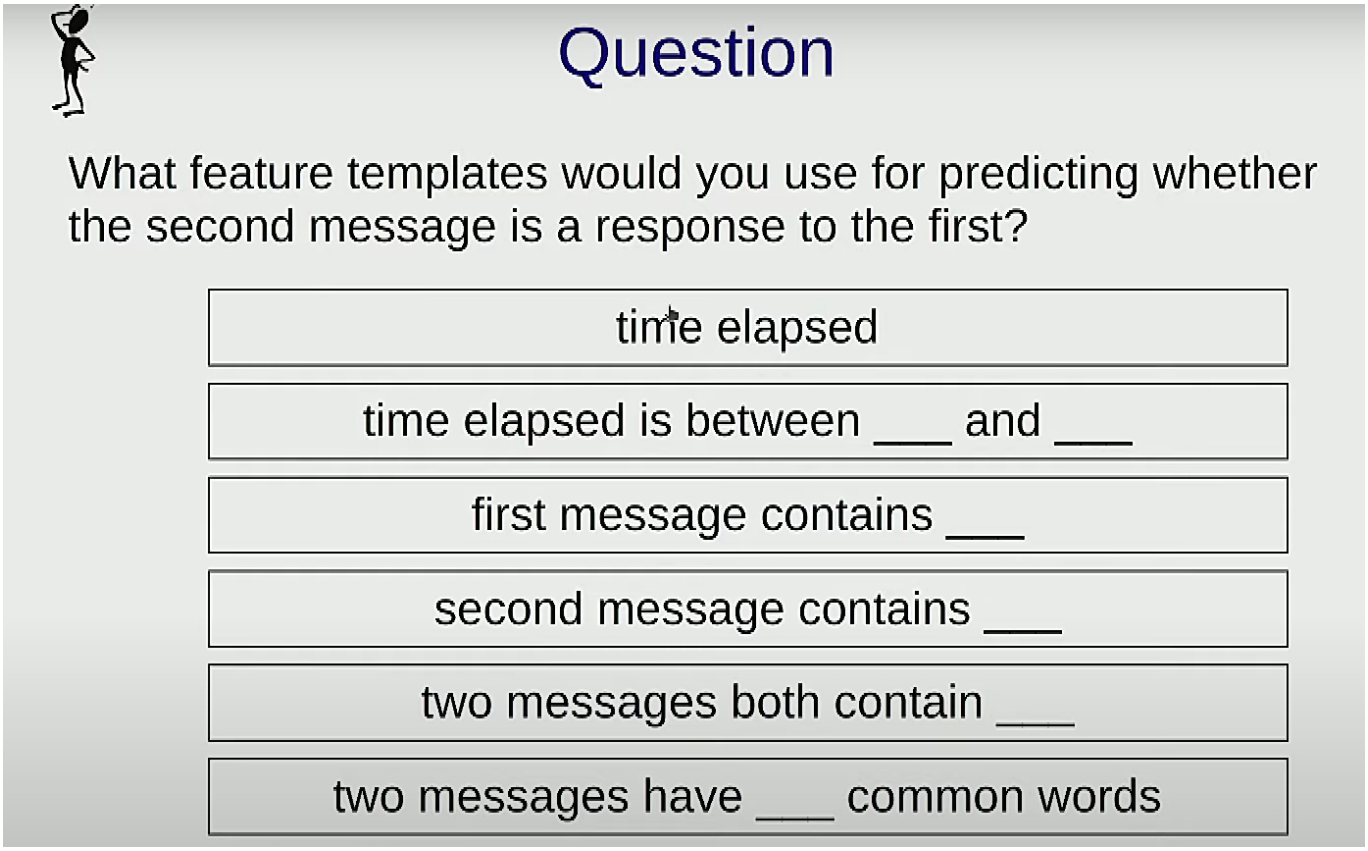
This means:

- 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



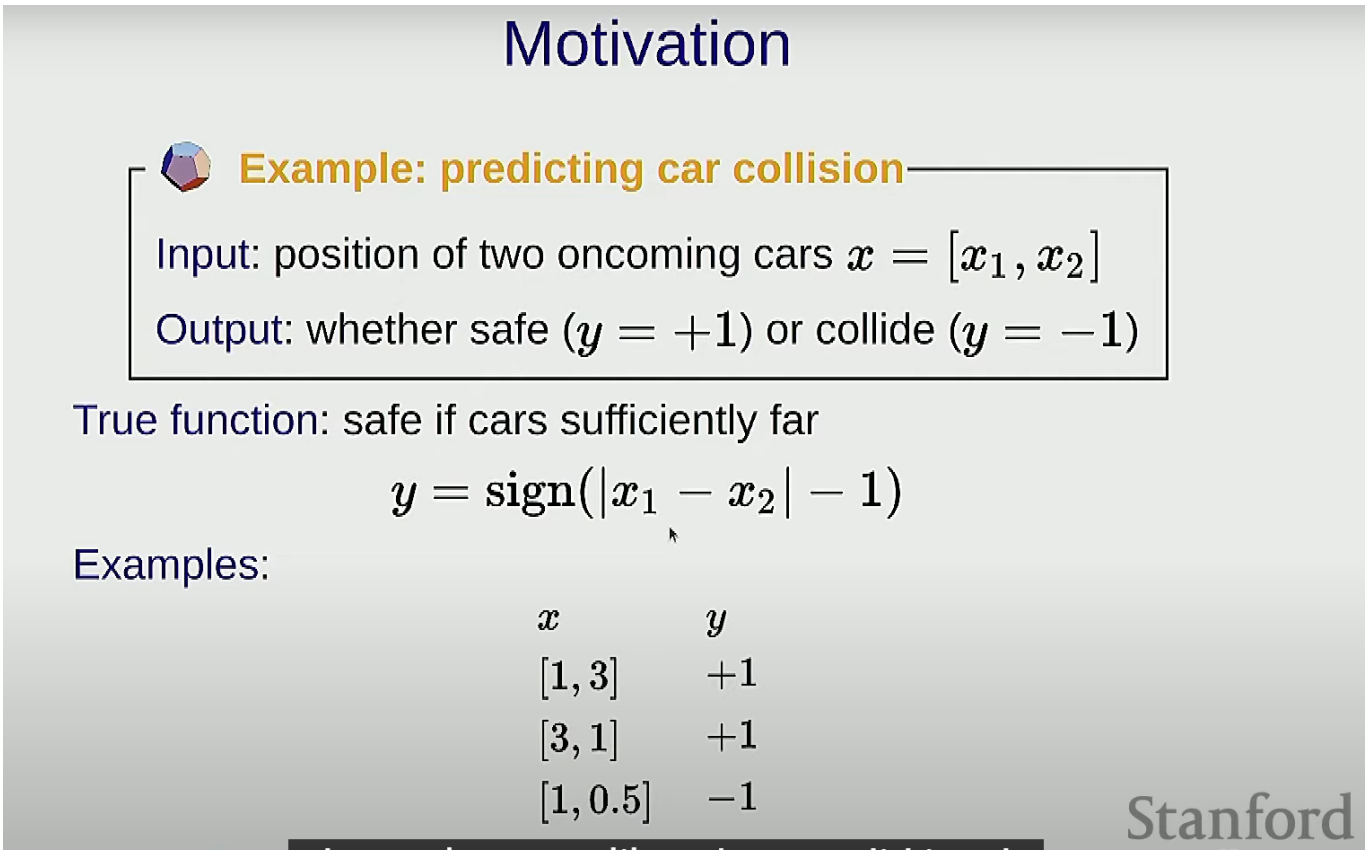
Feature tamplate:



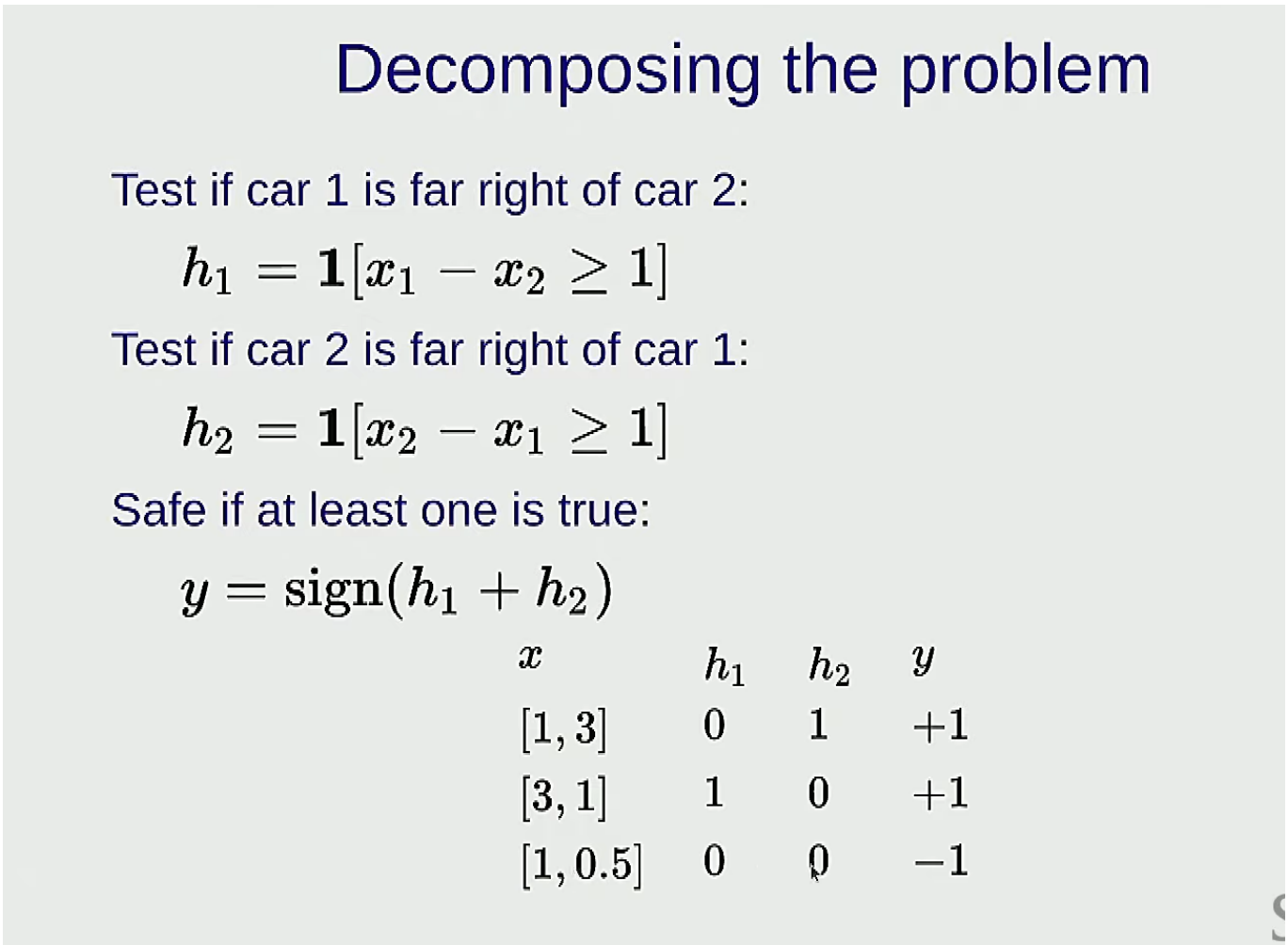
The `time elapsed` is a single number, it can be really small or large, which means, if it is really large, the contribution to the score gonna be really large, it going to affect the model too much, which you might not want to see. (too sensitive)

