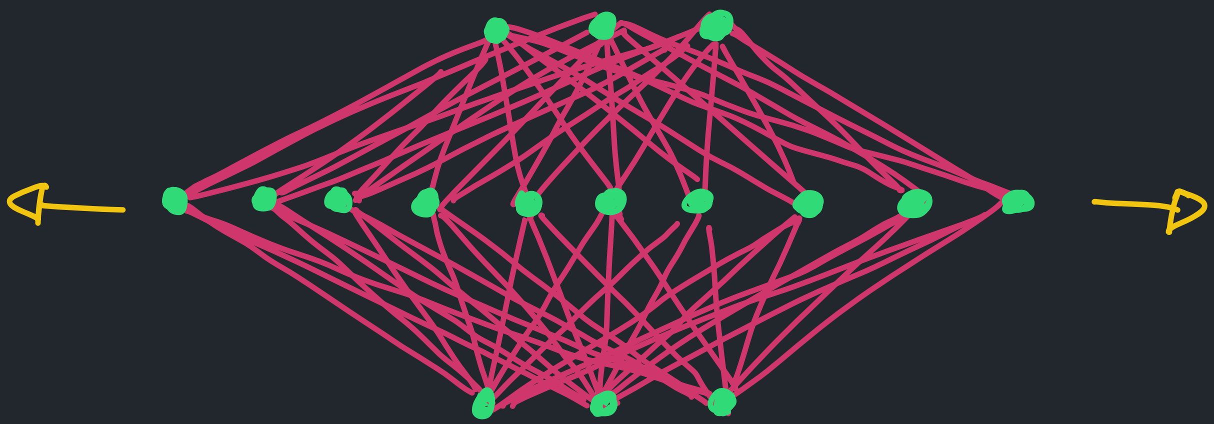
Approximation Theory

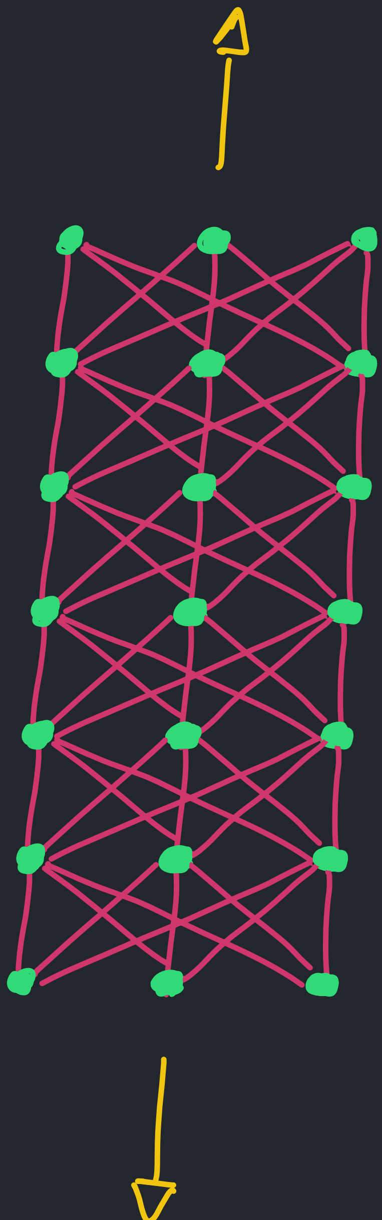
Jeremy Bernstein
jbernstein@mit.edub

6.7960 :: Lecture 3

Would you rather...



scale width



scale depth

Today's lecture

What class of functions can a neural net express?

"neural nets are universal function approximators"

Then what architecture should I use?

"three layers are enough"

"stack more layers!!!"

....then what should I do in practice?

The machine learning puzzle

Three pieces to the puzzle:

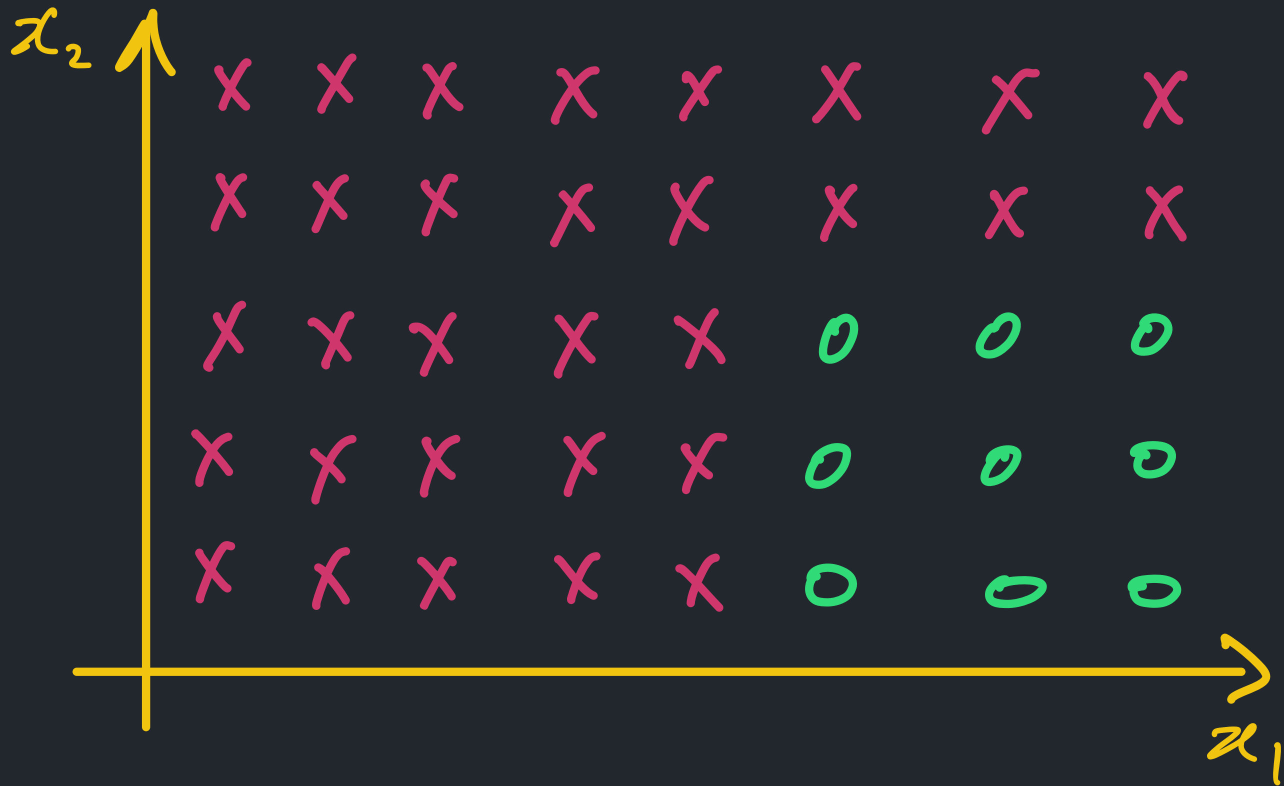
1. <u>Approximation</u> Does there <u>exist</u> a neural net in my model family that fits the training data?

