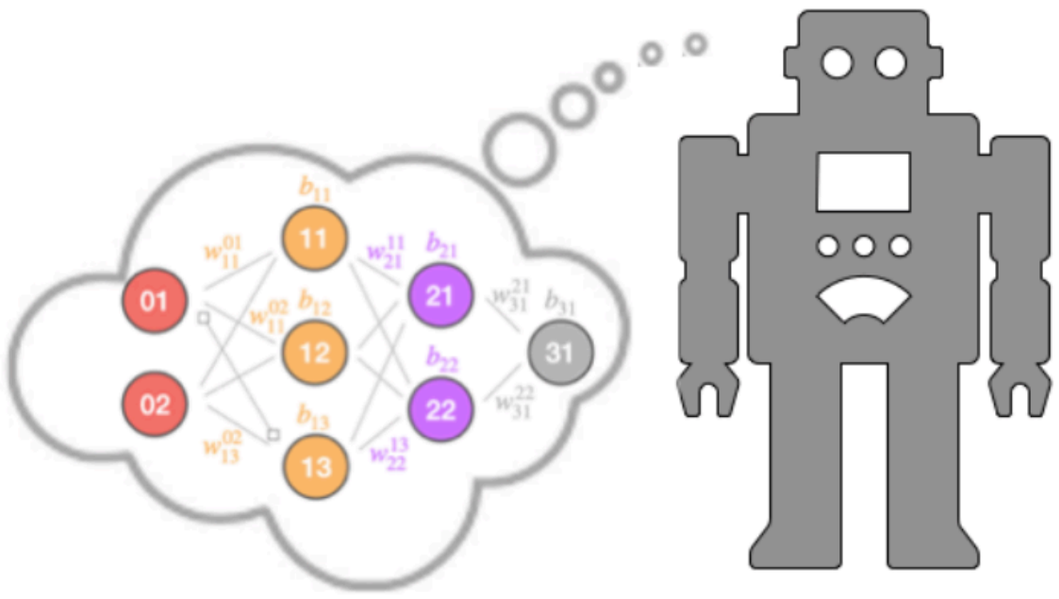
Deep learning

A two hour crash course

(07/13/2026)

Dr. James Hickman



A quick introduction!



Research Interests

- **Computational physics**
- **Atomistic simulations**
- **Machine learning in material science**
- **Neural network interatomic boding models**

PAST

Undergraduate
2007 to 2011

SHIPPENSBURG UNIVERSITY



B.S. in physics and applied mathematics

Graduate school
2012 to 2017

GEORGE MASON UNIVERSITY



M.S. Applied and engineering physics
Ph.D Computational Physics

Post-doc
2018 to 2021

NIST



Machine learning in material science

Assistant teaching professor

NIST +

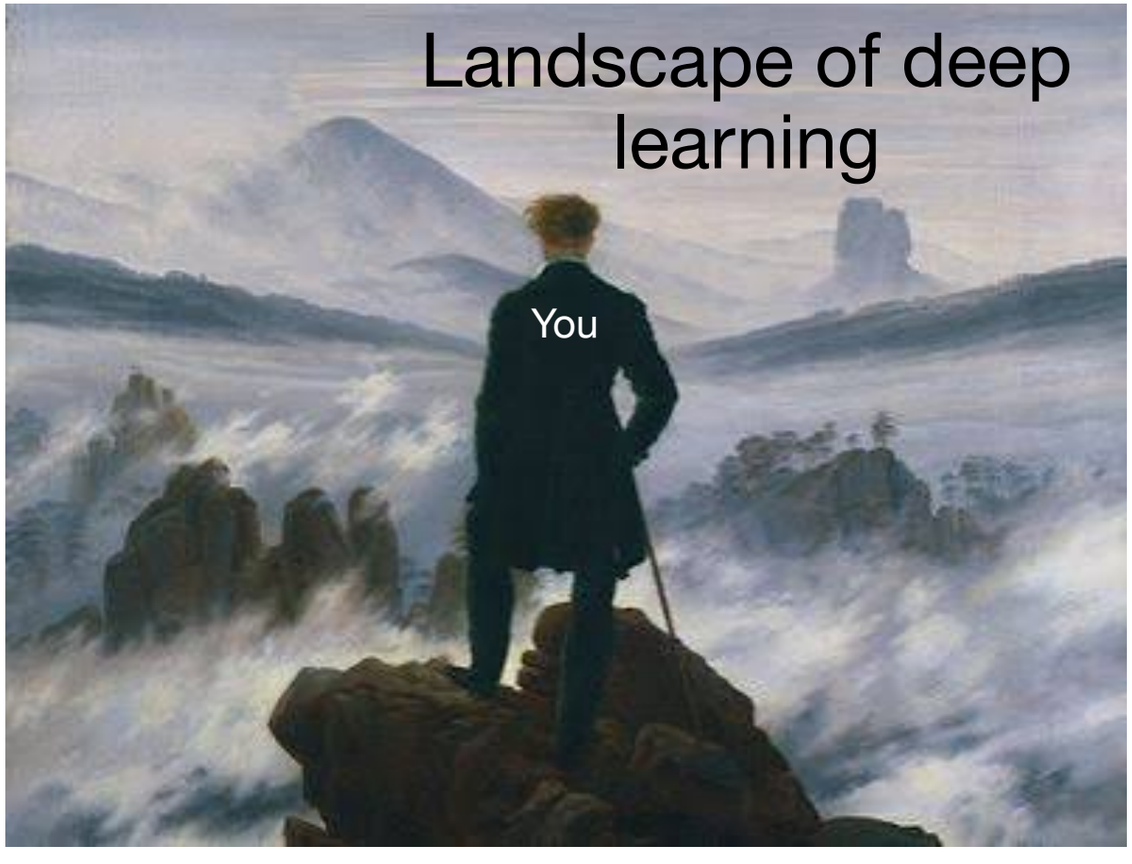


NOW

Todays: Learning objectives

- **Data structures** for deep learning
- **Big picture:** connecting simple regression concepts to deep learning
- **Neural networks:** what they are — an introduction to dense networks
- **Training & deployment:** building networks and avoiding overfitting
- **Learning tasks & model families:** common tasks and neural network architectures
- **Tooling:** intro to Google Colab and why GPUs fuel deep learning

Todays goal: High level survey of the landscape of the field deep learning in 2026



Where does deep-learning fit in the data science life-cycle?

To “get in the zone” after a long weekend let’s recall some data science fundamentals

Roadmap: The data science life-cycle

Everything starts with a question and Data!!



Share your results with the world

Deep learning largely lives in the modeling and validation phase

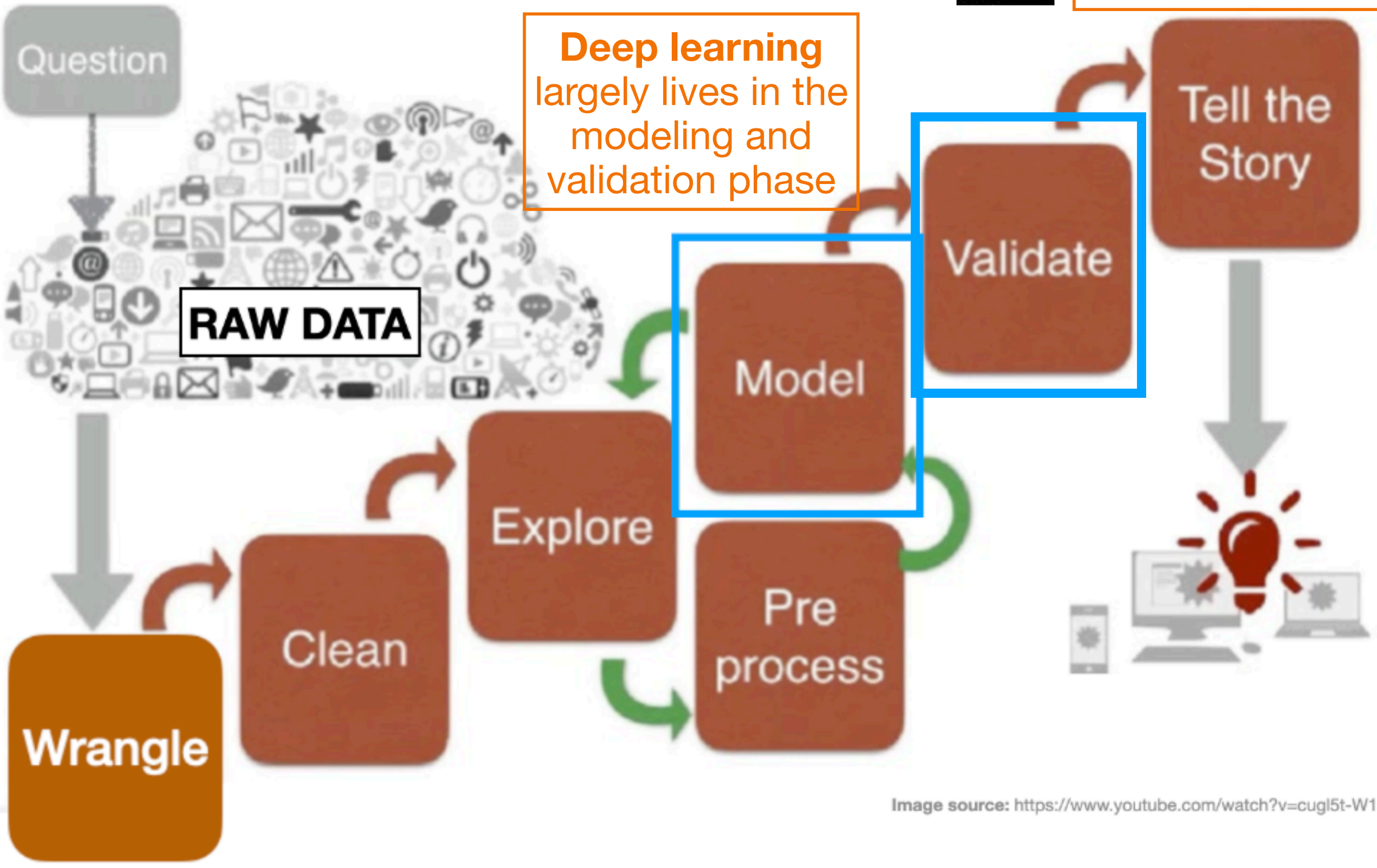


Image source: https://www.youtube.com/watch?v=cugl5t-W1sE



Everything starts with data!!

- What does data look like? The following are examples of common types of data

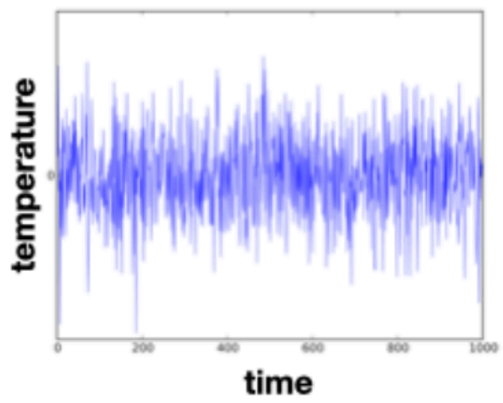
Record Data

ID	height (ft)	weight (lb)	adult/child
1	h_1	w_1	A
2	h_2	w_2	C
3	h_3	w_3	C
⋮	⋮	⋮	
N	h_N	w_N	A

DNA data

GGTTCCGCCTTCAGCCCCGCGCC
CGCAGGGCCCGCCCCGCGCCGTC
GAGAAGGGCCCGCCTGGCGGGCG

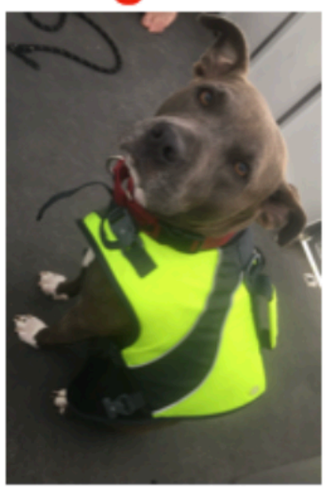
sensor data



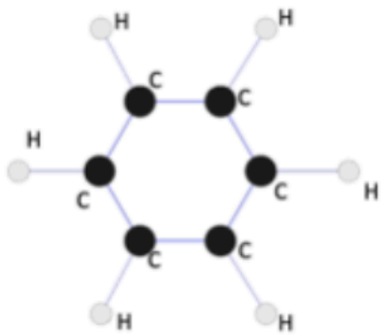
stock data



Image data



Network data



Text data

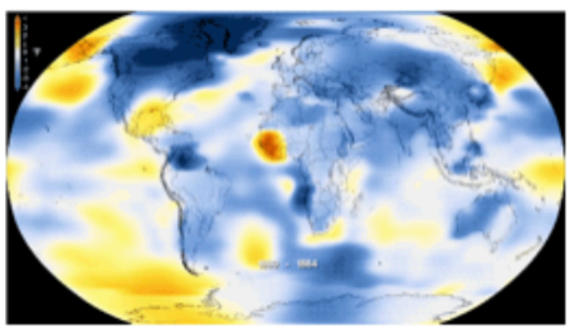


Transaction data

TID	*Items*
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Introduction to Data Mining, 2nd Edition

geo-spacial data



The internal “nature” or characteristic of the data dictate many downstream modeling decisions.

Unit of analysis/observation

- A "unit of analysis" refers to the individual entity or object that data is observed
- It is the fundamental element for which data is recorded, measured, or analyzed.

`Examples`

- Individual
- An image
- Household
- Organization
- Region
- Country
- Customer
- Participant
- Market Segment
- Network Node
- Product
- Event
- Species
- Time Interval
- Document
- Behavior
- Process
- Social Group
- Transaction

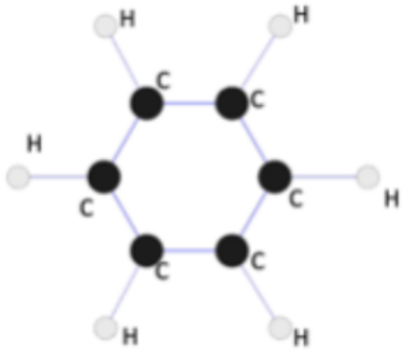
Record Data

ID	height (ft)	weight (lb)	adult/child
1	h_1	w_1	A
2	h_2	w_2	C
3	h_3	w_3	C
⋮	⋮	⋮	
N	h_N	w_N	A

Image data



Network data



This is typically what defines your "x" (model input) and "y" (model output)

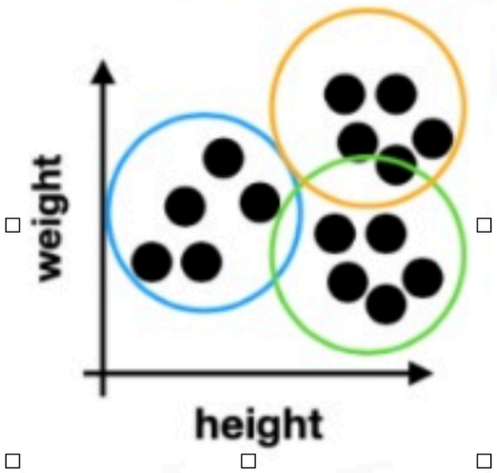
Machine learning paradigms

We will focus primarily on supervised learning today

Machine Learning

Unsupervised Learning

- Dimensionality Reduction
  - Meaningful Compression
  - Big data Visualisation
  - Feature Elicitation
- Clustering
  - Targetted Marketing
  - Customer Segmentation
- Recommender Systems

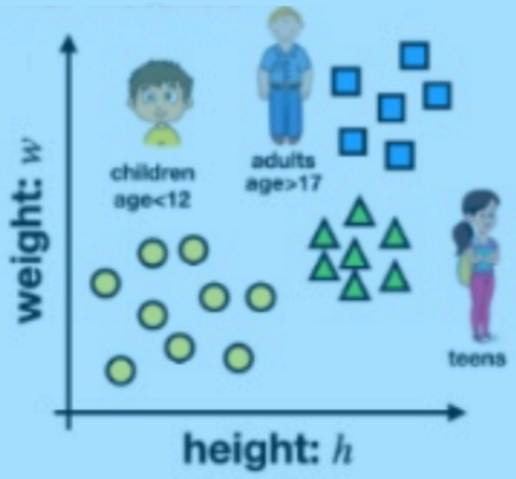


Self-Supervised Learning

- Structure Discovery
- Masked Modeling
- Predictive Coding
- Text Completion
- Image Inpainting
- Content Restoration
- Autoencoding

Supervised Learning

- Classification
  - Image Classification
  - OCR digit identification



- Regression
  - Weather Forecasting
  - Advertising Popularity Prediction
  - Real Estat Appraisal
  - Estimating life expectancy
  - Population Growth Prediction

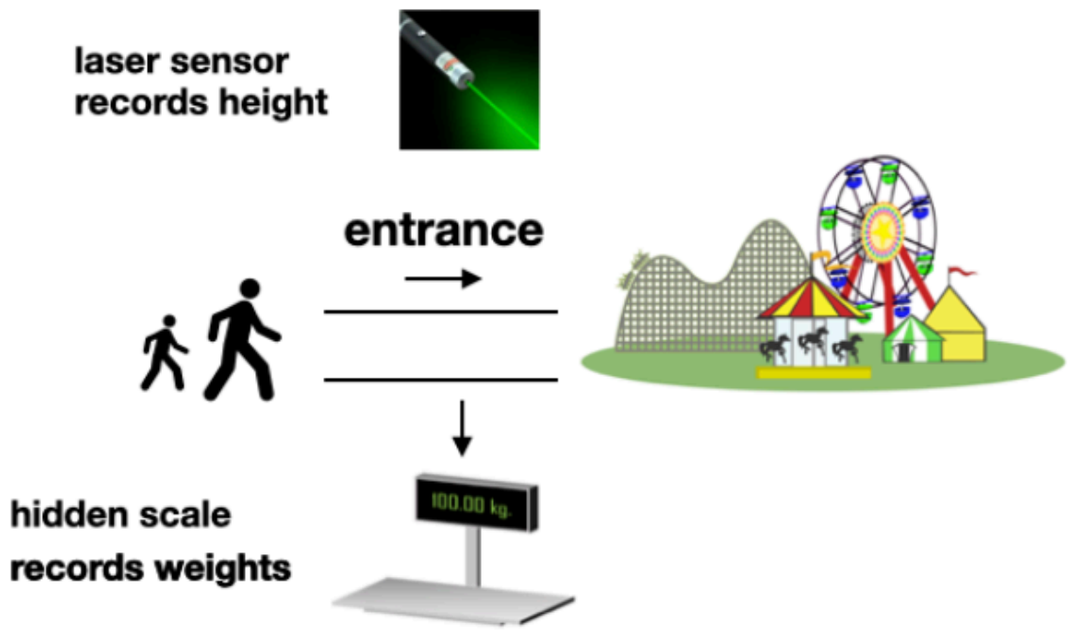


Reinforcement Learning

- Game AI
- Real-time decisions
- Skill Acquisition
- Robot Navigation

Supervision and labels

- Consider an amusement park with a hidden scale and height sensors



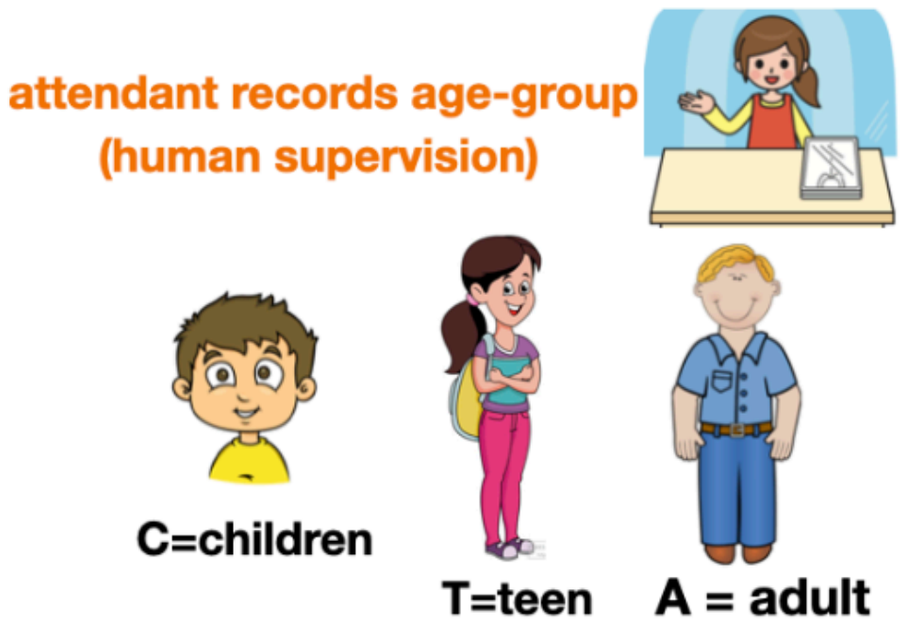
- Data collection is fully automated

Un-labeled data
(i.e. no categorical labels)

index	*ID*	height (ft)	weight (lb)
0	1	h_1	w_1
1	2	h_2	w_2
2	3	h_3	w_3
⋮	⋮	⋮	⋮
$N-1$	N	h_N	w_N



- Now consider a human `supervisor` manually recording categories



- This leads to labeled data

labeled data
(with categorical labels)

index	*ID*	height (ft)	weight (lb)	adult/teen/child
0	1	h_1	w_1	C
1	2	h_2	w_2	A
2	3	h_3	w_3	C
⋮	⋮	⋮	⋮	⋮
$N-1$	N	h_N	w_N	T

- **Labels tag categorical variables**

Categorial variables in deep learning are typically one hot encoded

Supervised learning: Big picture

- Models leverage structure and relationships in the data

Input space: X

- Features — Jargon
- Predictors
- Regressors
- Inputs
- Independent Variables
- Variables

Structure or **relationship**

exists in data

(No hat: Observed target)

Output space: y

- Targets
- Target Variables
- **dependent variables**

- *Target*—The truth. What your model should ideally have predicted, according to an external source of data.

Input space: X → **trained model** / **learns relationship** → **Output space:** $\hat{y}$

- *Prediction* or *output*—What comes out of your model.

