Machine Learning Methods for Atmosphere, Ocean, and Climate Science

Lecture 1: Machine Learning Fundamentals

Mathematical modeling of Climate, Ocean, and Atmosphere processes
International Centre for Theoretical Sciences, TIFR, Bengaluru, India

Aman Gupta

Stanford University

DATAWAVE

What this lecture series is …

✔ An introductory treatise to implementing Deep Learning (DL) algorithms

✔ ⟶ Develop an intuitive understanding of DL fundamentals
⟶ Code simple and functional neural nets
⟶ Stroll through ongoing ML research in climate science

✔ Focus on practical implementation more so than on theoretical derivations

What this lecture series is …

- ✔ An introductory treatise to implementing Deep Learning (DL) algorithms
- ✔ ⟶ Develop an intuitive understanding of DL fundamentals
  ⟶ Code simple and functional neural nets
  ⟶ Stroll through ongoing ML research in climate science
- ✔ Focus on practical implementation more so than on theoretical derivations

What this lecture series is not …

- ✘ A comprehensive set of lectures with theoretical derivations of the machine learning algorithms and their components
- ✘ Deploying complex ML models into existing software architectures
- ✘ We will not be creating a new ChatGPT like bot or a Dall-E like image generator :-)

Lecture 1

- Parametric estimation
- Introduction to deep neural networks
- The training algorithm

Lecture 2

- The PyTorch library
- Implementing Artificial Neural Nets in PyTorch

PyTorch

Lecture 3

- Implementing Convolutional Neural Networks in PyTorch

Lecture 4

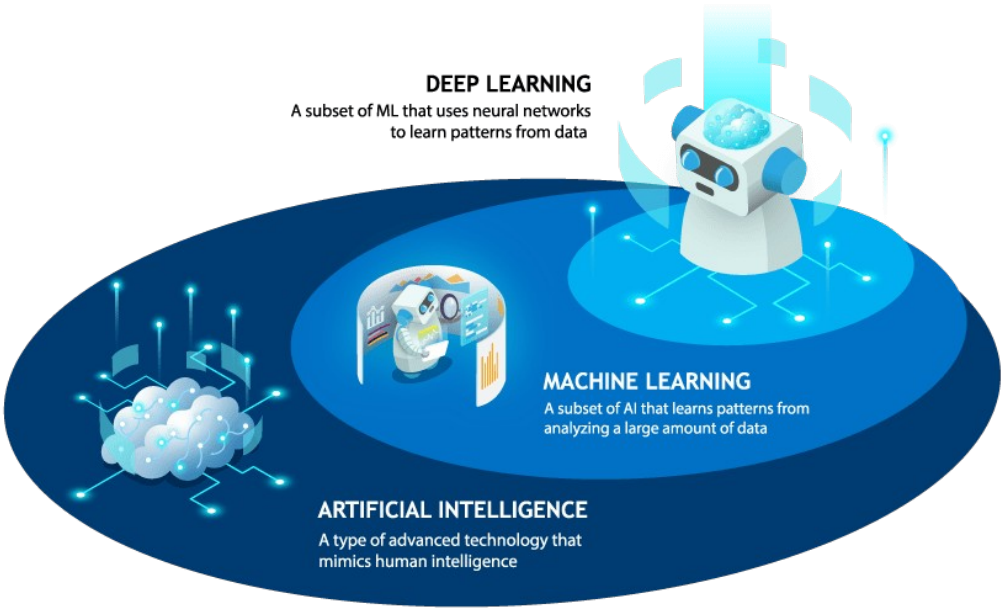
- Applications of ML in climate science

Machine Learning & Artificial Intelligence

Machine learning: is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

Focus on training the model rather than programming using explicit code.
⟶ Paint a picture of a ship
⟶ Differentiating between a dog and a house
⟶ Create a song
⟶ Write a novel
⟶ Forecast the weather for tomorrow

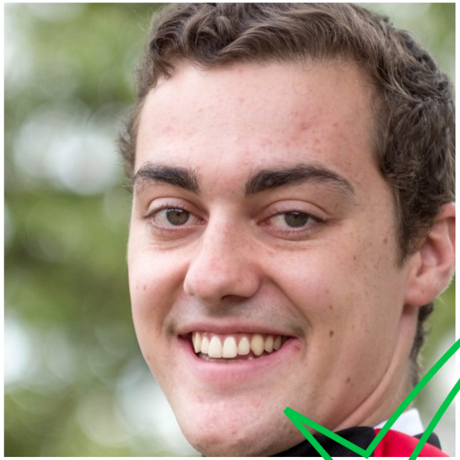
Let's hear from someone you might know!



Let's play a little game

Which of these images is real?

Let's play a little game



Which of these images is real?

Let’s play a little game

Which of these images is real?

Let's play a little game

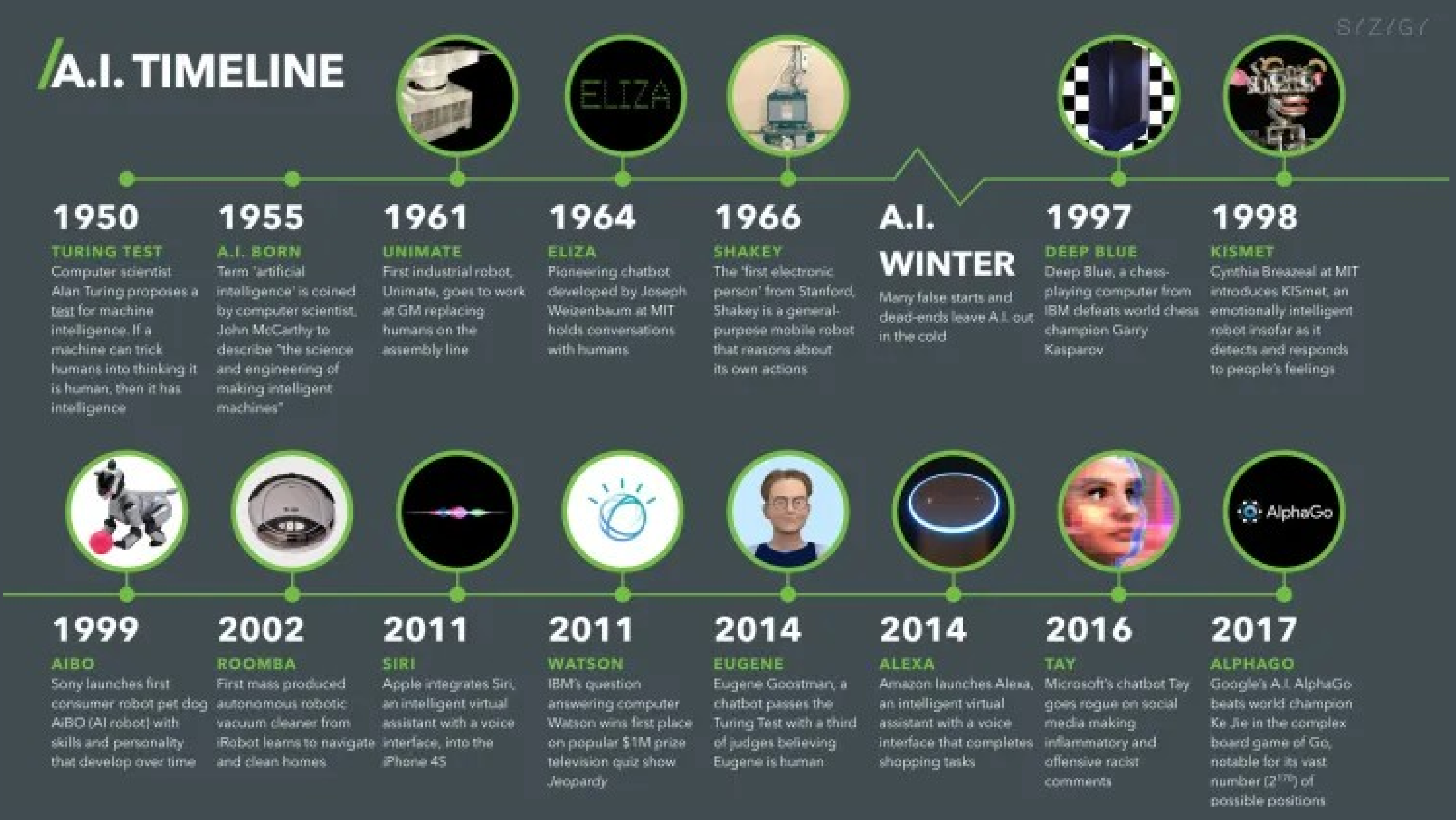
Which of these images is real?