2. <u>Optimization</u> If it does exist, can I <u>find</u> it?

3. <u>Generalization</u> Does it <u>work well</u> on <u>unseen</u> data?

This lecture will focus mainly on the first question.

A motivating problem



Can we classify this data with the function:

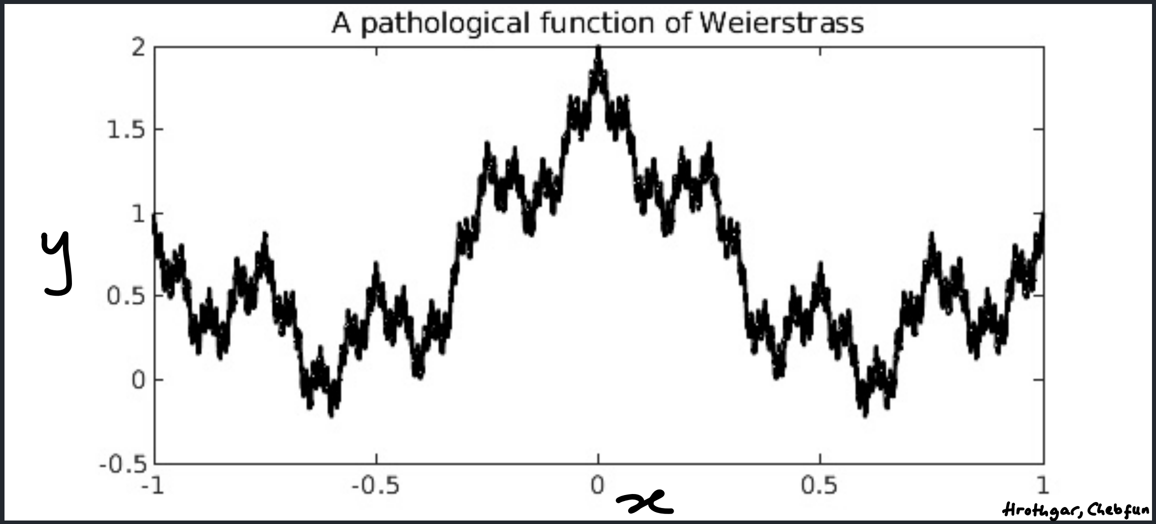
$$f(x) = \text{relu}(w^T x + b) \quad ?$$

Or what about with "two layers":

$$f(x) = \sum_i \alpha_i \, \text{relu}(w_i^T x + b_i) + \beta_i \quad ?$$

Another motivating problem

An example of a "fractal curve"



Weierstrass' function is everywhere continuous
nowhere differentiable

→ Can you fit it with a neural network?
If so, how big should the network be?

Formalizing the approximation problem

Given a family of curves G — exclude pathological functions

And a family of neural networks F — e.g. 5 layer relu MLPs

For any curve $g \in G$

Does there exist a neural net $f \in F$

Such that $\text{error}(f, g) < \varepsilon$? — think: small number

e.g. $\text{error}(f, g) \triangleq \max_x |f(x) - g(x)|$ "L_∞ error"

or $\text{error}(f, g) \triangleq \int dx\, |f(x) - g(x)|$ "L_1 error"

One nice family of curves G

Lipschitz continuous functions

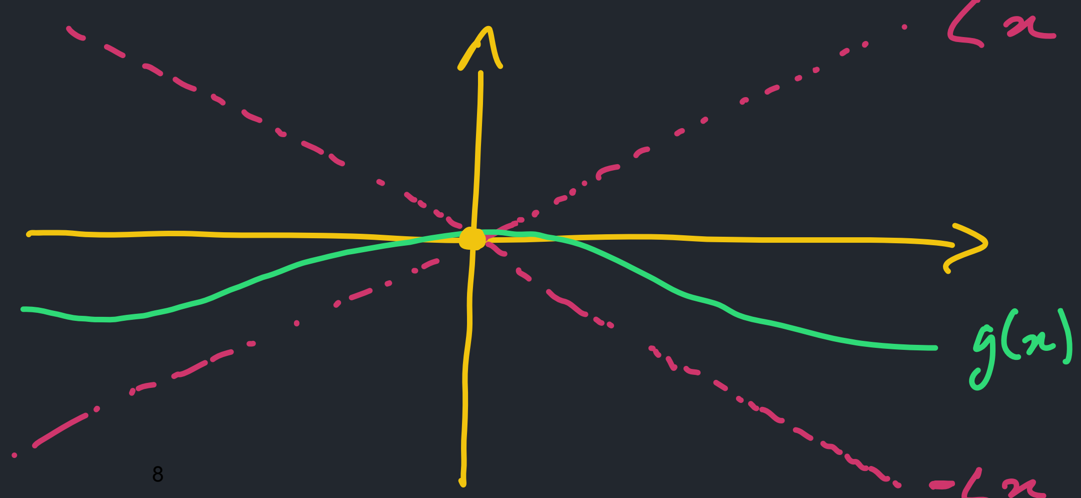
$g : \mathbb{R} \to \mathbb{R}$ is "L-Lipschitz" if

$$|g(x+\Delta x) - g(x)| \leq L\,|\Delta x|$$

for all inputs $x \in \mathbb{R}$ and all $\Delta x \in \mathbb{R}$.

Intuition: the slope of g cannot exceed L.

If g passes through the origin then it can never stray outside $\pm L x$.



Extending to multi-dimensional inputs

Lipschitz continuous functions

$g : \mathbb{R}^d \to \mathbb{R}$ is "L-Lipschitz" if

$$|g(x+\Delta x) - g(x)| \leq L \|\Delta x\|_{RMS}$$

for all inputs $x \in \mathbb{R}^d$ and all $\Delta x \in \mathbb{R}^d$.

Define the "RMS-norm"

$$\|x\|_{RMS} \triangleq \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2},$$

↳ it measures the average (root-mean-square) size of the entries of the vector.

In this lecture, we will prove...

d-dimensional hypercube.