Model: $y_{pred} = \hat{y} = m(\mathbf{x}|\mathbf{w})$

(hat: model prediction)

Parameters: Often our model has learnable fitting parameters

Alternative notations for parameters

$\theta \quad \beta$

- Once trained, we use the model to make predictions Y, given an `un-seen` input X
- If no relationship in the data exists, then the model with be useless, the stronger the relationship, the better a well trained model will likely be

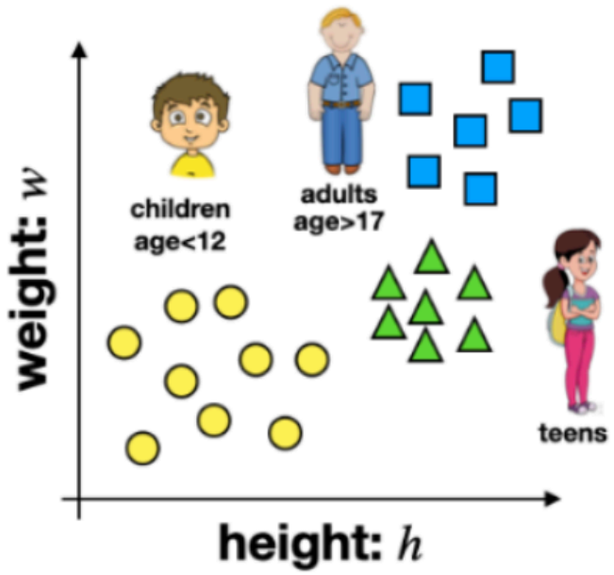
Supervised learning: Example

- The structure and type of data dictate what type of modeling options are available
- In many cases, "learning" just means fitting a parameterization of a function
- **Classification problem**: Known Numerical → Categorical relationships exist.



input : $\mathbf{x} = (x_1, x_2)$ output : $\mathbf{y} = C = (1,0,0)$

ID	height (ft)	weight (lb)	age (years)	income ($)	adult/teen/child
1	h_1	w_1	a_1	i_1	C
2	h_2	w_2	a_2	i_2	A
3	h_3	w_3	a_3	i_3	C
⋮	⋮	⋮	⋮	⋮	⋮
N	h_N	w_N	a_N	i_N	T
	numerical				categorical



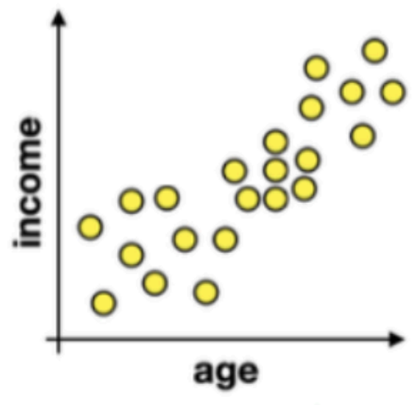
$(h, w) \rightarrow$ **C/T/A=(child/teen/adult)**

- **Regression problem**: Known Numerical → Numerical relationships exist.

ID	Education score	weight (lb)	age (years)	income ($)
1	h_1	w_1	a_1	i_1
2	h_2	w_2	a_2	i_2
3	h_3	w_3	a_3	i_3
⋮	⋮	⋮	⋮	⋮
N	h_N	w_N	a_N	i_N



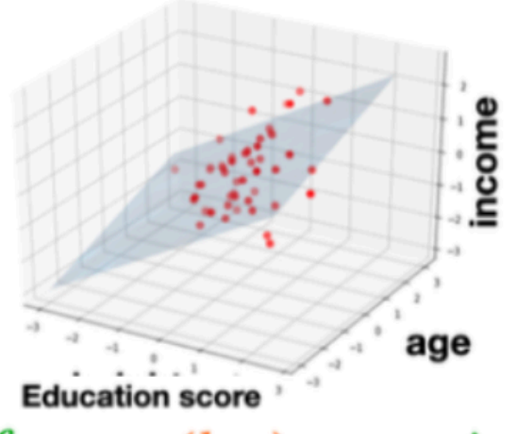
$\mathbb{R}^1 \rightarrow \mathbb{R}^1$



$x = age \rightarrow y = income$

continous continous

$\mathbb{R}^2 \rightarrow \mathbb{R}^1$



$\mathbf{x} = (h, a) \rightarrow y = i$

continous continous

Tabular data

Let's consider some toy data in the form of a standard data frame where we've one hot encoded our categorical variables.

Collect data on houses  **resulting in a cleaned Pandas data-frame**

Feature dimensions

sample (batch) dimension

	continuous		type (one-hot)			target
i	sqft	age	condo	house	town	price
0	1450	12	0	1	0	410.0
1	980	45	1	0	0	265.5
2	2100	3	0	1	0	620.0
3	1320	28	0	0	1	355.2

real-valued binary, exactly one 1 per row what we predict

i^{th} sample input $\mathbf{x}_i = (x_1^i, x_2^i, x_3^i, x_4^i, x_5^i)$ ⟶ $y = continous$

- *Sample* or *input*—One data point that goes into your model.

Learning tasks: Regression

- *Scalar regression*—A task where the target is a continuous scalar value. Predicting house prices is a good example: the different target prices form a continuous space.

	continuous		type (one-hot)			target
i	sqft	age	condo	house	town	price
0	1450	12	0	1	0	410.0
1	980	45	1	0	0	265.5
2	2100	3	0	1	0	620.0
3	1320	28	0	0	1	355.2

real-valued binary, exactly one 1 per row what we predict

Outputs determine learning task

Scalar regression

→ $y = 410.0$

- *Vector regression*—A task where the target is a set of continuous values: for example, a continuous vector. If you're doing regression against multiple values (such as the coordinates of a bounding box in an image), then you're doing vector regression.

	continuous		type (one-hot)			targets (2 outputs)	
i	sqft	age	condo	house	town	price	rent
0	1450	12	0	1	0	410.0	2400
1	980	45	1	0	0	265.5	1650
2	2100	3	0	1	0	620.0	3100
3	1320	28	0	0	1	355.2	2050

real-valued inputs binary, one 1 per row two continuous outputs

Vector regression

→ $\mathbf{y} = (y_1, y_2) = (410.0, 2400)$

Learning tasks: Classification

- *Binary classification*—A classification task where each input sample should be categorized into two exclusive categories.

	continuous		type (one-hot)			target
i	sqft	age	condo	house	town	sold
0	1450	12	0	1	0	1
1	980	45	1	0	0	0
2	2100	3	0	1	0	1
3	1320	28	0	0	1	0

real-valued inputs binary, one 1 per row binary label, 0 or 1

Binary classification

$y = 0$ or 1

- *Multiclass classification*—A classification task where each input sample should be categorized into more than two categories: for instance, classifying handwritten digits.

	continuous		type (one-hot)			label	one-hot for softmax		
i	sqft	age	condo	house	town	status	sold	pending	listed
0	1450	12	0	1	0	sold	1	0	0
1	980	45	1	0	0	listed	0	0	1
2	2100	3	0	1	0	pending	0	1	0
3	1320	28	0	0	1	listed	0	0	1

real-valued inputs one-hot inputs class label one-hot target (softmax)

Multi-class classification

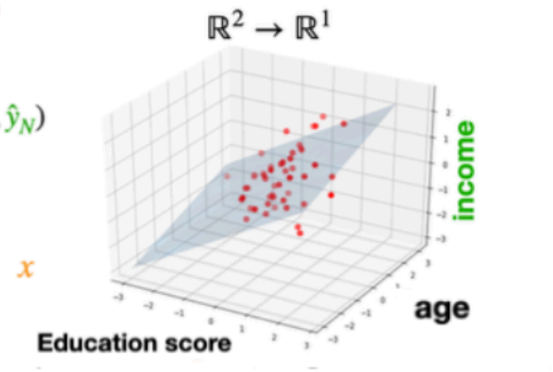
→ $\mathbf{y} = (y_1, y_2, y_3) = (1,0,0)$

Supervised learning: Jargon

Scalar regression (i.e Curve fitting)

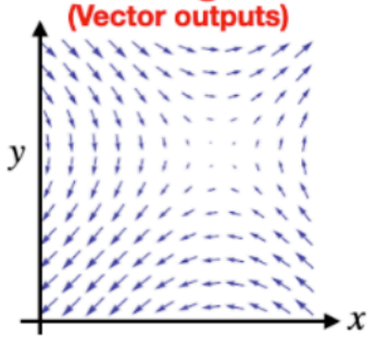
(Scalar outputs)





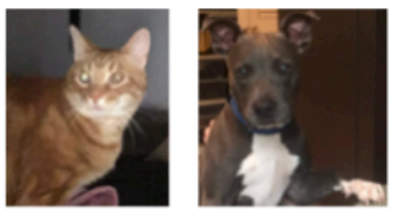
Vector regression

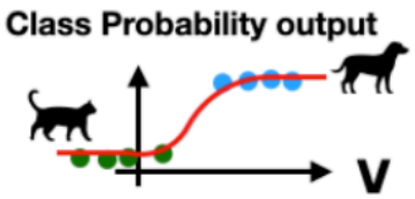
(Vector outputs)



ID	height (ft)	weight (lb)	age (years)	income ($)
1	h_1	w_1	a_1	i_1
2	h_2	w_2	a_2	i_2
3	h_3	w_3	a_3	i_3
⋮	⋮	⋮	⋮	⋮
N	h_N	w_N	a_N	i_N
	numerical		numerical	

Binary Classification

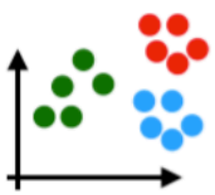




Multi-class Classification

Multiple Class-probability outputs





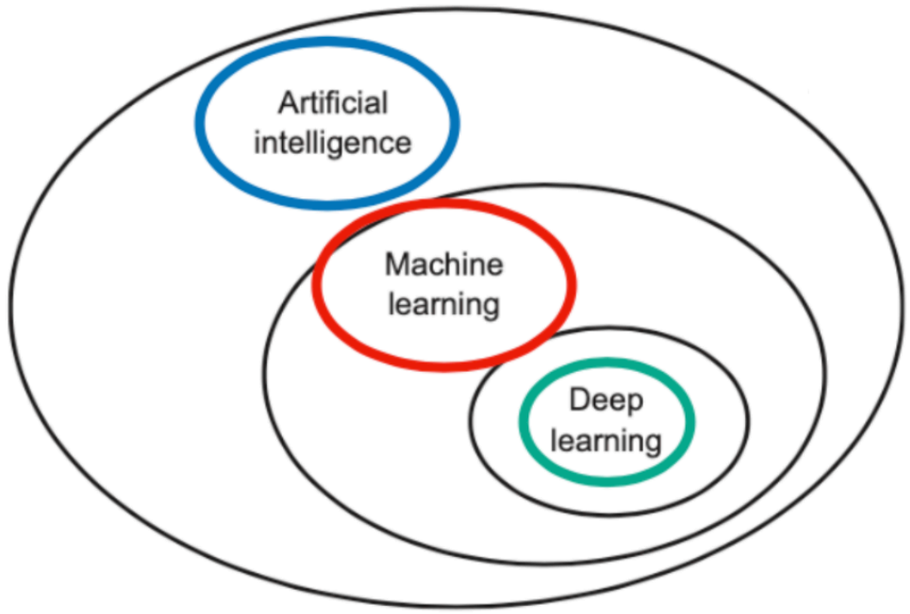
Classification and regression glossary

Classification and regression involve many specialized terms. You've come across some of them in earlier examples, and you'll see more of them in future chapters. They have precise, machine-learning-specific definitions, and you should be familiar with them:

- *Sample* or *input*—One data point that goes into your model.
- *Prediction* or *output*—What comes out of your model.
- *Target*—The truth. What your model should ideally have predicted, according to an external source of data.
- *Prediction error* or *loss value*—A measure of the distance between your model's prediction and the target.
- *Classes*—A set of possible labels to choose from in a classification problem. For example, when classifying cat and dog pictures, "dog" and "cat" are the two classes.
- *Label*—A specific instance of a class annotation in a classification problem. For instance, if picture #1234 is annotated as containing the class "dog," then "dog" is a label of picture #1234.
- *Ground-truth* or *annotations*—All targets for a dataset, typically collected by humans.
- *Binary classification*—A classification task where each input sample should be categorized into two exclusive categories.
- *Multiclass classification*—A classification task where each input sample should be categorized into more than two categories: for instance, classifying handwritten digits.
- *Multilabel classification*—A classification task where each input sample can be assigned multiple labels. For instance, a given image may contain both a cat and a dog and should be annotated both with the "cat" label and the "dog" label. The number of labels per image is usually variable.
- *Scalar regression*—A task where the target is a continuous scalar value. Predicting house prices is a good example: the different target prices form a continuous space.
- *Vector regression*—A task where the target is a set of continuous values: for example, a continuous vector. If you're doing regression against multiple values (such as the coordinates of a bounding box in an image), then you're doing vector regression.
- *Mini-batch* or *batch*—A small set of samples (typically between 8 and 128) that are processed simultaneously by the model. The number of samples is often a power of 2, to facilitate memory allocation on GPU. When training, a mini-batch is used to compute a single gradient-descent update applied to the weights of the model.

Source: Deep Learning with Python: FRANCOIS CHOLLET

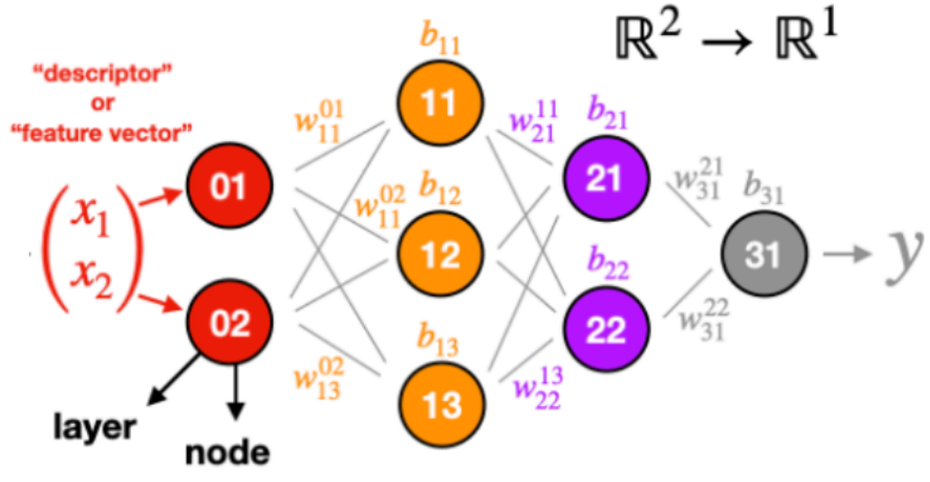
What is deep learning?



Before jumping into the details, let's start with a big picture summary.

Model: $y_{pred} = \hat{y} = m(\mathbf{x} \mid \mathbf{w})$

In deep learning function our model is a special function called a neural network



Deep learning

- Deep learning: learning successive layers of increasingly meaningful representations
  - Also known as layered representations learning or hierarchical representations learning.



Depth=Number of layers (i.e. representations)

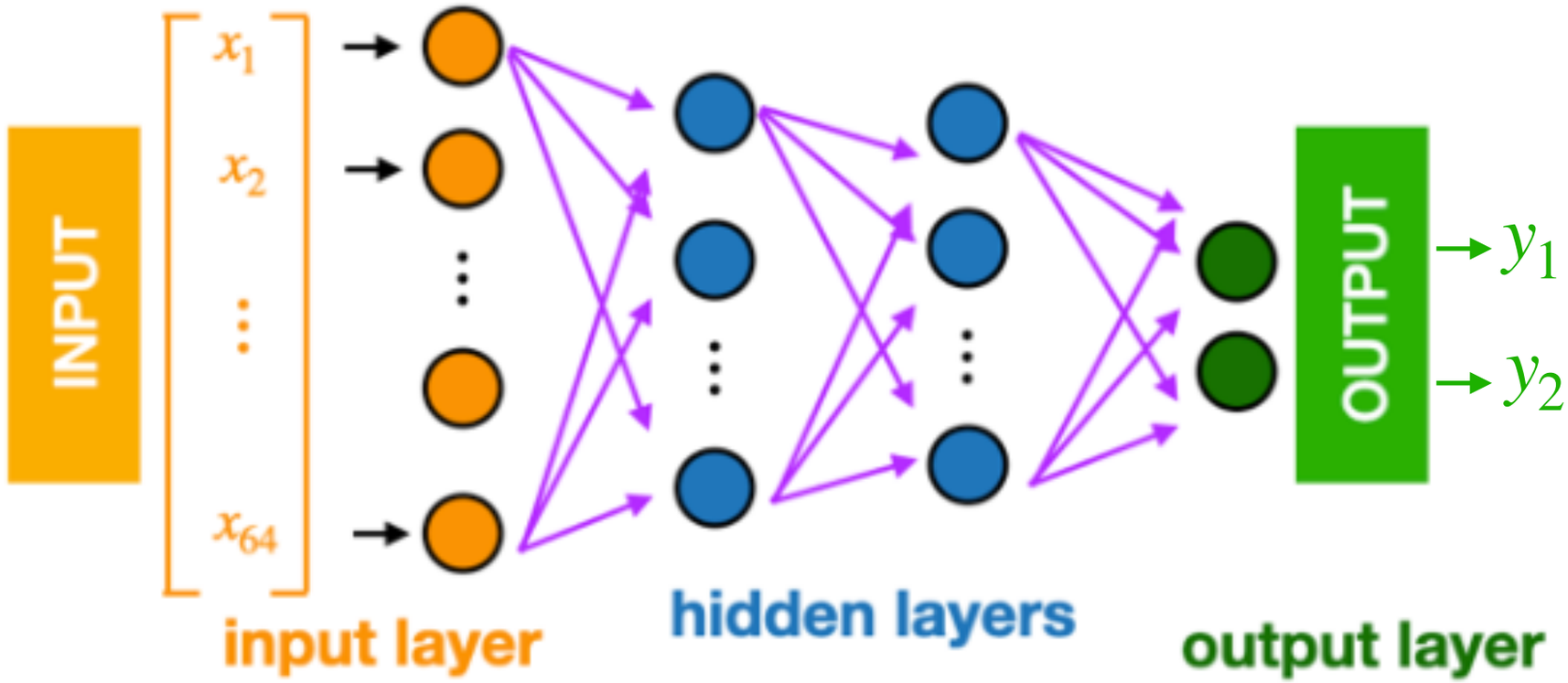
Shallow learning —> only 1 or 2 layers

Deep learning —> more than 2 layers

- **Typically "deep learning" refers to modeling with an artificial neural network**. Deep learning can be used in supervised, un-supervised, or RL applications

Dense artificial neural network (ANN) Model:

single sample point

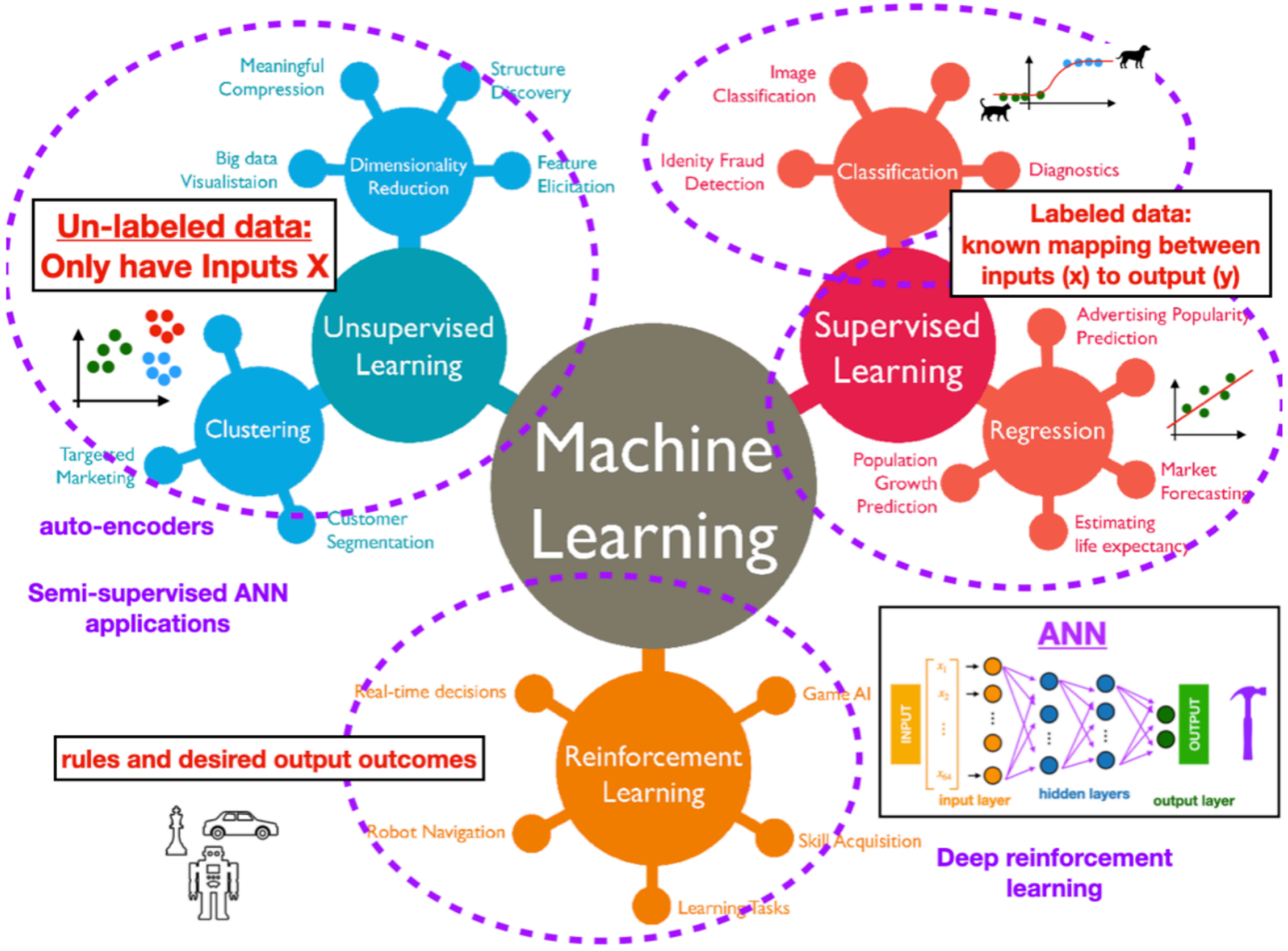


single output point

Todays goal is to understand this model and some of its variants

Where does deep learning live in the space of machine learning?

