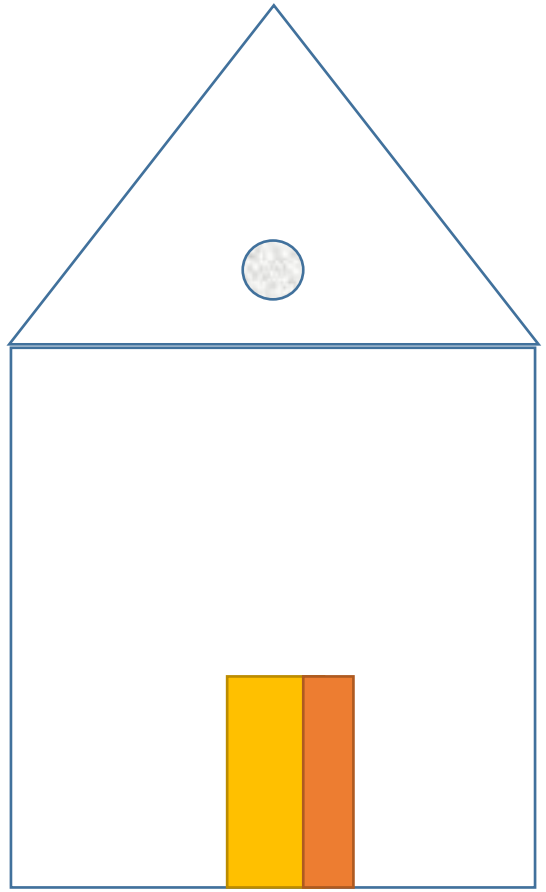
Simple Game

Draw a house on a paper

90% of people have drawn a house like



Question:

How many of your houses are like this?

Simple Song

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

Then the traveller in the dark
Thanks you for your tiny spark;
He could not see which way to go,
If you did not twinkle so.

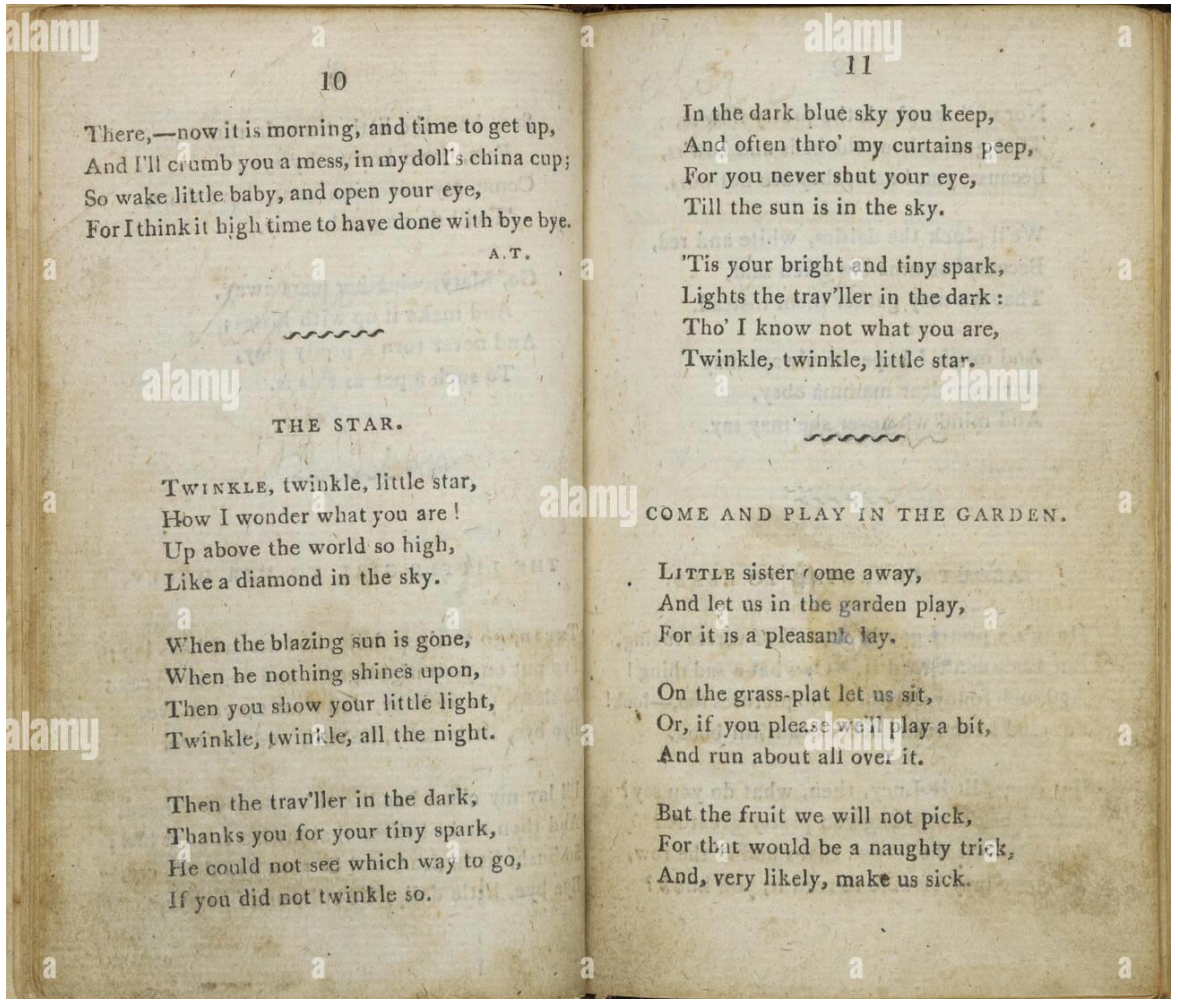
When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

In the dark blue sky you keep,
And often through my curtains peep,
For you never shut your eye
Till the sun is in the sky.

As your bright and tiny spark
Lights the traveller in the dark,
Though I know not what you are,
Twinkle, twinkle, little star.

Who wrote this?

(from "The Star," Jane Taylor, 1806)



10

There,—now it is morning, and time to get up,
And I'll crumb you a mess, in my doll's china cup;
So wake little baby, and open your eye,
For I think it high time to have done with bye bye.

A. T.

THE STAR.

Twinkle, twinkle, little star,
How I wonder what you are!
Up above the world so high,
Like a diamond in the sky.

When the blazing sun is gone,
When he nothing shines upon,
Then you show your little light,
Twinkle, twinkle, all the night.

Then the trav'ller in the dark,
Thanks you for your tiny spark,
He could not see which way to go,
If you did not twinkle so.

11

In the dark blue sky you keep,
And often thro' my curtains peep,
For you never shut your eye,
Till the sun is in the sky.

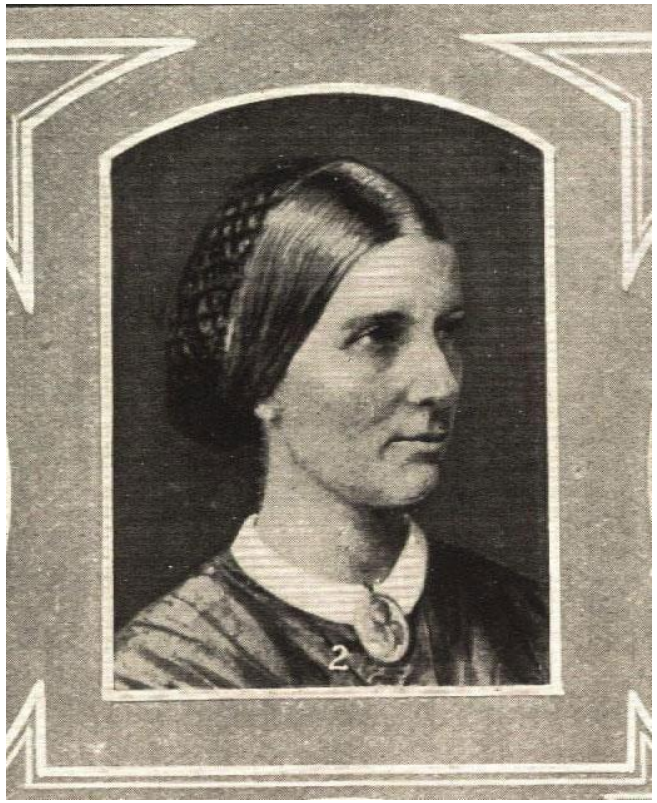
'Tis your bright and tiny spark,
Lights the trav'ller in the dark:
Tho' I know not what you are,
Twinkle, twinkle, little star.

COME AND PLAY IN THE GARDEN.

Little sister come away,
And let us in the garden play,
For it is a pleasant day.

On the grass-plat let us sit,
Or, if you please we'll play a bit,
And run about all over it.

But the fruit we will not pick,
For that would be a naughty trick,
And, very likely, make us sick.



Question:

How many words were in red/green/blue?

Another Game

Rapid Fire Round: Quiz

What does it represent?

$$A = lb$$

Rectangle

What does it represent?

$$W = mg$$

Weight = mass times gravity

What does it represent?

$$F = ma$$

Newton's Second Law

What does it represent?

$$y = mx$$

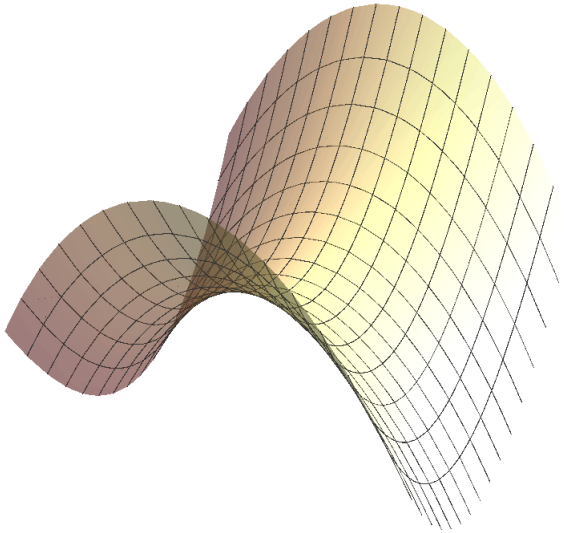
Equation of Straight Line

What does it represent?

$$z = xy$$

What does it represent?

$$z = xy$$



hyperbolic paraboloid

What does it represent?

$$A = \pi r^2$$

Area of a Circle

What does it represent?

$$E = mc^2$$

Einstein Equation

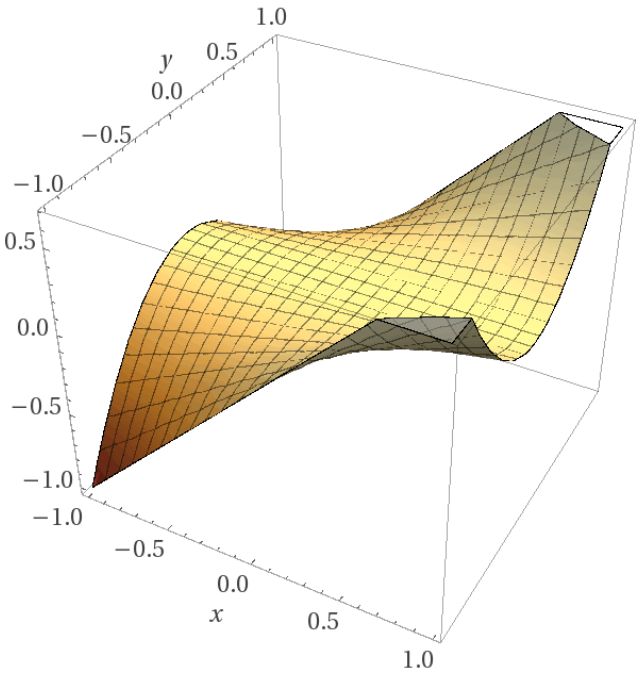
What does it represent?

$$y = ax^2$$

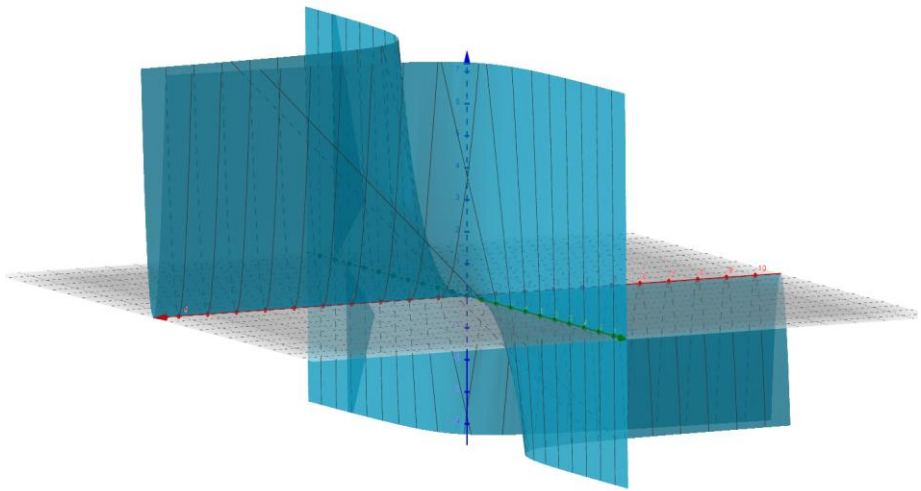
Parabola

What does it represent?

$$z = xy^2$$



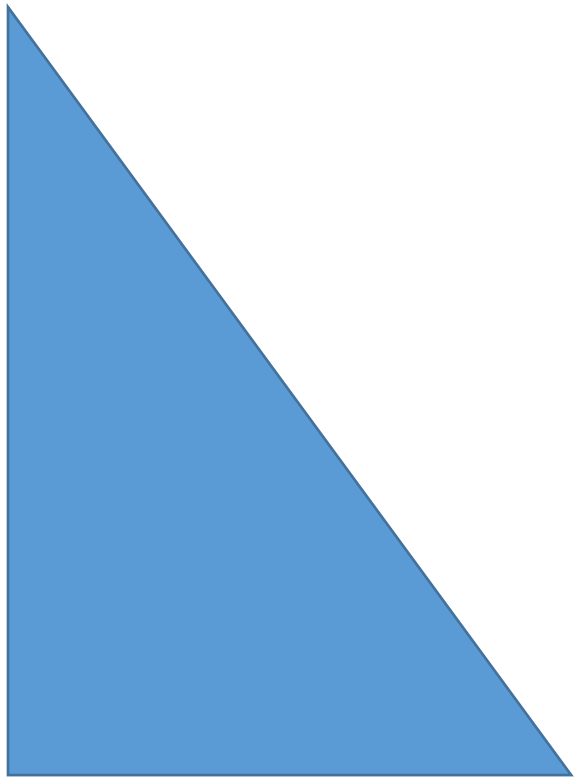
$$z = xy^2$$



What does it represent?

$$a^2 + b^2 = c^2$$

Pythagoras Theorem



What does it represent?

$$x^2 + y^2 = r^2$$

Equation of Circle

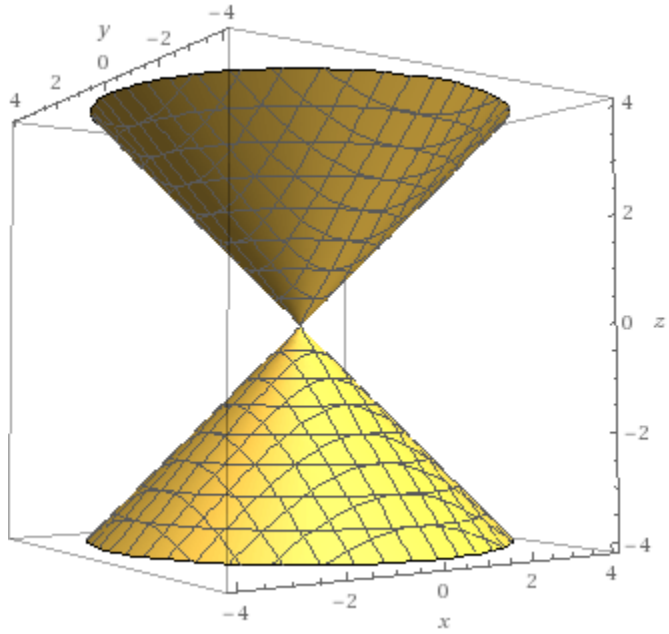
What does it represent?

$$x^2 + y^2 = z^2$$

Some 3D equation

What does it represent?

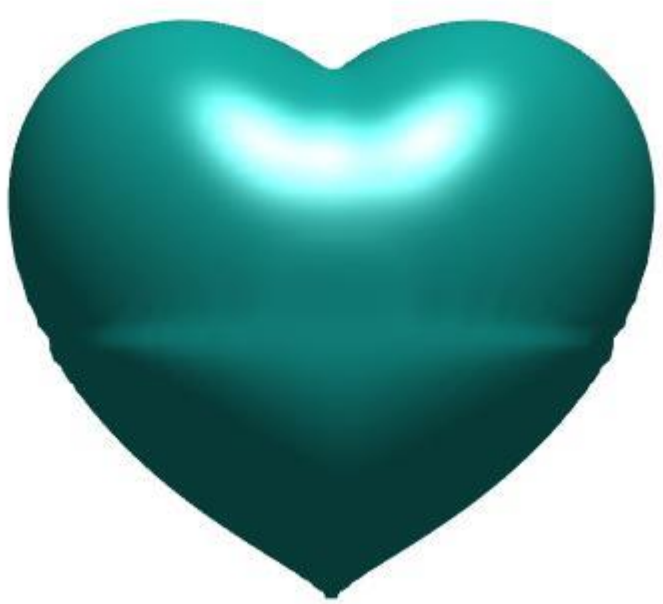
$$x^2 + y^2 = z^2$$



What does it represent?

$$\left(x^2+\frac{9}{4}y^2+z^2-1\right)^3-x^2z^3-\frac{9}{80}y^2z^3=0$$

$$\left(x^2+\frac{9}{4}y^2+z^2-1\right)^3-x^2z^3-\frac{9}{80}y^2z^3=0$$



$$y = mx$$

$$A = lb$$

$$F = ma$$

$$W = mg$$

$$z = xy$$

$$y = ax^2$$

$$A = \pi r^2$$

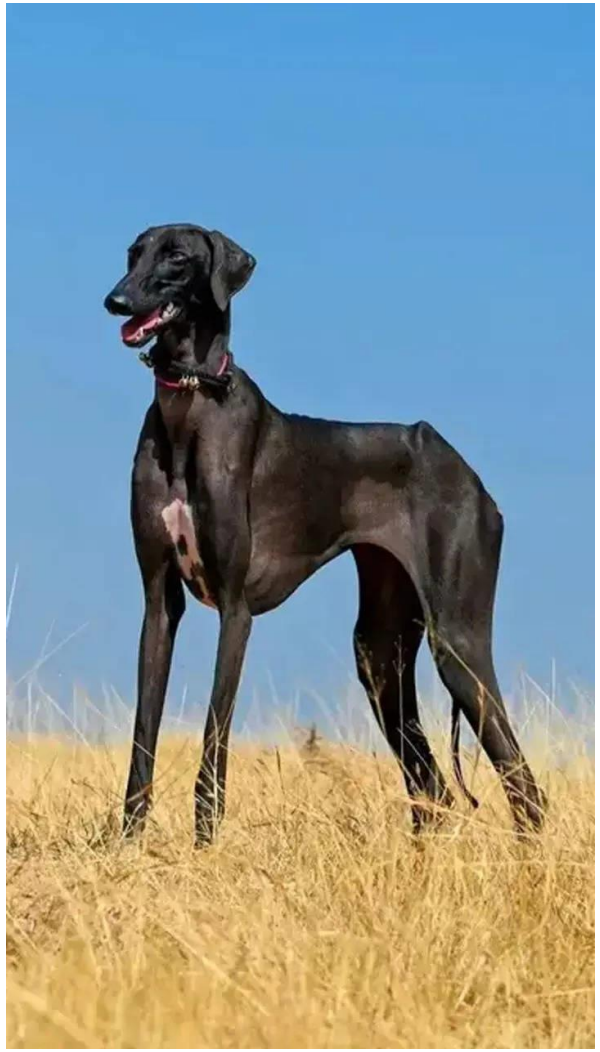
$$E = mc^2$$

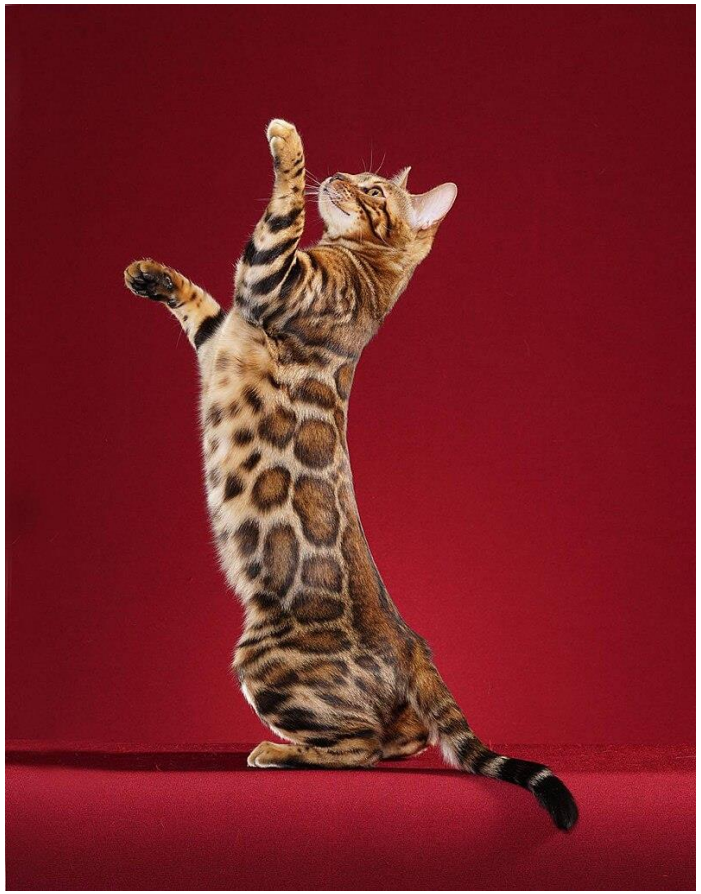
$$z = xy^2$$

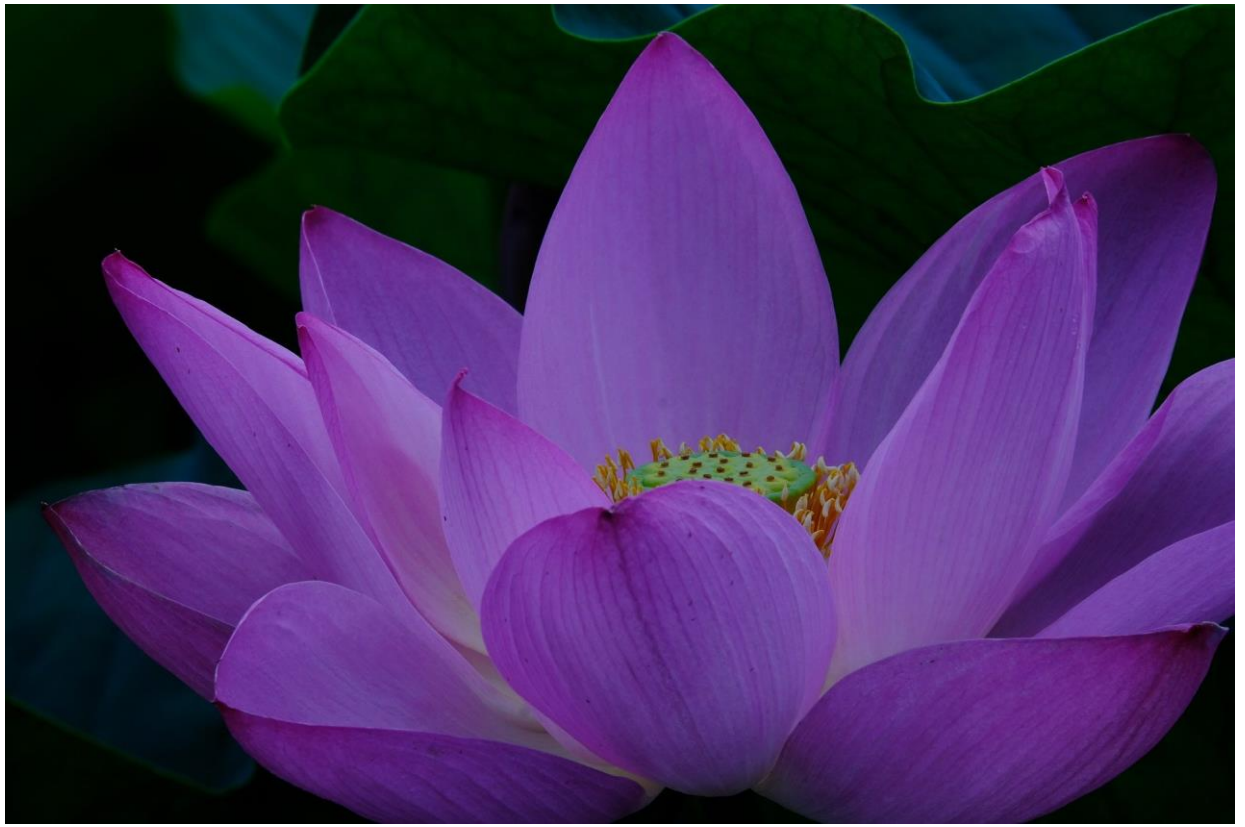
$$x^2 + y^2 = z^2$$

$$x^2 + y^2 = r^2$$

$$a^2 + b^2 = c^2$$







German Shepherd

Labrador Retriever

360 Globally Recognized Breeds

Rajapalayam Dog

French Bulldogs

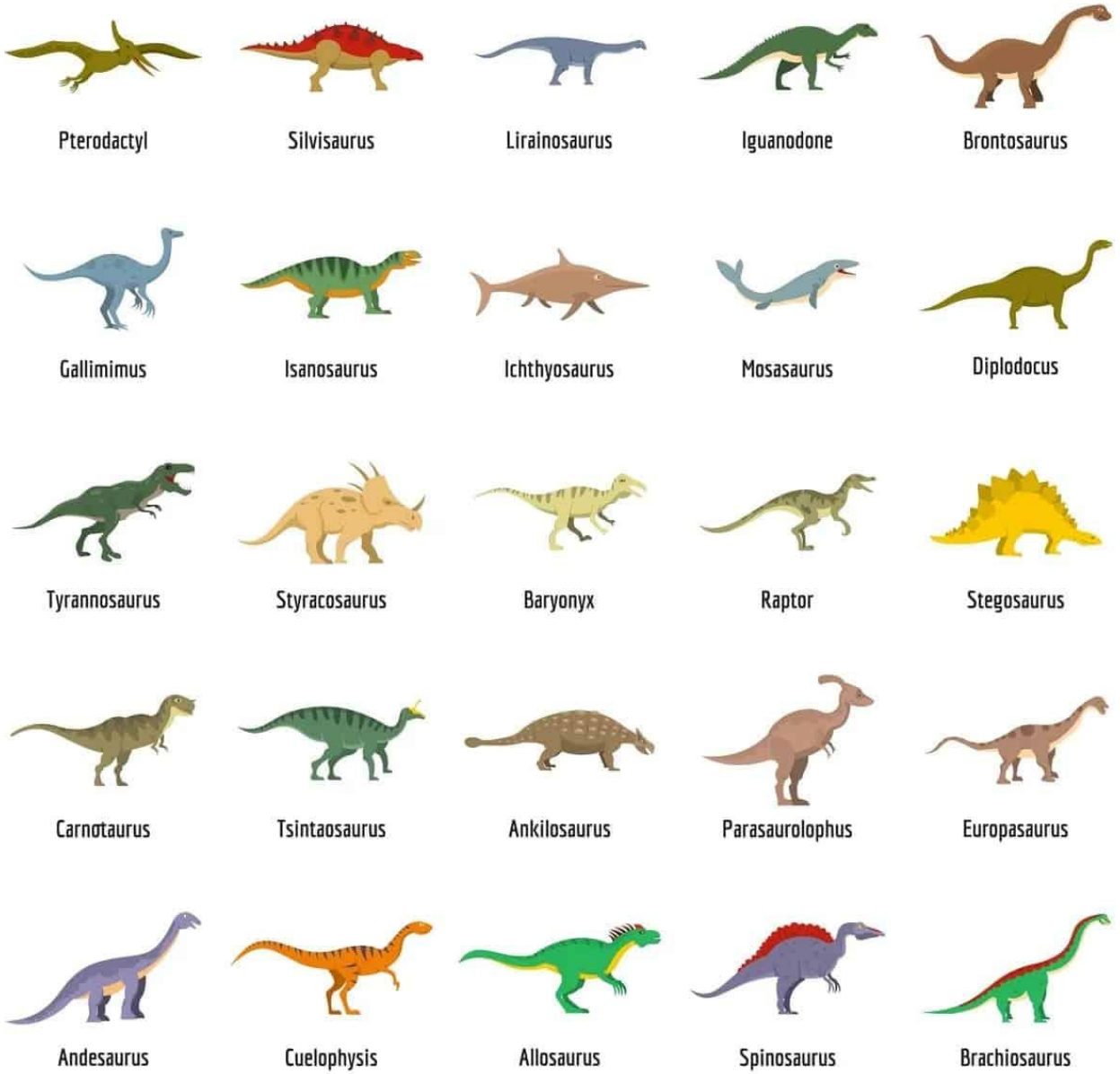
The American Eskimo Dog

73 Standardized Breeds

rhebamny.com

Approximately 400,000 Flowering Plants

https://nayturr.com/types-of-dinosaurs/



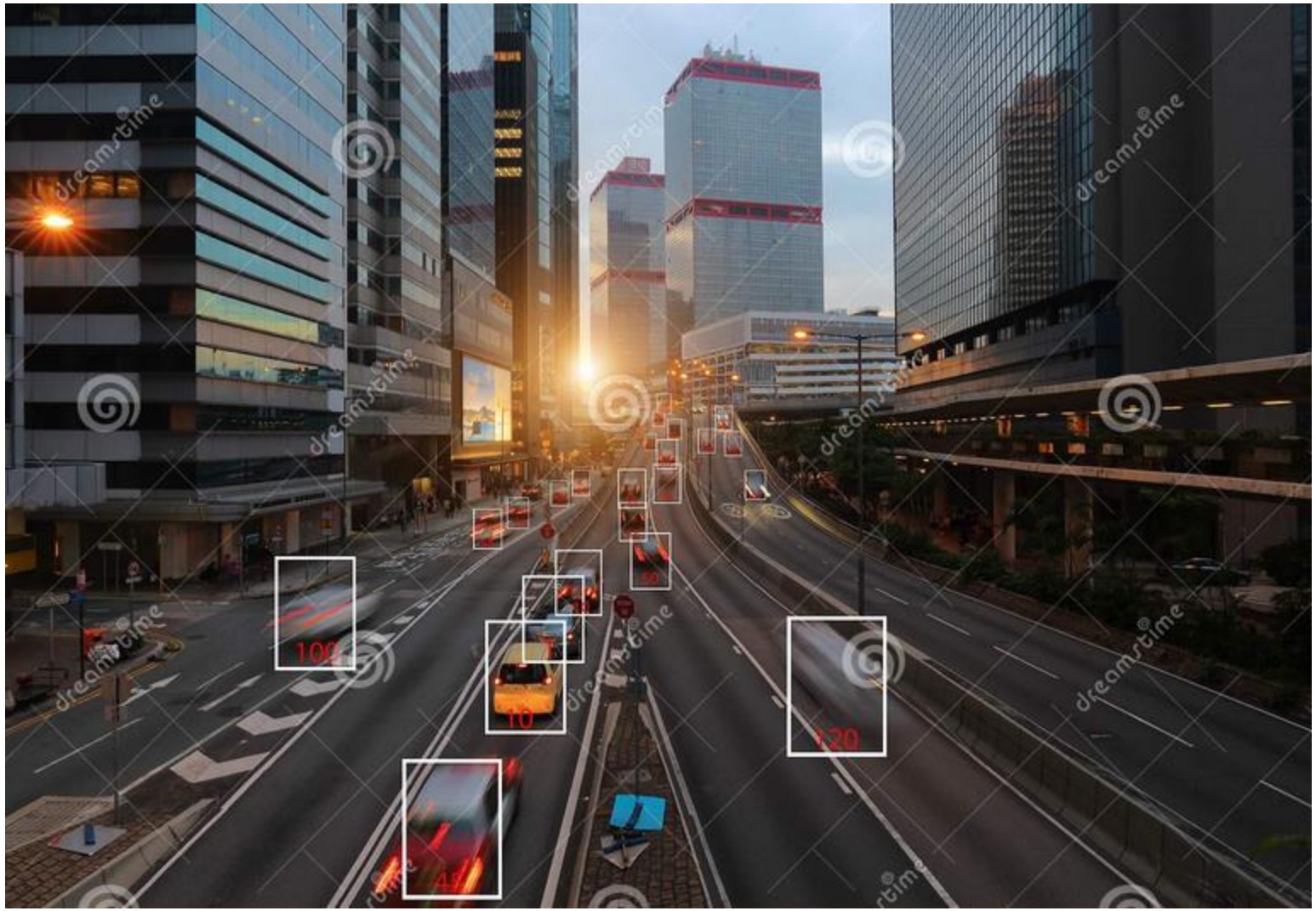
Congratulations! You have done the labelling job well.

I mean you are fit to learn machine learning concepts

Let us explore more details with mathematics


THE ART BOOK
THE GREAT MASTERS
AMY TAN


Newbie
23
50
0/5
HIDDEN 4 FUN
MORE GAMES
3
2
You are a police officer in this interesting game and your job is to help the old detective find the bank robbers.


100
10
120

What is "learning" in ML?

Hard question to answer. Let us give a fuzzy answer at a enough high level of abstraction

1. **Algorithms that solve some kind of inference problems**
2. **Models for datasets**

Does the image have only books?

Statistical Inference

Statistical inference is the process of using data analysis to deduce properties of an underlying probability distribution.

Inferential statistical analysis infers properties of a population, for example by testing hypotheses and deriving estimates.

Does the image have only books?

What does a ML algorithm do?

Machine learning algorithms are not algorithms for performing inference. Rather, they are algorithms for building inference algorithms from examples. An inference algorithm takes a piece of data and outputs a decision (or a probability distribution over the decision space).

Second type of problem associated to ML

"Given a dataset how I can succinctly describe it (in a quantitative, mathematical manner"

Example: Regression Analysis

Geometric Models:

The general problem is that we have example data points

$$x_1, x_2, \cdots, x_n \in \mathbb{R}^D$$

We want to find some kind of geometric structure that (approximately) describes them.

Probabilistic Models:

The basic task here is to find a probability distribution that describes the dataset $\{x_n\}$

ML Examples

- **Classification: From data to discrete classes**
  - Spam Filtering
  - Object Detection
  - Weather Prediction (rain, snow)
- **Regression: predict a numeric value**
  - Stock Market
  - Weather Prediction (Temperature)
- **Ranking: comparing items**
  - Web Search
  - Find Similar images
- **Collaborative Filtering: Recommendation**

ML Examples

Clustering: discovers structure in data

- Cluster Point or images
- Cluster web search

Embedding: Visualize data

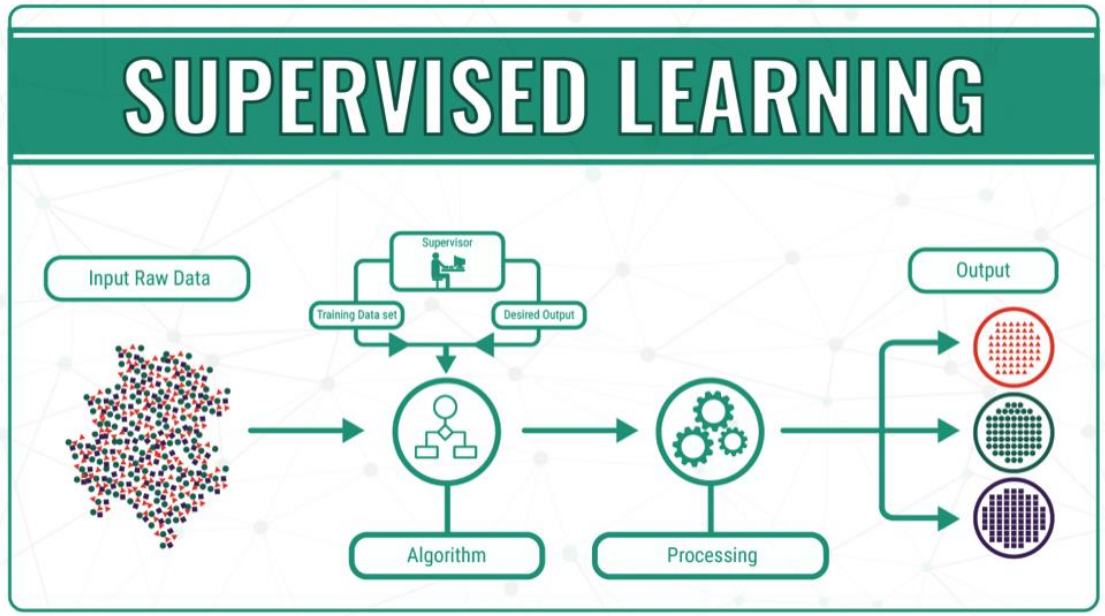
- Images words

Structured Prediction: from data to discrete classes

- Speech Recognition
- NLP

Supervised Learning

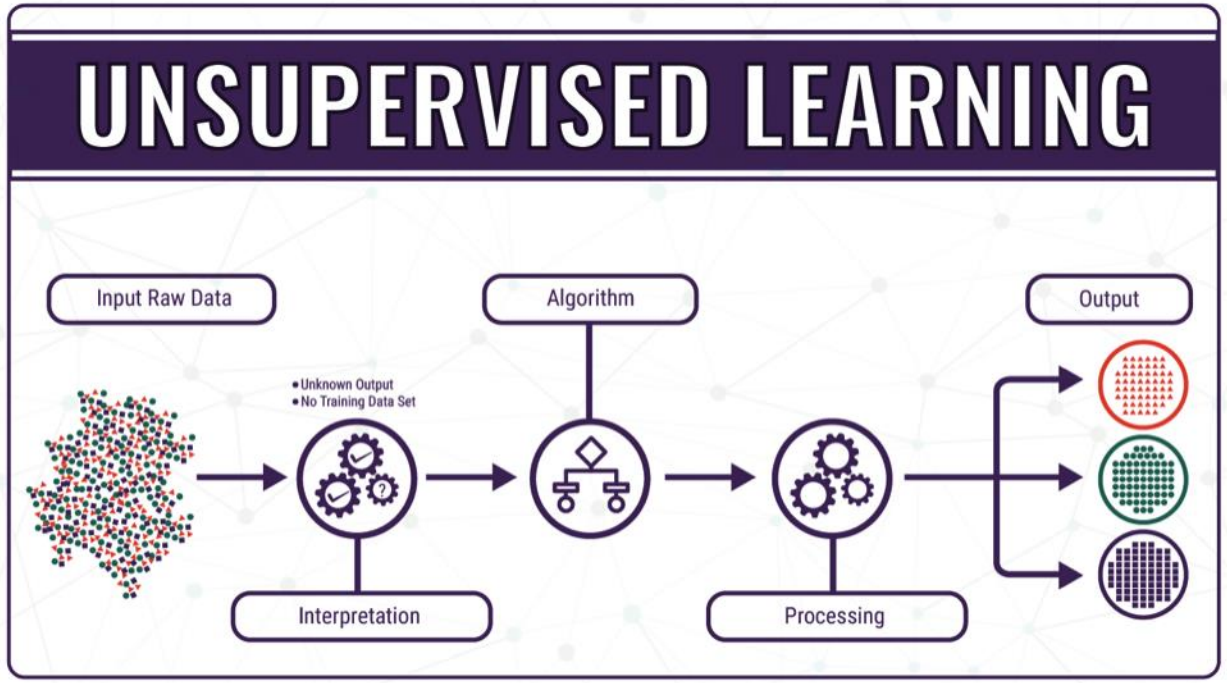
- Builds a Mathematical Model
- Contains input and output data: Training Data
- Relations : Supervisor Signals, F(x)
- Each training example: Array or Vector
- Training Data: Matrix
- Iterative optimization



Source: Educative.io

Unsupervised Learning

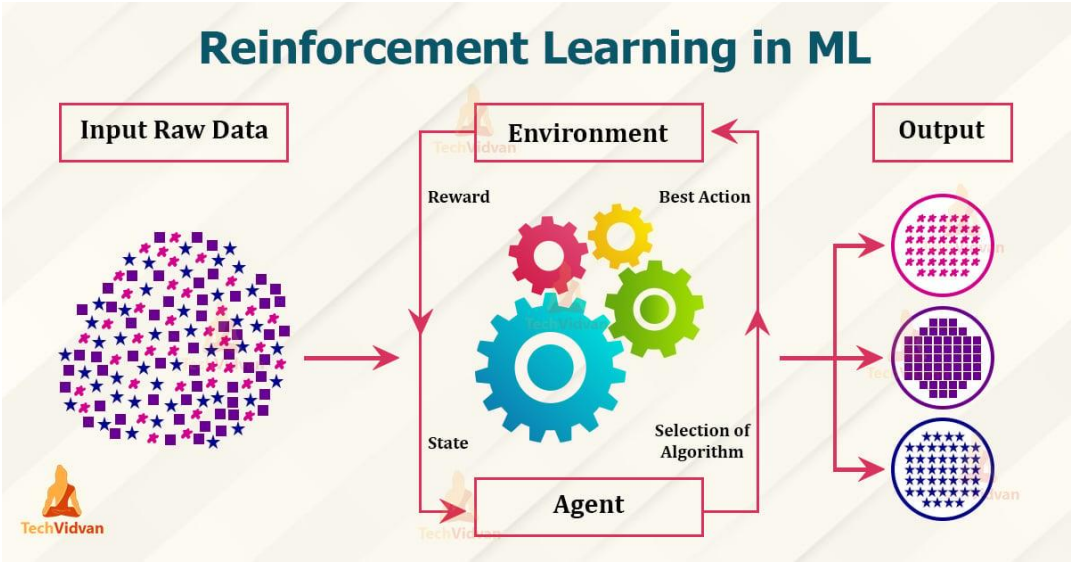
- Takes only input
- Finds the structure in the data
- Groups/Clustering data
- Classifies
- React based on the presence of such commonalities in each new piece of data
- Statistical analysis (density estimation function)
- Weighted to finding probabilities of outcomes (conditional probability)



Source: tecnative.io

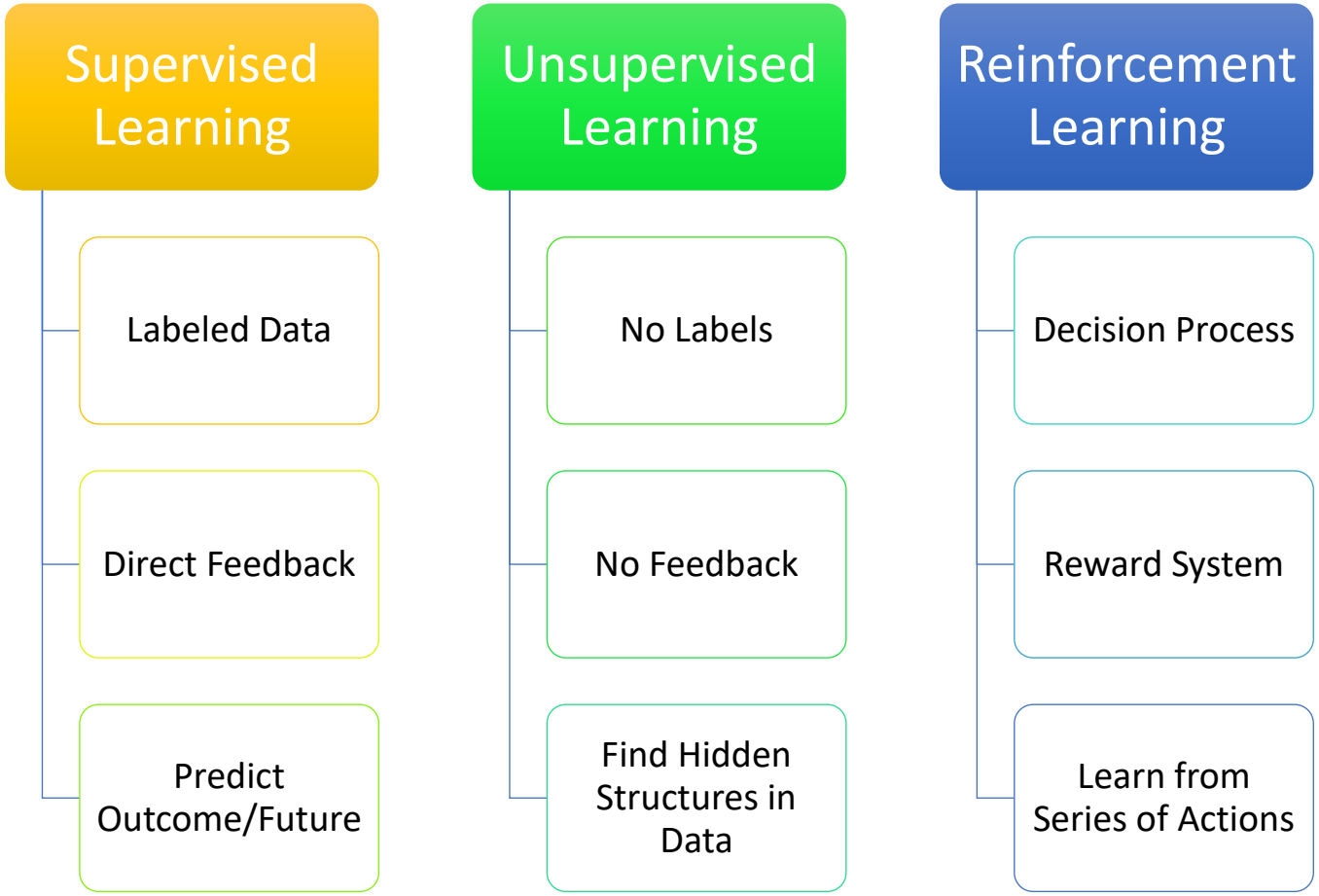
Reinforcement Learning

- Give rewards for every positive result and make based on an algorithm
- **Agent-Based Learning:** Learns by interacting with the environment
- **Trial-and-Error**: Receives rewards or penalties for actions
- **Objective**: Maximize cumulative rewards over time
- **Decision Process**: Uses **Markov Decision Process (MDP)** framework
- **Exploration vs. Exploitation**: Balances between trying new actions and using learned knowledge
- **Optimization**: Iterative improvement of policies (e.g., **Q-learning, Policy Gradient** methods)



Source:https://techvidvan.com/

Three Different Types of ML



Mathematics for Machine Learning

Panchatcharam Mariappan

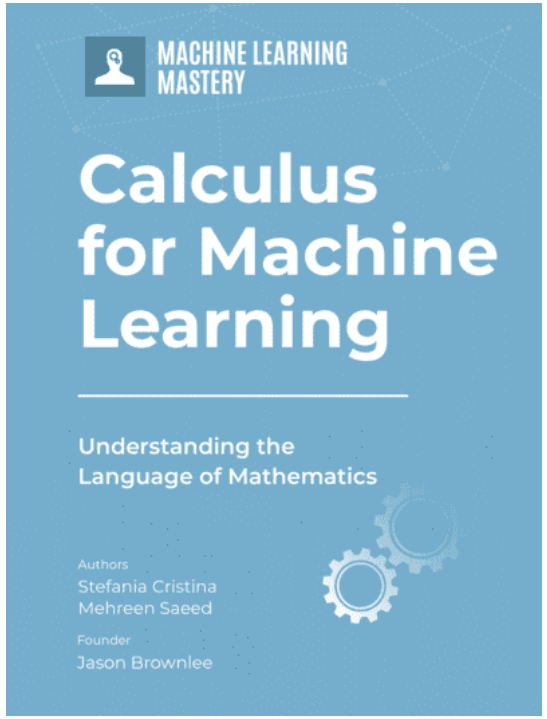
Associate Professor

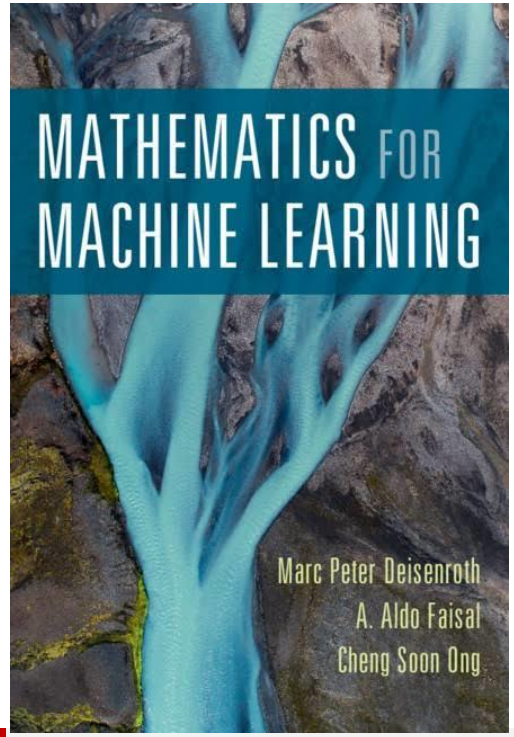
Department of Mathematics and Statistics,
IIT Tirupati

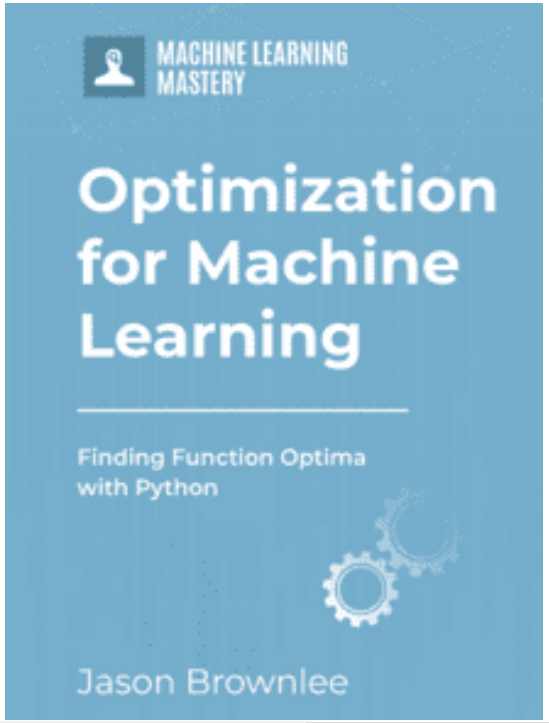
References

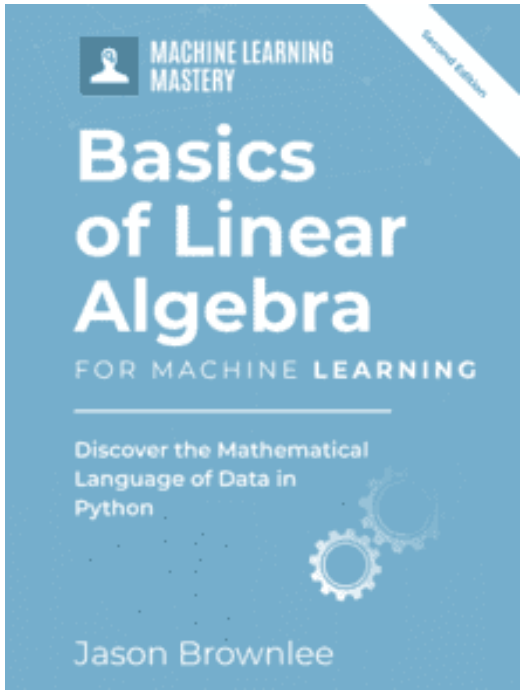
Books (https://machinelearningmastery.com/products/)

- S. Cristina, M. Saeed, Calculus For Machine Learning, Machine Learning Mastery
- M. P. Deisenorth, A. A. Faisal, C. S. Ong, Mathematics for Machine Learning
- J. Brownlee, Optimization for Machine Learning
- J. Brownlee, Basics of Linear Algebra for Machine Learning





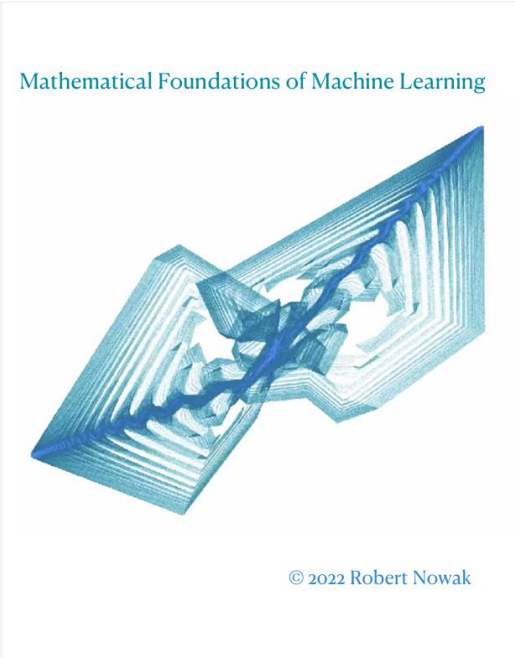


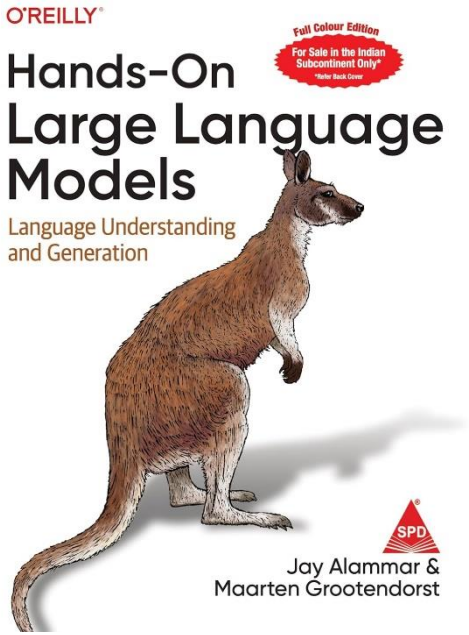


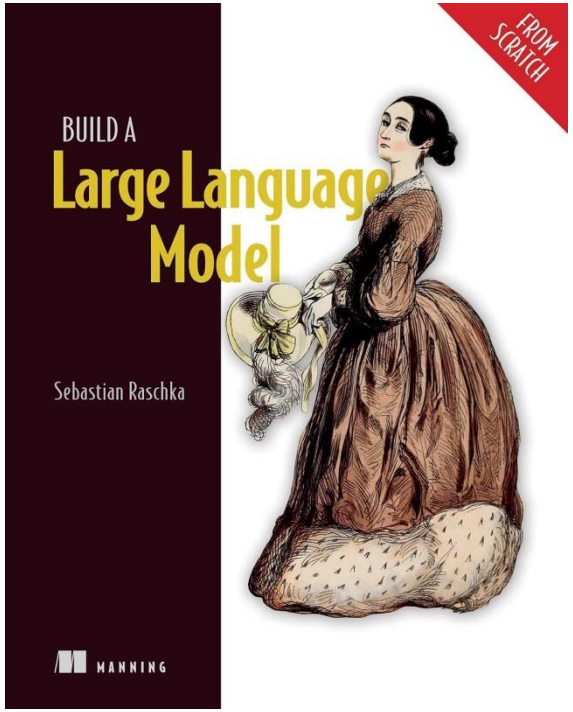
References

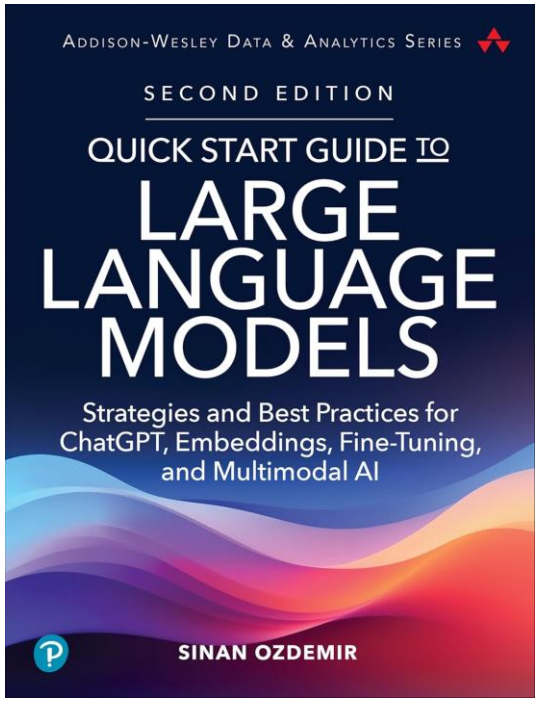
Books (https://machinelearningmastery.com/products/)

- Robert Nowak, Mathematical Foundations of Machine Learning
- T. Xiao and J. Zhu, Foundations of Large Language Models, http://arxiv.org/abs/2501.09223v1
- Seongjai Kim, Mathematical Foundations of Machine Learning, Lecture Notes
- Justing Romberg, Mathematical Foundations of Machine Learning, Lecture Notes



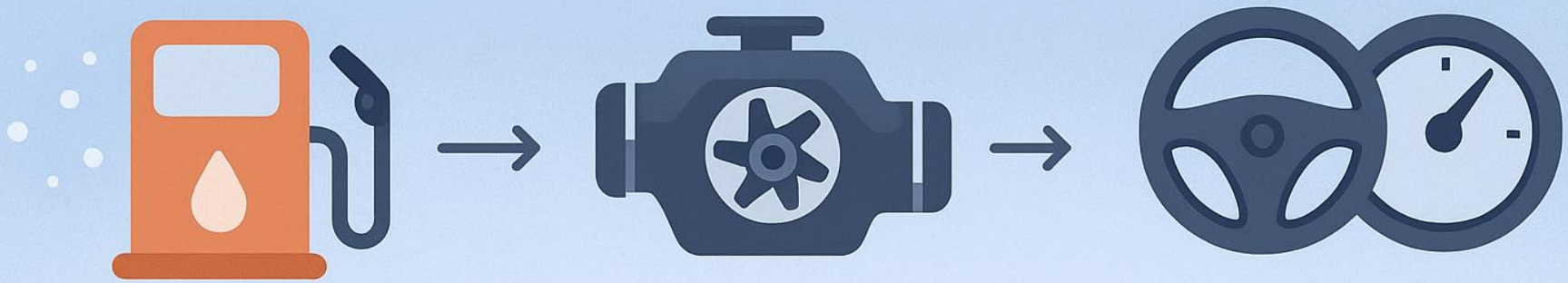






If Data is the fuel of AI, Linear Algebra is its Engine and Vector Calculus is the Navigation/Control System

Data, Linear Algebra, and Vector Calculus: AI's Powertrain



Data: AI's Fuel

Raw information powering the system

Linear Algebra: AI's Engine

Vectors, matrices, embeddings, transformations

Vector Calculus: AI's Navigation

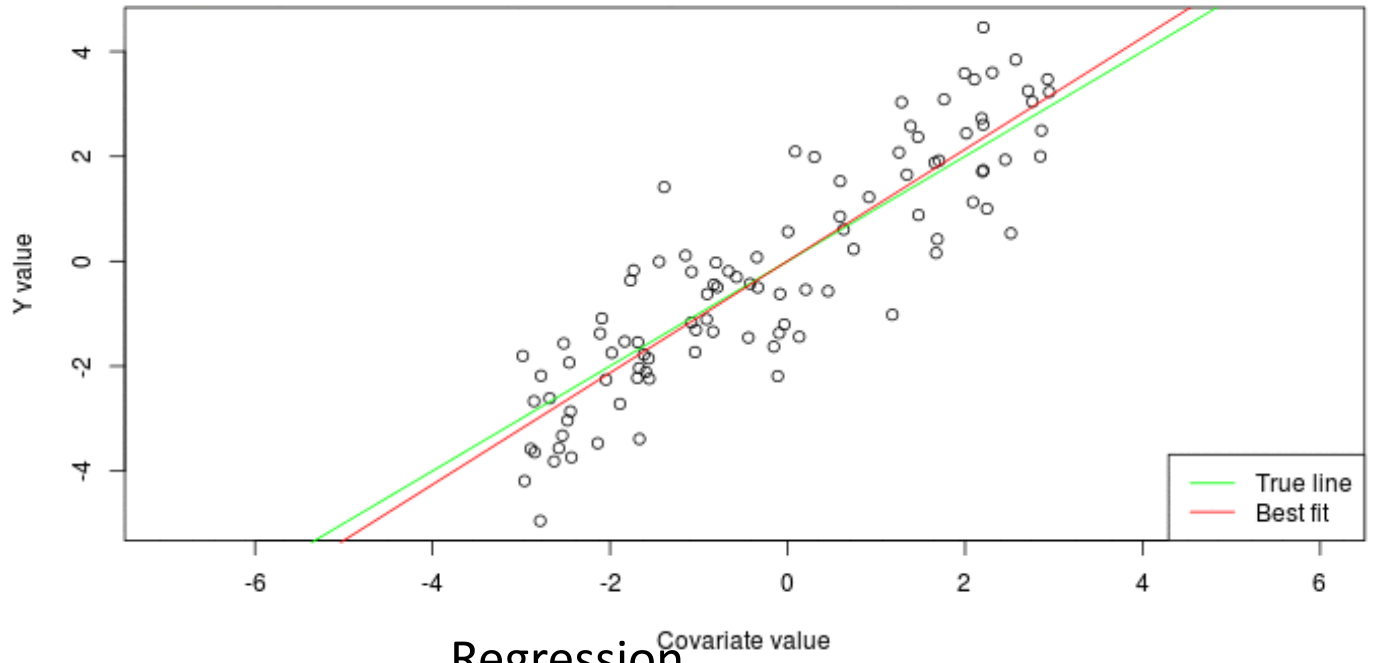
Guides learning, optimization, and information flow

Linear Algebra moves the AI; Vector Calculus tells it where and how to move efficiently.

Machine Learning in Mathematical Way

Supervised Learning

- **Assumption**: Given a data set $\{(x_i, y_i)\}$, $\exists$ a relation $f: X \rightarrow Y$
- **Supervised Learning:**
  - Given: Training Set $\{(x_i, y_i) | i = 1,2, \cdots, N\}$
  - Find: $\hat{f}: X \rightarrow Y$ a good approximation to f

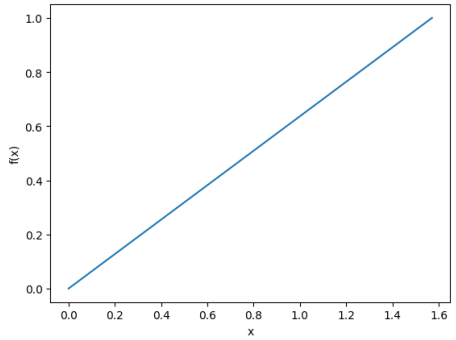


Regression

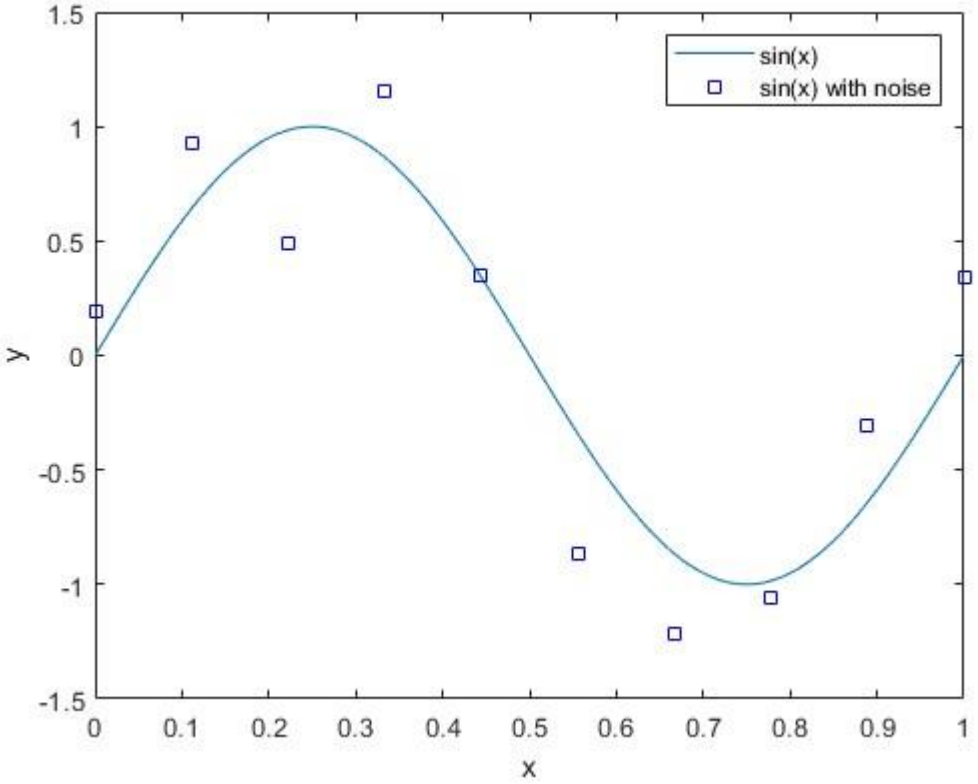
- Girls vs Boys

Simple Example

x	0	$\frac{\pi}{2}$
$f(x)$	0	1

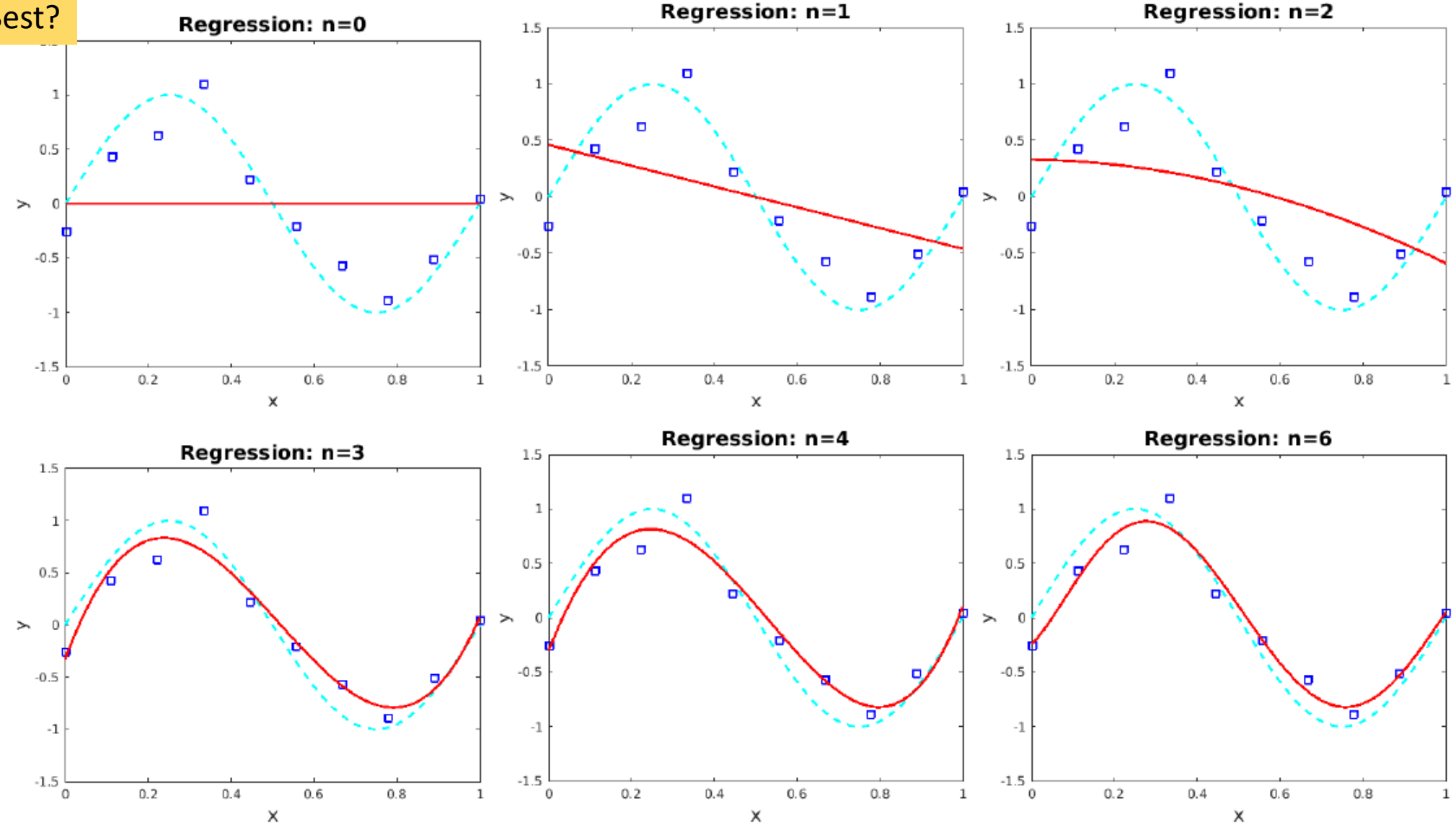


Consider 10 points generated from a sine function with noise



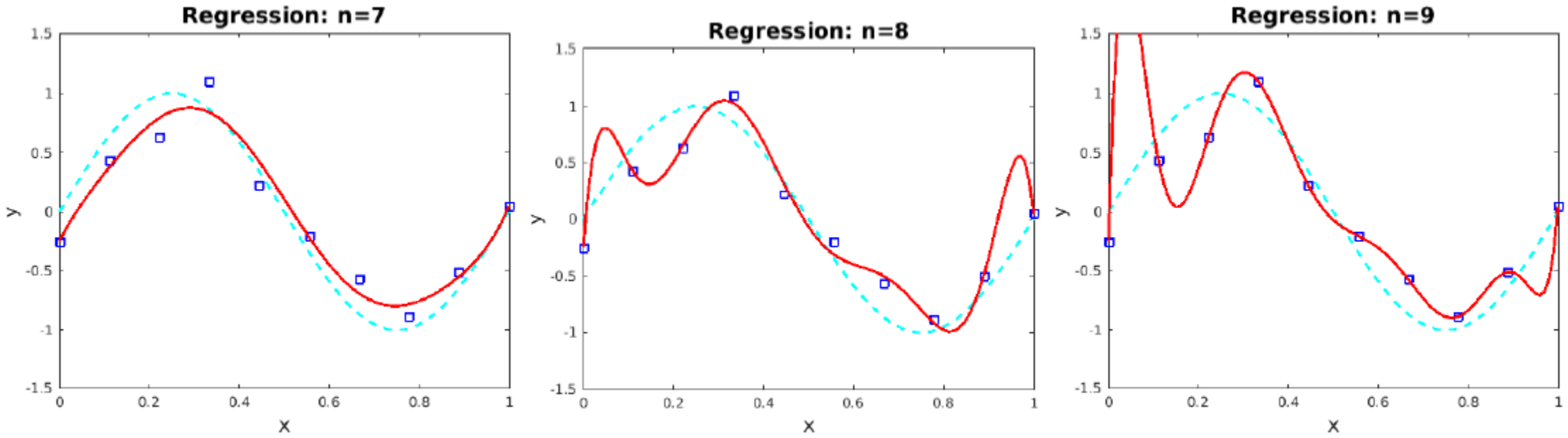
Simple Example

Which is Best?



Simple Example

Which is Best?



SSE or LSE

How do you measure it?

Given several models with similar explanatory ability, the simplest is most likely to be the best choice

Least Square Error or SSE

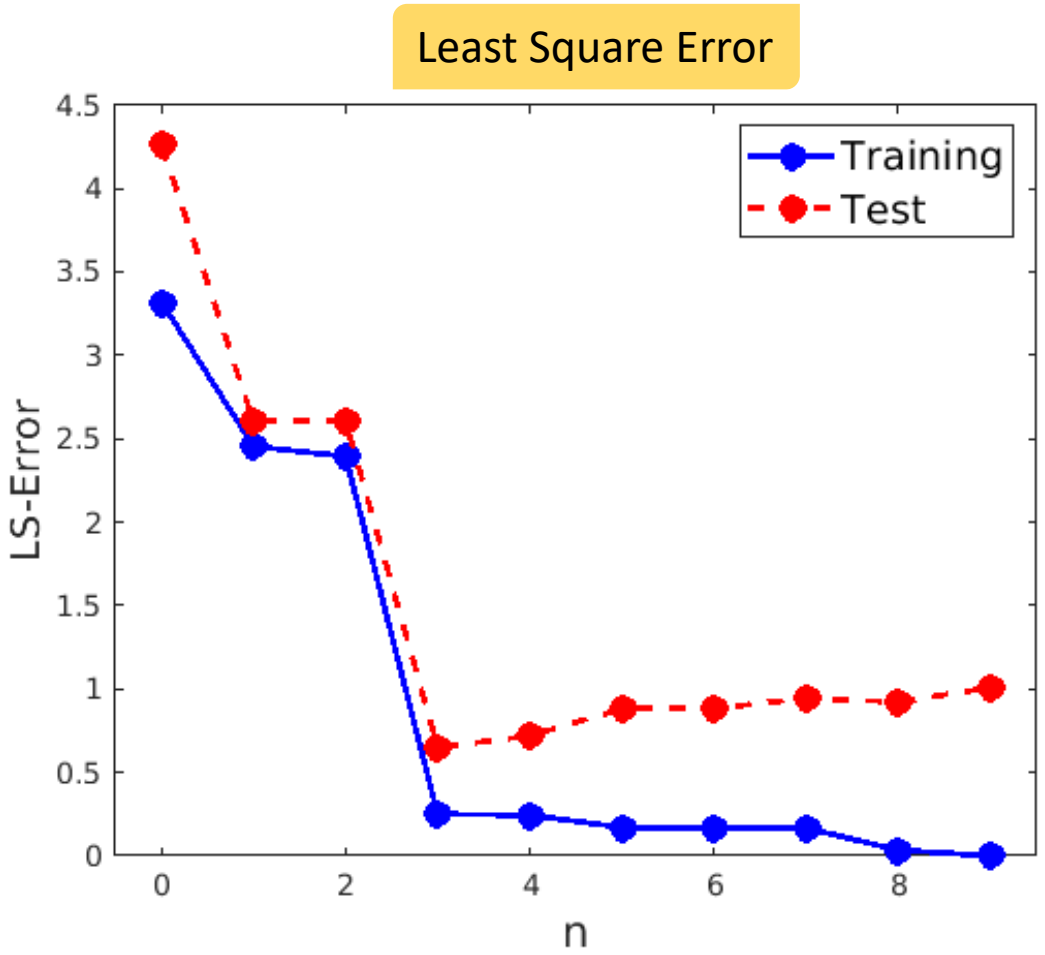
Given a dataset $\{(x_i, y_i) | i = 1,2,\cdots,m\}$ and the model P_n, define the LS Error as

$$E_n = \sum_{i=1}^{m} \left(y_i - P_n(x_i)\right)^2$$

It is also called the mean square error if we divide it by the sample size.

MS or LS Error

The best choice is P_3



Occam's Razor Principle

Law of Parsimony

One should not increase, beyond what is necessary, the number of entities required to explain anything

- When many solutions are available for a given problem, we should select the simplest one
- What do you mean by simple?
  - ❑ Use prior knowledge of the problem to solve to define what is a simple solution.

Binary Classifiers

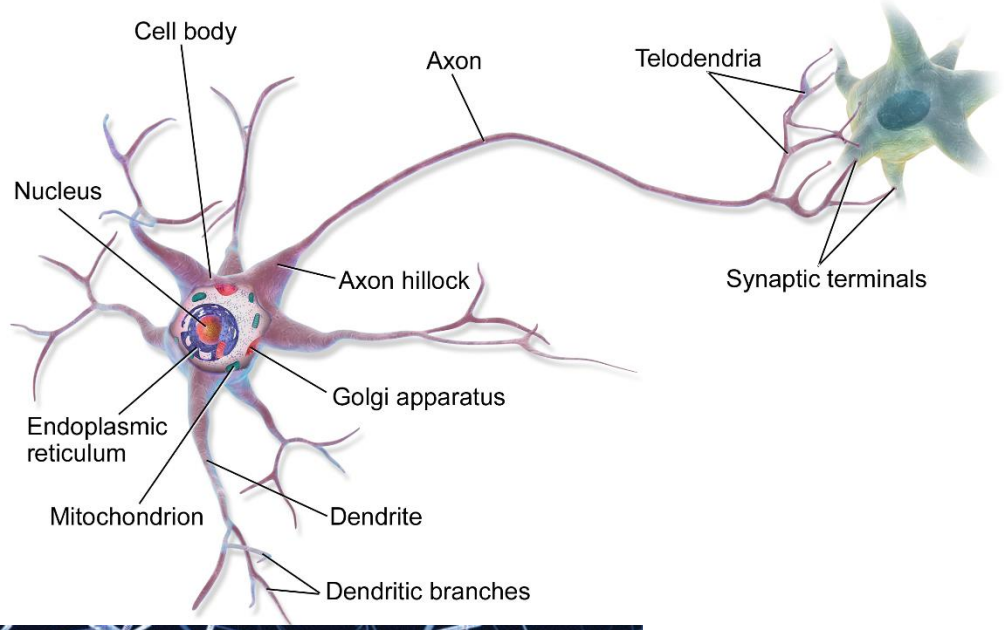
Binary Classifier

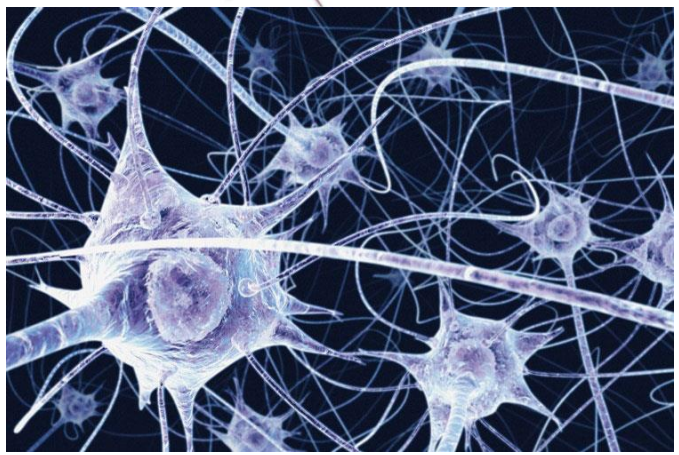
Binary Classifier

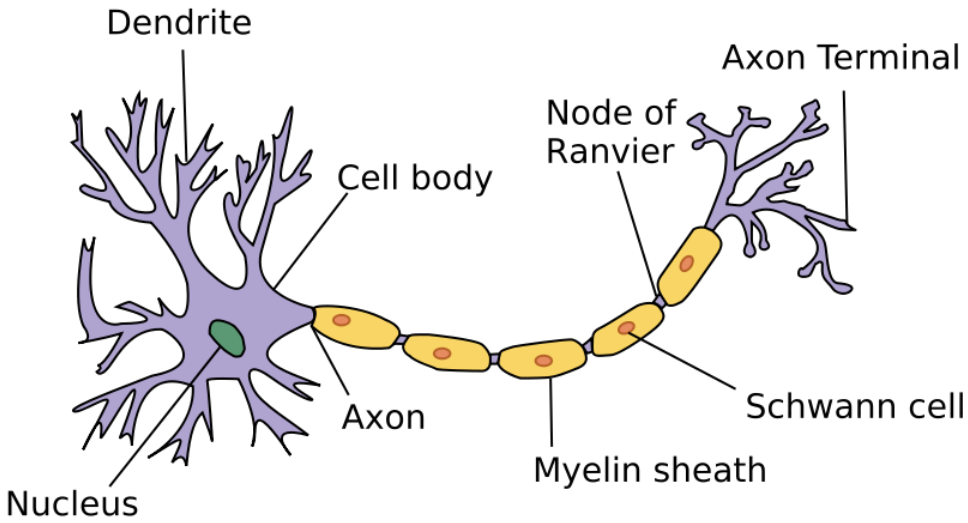
A function which can decide whether given input vector belongs to some specific class or not.

- It refers to those classification tasks that have two class labels
- A type of linear classifier
- A classification algorithm that makes its prediction based on a linear predictor function combining a set of weights with the feature vector
- Linear classifiers are artificial neurons

Human Brain: Mystique and Mystery



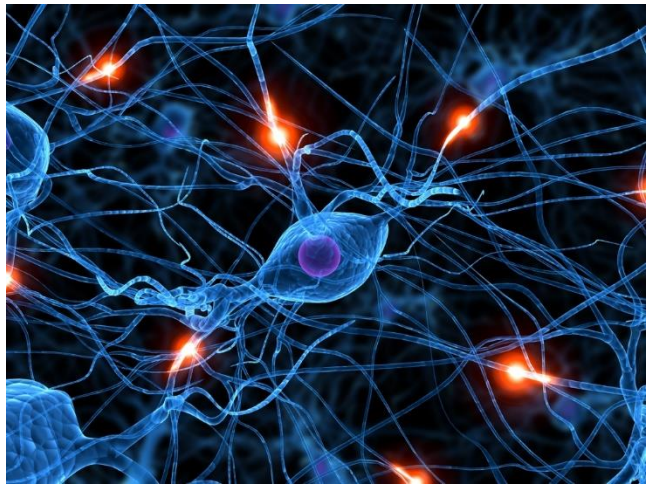


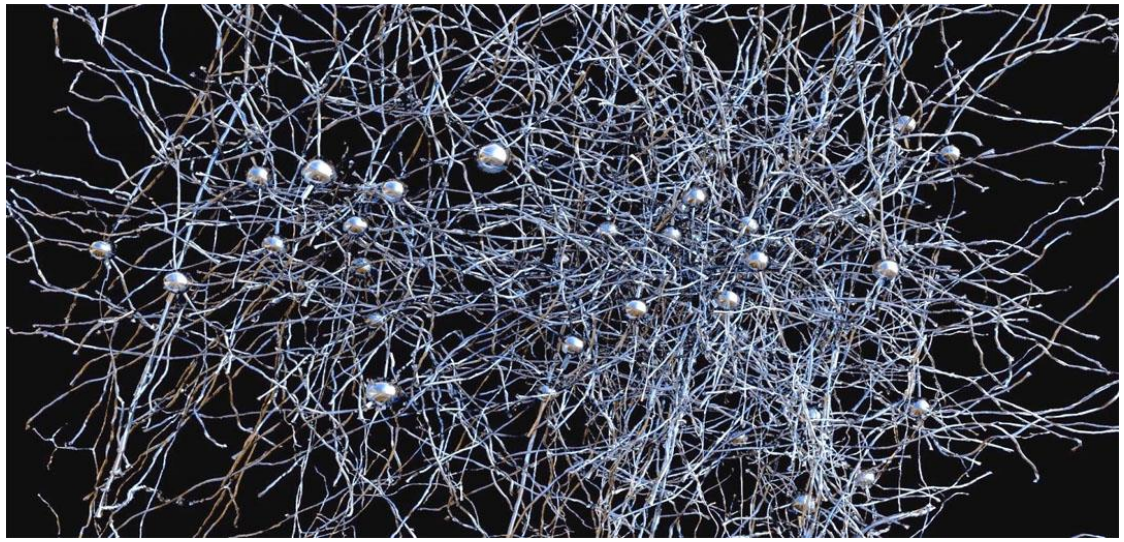


भारतीय प्रौद्योगिकी संस्थान तिरुपति

TIRUPATI

Human Brain: Mystique and Mystery





Linear Classifiers

Linear Classifier

As artificial neurons, Linear classifiers have the following characteristics

Inputs

- Feature Values x

Weight

- Each Feature has a weight w

Activation Function

- Weighted sum
- $\phi_w(x) = \sum_j w_j x_j = w^T x$

Decision/Output

- Class 1, if $\phi_w(x) > 0$
- Class 2, if $\phi_w(x) < 0$

Perceptron

Perceptron

Invented by Frank Rosenblatt (1957), Built on work of Hebbs (1949), Improved by Widrown-Hoff (1960),
Learning Methods for two-layer neural networks (1970)

Perceptron

Inputs

$$x = (x_1, x_2, \cdots, x_n)^T$$

Weight Vector

$$w = (w_1, w, \cdots, w_n)^T$$

Net Input

$$z = \sum_j w_j x_j = w^T x$$

Activation Function

$$\phi(z) = 1 \quad \text{If } z \geq \theta$$

$$\phi(z) = -1 \quad \text{otherwise}$$

θ is a threshold

Perceptron

A perceptron is a linear classifier that decides between two classes by drawing a straight line (or hyperplane) to separate them.

Perceptron

Inputs

$x = (1, x_1, x_2, \cdots, x_n)^T$

Weight Vector

$w = (w_0, w_1, w, \cdots, w_n)^T$

Net Input

$$z = w_0 + \sum_j w_j x_j = w^T x$$

Activation Function

$\phi(z) = 1$ If $z \geq 0$

$\phi(z) = -1$ otherwise

$w_0 = -\theta$, called bias in ML

Mathematical view of Perceptron

$$z = w_0 + \sum_j w_j x_j = w^T x$$

Let us take $n = 1$ and see, $z = w_0 + w_1 x_1 \Rightarrow y = ax + b$

The equation $z = w^T x$ represents a hyperplane in $\mathbb{R}^n$, whereas w_0 decides the intercept

What is unknown here?

1. Initialize weights to 0 or small random numbers
2. For each training sample x^i
   a) Find the output value $y^i = \phi(z^i)$
   b) Update the weights

Update Weight Vector

$$w = w + \Delta w, \Delta w = \eta\left(y^i - \bar{y}^i\right)x^i$$

η is the learning rate, $0 < \eta < 1$,
y^i is the true class label of the i^{th} training sample,
$\bar{y}^i$ is the predicted class label of the i^{th} training sample

What will be Δw?

1. If prediction is correct
2. What will be it if the prediction is wrong

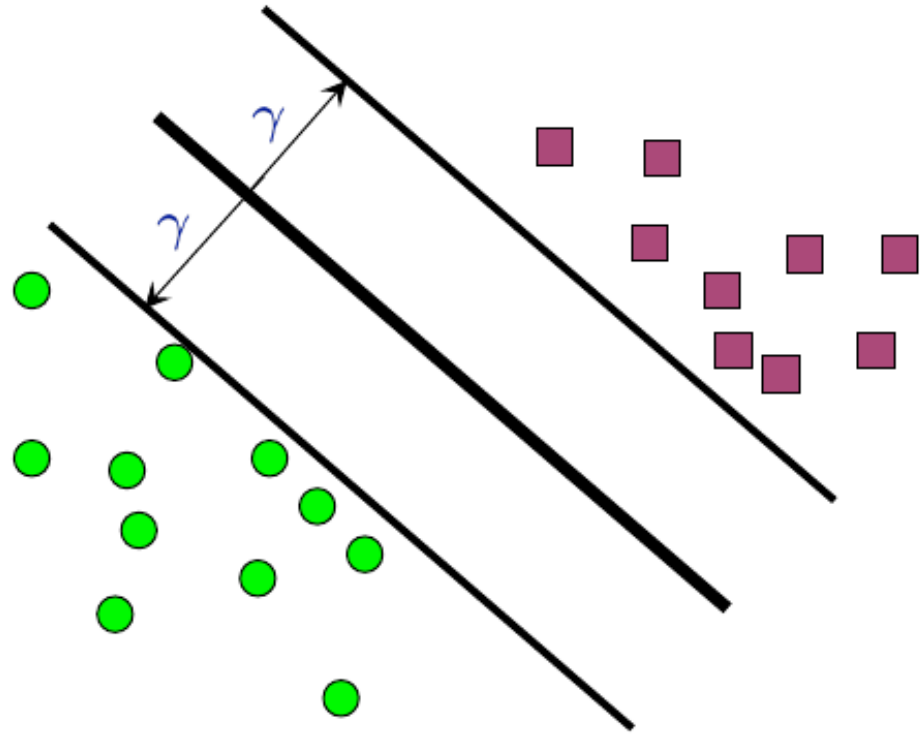
Separable Dataset

A dataset $\{(x^i, y^i)\}$ is linearly separable if there exists $\hat{w}$ and γ such that

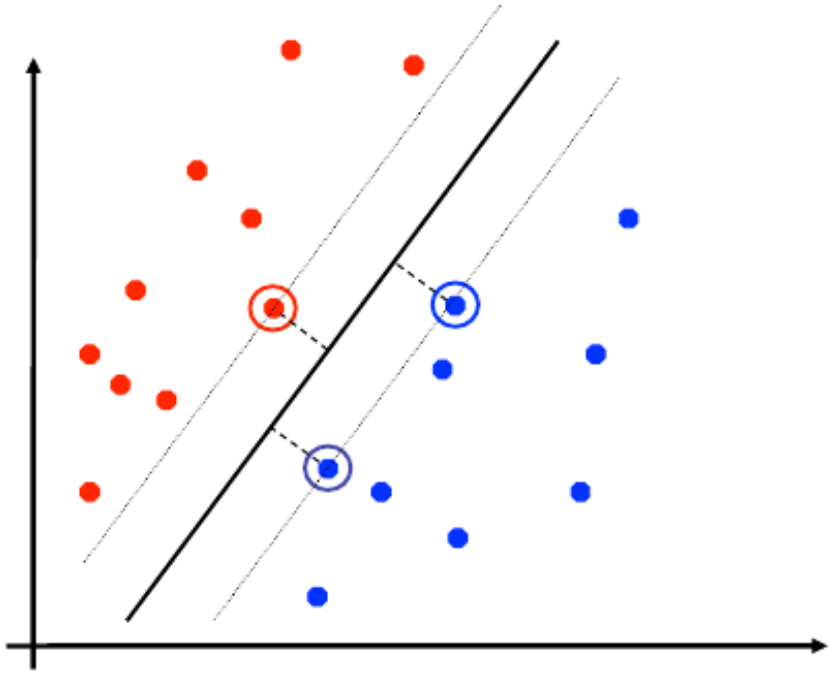
$$y^i \hat{w}^T x^i \geq \gamma > 0, \forall i$$

where γ is called the margin

Let X and Y be two sets of points in an $\mathbb{R}^n$. Then X and Y are linearly separable if there exists $w \in \mathbb{R}^n$ and $k \in \mathbb{R}$ such that every point $x \in X$ satisfies $w^T x > k$ and every point $y \in Y$ satisfied $w^T y < k$



SVM



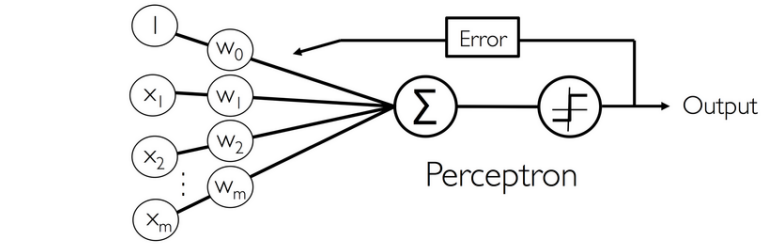
For linearly separable training dataset

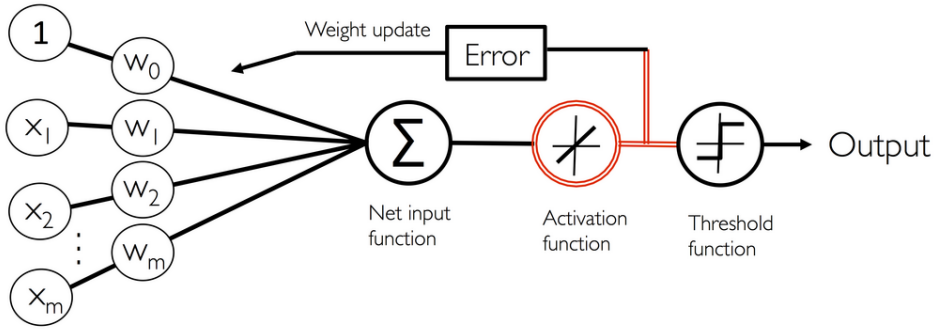
1. Perceptron always converge
2. **Separability:** Some weights get the training set perfectly correct

Support Vector Machine (SVM) chooses the linear separator with the largest margin.

Adaline Algorithm

1. Weights are updates based on $\phi(z)$
2. Suppose $\phi(z) = z$ (Identity Function)
3. This algorithm is interested to define a cost function and minimize it
4. Continuous cost function allow the ML optimization problem to Calculus Problem





Adaptive Linear Neuron (Adaline)

Adaline Learning

Given a dataset $\{(x^i, y^i), i = 1,2, \cdots, N\}$

Learn the weights w_i and bias $b = w_0$

Activation Function

$$\phi(z) = z$$

Cost Function (SSE)

$$\mathcal{J}(w, b) = \frac{1}{2}\sum_i \left(y^i - \phi(z^i)\right)^2$$

$$z^i = w^T x^i + b$$

Gradient Descent Method

Dominant algorithm for the minimization of the cost function

Compute $-\nabla\mathcal{J}$ for the search direction (update direction)

$$w = w + \Delta w = w - \eta\nabla_w\mathcal{J}(w,b)$$

$$b = b + \Delta b = b - \eta\nabla_b\mathcal{J}(w,b)$$

Where $\eta > 0$ is the step length (learning rate)

$$\Delta w = -\eta\nabla_w\mathcal{J}(w,b) = \eta\sum_i\left(y^i - \phi(z^i)\right)x^i$$

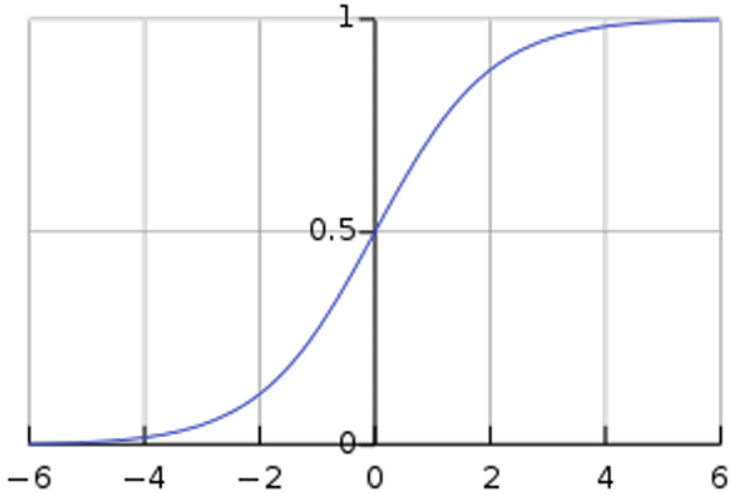
$$\Delta w = -\eta\nabla_b\mathcal{J}(w,b) = \eta\sum_i\left(y^i - \phi(z^i)\right)$$

Activation Function

$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

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

This helps to identify the probability of individual classes

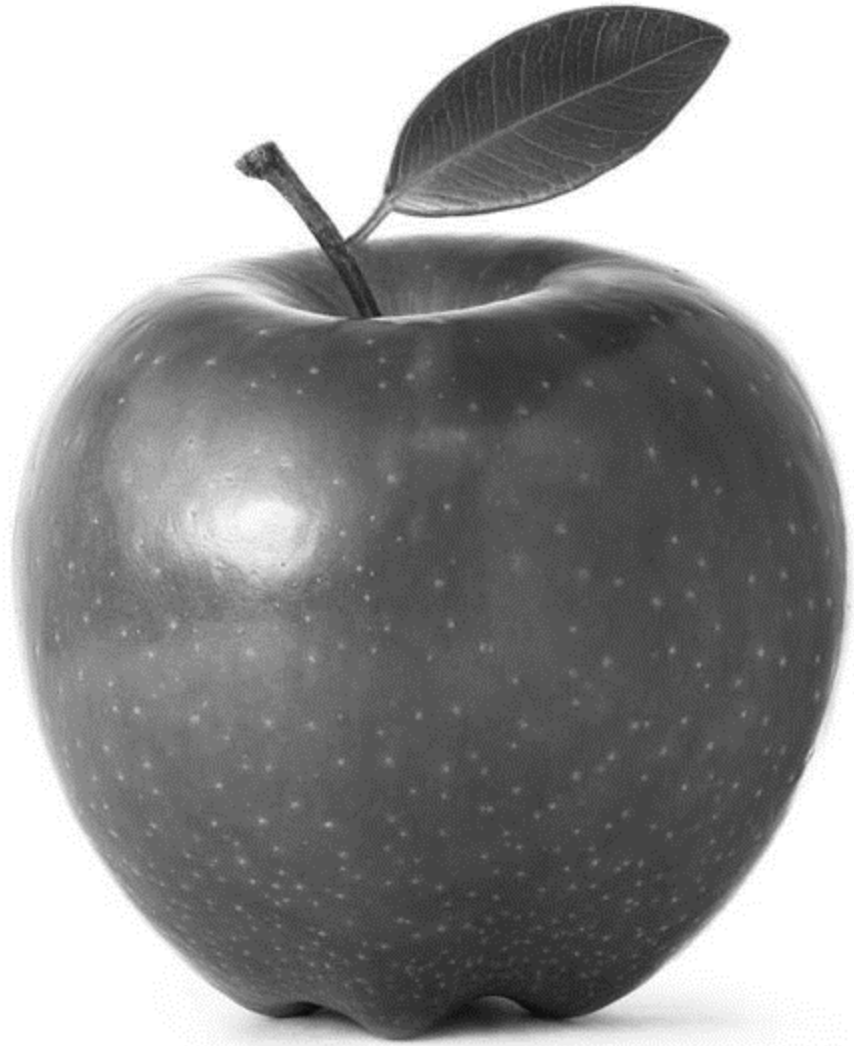


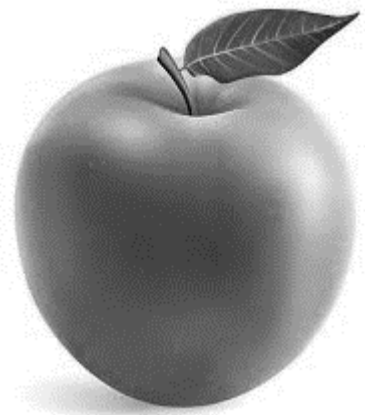
Unsupervised Learning

Unsupervised Learning

In unsupervised learning, we deal with unlabeled data or data of unknown structure. Using Unsupervised learning, we can explore the structure of our data to extract meaningful information without the guidance of a known outcome variable or reward function

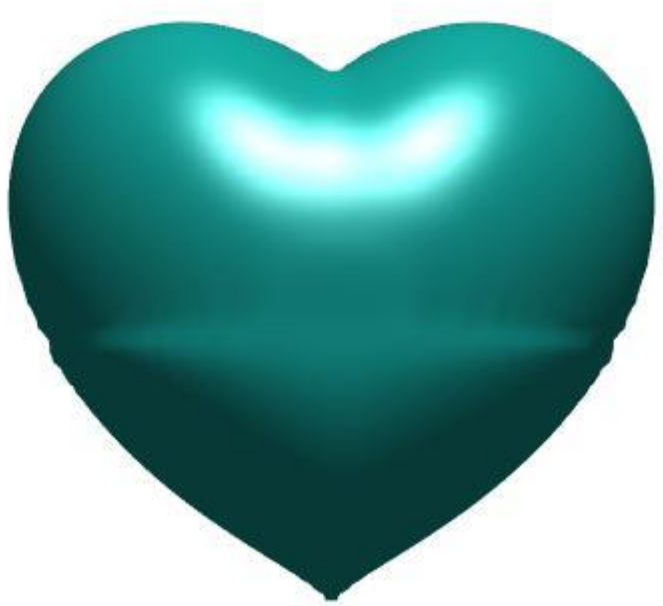
Source: Google Search, Unmanned Island





$$\left(x^2+\frac{9}{4}y^2+z^2-1\right)^3-x^2z^3-\frac{9}{80}y^2z^3=0$$

$$\left(x^2+\frac{9}{4}y^2+z^2-1\right)^3-x^2z^3-\frac{9}{80}y^2z^3=0$$



Unsupervised Learning

- **Assumption**: Given an unlabeled dataset $\{x_i\}$,
- **Unsupervised Learning:**
  - Given: Training Set $\{x_i | i = 1,2, \cdots, N\}$
  - Find a similar cluster or density estimation or dimensionality reduction

Unsupervised Learning: Clustering

Assumption: Given an unlabeled dataset $\{x_i\}$,

$$\min_{\mathcal{C}} \sum_{i} \sum_{c \in \mathcal{C}} \mathbb{I}(i,c) \|x_i - \mu_c\|^2$$

$\mathcal{C}$: set of clusters

$\mathbb{I}(i,c)$: indicator function

$$\mathbb{I}(i,c) = \begin{cases} 1 & x_i \in c \\ 0 & x_i \notin c \end{cases}$$

μ_c: Centroid of the cluster

Unsupervised Learning: Density Estimation

Assumption: Given a unlabeled dataset $\{x_i\}$, estimate the probability distribution (MLE)

$$\hat{p}(x) = \arg\max_{p(x)} \prod_i p(x_i)$$

$p(x)$: Probability density functions of the data

Find the distribution that maximizes the MLE.

Unsupervised Learning: PCA

Assumption: Given an unlabeled dataset $x_i \in \mathbb{R}^d$, reduce to a low-dimensional space $z_i \in \mathbb{R}^k$. Principal Component Analysis (PCA) can be formulated as finding the projection

$$z_i = W^T x_i$$

$W \in \mathbb{R}^{d \times k}$ is a projection matrix that maximizes the variance in the reduced space

$$\max_W \sum_i \|W^T x_i\|^2$$

Reinforcement Learning

Reinforcement Learning

It is the science of decision-making combining ML and Optimal Control.

- Learning the optimal behavior in a dynamic environment - maximum reward.
- Optimal behavior is learned through interactions with the environment and observations of how it responds
- No need for labeled input/output pairs
- In the absence of a supervisor, the learner must independently discover the sequence of actions that maximize the reward.
- This discovery process is similar to a trial-and-error search.

Reinforcement Learning

Agent: The learner or decision maker

Environment: The external system with which the agent interacts

State (s_t):The representation of the current system in the environment at time step t

Action (a_t):The action taken by the agent at time step t

Reward (r_t):The scalar feedback received after taking action (a_t) at time step t in state s_t

Policy $\pi: \mathcal{S} \rightarrow \mathcal{A}$, where $\mathcal{S}$ set of all states, $\mathcal{A}$ set of all actions

Value Function (V^{π}):Estimates how good a particular state

Action-Value Function (Q^{π}):Estimates the expected cumulative reward

Reinforcement Learning

Markov Decision Process

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$$

$\mathcal{S}$: Possible States

$\mathcal{A}$: Possible actions

$P(s_{t+1}|s_t, a_t)$: probability of moving from state s_t to s_{t+1} when action a_t is taken

r_t: reward function, immediate reward after taking action a_t

$\gamma \in [0,1]$: discount factor, helps to identify future rewards relative to immediate rewards

Bellman Optimality Equation

Value Function and Bellman Equation

$$V^{\pi}(s_t, a_t) = \mathbb{E}^{\pi}[r_t + \gamma V^{\pi}(s_{t+1})]$$

$$Q^{\pi}(s_t, a_t) = \mathbb{E}^{\pi}[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})]$$

$$V^{*}(s_t) = \max_{\pi} V^{\pi}(s_t) \;\; and \;\; Q^{*}(s_t, a_t) = \max_{\pi} Q^{\pi}(s_t, a_t)$$

Optimal Action-Value Function

$$Q^{*}(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \max_{a_{t+1}} Q^{*}(s_t, a_t)\right]$$

Self-supervised Learning

Self-supervised Learning

Self-supervised learning is a type of machine learning where a model learns from unlabeled data by creating its own supervision signal. In other words, the model generates pseudo-labels or uses part of the data to predict another part, which allows it to learn useful representations of the data without requiring human-provided labels.

Self-supervised Learning

- **Assumption**: Given a unlabeled dataset $\{x_i\}$,
- **Self-supervised Learning:**
  - Given: Training Set $\{x_i | i = 1,2,\cdots,N\}$
  - Define pretext task to generate a supervisory signal from the data
  - Corrupted or masked input x_i^m
  - Target $y_i = x_i$

Self-supervised Learning

- **Define $f(x)$** model (neural network) that learns the transformation of the input data xx into a useful representation.
- Learned embedding of the input x_i

$$\mathcal{L}(\theta) = \sum_i \mathcal{L}_{task}(f(x_i^m), y_i)$$

Physics Informed Neural Network

Panchatcharam Mariappan

Associate Professor

Department of Mathematics and Statistics,
IIT Tirupati

Supervised Learning

- Images → inputs
- Labels (“dog”, “rose”, “aeroplane”) → outputs
- Neural network → learns to map inputs to outputs by minimizing an error (difference between predicted and true labels).

Supervised Learning

- **What if I don’t have many labels, but I know the rules of the world — like Newton’s laws, or a heat equation?**

PINN

- **In supervised ML:**
  - ❑ **You train with data points and labels.**
- **In PINNs:**
  - ❑ **You train with data points (locations, times)**
  - ❑ **Instead of labels, you use physical laws (PDEs) as a *teacher.***

PINN

- In normal ML, the network learns from examples.
- In PINNs, the network learns from equations.

Analogy

Concept	ML (what they already know)	PINN (new link)
Input	Image pixels	Space-time coordinates (x, y, t)
Output	Label (dog/flower)	Field value (e.g., temperature, velocity, pressure)
Learning Source	Labeled data	Physics law (e.g., Navier–Stokes, diffusion)
Loss Function	Cross-entropy / MSE with labels	PDE residuals + boundary/initial condition losses
Goal	Classify correctly	Satisfy physics and match sparse data

Nutshell

Stage	Theme	Concept
Stage 1	**Learning from Labels**	**Supervised ML**
Stage 2	**Learning from Similarity**	**Unsupervised ML**
Stage 3	**Learning from Rules**	**PINN**

from **observing data** → **discovering structure** → **obeying laws of nature**.

Nutshell

“When you were children, you learned to identify dogs by seeing many of them — that was data-driven learning.
Later, you learned about bones, muscles, and motion — that’s physics.
PINNs are like scientists — they don’t just memorize; they reason with equations.”

PINN Introduction

PDEs

$$u_t = c^2 u_{xx}, x \in [0, 1], t > 0$$

- $u(x, t)$: temperature
- c^2: thermal diffusivity
- BCs: $u(0, t) = u(1, t) = 0$
- ICs: $u(x, 0) = sin\ (\pi x)$

Traditionally, we solve this with finite differences or FEM.
But what if a neural network could learn this solution?

Introduction

Ordinary Neural Network	Physics Informed Neural Network
Learns from data (x → y)	Learns from equations (e.g., PDEs)
Uses labeled examples	Uses physical laws
Needs lots of data	Can work with few data points
Input: Image → Output: Label	Input: (x, t) → Output: u(x, t)
Loss = (prediction - label)²	Loss = (PDE residual)² + BC/IC loss
Learns from data	Learns from Equations

Supervised Learning

- **Assumption**: Given a data set $\{(x_i, y_i)\}$, ∃ a relation $f: X \to Y$
- **Supervised Learning:**
  - Given: Training Set $\{(x_i, y_i) | i = 1,2, \cdots, N\}$
  - Find: $\hat{f}: X \to Y$ a good approximation to f

Mean Square Error or SSE

Given a dataset $\{(x_i, y_i) | i = 1,2, \cdots, m\}$ and the model P_n, define the LS Error as

$$E_n = \frac{1}{m} \sum_{i=1}^{m} \left(y_i - P_n(x_i)\right)^2$$

It is also called the mean square error (MSE)

Neural Network

$\hat{u}(x, t, \theta)$: Small fully-connected NN

Use Automatic Differentiation (AD)

$$f_\theta(x,t) = \frac{\partial \hat{u}}{\partial t} - c^2 \frac{\partial^2 \hat{u}}{\partial x^2}$$

Physics Loss:

$$L_{PDE} = MSE(f_\theta(x,t), 0)$$

BC Loss:

$$L_{BC} = MSE(\hat{u}(0,t), 0) + MSE(\hat{u}(1,t), 0)$$

Initial Loss:

$$L_{IC} = MSE(\hat{u}(x,0), \sin(\pi x))$$

Total Loss:

$$\boldsymbol{L = L_{PDE} + L_{BC} + L_{IC}}$$

How to Learn?

$\hat{u}(x,t,\theta)$: Small fully-connected NN

Use Automatic Differentiation (AD)

$$f_\theta(x,t) = \frac{\partial \hat{u}}{\partial t} - c^2 \frac{\partial^2 \hat{u}}{\partial x^2}$$

Physics Loss:

$$L_{PDE} = MSE(f_\theta(x,t), 0)$$

BC Loss:

$$L_{BC} = MSE(\hat{u}(0,t), 0) + MSE(\hat{u}(1,t), 0)$$

Initial Loss:

$$L_{IC} = MSE(\hat{u}(x,0), \sin(\pi x))$$

Total Loss:

$$\boldsymbol{L = L_{PDE} + L_{BC} + L_{IC}}$$

Level	New Concept	Example
Beginner	PDE residual only	Heat equation
Intermediate	Add ICs/BCs	Wave equation
Advanced	Add measured data	Wave tank, Navier–Stokes
Research	Domain decomposition, adaptive sampling	Real 2D/3D simulations

Libraries

Library	Use
PyTorch / TensorFlow	Custom PINN implementations
DeepXDE	Simplifies PINN coding for beginners
SciANN	TensorFlow-based PINN library
Modulus (NVIDIA)	For high-performance PDE PINNs (later stage)

Lecture Notes

1.SciML — Physics-Informed Neural Networks
Part of Scientific Machine Learning course lectures. KKS32 Courses
2.ETH Zurich — Deep Learning in Scientific Computing (Spring Semester 2023)
Lecture slides / applications of PINNs by Siddhartha Mishra, Ben Moseley. ETH Zürich
3.Oxford University — Physics Informed Neural Networks Course
Dept. of Computer Science, Oxford has a course page for PINNs. Department of Computer Science
4.Nature Article: Physics-Informed Machine Learning
5.Elsevier Article: Physics-informed machine learning: A comprehensive review on applications in anomaly detection and condition monitoring
6.Springer Article: **Physics-informed neural networks for PDE problems: a comprehensive review**

Lecture Notes

1. **IITU — "DL in Applied Mathematics" (Lecture on PINN)**
   Includes definitions, loss function discussions, advantages/disadvantages. IITU
2. **GitHub Tutorials**
   1. FilippoMB / Physics-Informed-Neural-Networks-tutorial (PyTorch notebook) GitHub
   2. AlirezaAfzalAghaei / PINN-tutorial (minimal PINN implementations in PyTorch) GitHub
   3. nguyenkhoa0209 / pinns_tutorial repository GitHub
3. **DeepChem Tutorial**
   Introductory tutorial on PINNs using JAX / other tools on DeepChem site. DeepChem
4. **MATHEMATICAL LAB / PINA Tutorial**
   Tutorial on multiscale PDEs using Fourier-Feature Networks with PINNs MathLab

Lecture Notes

1. **Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations** https://www.sciencedirect.com/science/article/pii/S0021999118307125 https://github.com/maziarraissi/PINNs
2. **SciML: Open Source Software for Scientific Machine Learning https://sciml.ai/**

A Few Fundamentals

Basics

This Part of the lecture is based on the paper

1. A Survey on Universal Approximation Theorems, Midhun T. Augustine, 2024
2. An elementary proof of a universal approximation theorem, Chris Monico, 2024
3. A visual proof that neural nets can compute any function
4. Approximation by superpositions of a sigmoidal function
5. Universality of deep convolutional neural networks
6. Recurrent Neural Networks Are Universal Approximators

NN or ANN

NN or ANN

A neural network (NN) or artificial neural network (ANN) is a network of artificial neurons arranged in layers.

Perceptron

The artificial neurons (also called perceptron) are inspired by biological neurons in biological neural networks (BNNs)

UATs

UATs

UATs are theorems associated with the approximation capabilities of NNs.
i.e., the ability of an NN to approximate arbitrary functions.

UATs

In general, UATs imply that NNs with appropriate parameters can approximate any continuous functions,
i.e. are generalized models that can represent complicated relationships in the data

Few Notations

- Let $\boldsymbol{x} \in \mathbb{R}^{\boldsymbol{n}}$. The p −norm or l^p −norm of a vector $\boldsymbol{x}$ is

$$\left|\left|\boldsymbol{x}\right|\right|_{\boldsymbol{p}} = (|\boldsymbol{x_1}|^{\boldsymbol{p}} + |\boldsymbol{x_2}|^{\boldsymbol{p}} + \cdots |\boldsymbol{x_n}|^{\boldsymbol{p}})^{\boldsymbol{1/p}}$$

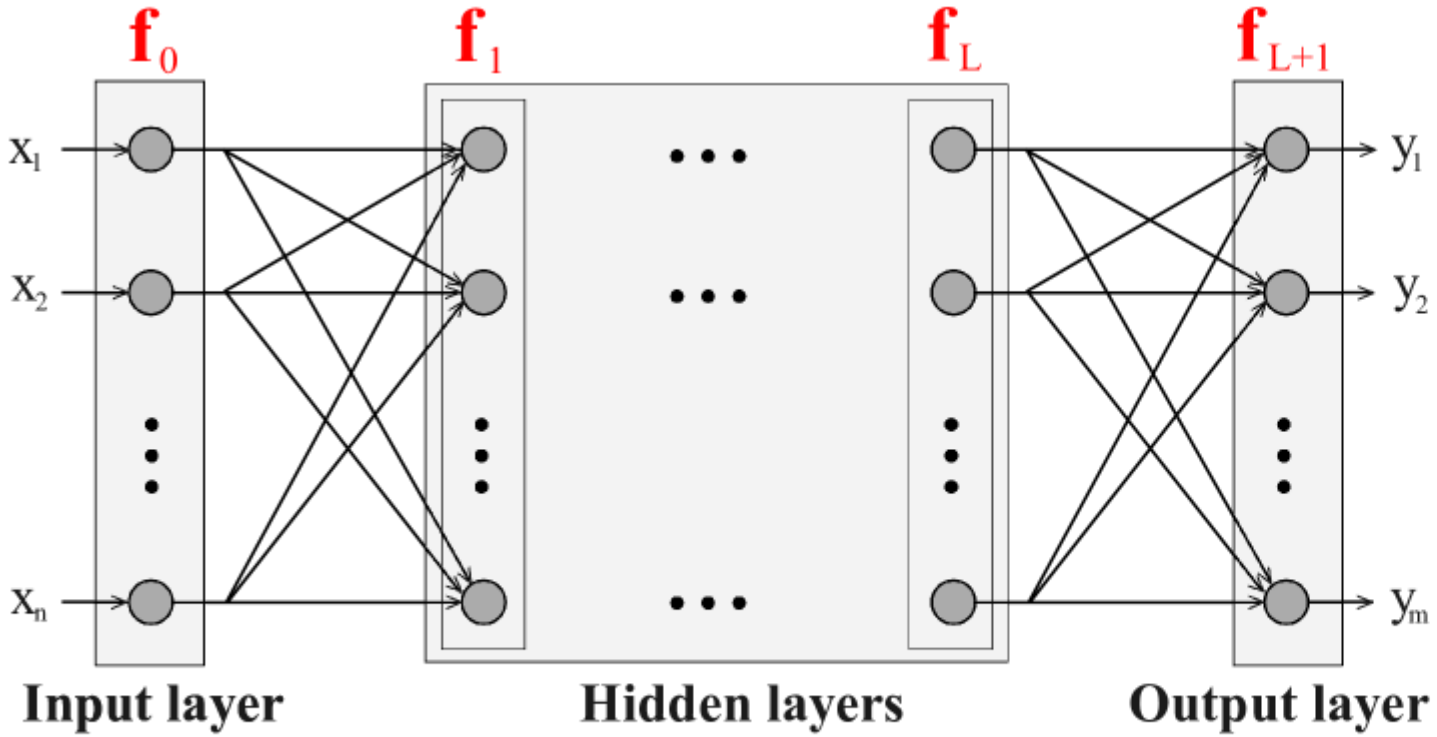
- The L^p norm of a function $f: X \to Y$ is defined as

$$\left|\left|f\right|\right|_p = \left(\int_X \left|\left|f(x)\right|\right|_p^p\right)^{\frac{1}{p}}$$

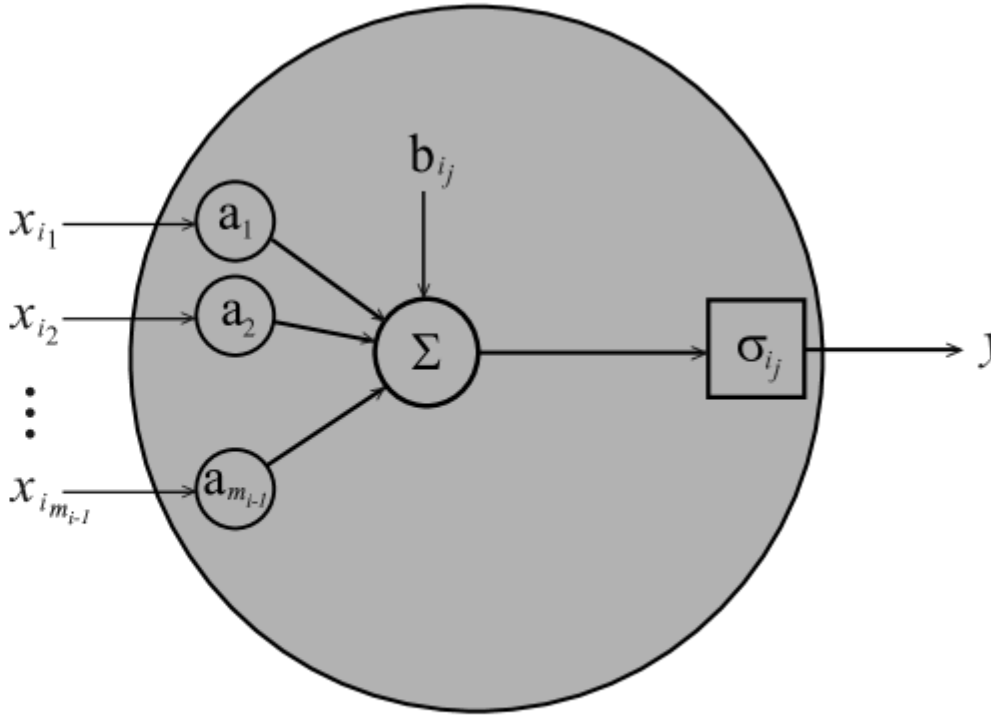
- A subset X of $\mathbb{R}^n$ is said to be compact if it is closed and bounded

Neural Network

- NN is a network of neurons arranged in layers



(a)



(b)

Neural Network

- NN is a network of neurons arranged in layers
- The layers are connected sequentially,

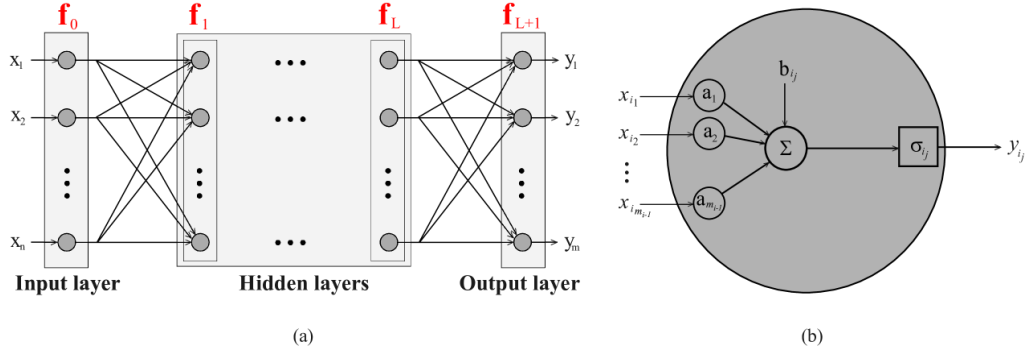
i.e., the output of each layer goes to the next layer as input.

- NNs can represented using composite functions of the form

$$\boldsymbol{y = f_{NN}(x) = f_{L+1}\left(f_L \cdots f_1\left(f_0(x)\right)\right)}$$

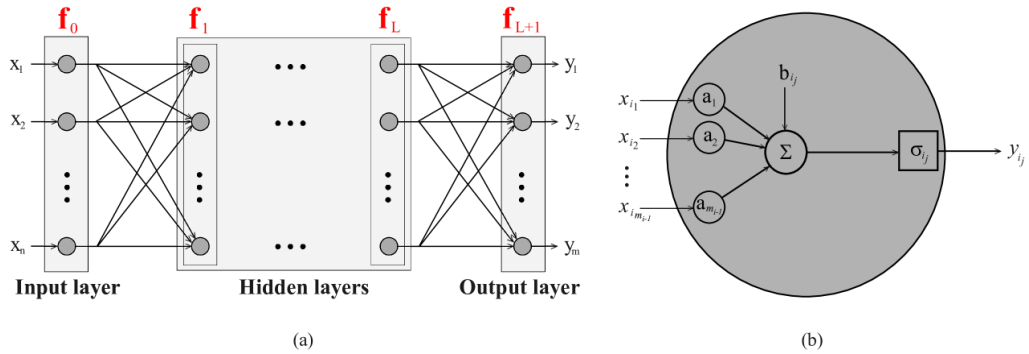
where $\boldsymbol{x} \in \mathbb{R}^n, y \in \mathbb{R}^m, \mathrm{f_{NN}}: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the NN function and $f_0, f_1, \cdots f_{L+1}$ are layers of the NN.

- f_0 is the input layer
- f_{L+1} is the output layer
- $f_1, f_2, \cdots f_L$ are hidden layers



Questions

- How many layers in total?
- If the number of hidden layers is L, then the total number of layers is _____



NN-Mathematically

- Input of the i^{th} layer be $\boldsymbol{x_i}$
- Output of the i^{th} layer be $\boldsymbol{y_i}$

$$\boldsymbol{y_i = f_i(x_i) = \sigma_i(A_i x_i + b_i)}$$

Where

$$\sigma_i = \left[\sigma_{i_1}, \sigma_{i_2}, \cdots \sigma_{i_{m_i}}\right]'$$

They are elementwise activation functions for the i^{th} layer, m_i is the number of neurons in i^{th} layer.