Theorem:

Let $g : [0,1]^d \to \mathbb{R}$ be any L-Lipschitz function.

Then for any error $\varepsilon > 0$

There exists a 3-layer relu network

With $N = 4d(L/\varepsilon)^d$ units

such that $\displaystyle\int_{[0,1]^d} |f(x) - g(x)|\, dx < 2\varepsilon$

Strategy for proving the result

Step one:

derive result for 1d inputs
via approximation with
"rectangular strips"

ie ignore relu
networks for now.

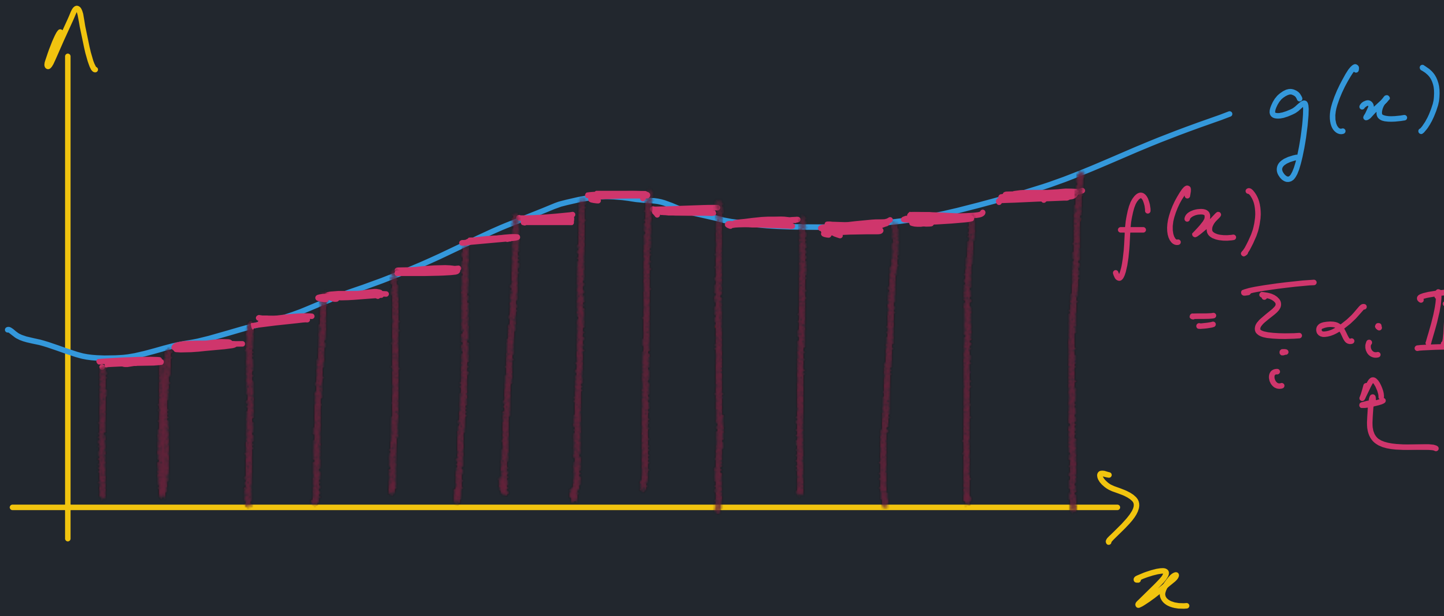
Step two:

generalise to higher input
dimension

Step three:

show that relu networks can
approximate rectangular strips

Step One: Approximation with rectangles



$$= \sum_i \alpha_i \, \mathbb{I}[x \in i^{th} \text{ interval}]$$

height of i^{th} rectangle

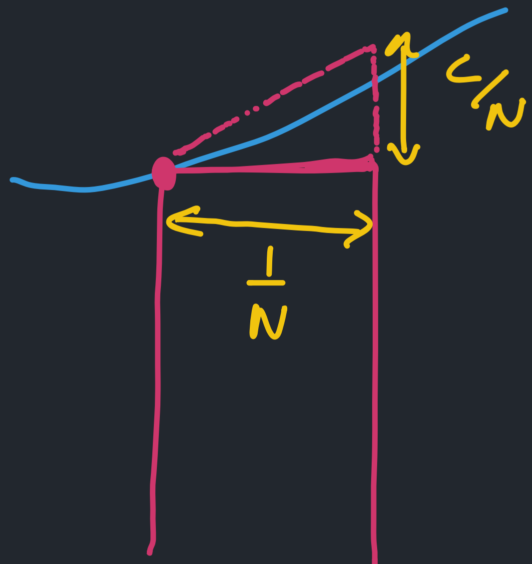
Given N rectangular strips, what is the approximation error?

CLAIM: error $\leq \frac{L}{2N}$

increases with Lipschitz constant

decreases with number of strips

N strips $\Rightarrow$ each strip has width $1/N$



by Lipschitzness the curve can't exceed the top side of the triangle

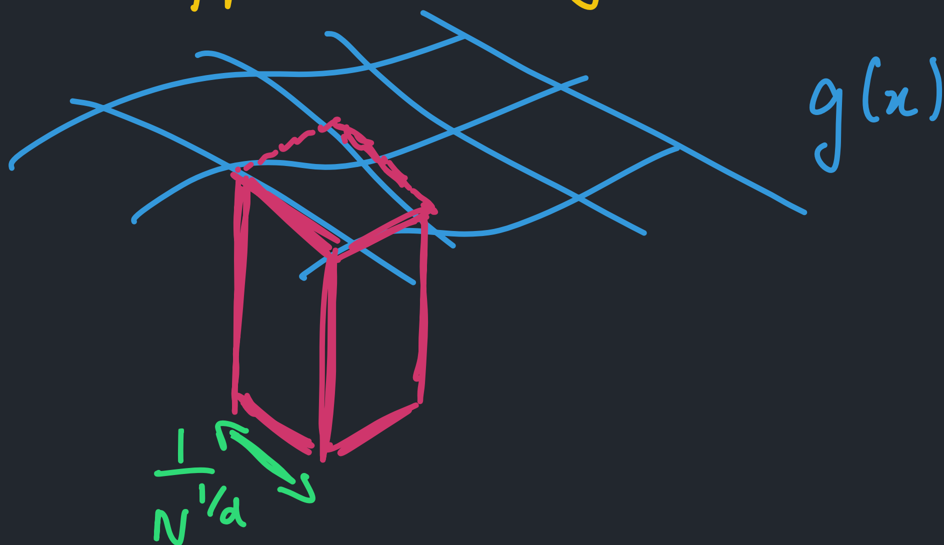
The triangles has area $\frac{1}{2}\frac{L}{N^2}$

$$\text{Total error} = \int dx \, |f(x) - g(x)| \leq N \times \frac{1}{2}\frac{L}{N^2}$$
$$= \frac{1}{2}\frac{L}{N}$$