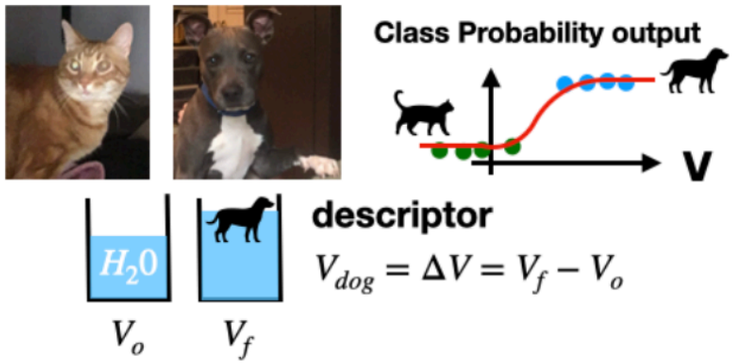
- Artificial neural networks have been so successful that they are commonly used in all subfields of machine learning



Common deep learning problems

- We can apply the ANN modeling paradigm to various familiar ML tasks, here are some of the most common.

Binary Classification

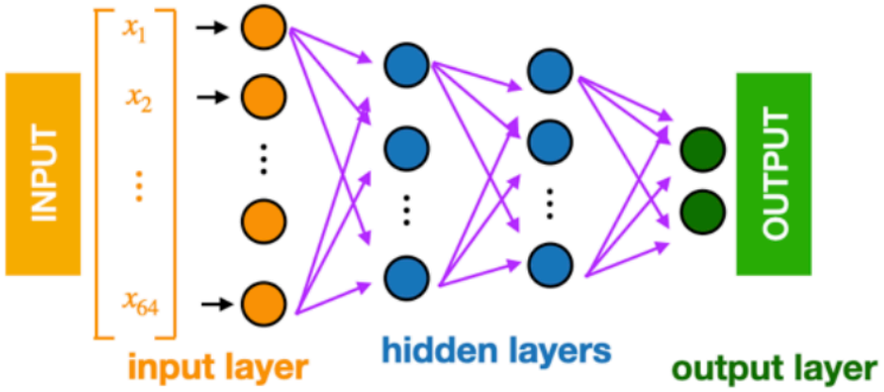


Multi-class Classification

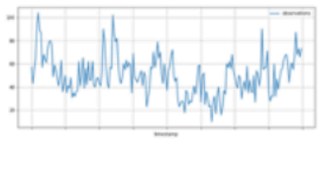
Multiple Class-probability outputs



ANN

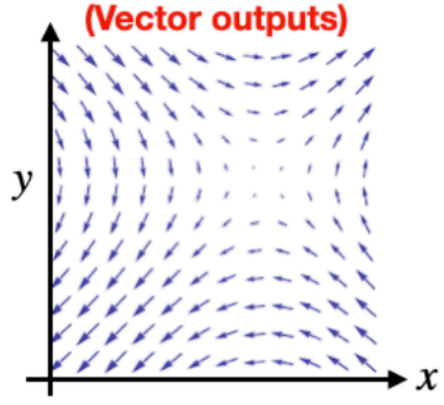


Sequential data and NLP

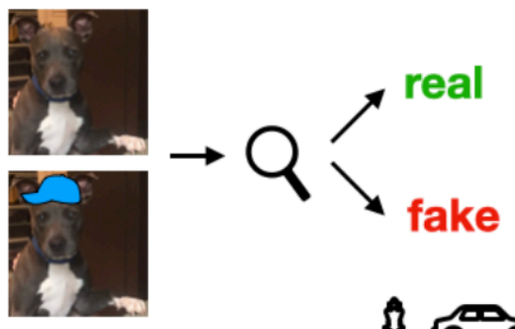


Vector regression

(Vector outputs)

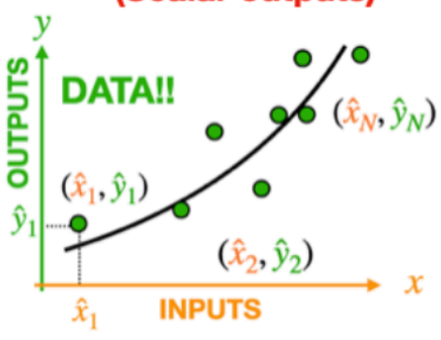


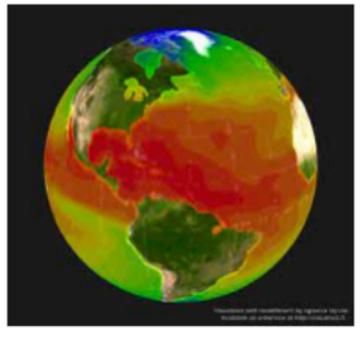
Generative adversarial Networks (GAN)



Scalar regression (i.e Curve fitting)

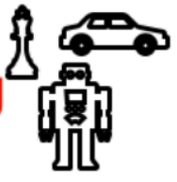
(Scalar outputs)





Temperature = $T(longitude, latitude)$

Deep reinforcement learning

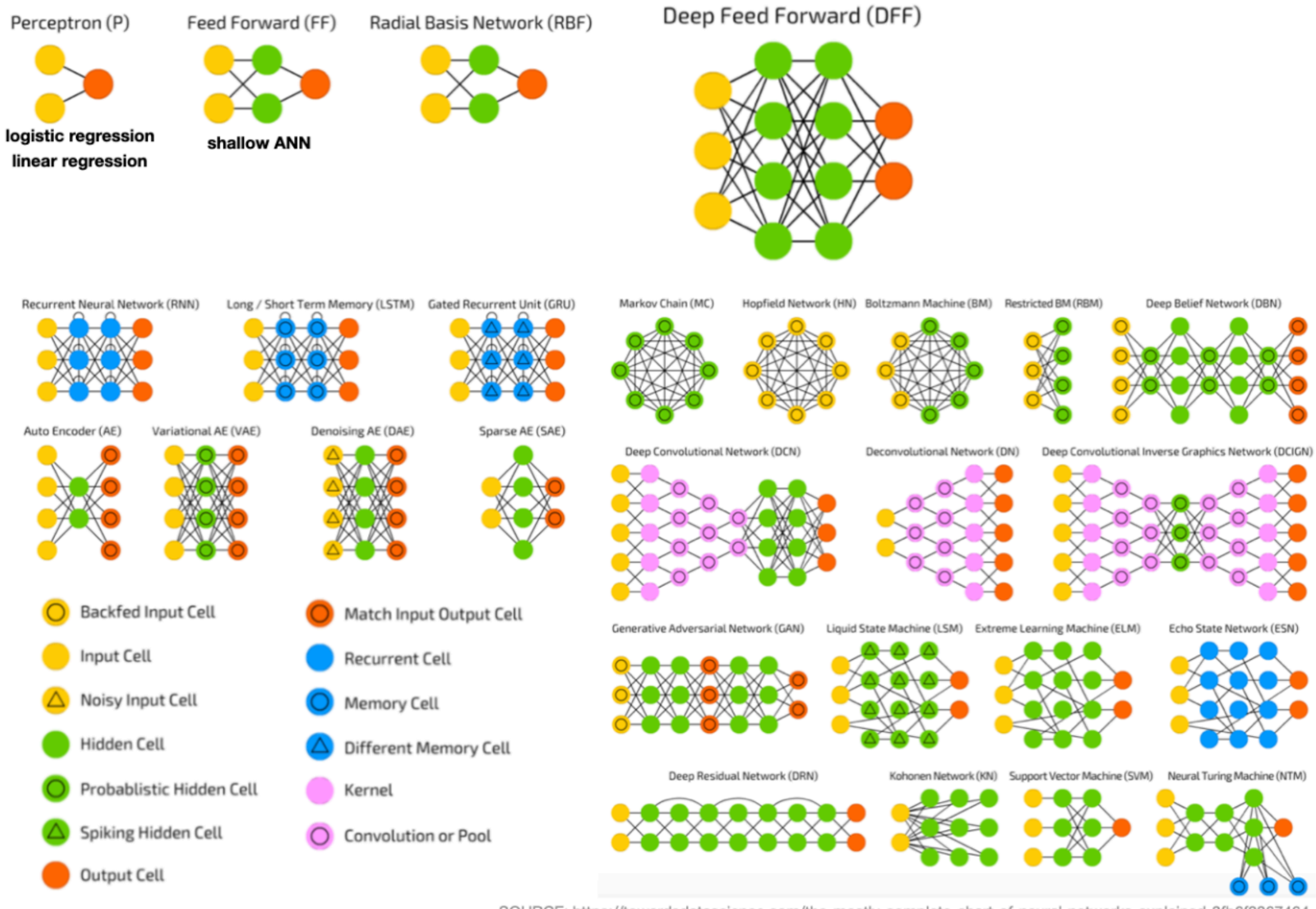




- AI Game Players
- robot navigation
- self driving cars
- google DeepMind
- Alpha Go

Common Neural network architecture

- The type of data you have dictates which architecture you should use
- General classifier/regressor (MLP), image data (CNN), text data (LSTM or Transformer).

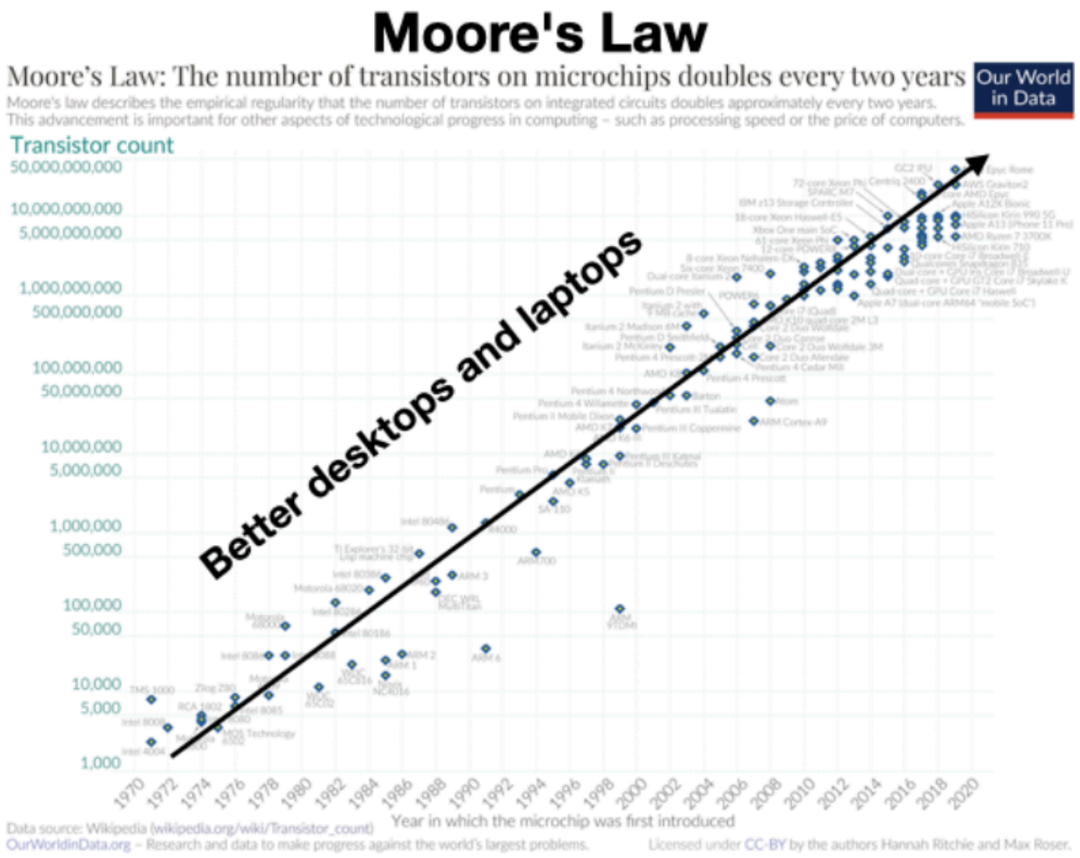


SOURCE: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

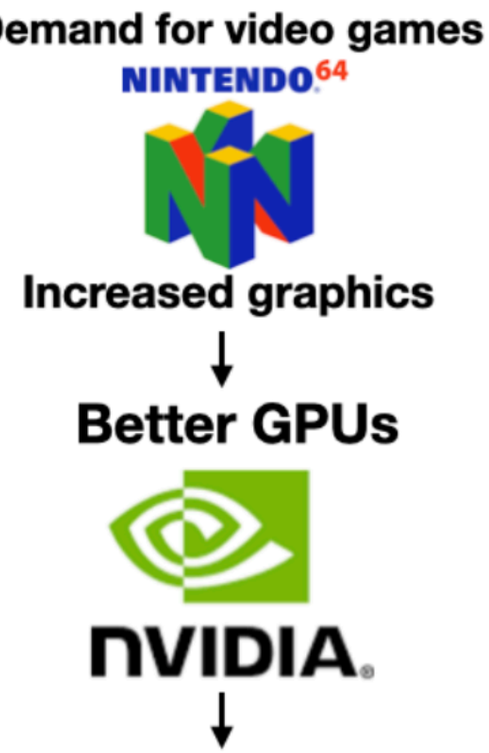
Why now?

- The explosion of computational power, due to Moore’s law and the gaming industry, has enabled efficient and robust training of ML algorithms.





https://en.wikipedia.org/wiki/Moore%27s_law





More efficent linear algebra computation







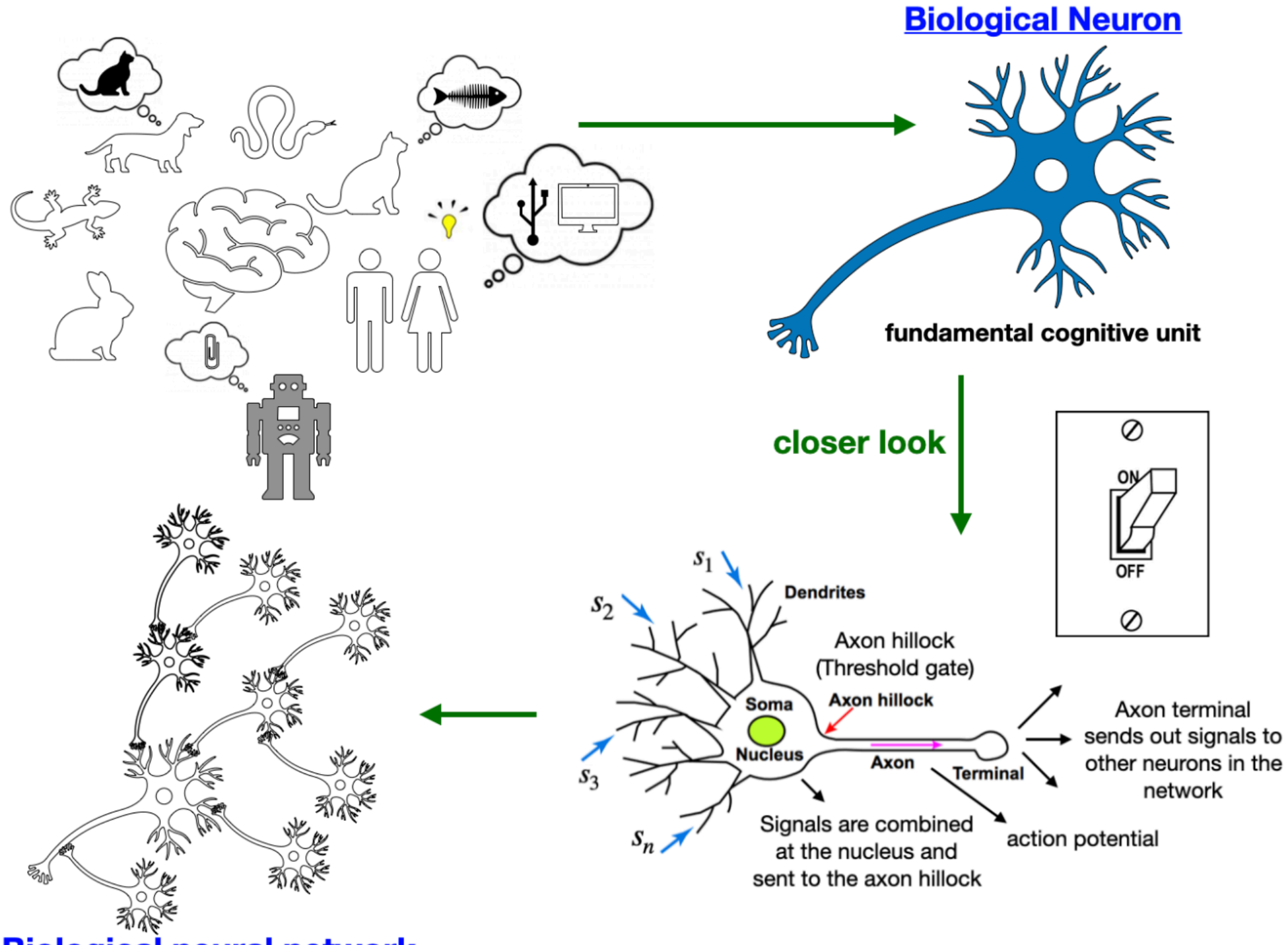






Biological inspiration: neural networks

- As the name suggests, ANNs are loosely inspired by biological neural networks.
- The central nervous system is built of neurons, i.e the fundamental cognitive cell

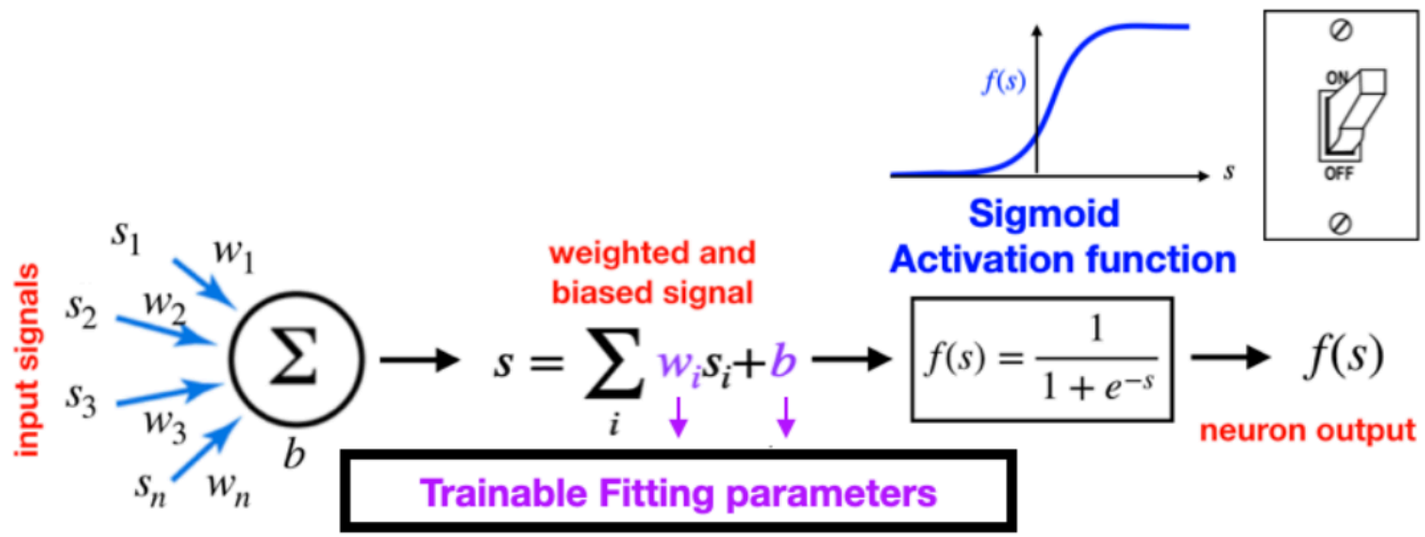


- We can combine these neurons together into a large network of connected cells

Artificial neural networks

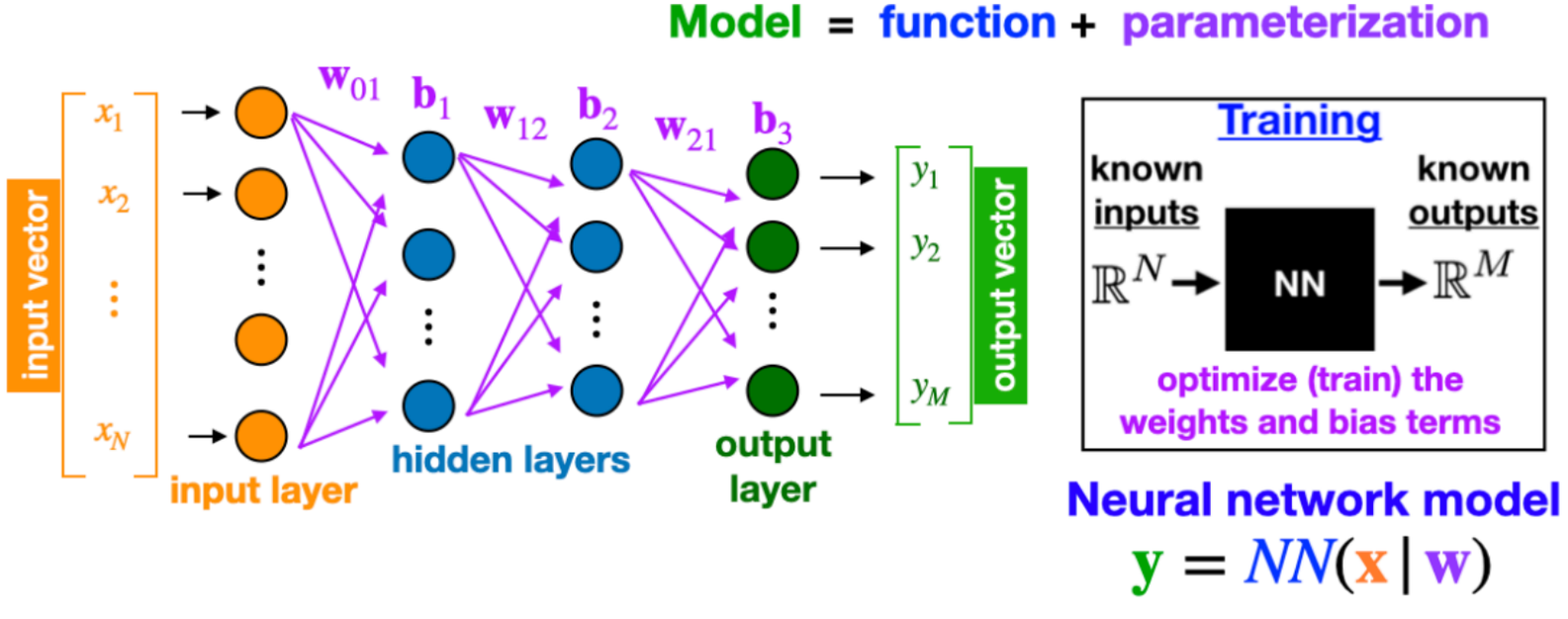
Artificial neuron

The fundamental unit of a neural network is the artificial neuron, which for sigmoidal activation, can be interpreted as a binary logistic regressor