- $A_i \in \mathbb{R}^{m_i \times m_{i-1}}$ is the weight matrix
- $b_i \in \mathbb{R}^{m_i}$ is the bias vector
- $x_i \in \mathbb{R}^{m_{i-1}}$
- $y_i \in \mathbb{R}^{m_i}$

List of Activation Function

ReLU or Rectified Linear Unit

$$\sigma(x) = \max(0, x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$$

Step Function

$$\sigma(x) = \begin{cases} 0 & if\ x \leq 0 \\ 1 & if\ x > 0 \end{cases}$$

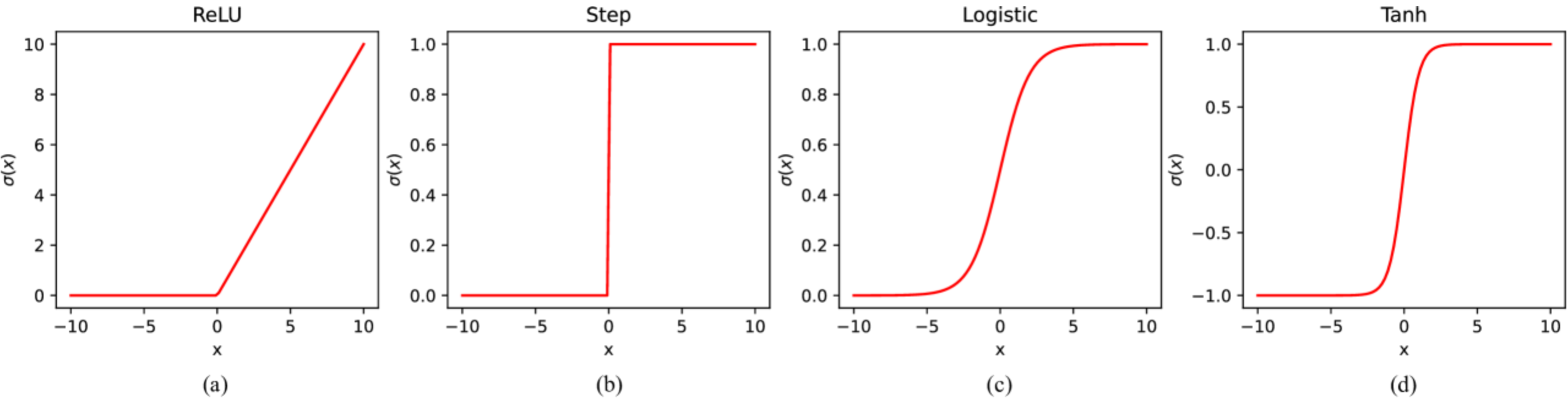
Logistic Function

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

Tanh Function

$$\sigma(x) = \tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

List of Activation Function



A few Terminologies

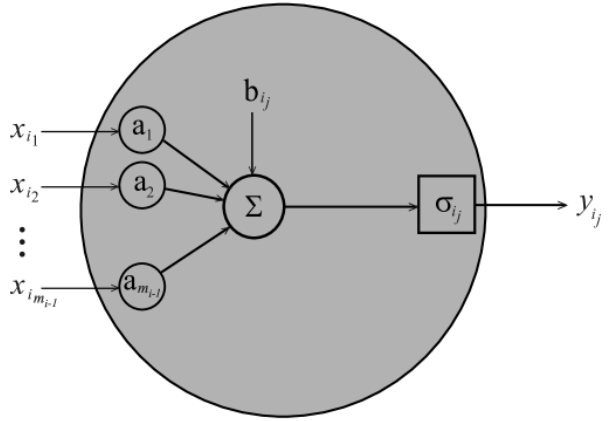
Neuron

Data processing units in NNs

$$y_{i_j} = \sigma_{i_j}(\boldsymbol{A_{i_j}x_i} + b_{i_j})$$

y_{i_j} is the output of the j^{th} neuron in the i^{th} layer.

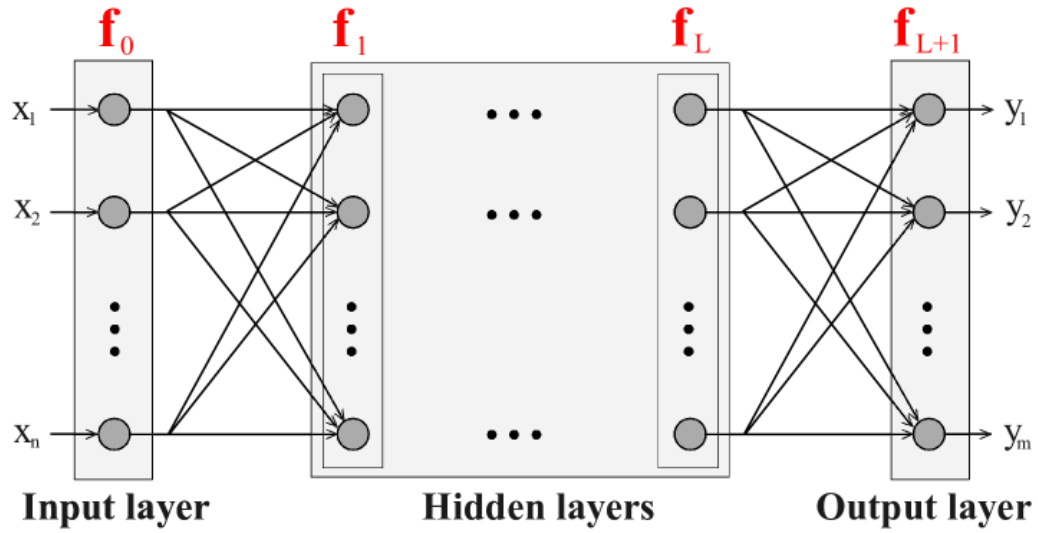
$A_{i_j} = [a_1\ a_2\ \ldots a_{m_{i-1}}]$ is the j^{th} row of the weight matrix A_i and b_{i_j} is the j^{th} element of the bias vector b_i



A few Terminologies

Layer

- A collection of neurons that takes the same inputs.
- In general, layers are vector-valued functions.
- Layers can grouped into three categories: input, output, and hidden layers.



(a)

Three Different Layers

Input

- each neuron has only one input
- it simply passes the input
- $\boldsymbol{\sigma_0(A_0x + b_0) = x}$

Hidden

- each neuron can have more than one input
- gives a scalar output
- σ : nonlinear.

Output

- each neuron can have more than one input
- gives a scalar output
- σ: linear for regression
  σ:nonlinear for classifciaton

Width, Depth, Shallow, Deep

Width

- maximum number of neurons in a layer
- $W = \max_i m_i$

Depth

- number of hidden layers plus output layer
- $D = L + 1$

Shallow NN

- NN with only one hidden layer
- $D = 2$

Deep NN

- NN with more than one hidden layer
- $D > 2$

Example

- One Input, One linear Output, One Hidden Layer with three Neuron

$$y = A_2\sigma_1(A_1x + b_1) + b_2$$

$$A_2 = [0.1\ 0.3\ 0.7], A_1 = \begin{bmatrix} 3 \\ -1 \\ 2 \end{bmatrix}, b_1 = \begin{bmatrix} -1 \\ 4 \\ 0 \end{bmatrix}, b_2 = 2, \sigma_1 = \tanh x$$

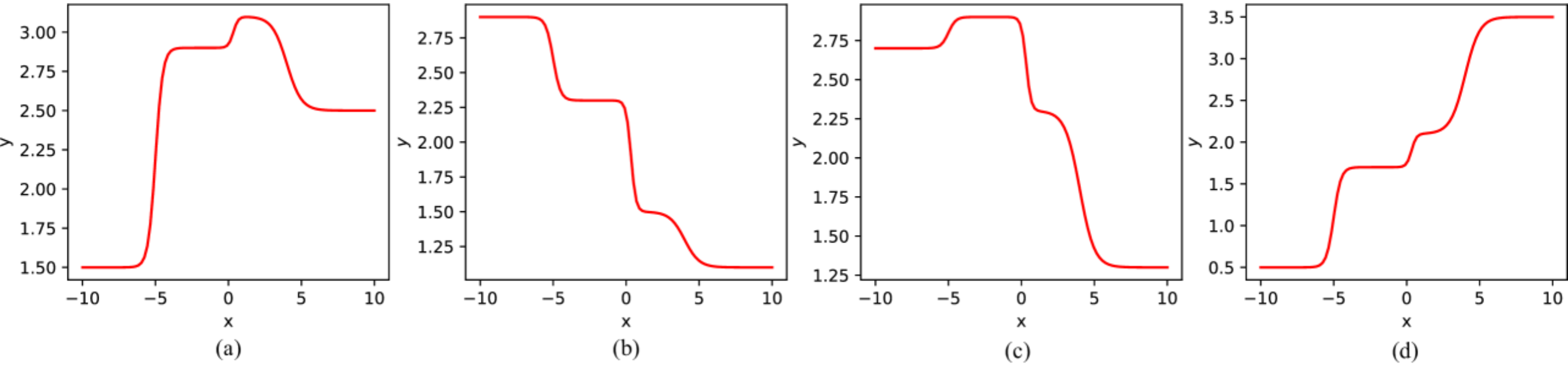
What is your y? Derive it

$$y = 0.1\tanh(3x - 1) + 0.3\tanh(4 - x) + 0.7\tanh(2x + 10) + 2$$

Change A_2 as follows

$$A_2 = [-0.4\ 0.2\ -0.3],$$
$$A_2 = [-0.3\ 0.5\ 0.1],$$
$$A_2 = [0.2\ -0.7\ 0.6],$$

Example


3.00
2.75
2.50
2.25
2.00
1.75
1.50
y
−10
−5
0
5
10
x
(a)
2.75
2.50
2.25
2.00
1.75
1.50
1.25
y
−10
−5
0
5
10
x
(b)
2.75
2.50
2.25
2.00
1.75
1.50
1.25
y
−10
−5
0
5
10
x
(c)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
y
−10
−5
0
5
10
x
(d)

Question

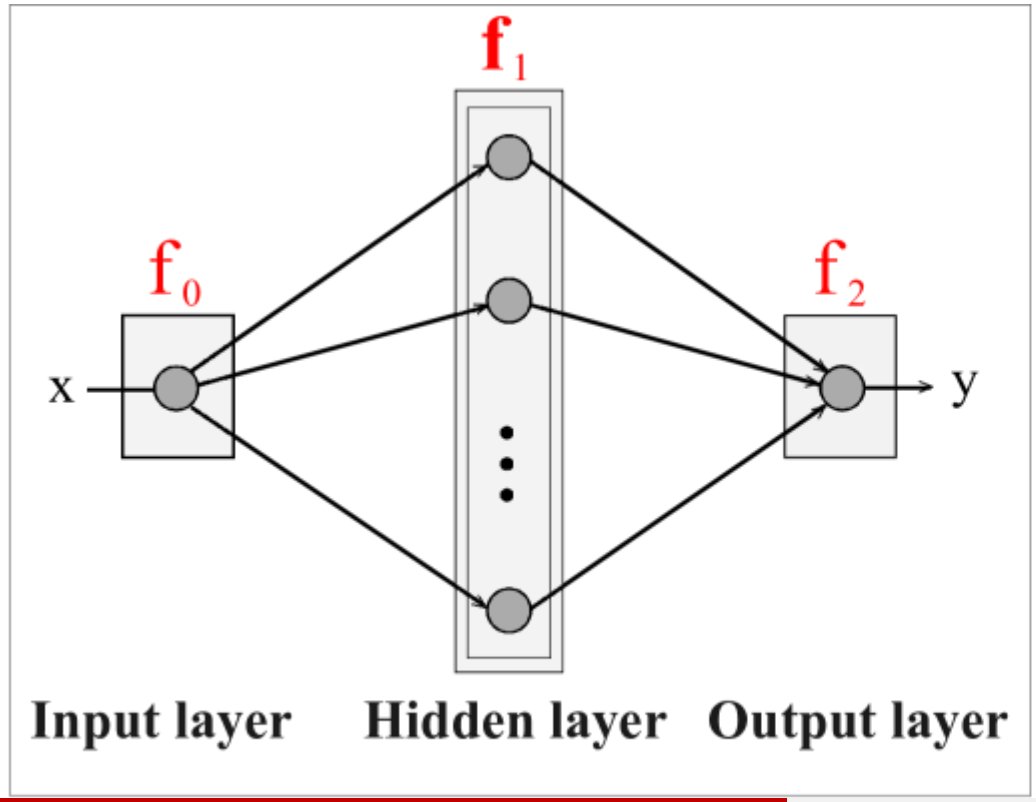
- These examples illustrate that we can represent complicated nonlinear relationships using NNs
- The main use of NNs is modeling relationships in the data, the question of the approximation capabilities of NNs gained interest naturally

Can we approximate any continuous function using NNs?

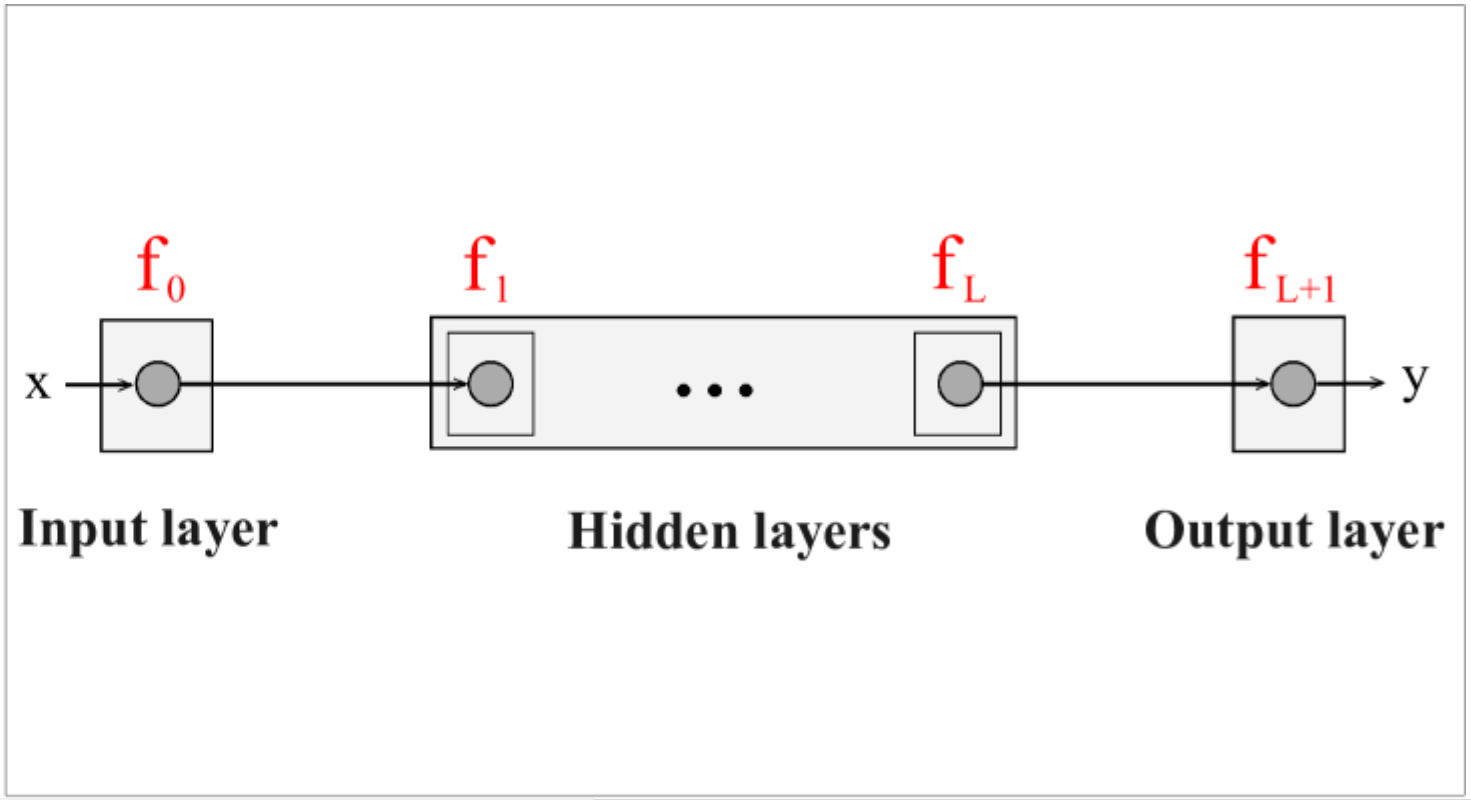
Universal Approximation Theorems (UATs)

Question

- Arbitrary width case
- Arbitrary depth case



(a)



(b)

Taylor's Theorem (1715)

Consider the continuous function $f: \mathbb{R} \to \mathbb{R}$. Let $a \in \mathbb{R}$ where f is N-times differentiable. Then f can be represented as a sum of polynomials

$$f(x) = f(a) + \sum_{n=1}^{N} \frac{f^{(n)}(a)(x-a)^n}{n!} + R_N(x)$$

Taylor's theorem can be considered as a mathematical foundation for linearization-based methods

Fourier (1807)

Consider the continuous and periodic function with period T. Then f can be represented as a sum of sinusoids

$$f(x) = A_0 + \sum_{n=1}^{N} A_n \cos\left(\frac{2\pi n x}{T} + \phi_n\right)$$

Here A_n is the amplitude and ϕ_n is the phase of the n^{th} harmonic component.

It can be extended to non-periodic functions as well →Fourier Transform

Weierstrass (1885)

Any continuous real-valued function $f: [a, b] \to \mathbb{R}$ can be approximated with a polynomial of

$$P_N(x) = \sum_{n=0}^{N} c_n x^n$$

such that

$$|f(x) - P_N(x)| < \epsilon$$

for any arbitrary $\epsilon > 0$

- Weierstrass Approximation theorem implies that any continuous function on a closed interval can be uniformly approximated by a polynomial function with arbitrary accuracy.
- The Weierstrass approximation theorem can be considered as a mathematical foundation for polynomial regression and interpolation methods.

Question (Hilbert)

Hilbert's 13th Problem

Can any continuous function of more than two variables be expressed as a superposition of finitely many continuous functions of two variables?

This problem was solved by Arnold and Kolmogorov in 1957 which resulted in the Kolmogorov-Arnold representation theorem

Kolmogorov-Arnold Representation Theorem

Any continuous real-valued function $f: [0,1]^n \to \mathbb{R}$ can be approximated with a polynomial of

$$f(\boldsymbol{x}) = f(x_1, x_2, \cdots, x_n) = \sum_{j=1}^{2n+1} \beta_j \left(\sum_{i=0}^{n} \alpha_{ij}\,(x_i) \right)$$

Where $\alpha_{ij}: [0,1] \to \mathbb{R}$ and $\beta_j: \mathbb{R} \to \mathbb{R}$

Summary

- Taylor → Approximation of differentiable function (N times) by a polynomial
- Fourier → Approximation of continuous function and periodic by sinusoids
- Weierstrass → Approximation of continuous function by polynomials
- Kolmogorov-Arnold → More than two variables as a superposition of finitely many continuous functions of two variables.

Universal Approximation Theorem – Width

- After the introduction of Neural Networks, approximation capabilities of sigmoid functions gained popularity, since in the initial versions of NNs sigmoid functions were used as activation functions

The depth or number of layers in the NN is considered to be bounded.

Examples

One Hidden Layer Neuron

$$y = ReLU(2x - 4)$$
$$y = ReLU(-x - 3)$$

Two Neurons

$$y = ReLU(2x - 4) + ReLU(-x - 3)$$

Three Neurons

$$y = 0.3ReLU(2x - 4) + 0.7ReLU(-x - 3) - 0.5ReLU(4x - 20)$$

ReLU or Rectified Linear Unit

$$\sigma(x) = \max(0, x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$$

Examples

One Hidden Layer Neuron

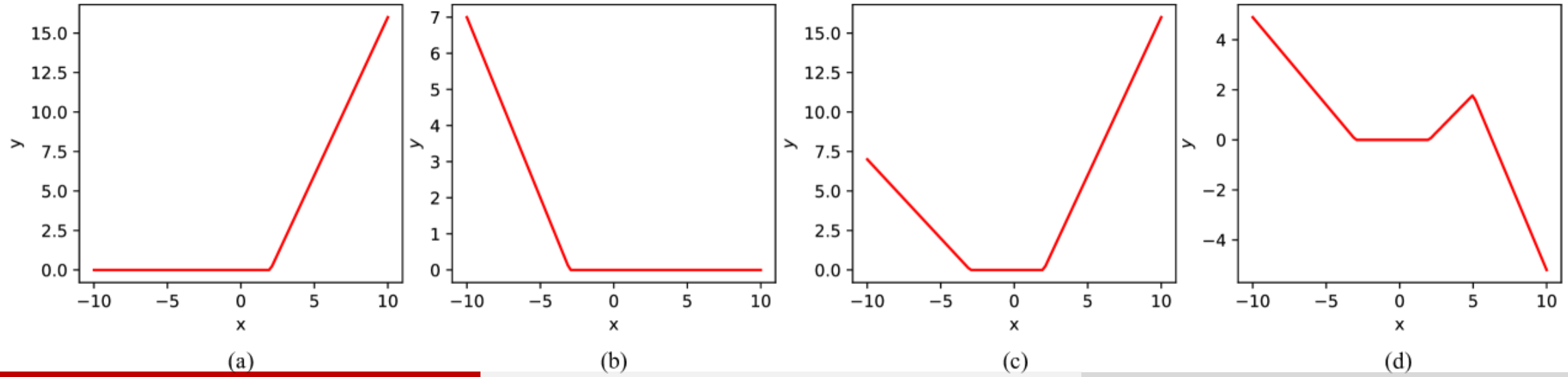
$$y = ReLU(2x - 4)$$
$$y = ReLU(-x - 3)$$

Two Neurons

$$y = ReLU(2x - 4) + ReLU(-x - 3)$$

Three Neurons

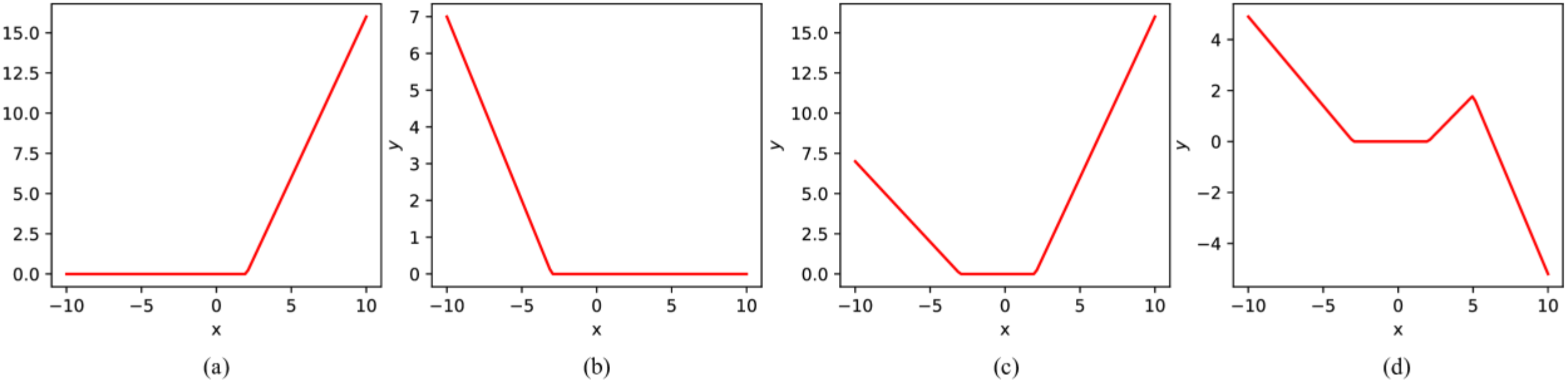
$$y = 0.3ReLU(2x - 4) + 0.7ReLU(-x - 3) - 0.5ReLU(4x - 20)$$



Questions

How many folds in two neurons?
How many folds in three neurons?

