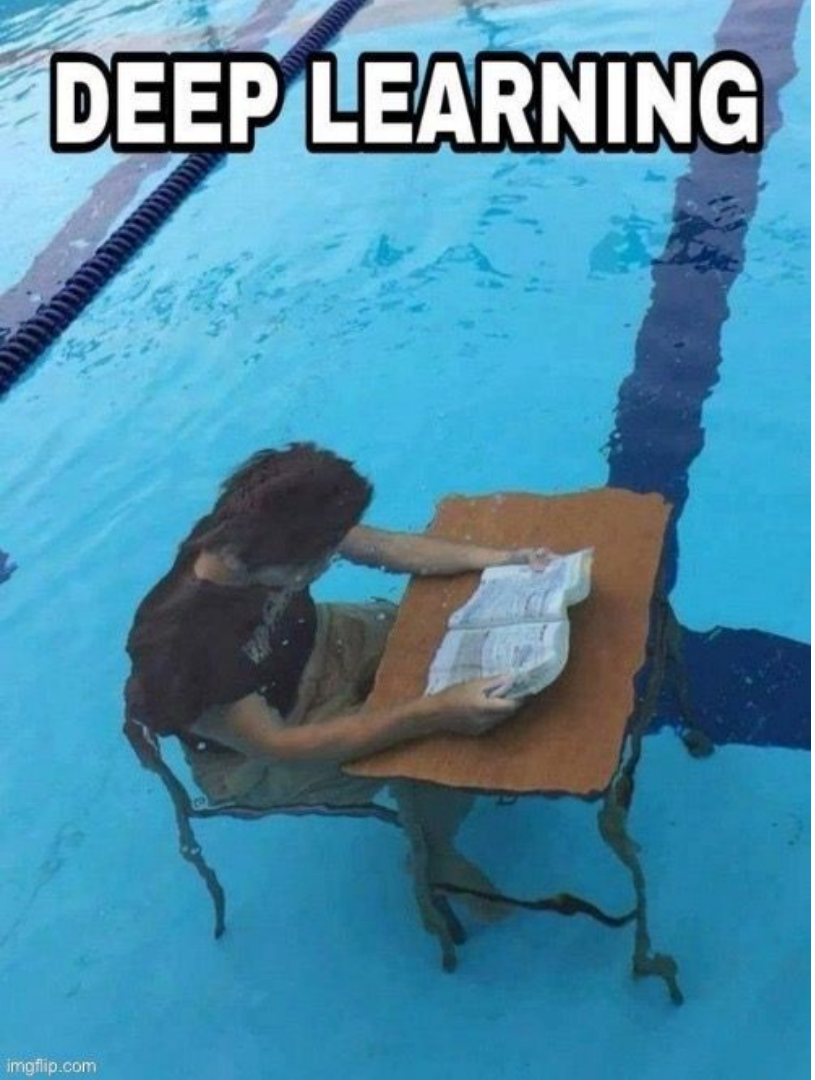


Deep Learning

Machine Learning Course, Day 3

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Syllabus

Part I: The World of Deep Learning

- Why Deep Learning?
- Two conceptual tricks that made Deep Learning possible
- Train your DL model in less than a minute - fast.ai hands-on
- Vibe Coding for SHAP values demo
- Prompt engineering

Part II: Neural Networks Dive-In

- A Neural Network at Work
- Convolutional Neural Networks
- Hands-on. Introduction to Pytorch.
    - A Basic Neural Network
    - Binary Classification of Tabular Data
    - Binary Classification of Images using CNNs

About us…

Anton Popov

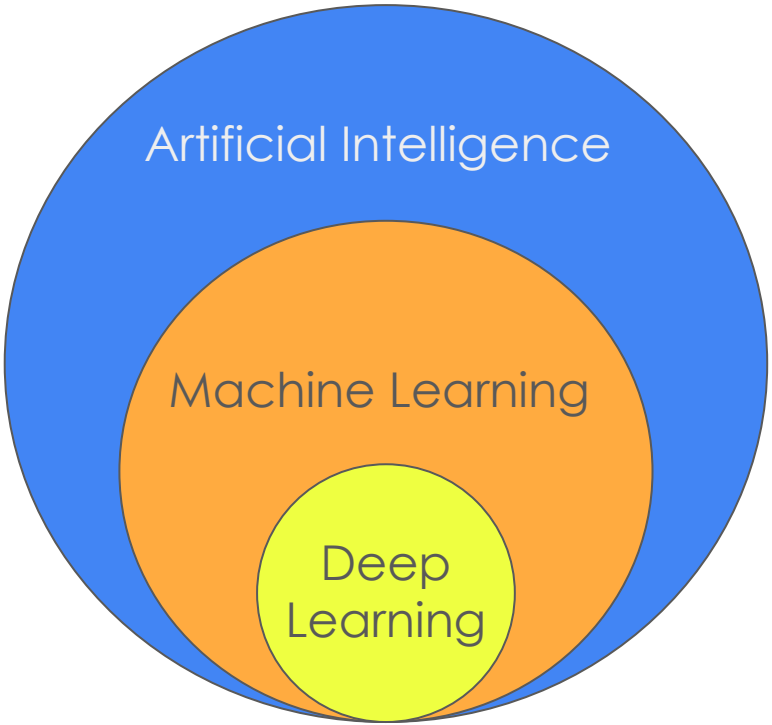
- 6+ years of experience of developing AI/ML solutions in start-ups, Big Pharma and academia
- Research Engineer @ BSC, part of ML4BMR Unit
- Applied Mathematician by training
- Top-down learner (Joker)

Guillermo Prol Castelo

- 3rd year PhD student in AI for synthetic cancer data generation
- Physicist by training, bioinformatician by testing :)
- Bottom-up learner (Batman)

Part I. The World of Deep Learning

Recap: AI/ML/DL



- Artificial Intelligence (AI): attempt to mimic human problem-solving and decision-making abilities

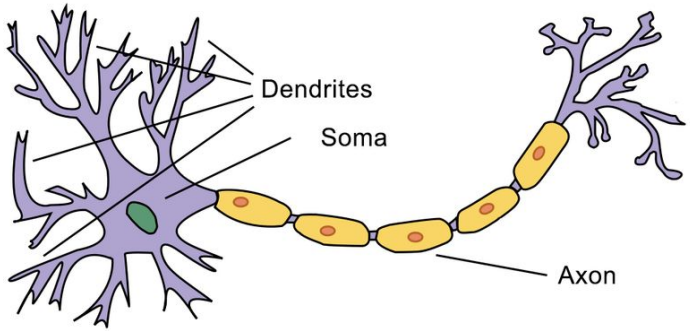
Alan Turing: "Can machines think?" → Turing test (1950)

- Machine Learning (ML): AI that gradually improves its accuracy—i.e., the AI that learns

- **Deep Learning** (DL): ML that uses multi-layered (deep) neural networks to learn

Deep Learning uses ANNs as a model

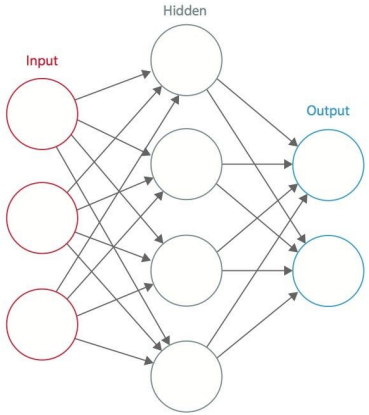
Biological Neural Networks



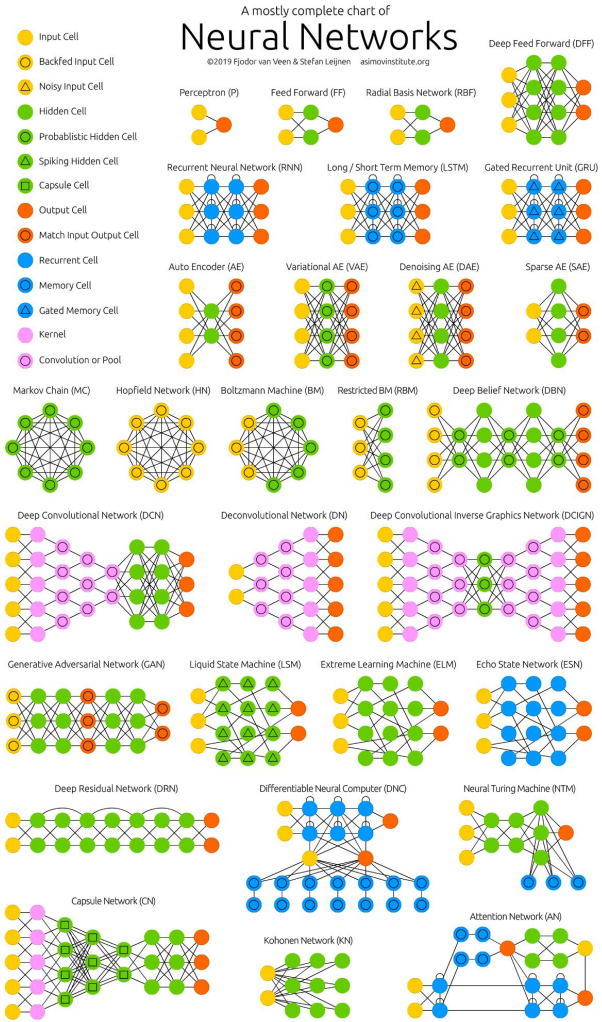
- Real biological organization of biological cells
- Hebbian learning inspired training

inspires

Artificial Neural Networks (ANNs)



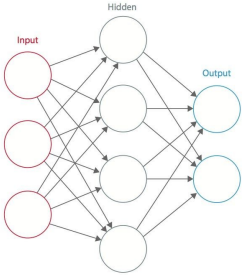
- Mathematical abstraction
- Infinitely flexible function
- Uses Gradient descent (for DL)



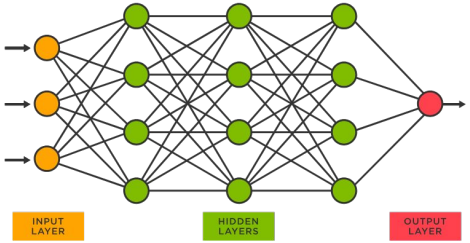
There are a lot of types of ANNs…

But today we are talking about Deep Neural Nets:

1. They have more than one hidden layer



Not a Deep Neural Network



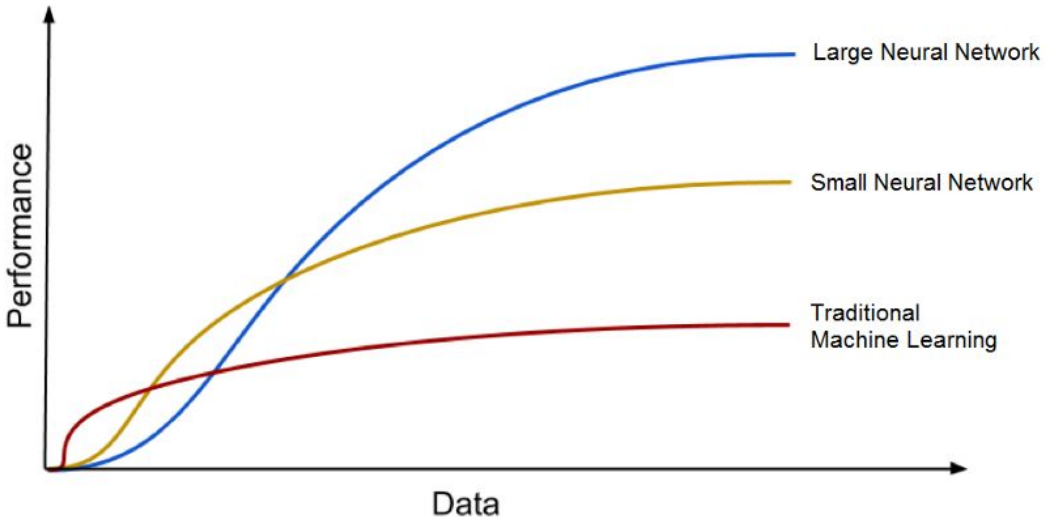
Deep Neural Network

2. They use Gradient Descent (*)

(*) most commonly used optimization method

https://www.asimovinstitute.org/neural-network-zoo/; https://www.youtube.com/watch?v=jmmW0F0biz0

Deep Learning VS Machine Learning



https://builtin.com/artificial-intelligence/ai-vs-machine-learning

Deep Learning algorithms perform better on larger datasets than traditional ML.

And now we have:

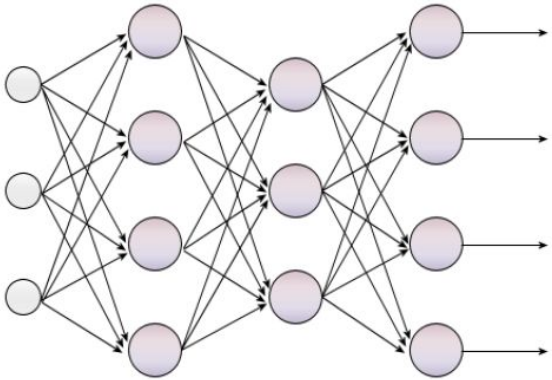
- More data
- More of compute (e.g. Nvidia H100 in MN5) to train bigger models

Bonus of DL:

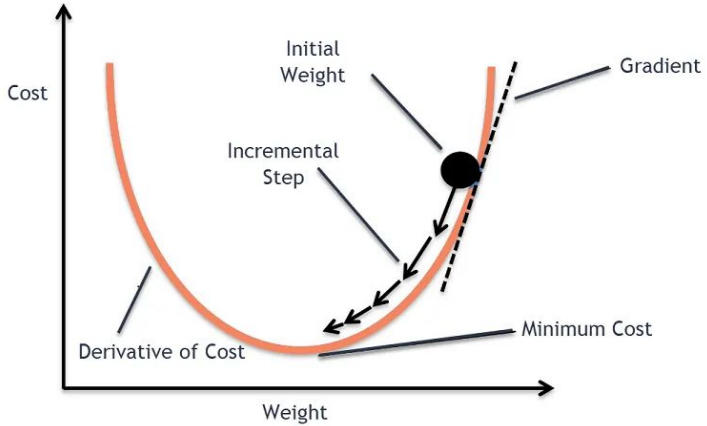
- No features engineering!

Deep Learning models has 2 main tricks

Trick #1: Stacking Layers

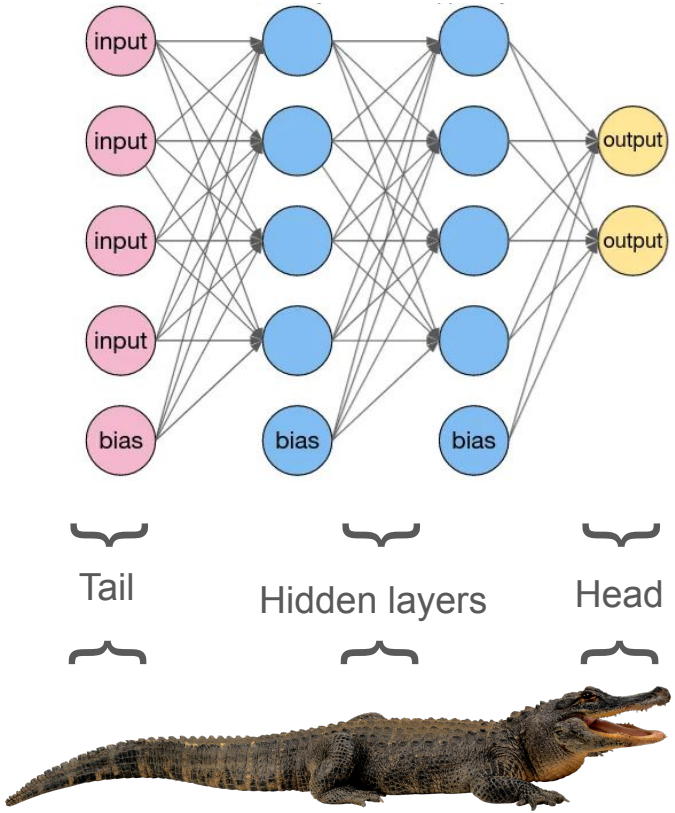


Trick #2: Gradient Descent



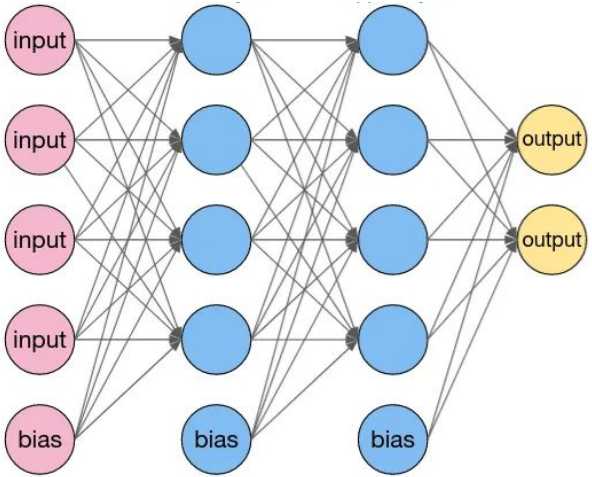
And that's actually it… really!

Feed forward neural network (Multi-layer perceptron)



- Simplest Deep Learning architecture
- Fully connected layers
  all neurons of layer *k* are connected to all neurons of layer *k+1*
- Non-linear activation functions
- Enough to solve all our problems… if we have infinite data and infinite compute

Feed forward neural network (Multi-layer perceptron)

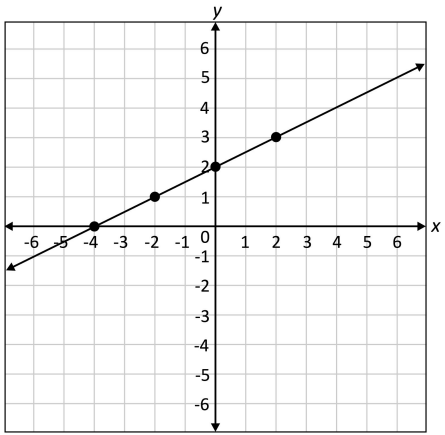


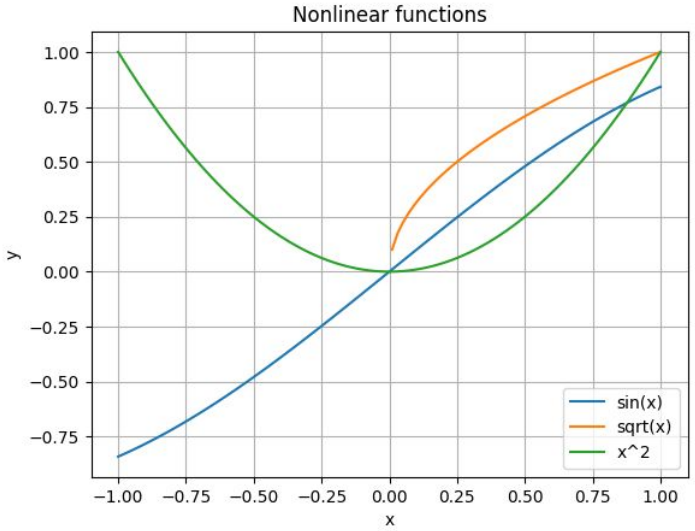
- Simplest Deep Learning architecture
- Fully connected layers
  all neurons of layer *k* are connected to all neurons of layer *k+1*
- Non-linear activation functions
- Enough to solve all our problems… if we have infinite data and infinite compute

Linear vs nonlinear functions

Linear function defines… a line!!!

$$y = k*x + b$$



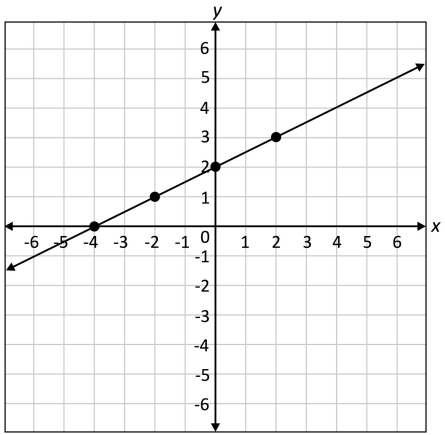


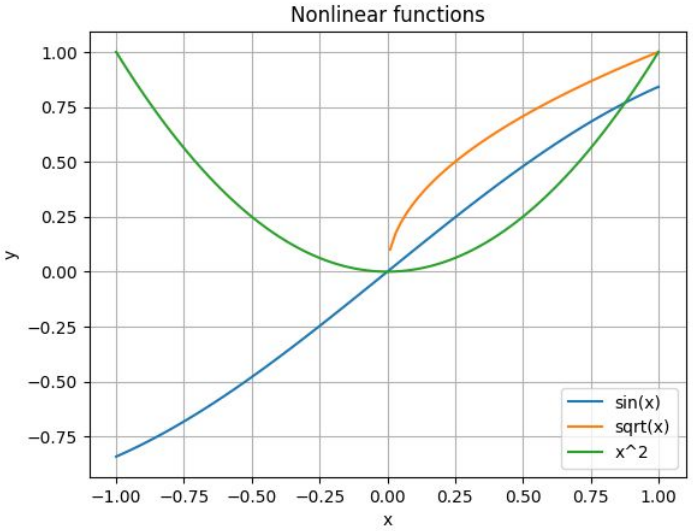
Non linear functions do not define line… but something more complex

Linear vs nonlinear functions

Linear function defines… a line!!!

$y = k*x + b$





Non linear functions do not define line… but something more complex

Hyperplane is an affine subspace of one dimension less than the space it is contained in.

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b$$

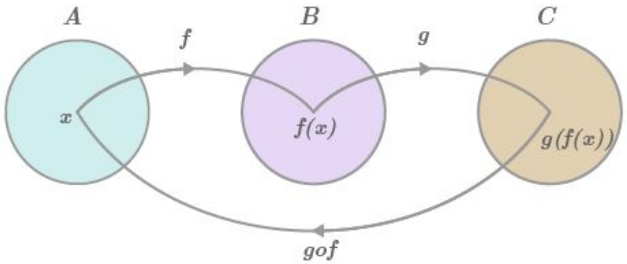
“line” in $R^{(n+1)}$

What will be if we compose 2 linear functions?

Ok, let's imagine we have two linear functions:

$$y_1 = k_1 * x_1 + b_1,$$
$$y_2 = k_2 * x_2 + b_2$$

What type will have the composition of these functions : $y_2(y_1(x_1))$?



Functions composition

What will be if we compose 2 linear functions?

Ok, let's imagine we have two linear functions:

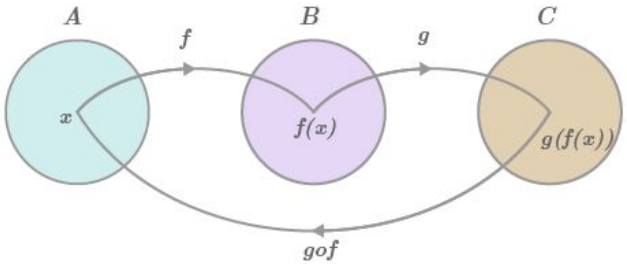
$$y_1 = k_1 * x_1 + b_1,$$
$$y_2 = k_2 * x_2 + b_2$$

What type will have the composition of these functions :

$$y_2(y_1(x_1))?$$

$y_2 = k_2 * x_2 + b_2 =$ | *let's substitute* $x_2 = k_1 * x_1 + b_1$| = =
$k_2 * (k_1 * x_1 + b_1) + b_2 = \underbrace{k_2 * k_1}_{k_3} * x_1 + \underbrace{k_2 * b_1 + b_2}_{b_3} =$

$= k_3 * x_1 + b_3$ <- linear function!



Functions composition

What will be if we compose 2 linear functions?

Ok, let's imagine we have two linear functions:

$$y_1 = k_1 * x_1 + b_1,$$
$$y_2 = k_2 * x_2 + b_2$$

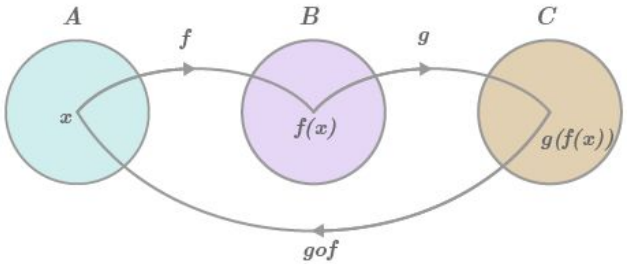
What type will have the composition of these functions :

$$y_2(y_1(x_1))?$$

$y_2 = k_2 * x_2 + b_2 =$ | *let's substitute* $x_2 = k_1 * x_1 + b_1$| = =

$k_2 * (k_1 * x_1 + b_1) + b_2 = \underbrace{k_2 * k_1}_{k_3} * x_1 + \underbrace{k_2 * b_1 + b_2}_{b_3} =$

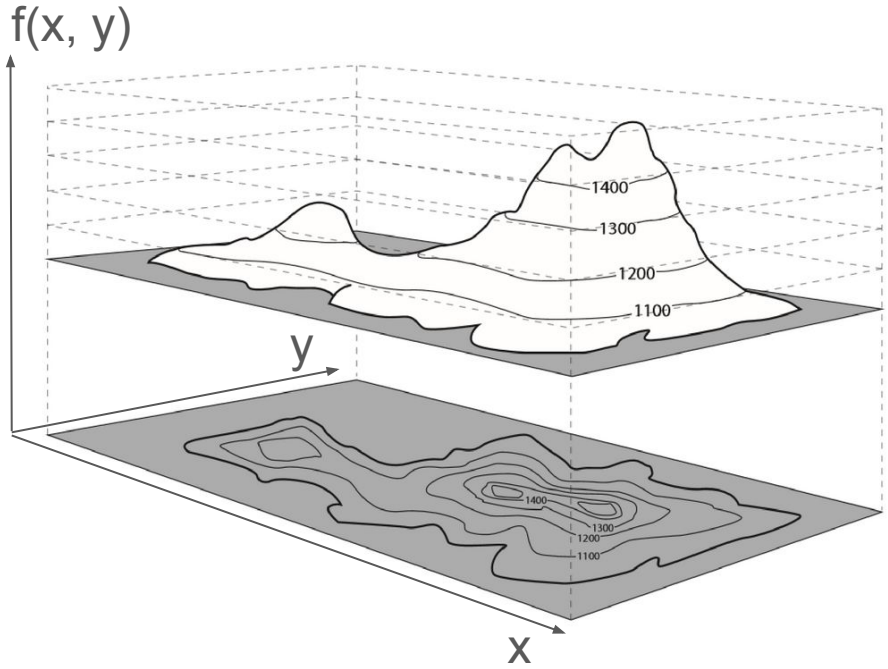
$= k_3 * x_1 + b_3$ <- linear function!



Functions composition

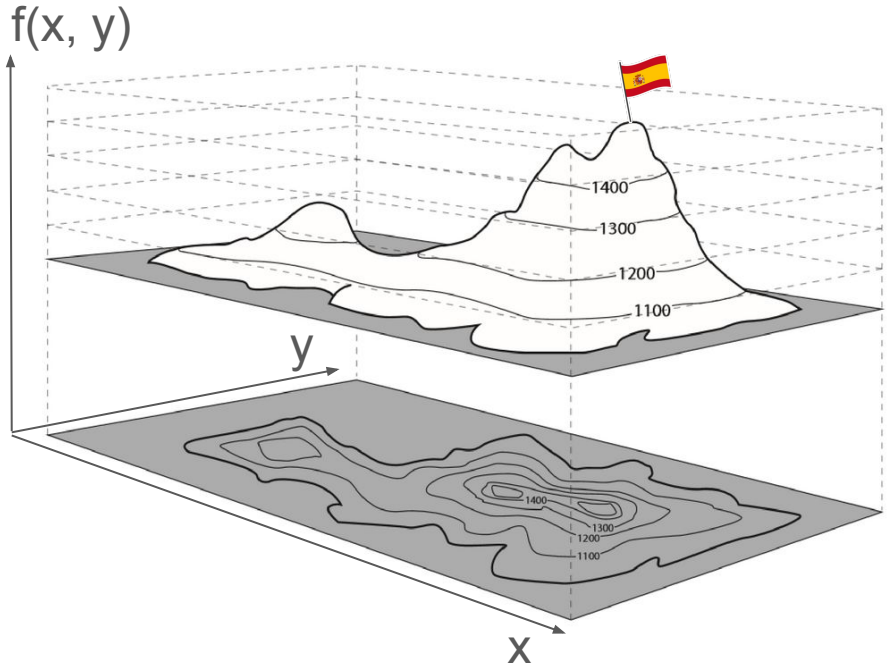
This was the trick #1:
stacking ~~functions~~ layers

Level curves and contour plots



Where is the maximum of the function on the left?

Level curves and contour plots

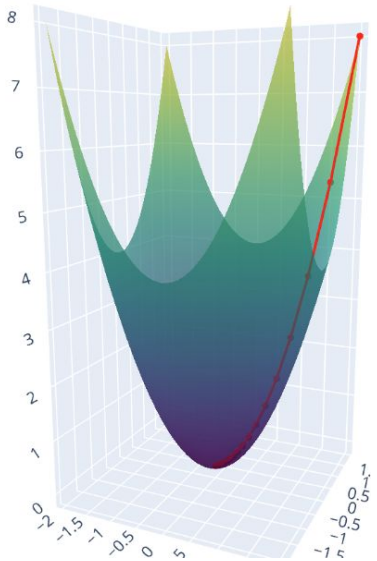


Where is the maximum of the function on the left?

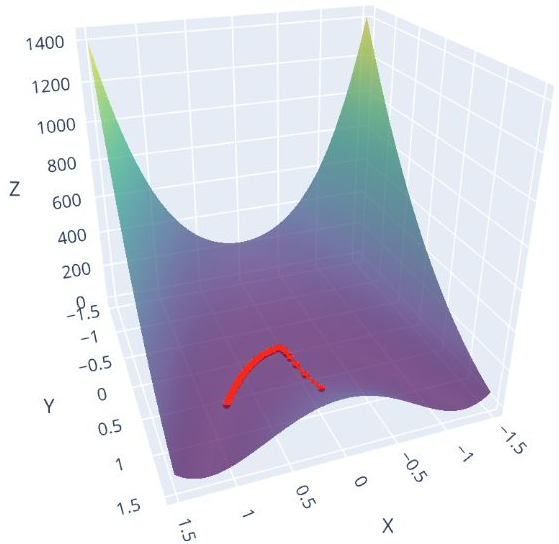
We have a good intuition to “grasp” the surface of function of 2 arguments…

But for functions with more than 2 arguments we need another methods.

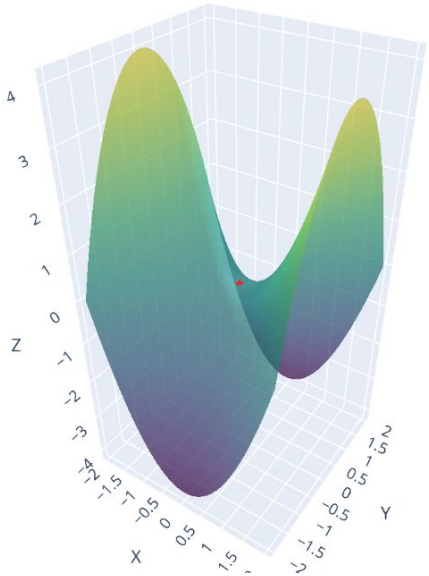
Gradient descent demo



Quadratic function

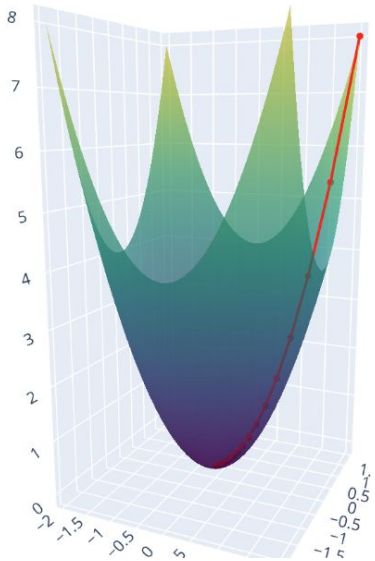


Rosenbrock function

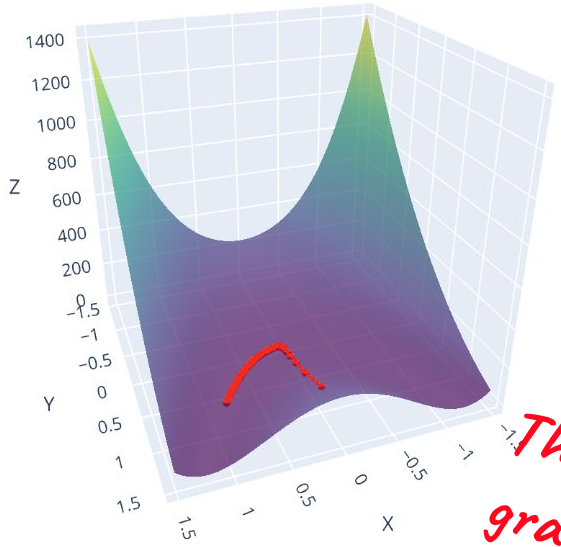


Saddle function

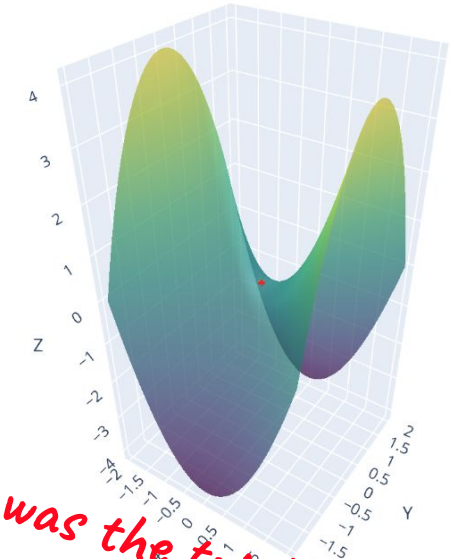
Gradient descent demo



Quadratic function



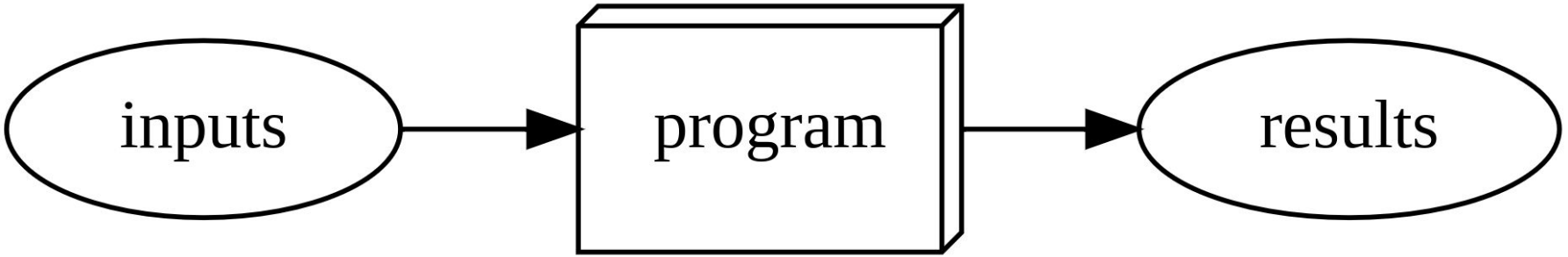
Rosenbrock function



Saddle function

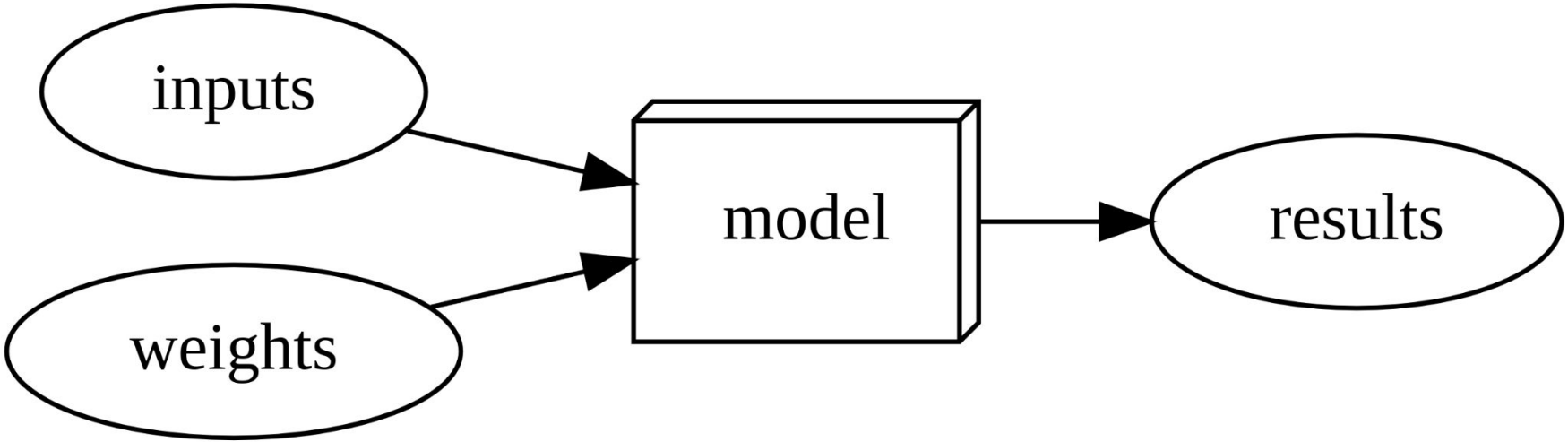
This was the trick #2: gradient descent

Traditional problem-solving approach



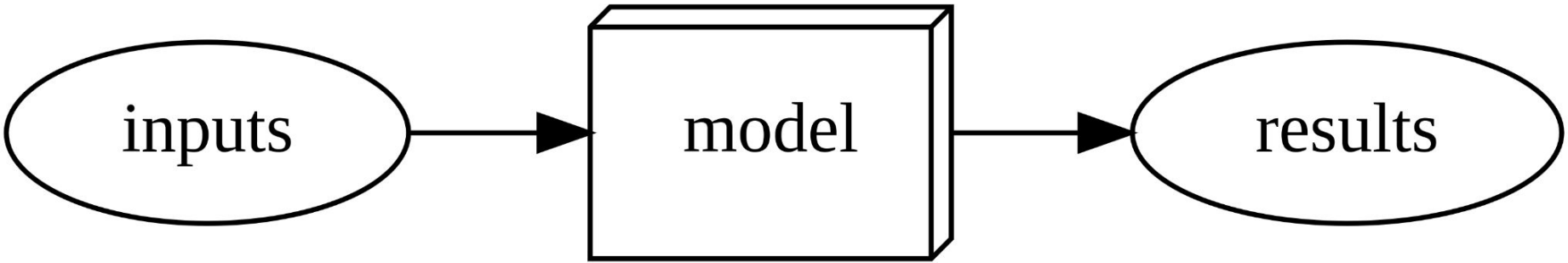
- You manually define each step of the program to solve the problem.
- But how to define “cat” if we want to classify images?