Artificial neural network

We can combine these together in a **layered fashion**, left to right, to form an ANN



Deep learning software tools

MOST POPULAR TOOLKITS

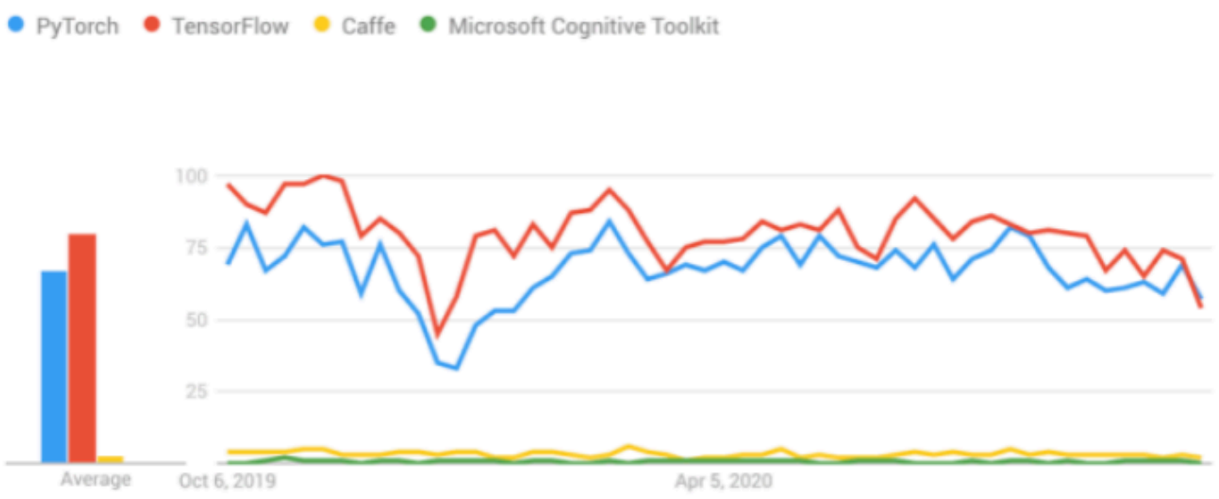
- **TENSOR-FLOW**
- **PYTORCH**
- **THEANO (MILA U-Montral)**
- **COGNITIVE TOOLKIT CNTK**
  - **(MICROSOFT)**

Keras

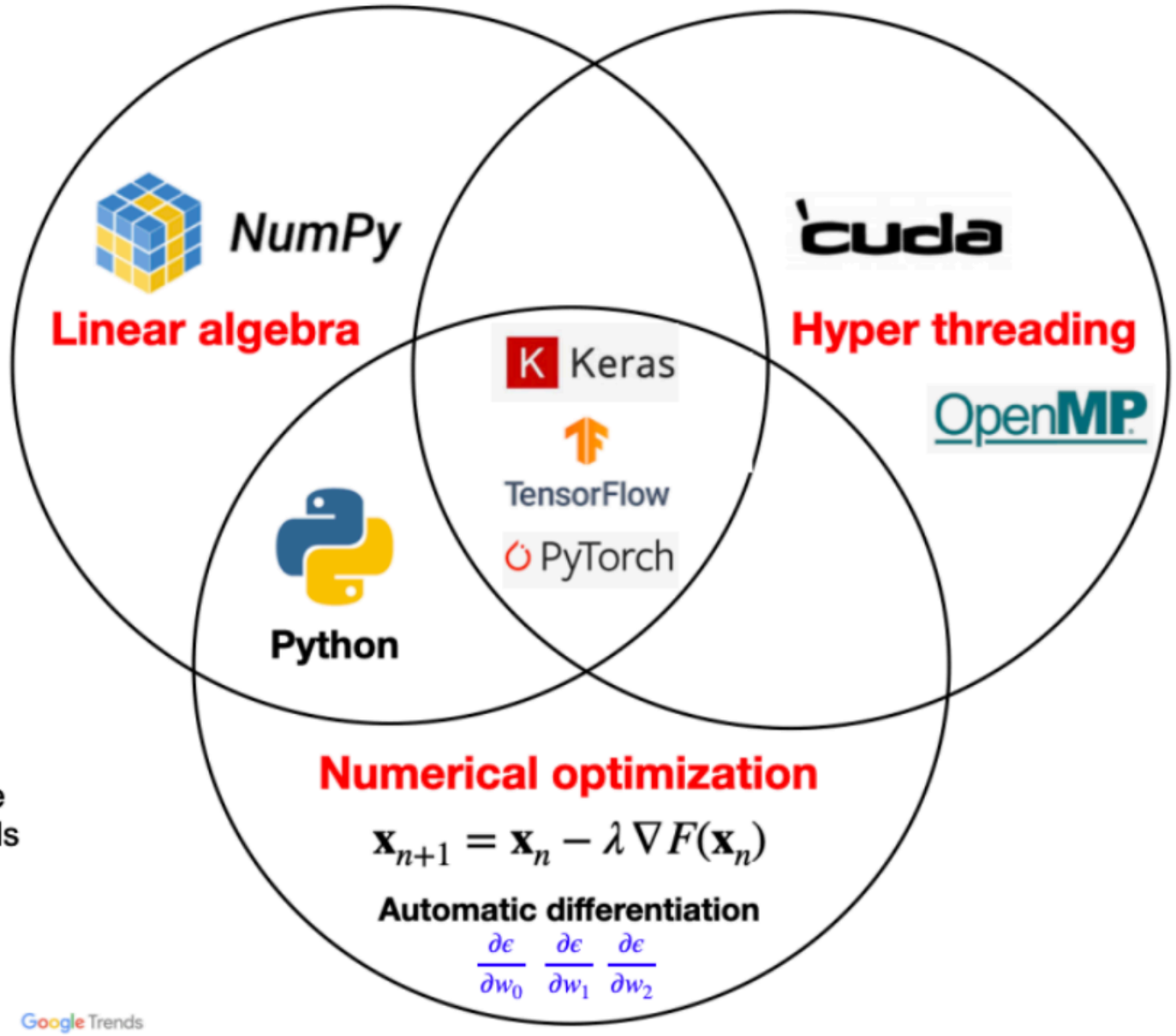
- Keras is an **open-source** software library
- **High-level API**: Application Programming Interface
- Provides a **python interface** for deep learning tools
- **Runs on top of TensorFlow,**
  - or Microsoft CNTK, or Theano
- **Easy to use**

Interest over time

Google Trends



United States. Past 12 months. Web Search. https://www.tooploox.com/blog/pytorch-vs-tensorflow-a-detailed-comparison

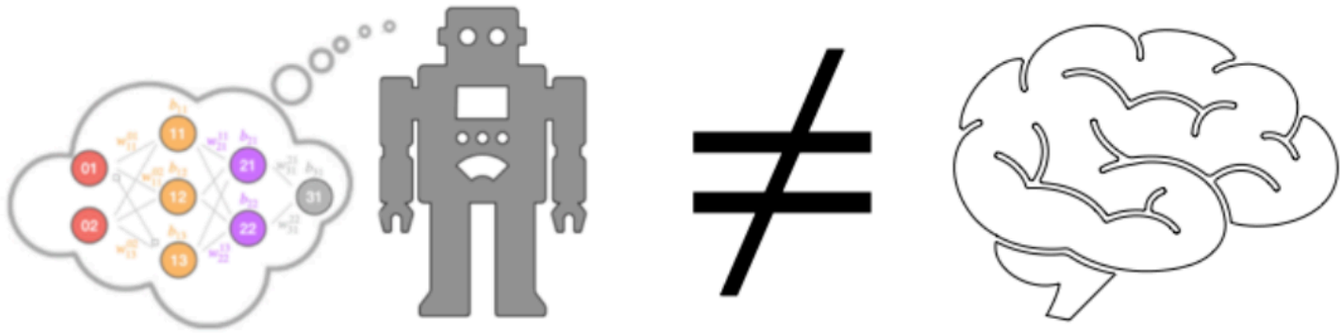


Keras + TensorFlow

PyTorch

- provide exposure
- Add extra line to vour resume

Word of caution: Biological vs ANN



Narrow vs general AI

It is important to distinguish between two fundamental paradigms of Artificial intelligence

Narrow AI	General AI
○ Application specific/ task limited	○ Perform general (human) intelligent action
○ Fixed domain models provided by programmers	○ Self-learns and reasons with its operating environment
○ Learns from thousands of labeled examples	○ Learns from few examples and/or from unstructured data
○ Reflexive tasks with no understanding	○ Full range of human cognitive abilities
○ Knowledge does not transfer to other domains or tasks	○ Leverages knowledge transfer to new domains and tasks
○ Today's AI	○ Future AI?

Pretrained vs. From Scratch

Trained from Scratch

- **Def:** Random-initialized weights, trained on *your* data only
- **Use when:**
  - Domain is unique, no relevant pretrained model exists
  - You have **large labeled datasets**
  - Architecture is novel
- **Examples:** custom tabular nets, novel research architectures, proprietary sensor/signal data

Pretrained (Transfer Learning) — *most common*

- **Def:** Reuse weights from a large general corpus (ImageNet, Common Crawl); freeze backbone, swap task head
- **Use when:**
  - Limited data or compute
  - Standard modality (image / text / audio)
  - Fast prototyping
- **Examples:** ResNet/ViT features, BERT/GPT embeddings, Whisper, CLIP
- **LLMs** (GPT, Llama, Claude) are a class of pretrained models — trained once on massive text corpora, then reused via prompting or fine-tuning

Fine-Tuning Overview

What It Is

- Start from **pretrained weights**, then **continue training** on task-specific data
- Adapts general knowledge to your domain without training from scratch
- Middle ground: more capable than frozen features, cheaper than scratch

How Much to Train

- **Feature extraction:** freeze backbone, train only new head (least compute)
- **Partial fine-tuning:** unfreeze top layers, keep lower layers frozen
- **Full fine-tuning:** update all weights (most compute, best fit)
- **PEFT (LoRA, adapters):** train small added params, freeze the rest — standard for LLMs

Use When

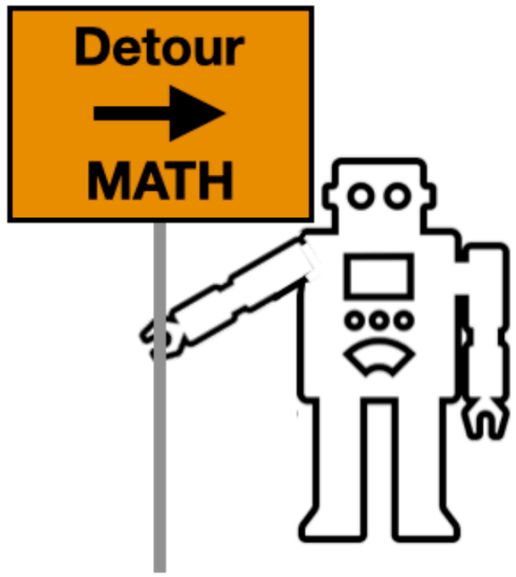
- Moderate task-specific data available
- Target domain shifts from the pretraining distribution
- You need peak task accuracy, not just off-the-shelf features

Key Considerations

- **Learning rate:** use small LR to avoid erasing pretrained knowledge
- **Overfitting:** small datasets overfit fast — use early stopping, regularization
- **Catastrophic forgetting:** aggressive tuning can wipe general capabilities
- **Cost vs. gain:** PEFT often matches full fine-tuning at a fraction of the compute

(Going deeper …)

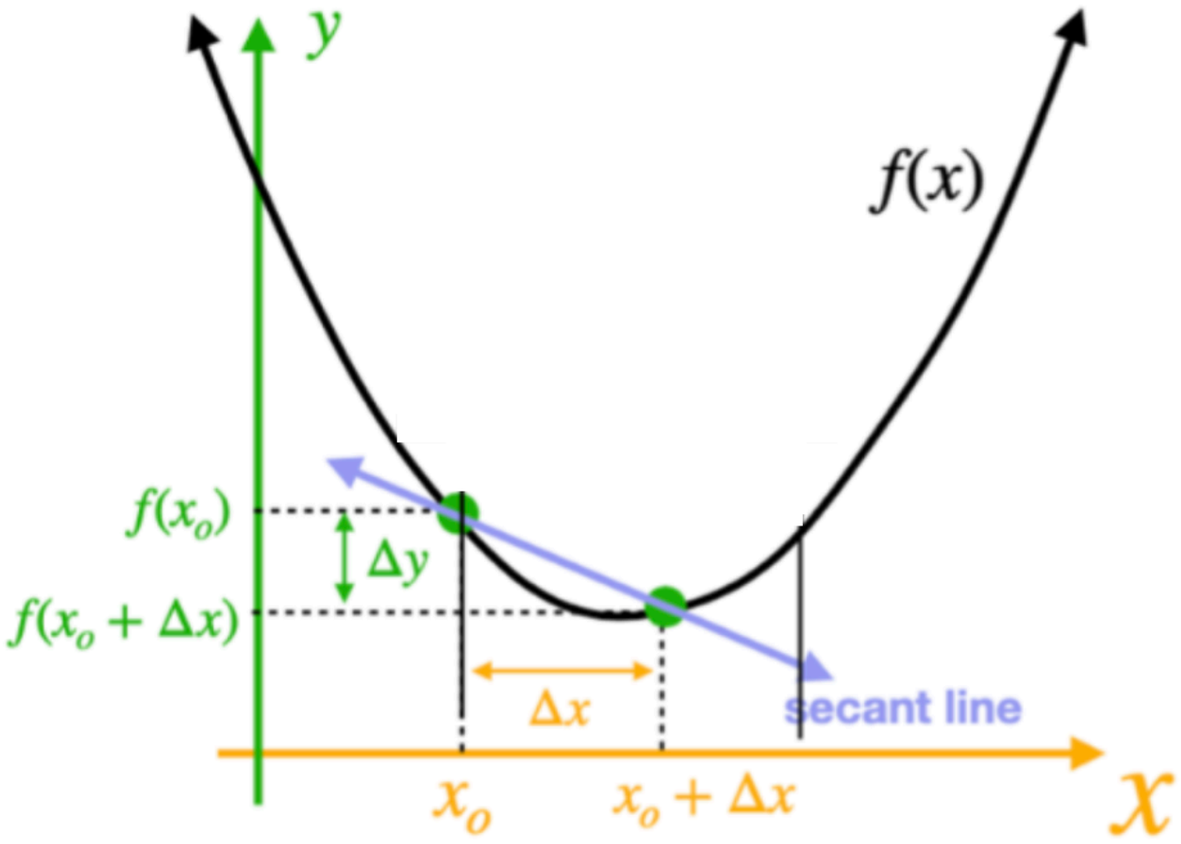
10 Minute Calculus Refresher



In this section we will use the standard mathematical notation where X is the independent variable and Y is the dependent variable.

Recall: Univariate Derivative of a function

- Differentiation finds the instantaneous slope of the function at a particular point x_0



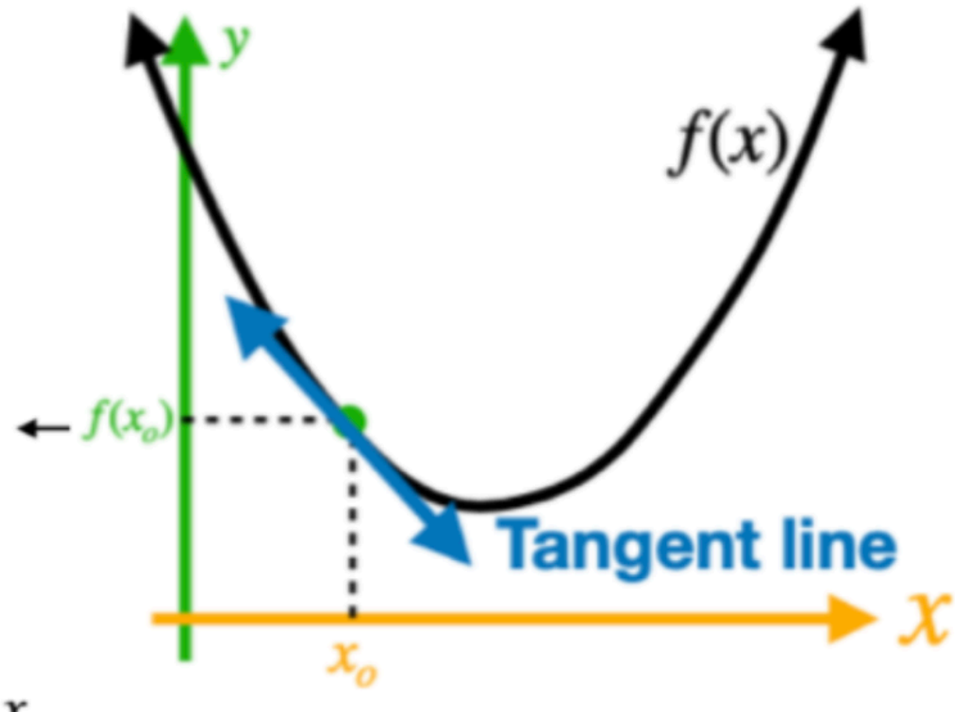
Slope of secant line

$$m = \frac{rise}{run} = \frac{\Delta y}{\Delta x} = \frac{f(x_o + \Delta x) - f(x_o)}{\Delta x}$$

↓

strink Δx till it's infinitally small

Secant line → Tangent line



Slope of tangent line

$$f'(x_o) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x_o + \Delta x) - f(x_o)}{\Delta x}$$

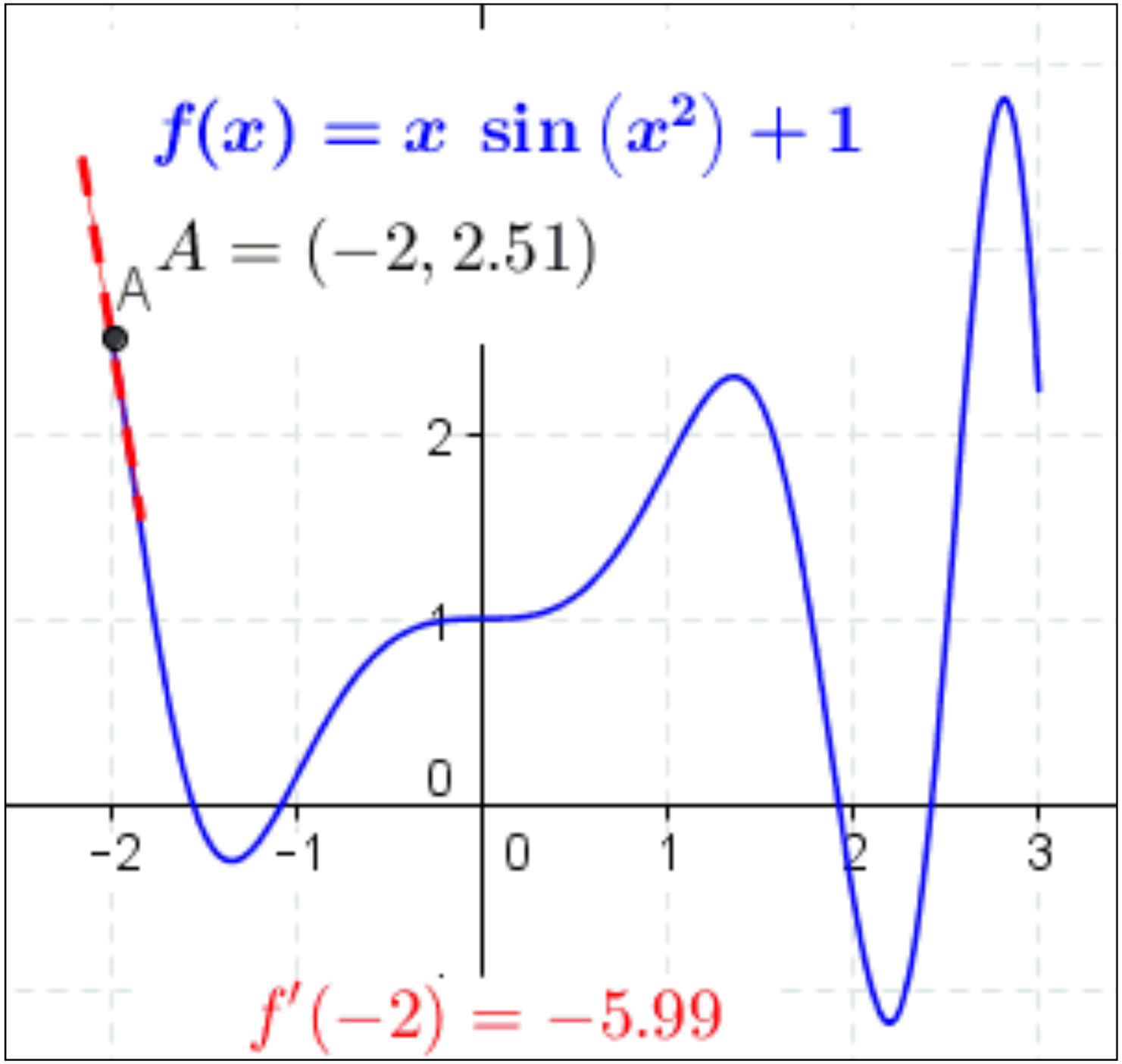
(Limit definition of derivative)

$f'(x_o)$ **= instantaneous rate of change of f(x) at** x_o

Think of the derivative as a “magic” compass that allows us to “find” the “downward” direction from any position on a function.

