From 0 to distributed training!

Niccolo Tosato

Chapter 1: Let the computer learn

Learning models

Models aren’t explicitly programmed - they learn from data!

- Traditional programming:

Rules + Data → Answers

- Machine learning:

Data + Answers → Rules

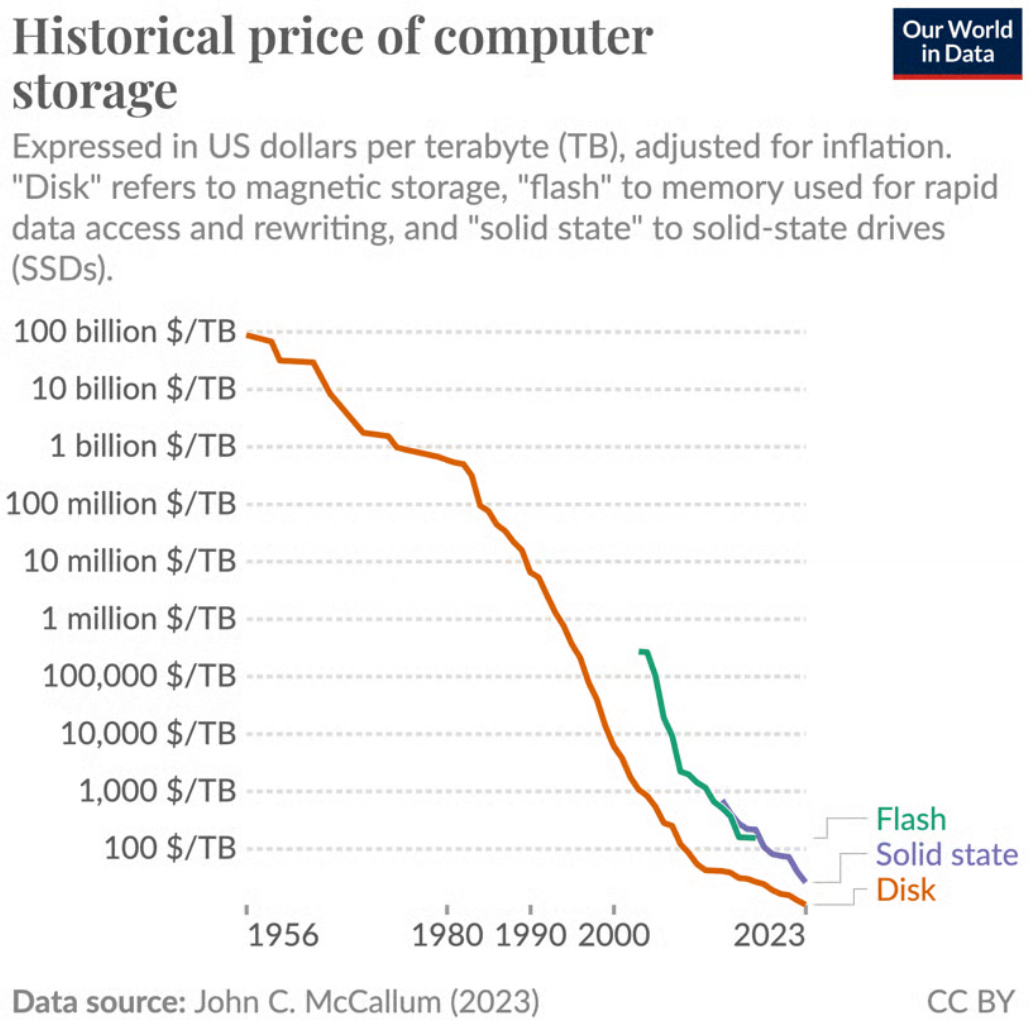
Data as the Starting Point

In machine learning, data isn’t just important - it’s everything.

- **Garbage in, garbage out**: poor data leads to poor models
- **Preprocessing is essential**: cleaning, normalization, balancing
- **Data is diverse**: text, images, time series, tabular…

Different data types require **different models and architectures**

Why?



Storage price

Why?

AlexNet milestone

In 2012, AlexNet marked a breakthrough in deep learning by training the first large-scale convolutional neural network:

- 60 million parameters
- 1.2 million images
- 1000 classes

Six days of training with 2 x **GTX 580 3GB** GPU !

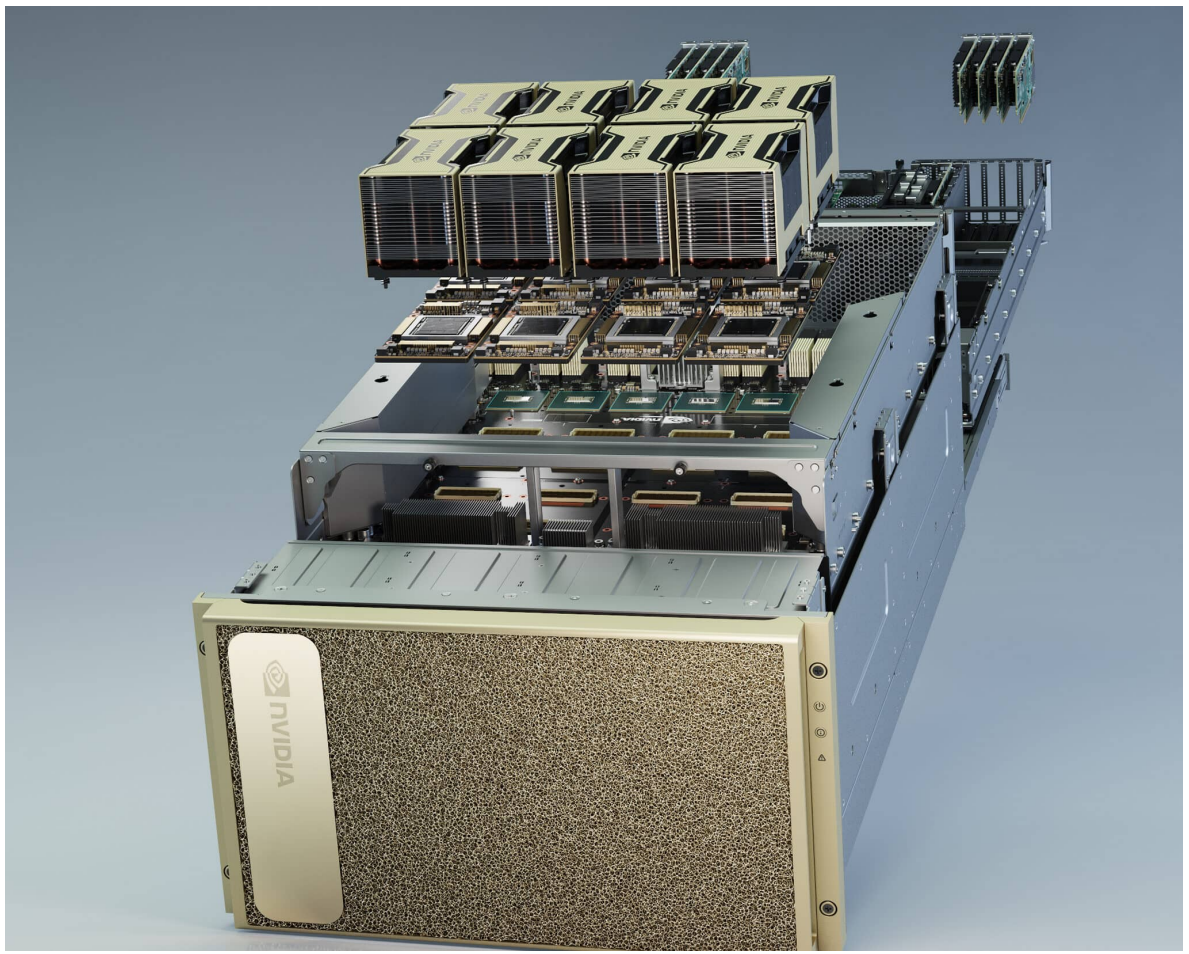
~ 13 years ago

All of our experiments suggest that our results can be improved simply by waiting for faster GPUs and bigger datasets to become available[1]

1. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems (Vol. 25)