Weight assignment



Instead of telling the computer the exact steps required to solve a problem, show it examples of the problem to solve, and let it figure out how to solve it itself

Using a trained model as a program



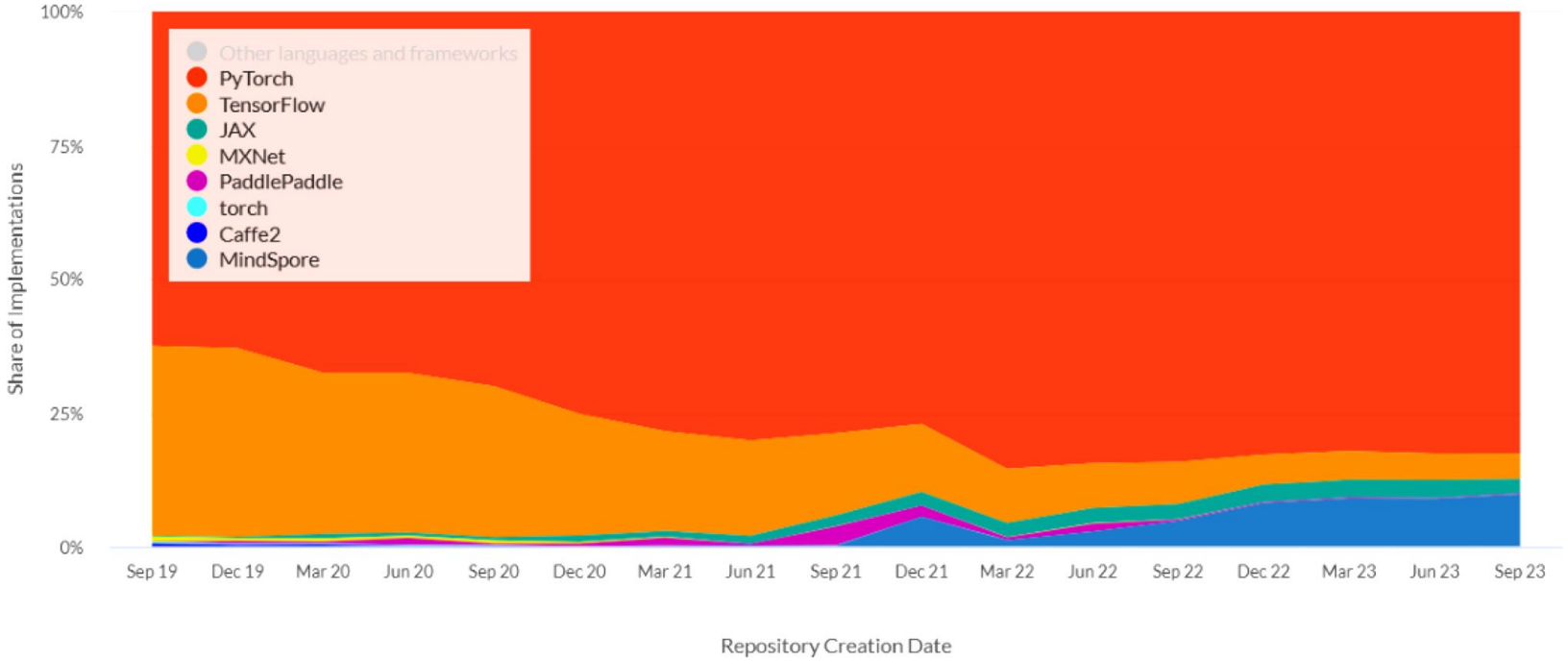
After adjusting model's parameters we use it with the set of optimal params.

- "Training" is a process of adjusting parameters based on the data we have
- "Inference" is a process of using the model with optimal parameters after the training completed

Framework	GitHub stars (May 2025)	Research Adoption	Industry Adoption	Notable Strengths
PyTorch	83.7k	Very High	Growing	Intuitive, research-friendly
TensorFlow	181k	Moderate	Very High	Mature, production-ready
Keras	62.2k	Moderate	High	Easy to use, integrated w/ TF
JAX	31.6k	Growing	Low	High performance, differentiable
PyTorch Lighting	28.5k	Growing	Low	PyTorch abstraction, fast prototyping
PaddlePaddle	22.8k	Growing (Asia)	Growing (Asia)	Speed, pretrained models, industry use
fast.ai	26.2k	Niche	Niche	High-level API on PyTorch, easy prototyping

Framework	GitHub stars (May 2025)	Research Adoption	Industry Adoption	Notable Strengths
PyTorch	83.7k	Very High	Growing	Intuitive, research-friendly
TensorFlow	181k	Moderate	Very High	Mature, production-ready
Keras	62.2k	Moderate	High	Easy to use, integrated w/ TF
JAX	31.6k	Growing	Low	High performance, differentiable
PyTorch Lighting	28.5k	Growing	Low	PyTorch abstraction, fast prototyping
PaddlePaddle	22.8k	Growing (Asia)	Growing (Asia)	Speed, pretrained models, industry use
fast.ai	26.2k	Niche	Niche	High-level API on PyTorch, easy prototyping

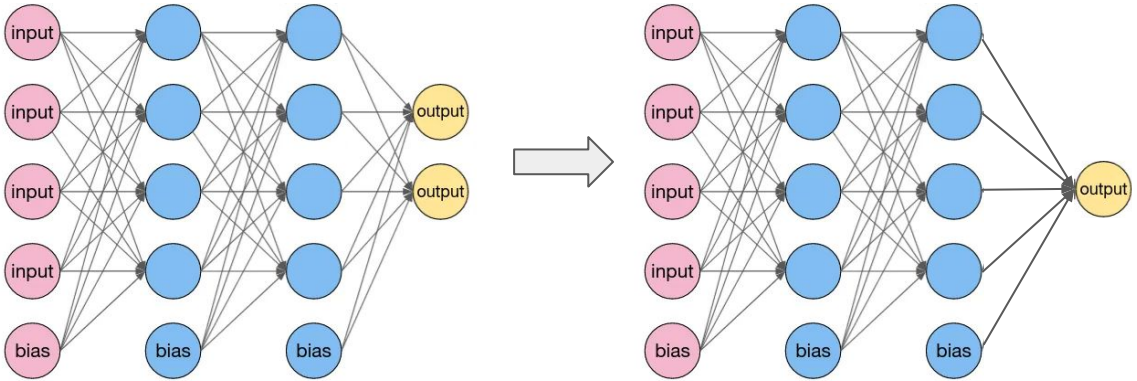
Trends of paper implementations grouped by framework



https://viso.ai/deep-learning/pytorch-vs-tensorflow/

fast.ai demo

Transfer learning



Transfer learning: using a pretrained model for a task different to what it was originally trained for.

So we replace the last layers of our network (“head”) and train only those on our limited data. The rest of the network is “frozen” i.e. untouched during the backpropagation.

Deep Learning Jargon

- The functional form of the *model* is called its *architecture* (but be careful—sometimes people use *model* as a synonym of *architecture*, so this can get confusing).

- The *weights* are called *parameters*.

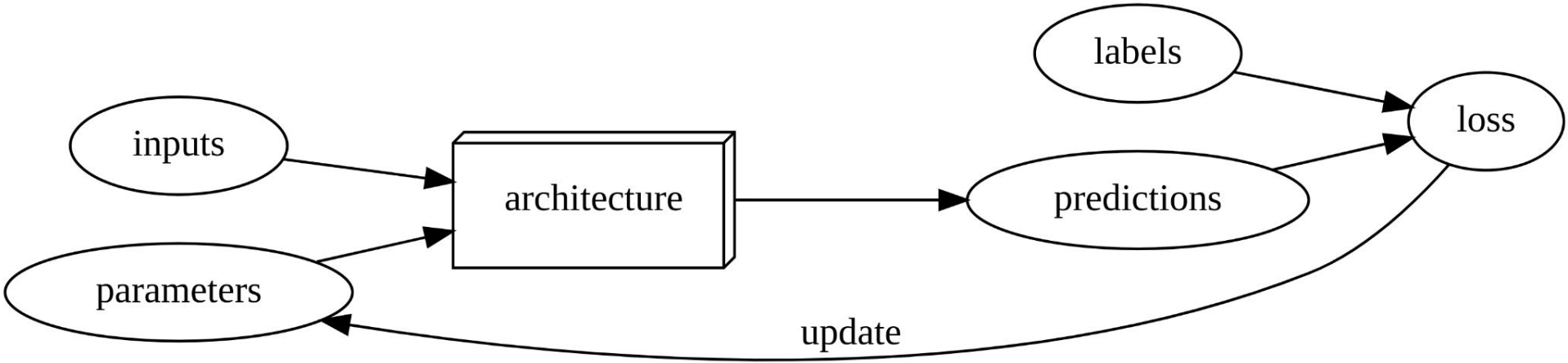
- The *predictions* are calculated from the *independent variable*, which is the *data* not including the *labels*.

- The *results* of the model are called *predictions*.

- The measure of *performance* is called the *loss*.

- The loss depends not only on the predictions, but also the correct *labels* (also known as *targets* or the *dependent variable*); e.g., "dog" or "cat."

Deep Learning architecture with modern jargon



How to evaluate quality?

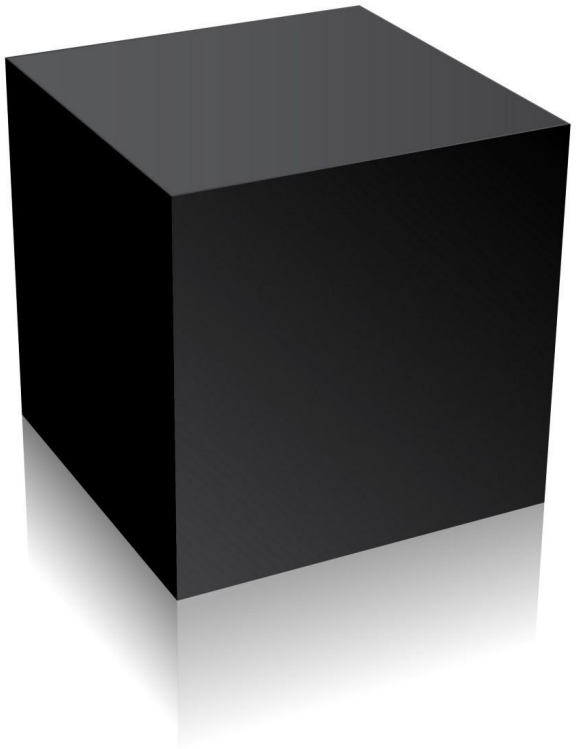
Click image above to play video

Captain - CNN that sees only local features

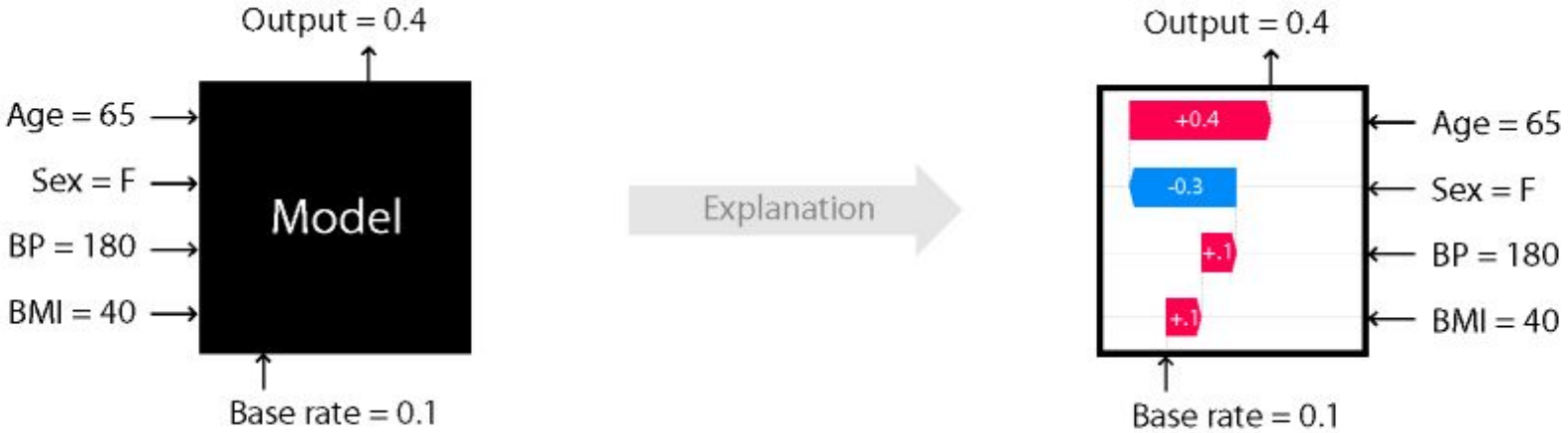
Art expert - Transformers model - has a global context

Guy in the balcony - DL expert who knows how to evaluate those models

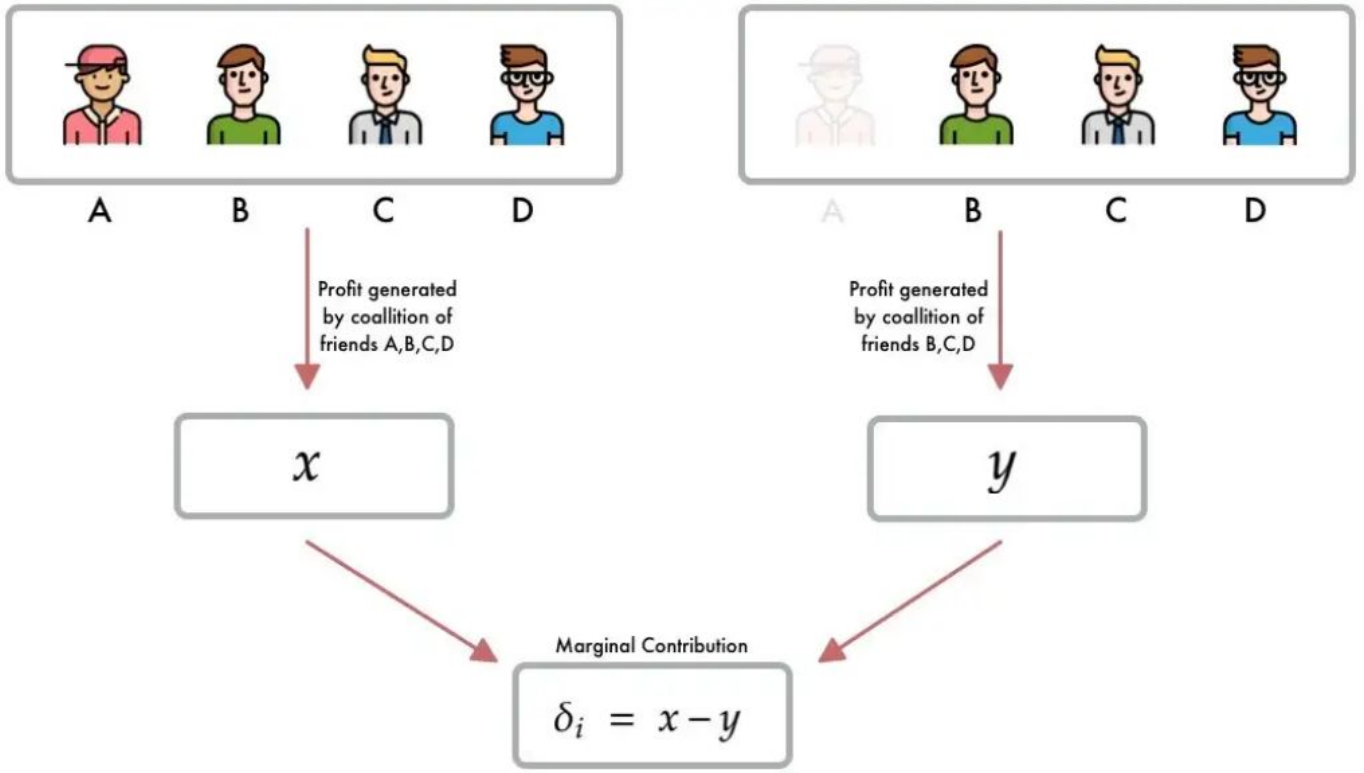
Black boxes



SHAP (Shapley Additive Explanations)

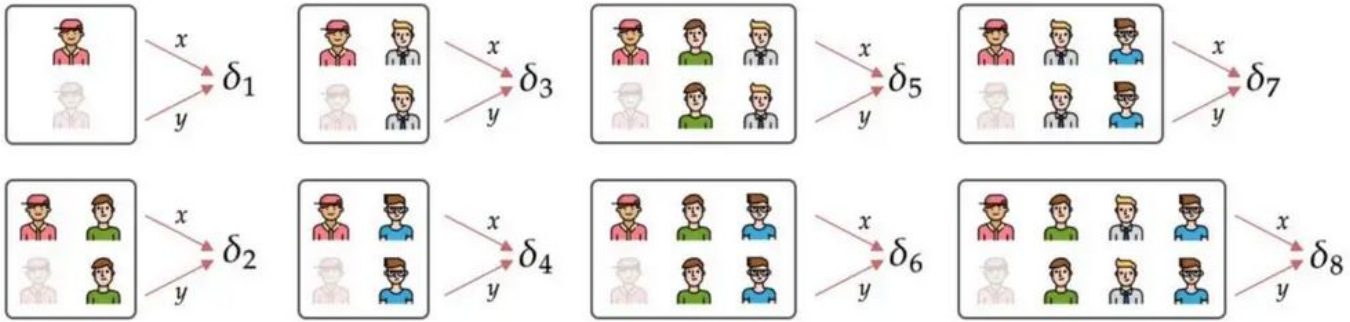


SHAP (Shapley Additive Explanations)



source:

SHAP (Shapley Additive Explanations)



The Shapley value for member 

is given by:

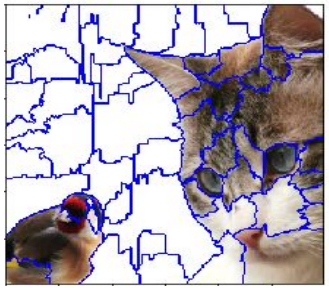
$$\phi_i = \frac{\delta_1 + \delta_3 + \delta_4 + \delta_5 + \delta_6 + \delta_7 + \delta_8}{8}$$

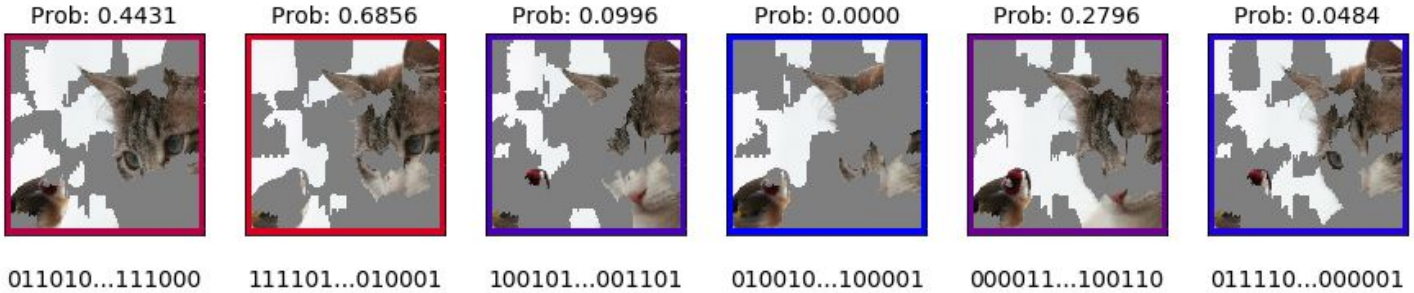
source: https://tutegomeze.medium.com/shap-shapley-additive-explanations-9a0d0e7e63d5

A practical implementation: KernelSHAP

Class	Probability
Tabby cat	**0.4271**
Goldfinch	0.0986
Egyptian cat	0.0437
Tiger cat	0.0162
Thimble	0.0126

Positive and **Negative** contributions in the explanation of the class: **Tabby cat.**





SHAP values demo: GIT + ChatGPT

- Code assistants & Git version control allow you to experiment and learn quicker, without worrying to lose results.
- And separate Python environment/docker image allow you not worry about messing up

LLMs are nice, but sometimes a bit unpredictable… like minions!

Prompt engineering 101

Prompt engineering is the art of crafting effective instructions (prompts) to guide AI models, especially Large Language Models (LLMs), toward generating desired responses.