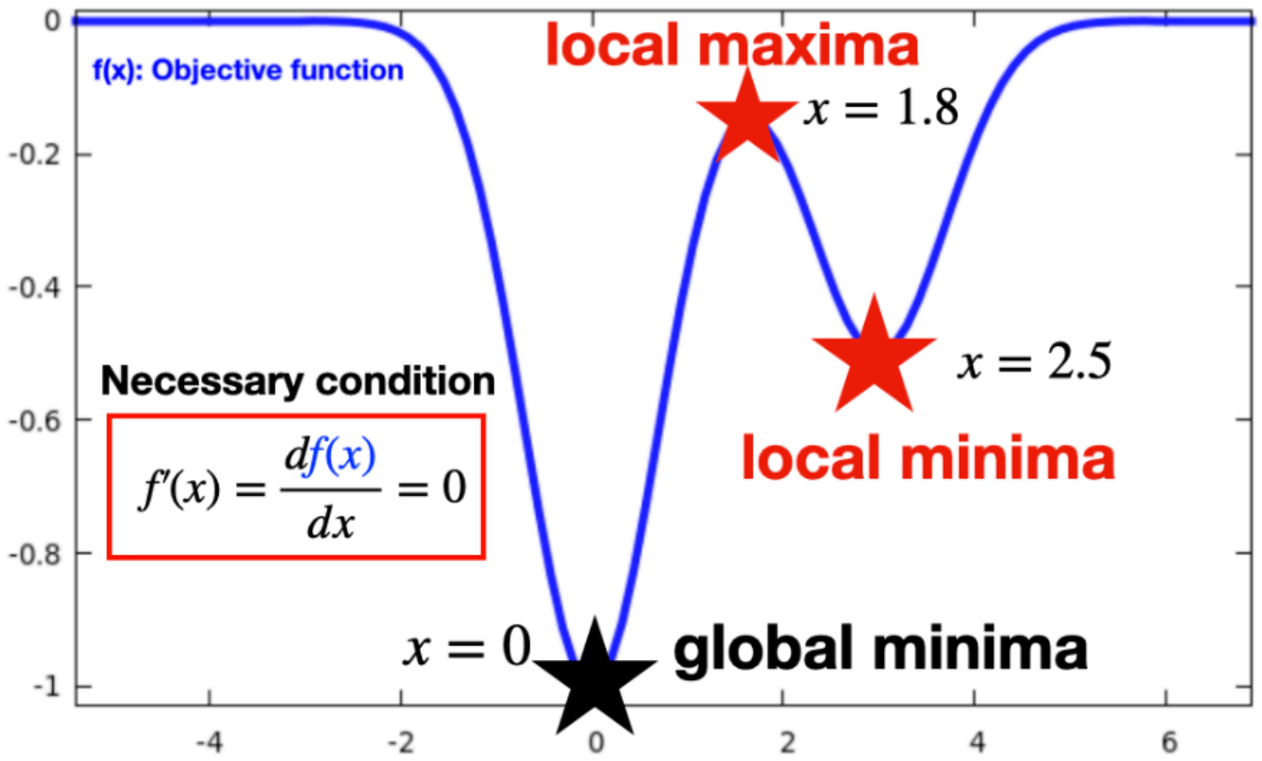
Example: Derivative of a function

- The derivative $f'(x) = \frac{df}{dx}$ is a new function describing the slope of the first.
- It provides the slope of the tangent line of the original function at each point
  - I.e. you give me a some x_0 and I'll evaluate $f'(x_0)$ to give you the slope at x_0



Stationary points

- Many properties of f can be determined by the derivatives f' and f''
- Stationary points, i.e. minima, maxima, and inflection points, refer to special points where the function does not change $f' = 0$ (i.e. horizontal tangent line).
- These can be either global (lowest or highest point) or local



Stationary points

$$\frac{df}{dx} = 0$$

minima
maxima
constants

- **Optimization** attempts to find minima or maxima of some **objective function**.

Why do we care? **(LEARNING = OPTIMIZATION)**

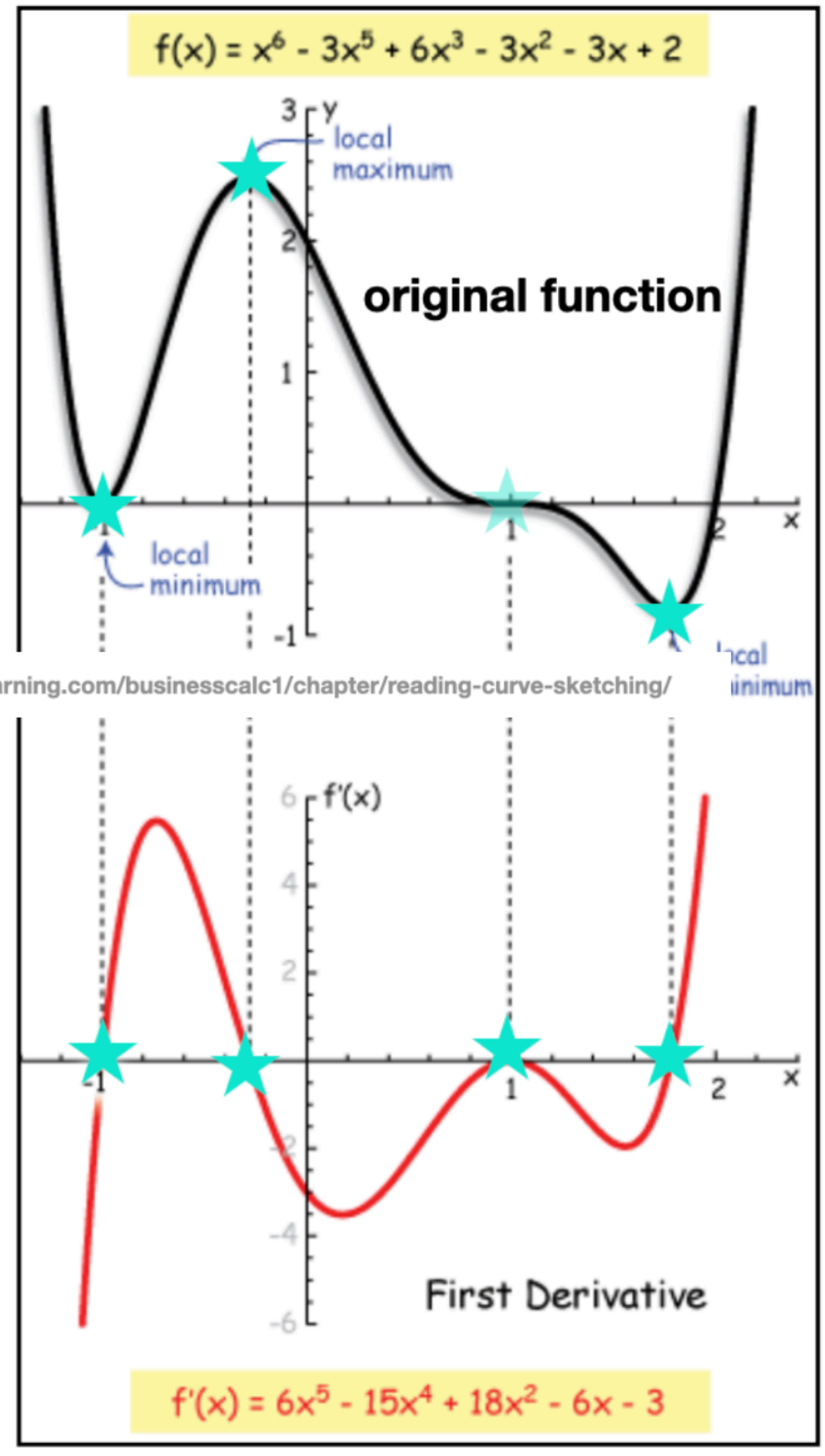
To build a good deep learning model, we use a **loss function** that **scores** the quality of a model's learning parameters. We then "learn" by **minimizing** that error described by the loss function. This connects deep learning directly to the world of mathematical optimization.

Example: Stationary points as roots of the derivative

Stationary points

$$\frac{df}{dx} = 0$$

minima
maxima
constants



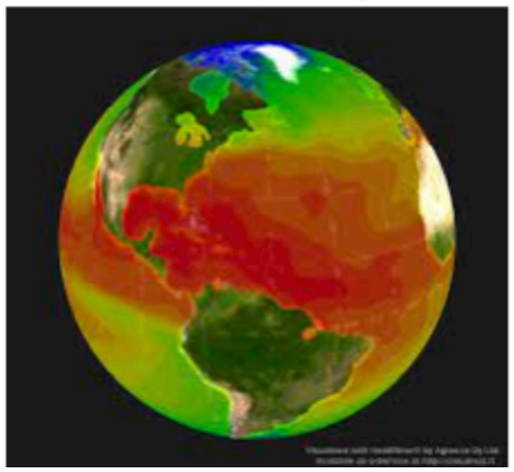
Multi-variable function types

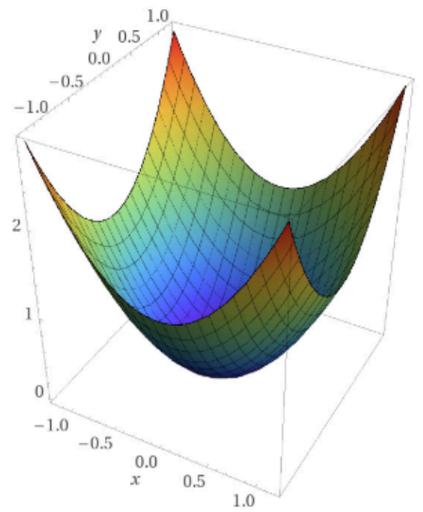
Don't over think it: you give the function an **input** vector (list of numbers), it outputs a scalar (number) in return

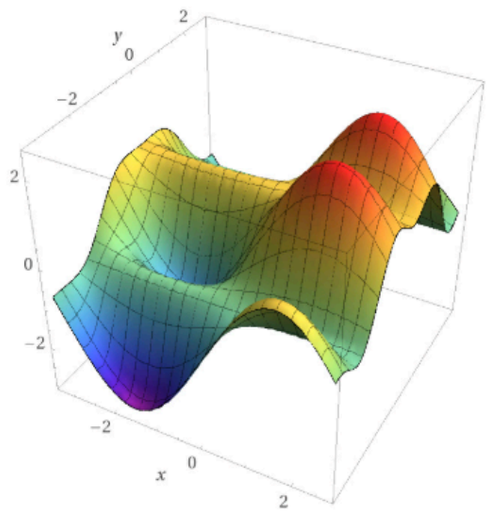
- **Scalar fields:** $y = f(\mathbf{x}) = f(x_1, x_2, \ldots, x_N) \quad \mathbb{R}^N \to \mathbb{R}^1.$
  - **IMPORTANT:** In regression, the objective function is called the loss function L (or loss surface). For parametric modeling, the loss surface is a scalar field over the model parameters. (e.g. $L(\mathbf{w})$ depending on the parameter notation).

vector input: $\mathbf{x}$
scalar output: y

$$T = T(\theta, \phi)$$



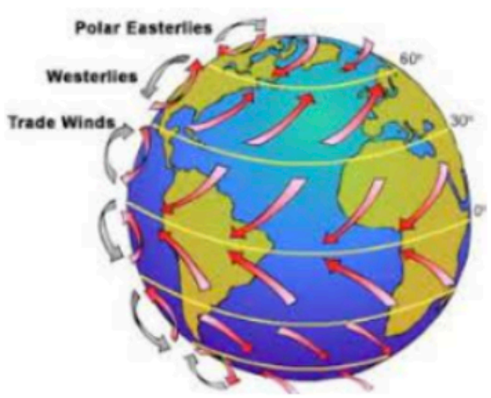


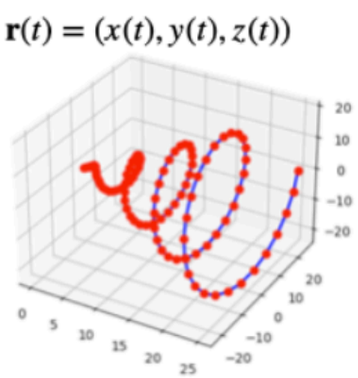


Don't over think it: you give the function an **input** vector (list of numbers), it outputs a list of numbers (vector) in return

- **Vector fields:** $\mathbf{y} = f(\mathbf{x}) = \mathbf{f}(x_1, x_2, \ldots, x_N) = (y_1, y_2, \ldots, y_M) \quad \mathbb{R}^N \to \mathbb{R}^M.$
  - **IMPORTANT:** The loss function's gradient $\nabla L(\mathbf{w})$ in regression is a vector field.

vector or scalar input: $\mathbf{x}$
vector output: $\mathbf{y}$

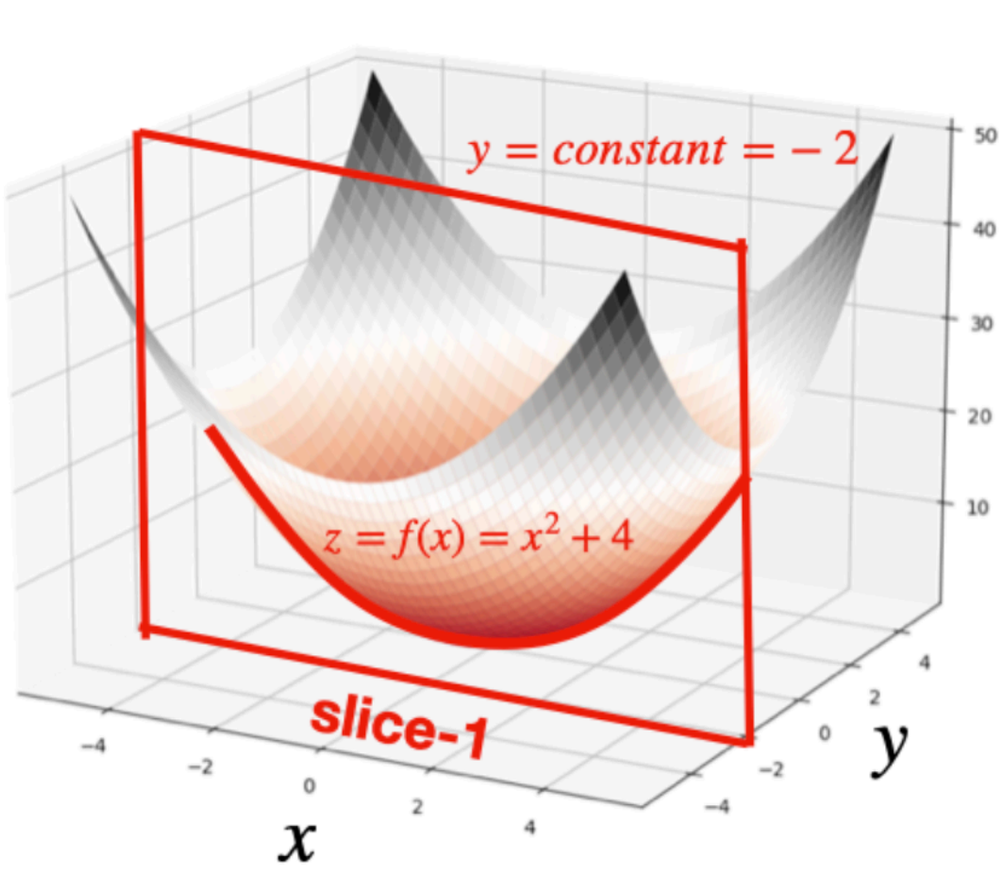


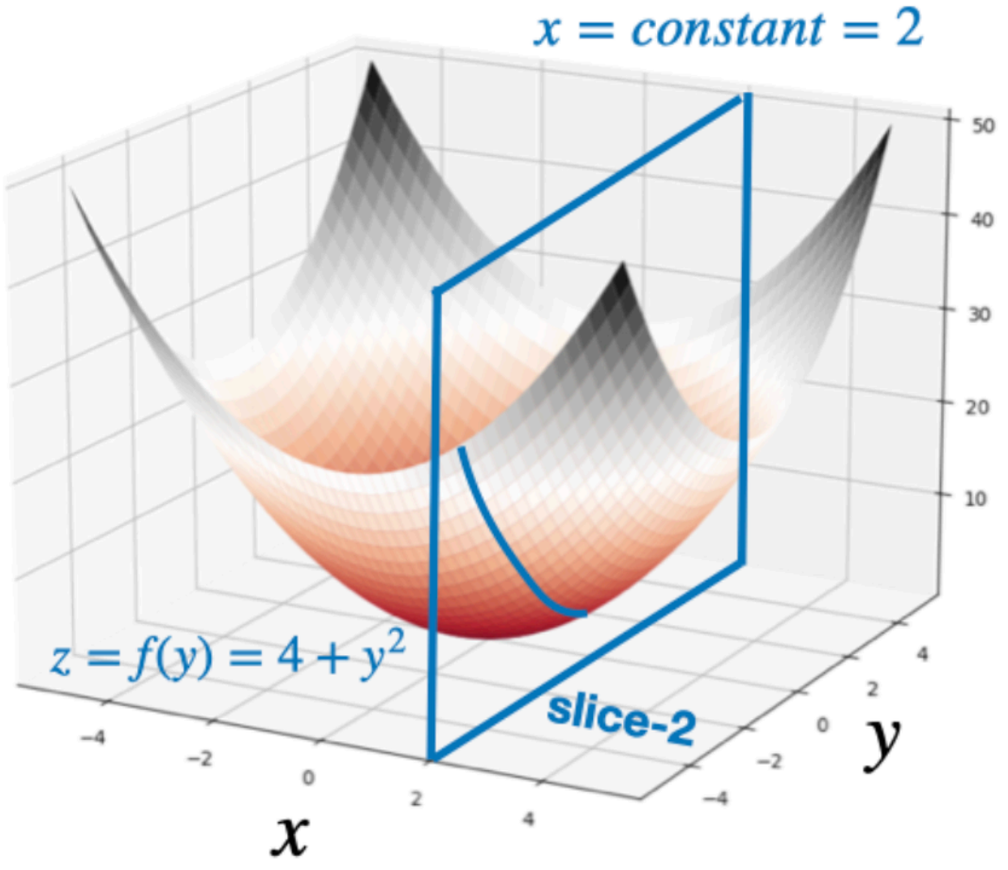


Real world deep learning problems require multivariable calculus

Example: Partial differentiation

$z = f(x, y) = x^2 + y^2$ **consider point:** $(x, y) = (2, -2)$





Partial derivative: x
hold y as constant and take derivative with respect to x

$$f_x(x, -2) = \frac{\partial f}{\partial x} = \frac{\partial}{\partial x} f(x, -2)$$

$$= \frac{\partial}{\partial x}\left(x^2 + 4\right) = 2x + 0 = 2x$$

Partial derivative: y
hold x as constant and take derivative with respect to y

$$f_y(2, y) = \frac{\partial f}{\partial y} = \frac{\partial}{\partial y} f(2, y)$$

$$= \frac{\partial}{\partial y}\left(4 + y^2\right) = 0 + 2y = 2y$$

Partial derivatives at x=2 , y=-2

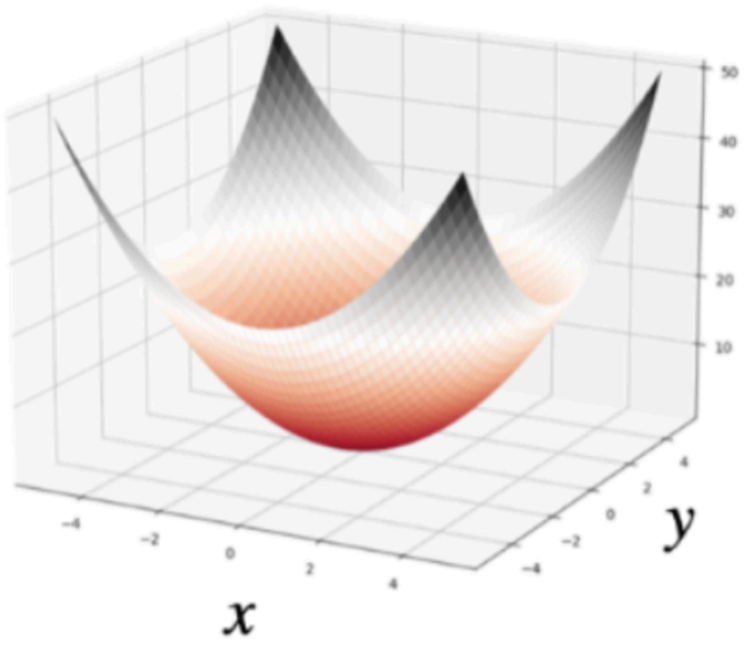
$$f_x(2, -2) = 2x\Big|_{x=2} = 4 \qquad f_y(2, -2) = 2y\Big|_{y=-2} = -4$$

Don't over think it: It's similar to single variable calculus, except for you now have a list of derivatives instead of just one.