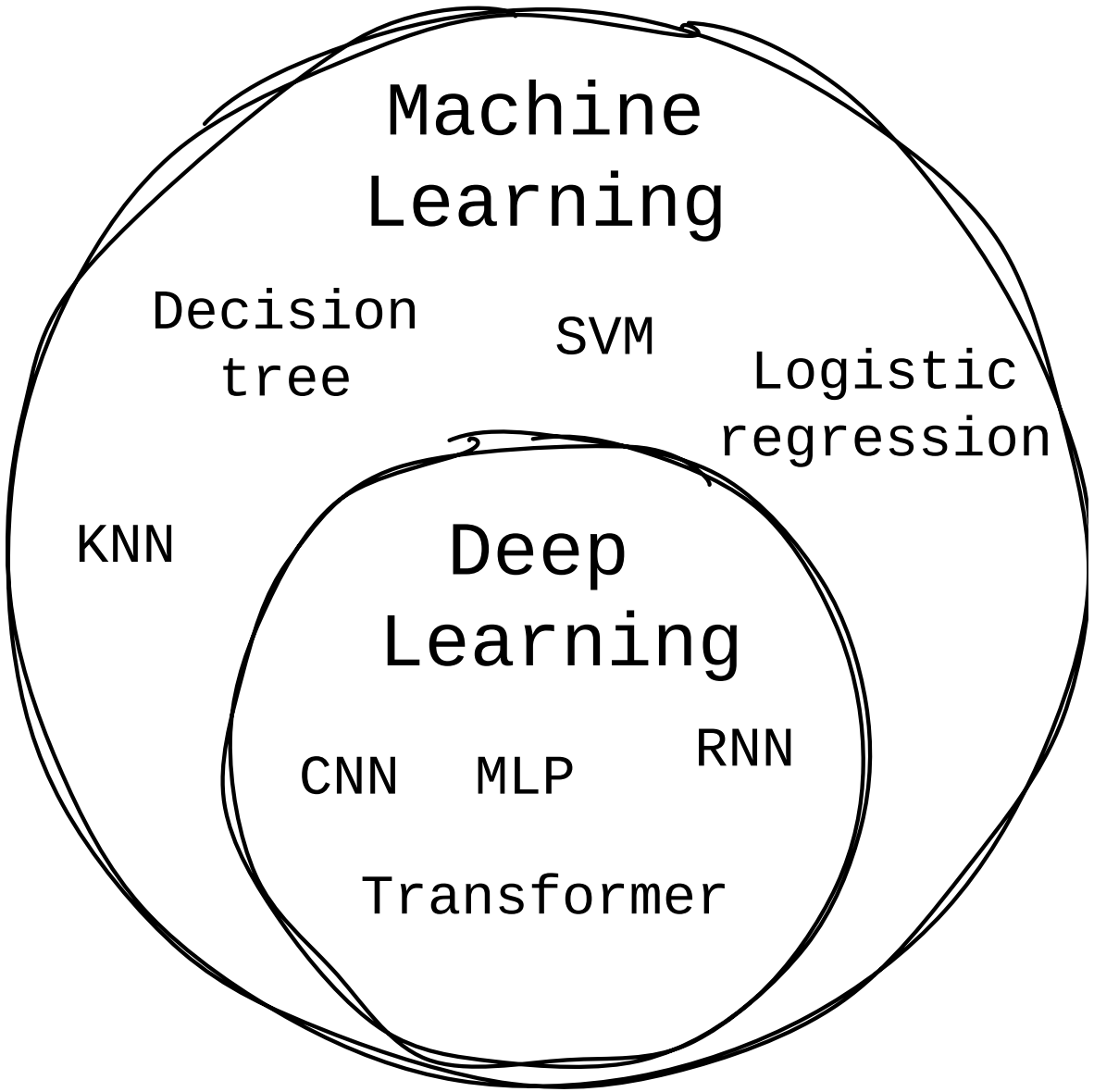
~ Today

Up to **8 GPUs** per node, **640 GB** of VRAM !



Nvidia DGX H100

Deep Learning or Machine Learning ?



Paradigms of Machine Learning

Machine learning can be categorized into three main paradigms:

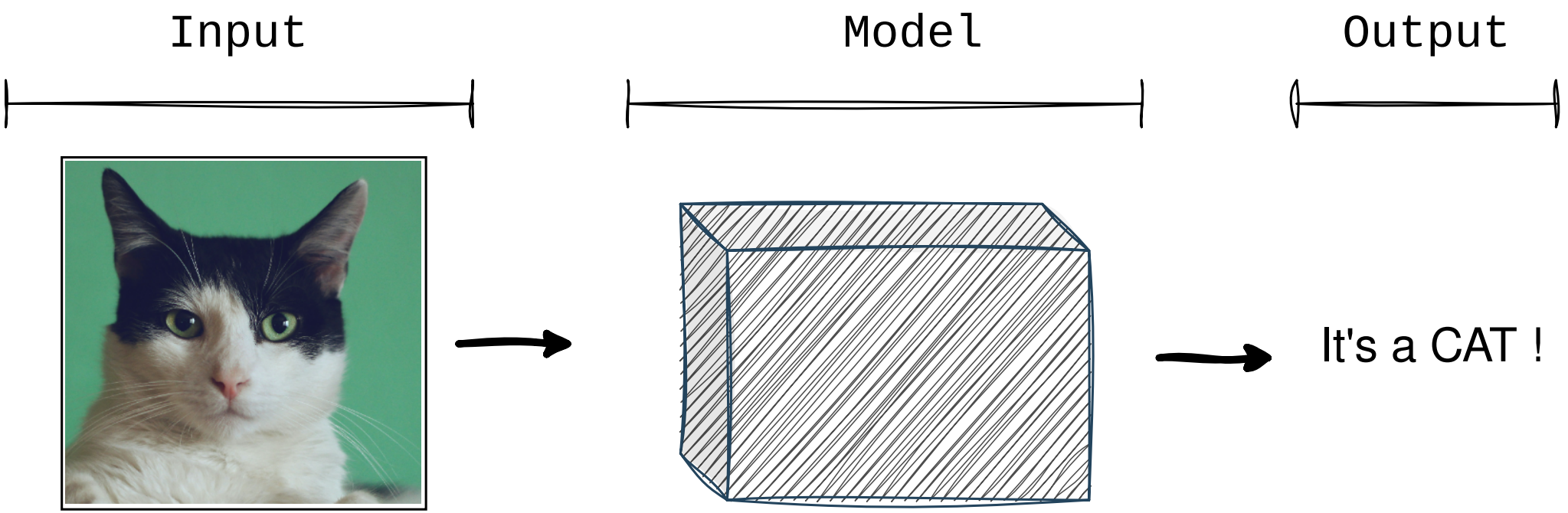
- **Supervised** Learning: Learn from labeled examples
- **Unsupervised** Learning: Find patterns without labels
- **Reinforcement** Learning: Learn from interaction

Supervised Learning

The model receives input-output pairs (X, Y) and learns to map X to Y.

- Classification: Y is a **discrete** label
- Regression: Y is a **continuous** value

Supervised Learning



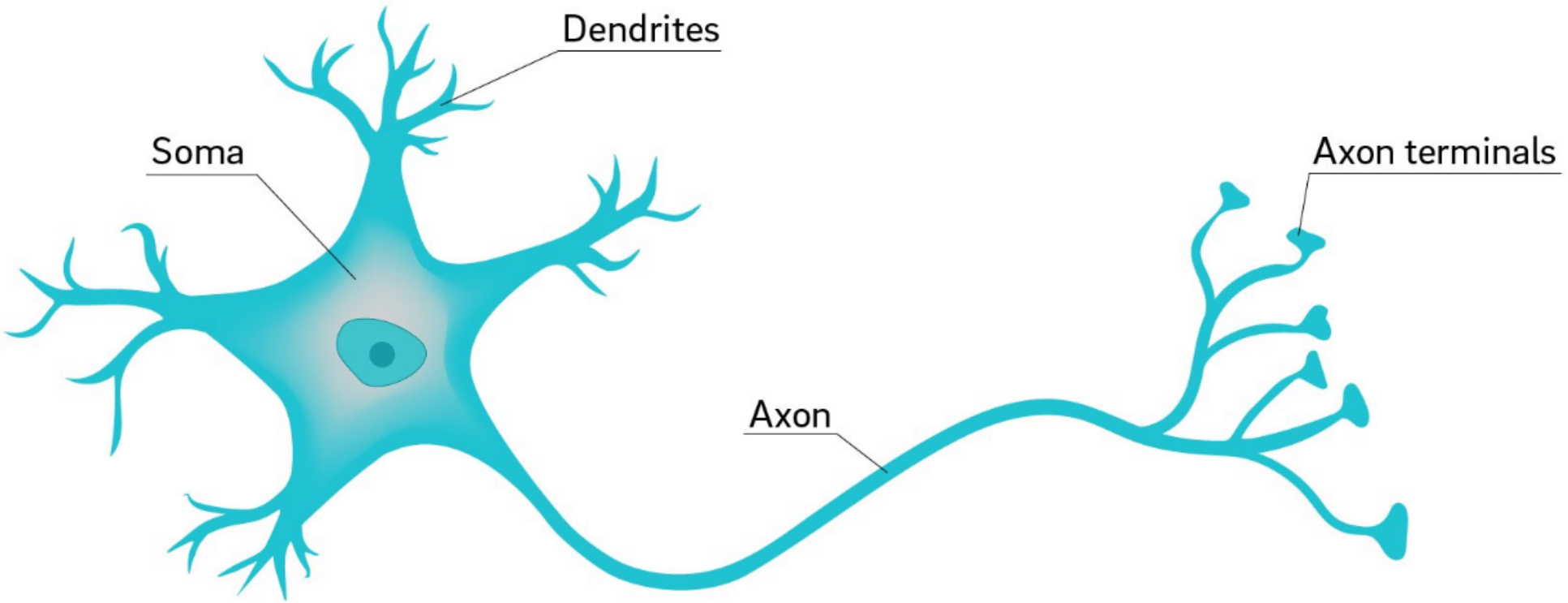
From 0 to distributed training

We will talk about:

- **Supervised** Learning
- With deep neural networks (**DNN**s)
- Performing a **classification** task

In parallel !