Let’s play a little game

AI generated characters from Ramayana

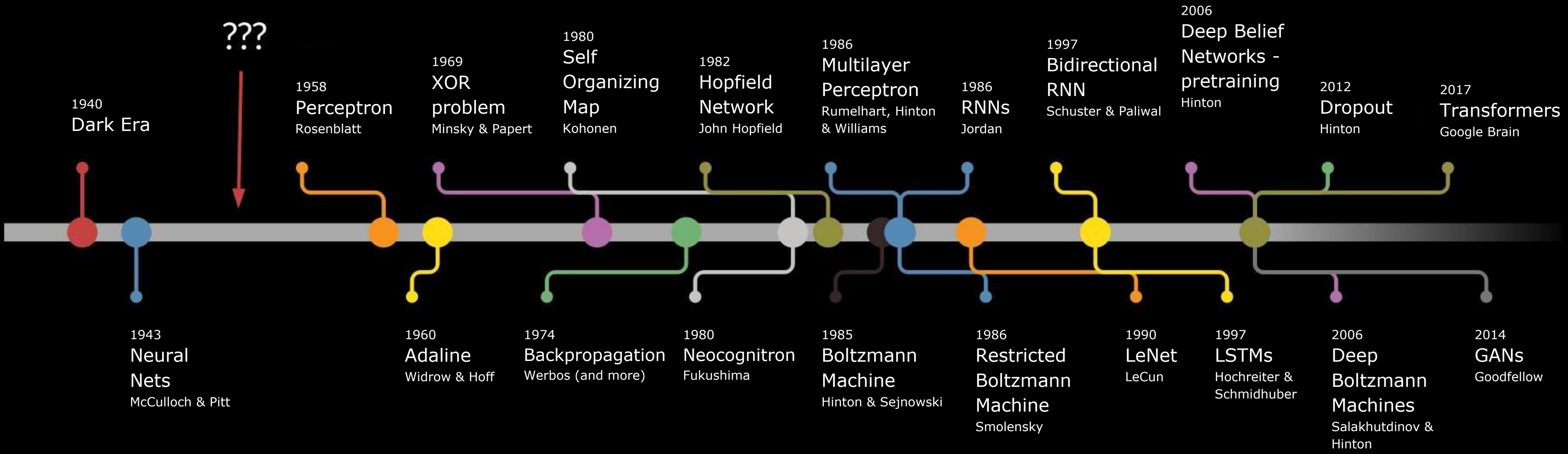
AI generated snowy winters in Delhi


Three people on couch



Credits: digitalwellbeing.com

Advances in AI driven by Advances in Deep Learning | Timeline



ML can be broadly classified into two types

PCA Analysis