The gradient

- The gradient is the vector of partial derivatives $\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$

$$z = f(x, y) = x^2 + y^2$$

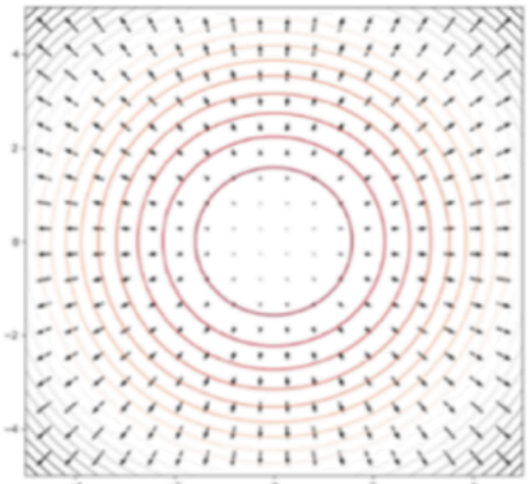


Partial derivatives can be calculated
for any arbitrary x and y points

$$f_x(x, y) = \frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(x^2 + y^2\right) = 2x + 0 = 2x$$

$$f_y(x, y) = \frac{\partial f}{\partial y} = \frac{\partial}{\partial y}\left(x^2 + y^2\right) = 0 + 2y = 2y$$

(Gradient Vector field)



Gradient —> vector of partial derivatives:

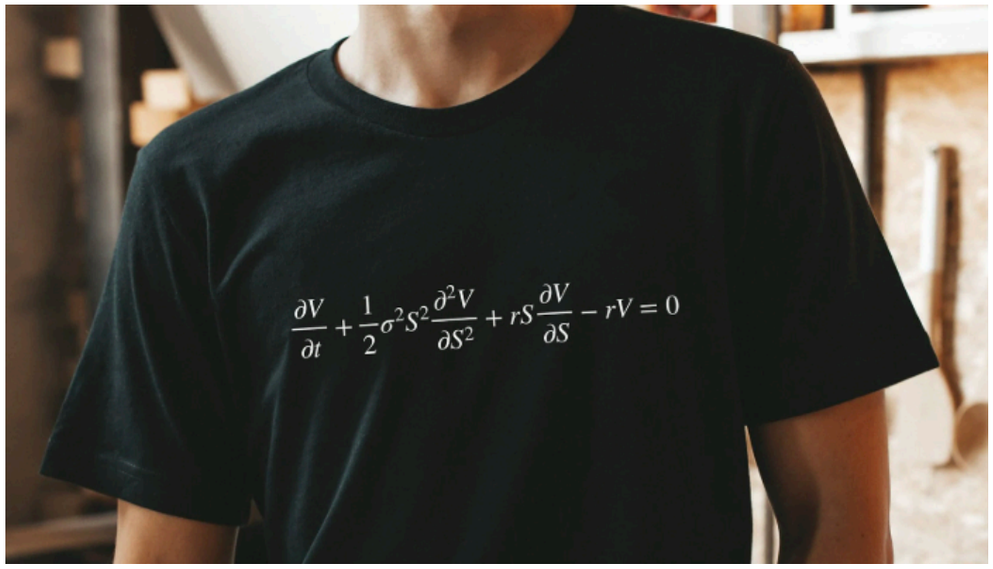
$$\nabla f(x, y) = \left(f_x(x, y), f_y(x, y)\right) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2x, 2y)$$

- **The gradient vector around a point $\mathbf{x}_0$ points in the direction of maximum increase of the function. This makes it incredibly useful for optimization**

Don't over think it: It's similar to single variable calculus, except for you now have a list of derivatives instead of just one.

For deep learning, just think of the gradient vector as a "magic compass" that always points us uphill, which allows us to find the top (or bottom) of the hill.

Aside: Jarreau's Shirt



Fortunately we won't be dealing with partial differential equations today!

The Black–Scholes Equation

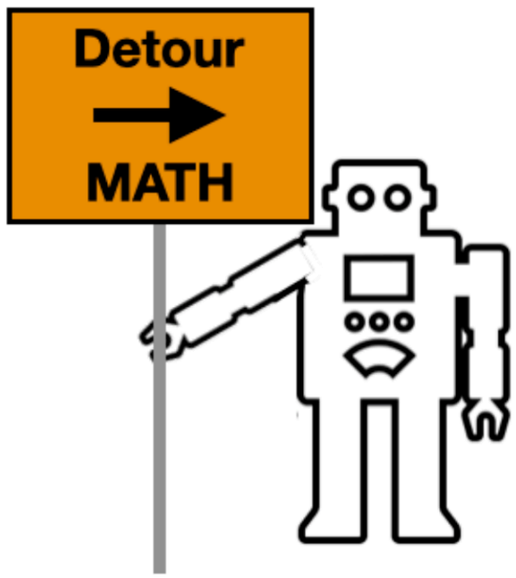
Source: Claude

Math </>

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{1}$$

- $V(S,t)$ **— the unknown scalar field.** Value of the derivative (e.g. a call option) as a function of the underlying price S and time t. Defined on the domain $S \in (0,\infty)$, $t \in [0,T]$.
- S **— the spatial coordinate.** Price of the underlying asset. Not a field, but the independent variable playing the role of "position."
- t **— the time coordinate.** Runs from now ($t=0$) to expiry ($t=T$).
- σ **— volatility.** A scalar *parameter* (constant in vanilla B–S): the diffusion strength of the underlying's log-price.
- r **— risk-free rate.** A scalar parameter: the drift/discount rate.
- $\partial V/\partial S$ **(delta) and** $\partial^2 V/\partial S^2$ **(gamma).** The first and second derivatives of the field w.r.t. S — the "gradient" and "curvature" that hedging exploits.

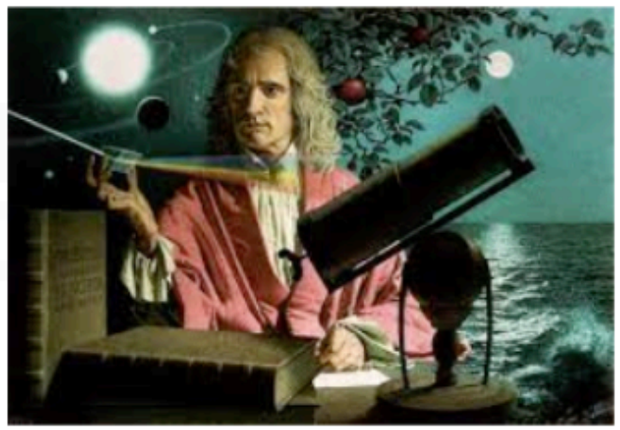
Numerical optimization = Training = learning



Mathematical paradigms

- Typically there are two options when it comes to doing mathematics
- **Analytical methods**: In this case means “by hand”, typically algebra or calculus



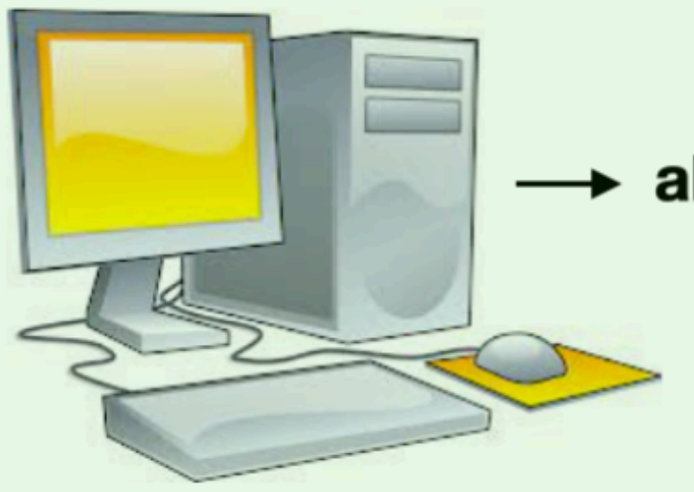


→ **elegant, intuitive, compact, efficent**

- **Numerical methods**: In this case means “by computer”, typically iterative “recipes”

Algorithms and computers



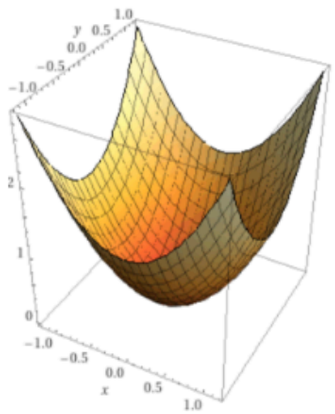


→ **allows us to attack more complex problems**

In deep learning we almost always use this option.

Example: Analytic (pen and paper) optimization problem

- Find the value of $\mathbf{x}_0$ that minimizes $f = f(\mathbf{x}) = f(x, y) = 5 + (x - 10)^2 + y^2$.

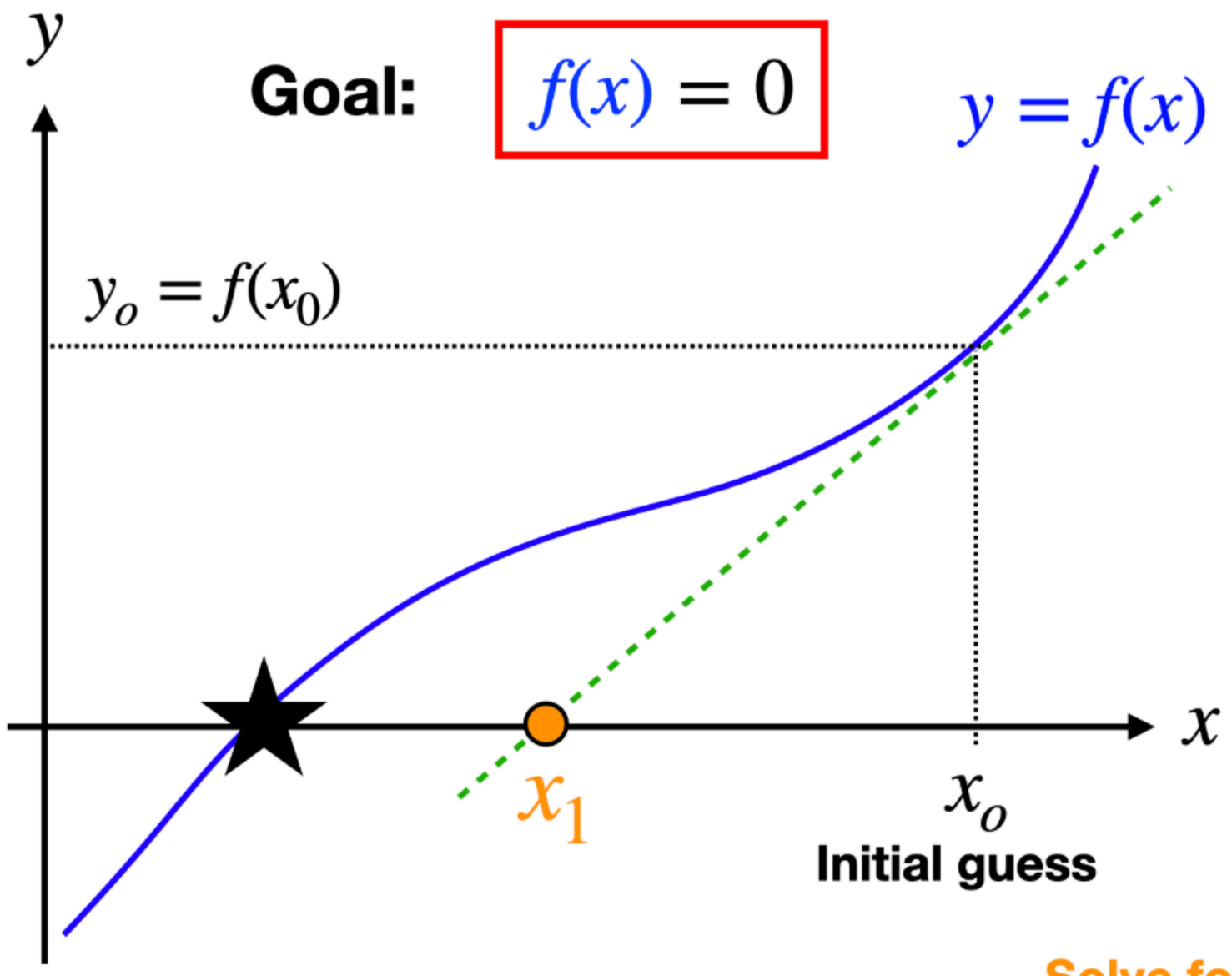


- This is the functional form of the loss function for single variable linear regression.
- It's such a simple function, we can infer the solution $\mathbf{x}_0 = (x_0, y_0) = (10, 0) \rightarrow f = 5$
- However, It can easily be proven using the condition $\nabla f(\mathbf{x}) = \mathbf{0} = (0, 0)$
  - $0 = \frac{\partial f}{\partial x} = 2(x - 10) \rightarrow x = 10$
  - $0 = \frac{\partial f}{\partial y} = 2y \rightarrow y = 0$

- **This is a simple example, but the process is general**
  - (1): Compute the gradient and set it equal to zero
  - (2): Results in a system of N-equation and N-unknowns: $f_x(\mathbf{x}) = 0,\ f_y(\mathbf{x}) = 0 \ldots$
  - (3): `Solve` this system of equations to find the $\mathbf{x}_0$ (roots) which satisfies it

Newton's method (For solving equations)

- Next, we follow the tangent line "down" to obtain an approximation of the root



Point-slope

$$m = \frac{rise}{run} = \frac{\Delta y}{\Delta x} = \frac{y - y_0}{x - x_0}$$

$$(y - y_0) = m(x - x_0)$$

Newton's method

initial guess $\rightarrow x_o$

Tangent line

$$(y - y_0) = m(x - x_0)$$

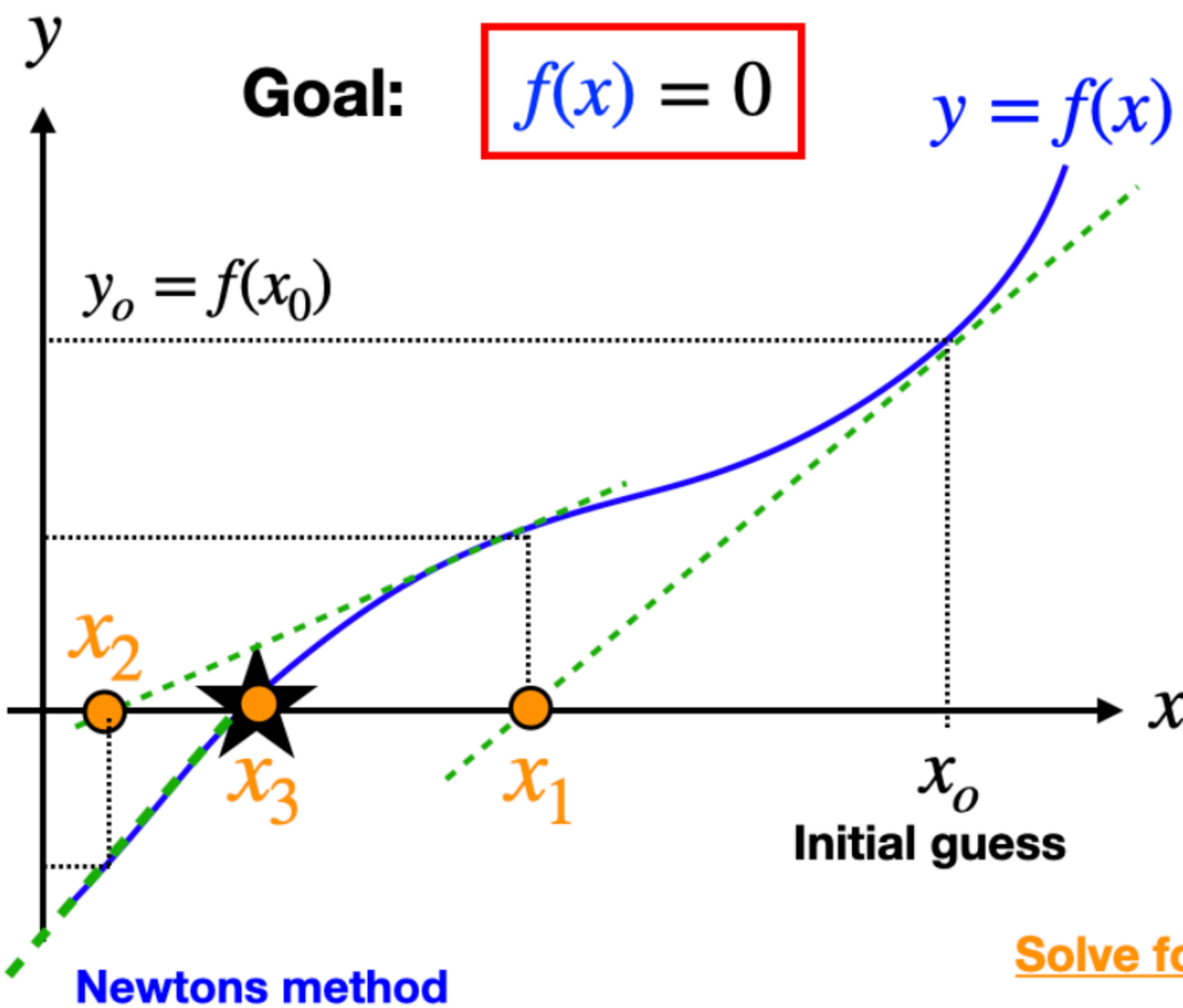
$$f'(x_0) = \left.\frac{\partial f}{\partial x}\right|_{x_o}$$

Solve for y=0

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Newton's method (For solving equations)

- By repeating the process we eventually converge to the solution



Point-slope

$$m = \frac{rise}{run} = \frac{\Delta y}{\Delta x} = \frac{y - y_0}{x - x_0}$$

$$(y - y_0) = m(x - x_0)$$

Newton's method

initial guess $\rightarrow x_o$

Tangent line

$$(y - y_0) = m(x - x_0)$$

$$f'(x_0) = \left.\frac{\partial f}{\partial x}\right|_{x_o}$$

Solve for y=0

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Repeat until converged

Newtons method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Requires explicit functional form of the derivative of f(x)

Gradient descent for optimization

- If the function only has a single-variable. Then the derivative is a function (not a vector of partial derivatives)

$$\text{Newton's method for solving f(x)=0: } x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

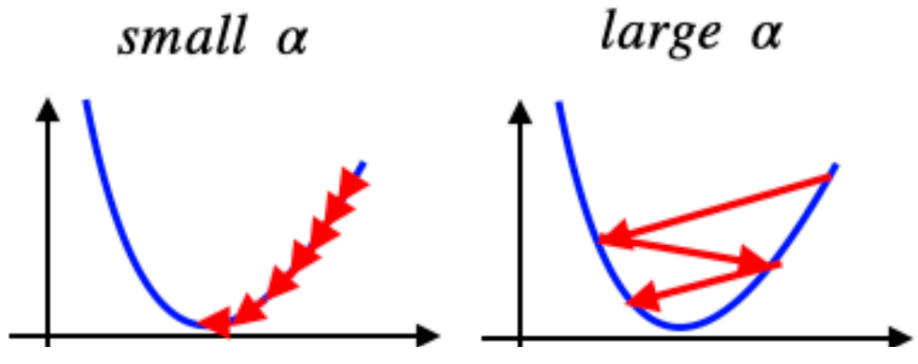
- In this case, optimization means finding the roots of the the derivative. We can just apply Newton's method to the derivative. We just get another differentiation in the formula

$$\text{Newton's method optimization f '(x)=0: } x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

- This is done by replacing the second derivative, with as hyper-parameter α, known as the learning rate (this simplification is known as gradient descent)

$$\text{Gradient descent optimization: f '(x)=0 } x_{n+1} = x_n - \alpha\, f'(x_n)$$

- Tuning α allows us to control how big our "steps" and therefore convergence rates.



Gradient Descent in Many Dimensions

The idea: you're standing on a hillside in fog. Feel which way is *steepest downhill*. Take a small step. Repeat.

The **gradient** $\nabla_{\mathbf{w}} L$ is the arrow pointing *steepest uphill*, so we walk the other way.

Vector form

new value = old value – step

$$\mathbf{w}_{\text{next}} \leftarrow \mathbf{w}_{\text{current}} - \eta \nabla_{\mathbf{w}} L$$

- $\nabla_{\mathbf{w}} L = (\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots)$
- $\mathbf{w}$ — all the weights, stacked into one vector (your position on the landscape)
- $L(\mathbf{w})$ — the loss (your altitude — we want to get *low*)
- η — the learning rate (how big a step you take)
- The **minus sign** is the whole trick: uphill is where the gradient points, so subtract it.