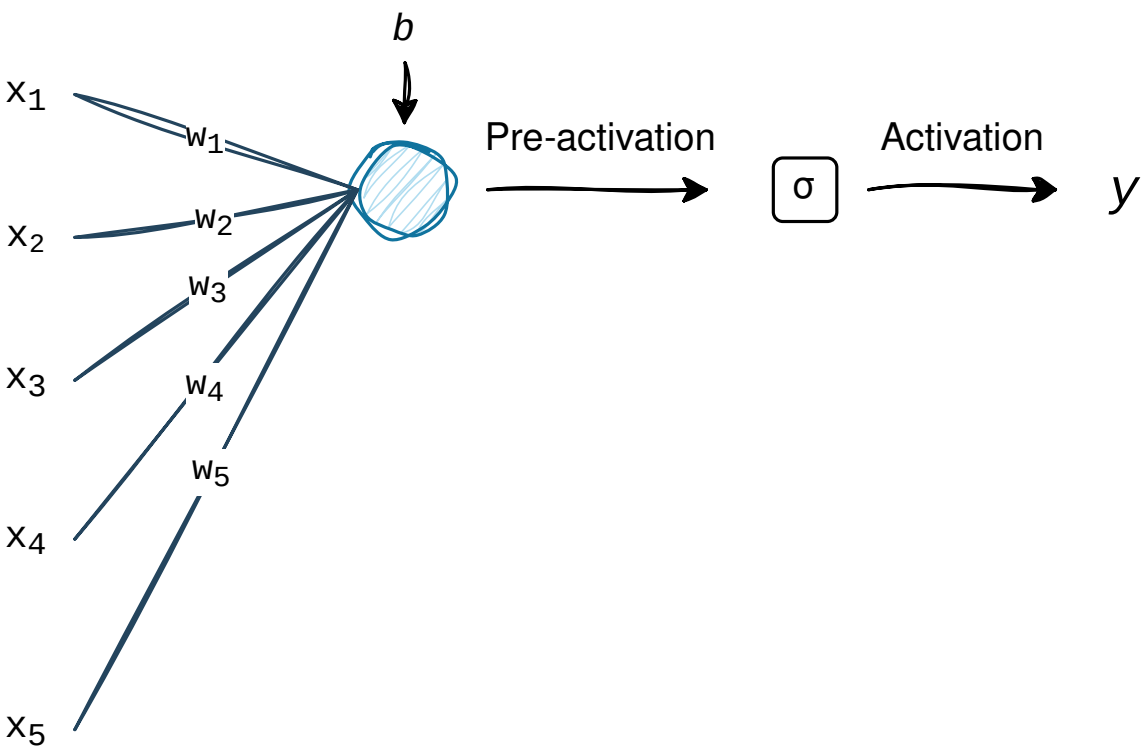
The neuron



Neuron

The artificial neuron

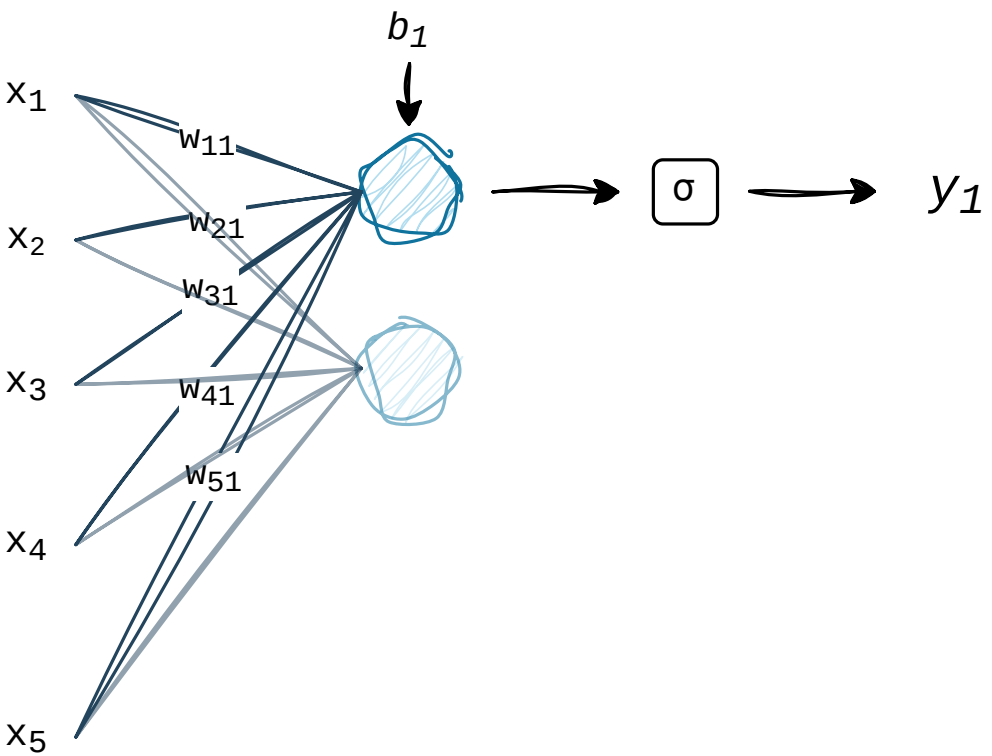
$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b)$$



n input dimensionality. b constant bias term. w_i weights for each input. σ is the activation function.

More neurons

$$y_i = \sigma(\sum_{i=1}^{n} w_{ij} x_i + b_i)$$



n is the input dimensionality. m is the number of neurons in a layer.

Just Matrix-Vector Multiplication

$$y_i = \sigma(\sum_{i=1}^{n} w_{ij} x_i + b_i)$$

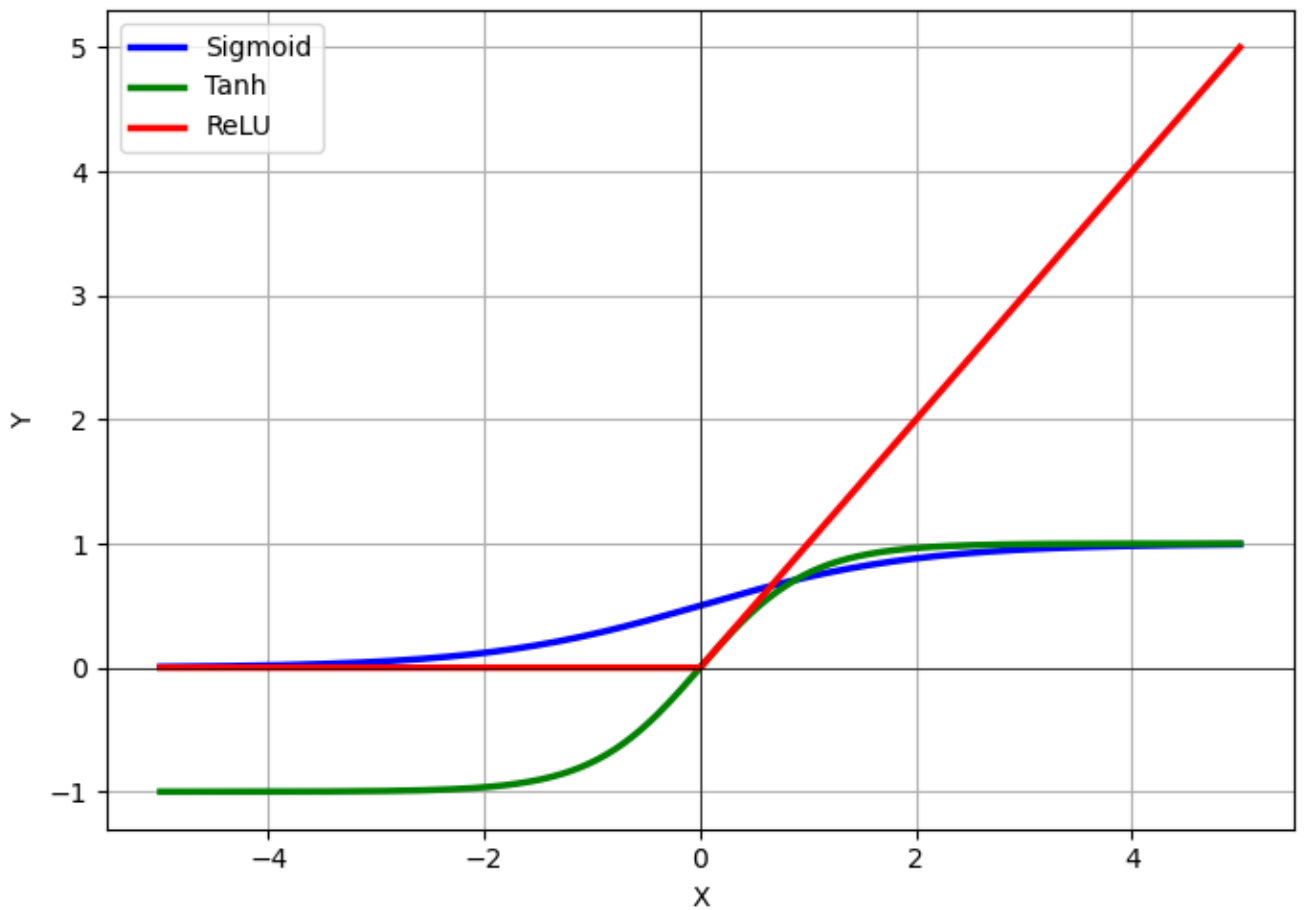
But what if we process many inputs at once?