k-means clustering

Gaussian mixing models

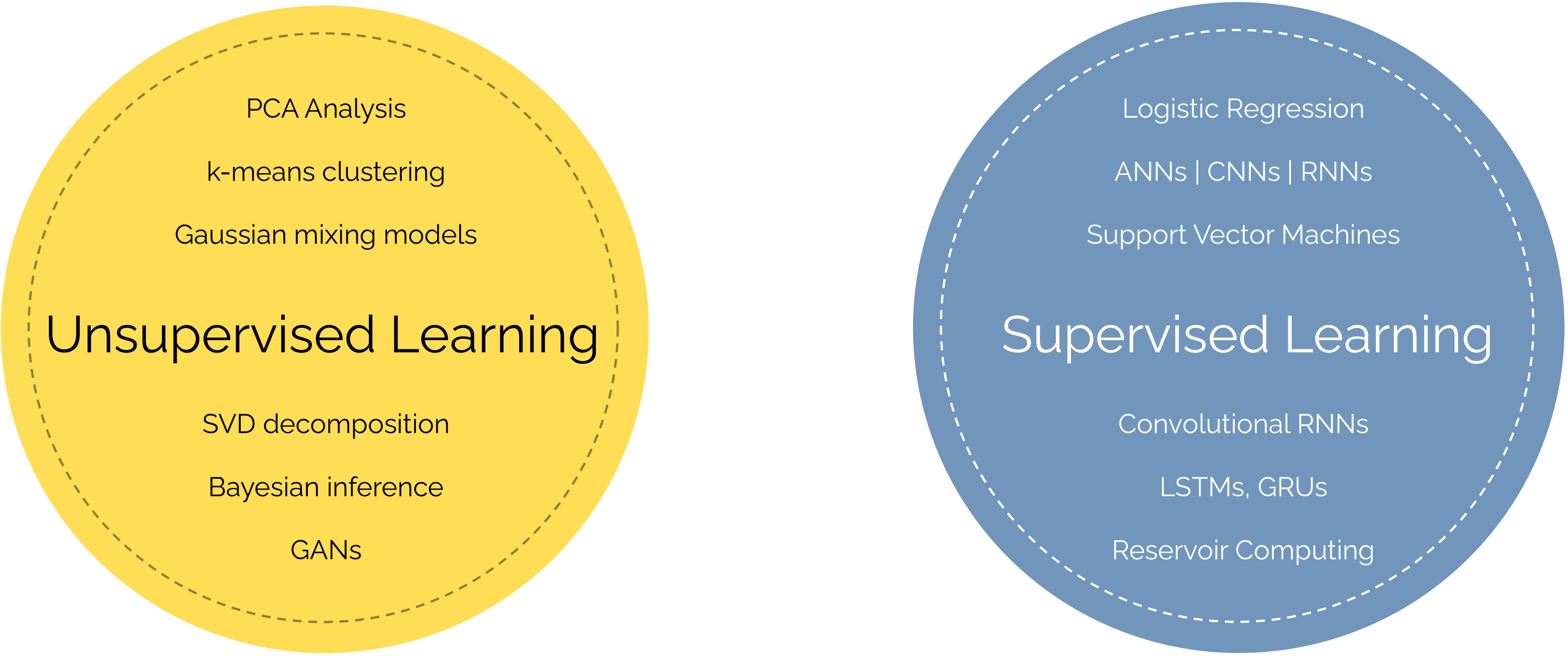
Unsupervised Learning

SVD decomposition

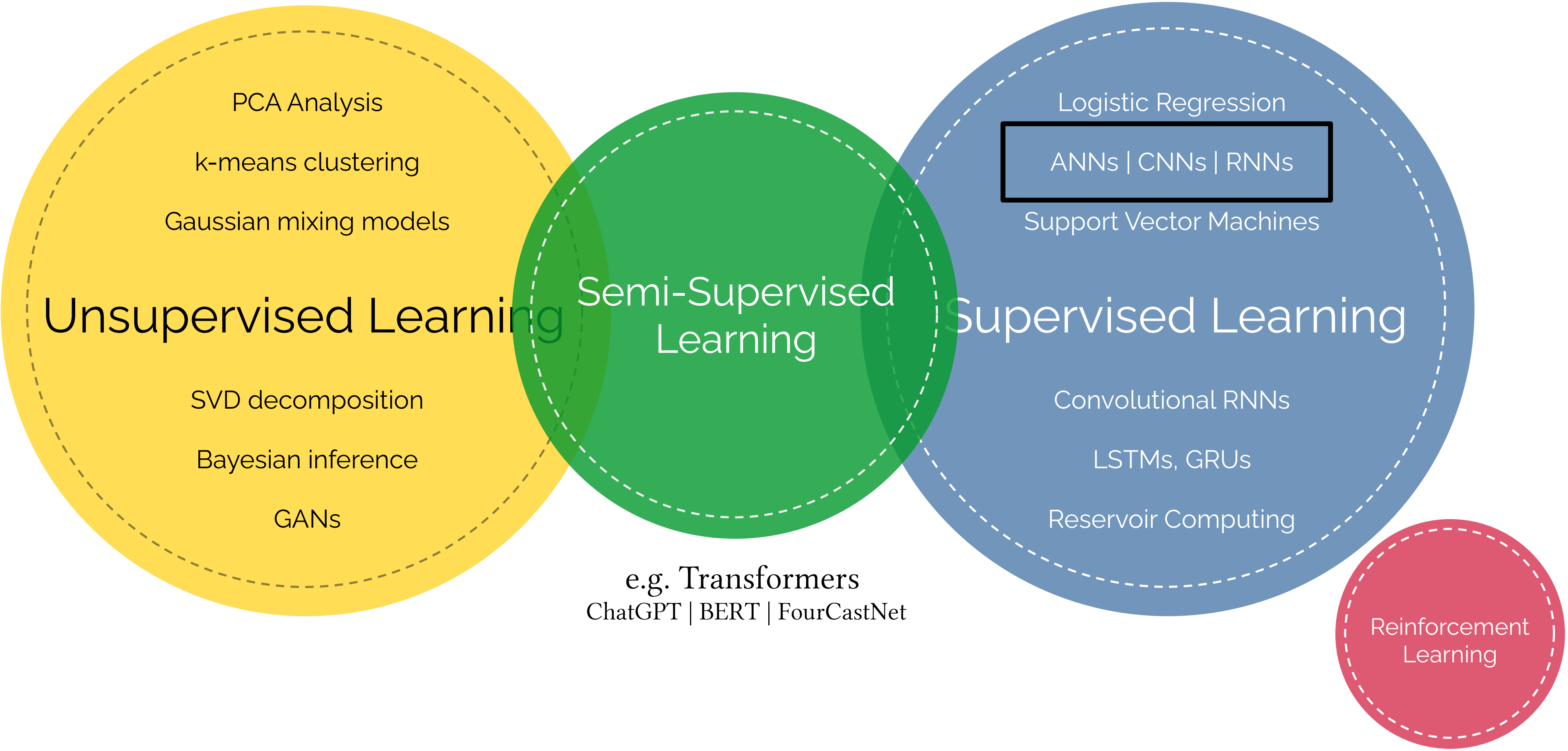
Bayesian inference

GANs

ML can be broadly classified into two types



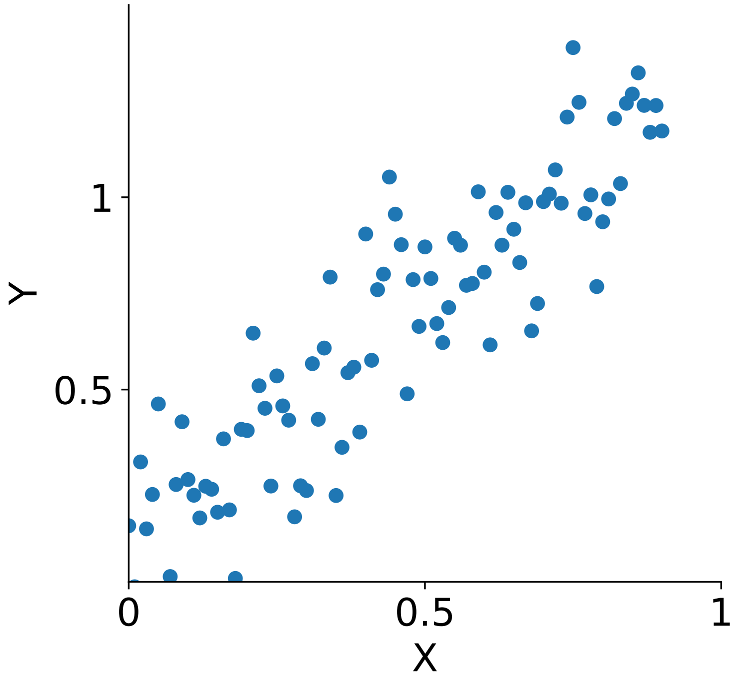
ML can be broadly classified into two types