Index form

new = old – step

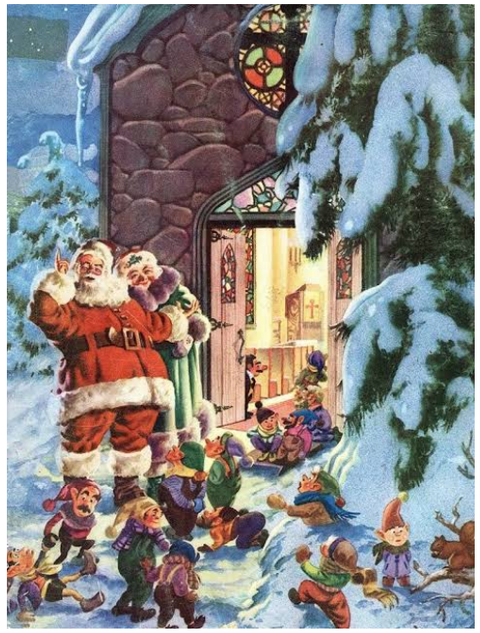
$$w_i^t \leftarrow w_i^t - \eta \frac{\partial L}{\partial w_i^t} \qquad \text{for every } i \text{ at iteration t}$$

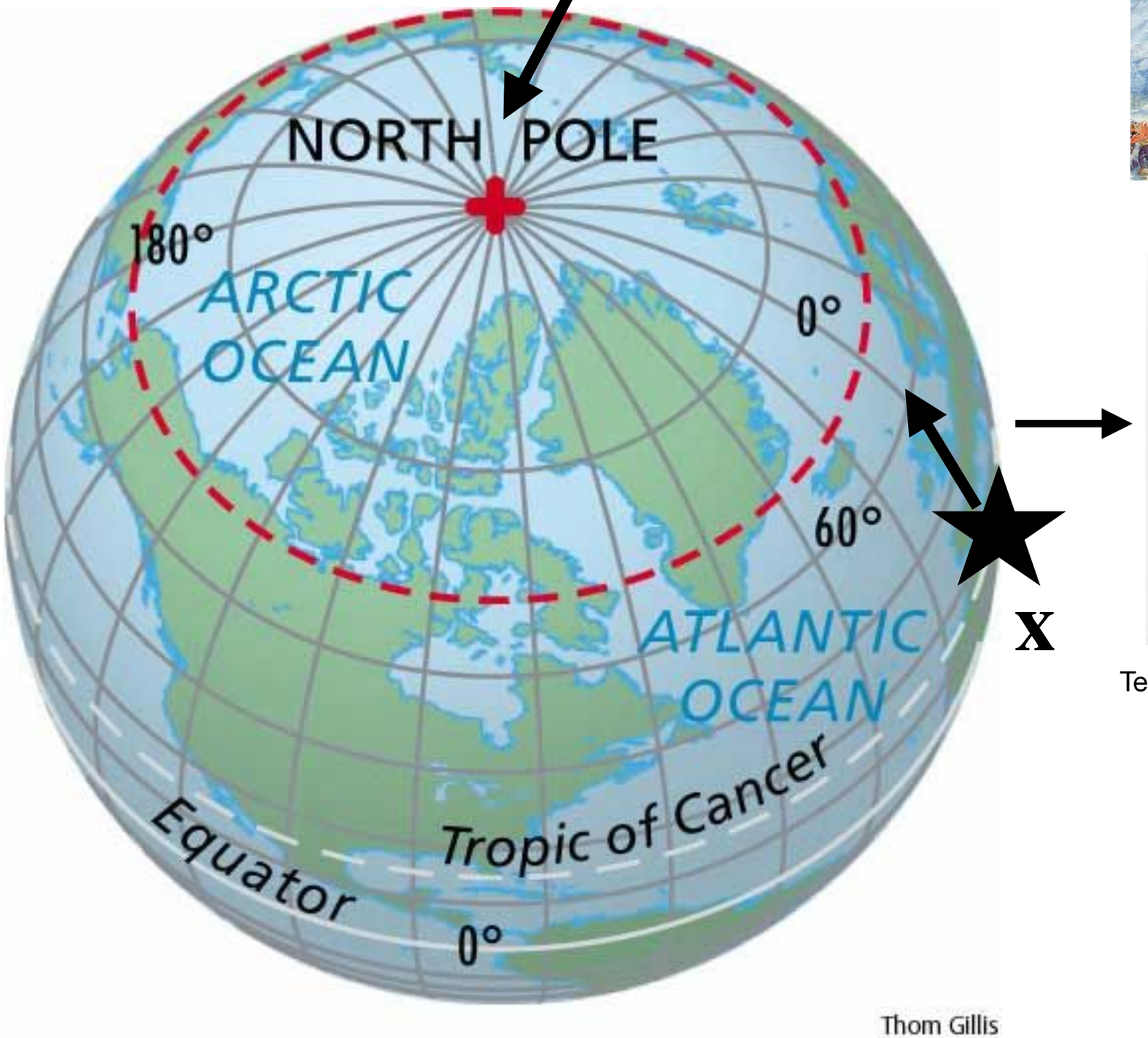
Each weight w_i is nudged on its own, using the partial derivative that answers:

Switching notation: x —> w and f —> L

Helpful Analogy

Goal: Get here



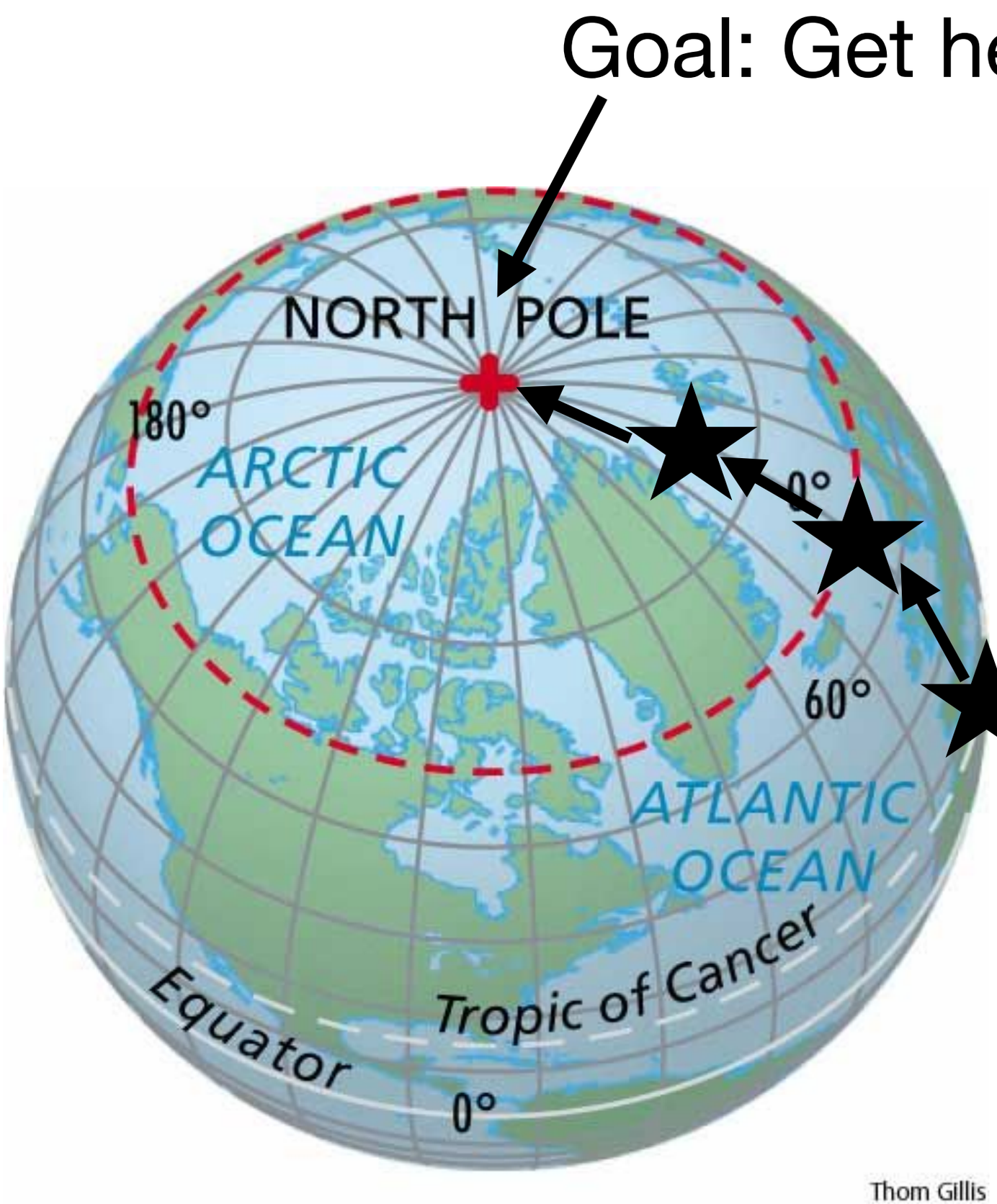


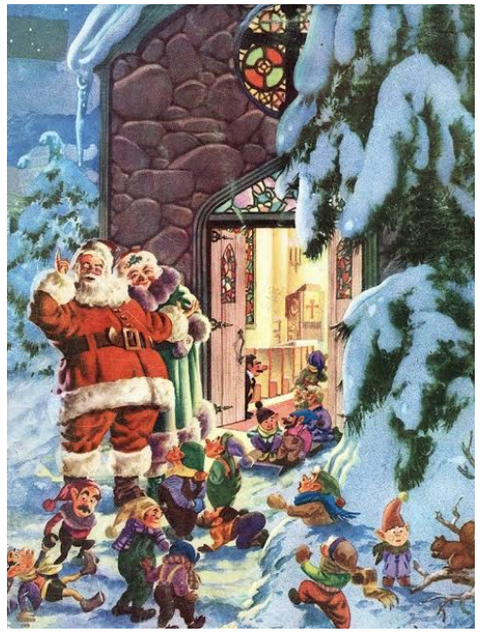
X



Tells you where north is from ANYWHERE like magic

Iterate: measure compass, get vector, step, repeat

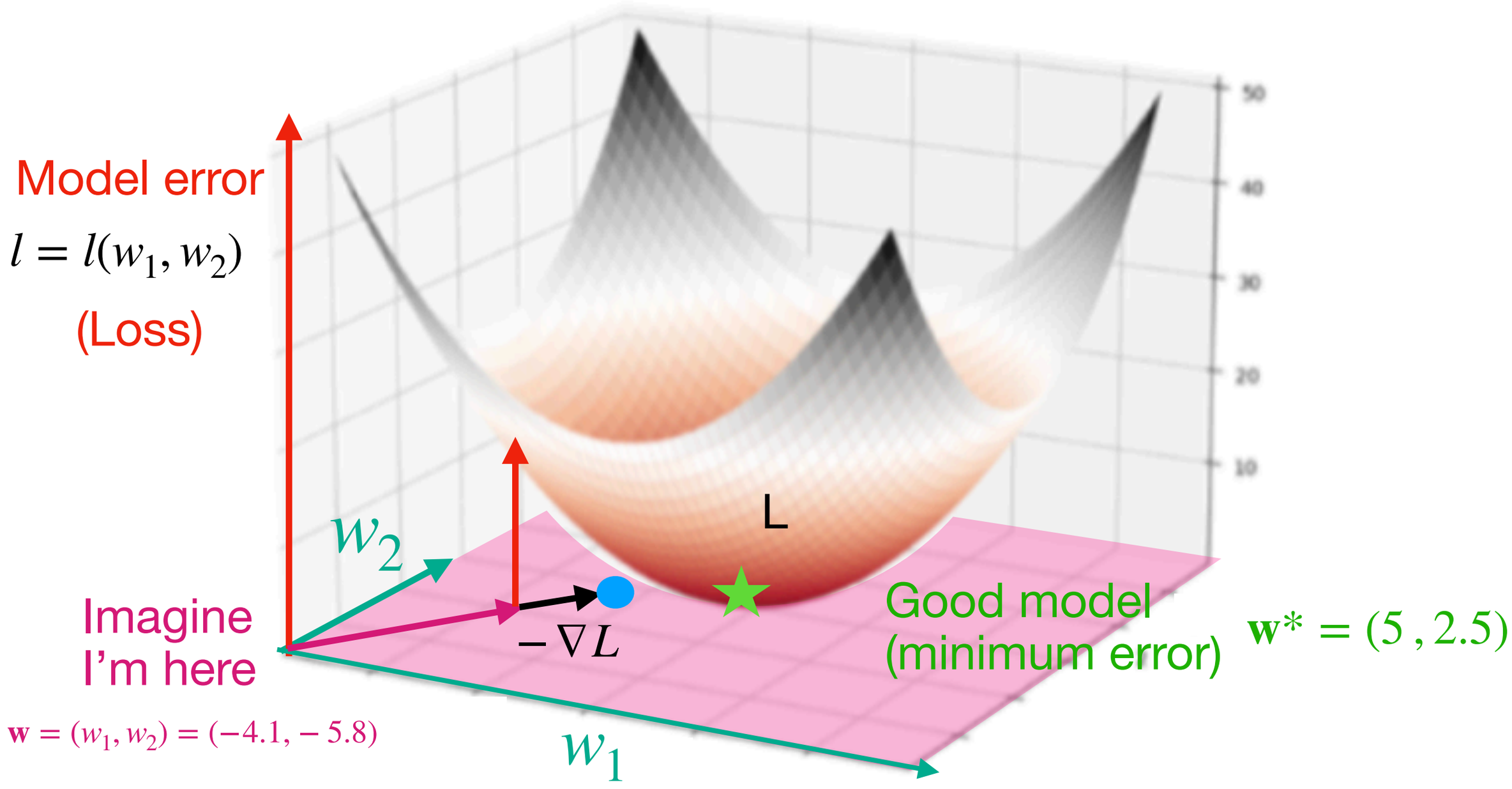






Tells you where north is from ANYWHERE like magic

Gradient Descent: Geometric construction



Computer calculates gradient vector for us via back-propagation (don't ask)

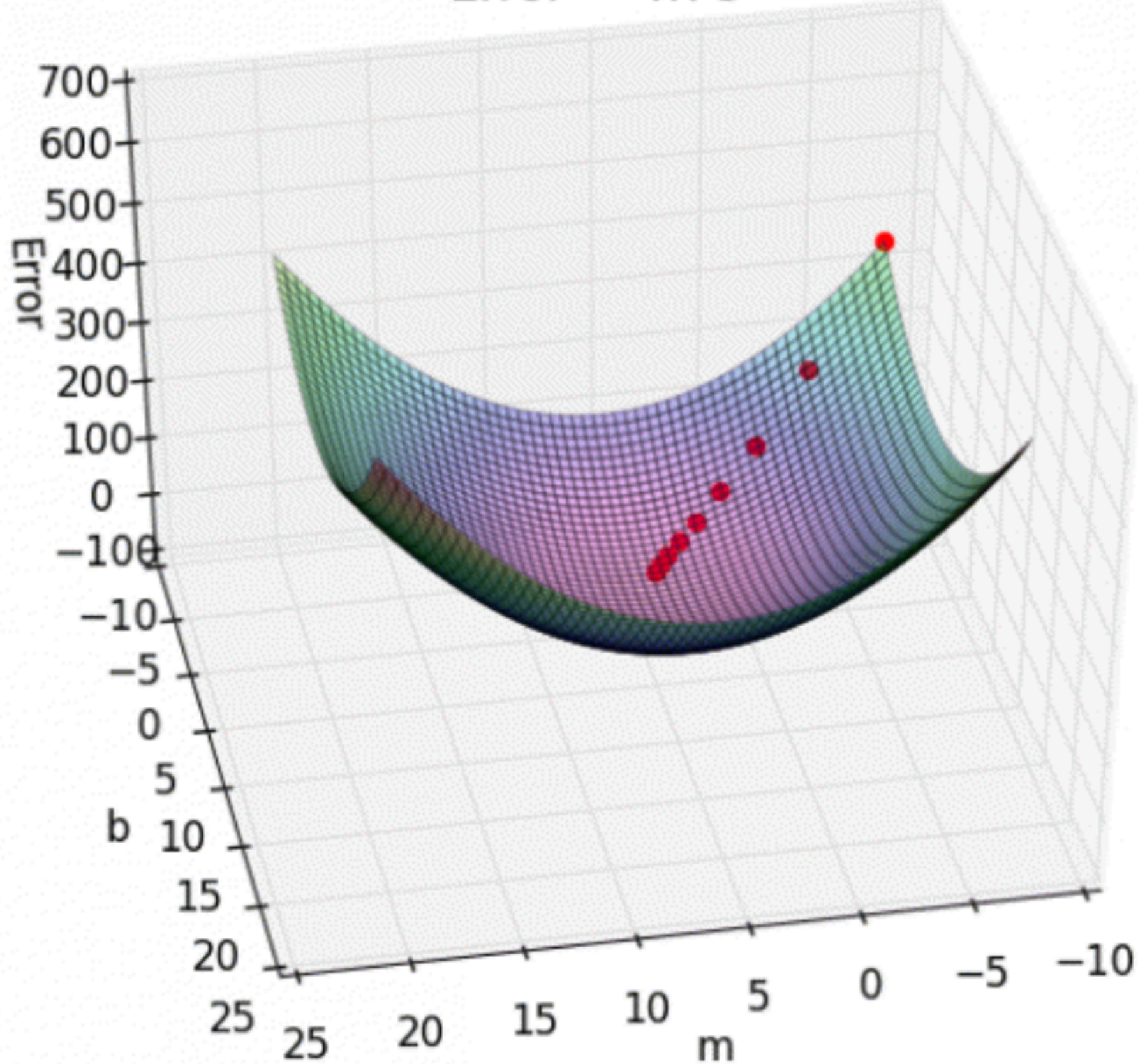
$$-\nabla L(\mathbf{w}) = -(-0.5, -0.5) = (0.5, 0.5)$$



new value = old value − step

$$\mathbf{w}_{\text{next}} \leftarrow \mathbf{w}_{\text{current}} - \eta \nabla_{\mathbf{w}} L$$

Iter

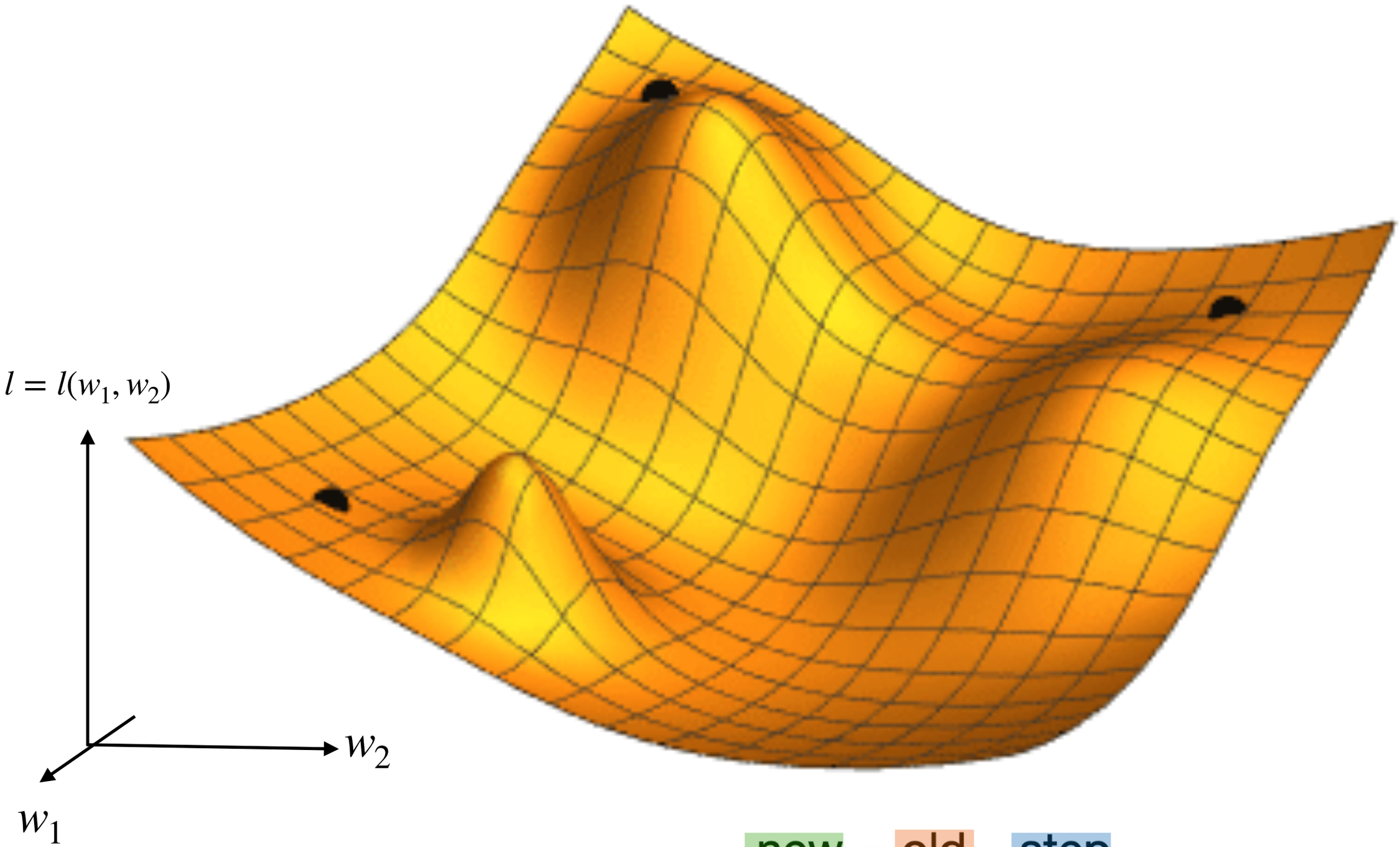


new = old − step

$$w_i^t \leftarrow w_i^t - \eta \frac{\partial L}{\partial w_i^t} \qquad \text{for every } i \text{ at iteration t}$$

Visualization of gradient descent

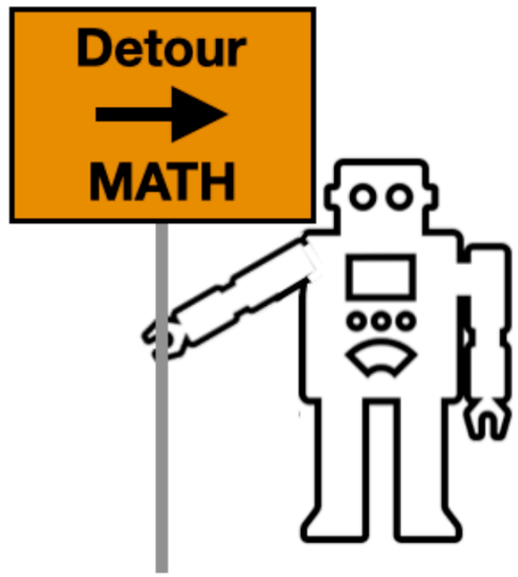
Starting from 3 different initial conditions and iterating



$$\text{new} = \text{old} - \text{step}$$

$$w_i^t \leftarrow w_i^t - \eta \frac{\partial L}{\partial w_i^t} \qquad \text{for every } i \text{ at iteration t}$$

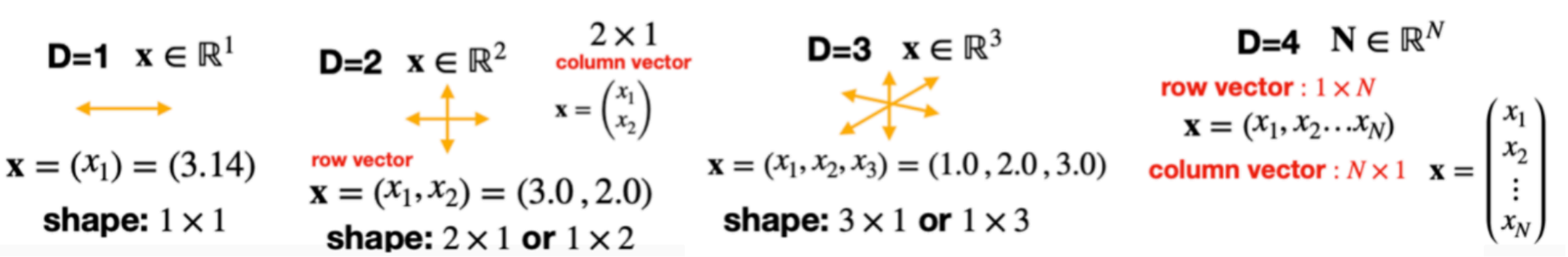
10 Minute Linear Algebra Refresher



Like it or not, the chosen data structures for deep learning are vectors, matrices, and tensors