Define LLMs role in initial system prompt.	“You are the best in the world AI engineer and gifted teacher.”
Use meta-prompting in your initial prompt, ask how to prompt LLM to simplify its job.	“What additional information should I provide so you can answer my question in the best possible way”
Use delimiters to separate different types of text: instructions, code, error messages.	“I have the following code: ``` shap.summary_plot(shap_values, X_train) ```”

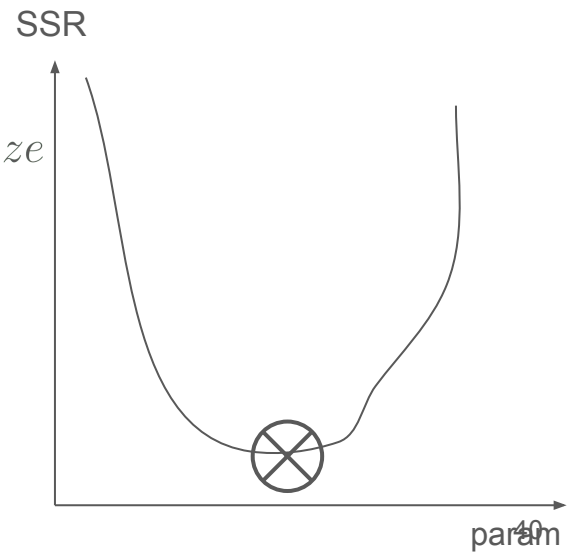
What have been covered so far…

- Two main tricks of Deep Learning:
  1. Layers - linear, non-linear
  2. Gradient descent (intuition)
- Frameworks: PyTorch and fast.ai
- Single approach - multiple applications
- SHAP values for interpretability
- “Vibe coding” with ChatGPT

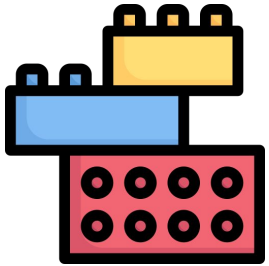
Backpropagation and Gradient Descent

1. Initialize w_i at random values and b_j at zero
2. Sum of Squared Residuals (SSR): $SSR = \sum_i r_i^2 = \sum_i (observed_i + predicted_i)^2$
3. Derivative of SSR: $\frac{dSSR}{dparam} = slope$
4. **Gradient descent**: $new = old - l \times slope = old - step\ size$

   where *l* is the learning rate
5. Repeat until Step Size or slope is close to zero

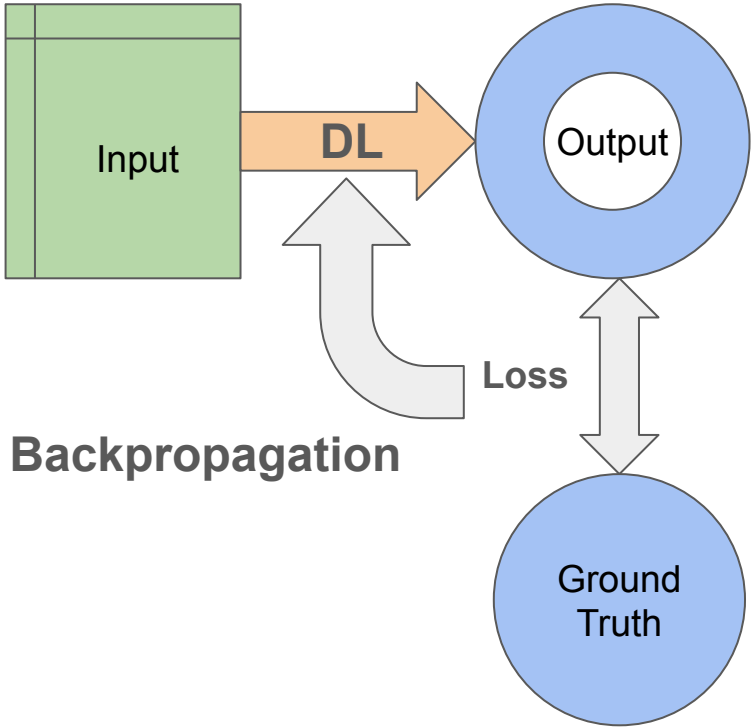


Part II. Neural Networks Dive-In



What is Deep Learning?

- Neural networks (NN) ~ human brain behavior
- Multi-layers: optimize and refine accuracy
- What do we need?
  - Input
  - Ground Truth (GT)
  - Loss ~ |Output − GT | (“accuracy”)
- Building a NN is no easy task: https://playground.tensorflow.org/

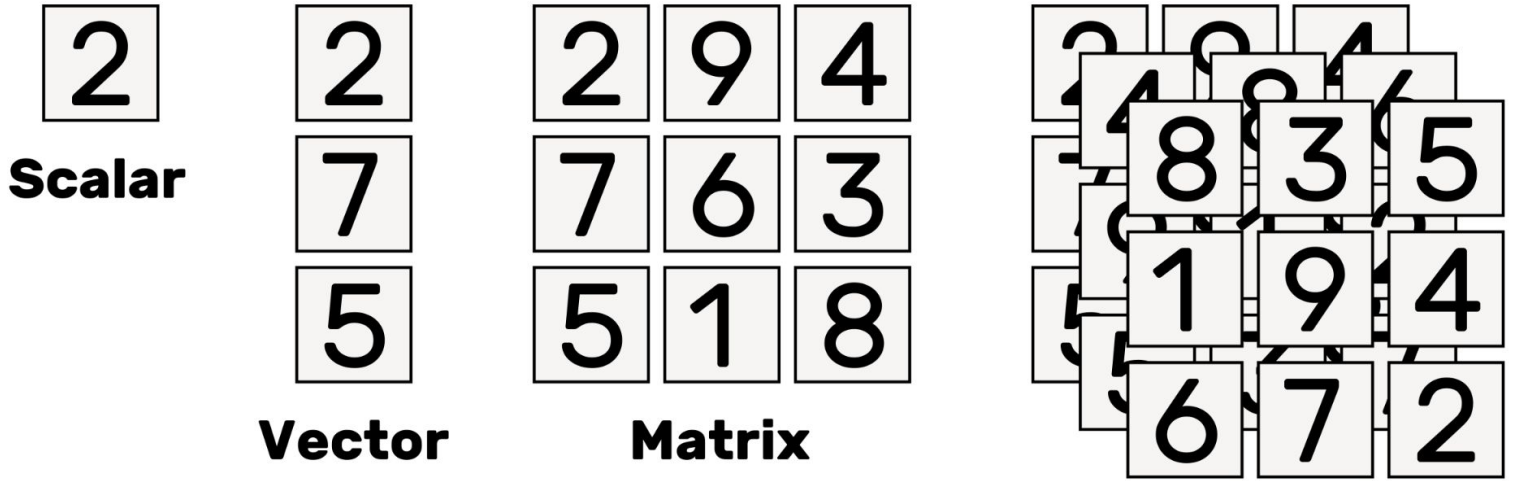


Backpropagation, main ideas:

https://www.youtube.com/watch?v=IN2XmBhILt4&list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1&index=5

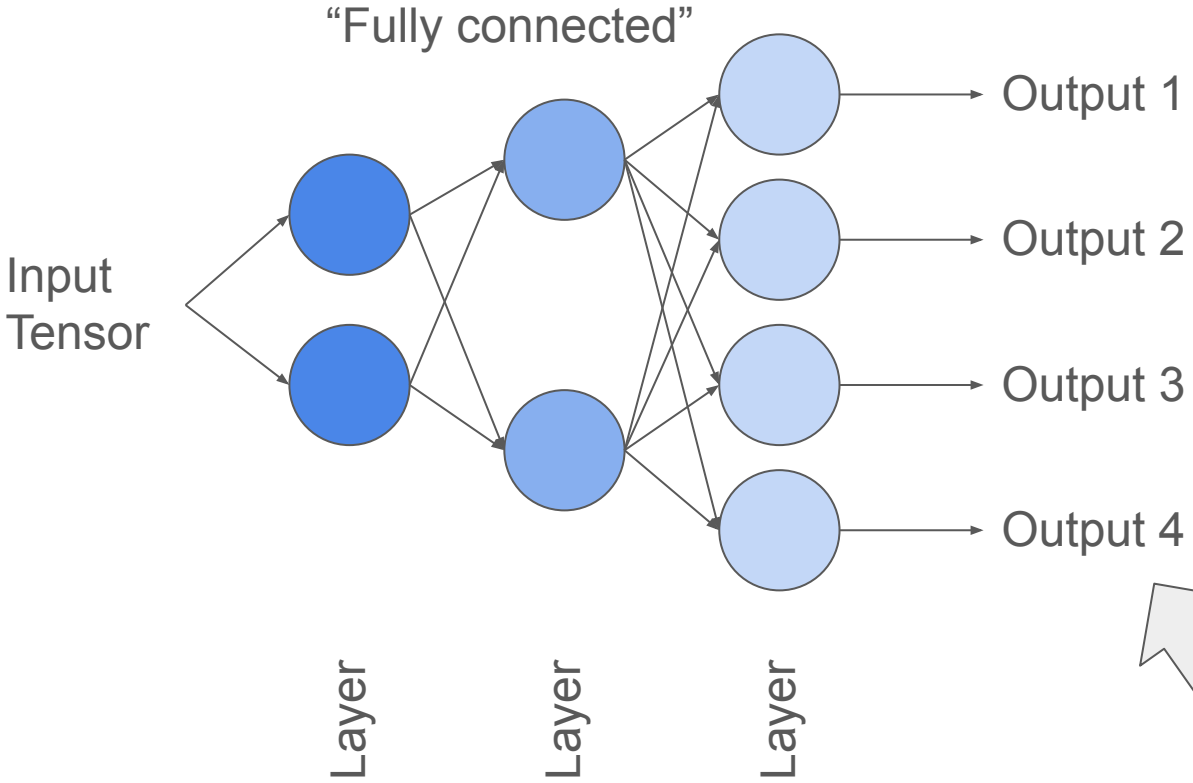
Our data are now **Tensors!**

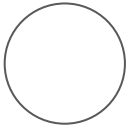
Like numpy arrays,
but optimized for GPUs



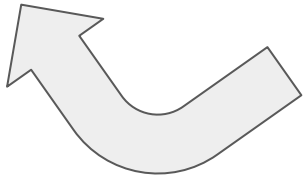
Tensor Rank 0 | Tensor Rank 1 | Tensor Rank 2 | Tensor Rank 3

Neurons and Layers



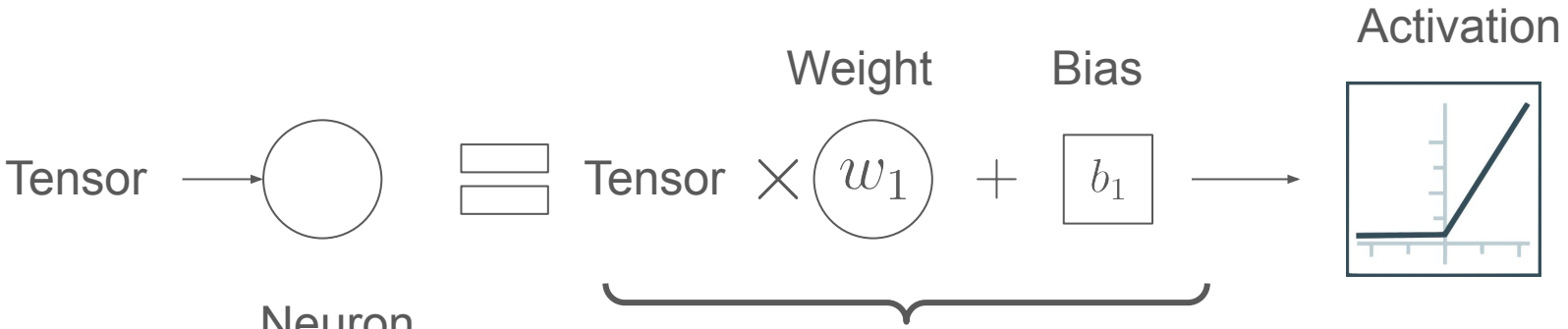


Neuron



Output can be a single number or a vector

A Neuron Performs a Non-Linear Math Operation

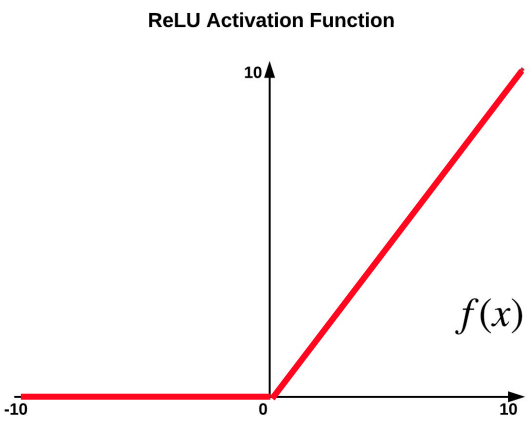


Neural networks learn complex patterns through hierarchical (layers of neurons) non-linear transformations.

Activation Functions

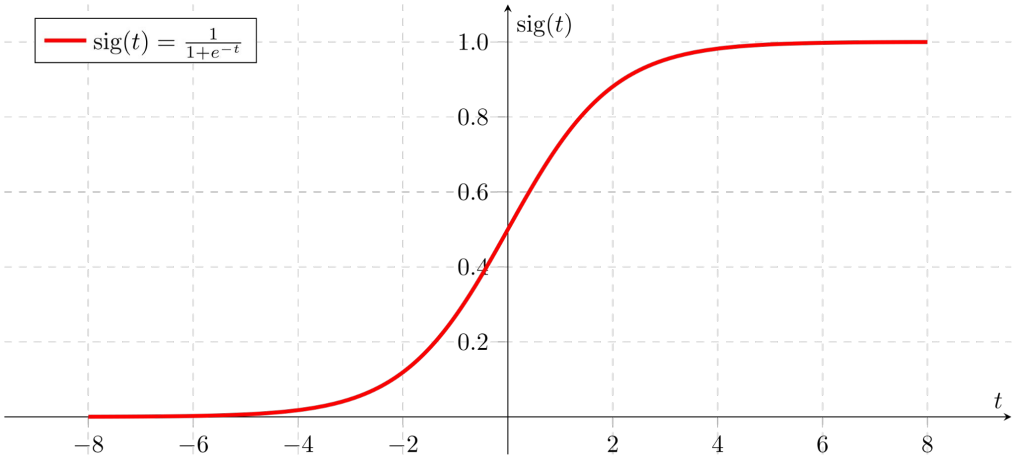
- Activation functions are essential for introducing **non-linearity**, enabling deep networks to learn hierarchical features and model complex patterns
- They tell us “**how activated**” is any given neuron (if a neuron outputs a zero, then it’s inactivated and it is unused for solving a given problem)

RELU (rectified linear unit)

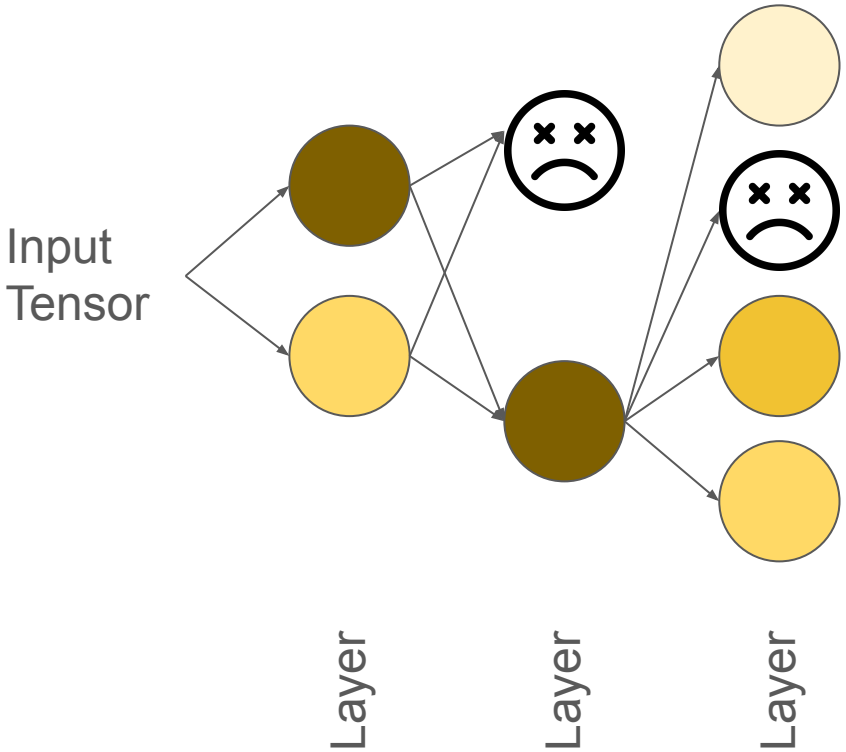


$$f(x) = max(0, x)$$

Sigmoid



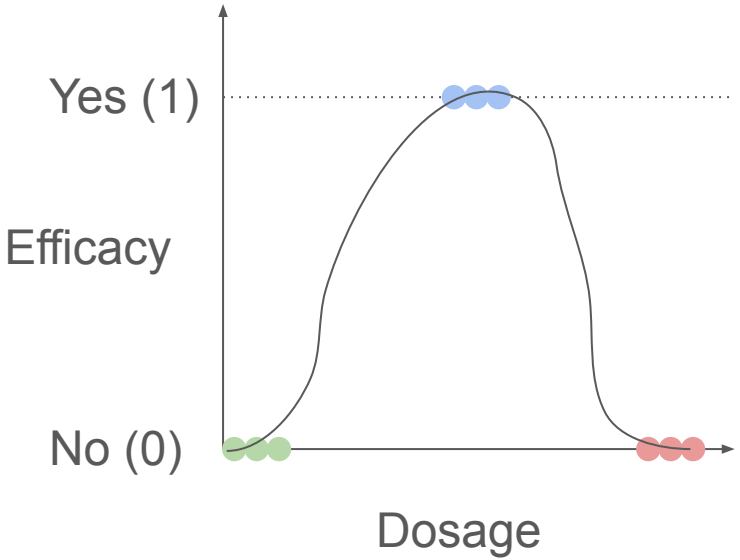
Activations determine how “active” a neuron is



Darker Colors → More active

Lighter Colors → Less active

A Neural Network at work



Dosage

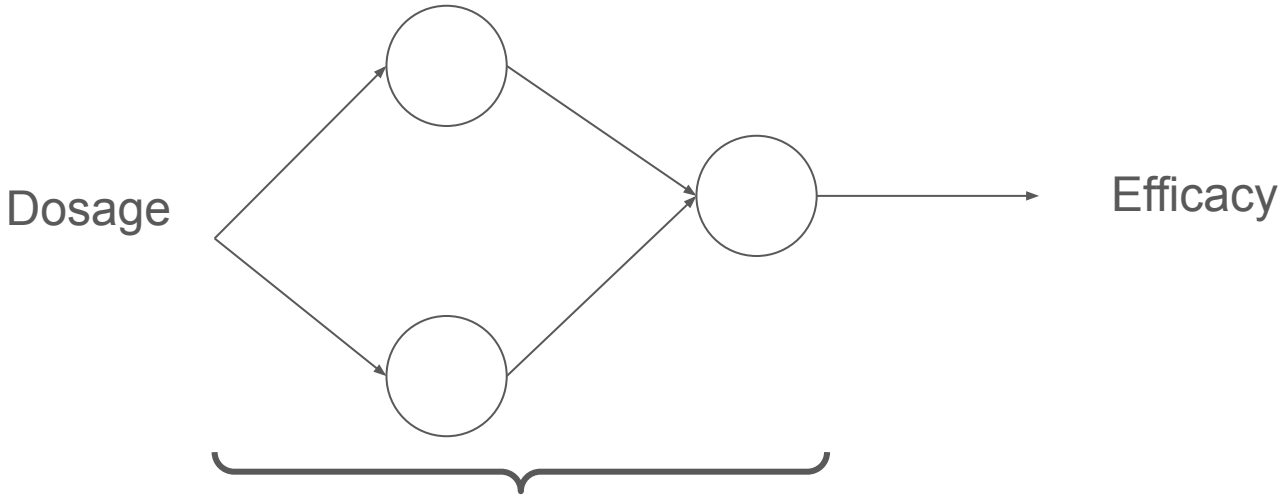
Low Medium High

This is a rather complex function to fit…

→ NNs can fit weird functions!