Then, we can perform **matrix-matrix** multiplication, which increases arithmetic intensity - more computations per memory access.

This is why we use accelerators like GPUs

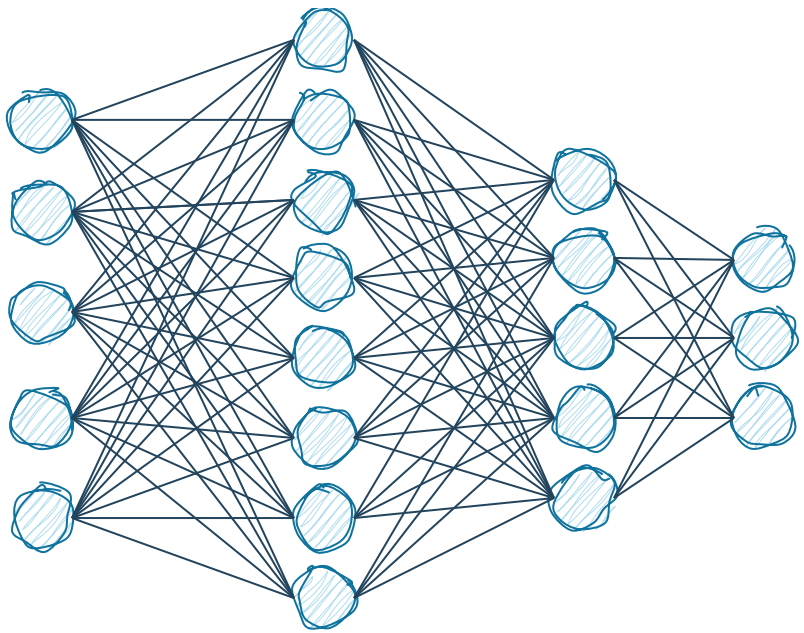
Activation functions

To allow neural networks to learn complex, non-linear patterns, we use activation functions between layers.



Neural networks

Multi layer perceptrons (MLP) are the simplest form of neural networks



It's a CAT !

Universal approximation theorem

A feedforward neural network with at least one hidden layer can approximate any continuous function to any desired accuracy, given enough neurons and the right activation function.

What this means in practise:

- Neural networks can approximate almost anything
- More neurons usually mean better performance
- In a realistic setting, depth is usually better than width

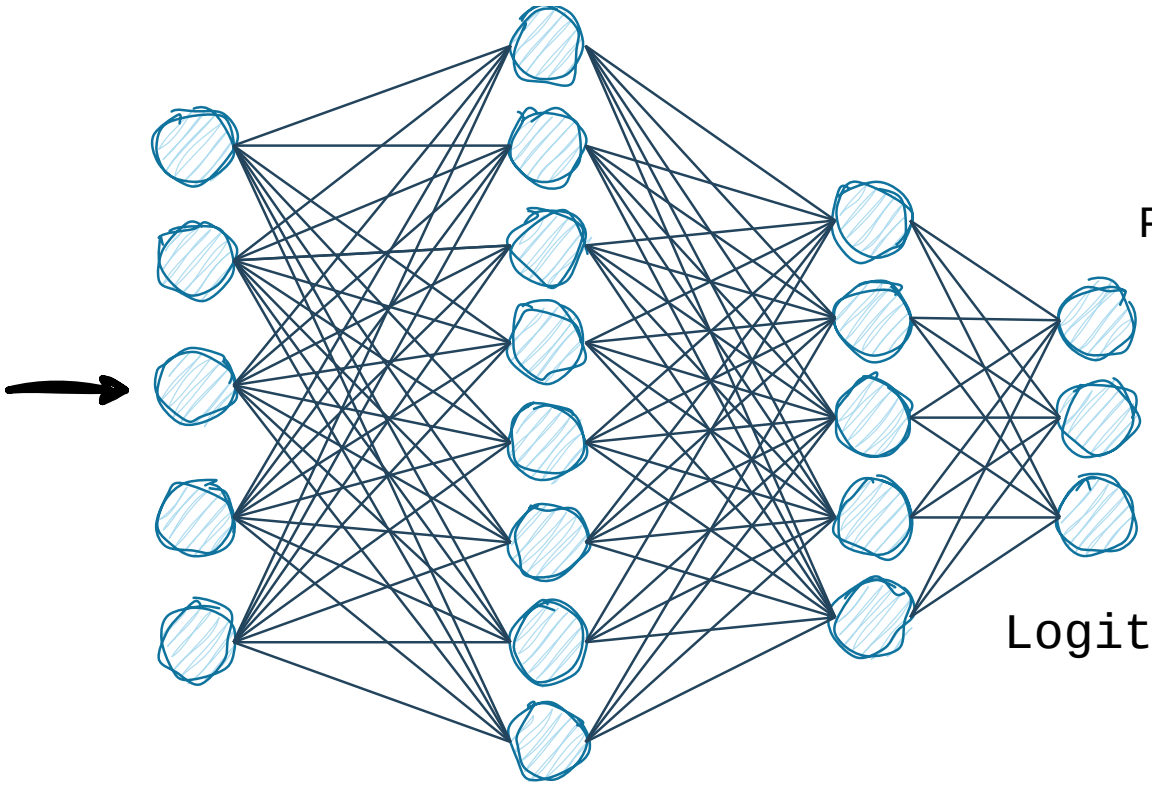
Chapter 2: Training details

Model parameters

Train a model mean:

Find the best paramters w and b to solve the task, just by looking at data samples.

Example of classification task



Logits

Prediction		Ground truth
0.1	Dog	0
0.7	Cat	1
0.2	Bird	0

High-level training ingredients

Core elements needed to train a neural network:

- A dataset of (x, y) pairs *(supervised learning setting)*
- A **loss** function to quantify prediction error
- A gradient-based **optimizer**
- A method to compute **gradients**

Dataset

Usually we use some data to train the model and other point to asses the model performance. We will call the partition: **trainset** and **testset**

Datasets are typically too large to fit entirely in memory.

To handle this, we **split the data into batches** - small subsets of the dataset processed one at a time.

Loss function

Usually a function that quantifies how well the model's predictions match the true labels. It is a scalar.

Regression tasks:

- Mean squared Error (L2 norm)
- Mean absolute error (L1 norm)

Classification tasks:

- Cross entropy loss

Computing the gradient

To train a neural network, we need to compute how the loss changes with respect to each weight.

Way to go: **Backpropagation** algorithm[1].

1. Introduced by Rumelhart, Hinton, and Williams in their landmark 1986 paper, “Learning representations by back-propagating errors”

Backpropagation

- Backpropagation is an application of the chain rule from calculus, used to compute how each parameter affects the loss.