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

Neural Networks



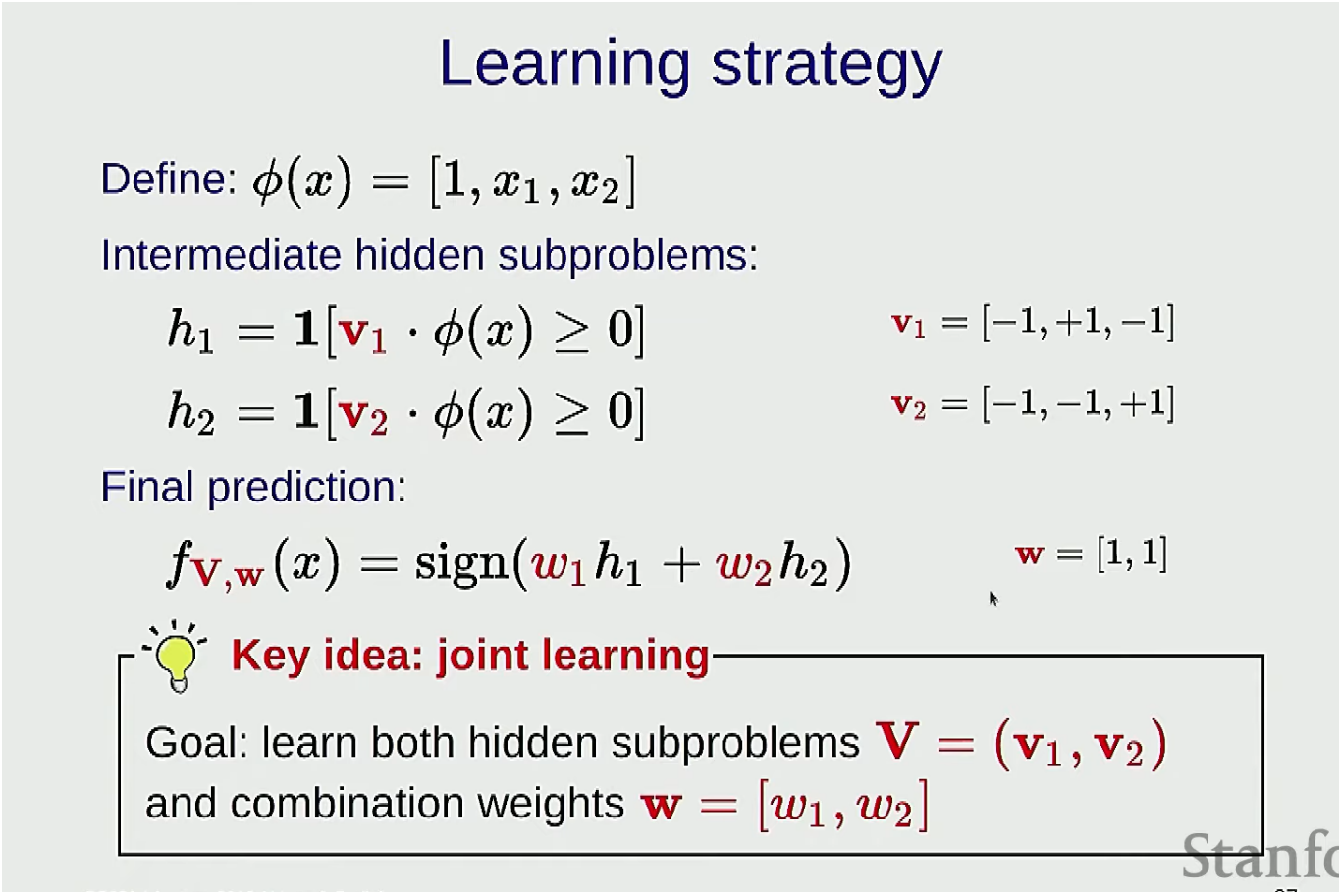
The true function: Check if the distance of two cars are larger than 1