e.g. Transformers
ChatGPT | BERT | FourCastNet

Let’s build up a theoretical model for Deep Learning

Consider the Linear Regression Problem



Assume a linear fit captures the relationship/function

$$Y = m \cdot X + c$$

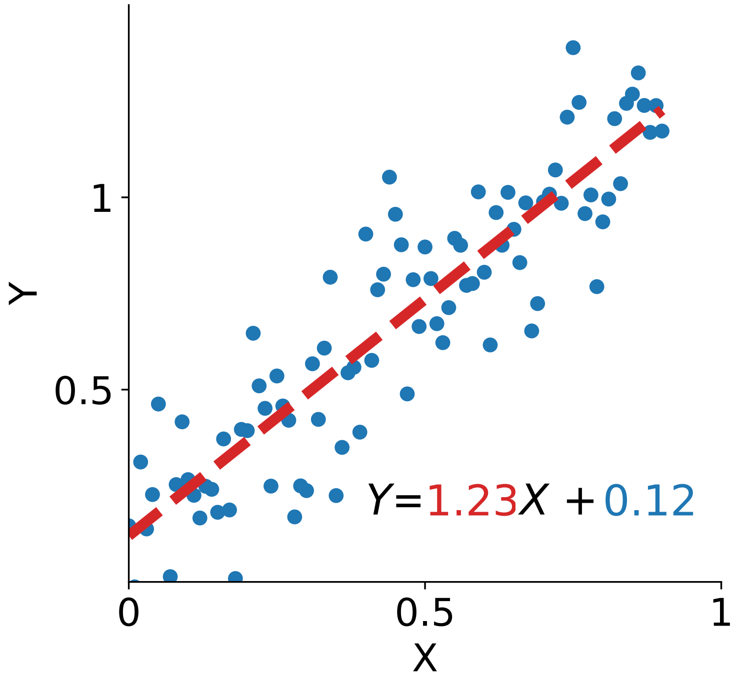
Choose the type of error to minimize

$$argmin_{m,c}\,(Y - (mX + c))^2$$

Estimate the parameters

$$m = \frac{Cov(X,Y)}{Var(X)} \quad c = \bar{Y} - m\bar{X}$$

Consider the Linear Regression Problem



Assume a linear fit captures the relationship/function

$$Y = m \cdot X + c$$

Choose the type of error to minimize

$$argmin_{m,c} \left(Y - (mX + c)\right)^2$$

Estimate the parameters

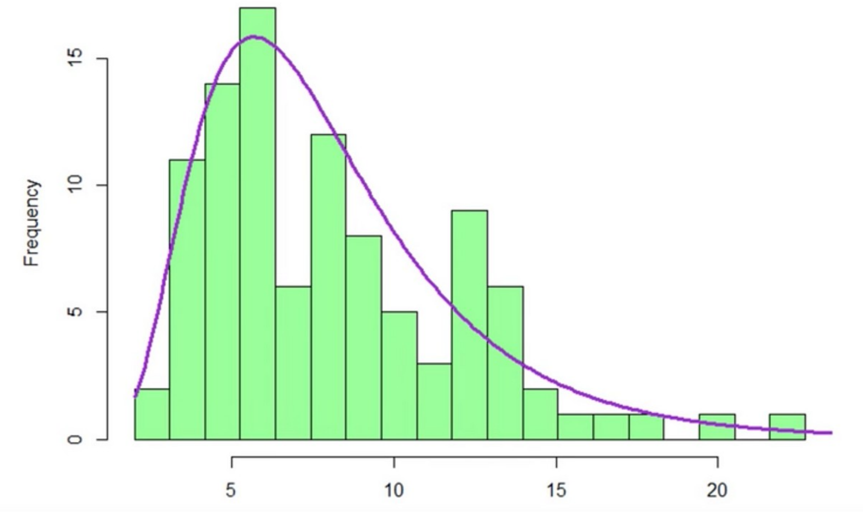
$$m = \frac{Cov(X,Y)}{Var(X)} \quad c = \bar{Y} - m\bar{X}$$

Similarly, for higher-order polynomials. Number of parameters scales with the degree of the approximating polynomial.

Require more advanced matrix algorithms to obtain the parameters (Normal equations, Vandermonde matrix, etc.)

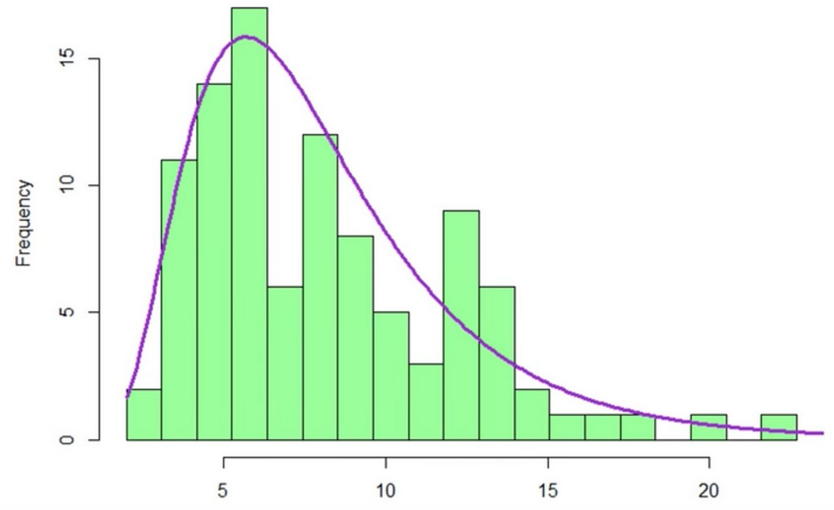
Similarly, Maximum Likelihood Estimation

Given data, find a parametric probability distribution that models the data with minimum error.



$$\theta_{MLE} = \arg\max_{\theta} \log P(X|\theta)$$

Similarly, Maximum Likelihood Estimation



$$\theta_{MLE} = \arg\max_{\theta} \log P(X|\theta)$$

Given data, find a parametric probability distribution that models the data with minimum error.

First, choose a log-normal distribution to model the data, reducing it to a parametric estimation problem

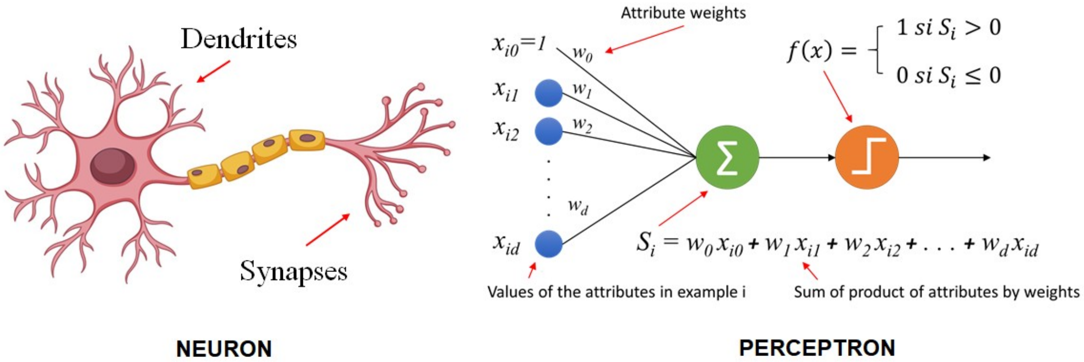
Then, choose the optimal parameters that minimize error (using grid-search, likelihood equations gradient descent, Newton's method etc.)

Residual determined by:
(1) Distribution used to model the data
(2) Algorithm used to solve for the parameters

Effectively, either the parameters can be obtained **analytically**, or they can be solved for **iteratively**.

Perceptrons: “Atoms” of Deep Neural Networks

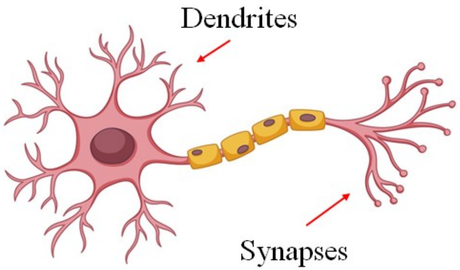
Perceptrons (Rosenblatt 1957) are a mathematical model of neurons. They are binary classifiers. They are the fundamental units of more complex deep learning models. Key components:



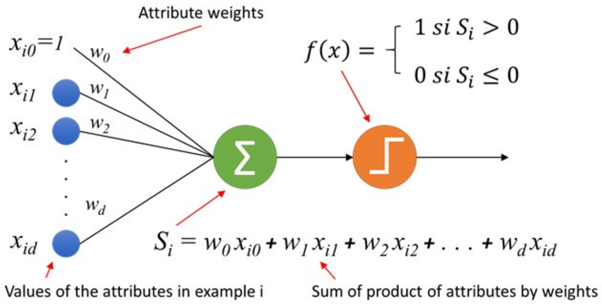
(Animation: https://towardsdatascience.com/what-the-hell-is-perceptron,
Image: https://inteligenciafutura.mx/english-version-blog/blog-06-english-version)

Perceptrons: “Atoms” of Deep Neural Networks

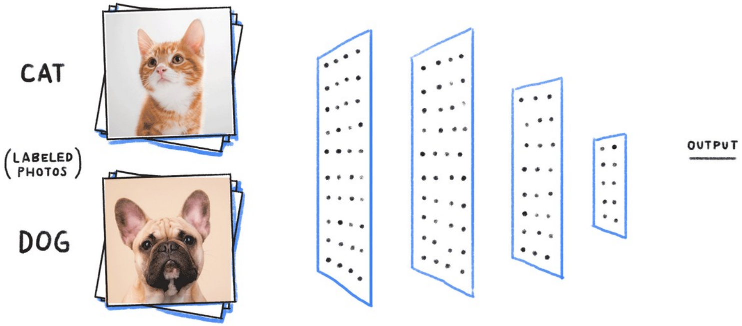
Perceptrons (Rosenblatt 1957) are a mathematical model of neurons. They are binary classifiers. They are the fundamental units of more complex deep learning models. Key components:



NEURON



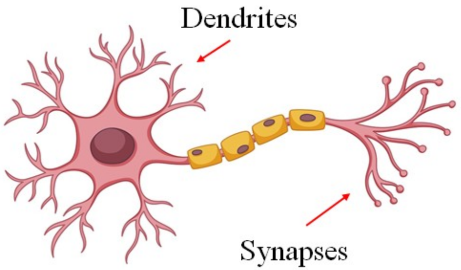
PERCEPTRON



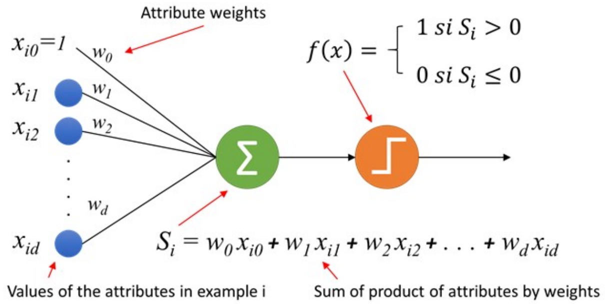
(Animation: https://towardsdatascience.com/what-the-hell-is-perceptron,
Image: https://inteligenciafutura.mx/english-version-blog/blog-06-english-version)

Perceptrons: “Atoms” of Deep Neural Networks

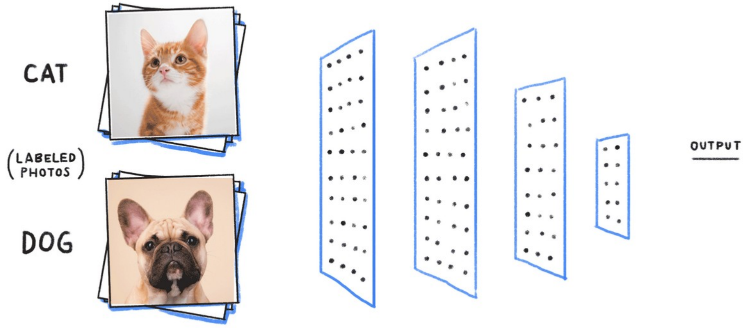
Perceptrons (Rosenblatt 1957) are a mathematical model of neurons. They are binary classifiers. They are the fundamental units of more complex deep learning models. Key components:



NEURON



PERCEPTRON



Neural activity → Parametric Estimation!

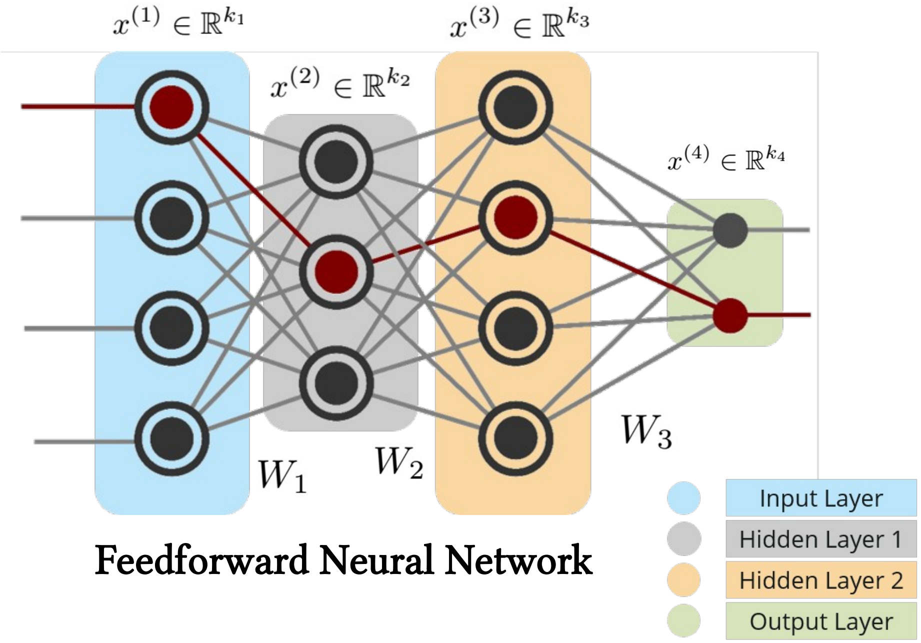
Key components: Input, weights, heaviside function, and output.

(Animation: https://towardsdatascience.com/what-the-hell-is-perceptron,
Image: https://inteligenciafutura.mx/english-version-blog/blog-06-english-version)

Neural Network as a Collection of Perceptrons

Brain is a network of interconnected neurons. For any input/actions, only selected neurons fire at a given time. A **multi-layer perceptron (MLP)** is a collection of neurons with equisized, fully-connected hidden layers. Similarly, a size-varying MLP without loops is called a **feedforward neural network**.

Consider a feedforward neural network arranged as an input layer, 2 hidden layers, and an output layer:



Feedforward Neural Network

Forward Propagation

(1) Each layer maps to the next using a set of weights

(2) The linear transformation is followed by a non-linear activation σ(.)

$$x^{(i+1)} = \sigma\left(W_i^T x^{(i)}\right)$$

$$W_i \in \mathbb{R}^{k_i \times k_{i+1}}, \sigma_i : \mathbb{R}^{k_{i+1}} \to \mathbb{R}^{k_{i+1}}$$

Universal Approximation Theorem

Number of neurons and number of hidden layers influence the learning capacity of a neural network.

- **Result 1**: Single-layer perceptrons are only capable of learning linearly separable patterns (1969)

- **Result 2:** Multi-layer perceptrons (MLPs) are capable of producing any possible boolean function.

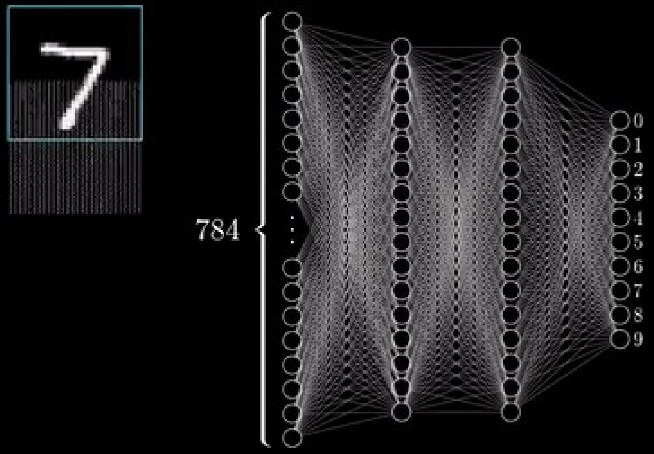
Removing the constraint of fully-connectedness and single activation function yields a
Feedforward Neural Network

- **Universal Approximation Theorem**: any continuous function $f : [0, 1]^n \longrightarrow [0, 1]$ can be approximated arbitrarily well by a neural network with at least 1 hidden layer with a finite number of weights.

It does not provide a construction for the weights, but surmises their existence.