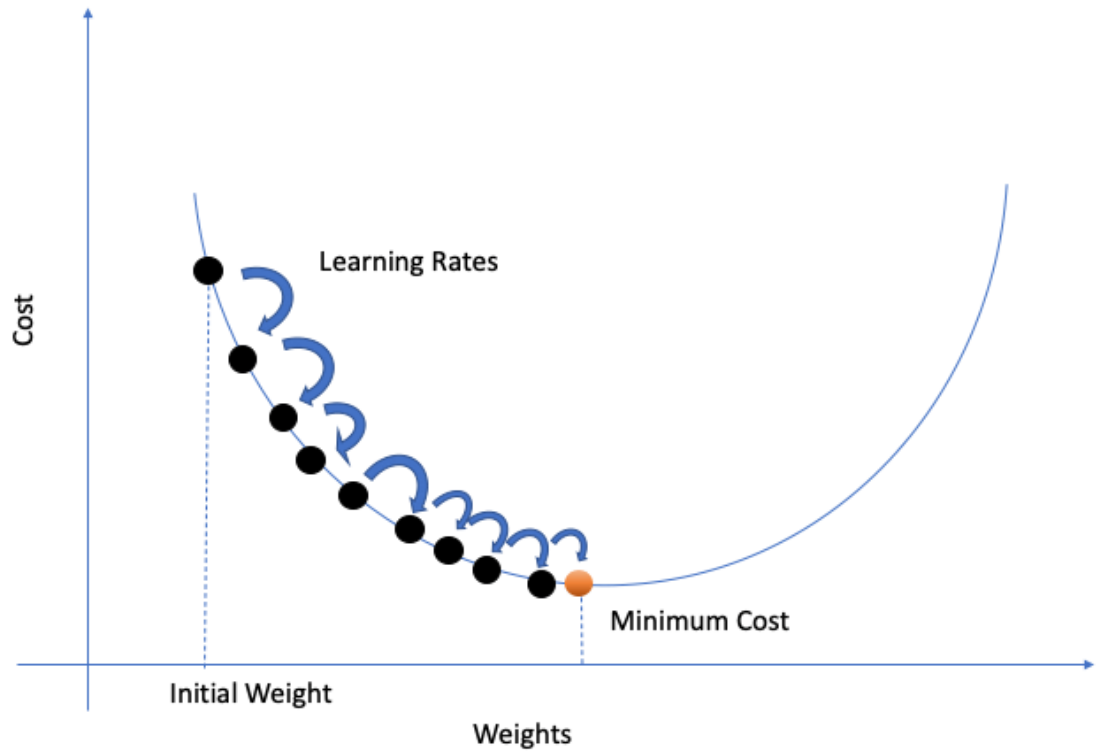
- It works from output to input, layer by layer, propagating the error backwards through the network.

It give us:

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial w_{ij}}$$

where L is the loss, y_j is the output of neuron j, and w_{ij} is the weight connecting input i to neuron j.

Gradient Descend



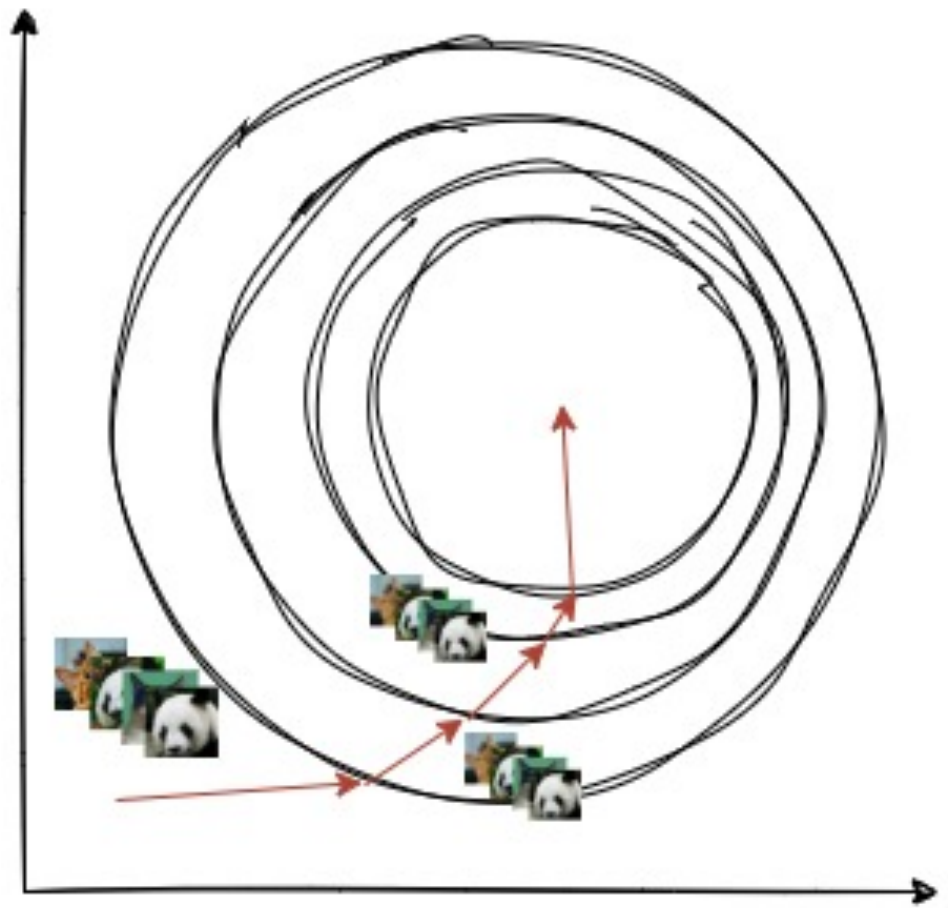
Source doi: 10.1109/UEMCON51285.2020.9298092

We use the gradient to update the weights iteratively:

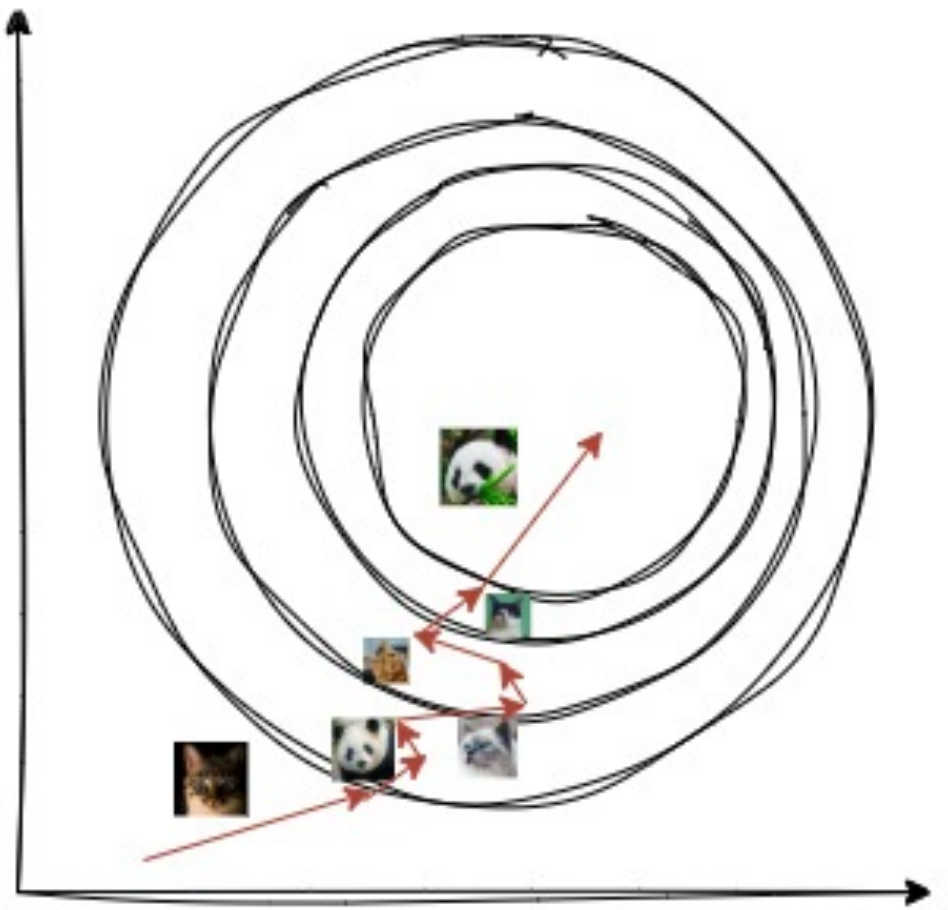
$$w_{ij} \leftarrow w_{ij} - \eta \cdot \frac{\partial L}{\partial w_{ij}}$$

where η is the **learning rate**.

Stochastic Gradient Descend

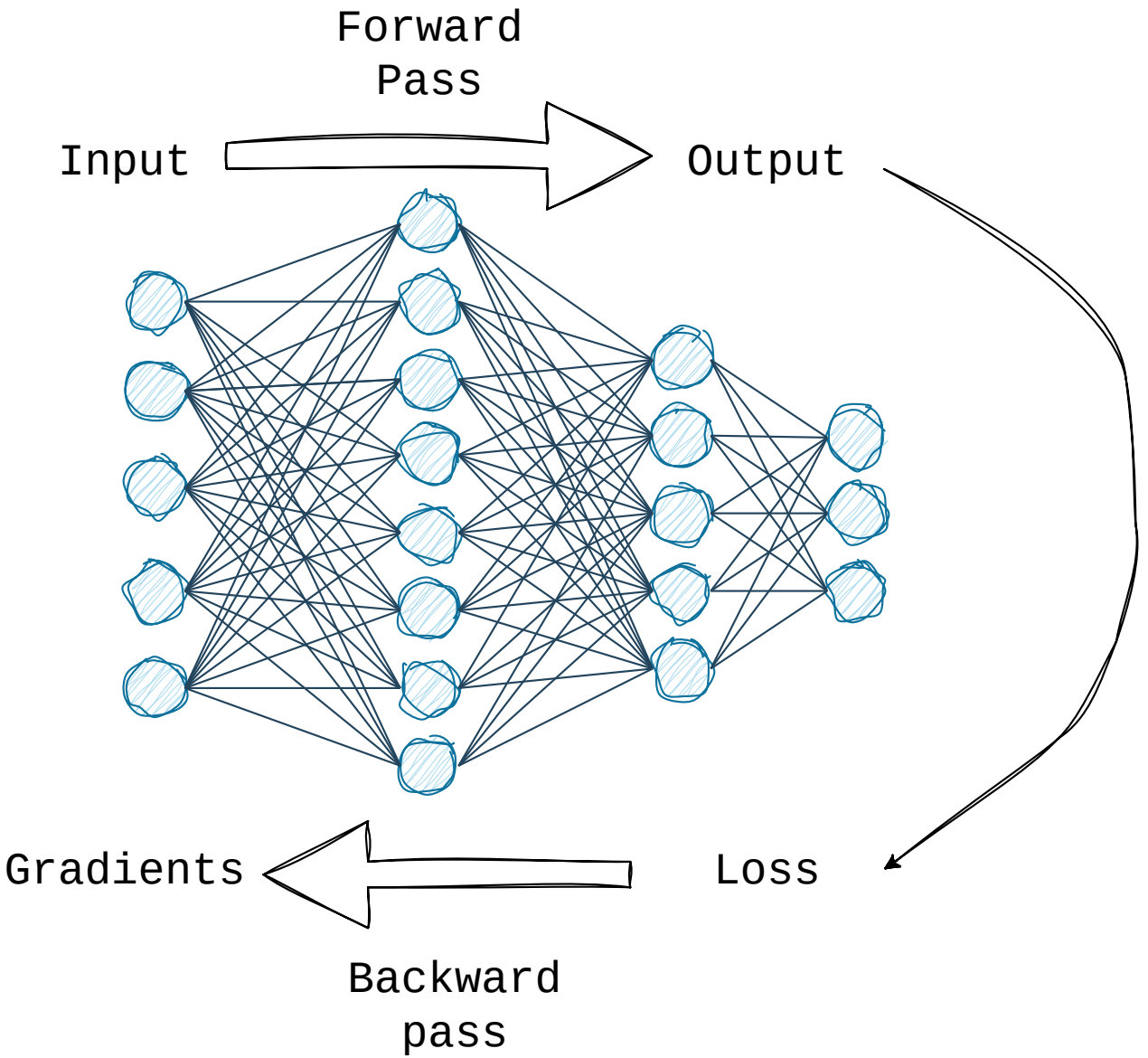


Gradient descent



Stochastic
Gradient descent

Training loop



Hypter parameters

We as humans need to set some hyperparameters before training:

- Number of epochs
- Learning rate
- Batch size
- Optimizer type
- Loss function
- Model architecture (number of layers, neurons, etc.)

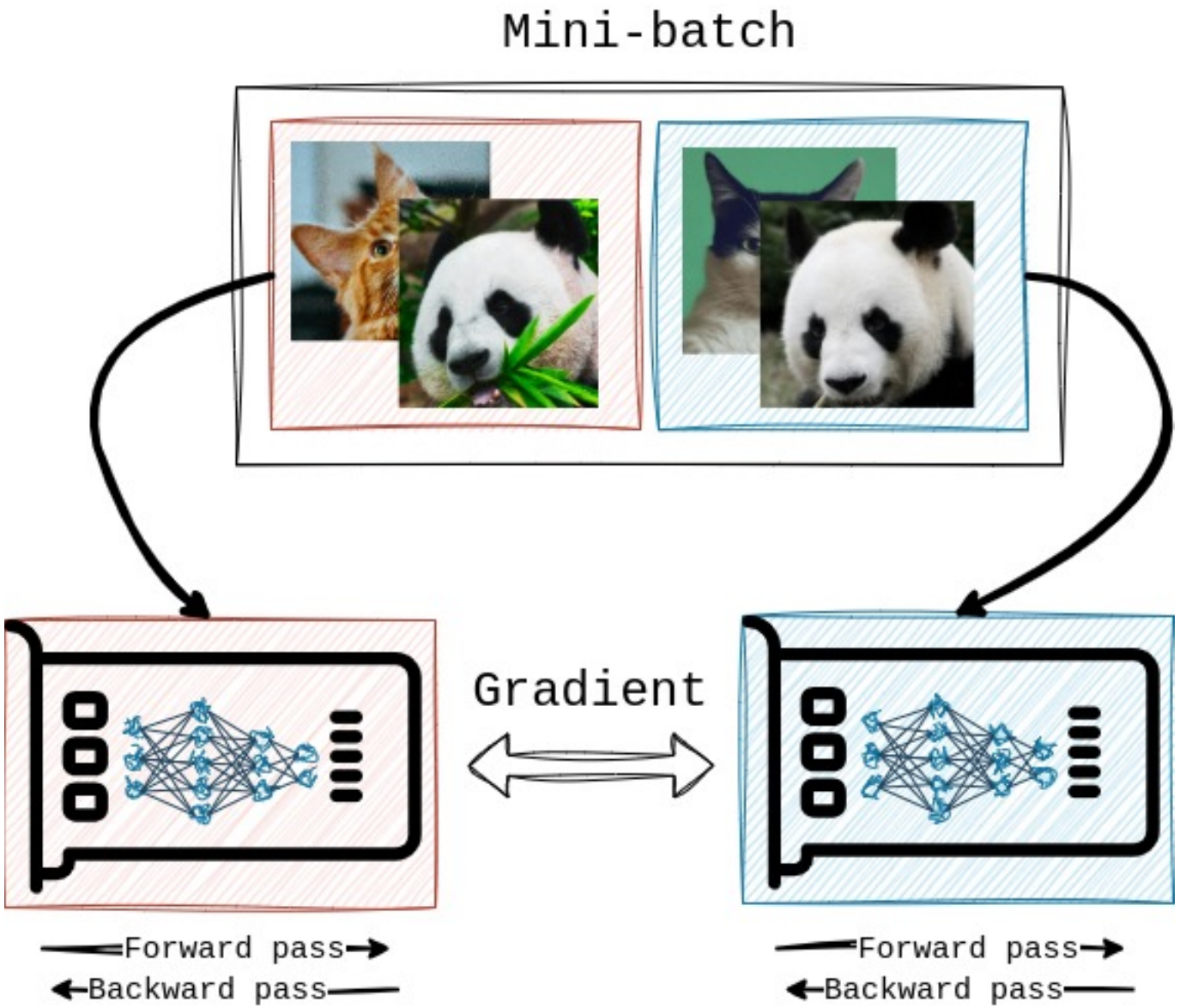
Chapter 3: Let’s go in parallel

Parallelism paradigms

- **Data** parallelism
- **Model** parallelism
- **Pipeline** parallelism
- **Tensor** parallelism

These are just concepts, there are no standard on their names !

Data parallelism



From 0 to distributed training

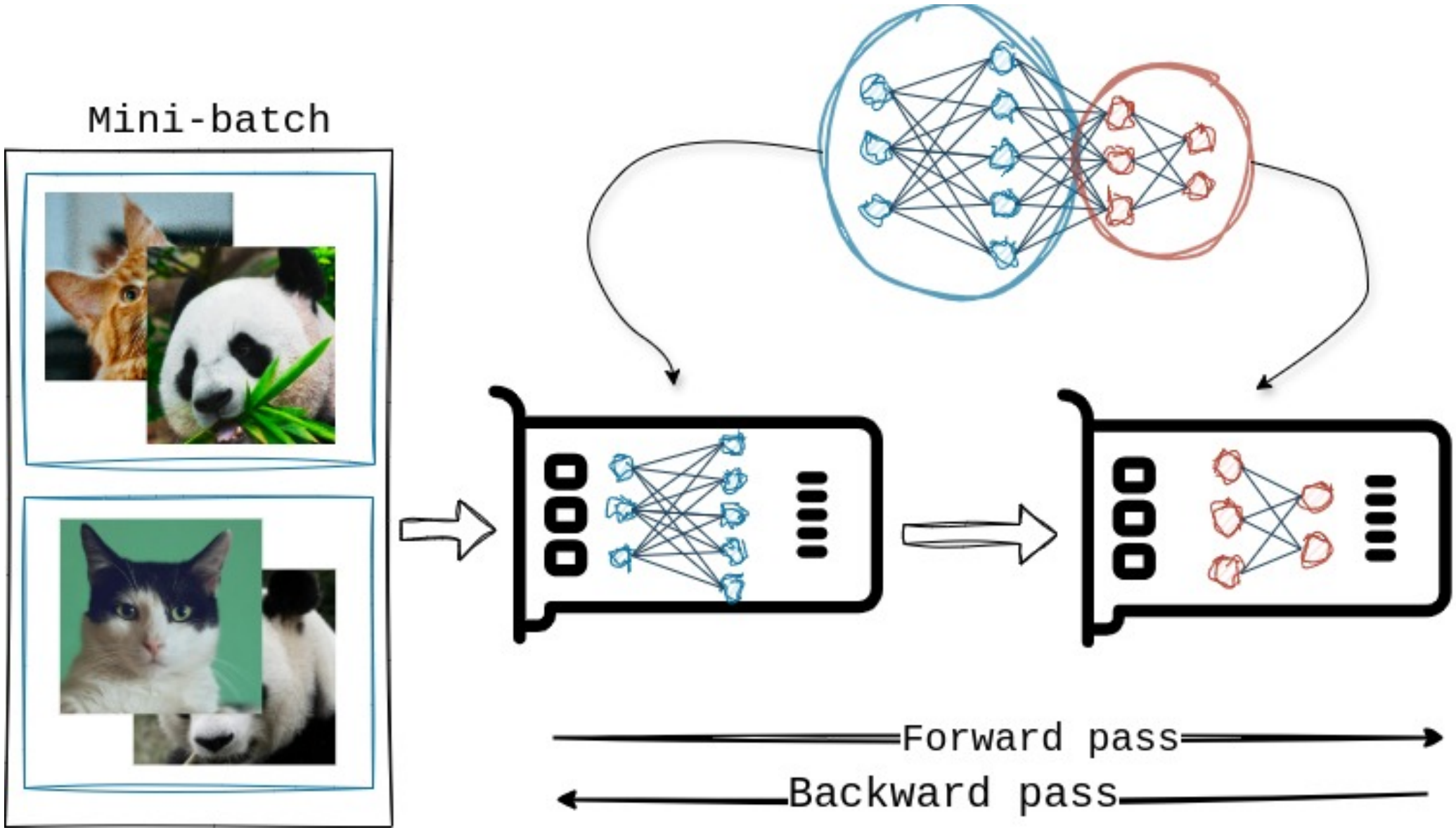
Data parallelism

- **Threoretical** speedup: N times faster with N GPUs.
- Each GPU gets a slice of the data and computes the gradients independently.
- After each batch, gradients are averaged across GPUs.

Limitations:

- **Memory constraints**: each GPU must fit the entire model.
- **Communication overhead**: synchronizing gradients can be costly.

Model parallelism



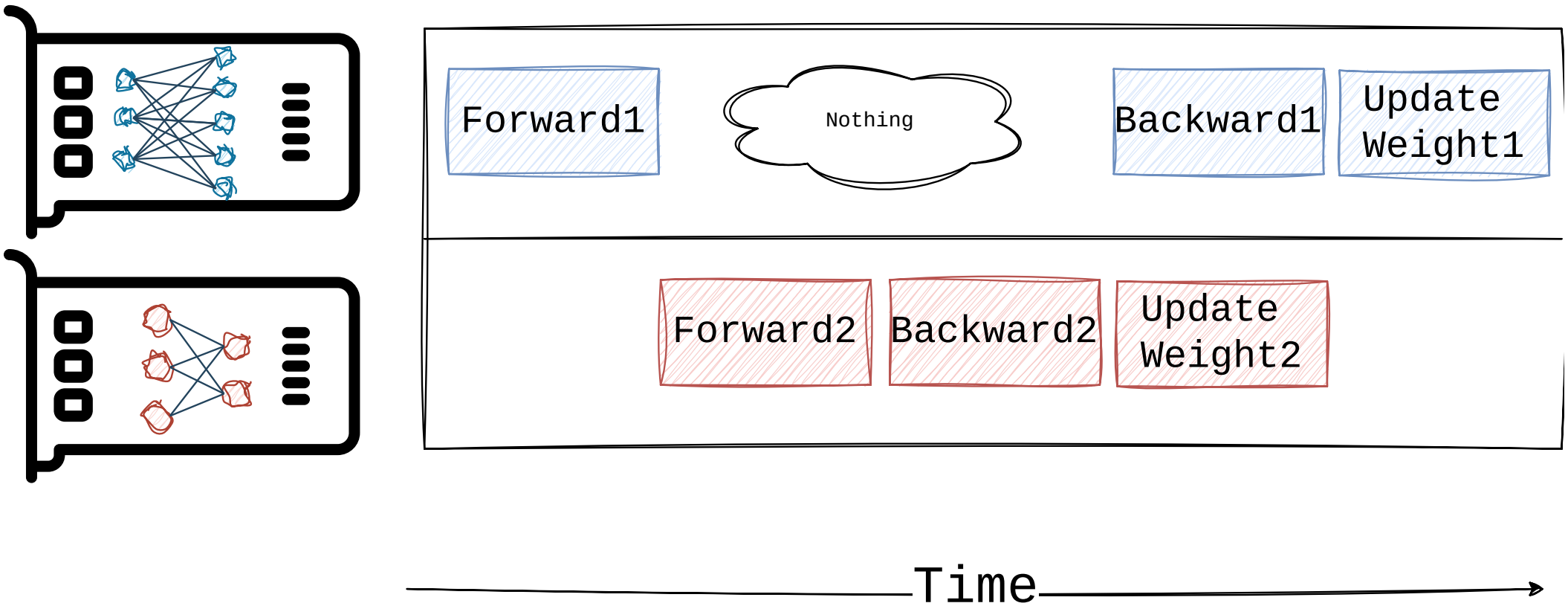
Model parallelism

- The maximum model size scales with the number of GPUs.
- No need to communicate gradients.
- Each GPU gets a slice of the model and computes the gradients independently.

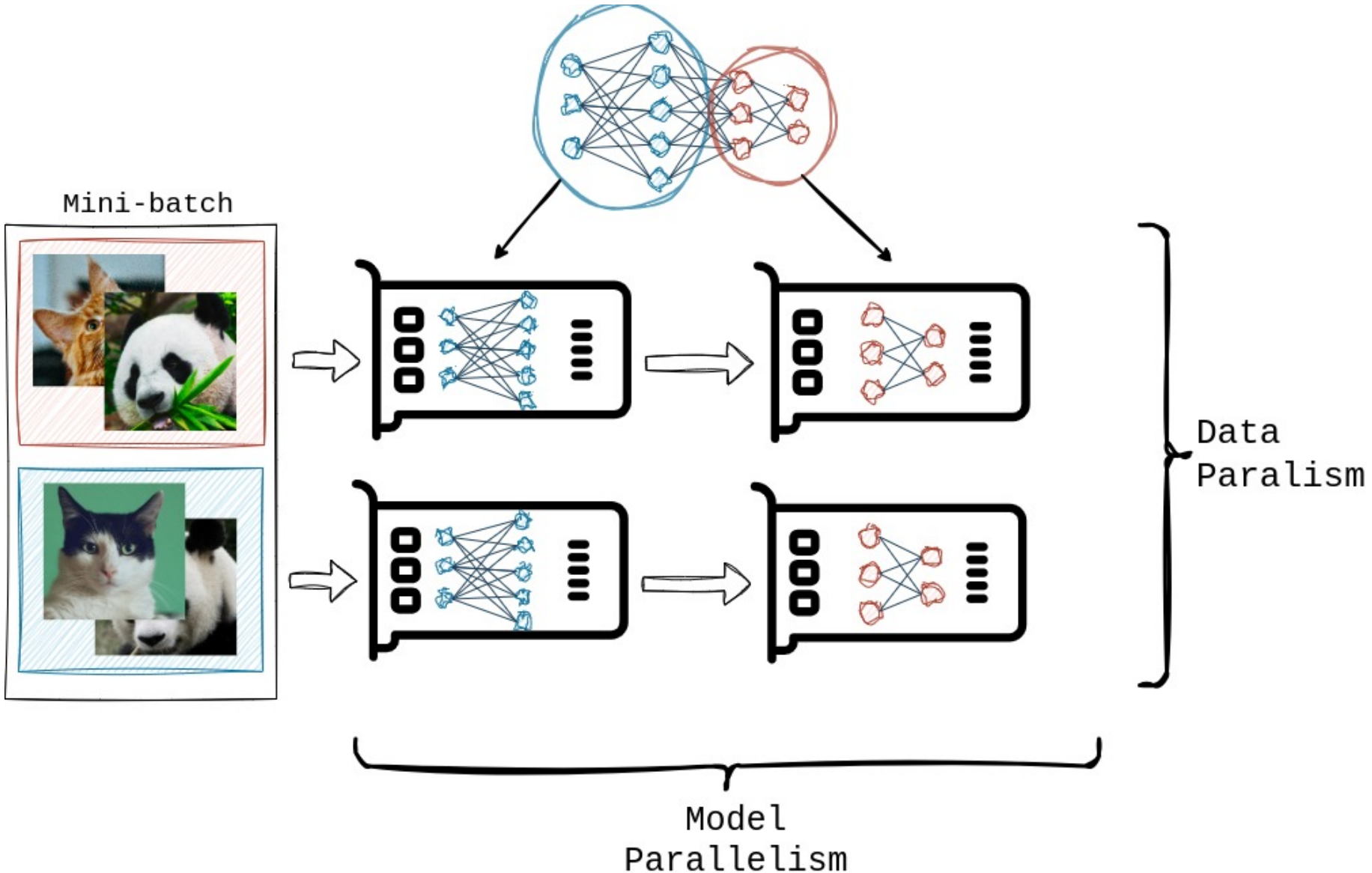
Limitations

- No speedup for training time.
- Each GPU must wait for the others to finish.