So to achieve error ε we need $N = \frac{1}{2}\frac{L}{\varepsilon}$

Step two: Higher input dimension

Think: approximating a surface with cuboids



N hyperrectangles $\Rightarrow$ each has side of Euclidean length $\frac{1}{N^{1/d}}$

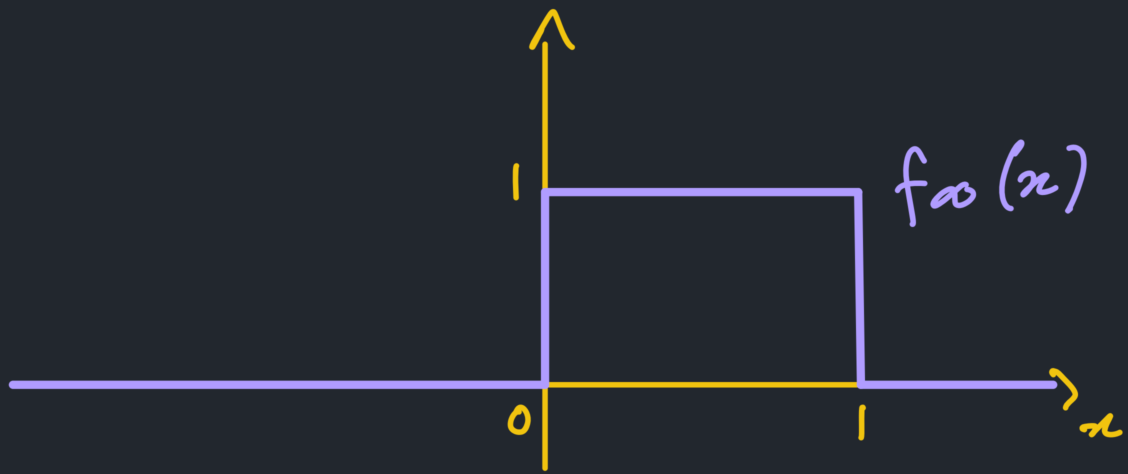
$$\text{total error} = \int dx\, |f(x) - g(x)| \leq \underbrace{N}_{\text{number of hyperrectangles}} \times \underbrace{\frac{\ell}{N^{1/d}}}_{\text{height of error cap}} \times \underbrace{\frac{1}{N}}_{\text{area of error cap.}} = \frac{\ell}{N^{\frac{1}{d}}}$$

To get error ε requires $N = \left(\frac{\ell}{\varepsilon}\right)^d$ hyperrectangles

Step Three: Relu networks can fit rectangles

Now define $$f_c(x) = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}^T \mathrm{relu} \begin{bmatrix} cx \\ cx - 1 \\ c(x-1) - 2 \\ c(x-1) - 3 \end{bmatrix}$$

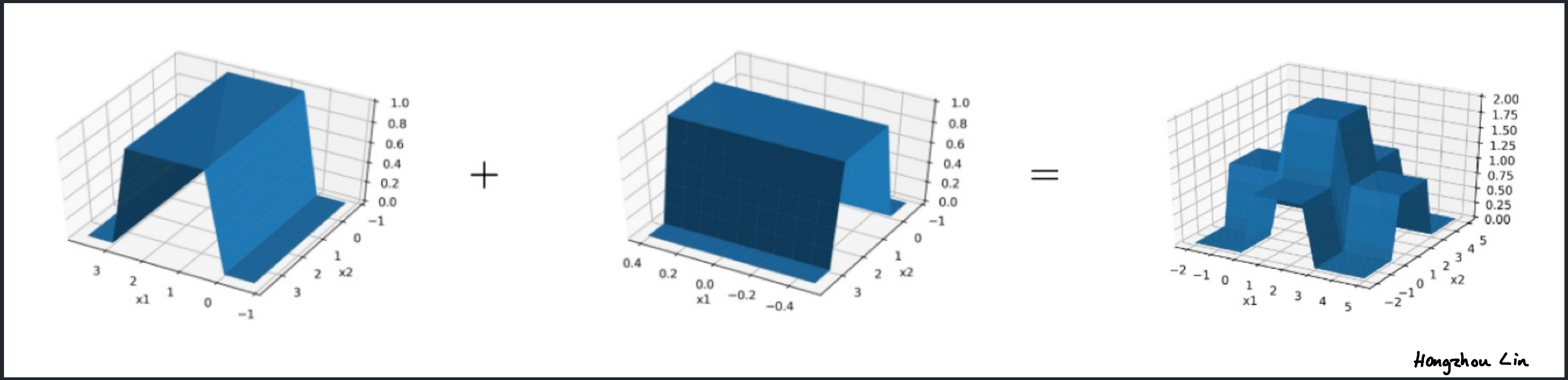
CLAIM: let $c \to \infty$ and we get:



↳ can also translate horizontally by adjusting weights and biases.

But we need d-dimensional hyperrectangles...

Solve by adding 1-dimensional rectangles and thresholding appropriately



Only exceeds d-1 when all rectangles are "on"

↳ so just threshold at d-1.

Assembling the pieces

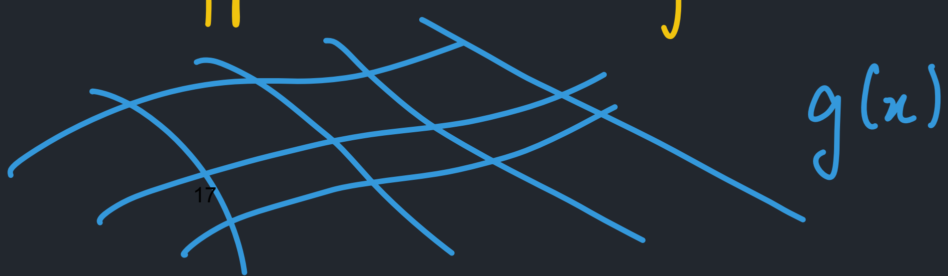
rectangle: $$f_c(x) = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}^T \mathrm{relu} \begin{bmatrix} cx \\ cx - 1 \\ c(x-1) - 2 \\ c(x-1) - 3 \end{bmatrix}$$

hypperrectangle: $$h_c(x) = \mathrm{relu}\left[\sum_{i=1}^{d} f_c(x_i) - (d-1)\right]$$

linear combination of hyperrectangles $$f(x) = \sum_i \alpha_i \, h_c(x - u_i)$$

position of i^{th} gridpoint

Let $c \to \infty$ and we get an approximation of an arbitrary Lipschitz surface!



Some comments

Theorem:

Let $g : [0,1]^d \to \mathbb{R}$ be any L-Lipschitz function.

Then for any error $\varepsilon > 0$

There exists a 3-layer relu network

With $N = 4d(L/\varepsilon)^d$ units

such that $\int_{[0,1]^d} |f(x) - g(x)| \, dx < 2\varepsilon$

General idea: approximate "bumps" then linearly combine

Needs exponentially many neurons in dimension

Taking $c \to \infty$ feels unrealistic

Approximating rectangles feels like a trick

Imagine training the rectangle representation

- would just get rectangles on the training points X
- regularisation (weight decay) would suppress the others
- would not generalise!

Further reading

More results on universal function approximation

- Barron's theorem

  "Smooth functions can be approximated with fewer neurons"

  leverages Fourier representation

- 2 layers are enough

  e.g. Hornik, Stinchcombe and White (1989)

  uses Stone-Weierstrass theorem

Is universal function approximation important?

Is "UFA" sufficient for learning to work?

No, there are many UFAs that we usually don't do ML with:

Examples: Fourier series

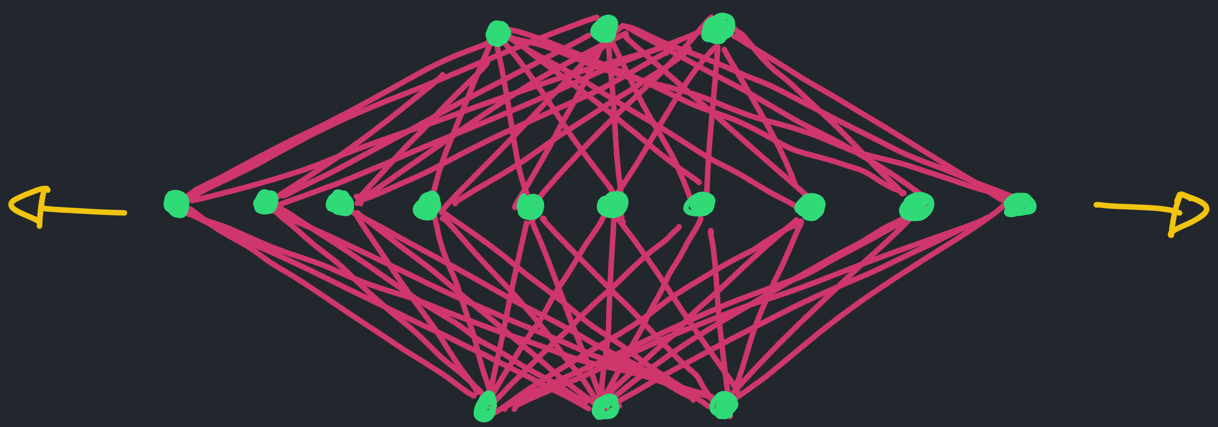
polynomials

the space of Python programs

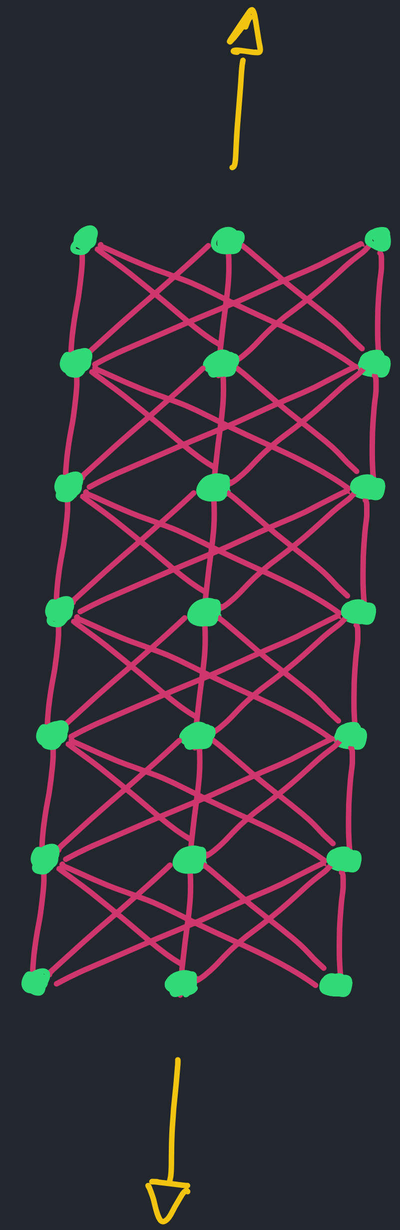
Is "UFA" necessary for learning to work?

Width versus depth

Would you rather...



scale width



scale depth

Width versus depth

<u>Advantages of scaling width</u>

- 3 layer (or even 2 layer) NNs are "universal function approximators"
- width is inherently parallelisable, depth is sequential
- width is easier to train, depth leads to "compound problems"

So, scaling width is <u>obviously</u> better!

Or is it?.... Depth separations

Universal function approximation results suggest needing exponentially many hidden units at small width

"depth separation" results construct deep networks that require exponentially more units to fit with a shallow network.

The shape of a result

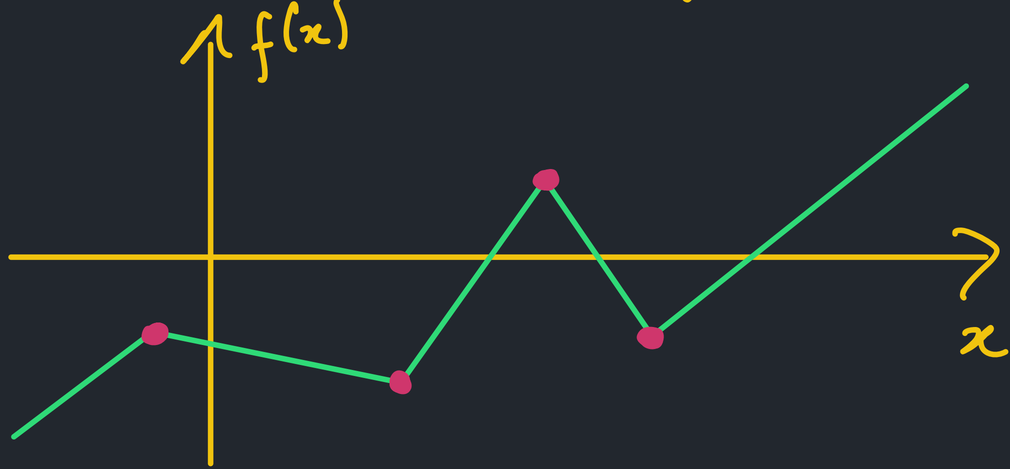
To prove a "depth separation"

1. pick a property of a function — e.g. "number of linear regions"

2. construct a deep network that has this property

3. prove that a shallow network would need exponentially more units to also have this property

Piecewise linear functions



we will define a "kink" to be a place where the gradient changes

Here, #kinks = 4

Claim: relu networks are piecewise linear ("PWL")

Why? relu is PWL

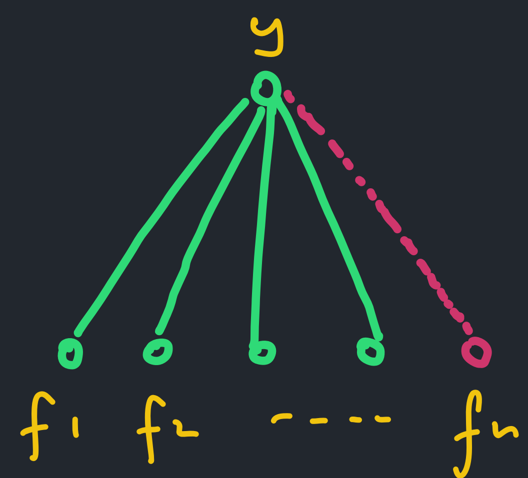
f, g both PWL $\Rightarrow$ $f+g$ PWL

f, g both PWL $\Rightarrow$ $f \circ g$ PWL

f PWL $\Rightarrow$ $\alpha \cdot f$ PWL for $\alpha \in \mathbb{R}$

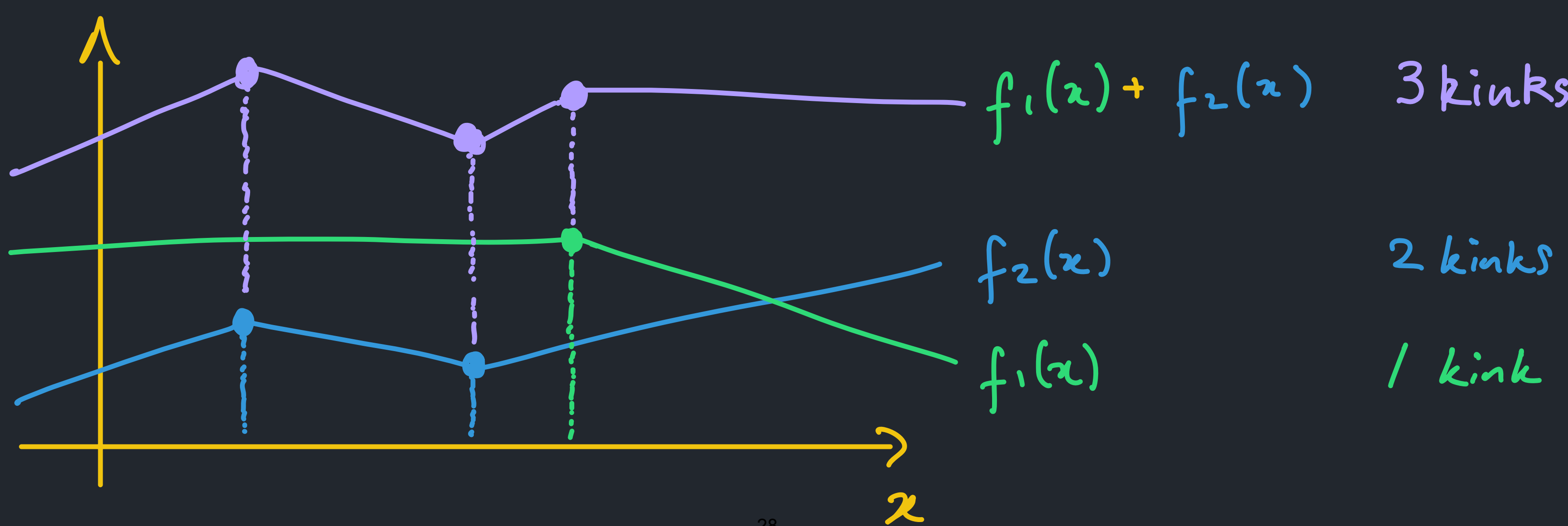
Okay, let's construct a depth separation based on #kinks

Intuition: Effect of adding width

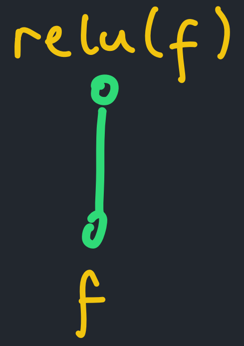


$$y(x) = \sum_{i=1}^{n} \alpha_i f_i(x)$$

When we <u>add</u> functions, at most we <u>add</u> the # kinks.

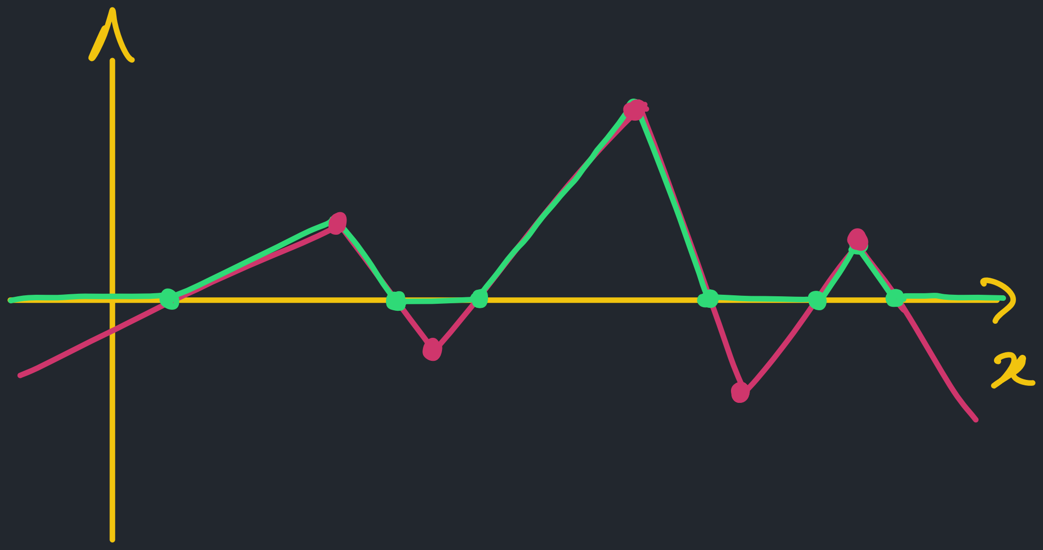


Intuition : Effect of applying relu



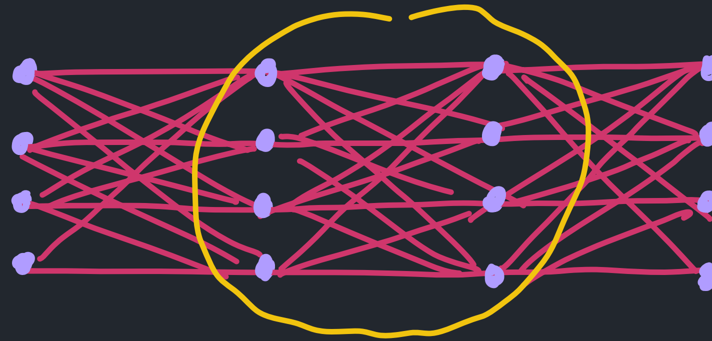
$$y(x) = \mathrm{relu}(f(x))$$

When we apply relu, at most we double the # kinks



Because linear pieces can be split in two.

More formally



Consider the l^{th} layer:

vectors in $\mathbb{R}^n$

$$\underbrace{f_l(x)}_{\text{vector in } \mathbb{R}^n} = \mathrm{relu}\Big(\underbrace{W_l}_{n \times n \text{ matrix}} \, f_{l-1}(x) + b_l \Big)$$

Let $KINKS_l$ denote the max number of kinks over the n coordinates of $f_l(x)$.

Then it holds that $KINKS_l \leq 2n \cdot KINKS_{l-1}$

Since $KINKS_0 = 1$, this implies $\boxed{KINKS_l \leq (2n)^l}$

Interpreting the result

We showed that: $\boxed{\text{KINKS}_L \leq (2n)^L}$

KINKS_L = # kinks in the function at layer L

n = width

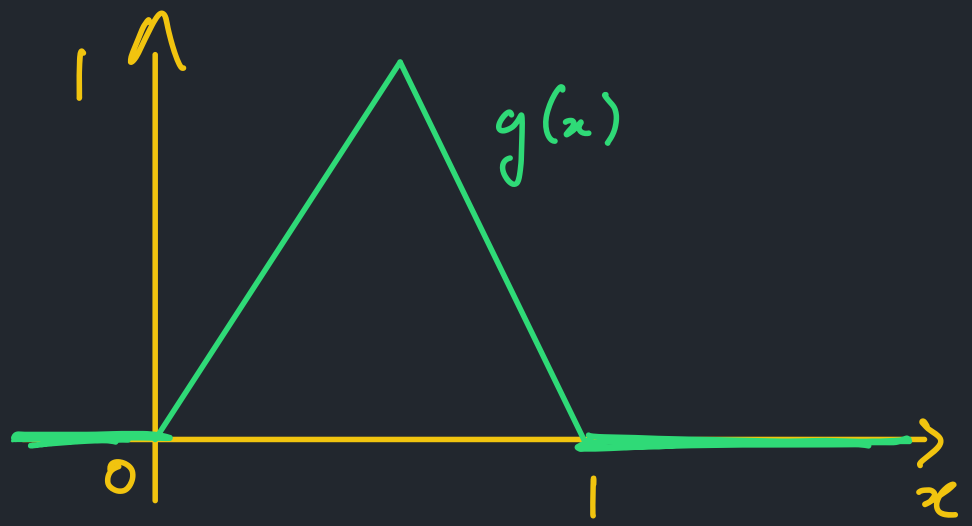
L = depth

$\Leftrightarrow$ upper bound grows at best polynomially in width but exponentially in depth

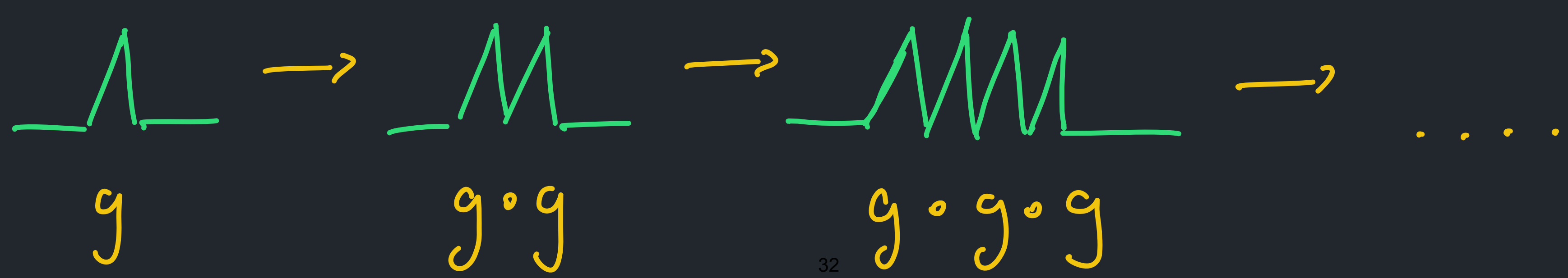
Need to ask: is the bound ever attained?

Short answer: Yes!

Define $g(x) = \mathrm{relu}\left[2 \cdot \mathrm{relu}(x) - 4 \cdot \mathrm{relu}\left(x - \frac{1}{2}\right)\right]$



Each time we "iterate" g — i.e. compose with self — we double the number of linear regions



⇒ Depth separation

- $g \circ g \circ \dots \circ g$ 500 times has $2^{500} - 1$ kinks
- it has 1000 layers of width 2
- to get the same number of kinks with a 3-layer network we would need a width n of

$$\text{KINKS} \leq (2n)^2$$

$$\Rightarrow \quad n \geq \frac{1}{2} \cdot \text{KINKS}^{1/2}$$

$$= \frac{1}{2} \cdot (2^{500} - 1)^{1/3}$$

$$= 7 \times 10^{49} \text{ units}$$

What does this not say?

It does not mean that :

- very deep networks are easy to train
  OPTIMISATION
- very deep networks would generalise well
  GENERALISATION

In our machine learning puzzle, it only tells us something about APPROXIMATION!

Further reading

- Telgarsky (2015, 2016)
  - our depth separation
- Safran and Shamir (2017)
  - a different depth separation
- Lu, Pu, Wang, Hu and Wang (2017)
  - results on "minimum width" needed to be a universal function approximator even at "large depth"
  - relates to rank of the weight matrices

Practical considerations

Let's consider the whole puzzle

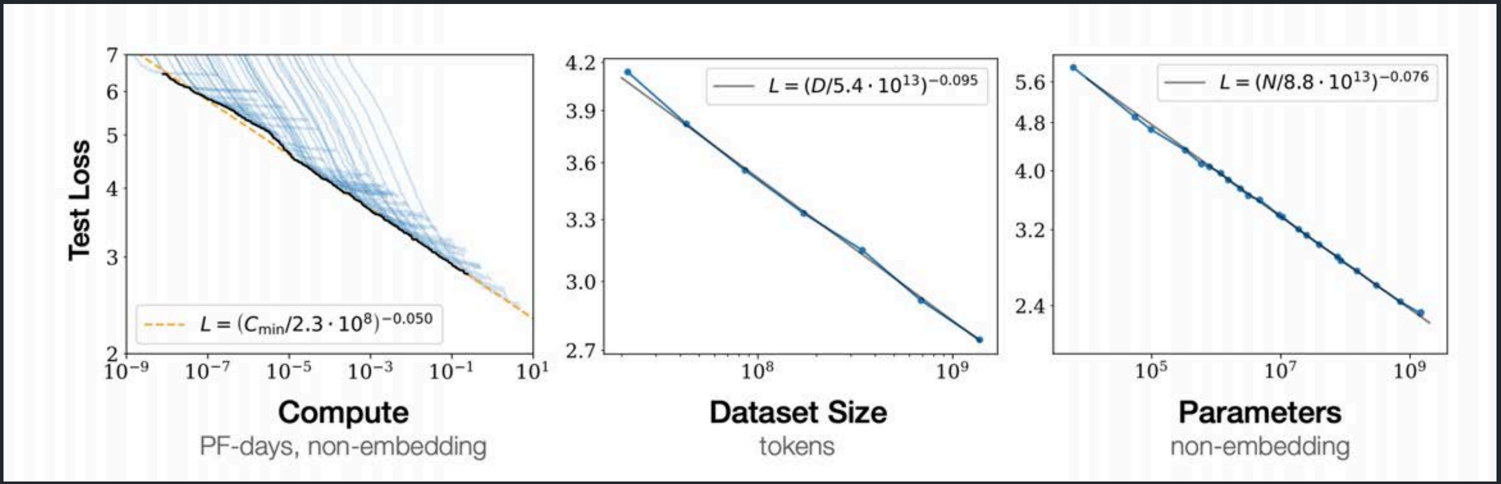
APPROXIMATION

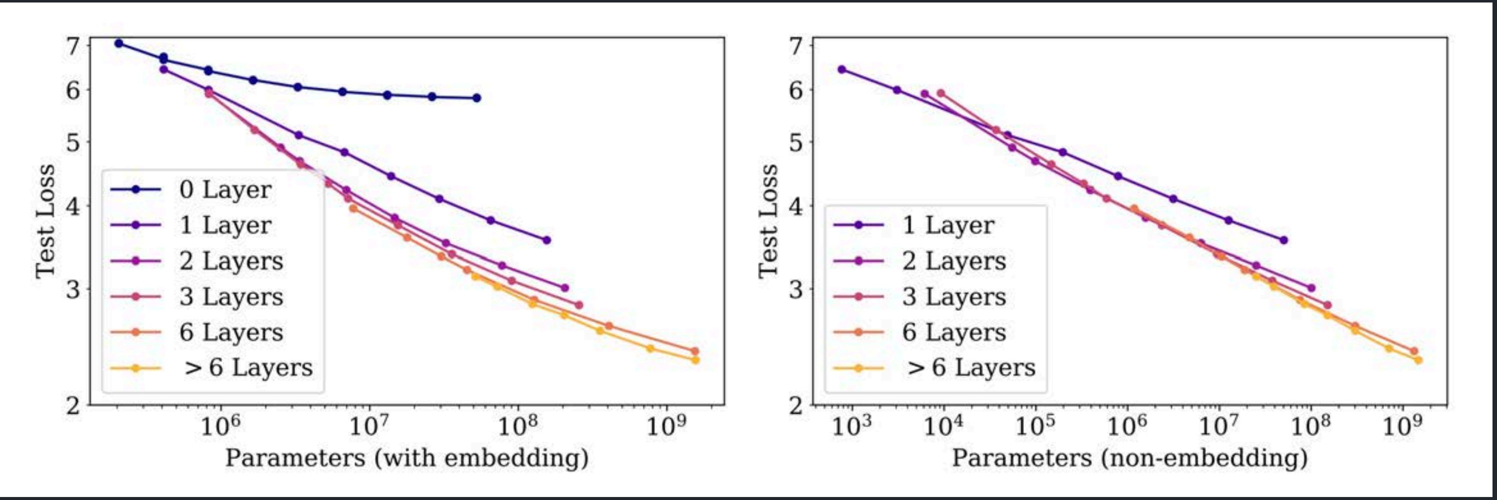
+ OPTIMISATION

+ GENERALISATION

- Pretend you work at an LLM startup
- You want to train the most efficient LLM possible
- You really care about the optimal width vs. depth

Scaling laws





From Kaplan, McCandlish et al (2020)

The importance of confounders

"Chinchilla scaling rules"

Hoffmann, Borgeau, Mensch et al (2020)

- question some results in Kaplan et al
- suggest a different learning rate schedule
- changes certain scaling results

→ to really answer

"what is the optimal width versus depth"

need to obsess over "minor details" of the training pipeline

Wrapping up

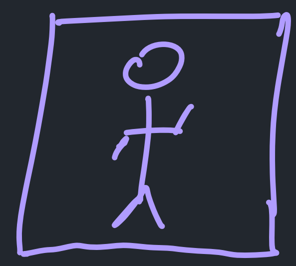
Summary

- Very wide shallow neural nets are universal function approximators
- Deeper networks can fit certain kinds of function with many fewer neurons —— "depth separations"
- Unclear how these results interact with training and generalisation

Preview: Inductive biases

Suppose we want to solve two different machine learning problems

"hello"



"human"

In both cases, we could flatten the inputs into vectors and just apply an MLP...

↪ Hey, it's a universal function approximator.

But, is this a good idea?