Break down the problem

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

Well now It's make it abstract:



h1 = -1 + x1 - x2 >= 0

=>

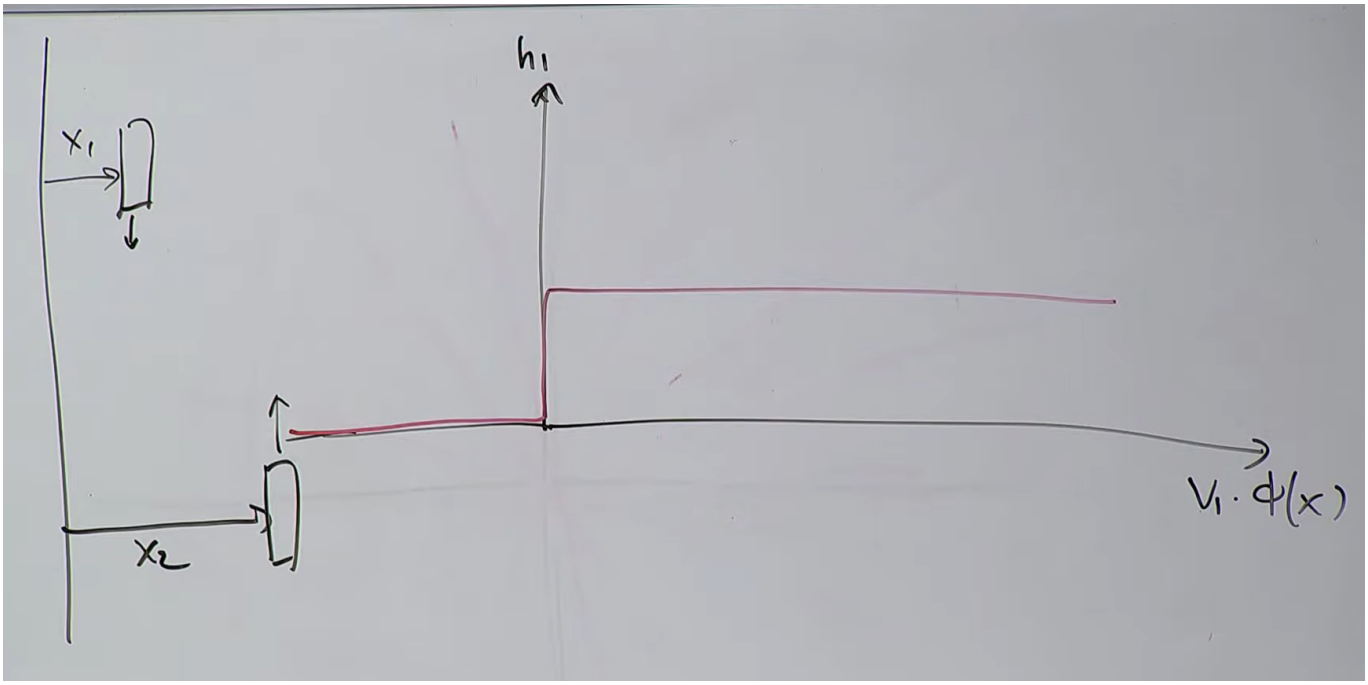
x1 - x2 >= 1

That are the same, just another forms of representation

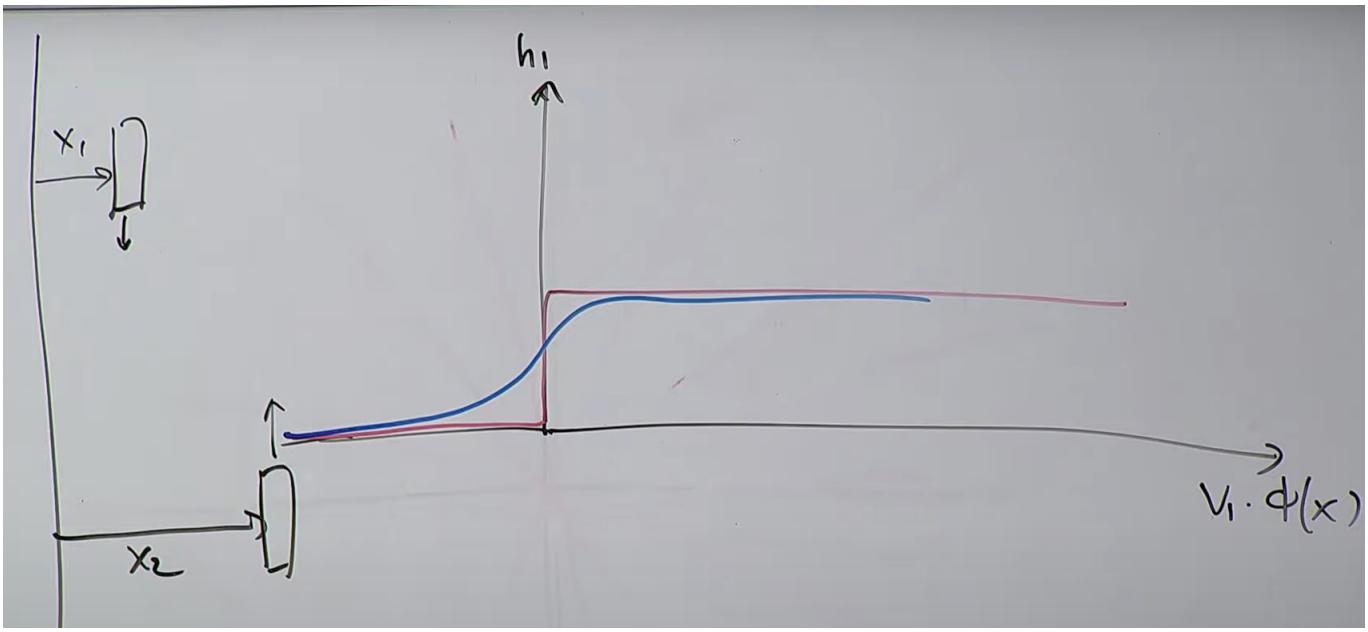
In the neural networks, we will leave `v1`, `v2`, `w` as unknown, and try to fit them through training