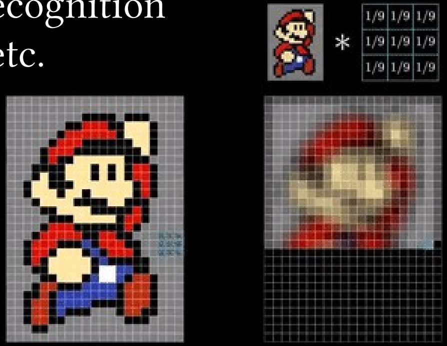
Examples of Deep Neural Networks

Vanilla Artificial NNs

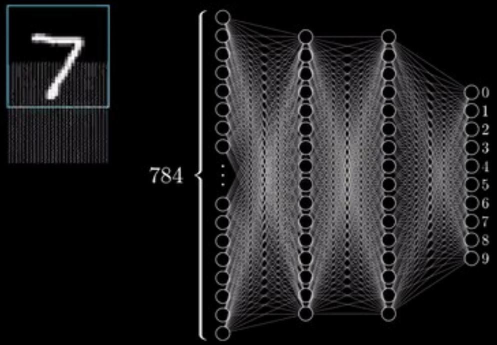


Examples of Deep Neural Networks

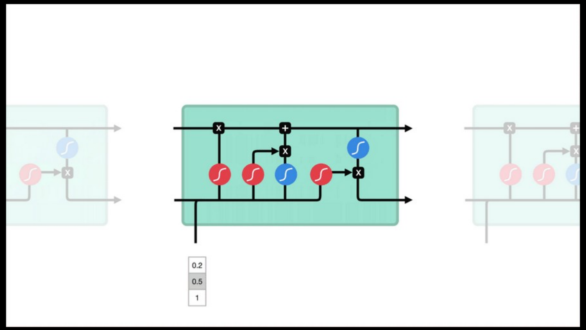
Convolutional NNs: for image/pattern recognition
e.g. image classifiers, facial recognition etc.



Vanilla Artificial NNs



Recurrent NNs: for sequence modeling
e.g. Long Short-Term Memory networks, Gated Recurrent Units for
language modeling,
music generation,
timeseries forecasting etc.



Credits: - youtube.com/@3blue1brown
- towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

How to Train Your Model

- In supervised learning, NNs learn the parameters by training on the data, i.e., a set of inputs and outputs, to obtain the optimal parameters that define the mapping
- Algorithm to train the model and update the weights:

Step 1: Start with (careful) random initialization of weights (**parameters**)

Step 2: **Forward Pass**: Propagate the input (**features**) through model layers to get an approximate output

Step 3: Compare the output with the truth (**label**). Compute the error using a **loss function (objective)** of choice

Step 4: **Backpropagation**: propagate the computed error backward through all the layers

Step 5: Update the weights using **optimizer** of choice

Continue for a number of steps (**epochs**) or until the output error reduces beyond a threshold

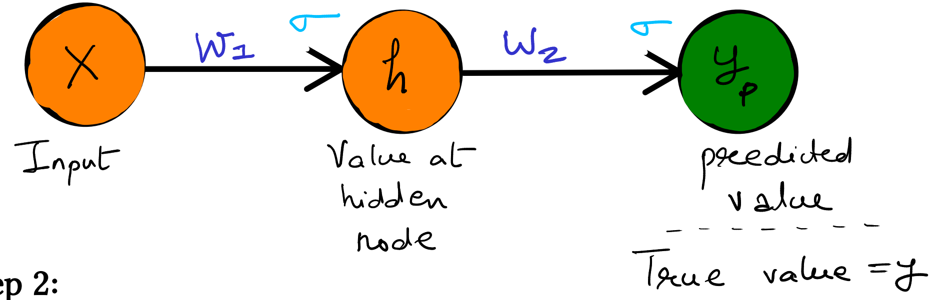
Once trained, use the model for evaluation/testing (**Inference**)

How to Train Your Model: Forward Pass

Inputs : $\vec{I} = \{x_1, x_2, \ldots, x_N\} \equiv X$

Outputs : $\vec{O} = \{y_1, y_2, \ldots, y_N\} \equiv y$

Step 1:



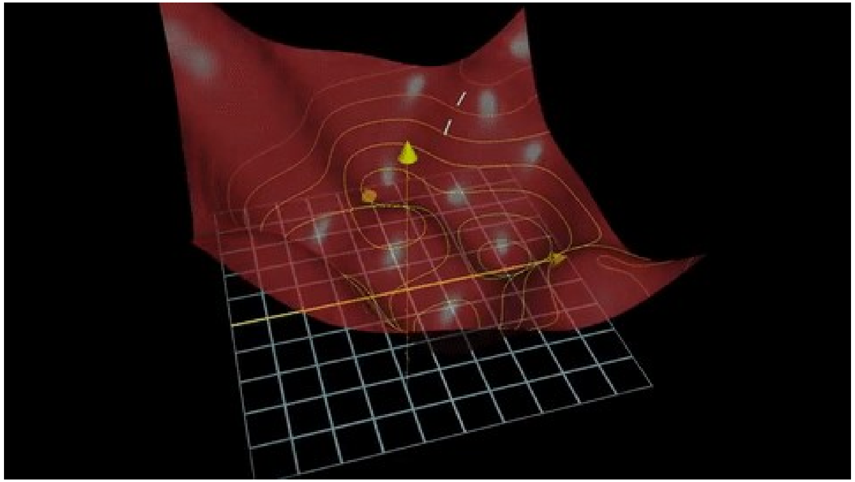
Step 2:

$$\boxed{x \to w_1 x \to \sigma(w_1 x) = h \to w_2\sigma(w_1 x) \to \sigma(w_2\sigma(w_1 x)) = y_p}$$

Step 3:

$$\boxed{\text{Loss/Error: } L(y, y_p; x, w_1, w_2) = \tilde{L}(y_p)}$$

How to Train Your Model: Forward Pass



Iteratively update parameters as

$$w_1 \leftarrow w_1 - \alpha \frac{\partial \tilde{L}}{\partial w_1}$$

$$w_2 \leftarrow w_2 - \alpha \frac{\partial \tilde{L}}{\partial w_2}$$

compute gradient for millions and billions of parameters. Good luck!

Step 1:

Step 2:

$$\boxed{x \to w_1 x \to \sigma(w_1 x) = h \to w_2 \sigma(w_1 x) \to \sigma(w_2 \sigma(w_1 x)) = y_p}$$

Step 3:

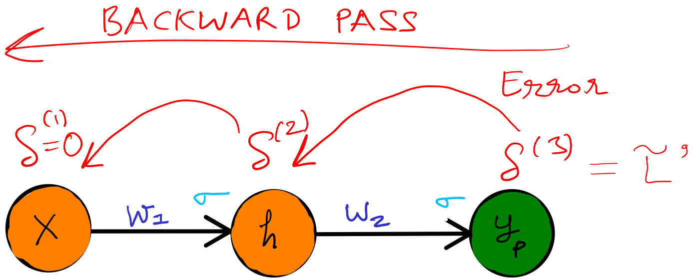
$$\boxed{\text{Loss/Error: } L(y, y_p; x, w_1, w_2) = \tilde{L}(y_p)}$$

$$\frac{\partial \tilde{L}}{\partial w_2} = \tilde{L}' \cdot \frac{d\sigma}{dw_2} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2 \sigma(w_1 x))}{dw_2} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot h$$

$$\frac{\partial \tilde{L}}{\partial w_1} = \tilde{L}' \cdot \frac{d\sigma}{dw_1} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2 \sigma(w_1 x))}{dw_1} = \tilde{L}' \cdot \sigma' \cdot w_2 \cdot \frac{d\sigma(w_1 x)}{dw_1} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot w_2 \cdot \sigma'(w_1 x) \cdot x$$

How to Train Your Model: Backward Pass

Backpropagation: a dynamic programming algorithm to compute the gradients efficiently. Start from the error in the output layer and propagate it backwards through iterative multiplication



$$\frac{\partial \tilde{L}}{\partial w_2} = \tilde{L}' \cdot \frac{d\sigma}{dw_2} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2\sigma(w_1 x))}{dw_2} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot h$$

$$\frac{\partial \tilde{L}}{\partial w_1} = \tilde{L}' \cdot \frac{d\sigma}{dw_1} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2\sigma(w_1 x))}{dw_1} = \tilde{L}' \cdot \sigma' \cdot w_2 \cdot \frac{d\sigma(w_1 x)}{dw_1} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot w_2 \cdot \sigma'(w_1 x) \cdot x$$

$$\frac{\partial \tilde{L}}{\partial \omega_2} = \tilde{L}'\sigma' h$$

$$\frac{\partial \tilde{L}}{\partial \omega_1} = \tilde{L}'\sigma'\omega_2\sigma' x$$

$$\begin{pmatrix}\omega_1\\ \omega_2\end{pmatrix} \leftarrow \begin{pmatrix}\omega_1\\ \omega_2\end{pmatrix} - \alpha\begin{pmatrix}\partial\tilde{L}/\partial\omega_1\\ \partial\tilde{L}/\partial\omega_2\end{pmatrix}$$