Number of folds in the curve increases with the number of neurons.



Pascanu et al (2013)

Consider the graph $\boldsymbol{f_{NN}(x)}$ which has folds along H hyperplanes defined linear functions $A_{1_j}x + b_{1_j} = 0, x \in \mathbb{R}^n, j = 1,2, \dots, H$. Then the number of linear pieces of f_{NN} **is**

$$\boldsymbol{n_l = \sum_{n=0}^{N} \binom{H}{n}}$$

UATs

- 1980s: UATs attempted to extend Kolmogorov-Arnold Representations
- Continuous functions can be arbitrarily approximated using NNs with two hidden layers and monotonic activation function [Cun, Farber]
- Arbitrary functions can be approximated by one hidden layer network with infinite number of neurons [Miyake]

UATs

- One hidden layer with a monotone cosine activation function can give Fourier series approximation to a given function as its output
  - ❖ Limited to only square-integrable functions on a compact set

The UATs are mostly t space can be approximated by f_{NN} .stated using the density of the functions $f_{NN}(X)$ generated by NNs within a given function space of interest
i.e. if $f_{NN}(X)$ is dense in a given space, then, any function in that space is approximated by f_{NN}

Funahashi, Hornick et al., and Cybenko, 1989

Let X be any compact subset of $\mathbb{R}^n$ and σ be any sigmoid activation function, then the sum of the form

$$f_{NN}(\boldsymbol{x}) = A_2\sigma(A_1x + b_1) = \sum_{j=1}^{m_1} \sigma_{2\,j}\sigma\left(A_{1\,j}x + b_{1_j}\right)$$

is dense in X.

Given any $f: X \to R$ and $\epsilon > 0$, there is a finite sum f_{NN} as above for which $|f - f_{NN}(x)| < \epsilon$ for all $x \in X$.

The above theorem means that NNs with one hidden layer and sigmoid activation function can approximate any continuous univariate function on a bounded domain with arbitrary accuracy

Leshno et al., 1993

Let X be any compact subset of $\mathbb{R}^n$ and σ be any sigmoid activation function, then the sum of the form

$$f_{NN}(\boldsymbol{x}) = A_2\sigma(A_1x + b_1) = \sum_{j=1}^{m_1} \sigma_{2\,j}\sigma\left(A_{1\,j}x + b_{1_j}\right)$$

is dense in X iff σ is not a polynomial function.

Similar theorems were developed for other activation functions such as ReLU, step, tanh, etc which are non-polynomial functions.

The above theorem means that NNs with one hidden layer and sigmoid activation function can approximate any continuous univariate function on a bounded domain with arbitrary accuracy

Universal Approximation Theorem – Depth

- The arbitrary depth case has attained a lot of research interest recently, especially after the introduction of deep learning as a separate domain in ML

Examples

One Hidden Layer Neuron

$$y = ReLU(0x + 5)$$
$$y = ReLU(2x + 4)$$

Two Hidden Layers

$$y = ReLU(-0.5ReLU(2x + 4) + 5)$$

Three Neurons

$$y = ReLU(-2ReLU(-0.5ReLU(2x + 4) + 5) + 3)$$

We assumed width $W = 1$

ReLU or Rectified Linear Unit

$$\sigma(x) = \max(0, x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$$

Examples

One Hidden Layer Neuron

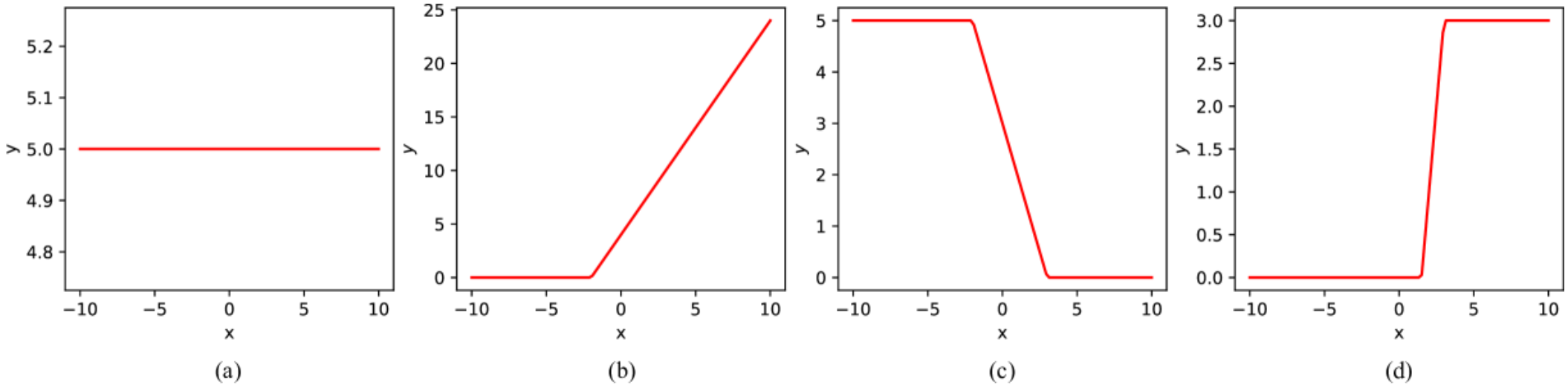
$$y = ReLU(0x + 5)$$
$$y = ReLU(2x + 4)$$

Two Hidden Layers

$$y = ReLU(-0.5ReLU(2x + 4) + 5)$$

Three Neurons

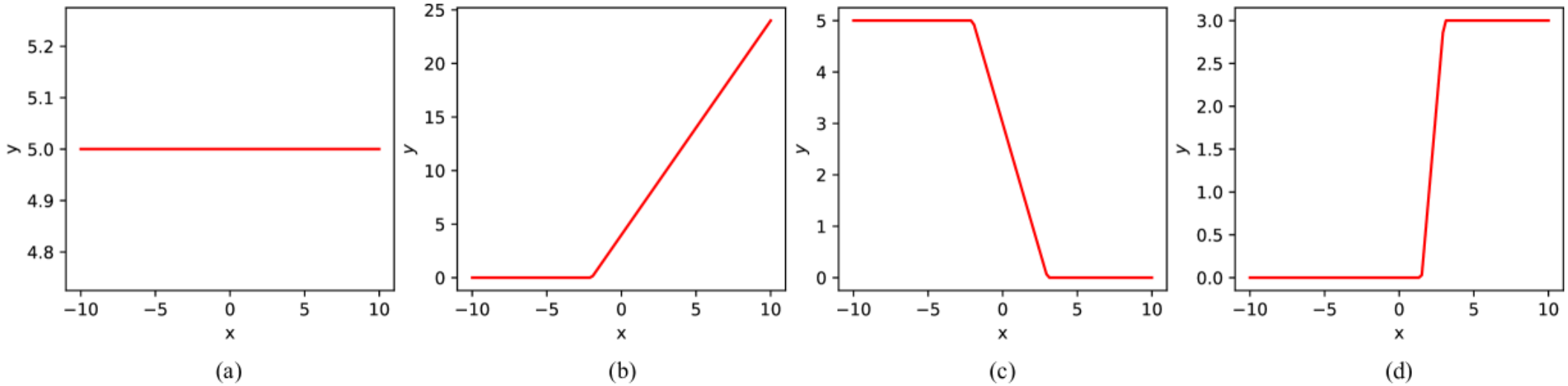
$$y = ReLU(-2ReLU(-0.5ReLU(2x + 4) + 5) + 3)$$



Examples

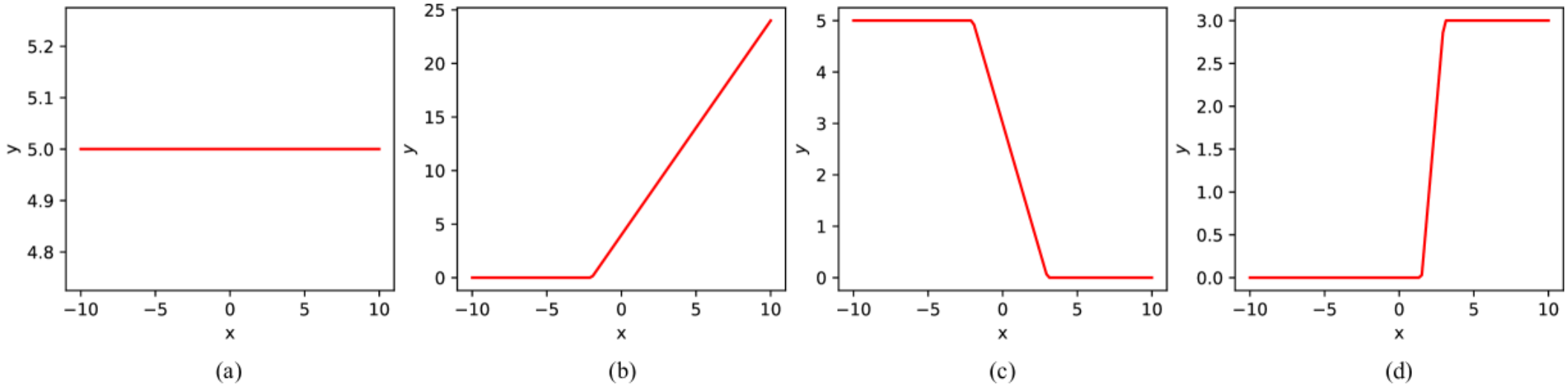
For NNs with one neuron per layer and ReLU activation function, the output can be either

- a straight line
- piecewise linear curve with one fold
- piecewise linear curve with two folds



Question

After two folds, can the subsequent layers add any further folds for one neuron per layer?
If so, how?



Lu et al., (2017)

Except for a negligible set, all functions f: $\mathbb{R}^n \rightarrow \mathbb{R}$ cannot be approximated by any ReLU network whose width $W \leq n$.

- Width-1 NNs can approximate only a small class of univariate functions
- The minimum width required for universal approximation should be greater than 1.

Lu et al., (2017)

Issue: *Finding the minimum width for universal approximation with deep NNs.*

Lu et al., (2017)

For any Lebesgue measurable function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ and $\epsilon > 0$, there exists a network f_{NN} width $W \leq n + 4$ with ReLU activation function which satisfies

$$\int |f(x) - \mathrm{f_{NN}(x)}| \mathrm{dx} < \epsilon$$

- NNs with arbitrary hidden layers and at most $n + 4$ number of neurons per layer can approximate any functions in a Lebesgue integrable space with sufficient accuracy

Park et al., (2017)

The minimum width required for universal approximation of

Lebesgue integrable functions $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is $max[n+1, m]$

- the minimum width required for universal approximation in deep NNs is $n+1$
- Width-2 NNs with a suitable activation function are also universal approximators for continuous univariate functions.

Proof of Universal Approximation Theorem

Notation

0-1 Squashing Function

Let $\sigma: \mathbb{R} \to \mathbb{R}$ be increasing, continuous and $\lim_{x \to -\infty} \sigma(x) = 0$ and $\lim_{x \to \infty} \sigma(x) = 1$

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

K be a compact subset of $\mathbb{R}^n$

Notation

$$\mathcal{N}_1 = \left\{ f \in C(K) : f(\mathrm{x}_1, \mathrm{x}_2, \cdots, x_n) = \sum_{i=0}^{n} a_i x_i \right\}$$

For some $a_0, a_1, \ldots, a_n \in \mathbb{R}$

$$\mathcal{N}_1^{\sigma} = \{ f \in C(K) : F = \sigma \circ f, \text{for some f} \in \mathcal{N}_1 \}$$

- $\mathcal{N}_1$ is the set of all affine functions of $x_1, x_2, \ldots, x_n$
- $\mathcal{N}_1^{\sigma}$ is the set of all possible node output functions in Layer 1

Notation

$$\mathcal{N}_{k+1} = \left\{ g \in C(K) : g = a_0 + \sum_{i=1}^{m} a_i F_i \right\}$$

For some $a_0, a_1, \ldots, a_n \in \mathbb{R}, F_i \in \mathcal{N}_k^{\sigma}$

$$\mathcal{N}_{k+1}^{\sigma} = \{ G \in C(K) : G = \sigma \circ \mathrm{g}, \text{for some } \mathrm{f} \in \mathcal{N}_{k+1} \}$$

- $\mathcal{N}_{k+1}^{\sigma}$ is the set of all possible node output function in Layer $k+1$
- $\mathcal{N}_k^{\sigma} \subset \mathcal{N}_{k+1}$
- If $g_1, g_2 \in \mathcal{N}_k$, then $a_0 + a_1 g_1 + a_2 g_2 \in \mathcal{N}_k$ for all $a_0, a_1, a_2 \in \mathbb{R}$

Separation Lemma

Let us show that σ separates points in $\mathbb{R}^n$ in a strong sense

Let x_0 and x_1 be distinct real numbers. For each $\epsilon > 0$, there exist $s, t \in \mathbb{R}$ such that $\sigma(s + tx_0) < \epsilon$ and $\sigma(s + tx_1) > 1 - \epsilon$. If in addition, $x_0 < x_1$ and $\epsilon < 1/2$, then $\sigma(s + tx) < \epsilon$ on the interval $(-\infty, x_0]$ and $\sigma(s + tx) > 1 - \epsilon$ on the interval $[x_1, \infty)$

Separation Lemma

Let x_0 and x_1 be distinct real numbers. For each $\epsilon > 0$, there exist $s, t \in \mathbb{R}$ such that $\sigma(s + tx_0) < \epsilon$ and $\sigma(s + tx_1) > 1 - \epsilon$. If in addition, $x_0 < x_1$ and $\epsilon < 1/2$, then $\sigma(s + tx) < \epsilon$ on the interval $(-\infty, x_0]$ and $\sigma(s + tx) > 1 - \epsilon$ on the interval $[x_1, \infty)$

Proof [First Part]:

WLOG: $\epsilon < 1$. By 0-1, squashing function, there exist $y_0, y_1 \in \mathbb{R}$ such that $\sigma(y_0) = \frac{\epsilon}{2}$ and $\sigma(y_1) = 1 - \frac{\epsilon}{2}$.

$$\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}$$

When does it have a solution? Does the condition satisfied from our hypothesis? Can you fill the remaining proof?

Separation Lemma

Let x_0 and x_1 be distinct real numbers. For each $\epsilon > 0$, there exist $s, t \in \mathbb{R}$ such that $\sigma(s + tx_0) < \epsilon$ and $\sigma(s + tx_1) > 1 - \epsilon$. If in addition, $x_0 < x_1$ and $\epsilon < 1/2$, then $\sigma(s + tx) < \epsilon$ on the interval $(-\infty, x_0]$ and $\sigma(s + tx) > 1 - \epsilon$ on the interval $[x_1, \infty)$

Proof [Second Part]:

$x_0 < x_1$ and $\epsilon < \frac{1}{2}$. σ is monotone and

$$\sigma(s + tx_0) < \epsilon < \frac{1}{2} < 1 - \epsilon < \sigma(s + tx_1)$$

$\sigma(s + tx)$ is increasing. Hence the lemma

Separation Lemma

Let us show that σ separates points from closed sets using functions in $\mathcal{N}_2$ in strong sense. i.e, one layer is sufficient to separate points from closed sets.

Let $B \subset K$ be a closed set and $\boldsymbol{x_0} \in K - B$. For each $\epsilon > 0$, there exists $g \in \mathcal{N}_2$ such that $g > 1 - \epsilon$ on B and $g(\boldsymbol{x_0}) < \epsilon$.

Using a 2-hidden-layer network, we can **separate** one outside point from an entire closed set - do a "soft indicator" that is near 0 at $x_0$and near 1 on B

Separation Lemma

Let $B \subset K$ be a closed set and $\boldsymbol{x_0} \in K - B$. For each $\epsilon > 0$, there exists $g \in \mathcal{N}_2$ such that $g > 1 - \epsilon$ on B and $g(\boldsymbol{x_0}) < \epsilon$.

Proof:

WLOG: $\epsilon < \frac{1}{3}$. Let $\boldsymbol{b} \in B \Rightarrow \boldsymbol{b} \neq \boldsymbol{x_0} \Rightarrow \exists f_b \in \mathcal{N}_1$ s.t. $f_b(\boldsymbol{x_0}) < \frac{\epsilon}{2}$ and $f_b(x_0) > 1 - \frac{\epsilon}{2}$

$$U_b = \{\boldsymbol{x} \in K : f_b(\boldsymbol{x}) > 1 - \epsilon\}$$

f_b is continuous $\Rightarrow U_b$ is open and $b \in U_b$. $\{U_b\}_{b \in B}$ is an open cover of the compact set B.

Therefore, $\exists$ a finite subcover $\{U_{b_1}, U_{b_2}, \ldots, U_{b_n}\}$ that covers B

By above lemma, $\exists s, t \in \mathbb{R}, s.t. \sigma(s + tx) < \epsilon/N$ on $(-\infty, \epsilon)$ and $\sigma(s + tx) > 1 - \epsilon$ on $(1 - \epsilon, \infty)$

Define $F_j = \sigma\left(s + t f_{b_j}\right) \Rightarrow F_j \in \mathcal{N}_1^{\sigma}, F_j(x_0) < \epsilon/N$ and $F_j > 1 - \epsilon$ on U_{b_j}

Define $g = \sum_j^N F_j$, proof follows immediately.

Separation Lemma

Let us show that two hidden layers suffice to separate disjoint closed sets, in a sense very similar to the previous lemma.

Let A and B be disjoint closed subsets of K. Then for each $\epsilon > 0$

1. $\exists h \in \mathcal{N}_3$ such that $h < \epsilon$ on B and $h > 1 - \epsilon$ on A
2. $\exists\, H \in \mathcal{N}_3^{\sigma}$ such that $0 \leq H < \epsilon$ on B and $1 - \epsilon < H \leq 1$ on A

Separation Lemma

Let A and B be disjoint closed subsets of K. Then for each $\epsilon > 0$

1. $\exists h \in \mathcal{N}_3$ such that $h < \epsilon$ on B and $h > 1 - \epsilon$ on A
2. $\exists\, H \in \mathcal{N}_3^{\sigma}$ such that $0 \leq H < \epsilon$ on B and $1 - \epsilon < H \leq 1$ on A

Proof:

WLOG: $\epsilon < \frac{1}{3}$. Let a $\in A$, by above lemma, $\exists \widetilde{g_a} \in \mathcal{N}_2$ such that $\widetilde{g_a} > 1 - \frac{\epsilon}{2}$ on B and $\widetilde{g_a}(a) < \frac{\epsilon}{2}$. Let $g_a = 1 - \widetilde{g_a} \Rightarrow g_a \in \mathcal{N}_2$ and $g_a < \frac{\epsilon}{2}$ on B and $g_a(a) > 1 - \frac{\epsilon}{2}$

$$U_a = \{x \in K : g_a(x) > 1 - \epsilon\}$$

g_a is continuous$\Rightarrow U_a$ is open. Since $a \in U_a \Rightarrow \{U_a\}_{a \in A}$ is an open cover of A $\Rightarrow$Finite subcover for A (Why?)