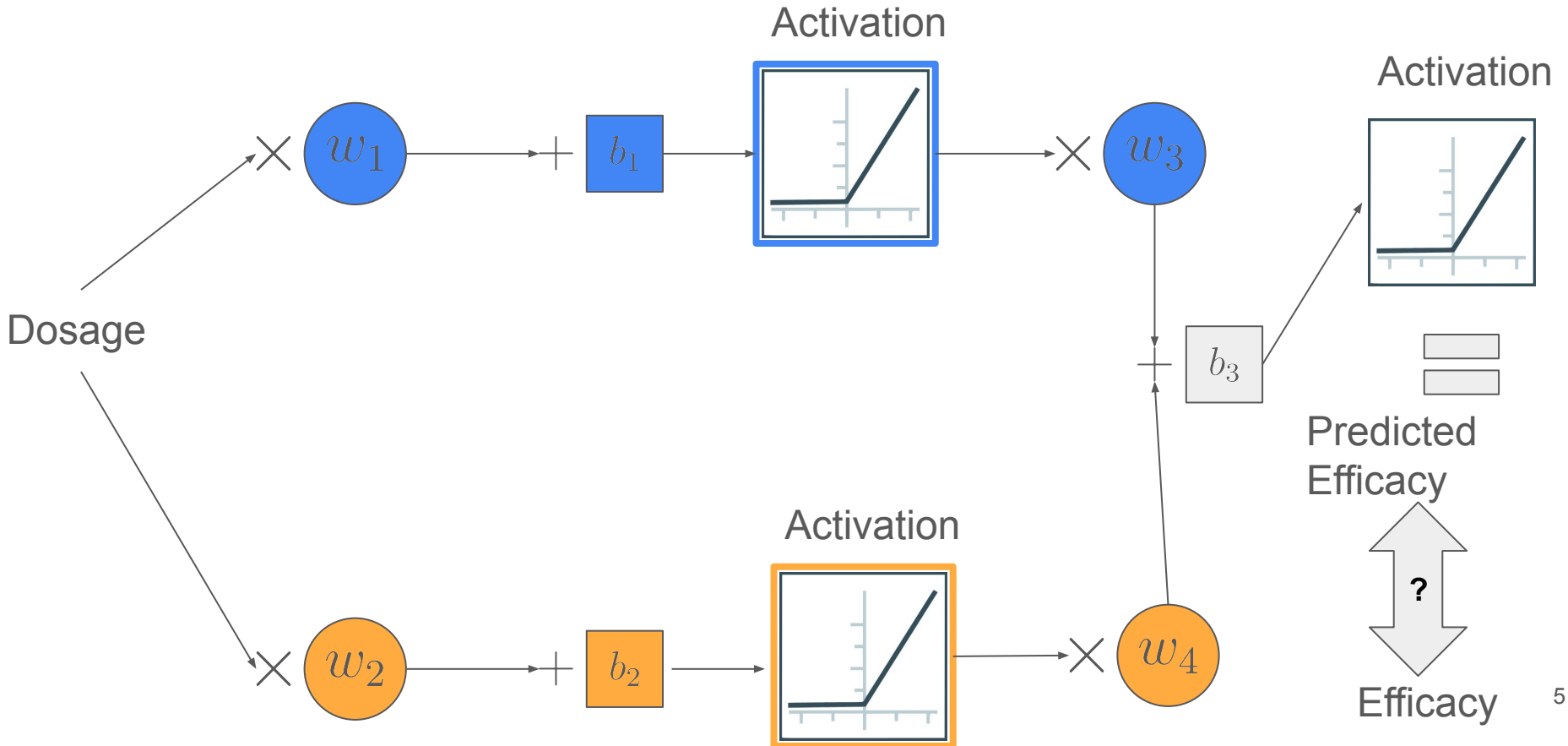
A Neural Network at work

We will use a Neural Network with two Hidden Layers (one with two neurons and the other with one neuron):



A Neural Network at work

Efficacy = **Upper path**
+ Lower path
+ Bias



Cost Functions *C(x)*

Patient	Efficacy	Prediction
Low	0	0.5
Medium	1	0.5
High	0	0.5

The difference between our prediction and the real value is the **loss**

- Mean Squared Error (MSE)
  - "How far off are my predictions?" (for numbers)

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- Binary Cross Entropy (BCE):
  - "How well do my probabilities match reality?" (for yes/no).

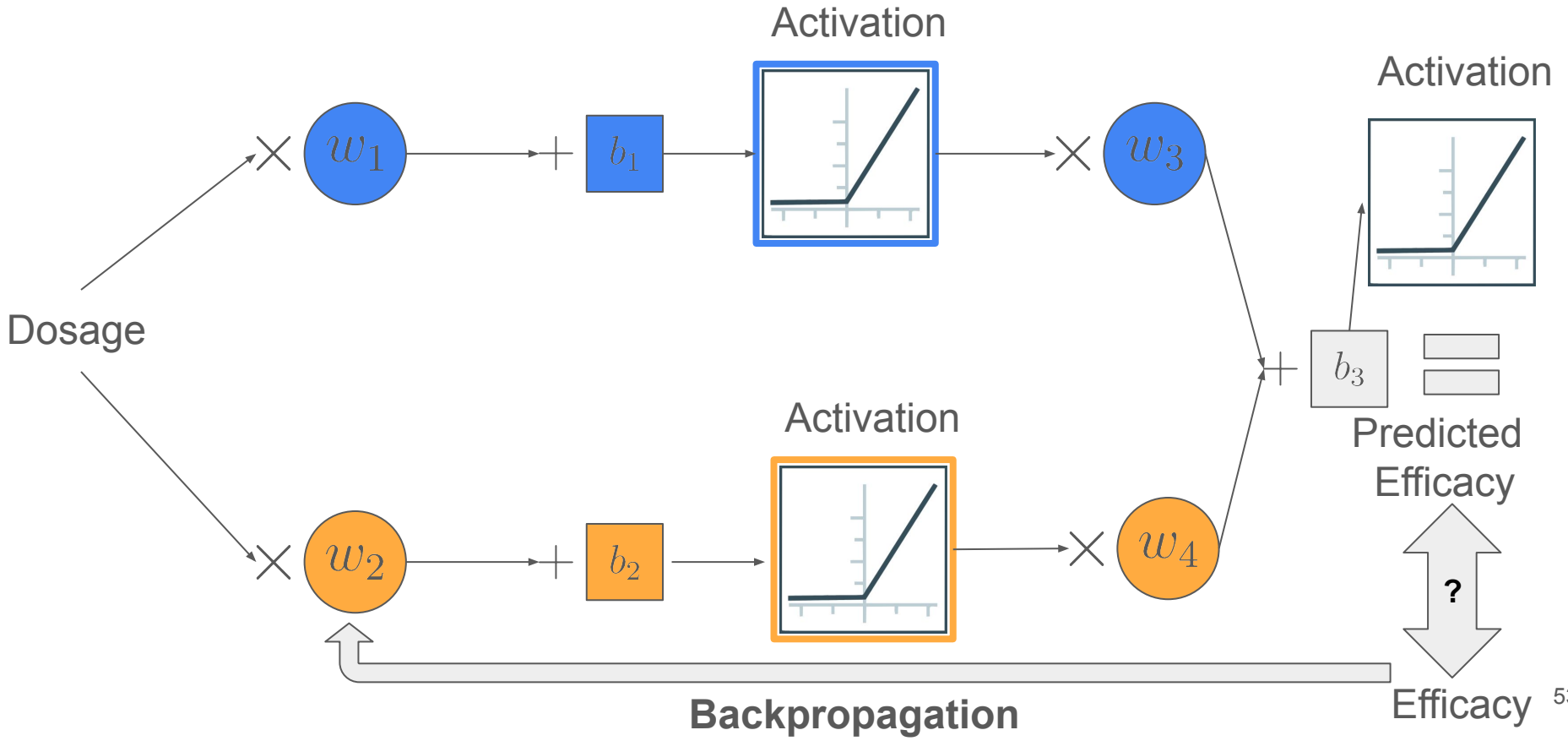
$$\mathrm{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

y_i → True label

$\hat{y}_i$ → Prediction

A Neural Network at work

Efficacy = **Upper path**
\+ **Lower path**
\+ Bias



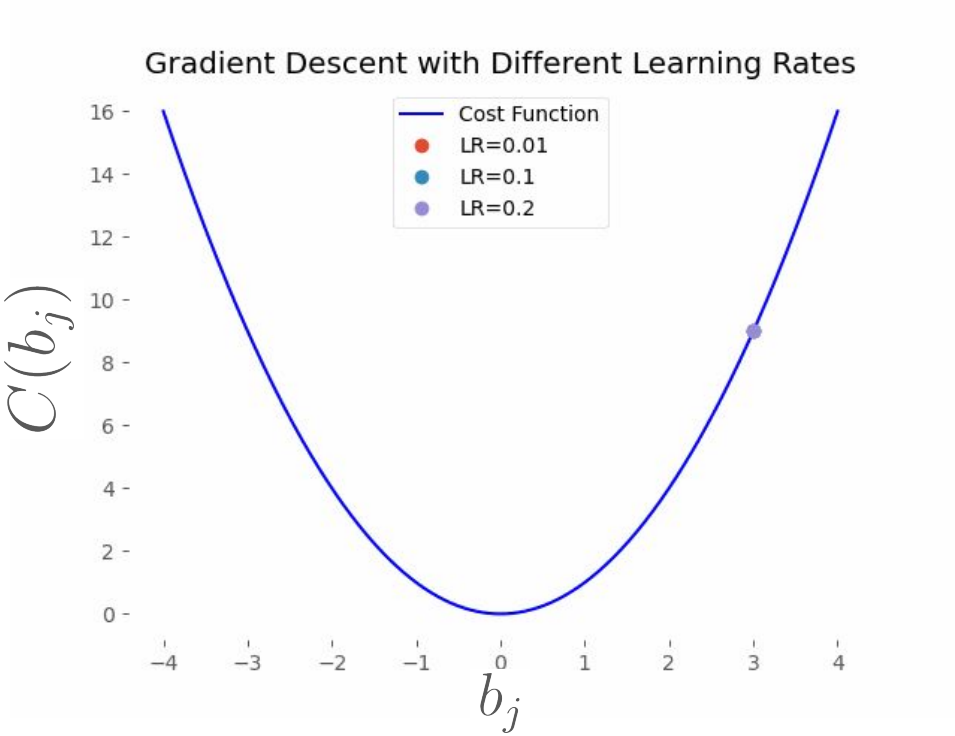
Backpropagation and Gradient Descent

1. Initialize w_i and b_j
2. Minimize cost function $C(x)$
3. **Gradient descent**:

$$new = old - l \times slope$$

where *l* is the learning rate

Repeat a number of times, or "**epochs**"



Stochastic Gradient Descent (SGD)

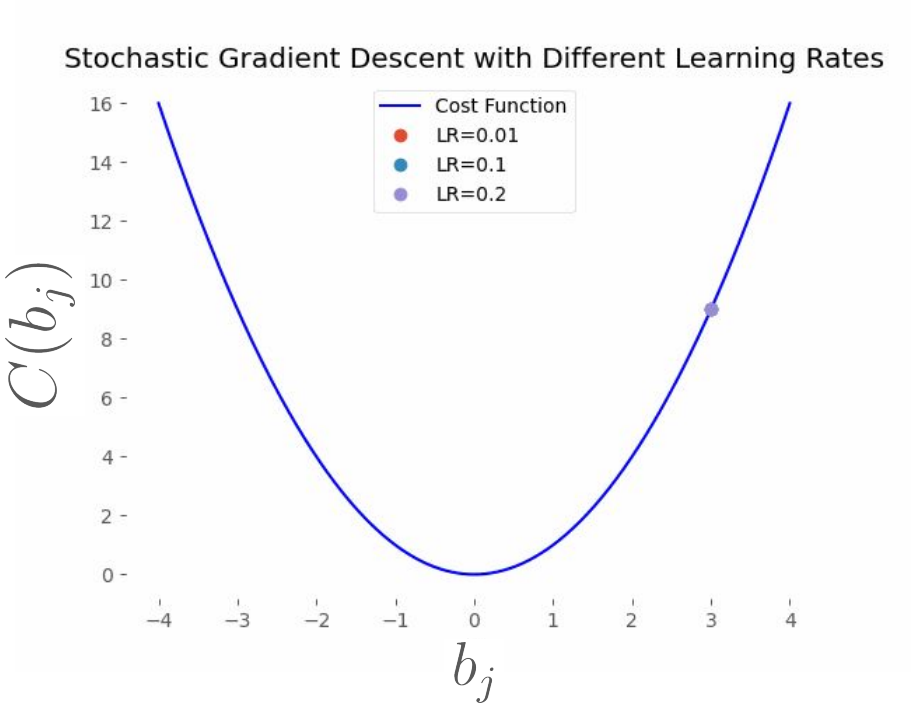
"Drunk man walking downhill"

1. **Randomly** pick **a few** samples
   a. **Mini-batch**: the subset of the data
   b. **Scheduling**: change learning rate from large to small with epochs
2. Pick other samples, at random, N times.

   Repeat for a number of epochs.

   This and other features are already implemented in **optimizers** such as the **Adam optimizer**

   torch.optim.Adam



Parameters vs Hyperparameters

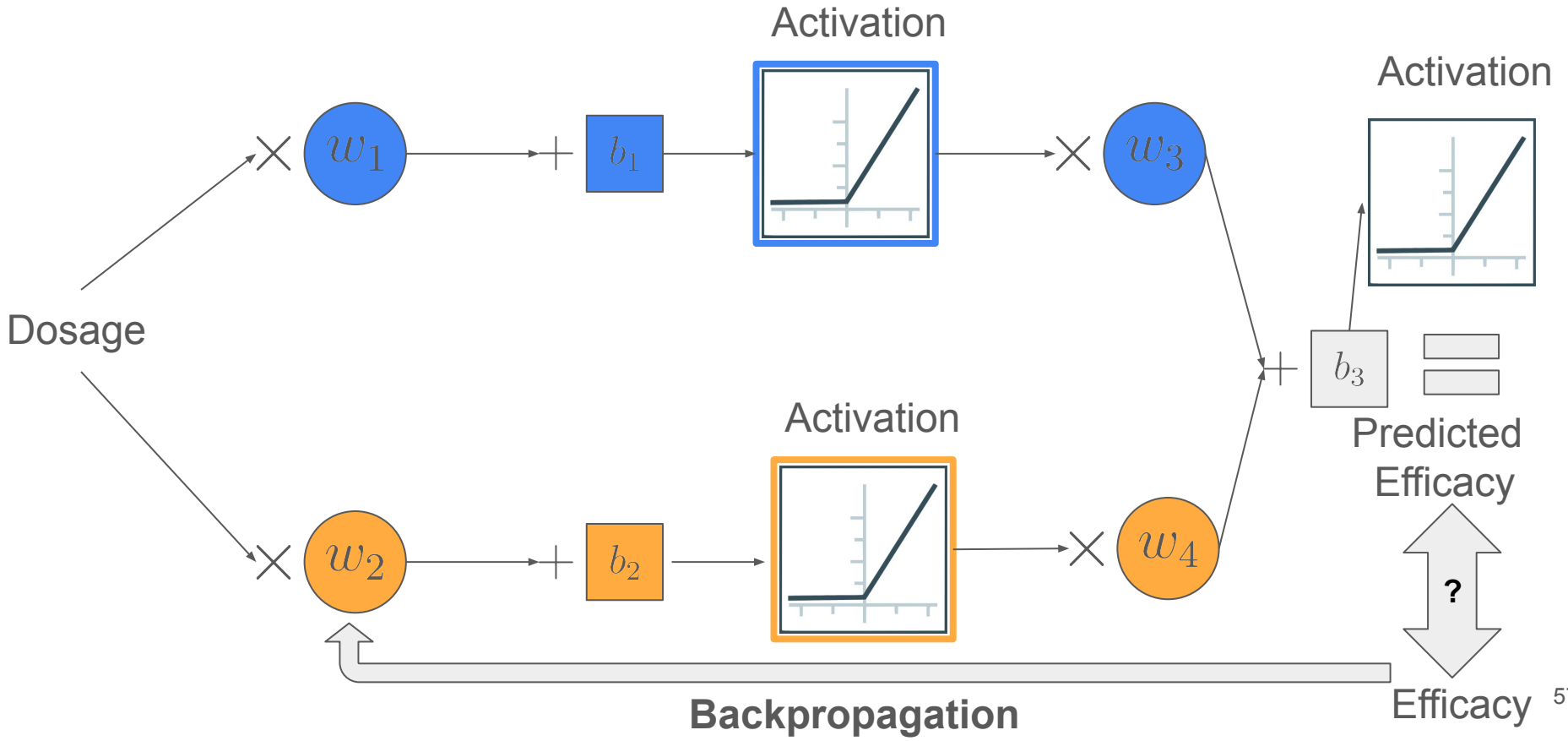
- **Parameters**: a variable **inherent** to a model (weights and biases)

- **Hyperparameter**: a variable used **outside** of a model (learning rate)

Hyperparameters are set before training to estimate a model's parameters.