How do we learn it

There is a problem, h1 with respect to v1*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:

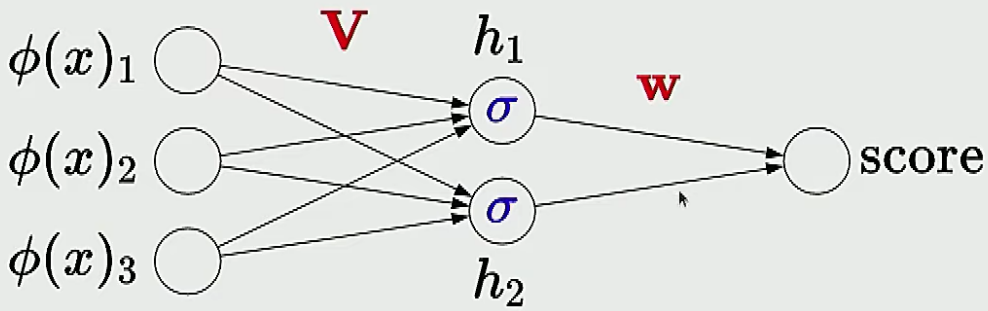
$$h_1 = \sigma(\mathbf{v}_1 \cdot \phi(x))$$

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:

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

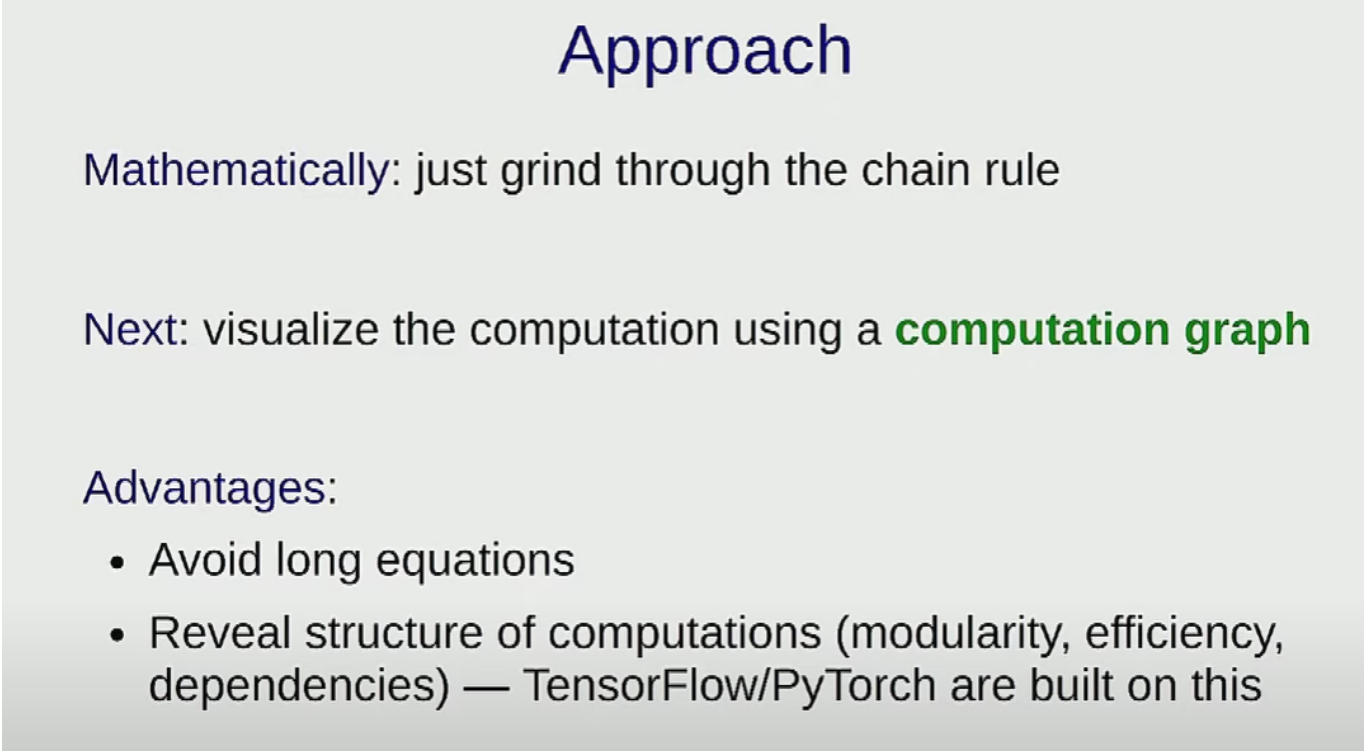
The neural nets are breaking a problem into subproblems, have the result of intermidiate computations, and they become feature of next layer

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 proceeds, you can derive more abstract features

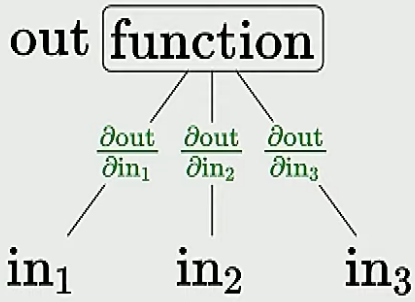
Do the neural learning

How calculate the gradient



Computation graph

Functions as boxes

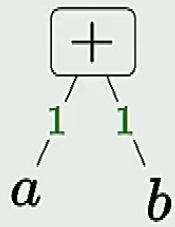


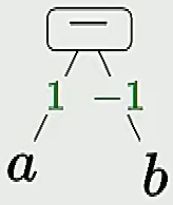
Partial derivatives (gradients): how much does the output change if an input changes?