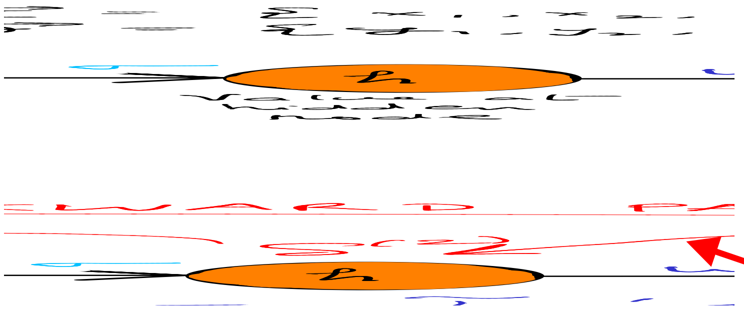
How to Train Your Model: Backward Pass

Backpropagation: a dynamic programming algorithm to compute the gradients efficiently. Start from the error in the output layer and propagate it backwards through iterative multiplication

$$\frac{\partial \tilde{L}}{\partial w_2} = \tilde{L}' \cdot \frac{d\sigma}{dw_2} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2\sigma(w_1 x))}{dw_2} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot h$$

$$\frac{\partial \tilde{L}}{\partial w_1} = \tilde{L}' \cdot \frac{d\sigma}{dw_1} = \tilde{L}' \cdot \sigma' \cdot \frac{d(w_2\sigma(w_1 x))}{dw_1} = \tilde{L}' \cdot \sigma' \cdot w_2 \cdot \frac{d\sigma(w_1 x)}{dw_1} = \tilde{L}'(y_p) \cdot \sigma'(w_2 h) \cdot w_2 \cdot \sigma'(w_1 x) \cdot x$$

σ’



More generally, in higher dimensions,
Backpropagate the error from the output layer to the input layer:

Step 4: $\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} \cdot \sigma'(z^l)$

Compute the derivative of the loss w.r.t. the layer parameters:

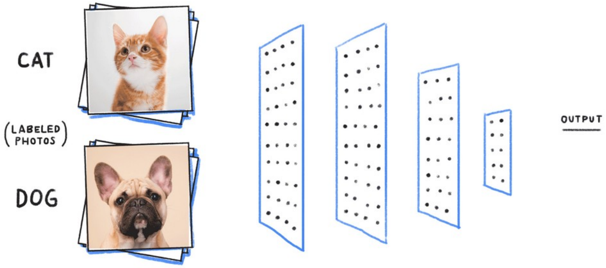
$$\frac{\partial \tilde{L}}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} \cdot \delta_i^{(l+1)} \quad \Bigg| \quad \Theta_{ij}^{(l)} \leftarrow \Theta_{ij}^{(l)} - \alpha \frac{\partial \tilde{L}}{\partial \Theta_{ij}^{(l)}}$$

: Step 5

We are now ready to create our own neural networks in Python!

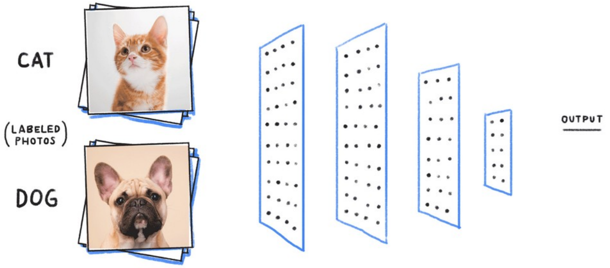
To Summarize

1) Machine Learning is increasingly used in all spheres of digital life.
   It can be broadly categorized into two categories: **Supervised Learning and Unsupervised Learning.**

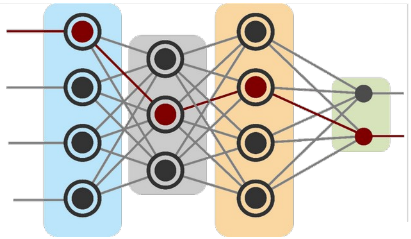


To Summarize

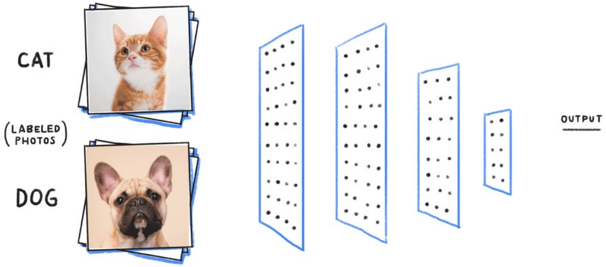
1) Machine Learning is increasingly used in all spheres of digital life. It can be broadly categorized into two categories: **Supervised Learning and Unsupervised Learning.**



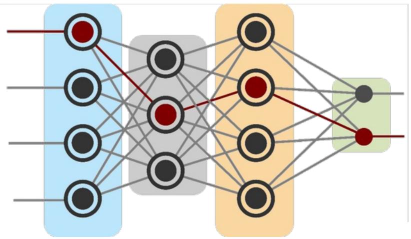
2) The widely popular tools from supervised learning, **neural networks**, claim to approximate any function through a set of linear transformations and non-linear activations.



To Summarize



1) Machine Learning is increasingly used in all spheres of digital life.
   It can be broadly categorized into two categories: **Supervised Learning and Unsupervised Learning.**



2) The widely popular tools from supervised learning, **neural networks**, claim to approximate any function through a set of linear transformations and non-linear activations.

3) The parameters of the neural network architecture can be estimated, i.e., the neural network can be trained using a iterative training method which relies on backpropagation and stochastic optimization for parameter update.

Setting up PyTorch

1) Install conda (or pip):

https://docs.conda.io/projects/conda/en/latest/user-guide/install/linux.html

2) Install Jupyter notebook:

```
conda install -c anaconda jupyter
```

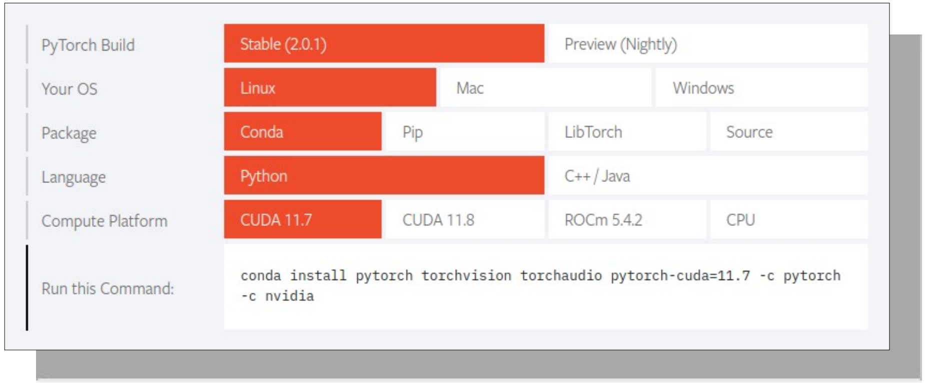
3) Install numpy and mathplotlib:

```
conda install -c anaconda numpy
conda install -c conda-forge matplotlib
```