Model parallelism



2D parallelism

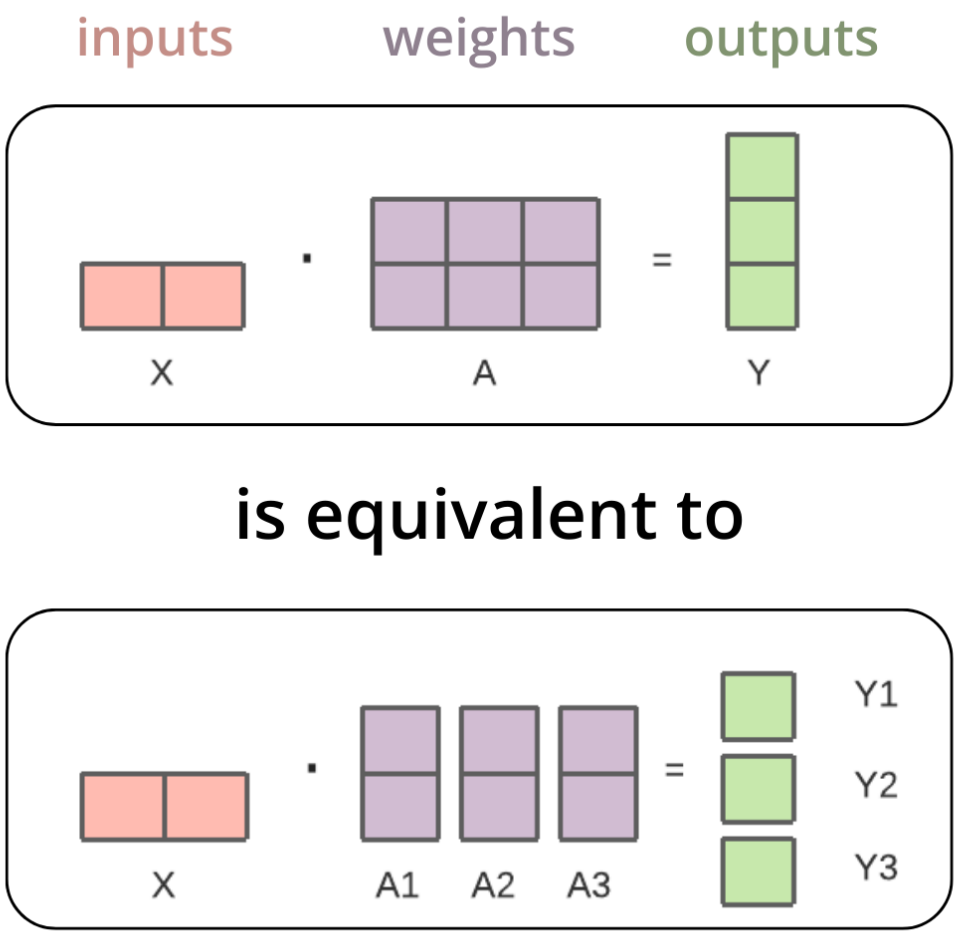


Pipeline parallelism

Split the model into stages, each stage runs on a different GPU.

But run multiple batches in parallel avoiding idle time !

Tensor parallelism



Tensor parallelism: source huggingface.co

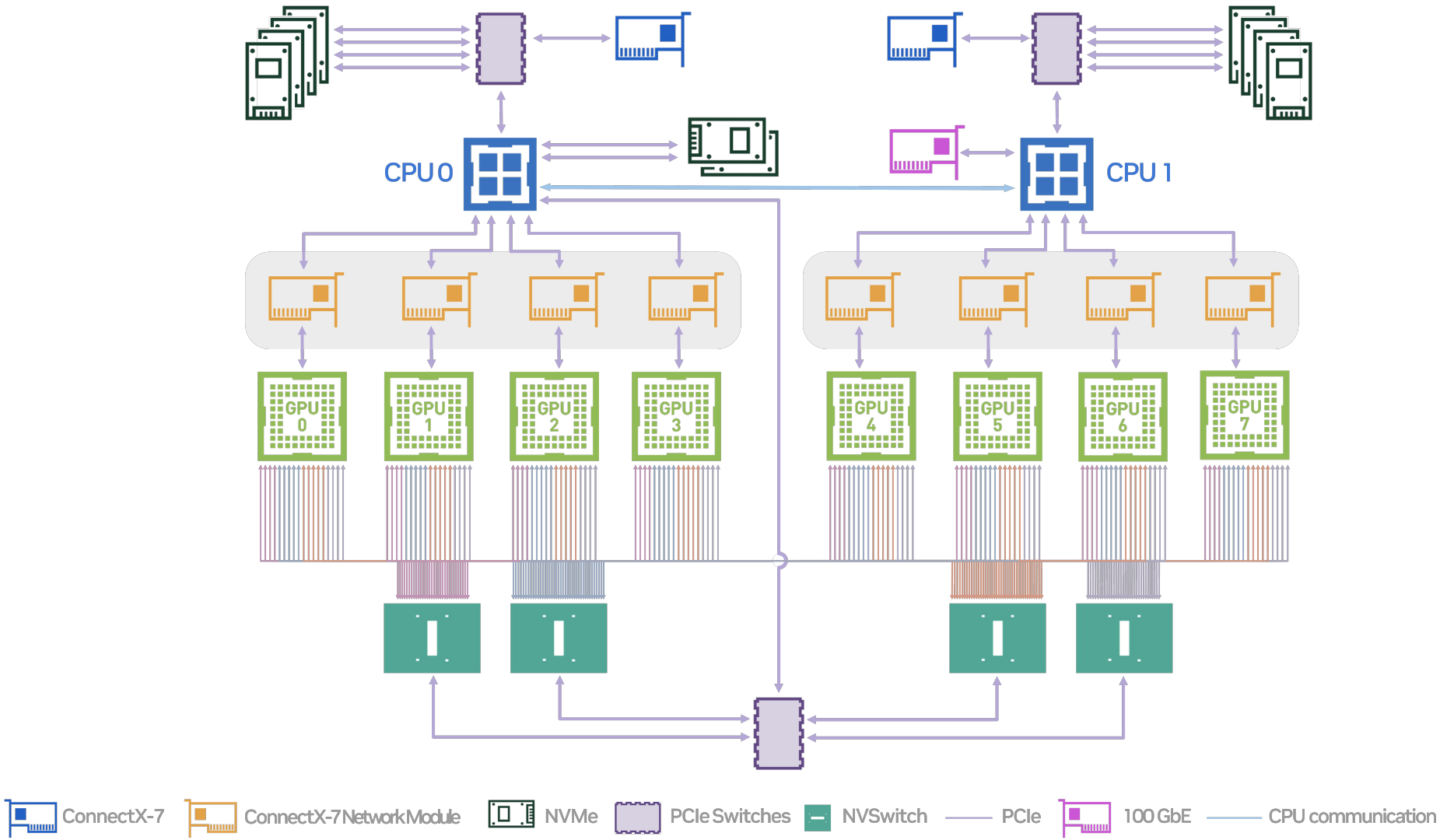
Technologies

This form of parallelism has influenced the design of both **hardware** and **software** systems:

- GPUs within the same node are interconnected via specialized **high-speed fabric** offering up to **900 GB/s** bandwidth (yes, gigabytes per second).

- Each node is equipped with high-performance network adapters enabling **Remote Direct Memory Access (RDMA)** across nodes, with bandwidths reaching **100 GB/s**.

Hardware Topology



NVIDIA DGX H100 Topology

Software Stack

Built on top of this advanced hardware are highly optimized software libraries designed to minimize communication overhead and maximize performance.

- **PyTorch** relies on **NCCL** (NVIDIA Collective Communications Library), which is topology-aware and fabric-optimized.
- NCCL intelligently manages inter-GPU communication, leveraging hardware capabilities to streamline both **data movement** and **synchronization**.