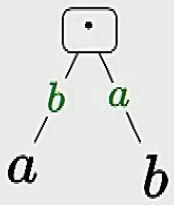
Example:

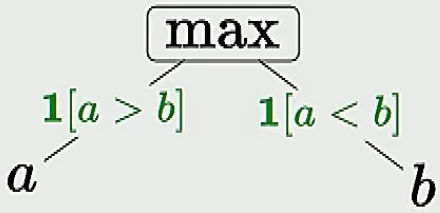
$$2(\text{in}_1 + \epsilon) + \text{in}_2\text{in}_3 = \text{out} + 2\epsilon$$

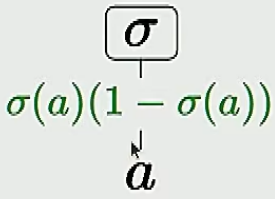
Basic building blocks





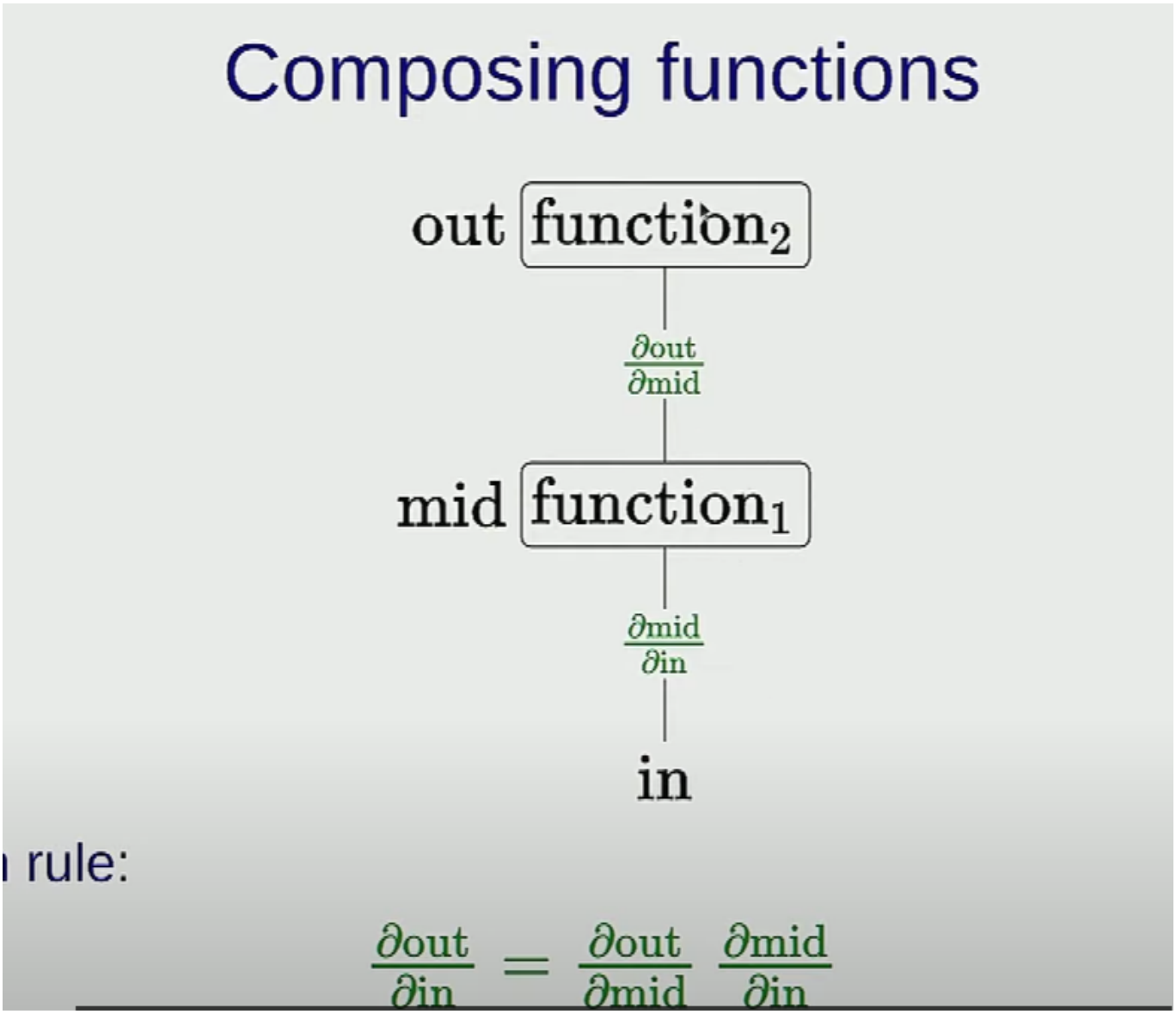




