Deep Learning

Sunday, November 21, 2021 9:23 PM

$(x, f(x))$ but the nature of f is unknown.

What to try?

Halfspaces?

Kernels? Which one(s)?

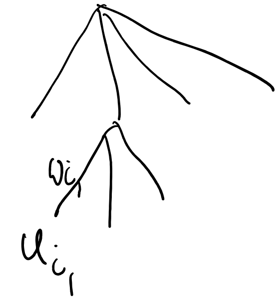
<u>Generic answer</u>: deep Feed Forward network with nonlinear activations

$\underbrace{\text{Train}}_{\text{Gradient Descent}}$ to minimize $\underbrace{\text{error of output}}_{\text{Loss}}$.

$$f(x) = \sum_i w_i u_i$$

$$\frac{\partial f}{\partial w_i} = u_i$$

$$f(x) = \sum_i w_i u_i \Big(\sum_{i_1} w_{i_1 i} u_{i_1} \Big)$$



$$\frac{\partial f}{\partial w_{i_1 i}} = \frac{\partial f}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{i_1 i}} = w_i \frac{\partial u_i}{\partial w_{i_1 i}}$$

$$f(x) = \sum_i w_i u_i \Big(\sum_{i_1} w_{i_1} u_{i_1} \Big(\sum_{i_2} w_{i_2} u_{i_2} \Big) \Big)$$

$$\frac{\partial f}{\partial w_{i_2}} = \sum_i \frac{\partial f}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{i_2}}$$

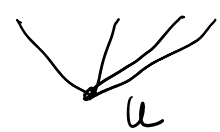
$$= \sum_i w_i \cdot \frac{\partial u_i}{\partial u_{i_1}} \cdot \frac{\partial u_{i_1}}{\partial w_{i_2}}$$

$$= \Big(\sum_i w_i \cdot w_{i_1, i} \Big) \frac{\partial u_{i_1}}{\partial w_{i_2}}$$

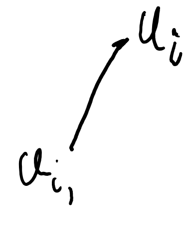
Computing $\frac{\partial f}{\partial u}$ for all u suffices

Backprop.

- $\frac{\partial f}{\partial f} = 1$, send downward



- For node u
  - sum all incoming values to get δ
  - send $\delta \cdot \frac{\partial u}{\partial u_{i_1}}$ to each u_{i_1} below.



<u>Lemma</u>. $\delta = \frac{\partial f}{\partial u}$.

<u>Pf</u>. Induction + Chain Rule.

base: $\frac{\partial f}{\partial f} = 1$.

step: u_{i_1} receives $\underline{\frac{\partial f}{\partial u}} \cdot \frac{\partial u}{\partial u_{i_1}}$ from u

Hence computes induction hypothesis

$$\sum_u \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial u_{i_1}} = \frac{\partial f}{\partial u_{i_1}}.$$

Time: $O(m)$.

Fast, general, but is it any good?

- If no hidden layer then GD $\longrightarrow$ OPT.
- If hidden layers, heavily <u>overparametrized</u> more parameters than data then GD $\longrightarrow$ OPT. (<u>But does it generalize?!</u>)

Thm. ... approximate any

Thm 1. Neural Networks can approximate any continuous function. [Cybenko 89, Hornik-Stinchcombe-White 89, Barron 93]

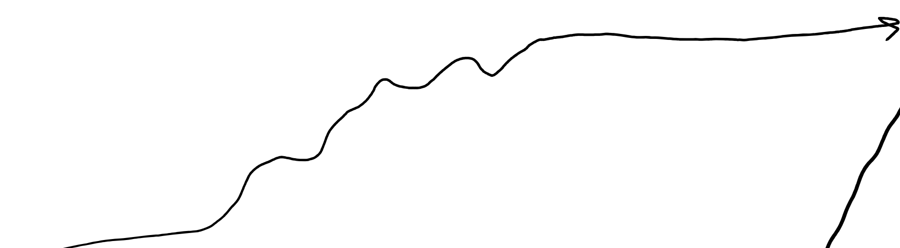
In fact, depth 2 suffices! Sufficiently wide
(= 1 hidden layer)

But depth can help — $\exists$ functions that need a large representation with small depth and a small representation with moderate depth.

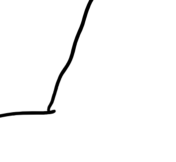
Thm 1 needs activation units to be "sigmoidal"

$\sigma(x) \to 0 \quad x \to -\infty$

$\sigma(x) \to 1 \quad x \to \infty$



What about ReLU?



Put two together:

put two together: