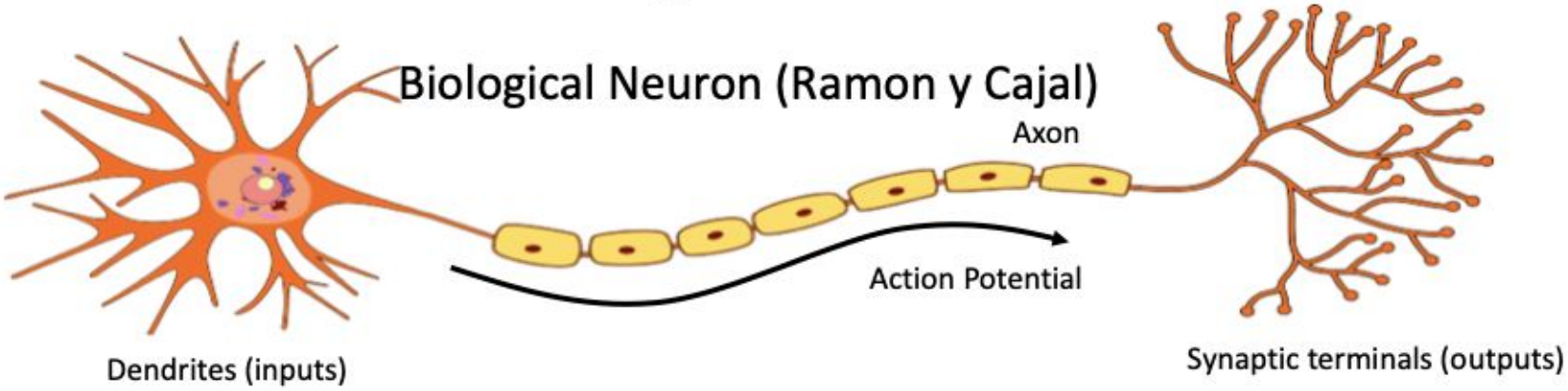
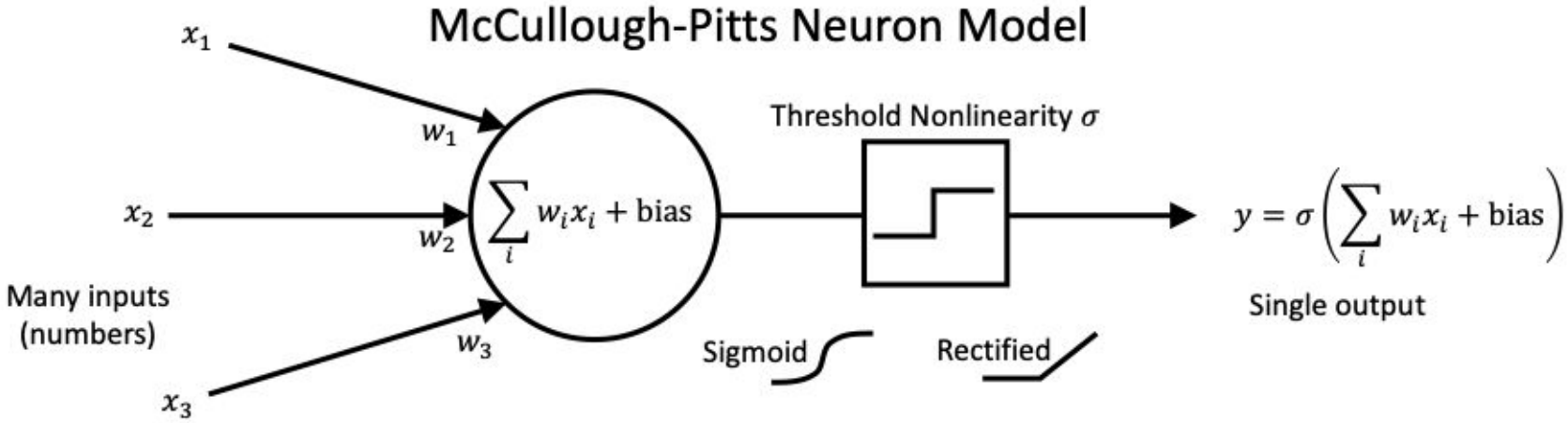
Kolmogorov-Arnold Networks and Implicit Neural Representations (INRs)

Ben Wyant

McCullough-Pitts neuron



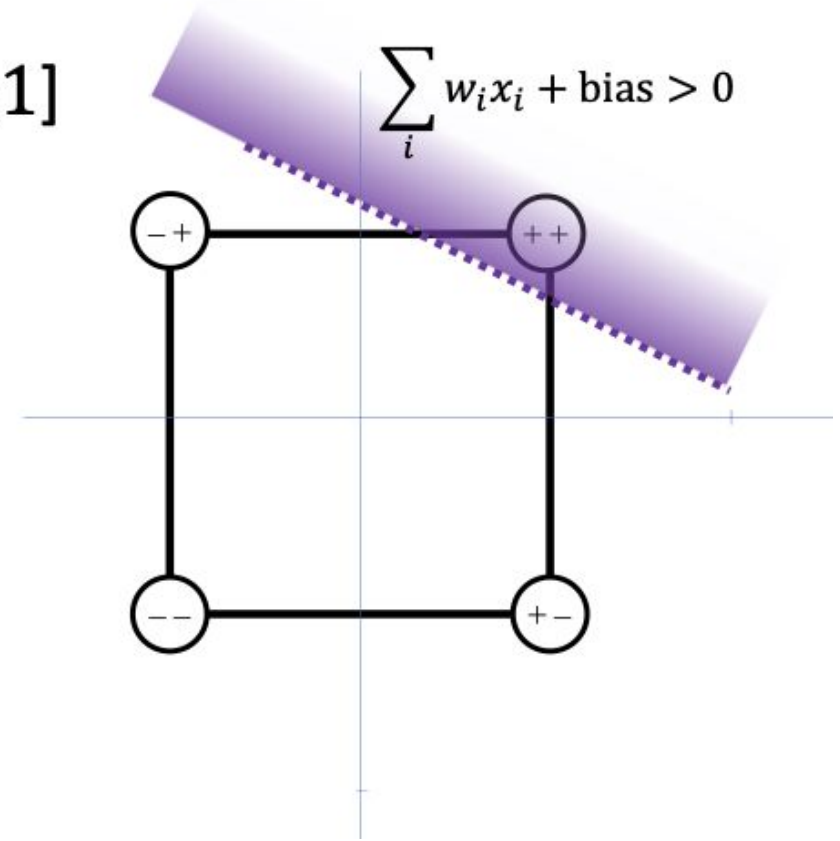


Slide from David Bau

What logic can be learned by one neuron?

$$x_a, x_b = [\pm 1, \pm 1]$$

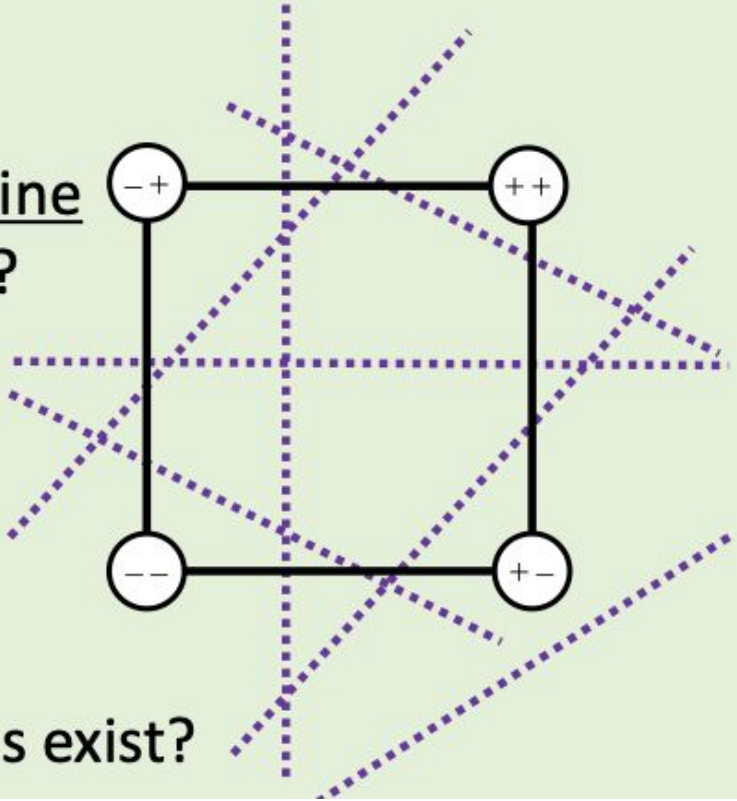
a	b	out
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1



Slide from David Bau

What logic can be learned by one neuron?

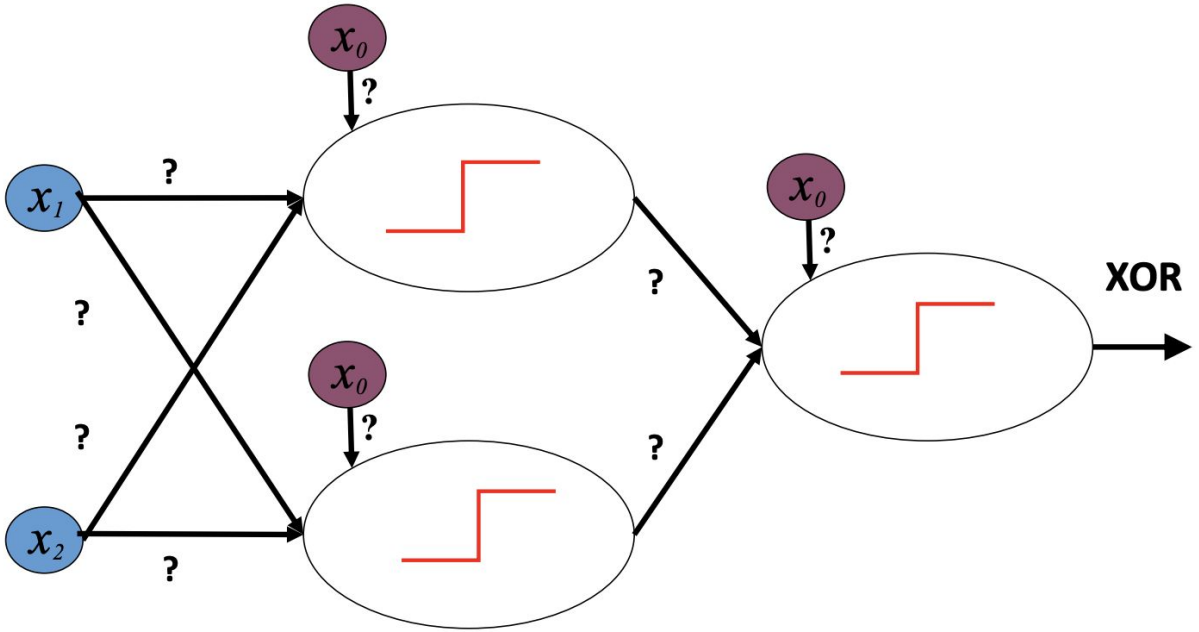
How many ways can a straight line partition corners of a square?



How many d-variable truth-tables exist?

Slide from David Bau

Combining perceptrons can make any Boolean function …if you can set the weights & connections right

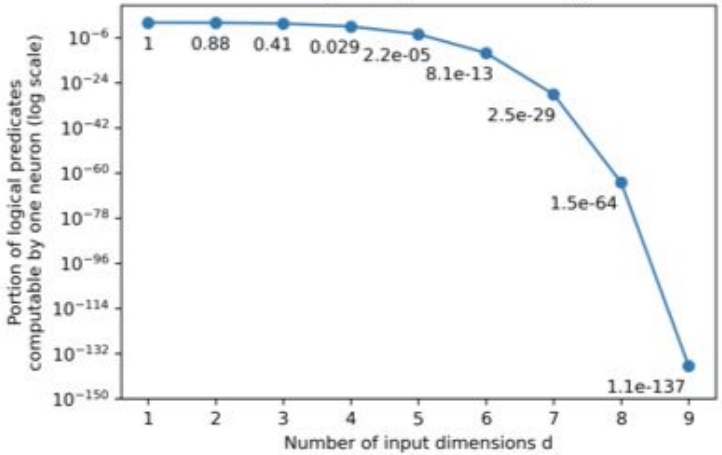


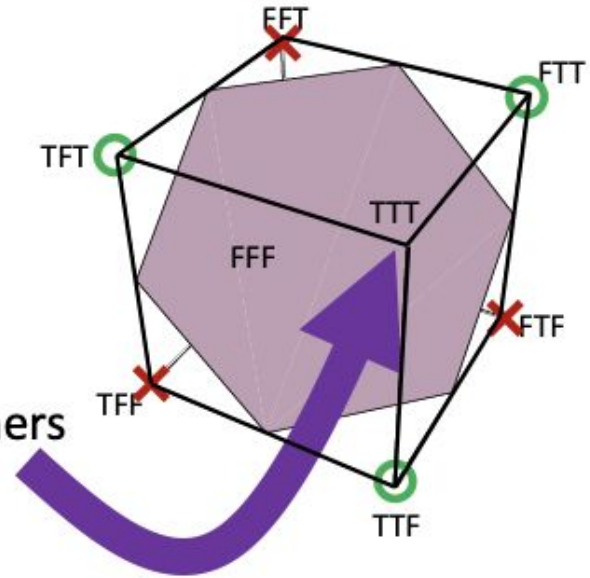
How would you set the weights and connections to make XOR?

Slide from Bryan Pardo

Not all d-variable truth tables are linearly separable

Portion of linearly separable predicates





In a linear model, some corners determine other corners

In d=2, 88% of predicates are linearly separable.
In d=3, 41% of predicates are linearly separable.
In d=4 it is only 2.9%.

[counting for d>10 remains unsolved!]

Slide from David Bau

Puzzle

How many layers of neurons are needed to be able to express all truth tables on d inputs?

Universal Approximation Theorem

Given:

any continuous function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$,
a compact input domain $X \subset \mathbb{R}^n$
and an error interval $\varepsilon > 0$

There is guaranteed to be:

a finite network of neurons g: $\mathbb{R}^n \rightarrow \mathbb{R}^m$
that approximates $|g(x) - f(x)| < \varepsilon$ for all $x \in X$

Cybenko, 1989 proves this for sigmoid/step neurons

Hornik, 1991 proves almost any nonlinearity will work

Slide from David Bau

Visual Proof (Figures from Andrea Lörke)

Two neurons (step-up, step-down) can be summed to make a bump

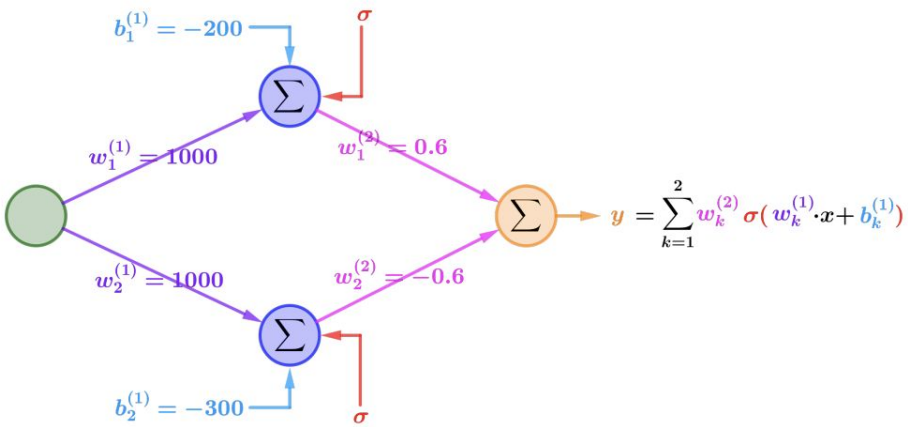


Figure 15: Neural network with 2 hidden nodes

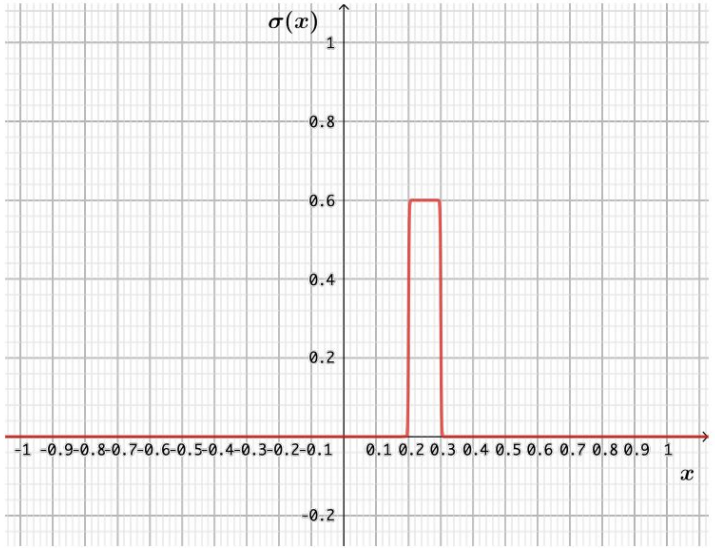


Figure 16: Coupled nodes

https://www.mathematik.uni-wuerzburg.de/fileadmin/10040900/2019/Seminar__Artificial_Neural_Network__24_9__.pdf

Visual Proof (Figures from Andrea Lörke)

Many pairs of neurons can be summed up to sum up bumps

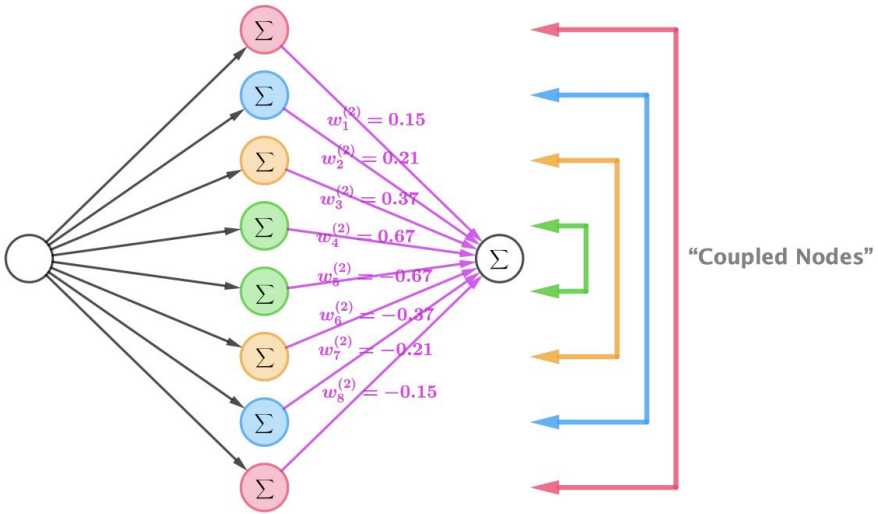


Figure 17: neural network with coupled nodes

Figure 18: Approximation (N=4)

https://www.mathematik.uni-wuerzburg.de/fileadmin/10040900/2019/Seminar__Artificial_Neural_Network__24_9__.pdf

Visual Proof (Figures from Andrea Lörke)

By increasing the number of neuron pairs, error can be made < ε

Figure 18: Approximation (N=4)

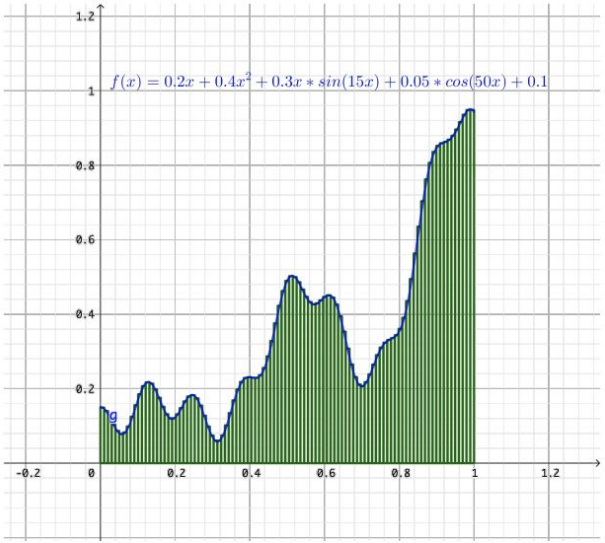


Figure 19: Approximation (N=100)

https://www.mathematik.uni-wuerzburg.de/fileadmin/10040900/2019/Seminar__Artificial_Neural_Network__24_9__.pdf

Visual Proof (Figures from Andrea Lörke)

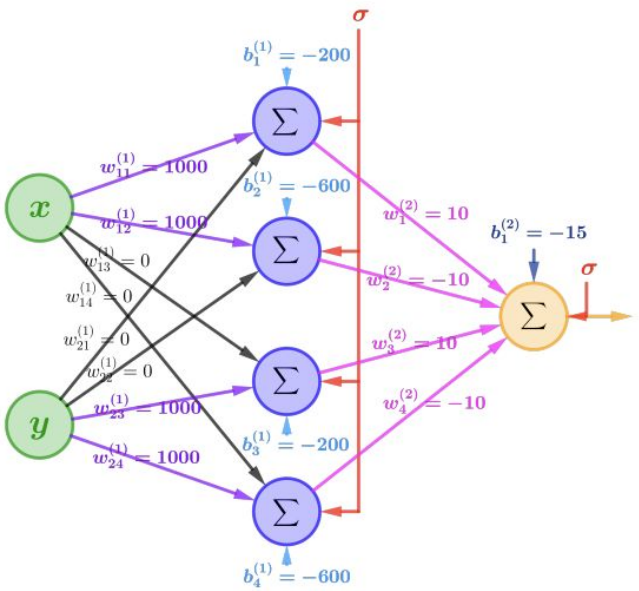


Figure 32: Nodes to compute a Tower-function at any Position

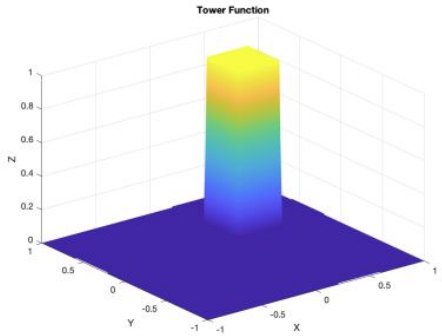


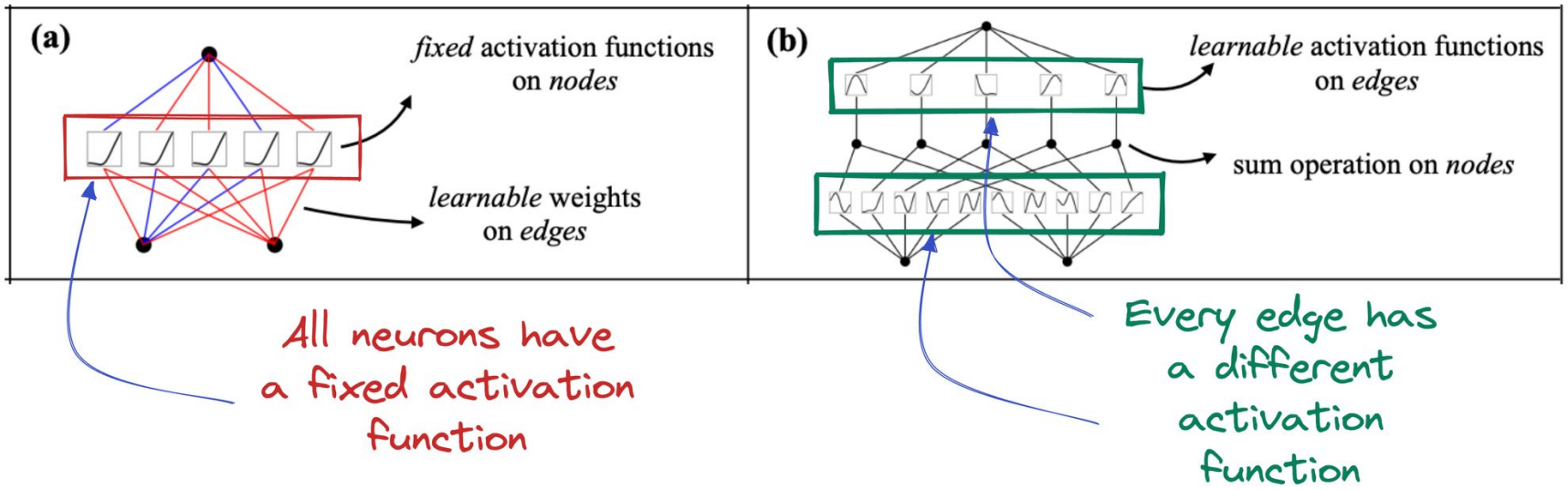
Figure 33: "Tower"-function

https://www.mathematik.uni-wuerzburg.de/fileadmin/10040900/2019/Seminar__Artificial_Neural_Network__24_9__.pdf

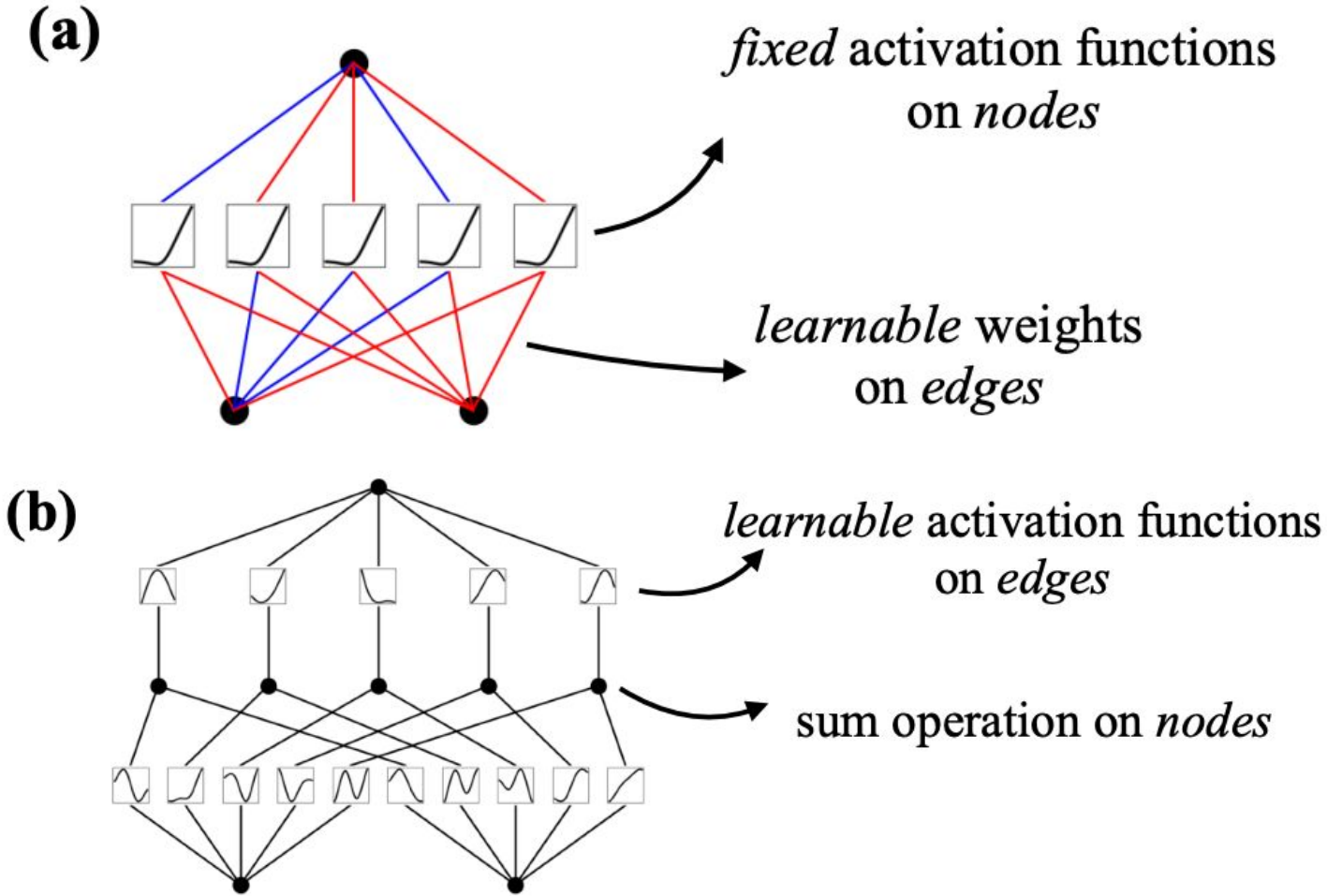
Kolmogorov Arnold Network (KAN)

Neural Network

KAN



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Kolmogorov-Arnold Representation Theorem

- Any multivariate continuous function can be represented as the composition of a **finite** number of continuous functions of a single variable

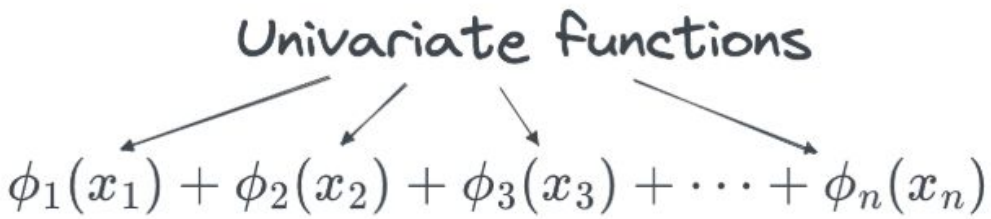
$$\sum_{j=1}^{m} \psi_j \left(\sum_{i=1}^{n} \phi_{ij}(x_i) \right)$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

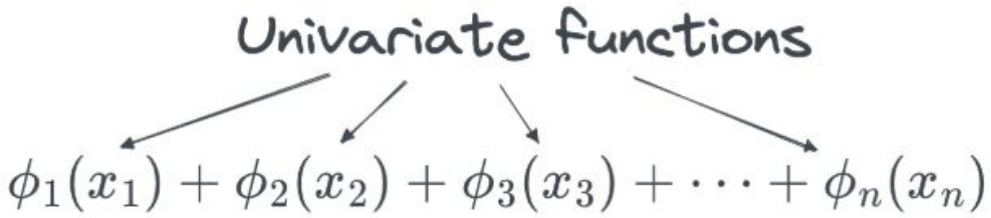
- “Multivariate continuous function” is a function that accepts multiple parameters:

$$y = F(x_1, x_2, x_3, \cdots, x_n)$$

- “A finite number of continuous functions of a single variable” means:



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/



- The above sum is passed through one more functions ψ and then a composition is applied

$$\psi\left(\sum_{i=1}^{n} \phi_i(x_i)\right) \qquad \sum_{j=1}^{m} \psi_j\left(\sum_{i=1}^{n} \phi_{ij}(x_i)\right)$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Multivariate continuous function

Composition of univariate functions

$$F(x_1, x_2, x_3, \cdots, x_n) = \sum_{j=1}^{m} \psi_j \left(\sum_{i=1}^{n} \phi_{ij}(x_i) \right)$$

Multivariate continuous function

Composition of univariate functions

$$\begin{aligned} F(x_1, x_2, x_3, \cdots, x_n) = {} & \psi_1(\phi_{11}(x_1) + \phi_{21}(x_2) + \cdots + \phi_{n1}(x_n)) \\ & + \psi_2(\phi_{12}(x_1) + \phi_{22}(x_2) + \cdots + \phi_{n2}(x_n)) \\ & \vdots \\ & + \psi_m(\phi_{1m}(x_1) + \phi_{2m}(x_2) + \cdots + \phi_{nm}(x_n)) \end{aligned}$$

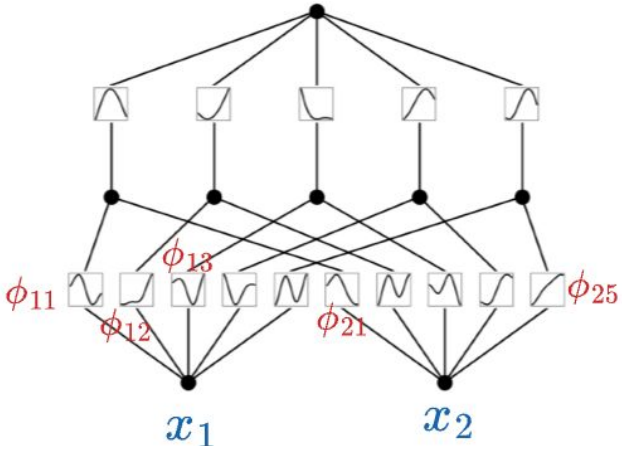
Multivariate continuous function

$$F(x, y) = xy$$

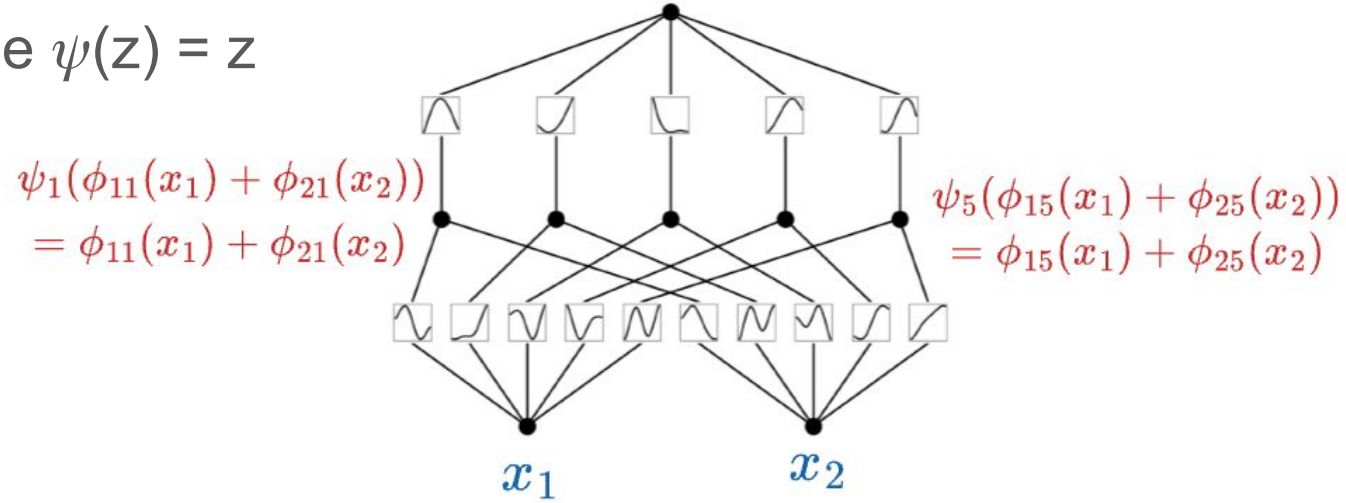
Composition of univariate functions

$$F(x, y) = \underbrace{exp}_{\psi}\left(\underbrace{log(x)}_{\phi_x} + \underbrace{log(y)}_{\phi_y}\right)$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/



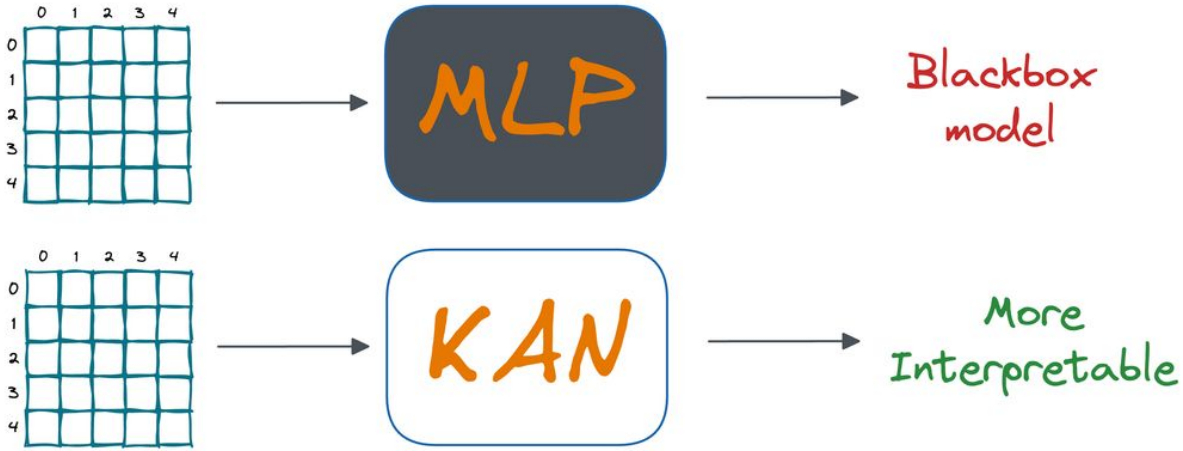
In this case $\psi(z) = z$



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Key Differences

- Universal Approximation Theorem deals with **approximating functions** using neural networks
- Kolmogorov-Arnold theorem provides a way to **exactly represent** continuous functions through sums of univariate functions

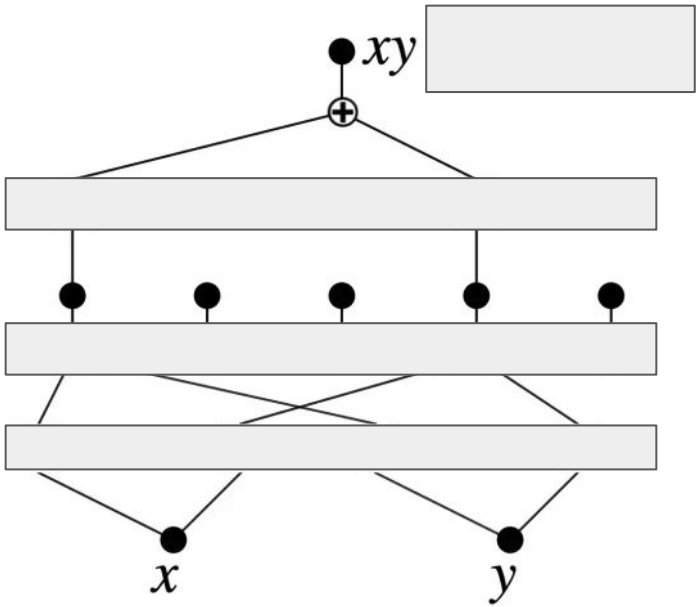


https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Can MLPs do multiplication?

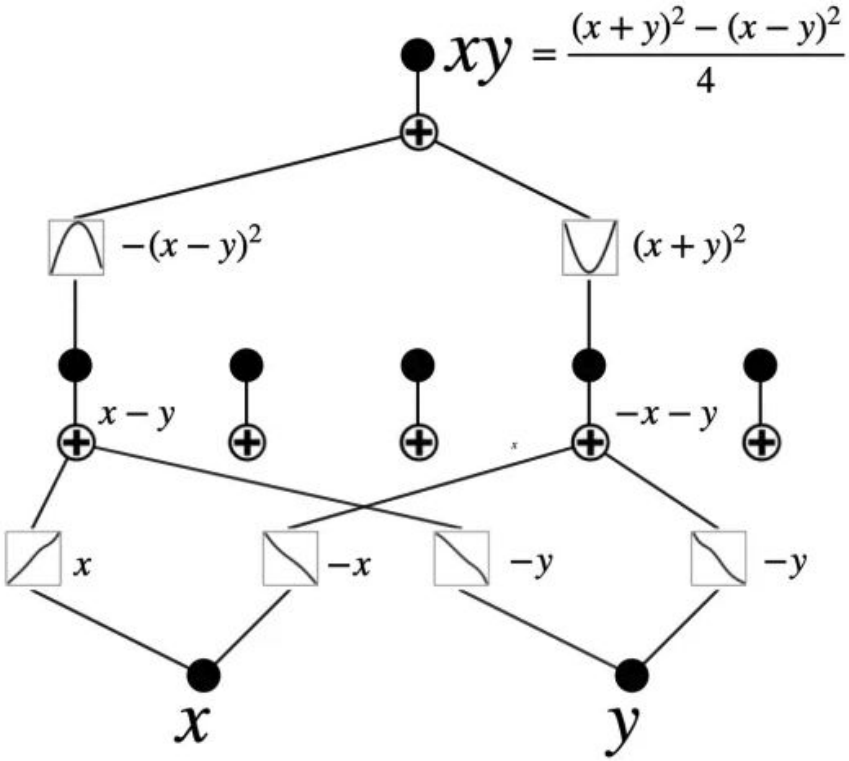
Puzzle

Can you come up with another way to represent xy?



https://medium.com/@volzhinnv/kan-2-0-rival-of-ann-is-getting-stronger-c282feb6717e

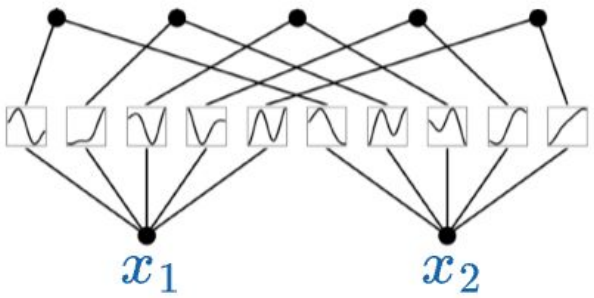
Answer



https://medium.com/@volzhinnv/kan-2-0-rival-of-ann-is-getting-stronger-c282feb6717e

KAN matrix representation

- A single KAN layer is taking the input (x_1,x_2, …, x□) and applying a transformation ϕ
- ϕ_1 can be represented with the matrix below where n is the number of inputs and m is the number of outputs to layer l_1



$$\phi^1$$

$$\phi^1 = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn} \end{bmatrix}$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Transformation matrix ϕ^1 Input X

Output of first layer

$$z^1 = \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{12} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{m1} & \phi_{m2} & \cdots & \phi_{mn} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Function applied to x_1 Function applied to x_n

$$z^1 = \begin{bmatrix} \phi_{11}(x_1) + \phi_{12}(x_2) + \cdots + \phi_{1n}(x_n) \\ \phi_{21}(x_1) + \phi_{22}(x_2) + \cdots + \phi_{2n}(x_n) \\ \vdots \\ \phi_{m1}(x_1) + \phi_{m2}(x_2) + \cdots + \phi_{mn}(x_n) \end{bmatrix}$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

KAN Formula Representation

- The entire KAN network can be condensed into one formula:

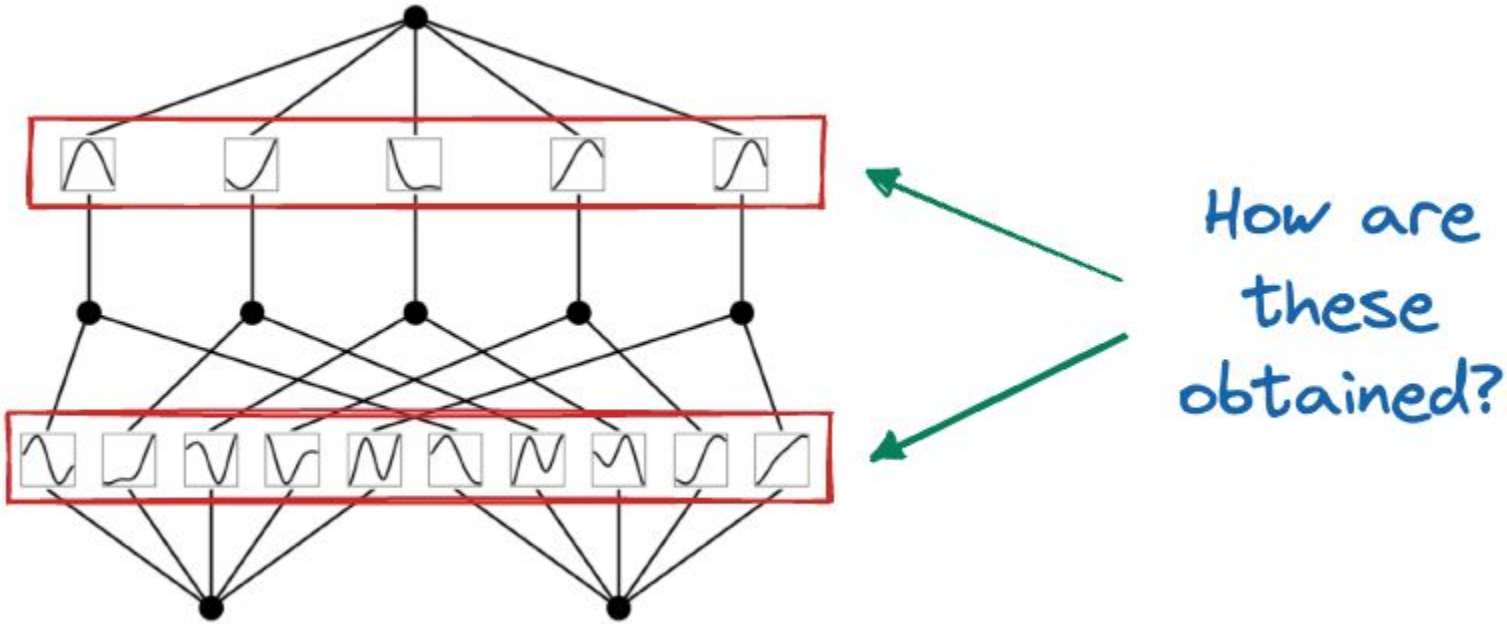
$$KAN(x) = \phi^{L}\left(\phi^{L-1}\left(\ \ldots\ \left(\phi^{2}\left(\phi^{1}(x)\right)\right)\right)\right)$$

Where:

- x is the input vector
- ϕ^{k} is the function transformation matrix of layer k
- KAN(x) is the output of the KAN

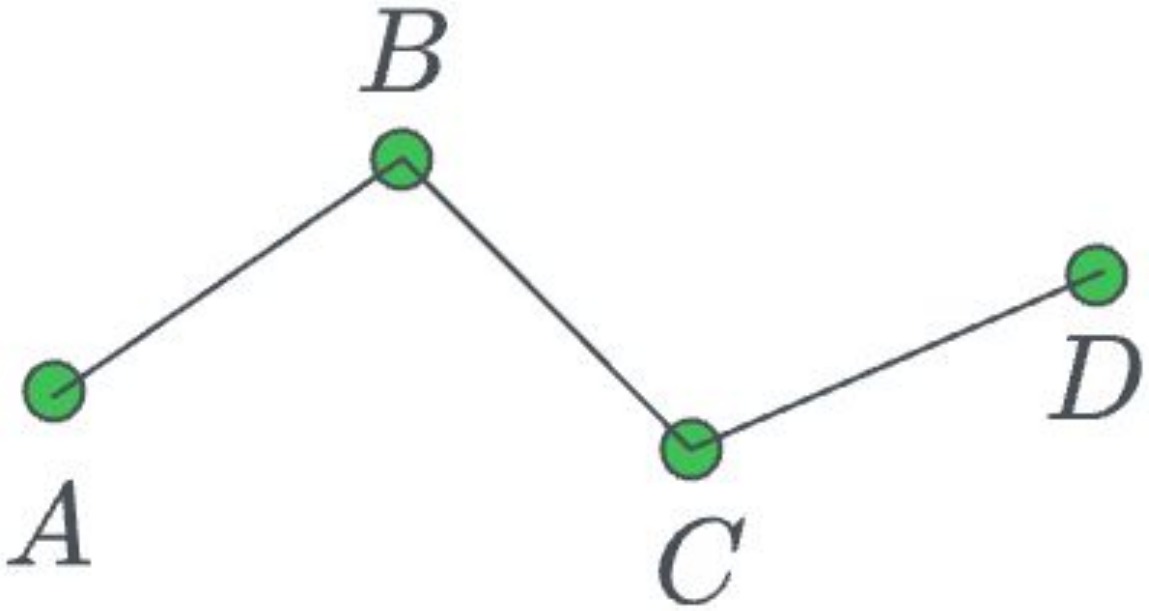
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

How are KANs trained?



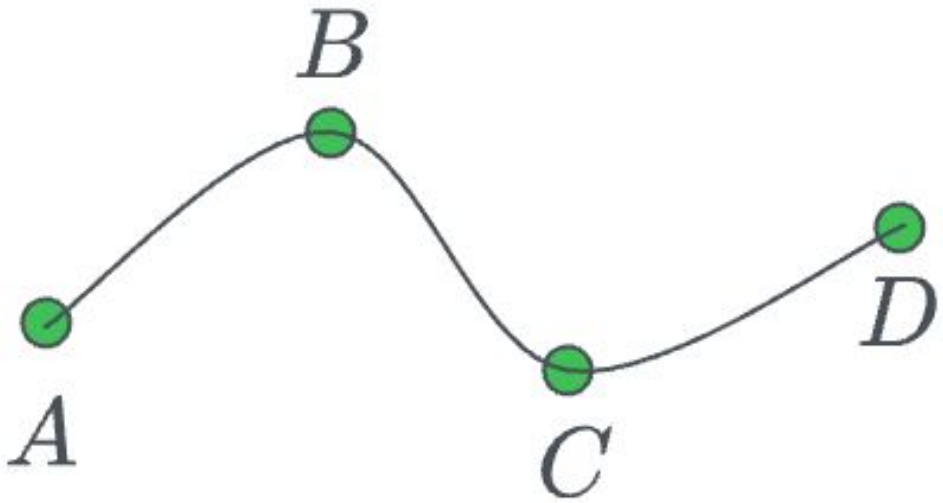
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Let’s talk about splines



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

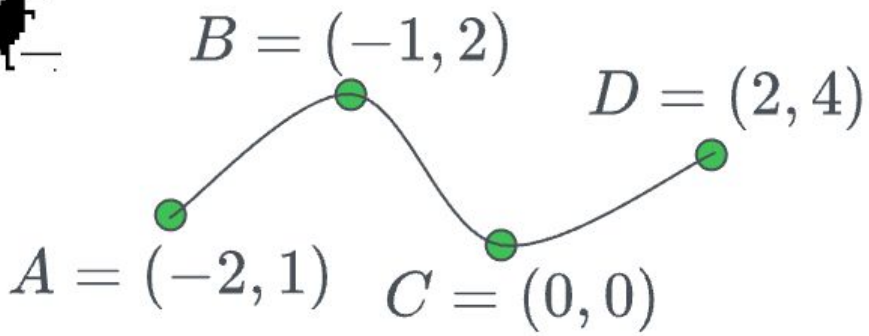
Bezier Curves



$$f(x) = ax^3 + bx^2 + cx + d$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Character



Solve these equations

$$A \to 1 = a(-2)^3 + b(-2)^2 + c(-2) + d$$

$$B \to 2 = a(-1)^3 + b(-1)^2 + c(-1) + d$$

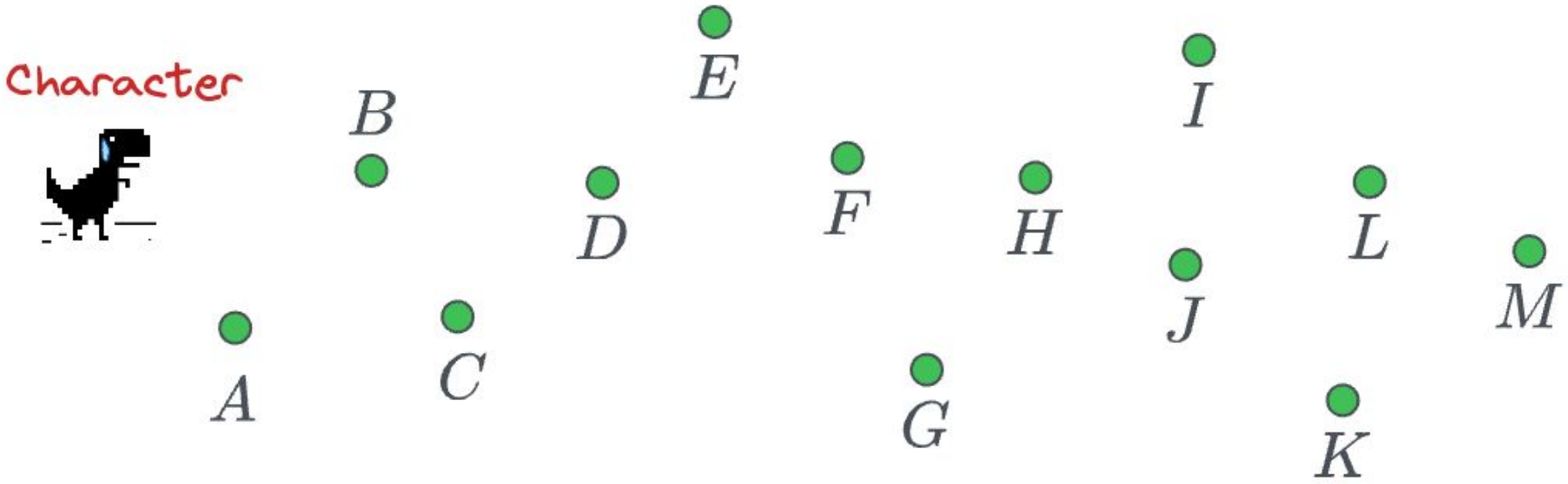
$$C \to 0 = a(0)^3 + b(0)^2 + c(0) + d$$

$$D \to 4 = a(2)^3 + b(2)^2 + c(2) + d$$

- We need 4 points to solve for an equation with 4 variables

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

What do we do if we have hundreds of data points?



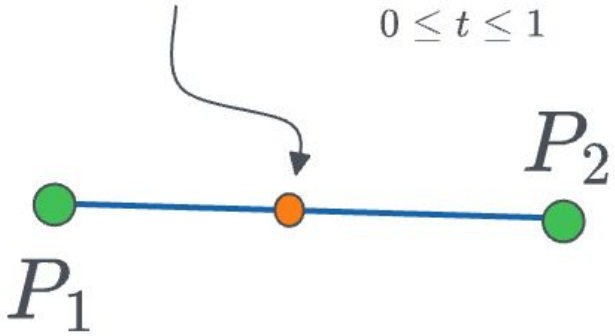
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Bezier Curves

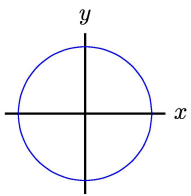
- When $t = 0$, P will be P_1
- When $t = 1$, P will be P_2
- $t \in [0,1]$

$$P = (1-t) \cdot P_1 + t \cdot P_2$$

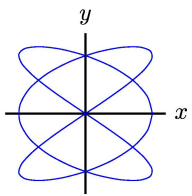
$$0 \le t \le 1$$



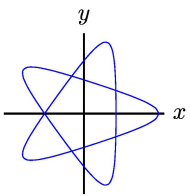
Some other parametric equations:



$x(t) = \cos(t)$
$y(t) = \sin(t)$



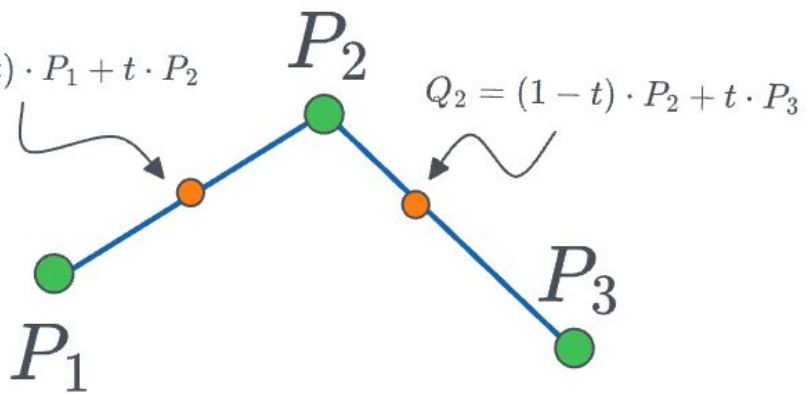
$x(t) = 3\cos(t)$
$y(t) = 2\sin(t)$



$x(t) = 2\cos(x) + 5\cos(\frac{2}{3}x)$
$y(t) = 2\sin(x) + 5\sin(\frac{2}{3}x)$

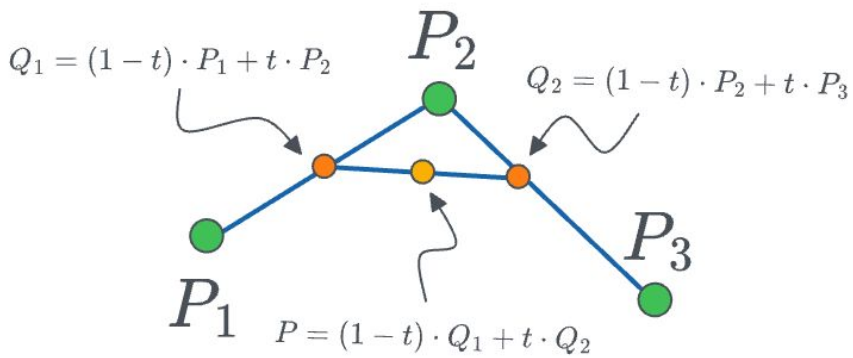
$$Q_1 = (1-t) \cdot P_1 + t \cdot P_2$$

$$Q_2 = (1-t) \cdot P_2 + t \cdot P_3$$

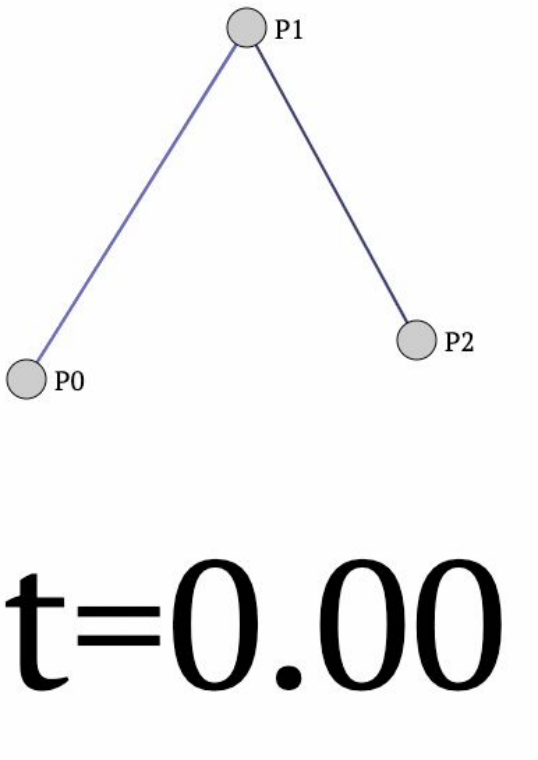


https://mathbooks.unl.edu/Calculus/sec-3-7-parametric.html
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Between points Q_1 and Q_2 we can create a linear interpolation resulting in the output curve



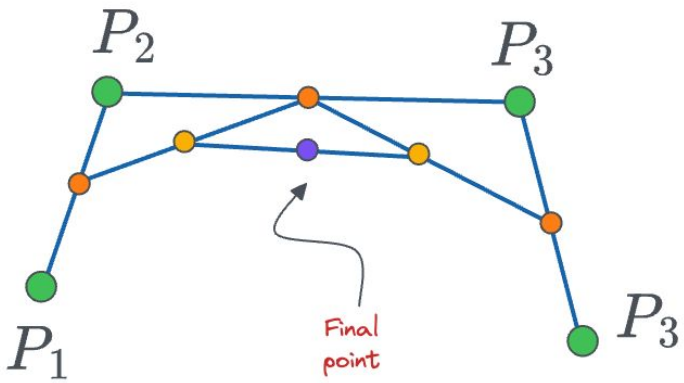
$$B(t) = (1-t)^2 P_1 + 2(1-t)tP_2 + t^2 P_3,\ t \in [0,1]$$



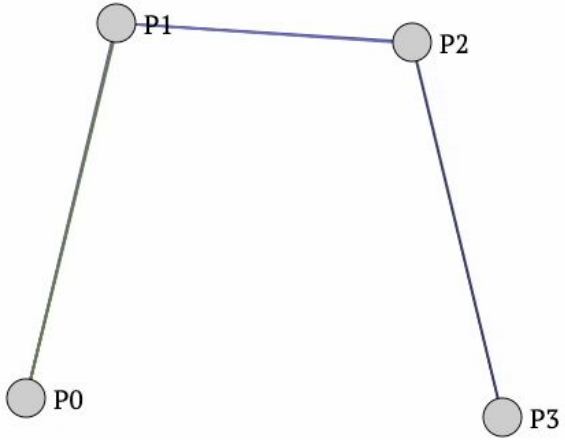
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Cubic Bezier Curve

We can extend this same idea to 4 points which results in this new output curve



$$B(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t) t^2 P_3 + t^3 P_4, \; t \in [0,1]$$

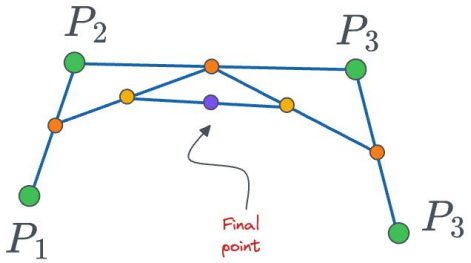


t=0.00

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Bezier Curve Recap

These coefficients match with the binomial expansion of the polynomial $(1+x)^n$



$$B(t) = \boxed{(1-t)^3}P_1 + \boxed{3(1-t)^2 t}P_2 + \boxed{3(1-t)t^2}P_3 + \boxed{t^3}P_4,\ t \in [0,1]$$

2 points

$$B(t) = (1-t)P_1 + tP_2,\ t \in [0,1]$$

3 points

$$B(t) = (1-t)^2 P_1 + 2(1-t)tP_2 + t^2 P_3,\ t \in [0,1]$$

4 points

$$B(t) = (1-t)^3 P_1 + 3(1-t)^2 tP_2 + 3(1-t)t^2 P_3 + t^3 P_4,\ t \in [0,1]$$

$$B(t) = \sum_{i=0}^{n} \boxed{\binom{n}{i}(1-t)^{n-i}\, t^i} P_i \ ; \ \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

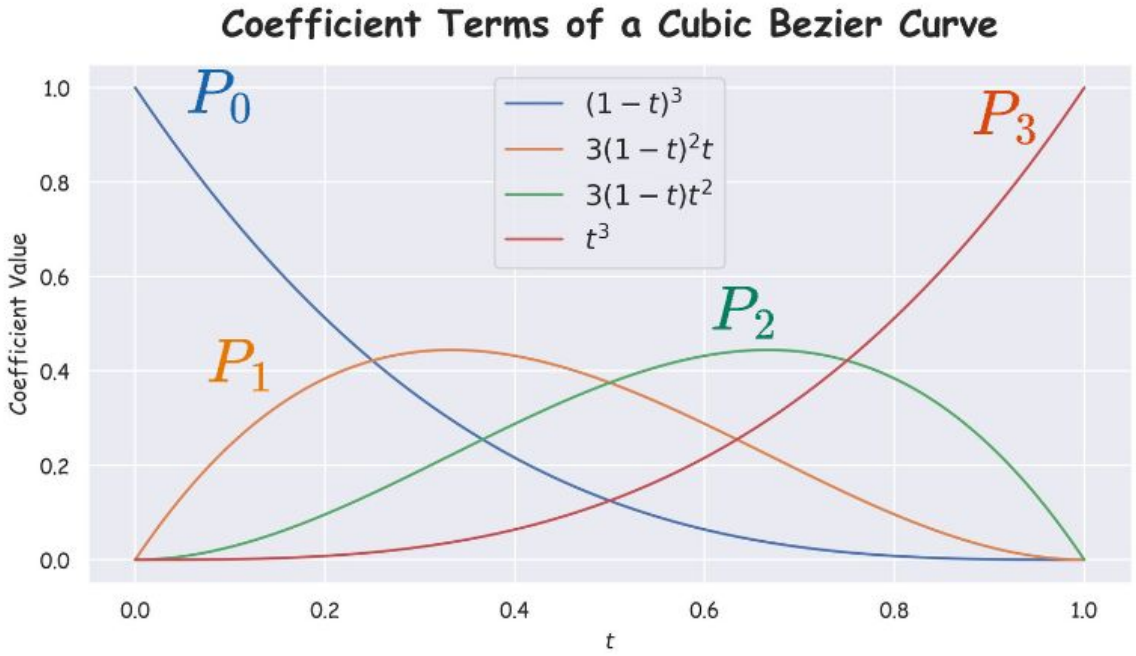
Binomial coefficient

$$= \sum_{i=0}^{n} c_{i,n}(t)\ P_i$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Plotting the Bezier Curve

- At $t = 0$, except P_0 the coefs of all points are 0 so the curve starts from P_0
- From $t \in (0.25, 0.5)$, P_1 is max, the curve is closest to P_1 in that duration
- From $t \in (0.5, 0.75)$, P_2 is max, the curve is closest to P_2 in that duration
- At $t = 1$, except P_3, coefs of all points are 0 so the curve ends at P_3



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Problem with Bezier Curves

- Having 100 data points will result in a polynomial of degree 99, which will be computationally expensive
- Factorial terms are going to be expensive as well

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i}\, t^{i}\, P_i \;\; ; \;\; \binom{n}{i} = \frac{n!}{i!(n-i)!}$$

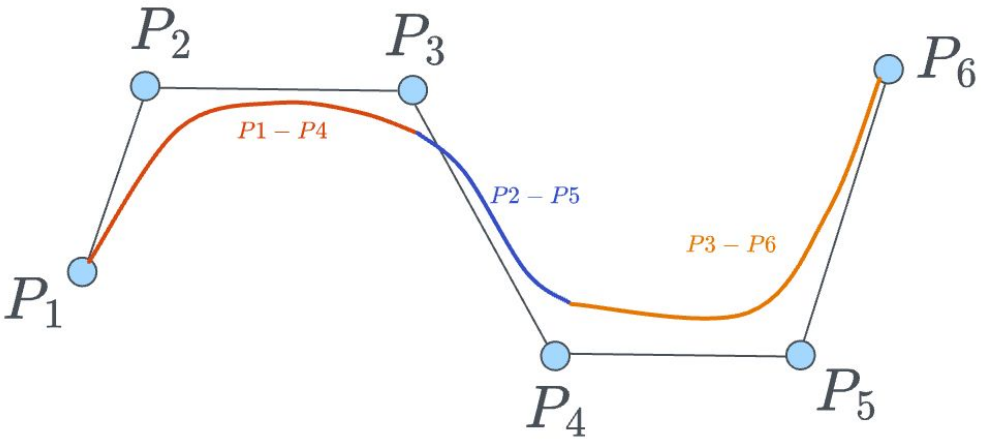
Binomial coefficient

$$= \sum_{i=0}^{n} c_{i,n}(t)\; P_i$$

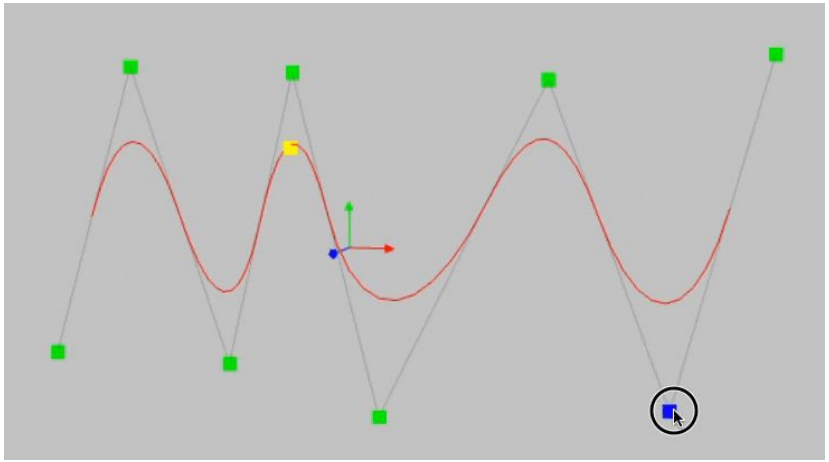
https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

B-Splines

With 6 points we could generate a degree 5 Bezier curve - this is still expensive



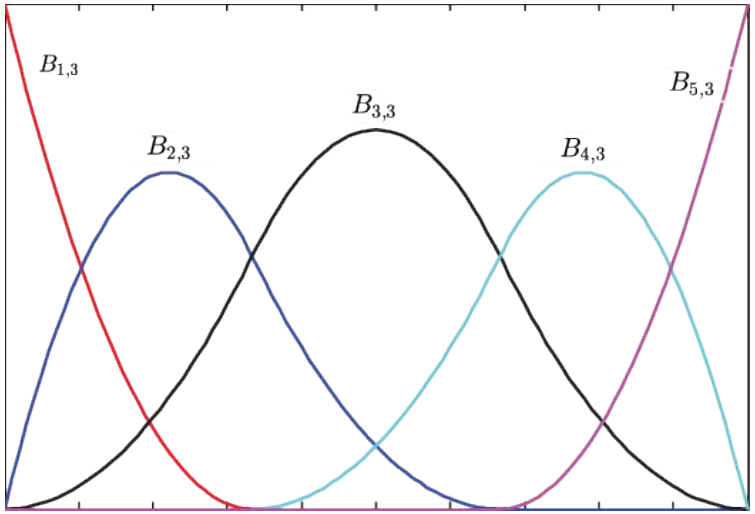
Instead, we can create curves of smaller degrees, say 3, and connect them



When we have n control points and create k degree polynomial Bezier curves we get (n-k) Bezier curves in the final B-spline

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

How do we connect these curves?



We must ensure C^2 continuity

- The curvature is smooth across the joins (also called knots)
- P_i : Control points that define the shape of the curve
- $N_{i,k}(t)$: B-spline basis functions of degree k associated with each control point P_i

$$S(t) = \sum_{i=0}^{n} P_i \cdot N_{i,k}$$

control points

Basis function

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/
https://www.researchgate.net/figure/Uniform-B-spline-basis-functions-for-N-5-k-3-and-X-0001323111_fig2_282803553

Building a KAN layer

By adjusting the positions of the control points during training, the KAN model can dynamically shape the activation functions that best fit the data

- During training, update the positions of these control points through backprop
- Use gradient descent to adjust the control points

$$\phi(x) = w(\ b(x) \ + \ spline(x)\ )$$

- $b(x)$ is a basis function which acts as residual connection
- $b(x)$ helps mitigate the vanishing gradient problem
- $\text{spline}(x)$ is learnable, specifically, the points c_i
- In principle, w is redundant, but it is included because it helps to better control magnitudes of the activation function

learnable fixed learnable

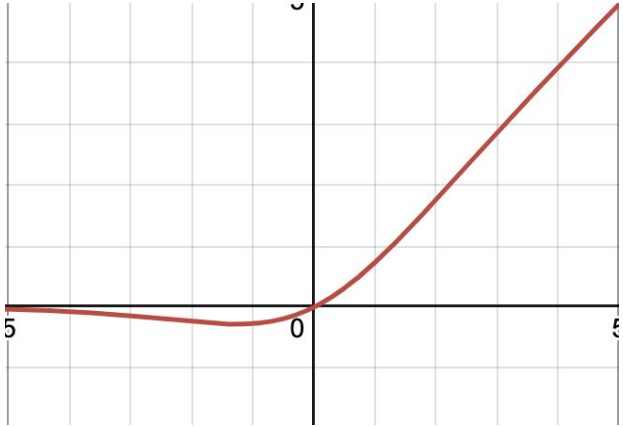
$$\phi(x) = w\,(b(x) + spline(x))$$

$$b(x) = \frac{x}{1 + e^{-x}}$$

$$spline(x) = \sum_i c_i\, B_i(x)$$

learnable

$$b(x) = silu(x)$$



https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Initializing the weights

$$\phi(x) = w_b b(x) + w_s \text{spline}(x)$$

$$\text{spline}(x) \approx 0 \qquad w_s = 1$$

$$c_i \sim \mathcal{N}(0, \sigma^2)$$

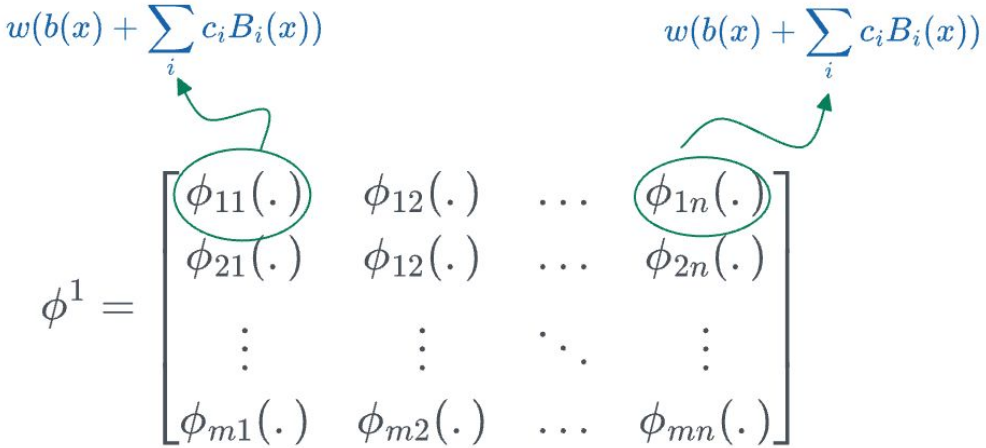
small value

$$w_b \sim N\left(0, \frac{2}{n_{\text{in}} + n_{\text{out}}}\right)$$

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

Now we can train a network

- Initialize each of the ϕ matrices:



- Run the forward pass

$$KAN(x) = \phi^L \left(\phi^{L-1} \left(\dots \left(\phi^2 \left(\phi^1(x) \right) \right) \right) \right)$$

- Calculate loss and run backpropagation

https://www.dailydoseofds.com/a-beginner-friendly-introduction-to-kolmogorov-arnold-networks-kan/

KAN Simplification techniques - Sparsification

For MLPs, L1 regularization of linear weights is used to favor sparsity. KANs can adapt this high-level idea, but need two modifications:

1. There is no linear “weight” in KANs. Linear weights are replaced by learnable activation functions, so we should define the L1 norm of these activation functions.

2. L1 regularization is insufficient for sparsification of KANs; instead an additional entropy regularization is necessary

https://arxiv.org/pdf/2404.19756

L1 norm for KAN

- L1 norm of an activation function ϕ to be its average magnitude over its N_p inputs

$$|\phi|_1 = \frac{1}{N_p} \sum_{s=1}^{N_p} \left| \phi \left(x^{(s)} \right) \right|$$

- For a KAN layer $\mathbf{\Phi}$ with n_{in} inputs and n_{out} outputs, the L1 norm of $\mathbf{\Phi}$is the sum of L1 norms of all the activation functions:

$$|\mathbf{\Phi}|_1 = \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} |\phi_{i,j}|_1$$

https://arxiv.org/pdf/2404.19756

KAN entropy

- We define the entropy of $\mathbf{\Phi}$ to be:

$$S\left(\mathbf{\Phi}\right) \equiv -\sum_{i=1}^{n_{\text{in}}}\sum_{j=1}^{n_{\text{out}}} \frac{\left|\phi_{i,j}\right|_1}{\left|\mathbf{\Phi}\right|_1} \log\left(\frac{\left|\phi_{i,j}\right|_1}{\left|\mathbf{\Phi}\right|_1}\right)$$

- The total training objective is the prediction loss plus L1 entropy regularization of all the KAN layers:

$$\ell_{\text{total}} = \ell_{\text{pred}} + \lambda\left(\mu_1 \sum_{l=0}^{L-1}\left|\mathbf{\Phi}_l\right|_1 + \mu_2 \sum_{l=0}^{L-1} S\left(\mathbf{\Phi}_l\right)\right)$$

https://arxiv.org/pdf/2404.19756

Pruning

- After training with sparsification penalty, we may want to prune the network
- KANs are pruned on the node level (rather than edge level)

For each node (say the i^{th} neuron in the l^{th} layer, the incoming and outgoing score are defined as:

$$I_{l,i} = \max_k \left(\left| \phi_{l-1,i,k} \right|_1 \right), \qquad O_{l,i} = \max_j \left(\left| \phi_{l+1,j,i} \right|_1 \right)$$

A node is considered to be important if both incoming and outgoing scores are greater than a hyperparameter $\theta = 10^{-2}$ by default

https://arxiv.org/pdf/2404.19756

Symbolic Regression Process

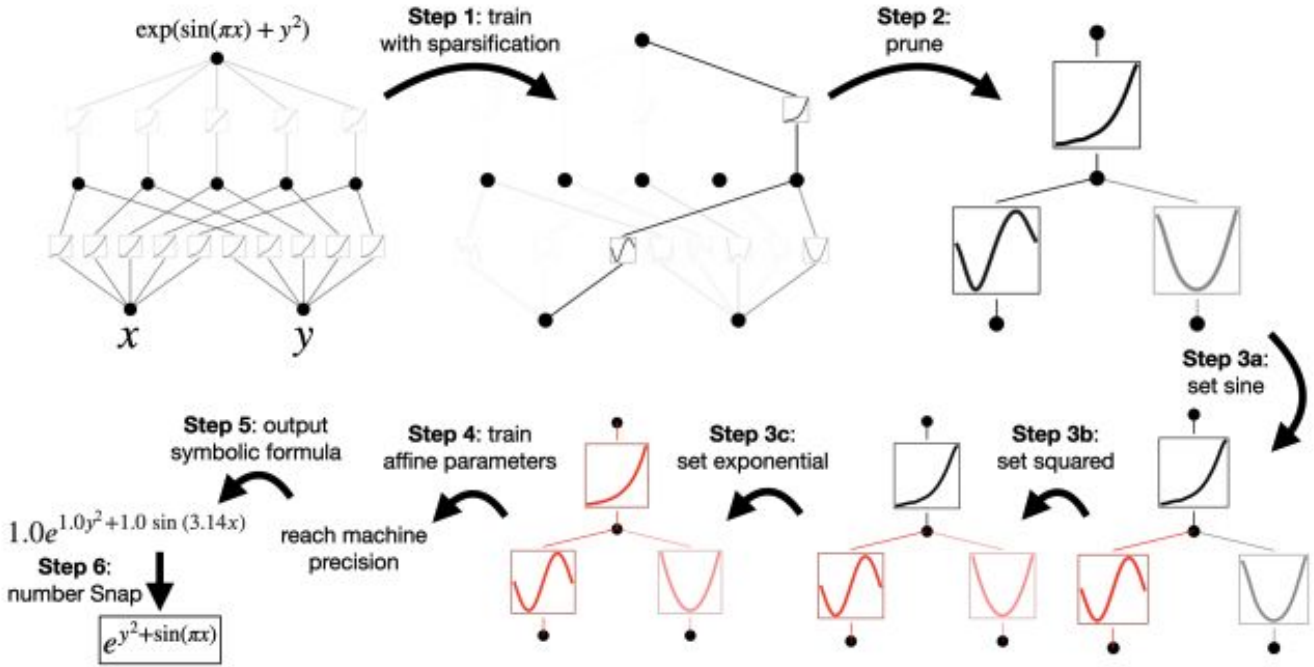
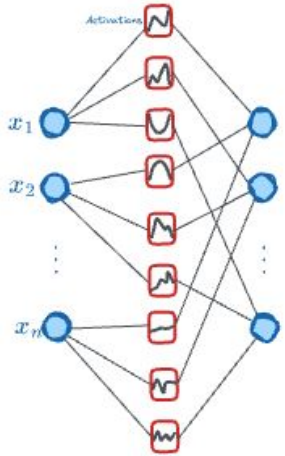


Figure 2.4: An example of how to do symbolic regression with KAN.

https://arxiv.org/pdf/2404.19756

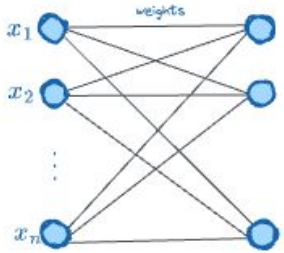
KAN vs MLP parameter count

KAN Layer



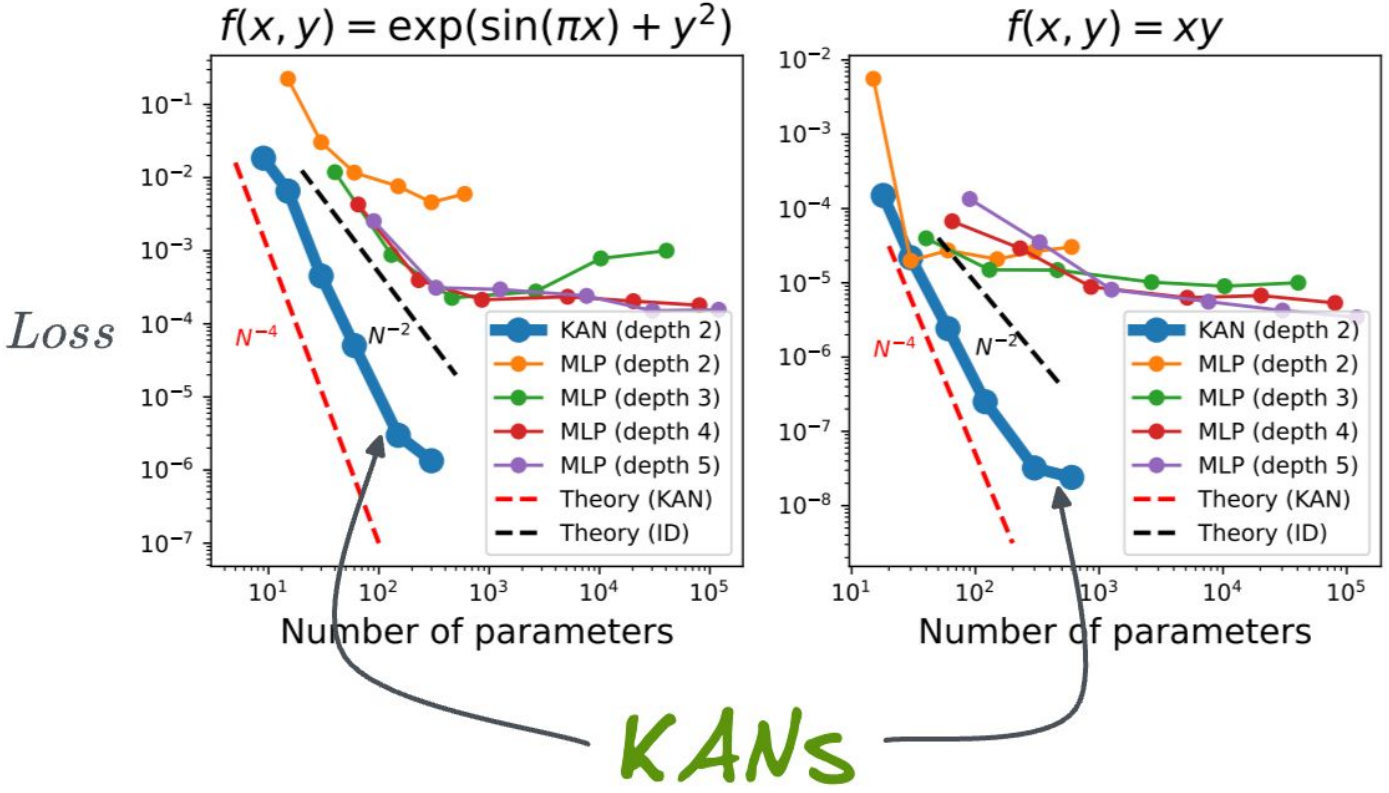
$\sim N^2 L(G+k)$

MLP Layer

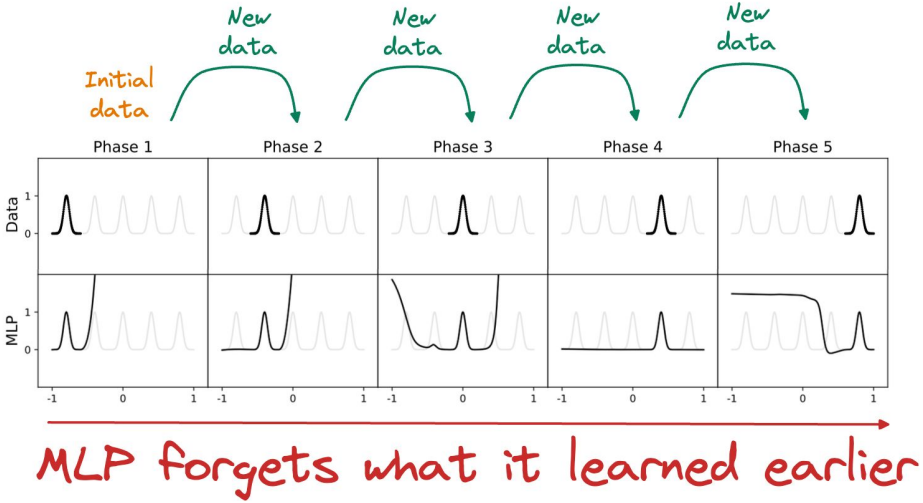


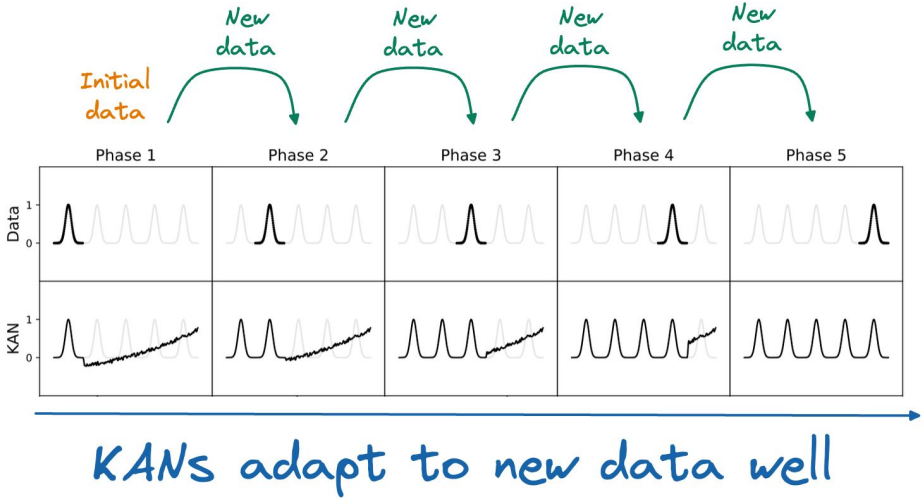
$N^2 L$

KANs are accurate with fewer parameters



Continual Learning





Interpretability

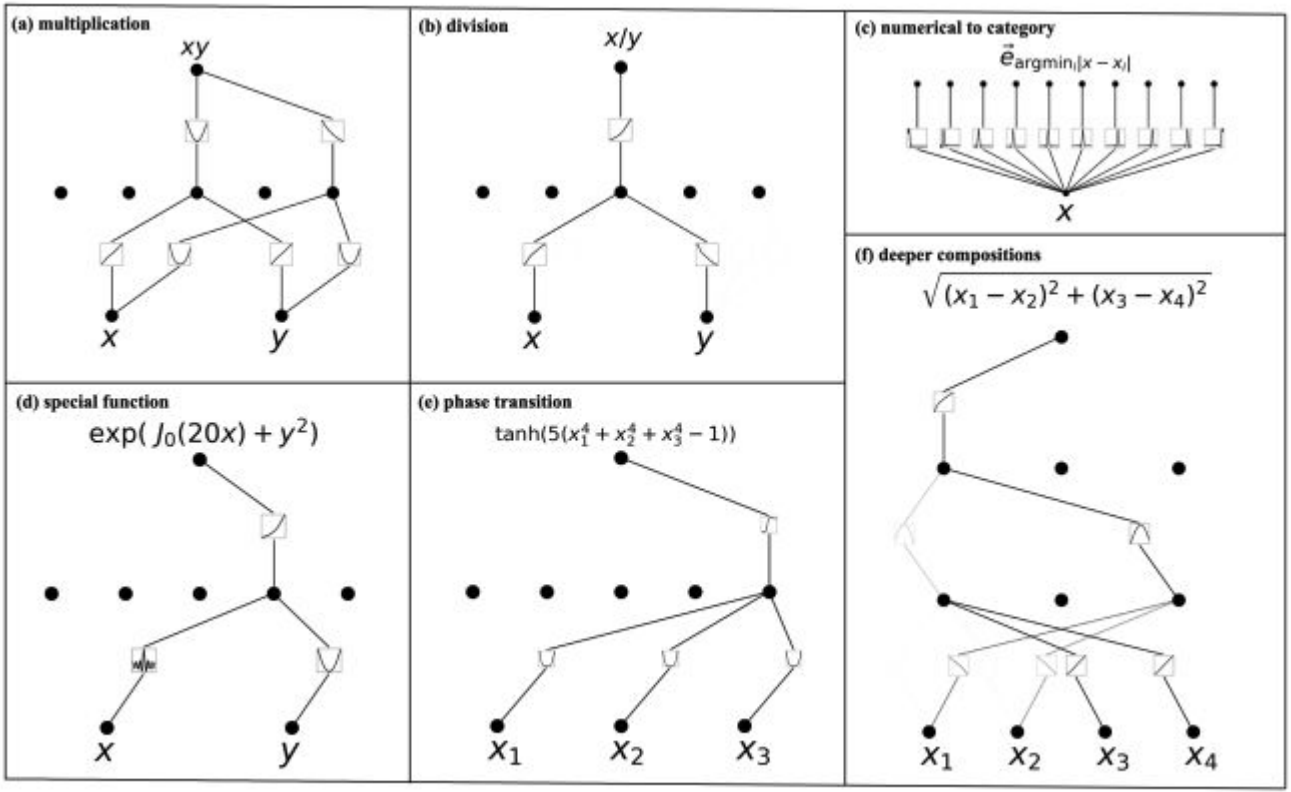
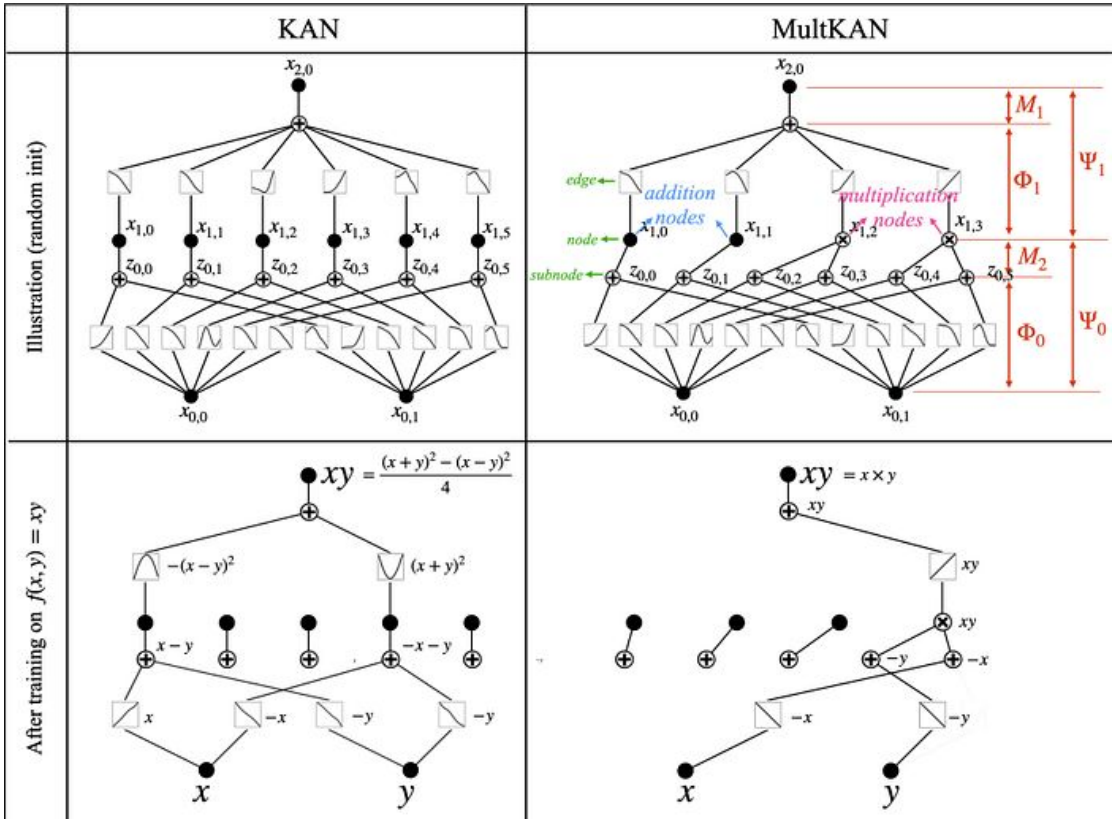


Figure 4.1: KANs are interepretable for simple symbolic tasks

https://arxiv.org/pdf/2404.19756

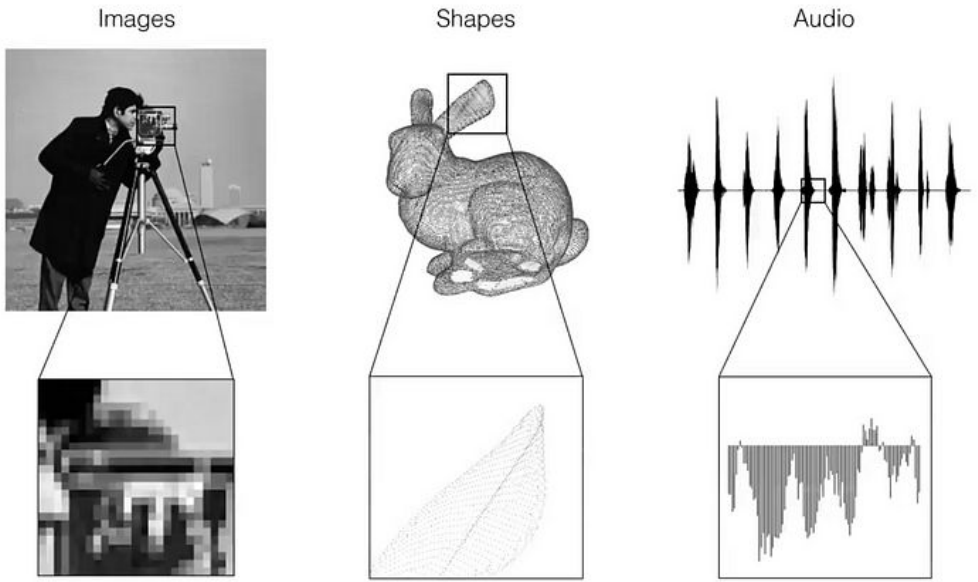
MultKAN



https://arxiv.org/pdf/2408.10205

Implicit Neural Representations (INRs)

- Also referred to as coordinate-based representations or neural fields, INRs are a way to parameterize signals of all kinds



https://medium.com/@nathaliemariehager/an-introduction-to-neural-implicit-representations-with-use-cases-ad331ca12907

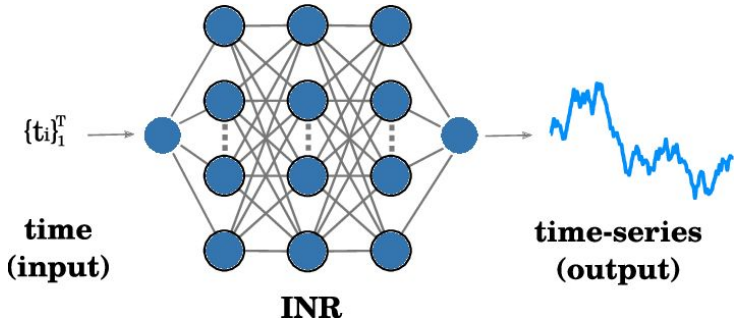
Implicit Neural Representations

Coordinate-based networks model signals as continuous mappings:

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^m$$

where coordinates $x \in \mathbb{R}^d$

map to attributes $y \in \mathbb{R}^m$



https://www.researchgate.net/figure/Diagram-of-the-implicit-neural-representation-INR-for-univariate-time-series-Neurons_fig1_362643443

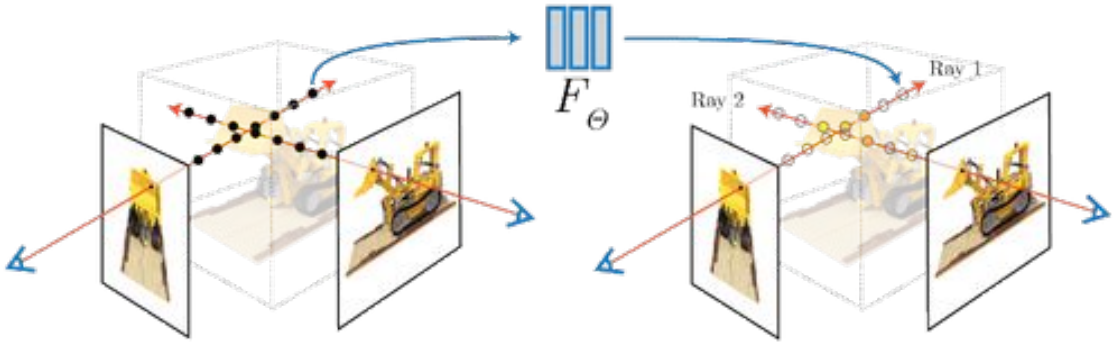
Differentiable Signal Processing

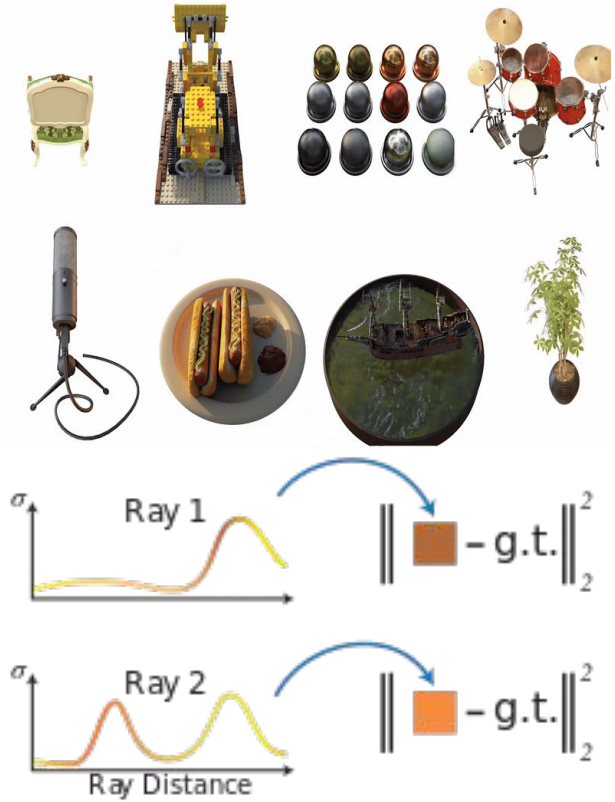
- Unlike discrete representations (grids or meshes) INRs learn a continuous mapping, this means every point in the domain has a derivative
- Many objective functions depend on derivatives - continuous differentiability means gradient-based optimizations can find optimal solutions not just approximations

Solving the wave equation

Neural Radiance Fields (NeRF)

- NeRF uses a fully-connected deep network
- Input is a single 5D coordinate: spatial location (x,y,z) , and viewing direction (θ, φ)
- Output is the volume density and view-dependent emitted radiance at that spatial location

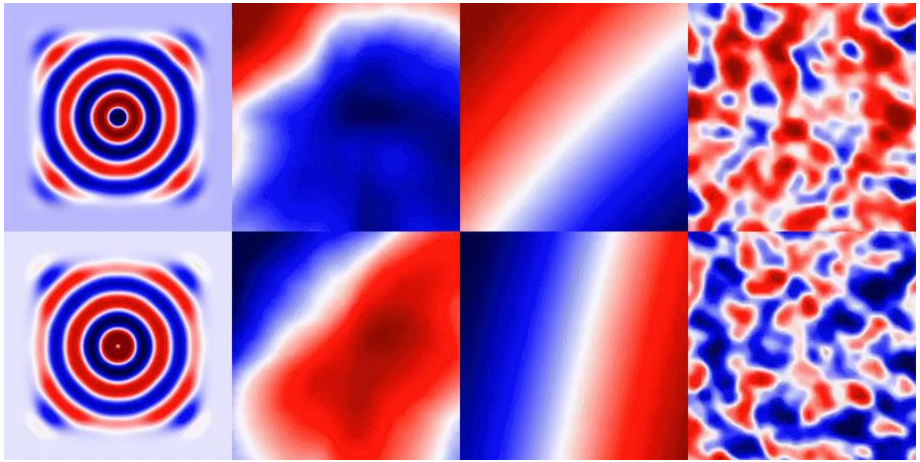




https://www.matthewtancik.com/nerf

Physics-informed applications

- Compact neural encoding allows rapid evaluation across different physical parameters without full resimulation
- Continuous mappings allow complex physical phenomena to be stored compactly without discretization
- The differentiable nature of INRs allows exact computation of derivatives, improving PDE solution accuracy



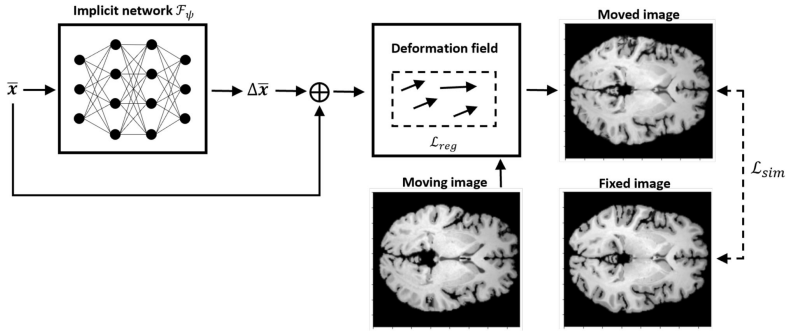
$$H(m)\Phi(\mathrm{x}) = -f(\mathrm{x}), \text{ with } H(m) = \left(\Delta + m(\mathrm{x})\, w^2\right)$$

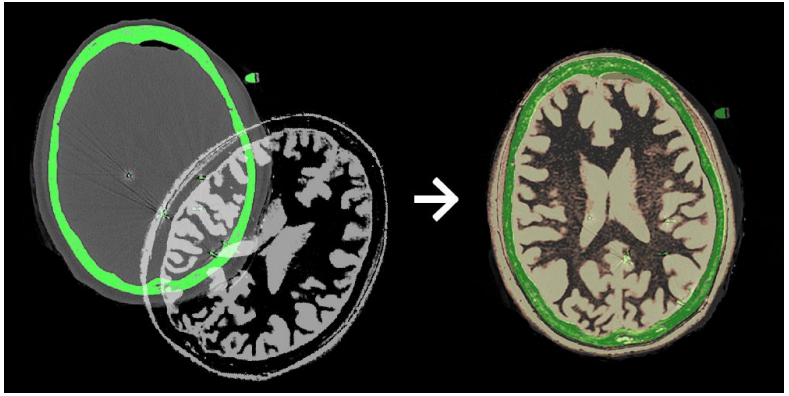
Medical Applications

If you have MRIs undersampled in the k-space (for an accelerated MRI scan) an INR can be used to reconstruct it with missing data

For image registration, INRs naturally model continuous deformation fields between images, producing mappings that preserve topology while allowing computation of exact spatial derivatives for optimization

INRs support arbitrary slice orientations and zoom levels after training, making it well-suited for interactive medical visualization and analysis

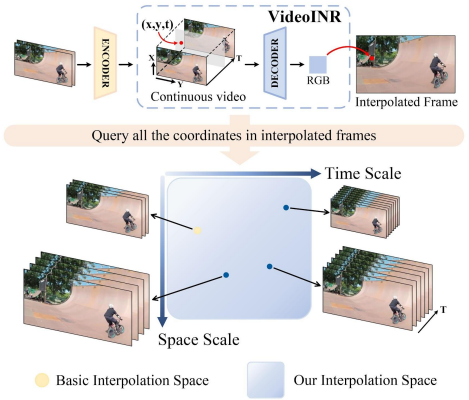




https://www.nature.com/articles/s41598-023-44517-5/figures/1
https://3dqlab.stanford.edu/wp-content/uploads/2023/04/registered-final.png

Resolution Independence (VideoINR)

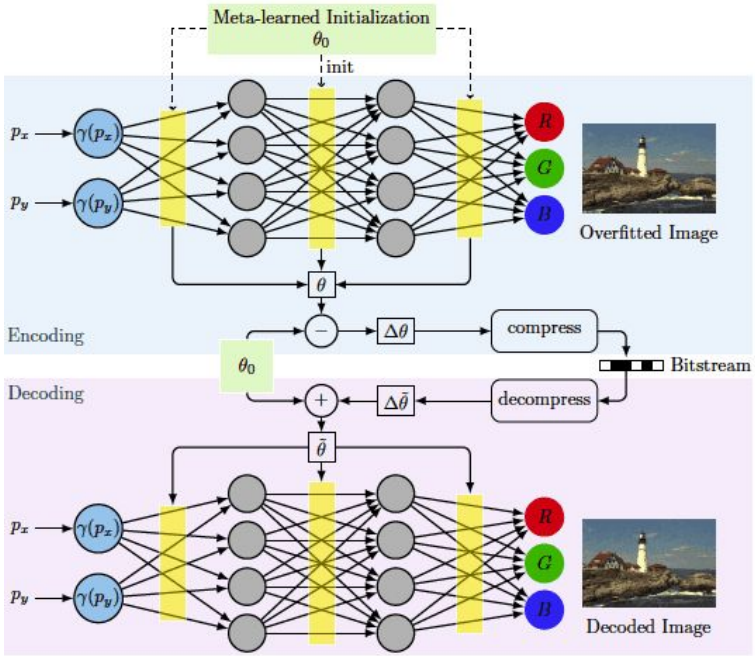
- (x,y,t) are inputs to the model
- Model learns the resulting frame
- Allows for upsampling in resolution and framerate



https://zeyuan-chen.com/VideoINR/

Memory efficiency

- INRs achieve high compression ratios by encoding complex signals through compact networks
- You can use the same network structure to learn signals in different domains
- INRs preserve sharp features and do not introduce blocking artifacts or ringing

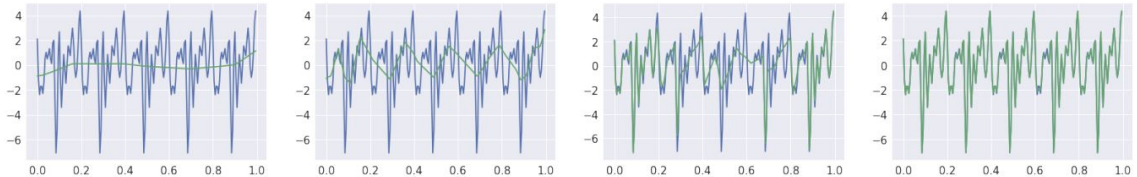


https://arxiv.org/pdf/2112.04267

INR Challenges

Spectral bias - neural networks prioritize learning the low frequency modes first

- Even if a high frequency component has a larger amplitude in the target signal, the network will still learn to approximate lower frequency components that are less prominent first



(a) Iteration 100 (b) Iteration 1000 (c) Iteration 10000 (d) Iteration 80000

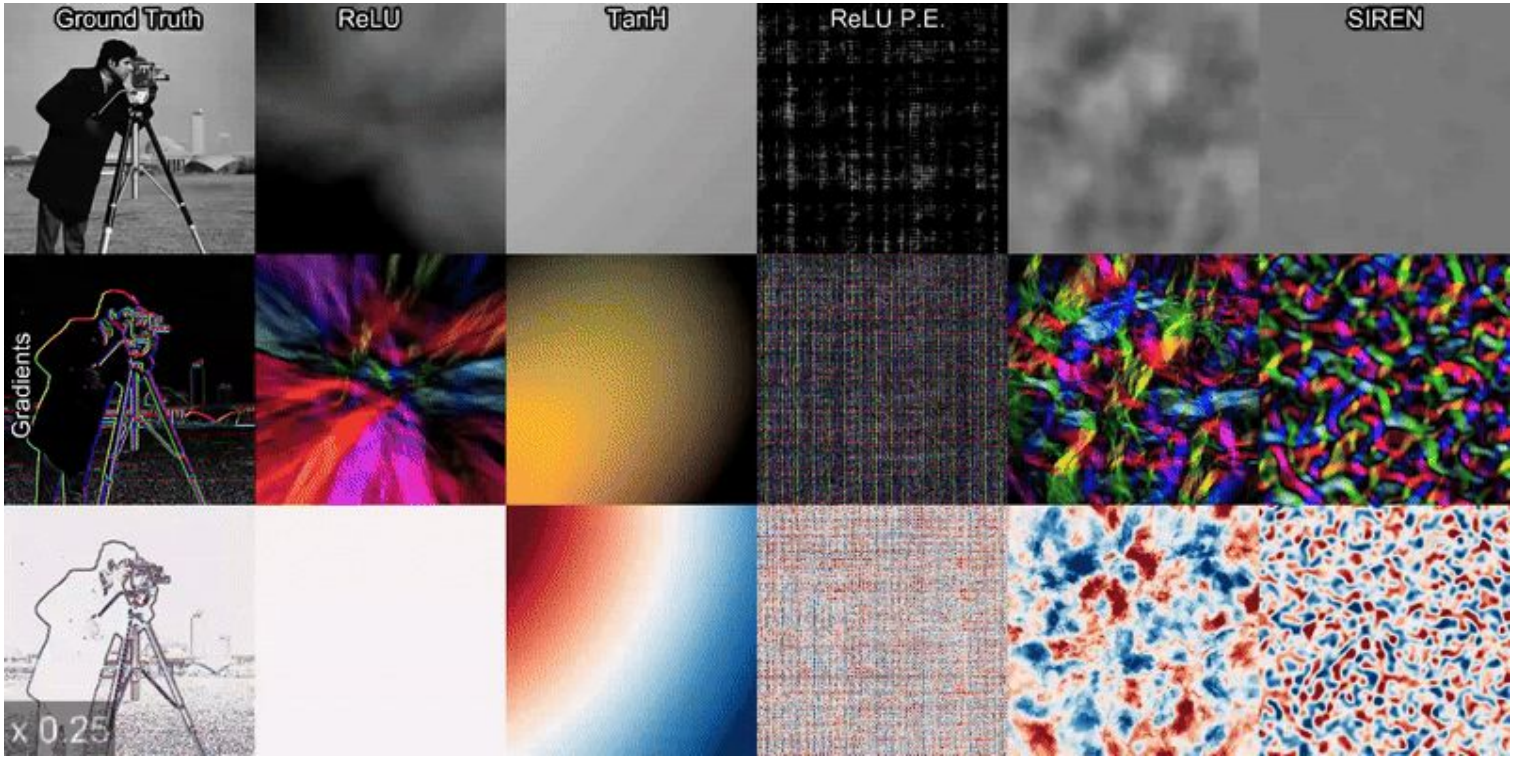
Figure 2. The learnt function (green) overlayed on the target function (blue) as the training progresses. The target function is a superposition of sinusoids of frequencies $\kappa = (5, 10, ..., 45, 50)$, equal amplitudes and randomly sampled phases.

Inference requires a separate network evaluation for each point, making sampling of the domain expensive vs. grid based lookups

INRs also suffer from the **curse of dimensionality** as the required number of training samples grows exponentially with input dimensions

https://arxiv.org/pdf/1806.08734

Sinusoidal Representation Networks (SIREN)



Periodic Activations for INRs

$$F\left(\mathbf{x}, \Phi, \nabla_{\mathbf{x}}\Phi, \nabla^2_{\mathbf{x}}\Phi, \ldots\right) = 0, \quad \Phi : \mathbf{x} \mapsto \Phi(\mathbf{x})$$

This formulation takes as input the spatial or spatio-temporal coordinates $x \in \mathbb{R}^m$ and, optionally, derivatives of $\mathbf{\Phi}$ with respect to these coordinates.

$$\Phi\left(\mathbf{x}\right) = \mathbf{W}_n\left(\phi_{n-1} \circ \phi_{n-2} \circ \ldots \circ \phi_0\right)\left(\mathbf{x}\right) + \mathbf{b}_n,$$

$$\mathbf{x}_i \mapsto \phi_i\left(\mathbf{x}_i\right) = \sin\left(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i\right).$$

$$\phi_i : \mathbb{R}^{M_i} \rightarrow \mathbb{R}^{N_i} \text{ is the } i^{th} \text{ layer of the network}$$

https://www.vincentsitzmann.com/siren/

Simple example: fitting an image

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3,\ x\ \rightarrow \Phi\left(x\right)$$

$$\text{dataset } D = \{(\mathrm{x}_i,\ f\left(\mathrm{x}_i\right))\}_i \text{ of pixel coordinates } \mathrm{x}_i = (x_i, y_i) \text{ with RGB colors } f\left(\mathrm{x}_i\right)$$

$$\mathcal{L} = \sum_i \left|\left|\Phi\left(\mathrm{x}_i\right) - f\left(\mathrm{x}_i\right)\right|\right|^2$$

SIREN initialization scheme

The key idea in the initialization scheme is to preserve the distribution of activations through the network so that the final output does not depend on the number of layers

- The authors propose to draw weights with:

$$c = 6 \text{ so that } w_i \sim \mathcal{U}\left(-\sqrt{6/n}, \sqrt{6/n}\right)$$

This is because when each component of weights a layer $\mathbf{w}$ is uniformly distributed such as

$$w_i \sim \mathcal{U}\left(-c/\sqrt{n}, c/\sqrt{n}\right)$$

The dot product of converges to the normal distribution $\mathbf{w}^T\mathbf{x} \sim \mathcal{N}\left(0, c^2/6\right)$ this ensures the input to each sine activation is normally distributed with a standard deviation of 1

Finally, the first sine layer needs to be initialized with weights so that the sin function $\sin\left(\omega_0 \cdot \mathbf{W}\mathbf{x} + \mathbf{b}\right)$ spans multiple periods over [-1,1]

WIRE: Wavelet Implicit Neural Representations

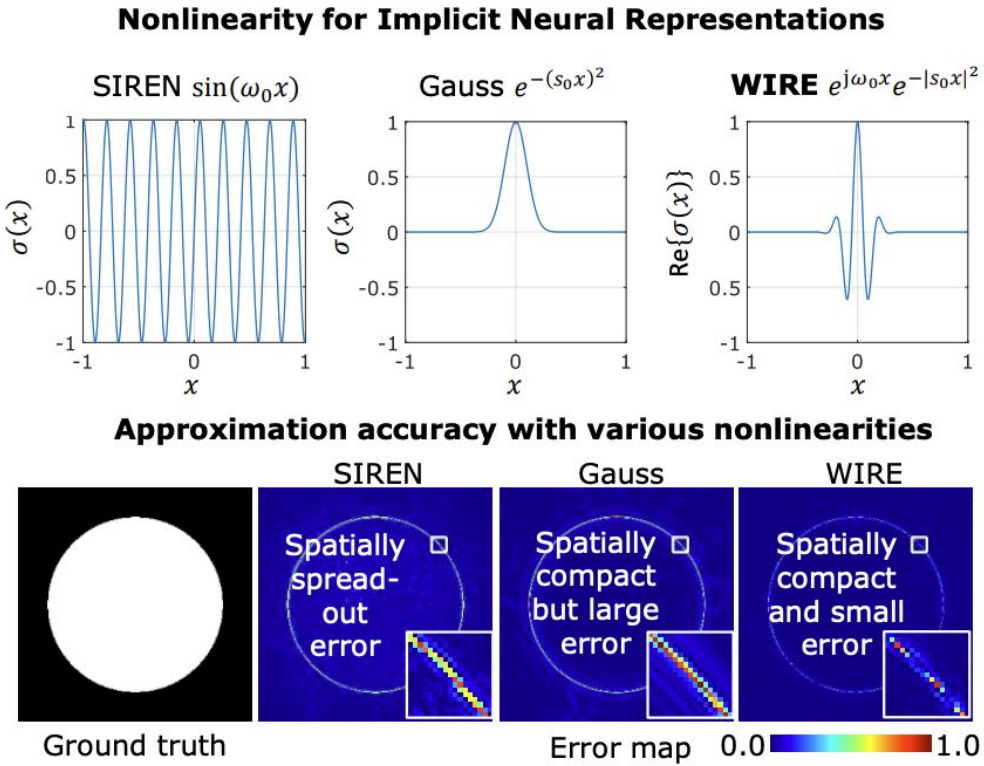
WIRE uses the continuous complex Gabor wavelet ψ for its activation:

$$\sigma(x) = \psi(x; \omega_0, s_0) = e^{j\omega_0 x} e^{-|s_0 x|^2}$$

ω_0 controls the frequency
s_0 controls the width

The first layer activations have the form:

$$\mathrm{y}_1 = \mathrm{psi}(\mathrm{W}_1 \mathrm{x} + \mathrm{b}_1; \omega_0, s_0)$$



https://arxiv.org/pdf/2301.05187

Conv-INR

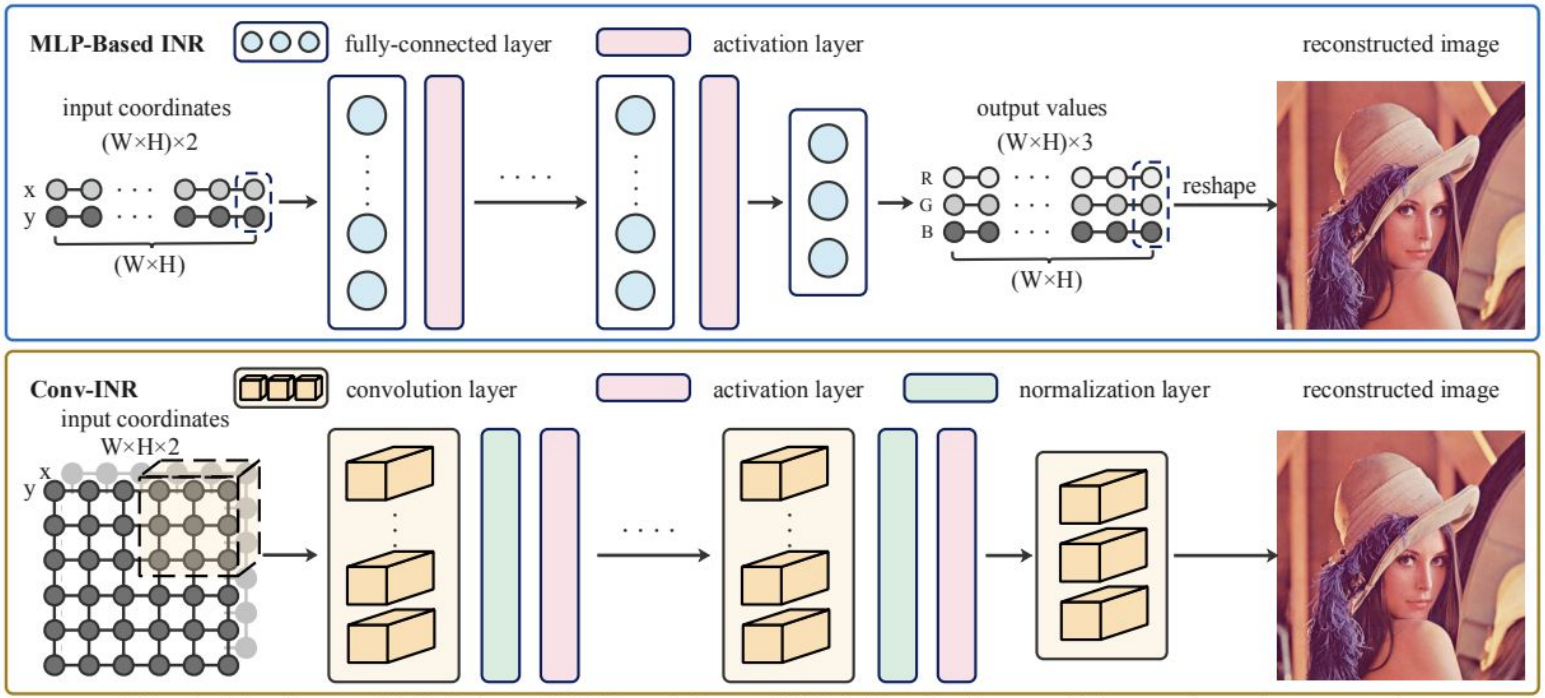


Fig. 1: Pipelines of MLP-based INR and Conv-INR. Take the image fitting task as the example.

https://arxiv.org/pdf/2406.04249