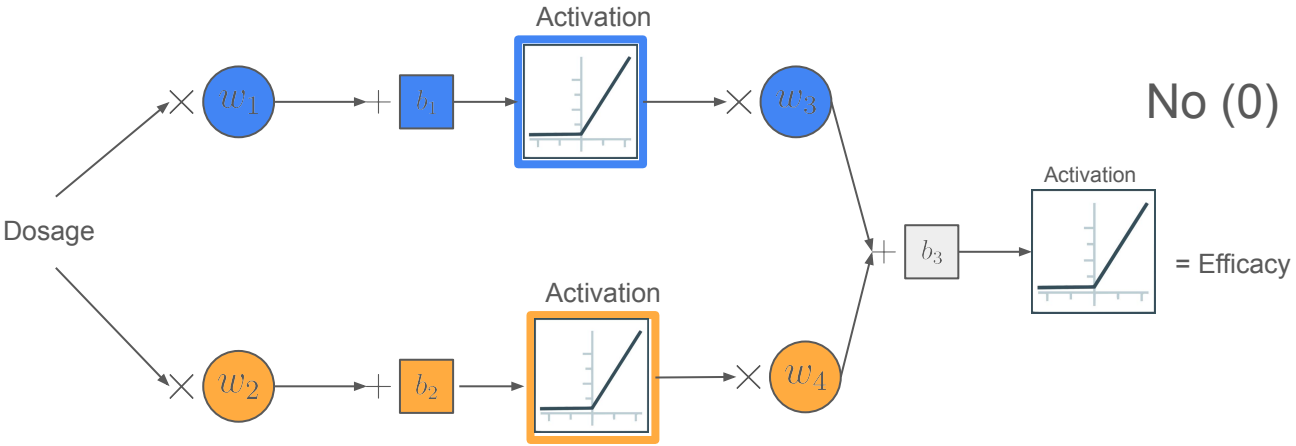
A Neural Network at work

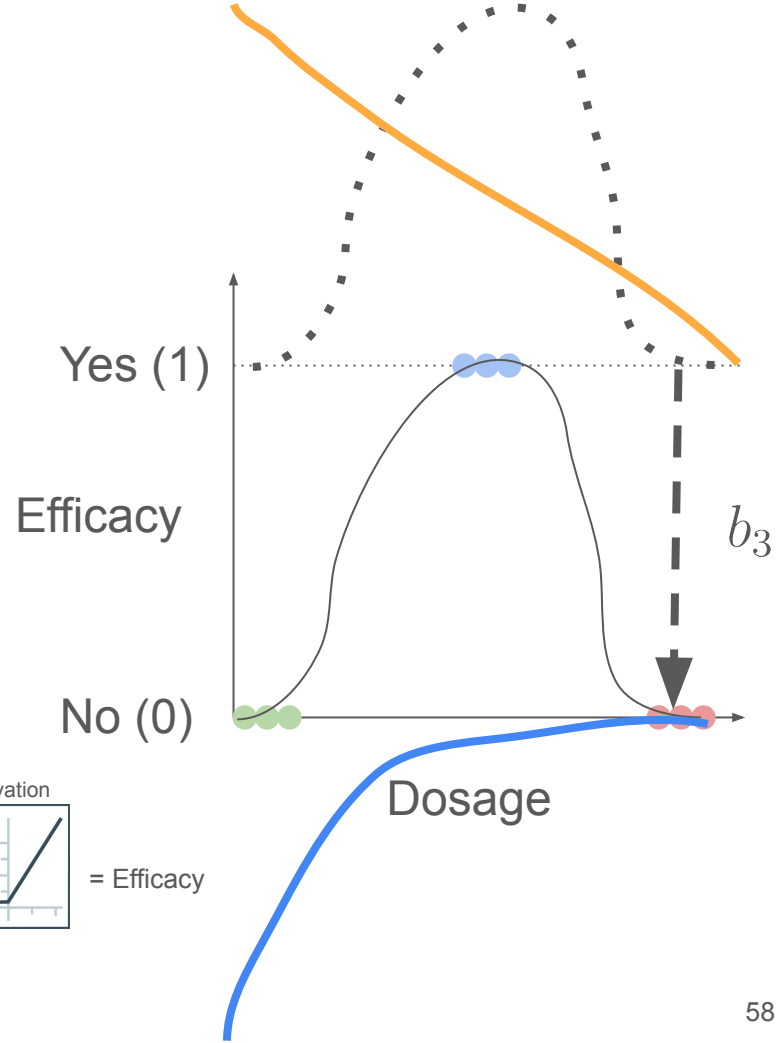
Efficacy = **Upper path**
\+ **Lower path**
\+ Bias



A Neural Network at work

Efficacy = **Upper path**
+ **Lower path**
+ Bias





Data Scaling

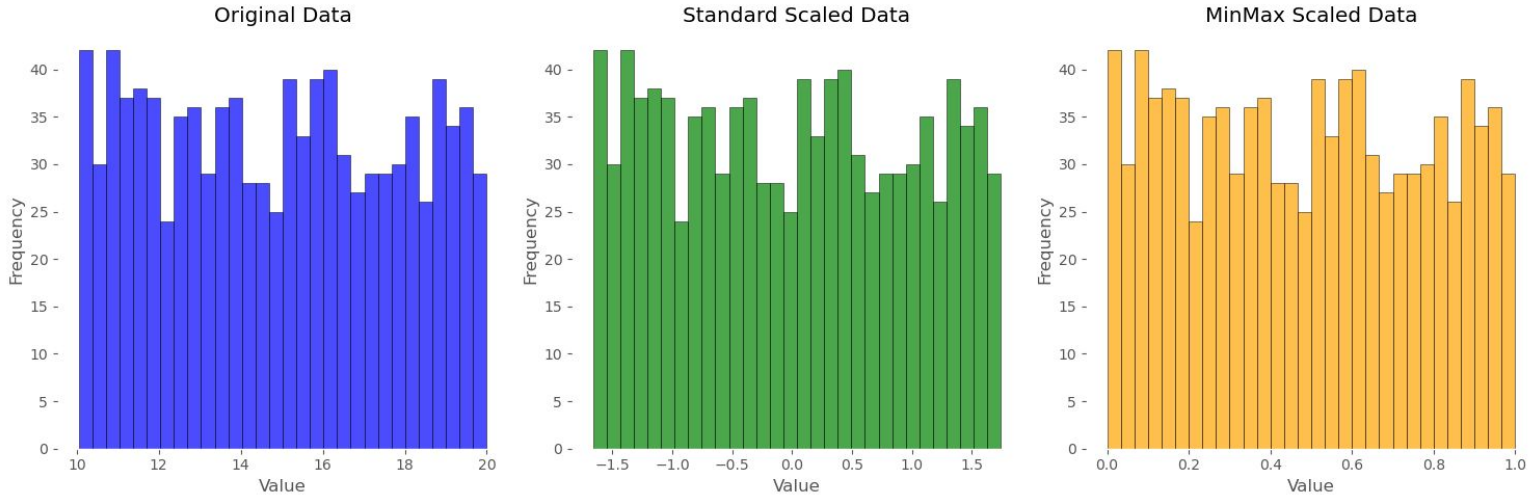
- Features with vastly different scales (e.g., age [0–100] vs. income [0–1,000,000]) cause uneven updates during gradient descent.
- Activation functions like *sigmoid* saturate (stop learning) when inputs are too large or small.

Standard Scaler

- Centers data to have mean = 0 and scales it to have standard deviation = 1.
- For data without strict limits (e.g., temperature, income).

MinMax Scaler

- Scales data to a fixed range (usually [0, 1])
- Data with known limits (e.g., pixel values [0–255], sensor readings [0–100])



Key Concepts

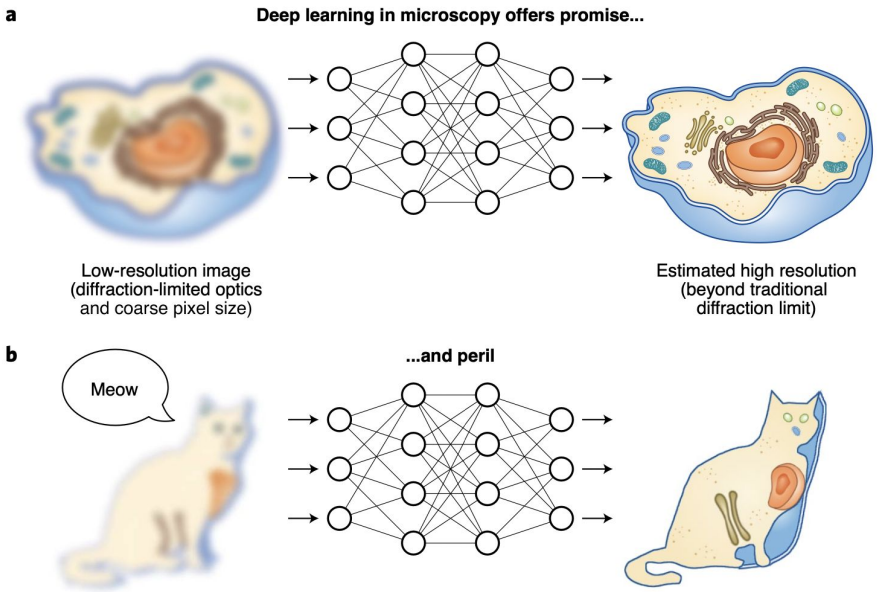
- **Tensor**: data-storage optimized for deep learning in GPUs
- **Loss**: difference between reference and prediction
- **Optimizer**: algorithms that serve as an extension to stochastic gradient descent; they update network weights iterative based in training data
- **Epoch**: a full pass through the training set
- **Batch size**: small subset of data used for training in stochastic gradient descent
- **Backpropagation**: algorithm used to calculate the optimal parameters implied in a machine learning problem
- **Gradient Descent**: algorithm used to update the estimated parameters during backpropagation
- **Learning rate**: rate at which parameters change in gradient descent
- **Layers**: chain of neurons in a neural network
  - **Hidden Layers**: neurons that perform a regression
  - **Activation Layers**: layers that adjust the result of a regression
  - **Activation Functions**: SoftPlus, ReLU, Sigmoid…
- **Weights and biases**: parameters used for multiplication and addition, respectively at each neuron in a neural network

Limitations

- Computationally demanding (GPUs)
- *Used* as black-boxes ("alchemy")
- Memorizes patterns!

  "Its abundance of power is matched by its lack of wisdom" (McElreath et al.,Taylor and Francis, 2020).

- Ground Truths are fundamental



Hoffman et al., Nature Methods, 2021

Limitations

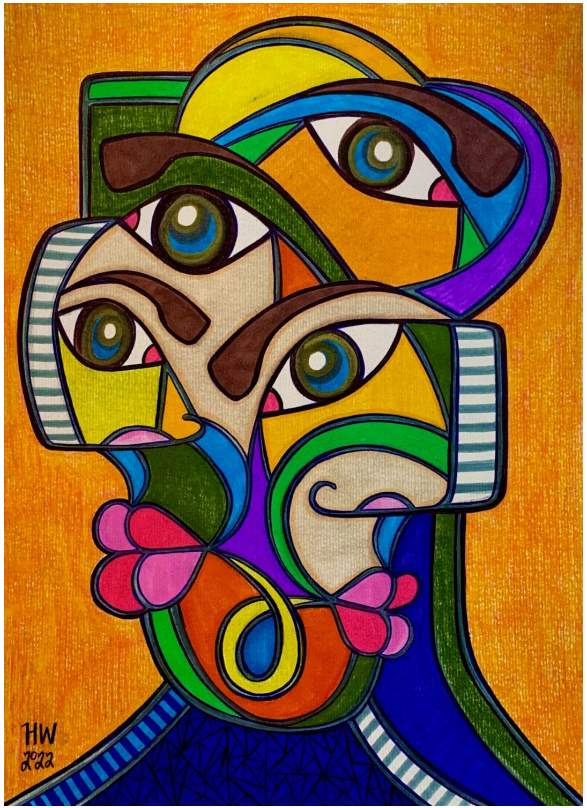
- Computationally demanding (GPUs)
- *Used* as black-boxes ("alchemy")
- Memorizes patterns!

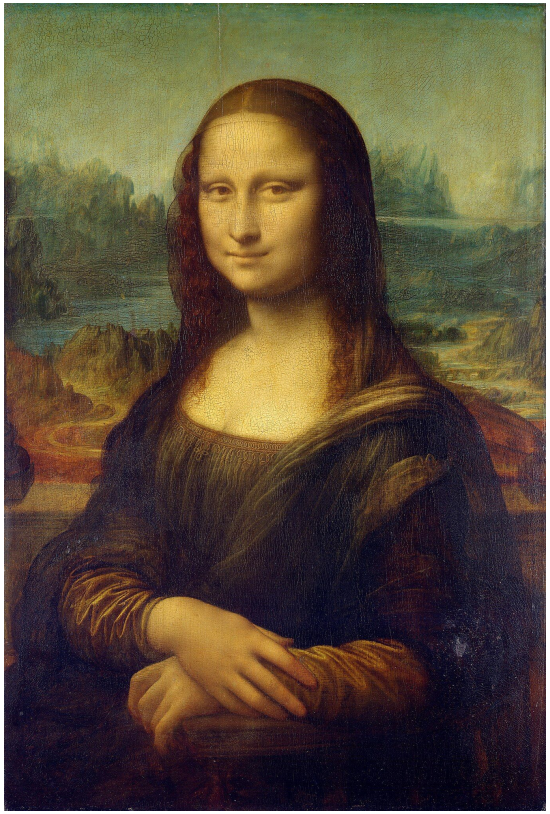
  "Its abundance of power is matched by its lack of wisdom" (McElreath et al.,Taylor and Francis, 2020).

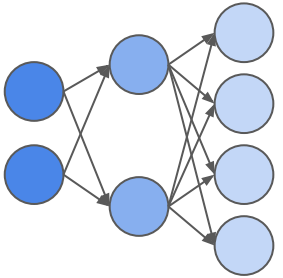
- Ground Truths are fundamental


PyTorch

Neural Networks limitations







No **spatial** context!

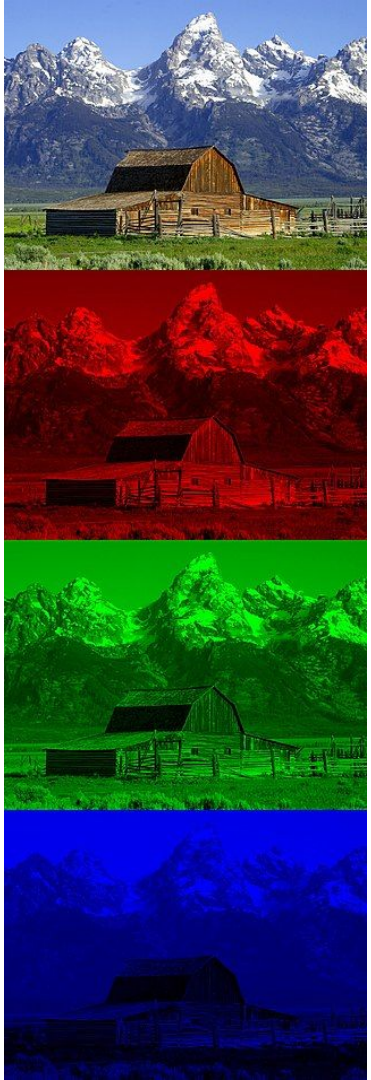
https://www.reddit.com/r/Art/comments/va4wfg/abstract_portrait_me_ink_and_colored_pencil_2022/

What is an Image?

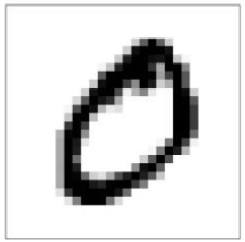
- Images are **grids** of pixels (e.g., 12 Mega Pixels= 12M inputs!).
  - If it is a color image, then it is a Rank 3 Tensor (RGB)—each channel is its own matrix This means 3x the number of inputs!

- Raw pixels lack **spatial context** (e.g., a "barn" is a pattern, not just individual pixels).

- Too many parameters: A regular NN would be slow and inefficient.

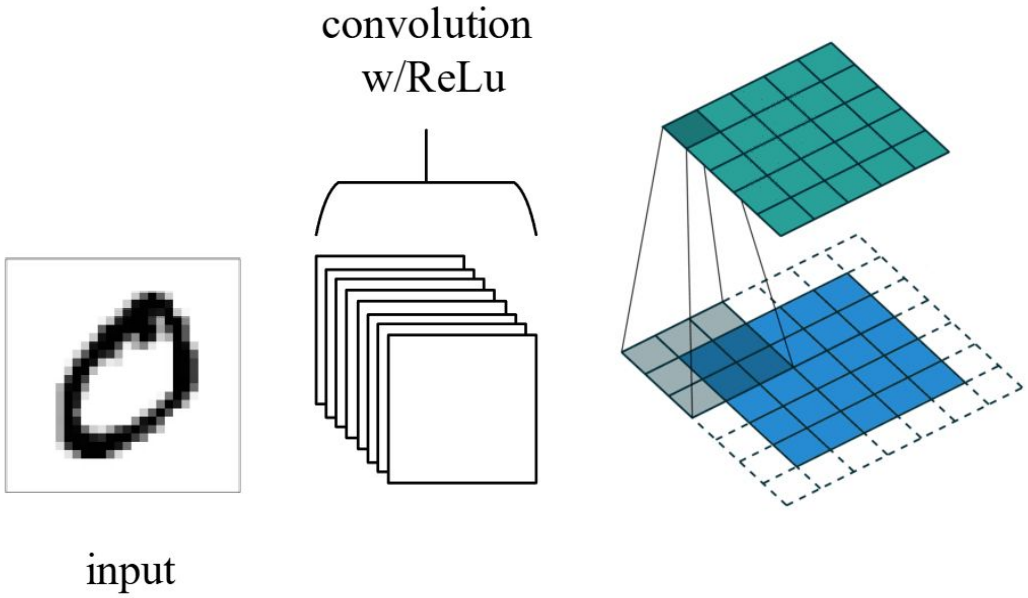


A Convolutional Neural Network At Work



input

A Convolutional Neural Network At Work



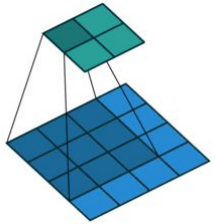
Convolution Layer

Slide small filters called **kernels** (e.g., 3x3 grids) across the image to **detect** edges, textures, or shapes.

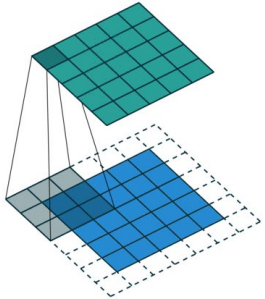
Example: A filter might activate for "vertical edges" or "curves"

https://github.com/vdumoulin/conv_arithmetic

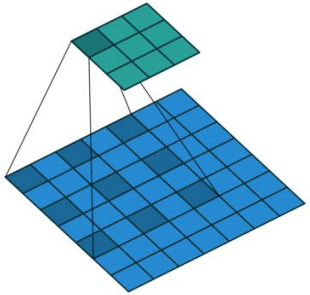
Convolution



No padding,
no strides



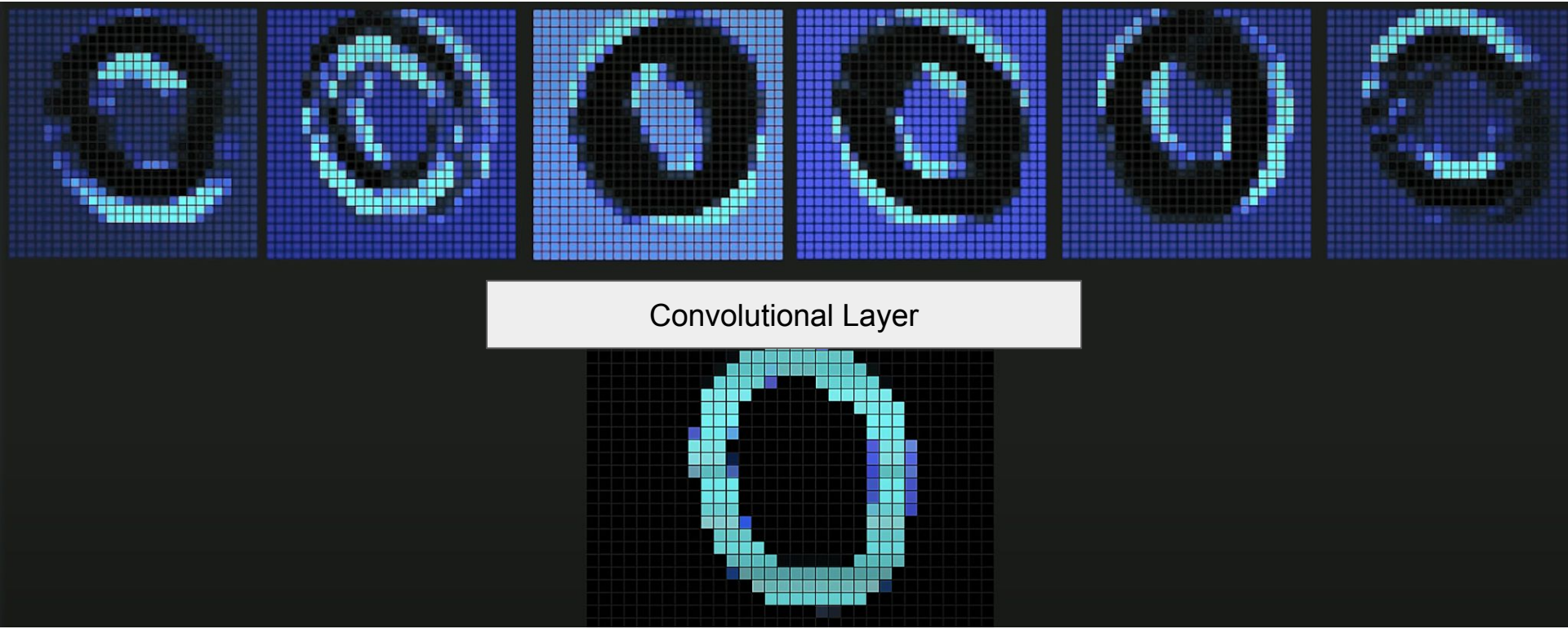
Same padding,
no strides



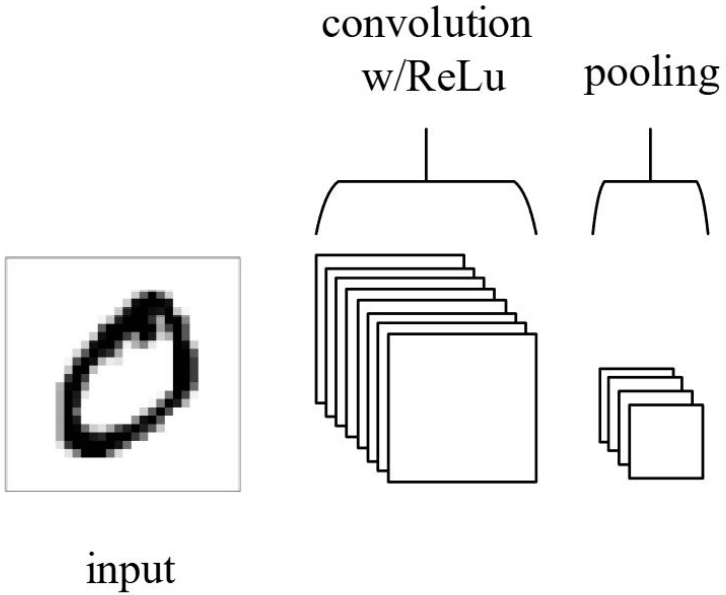
No padding, no
stride, dilation

Note: Blue maps are inputs, and cyan maps are outputs.

https://github.com/vdumoulin/conv_arithmetic


Convolutional Layer

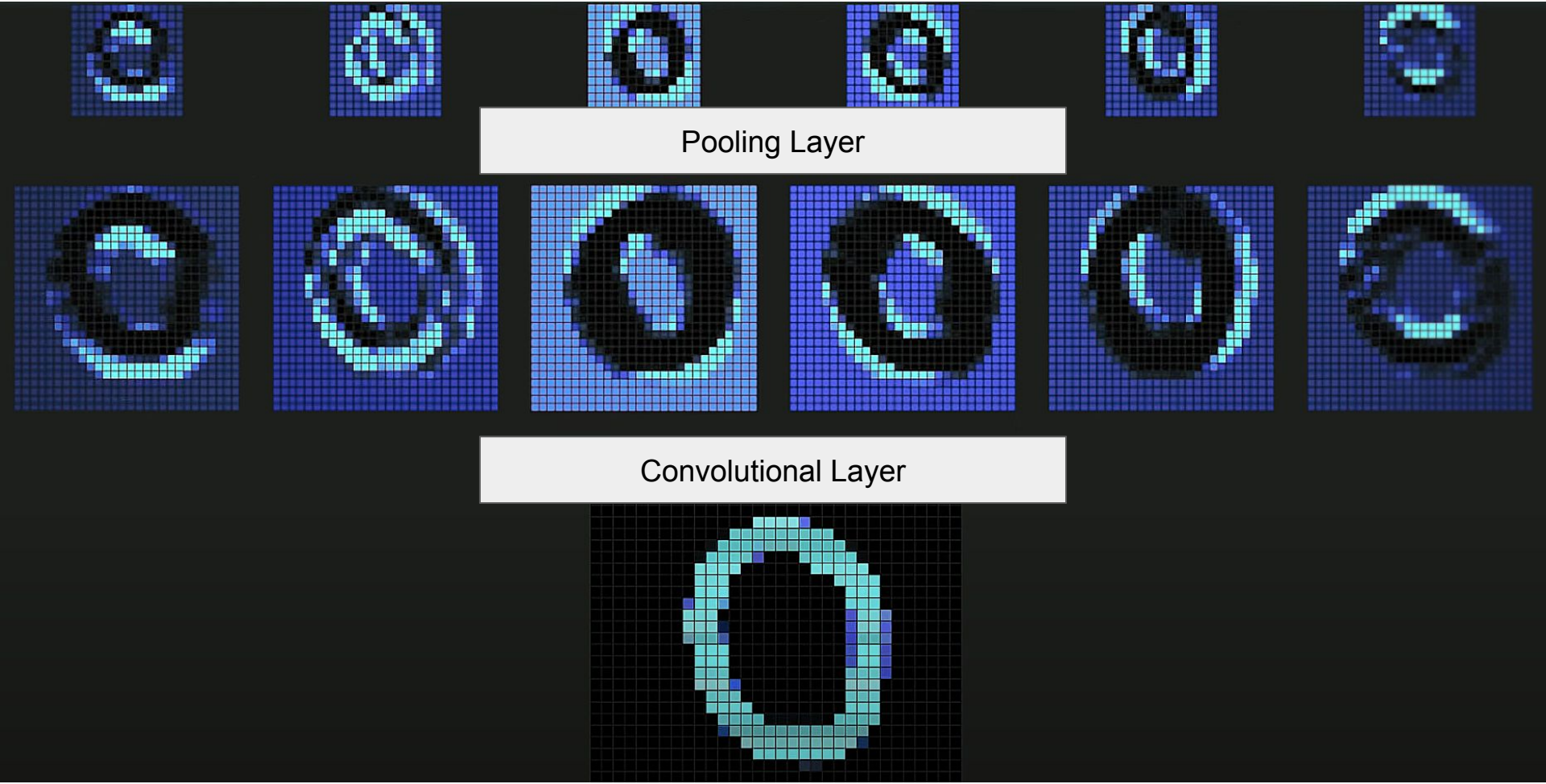
A Convolutional Neural Network At Work



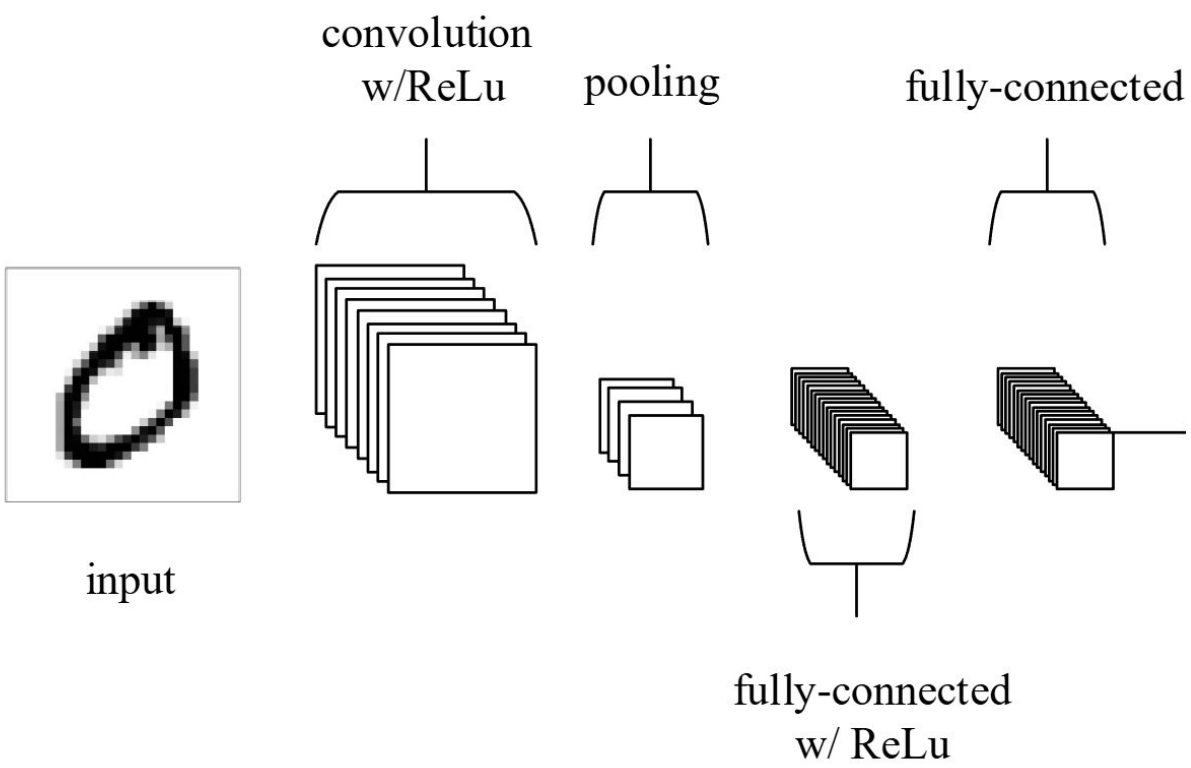
Pooling Layer

Downsample the image (e.g., keep the max value in a 2x2 grid → "Did we see this feature in this region?").

Reduces size and computation.


Pooling Layer
Convolutional Layer

A Convolutional Neural Network At Work

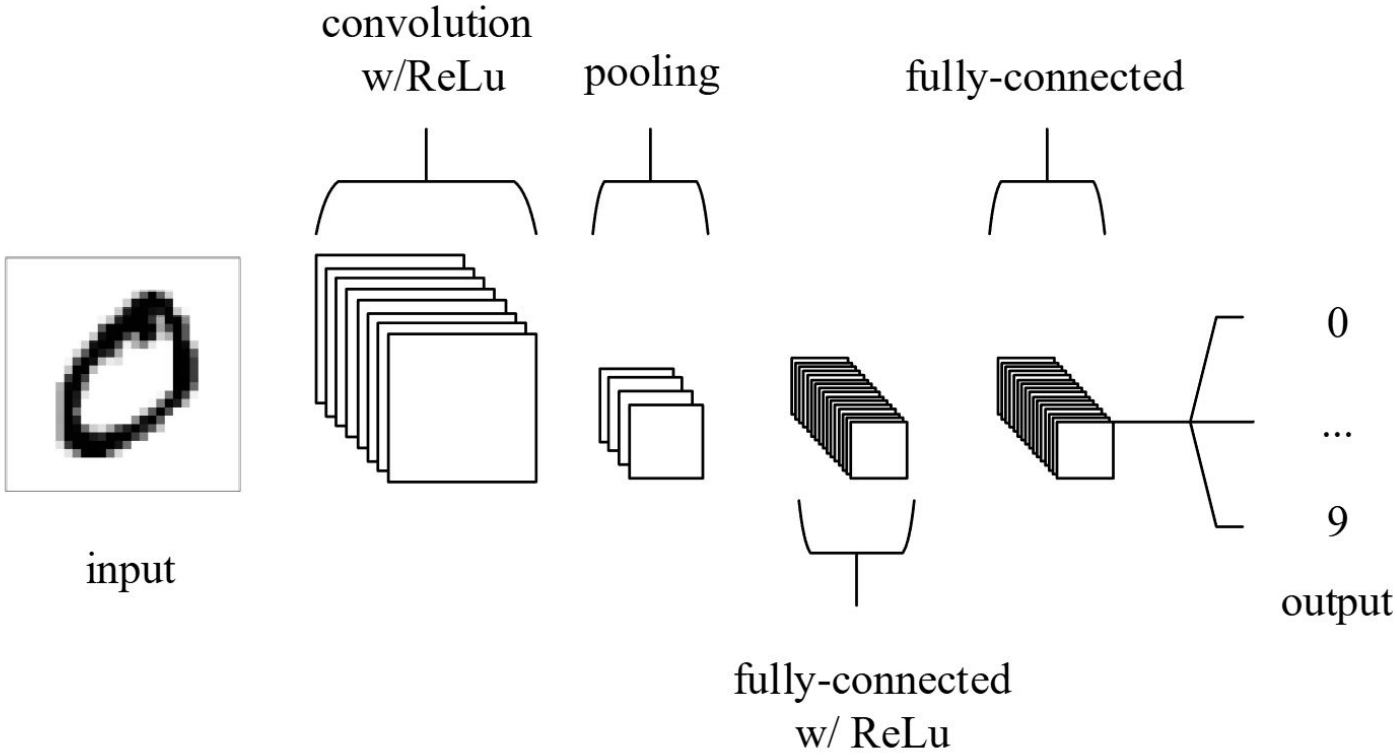


Fully Connected Layers

Combine detected features to classify (e.g., "0" vs. "1").

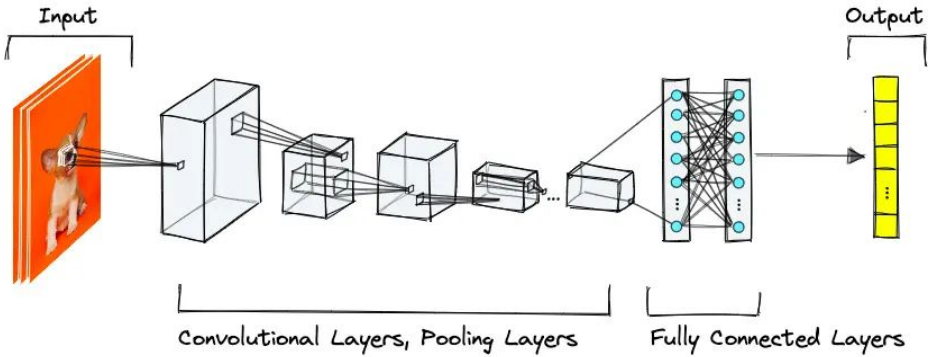
Visualization:
https://adamharley.com/nn_vis/cnn/2d.html

A Convolutional Neural Network At Work



→ After pooling, we can build a NN for **classification**

Typical CNN



Input: 5x5px

1	0	0	0	0
1	0	1	1	1
0	0	1	1	1
0	1	0		0
0	1	1	1	1

Filter: 3x3px

1	0	1
1	1	0
0	0	1

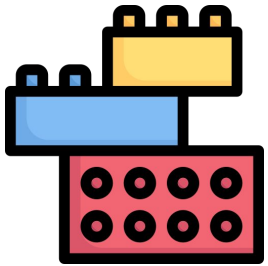
Element-wise multiplication

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 3$$

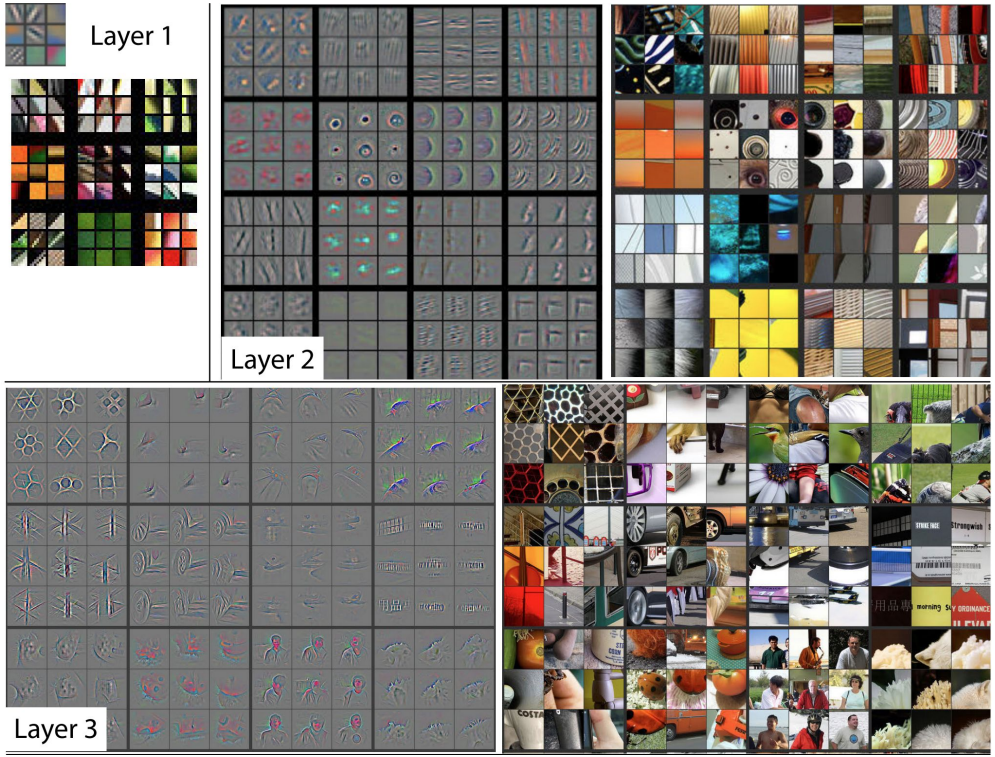
https://www.pinecone.io/learn/series/image-search/cnn/

Why CNNs?

- Biologically inspired
- Sharing of parameters
- Different layers that perform different tasks. You can combine different networks, the same way you combine the pieces of a puzzle!



Convolutional Neural Networks (CNN) (2/3)

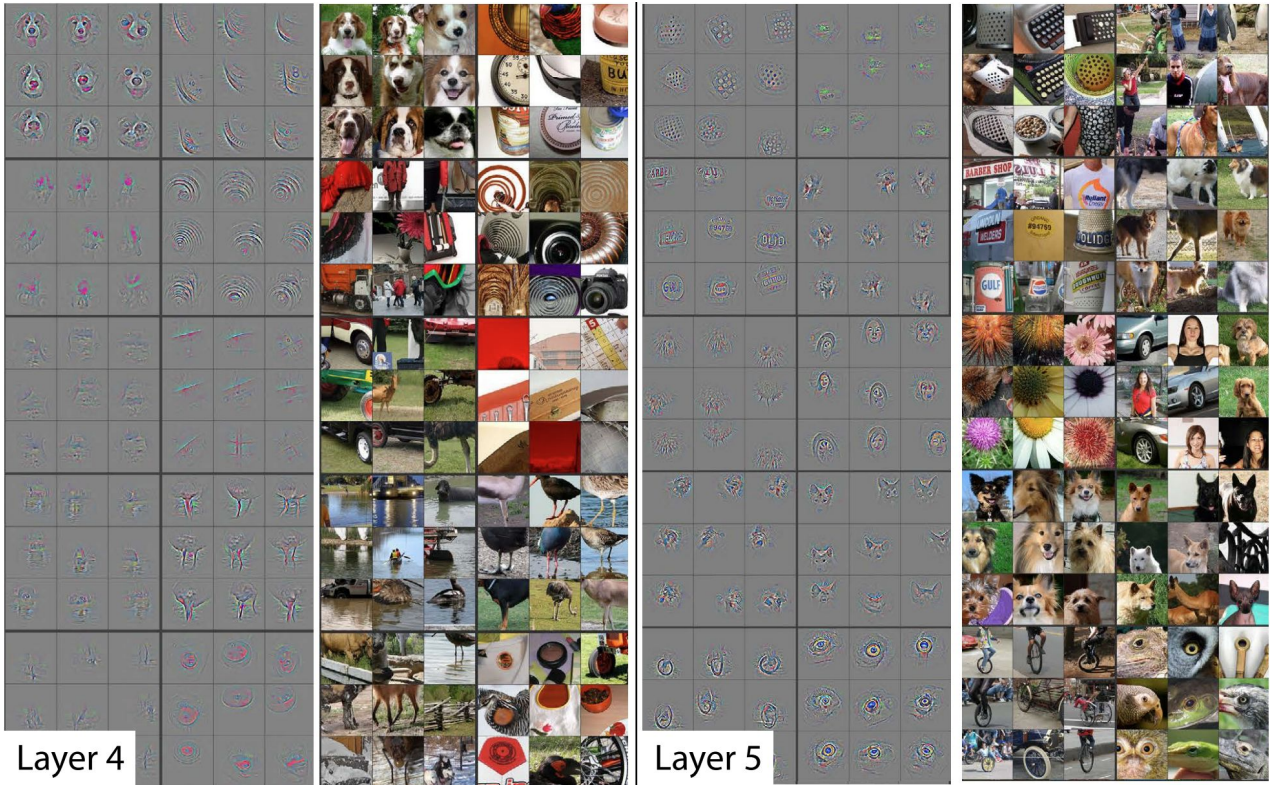


Deep Learning is not a “black box”

For each layer, the image part with the light gray background shows the reconstructed weights pictures, and the larger section at the bottom shows the parts of the training images that most strongly matched each set of weights. For layer 1, what we can see is that the model has discovered weights that represent diagonal, horizontal, and vertical edges, as well as various different gradients.