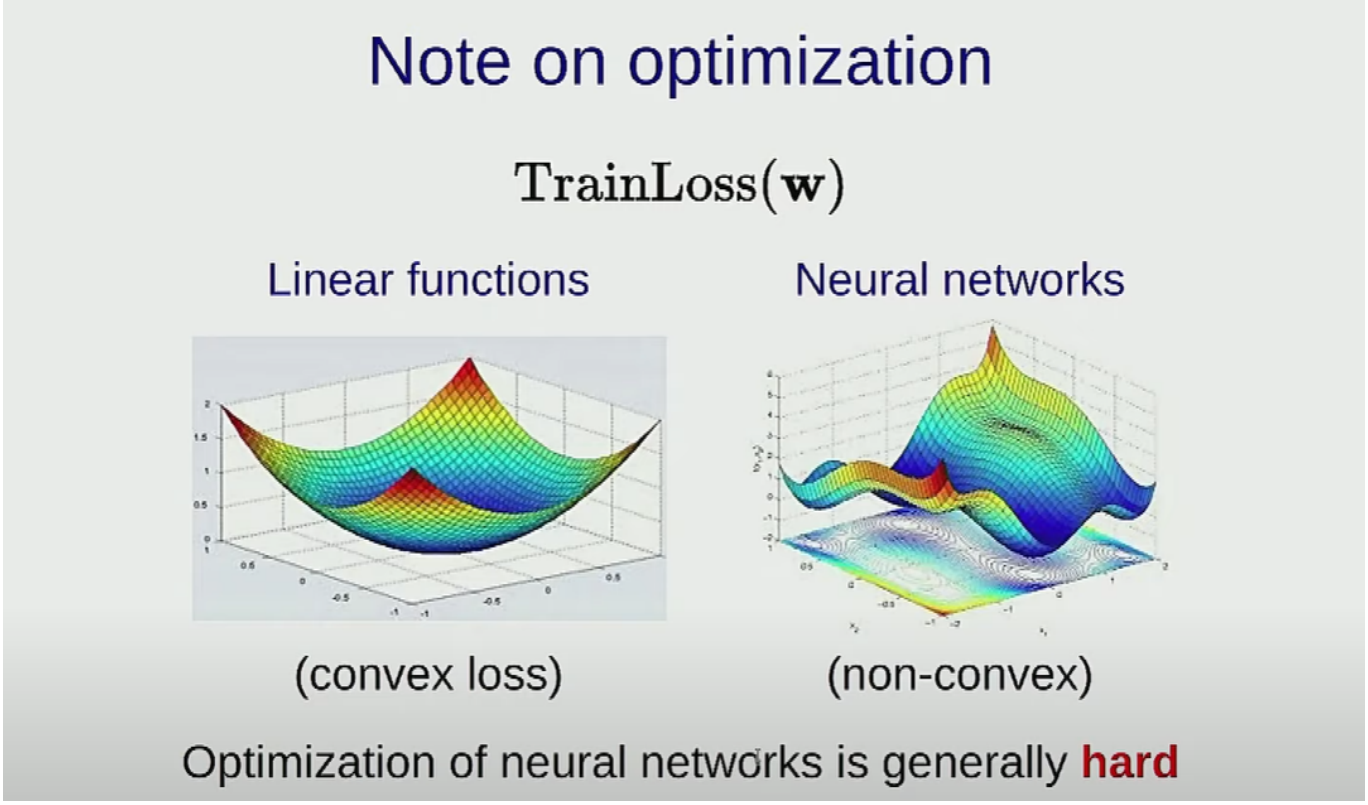


Think what if we change the input a little bit

The chain rule



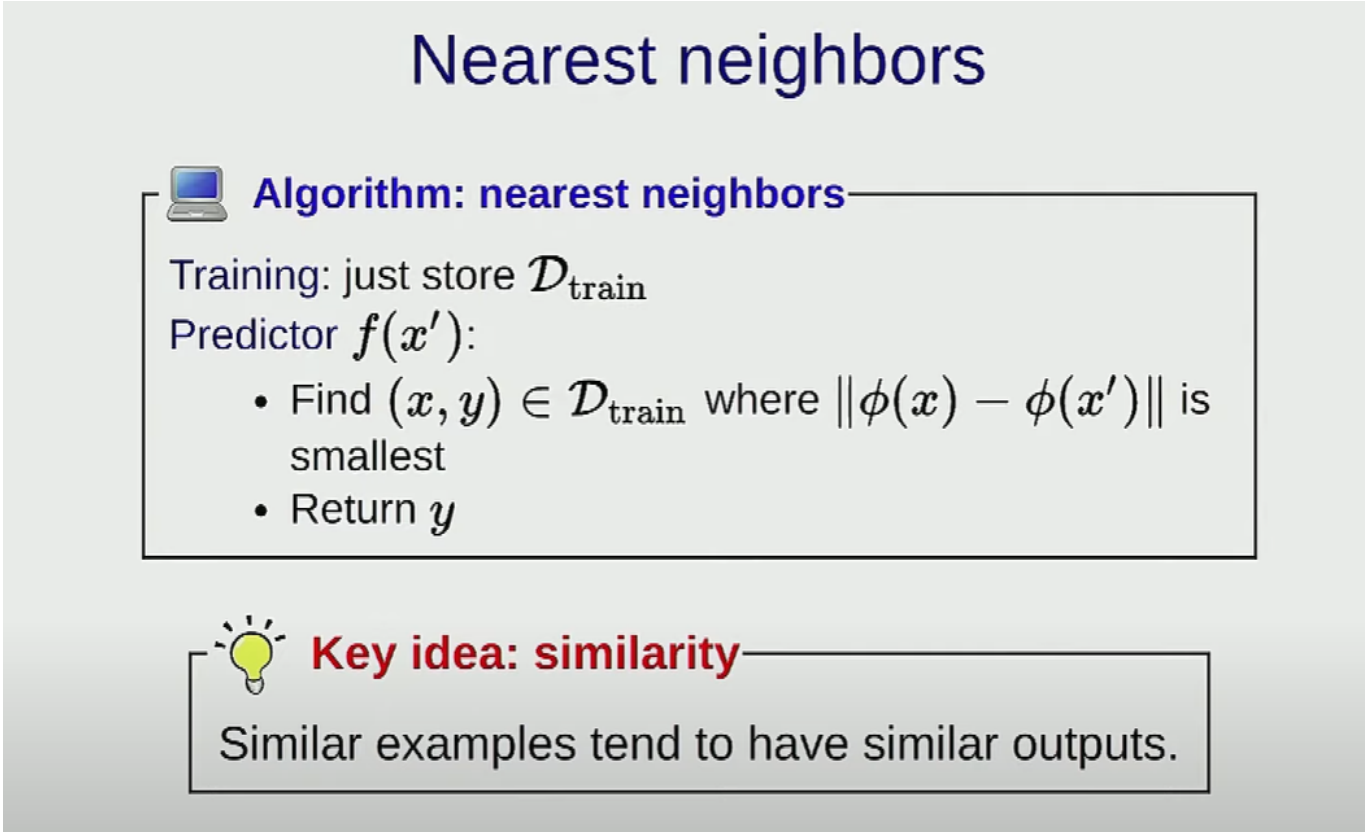
Note on optimization for Neural nets



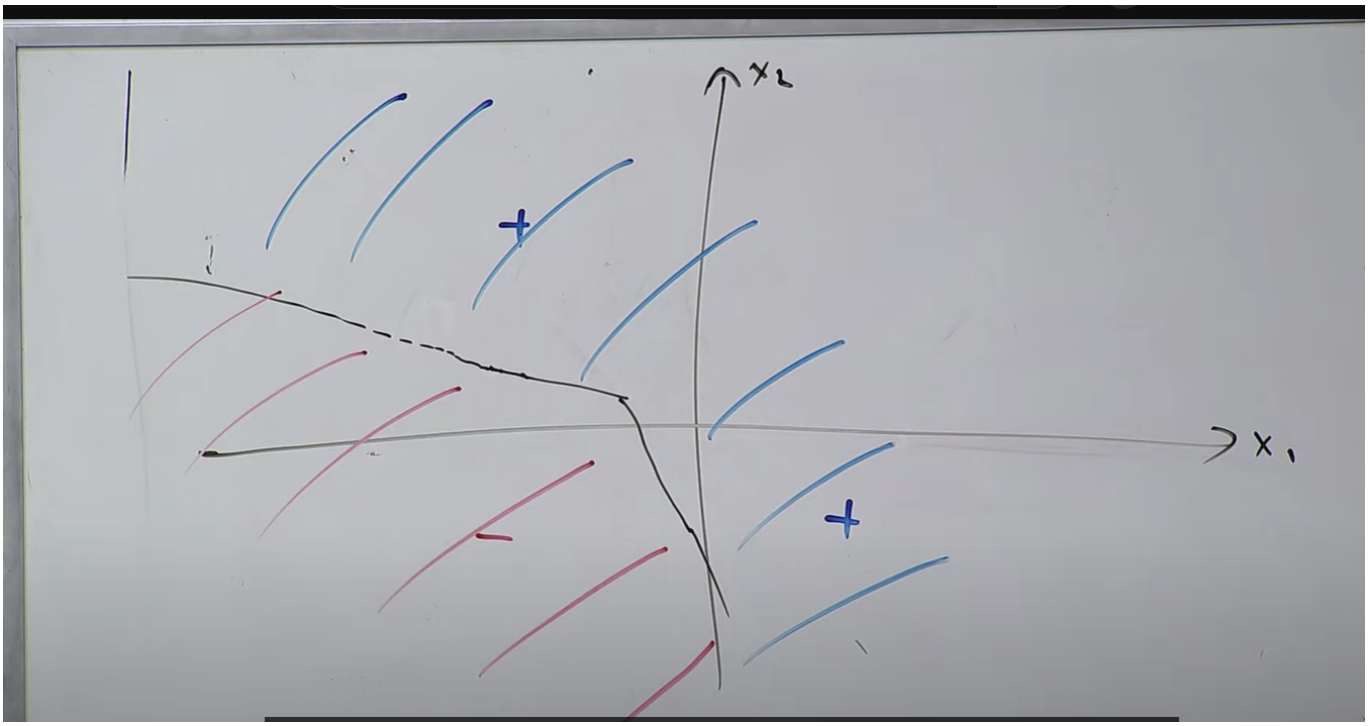
The convex functions are those you can hold in hands 😃

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 endup in local optimum.

Nearest Neighbours



离得越近 越有可能是一类



Can be computational expensive because u have store entire trainning 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