4) Install PyTorch

Go to: https://pytorch.org/get-started/locally/

```
conda install pytorch torchvision torchaudio
pytorch-cuda=11.7 -c pytorch -c nvidia
```



Supplementary Slides

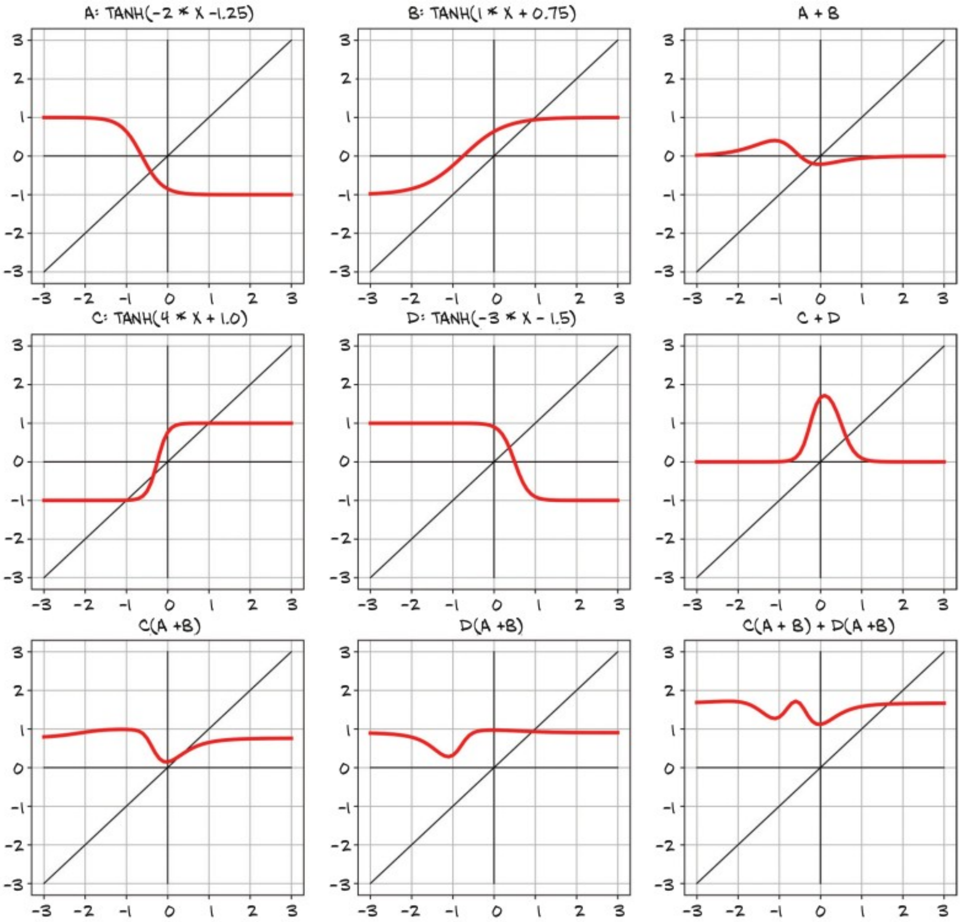
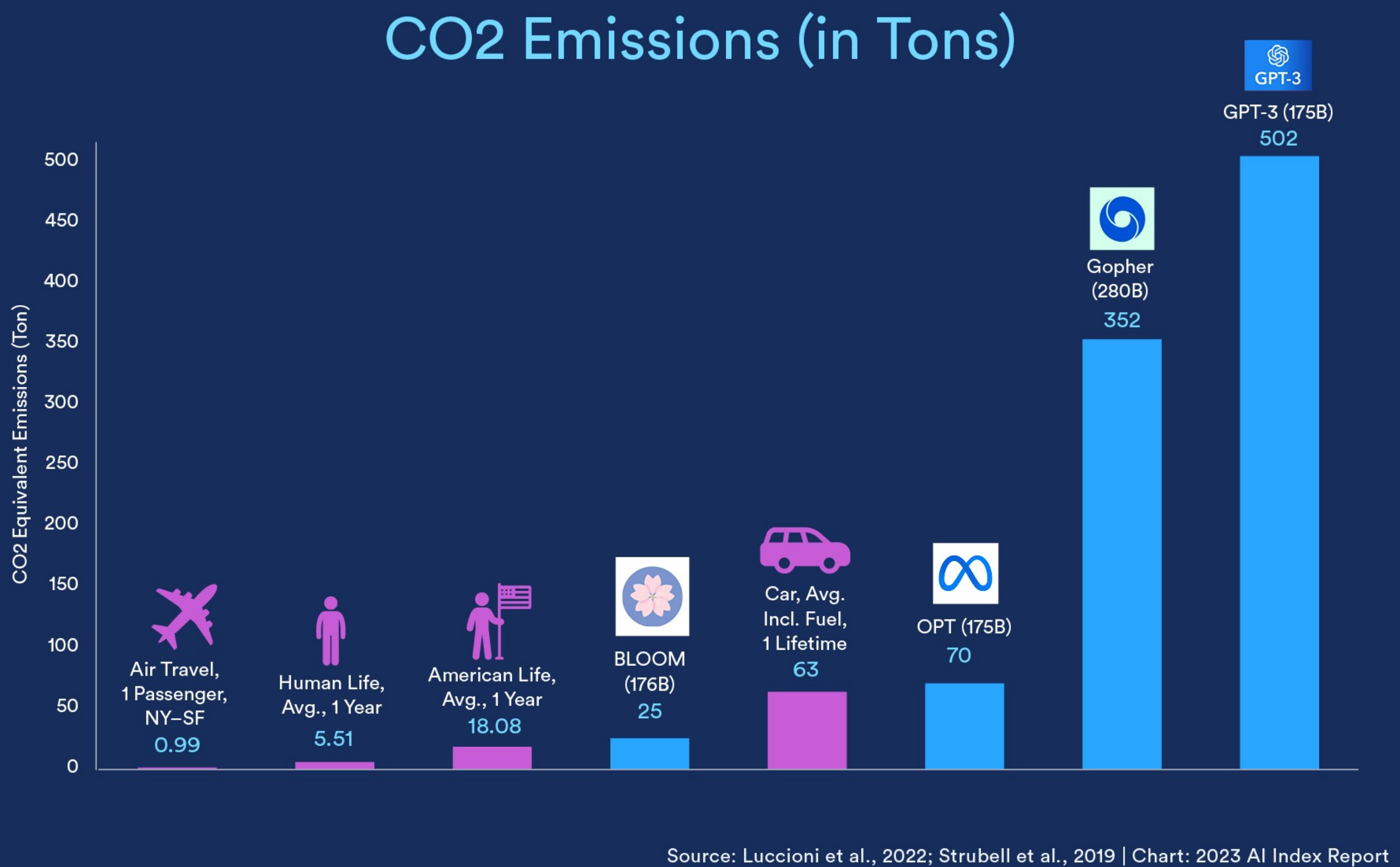


Figure 6.6 Composing multiple linear units and `tanh` activation functions to produce nonlinear outputs


CO2 Emissions (in Tons)
CO2 Equivalent Emissions (Ton)
500
450
400
350
300
250
200
150
100
50
0
Air Travel, 1 Passenger, NY–SF
0.99
Human Life, Avg., 1 Year
5.51
American Life, Avg., 1 Year
18.08
BLOOM (176B)
25
Car, Avg. Incl. Fuel, 1 Lifetime
63
OPT (175B)
70
Gopher (280B)
352
GPT-3
GPT-3 (175B)
502
Source: Luccioni et al., 2022; Strubell et al., 2019 | Chart: 2023 AI Index Report