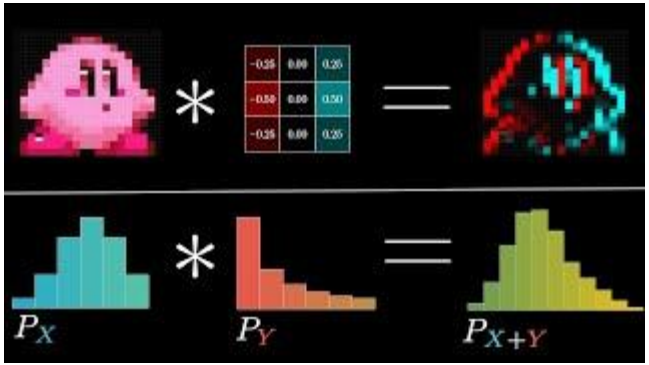
“Visualizing and Understanding Convolutional Networks”, Matthew D. Zeiler, Rob Fergus https://arxiv.org/pdf/1311.2901

Convolutional Neural Networks (CNN) (3/3)



"Visualizing and Understanding Convolutional Networks", Matthew D. Zeiler, Rob Fergus https://arxiv.org/pdf/1311.2901

In case your thirst for knowledge has not yet been quenched …



CNN in PyTorch

Jupyter notebook from:
https://docs.pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Why CNNs Are Powerful for Images

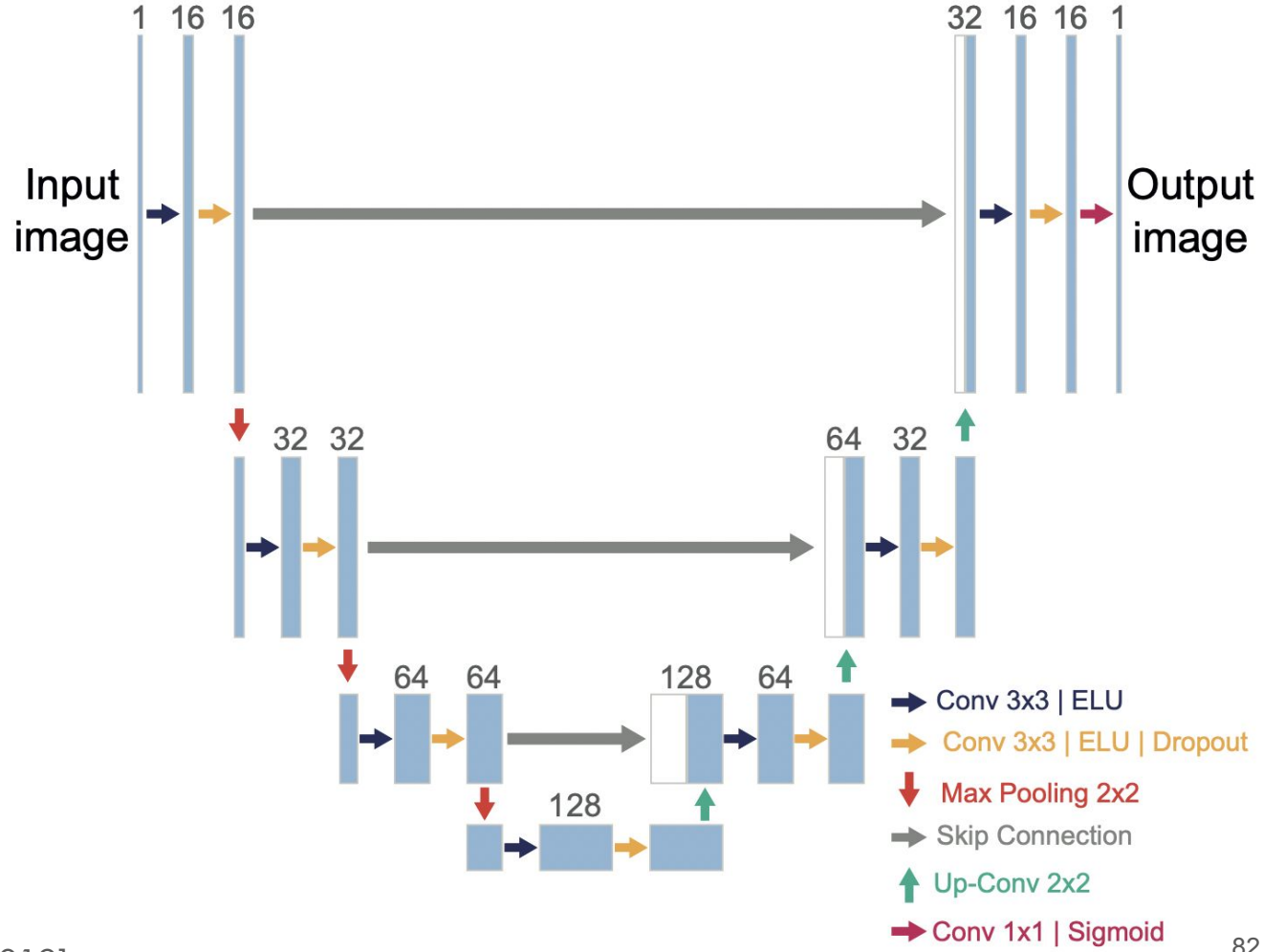
- Translation Invariance:
  - Detects features anywhere in the image.
- Parameter Sharing:
  - The same filter scans the entire image (efficient!).
- Hierarchy:
  - Layers detect increasingly complex patterns:

Edges → Textures → Shapes → Objects


PyTorch

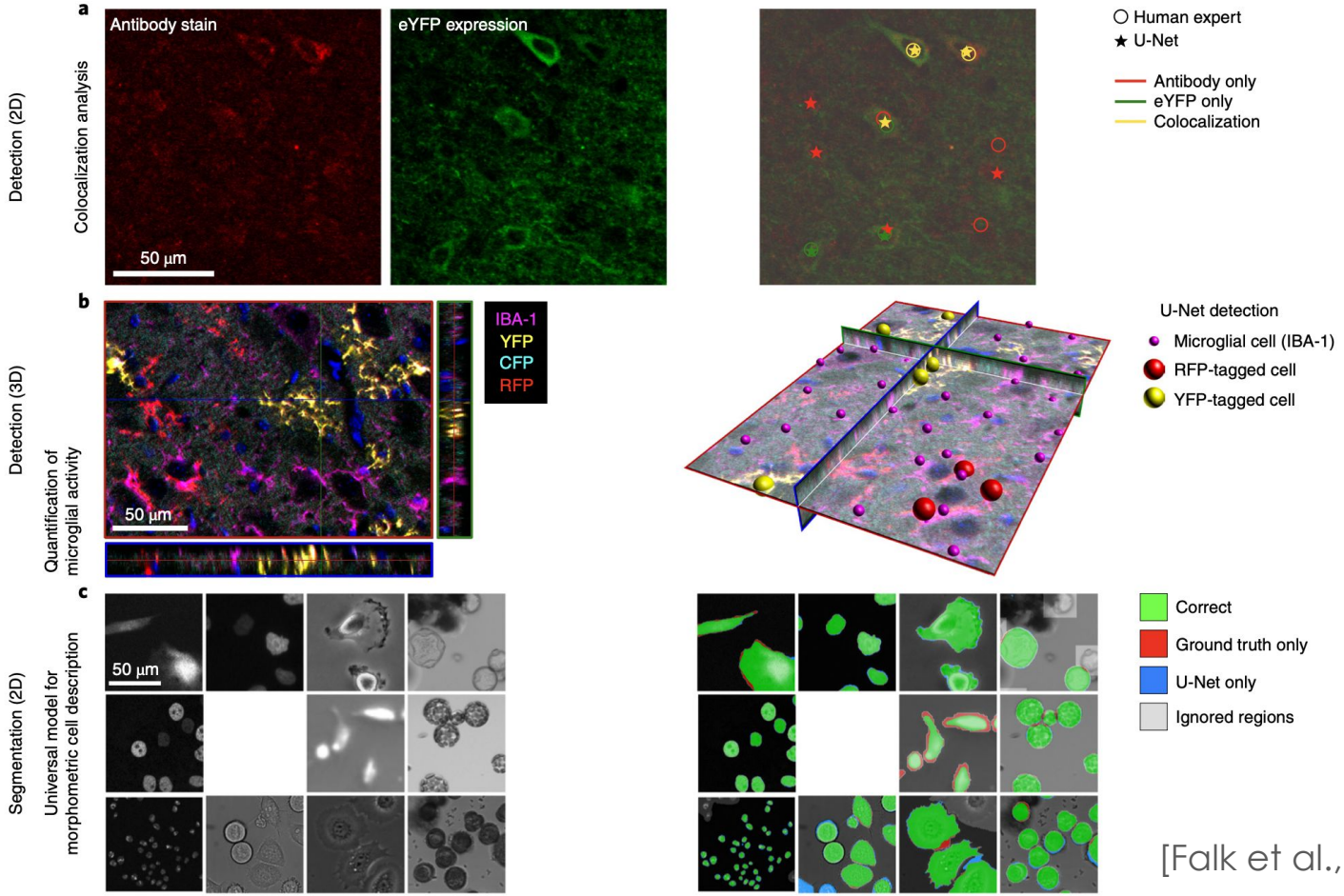
Advanced Concepts

U-Net



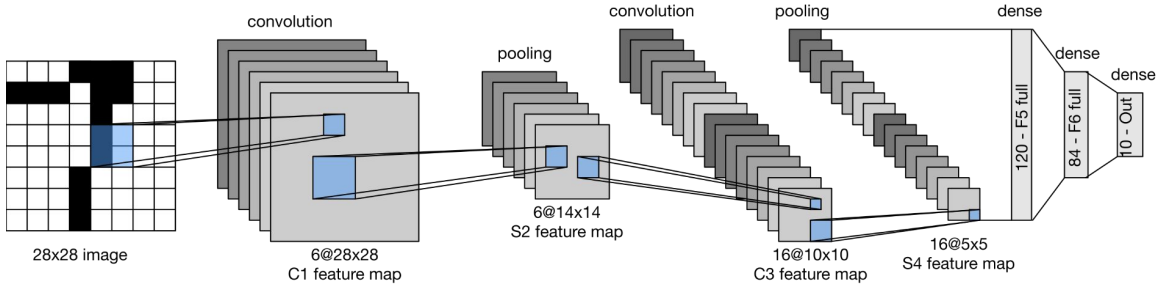
[Falk et al., Nature Methods, 2019]

U-Net applications



[Falk et al., Nature Methods, 2019]

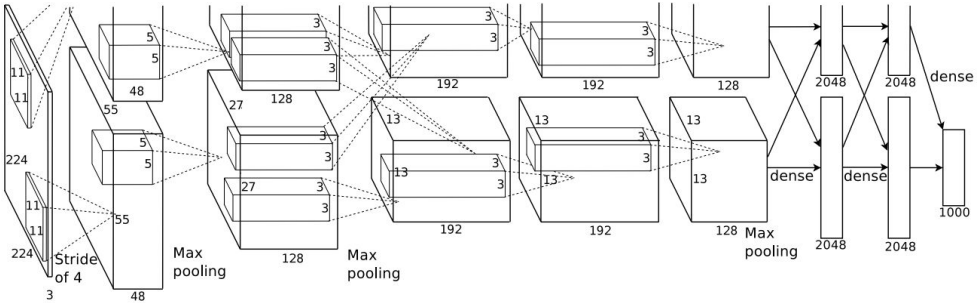
LeNet

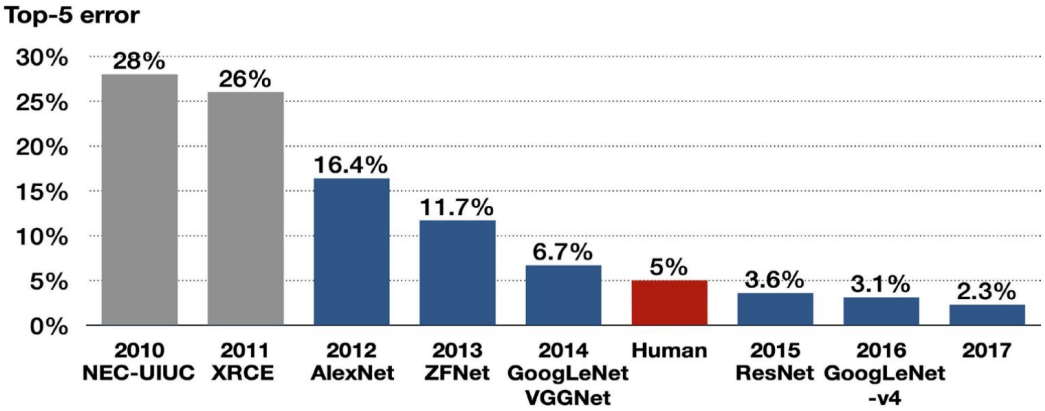


- First CNN
- Proposed in 1989(!)
- Dataset is 28x28 (MNIST)

http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

AlexNet

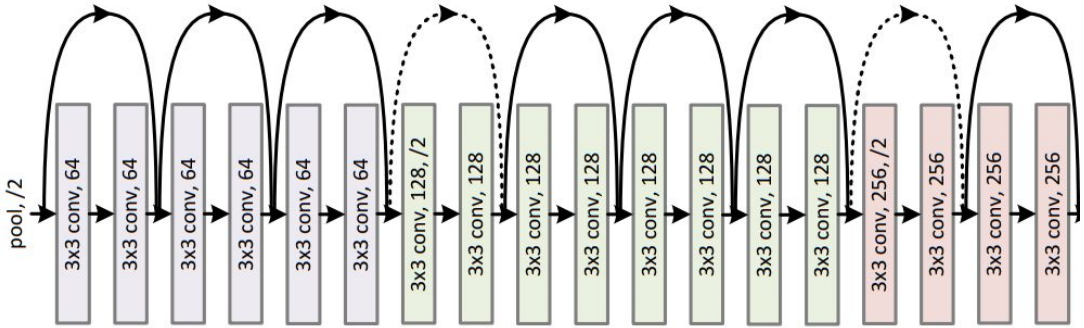




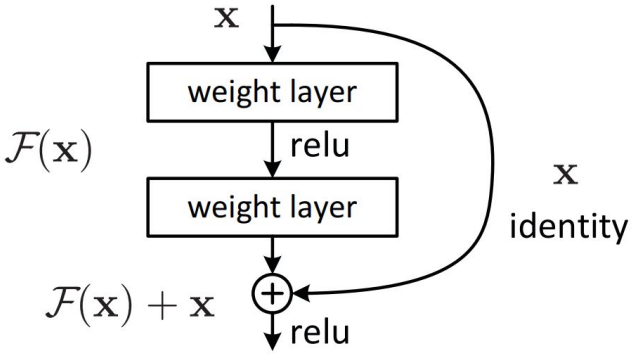
- ImageNet 2012 winner with a big margin
- Start of Deep Learning era
- Manual GPU programming

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
https://www.implantology.or.kr/articles/article/RvNO/

ResNet

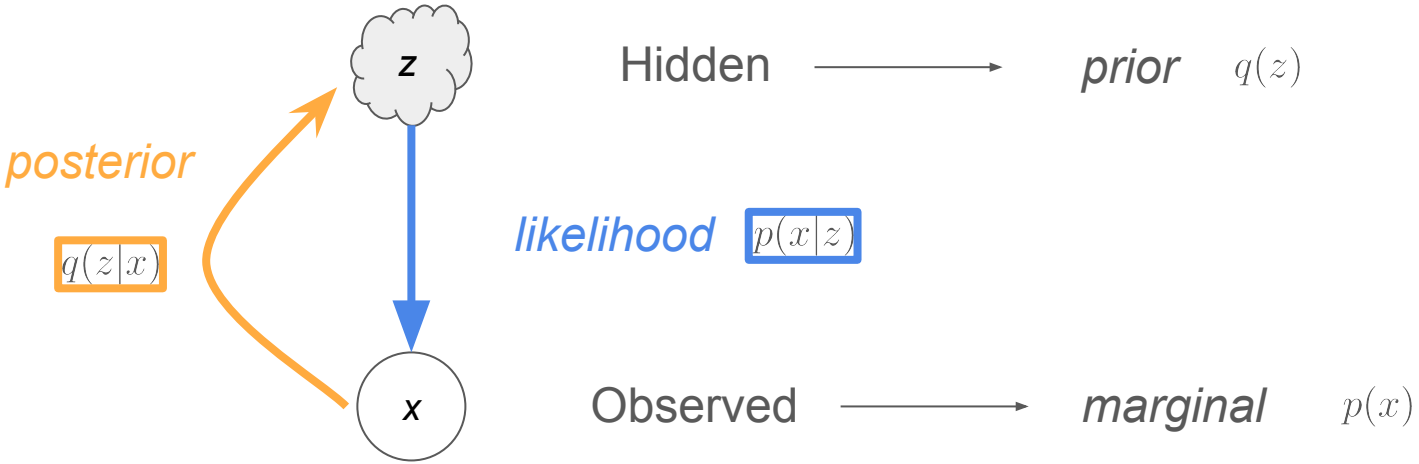


Introduces the "skip-connection" which allows to train really deep networks by addressing vanishing gradients problem.



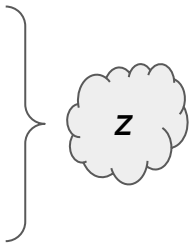
https://arxiv.org/abs/1512.03385

Variational Inference: Bayes' Theorem



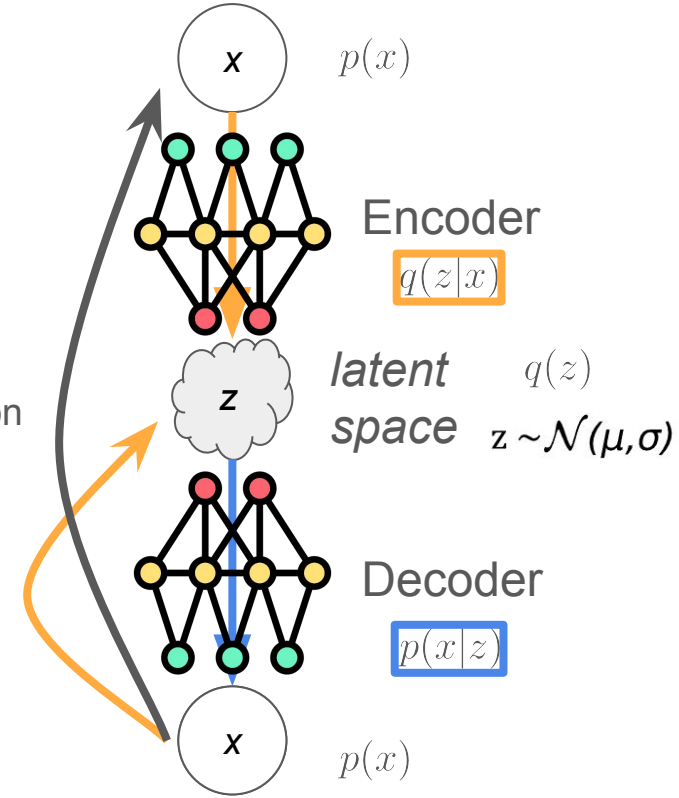
Bayes: $q(z|x) = \frac{p(x|z)q(z)}{p(x)}$

Total probability: $p(x) = \int p(x|z)q(z)dz$



Is hidden!!! Unknown!!!

Variational Inference: The Variational AutoEncoder (VAE)



- Loss: Evidence Lower Bound (ELBO)
  - Difference between true and approximate posterior

$$ELBO = D_{KL}\left(q(z) \parallel q(z|x)\right) + \varepsilon$$

Where

- $D_{KL} \rightarrow$ Kullback-Leibler (KL) divergence
- $\mathcal{E} \rightarrow$ Reconstruction loss