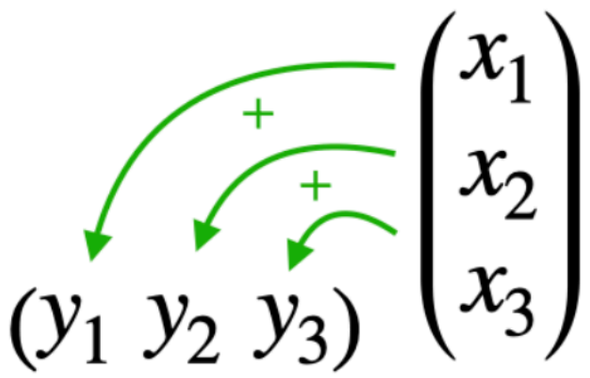
Vectors

- Vectors are one dimensional lists of scalars (requires one index to specify a site)
  - **Column vector:** $\mathbf{v} = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_N \end{pmatrix} \rightarrow$ shape=$[N \times 1]$
  - **Row vector:** $\mathbf{v} = (v_0, v_0 \ldots v_N) \rightarrow$ shape=$[1 \times N]$
- An important attribute of a matrix (and vectors) is shape, denoted by $N_{rows} \times N_{columns}$
- The Transpose $\mathbf{v}^T$ reverses a matrix's shape and switches the rows into columns
- Vectors can also be transposed: A column vector is the transpose of a row vector



Dot product

- The dot product is one of the most important operations associated with vectors
- The operation takes two vectors and converts them into a scalar.



multiply like components and add

$$= y_1x_1 + y_2x_2 + y_3x_3$$

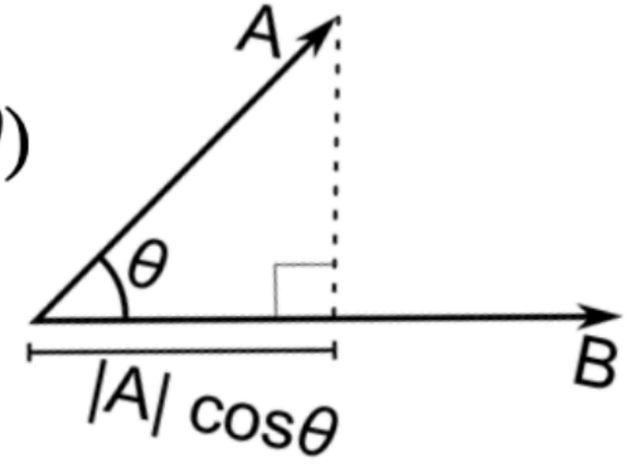
row vector times column vector

$$\mathbf{y}^T\mathbf{x} = \mathbf{y} \cdot \mathbf{x}$$

$$1 \times 3 \quad 3 \times 1 \rightarrow 1 \times 1$$

- The dot product can be used to determine the amount of $\mathbf{A}$ which is parallel to $\mathbf{B}$
- Therefore, it is useful as a similarity metric.

$$\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}||\mathbf{B}|cos(\theta)$$



othogonality

$$\mathbf{y} \cdot \mathbf{x} = 0$$

$$cos(90^o) = 0$$

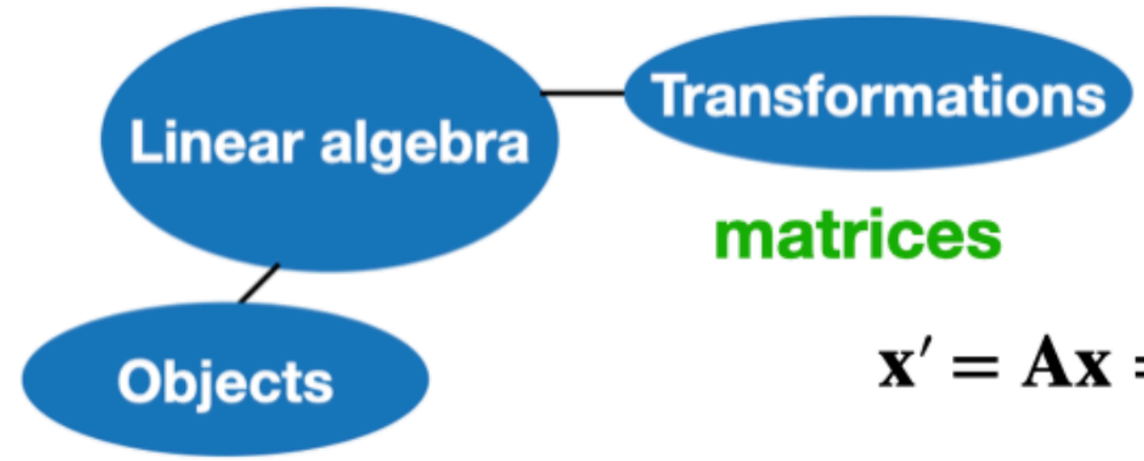
$$cos(0^o) = 1$$

Cosine similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Spaces and transformations

- A vector space is a collection of vectors closed under addition and scalar multiplication
- Linear algebra explores how vectors change via linear transformations (matrices).



SQUARE MATRIX = LINEAR TRANSFORMATION

$$\mathbf{x} \rightarrow \mathbf{x}' = \mathbf{A}\mathbf{x}$$

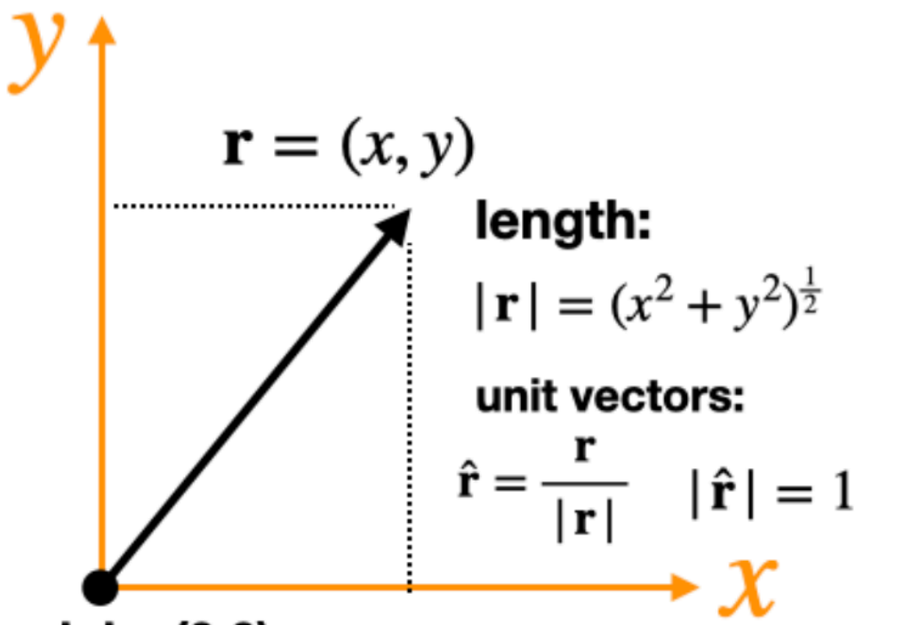
matrices

$$\mathbf{x}' = \mathbf{A}\mathbf{x} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{pmatrix}$$

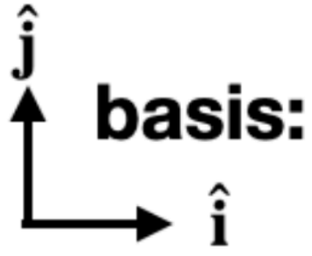
$2 \times 2 \quad 2 \times 1 \quad 2 \times 1$

vectors ∈ vector space

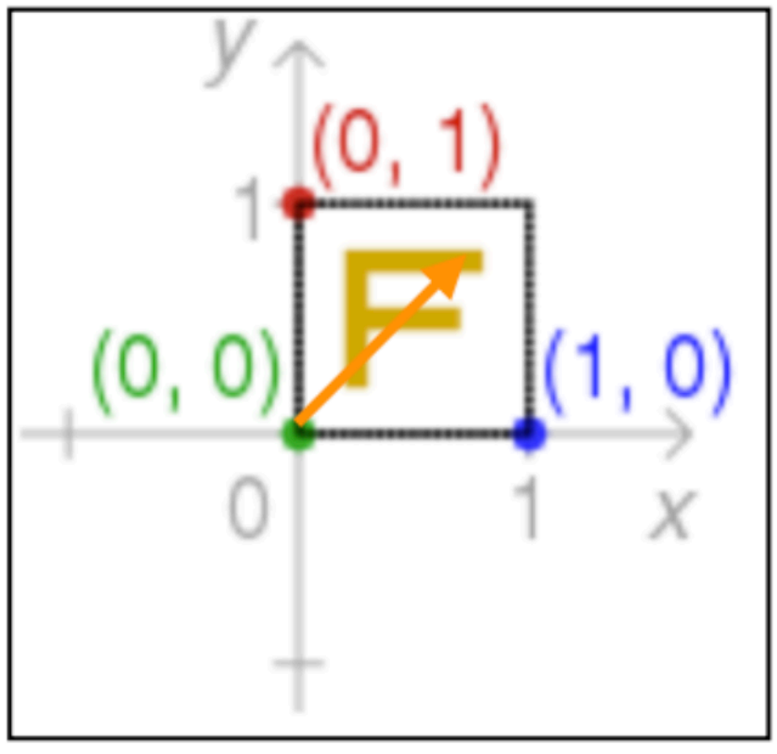
Example: 2D cartesian space



$$\mathbf{r} = (x, y) = x\hat{\mathbf{i}} + y\hat{\mathbf{j}}$$



Each pixel has a position vector



Transformations

Linear transformations are constructed from basic geometric actions like rotation, scaling, contraction, skewing, and projection.

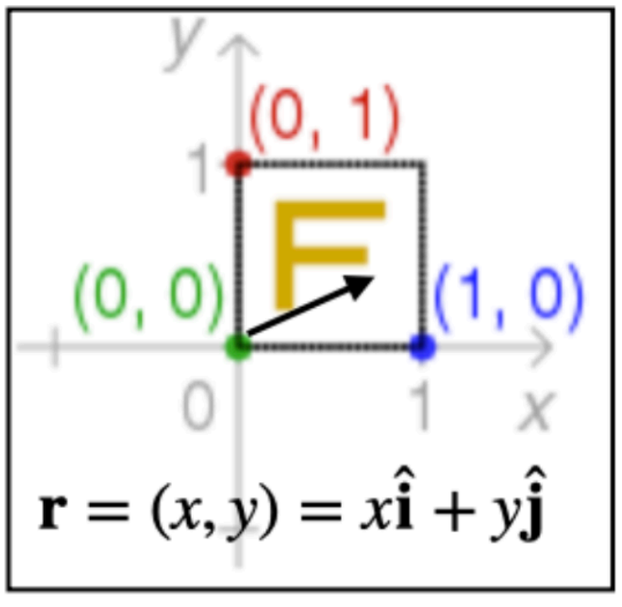
$$\mathbf{x} \rightarrow \mathbf{x}' = \mathbf{A}\mathbf{x}$$

Transformations

matrices

$$\mathbf{x}' = \mathbf{A}\mathbf{x} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{pmatrix}$$

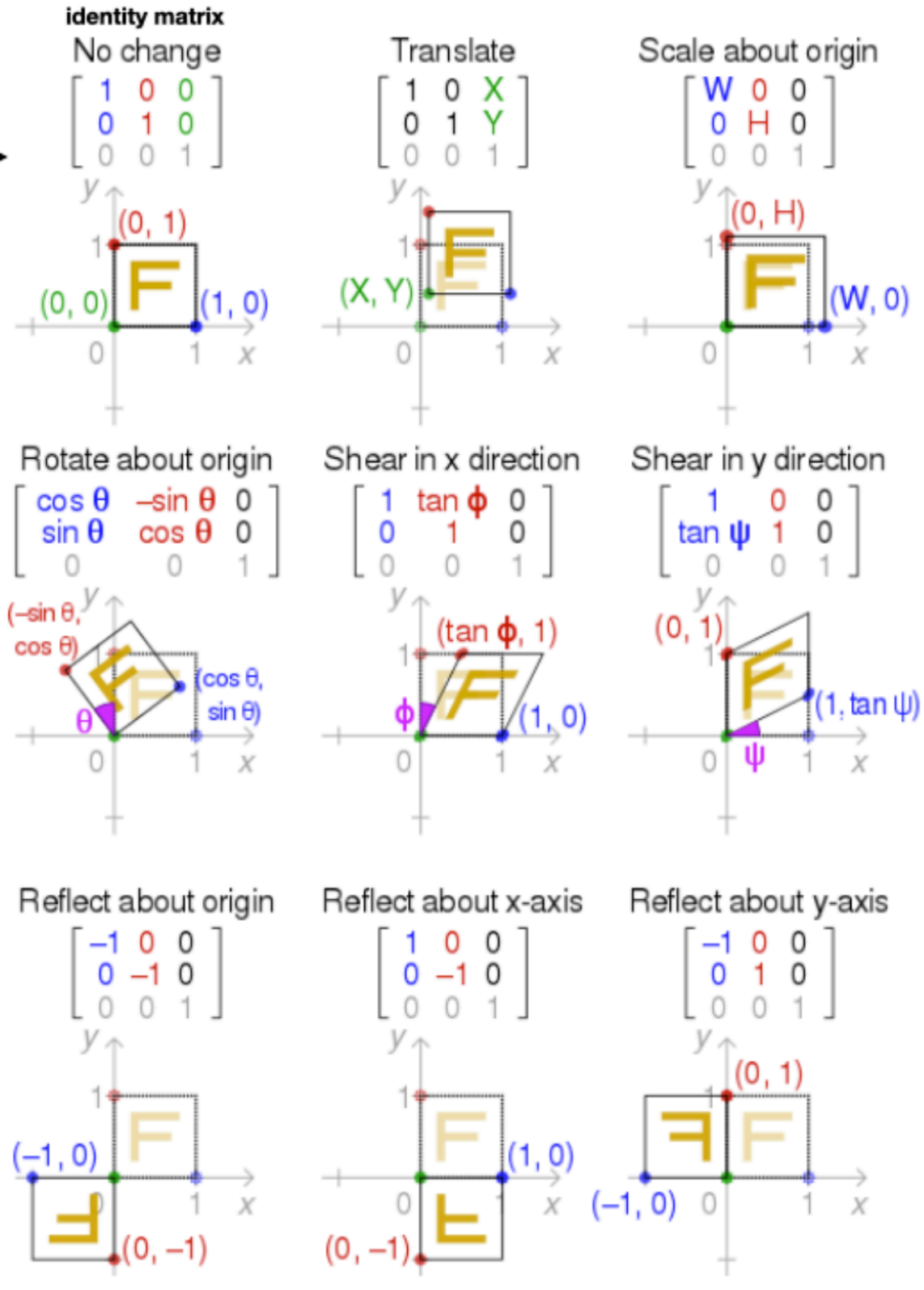
Original image



Inverse tranformation:

$$\mathbf{x}' \rightarrow \mathbf{x} = \mathbf{A}^{-1}\mathbf{x}' \qquad \mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{1}$$

inverse of A



Matrix shape

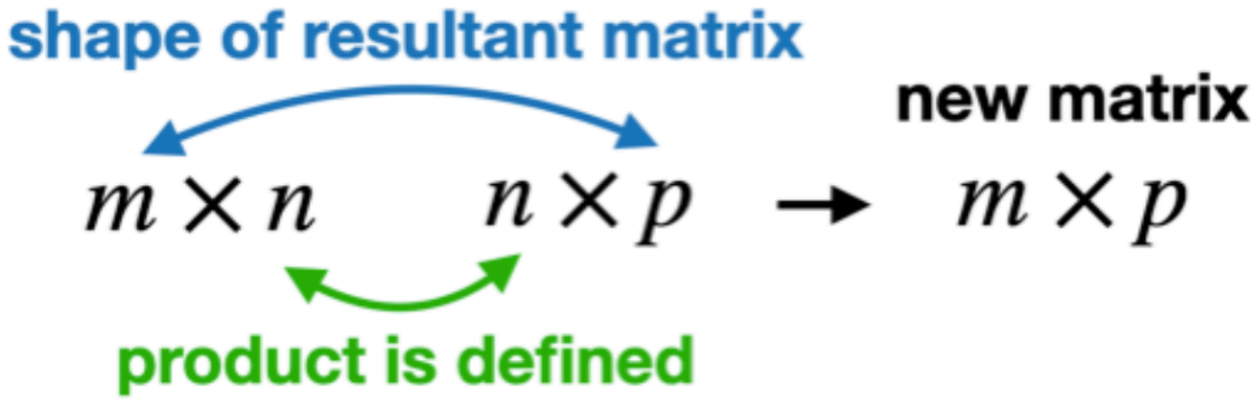
- Transformations on vectors are described by matrices $\mathbf{A}$
- Matrices are two dimensional arrays of scalars & requires two index to specify site
- $\mathbf{w} = \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix} \rightarrow \mathbf{shape}{=}[2 \times 3]$
- $\mathbf{w}^T = \begin{pmatrix} w_{00} & w_{10} \\ w_{01} & w_{11} \\ w_{02} & w_{12} \end{pmatrix} \rightarrow \mathbf{shape}{=}[3 \times 2]$

- A square matrix has the same number of rows as columns:

$$\mathbf{A} = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

Matrix multiplication

- Matrix multiplication is basically repeated dot products: The new entry for row-i column-j is A_row_i DOT B_column_j
- Two matrices can be multiplied if they obey the following rule.



- **Example-1:** (matrix-vector product).

$$\mathbf{Ax} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{pmatrix}$$

$$2 \times 2 \quad 2 \times 1 \qquad 2 \times 1$$

$$\mathbf{x}^T\mathbf{A} = (x_1\ x_2) \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = (x_1a_{11} + x_2a_{21} \quad x_1a_{12} + x_2a_{22})$$

$$1 \times 2 \quad 2 \times 2 \qquad 1 \times 2$$

Example

- **Example-2:** (matrix-matrix product).

$$2\times 3 \quad \mathbf{A} \quad \mathbf{B} \quad 3\times 2$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 10 & 11 \\ 20 & 21 \\ 30 & 31 \end{bmatrix}$$

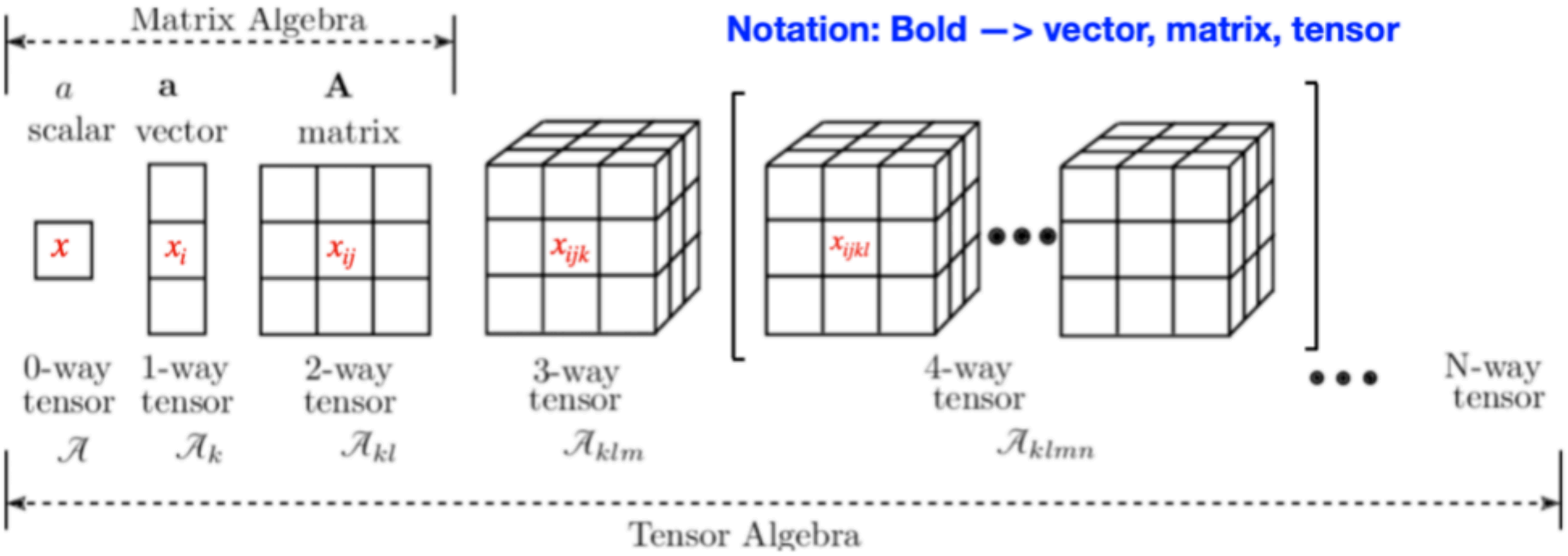
$$= \begin{bmatrix} 1x10 + 2x20 + 3x30 & 1x11 + 2x21 + 3x31 \\ 4x10 + 5x20 + 6x30 & 4x11 + 5x21 + 6x31 \end{bmatrix}$$

$$= \begin{bmatrix} 10+40+90 & 11+42+93 \\ 40+100+180 & 44+105+186 \end{bmatrix} = \begin{bmatrix} 140 & 146 \\ 320 & 335 \end{bmatrix}$$

$$2\times 2$$

Tensors

- `Tensor algebra` extends linear algebra to higher-dimensional data structures called tensors, crucial in physics, engineering, and machine learning
- Allow for multi-dimensional data representation and analysis.
- `tensor rank`=number of indices needed to specify a site

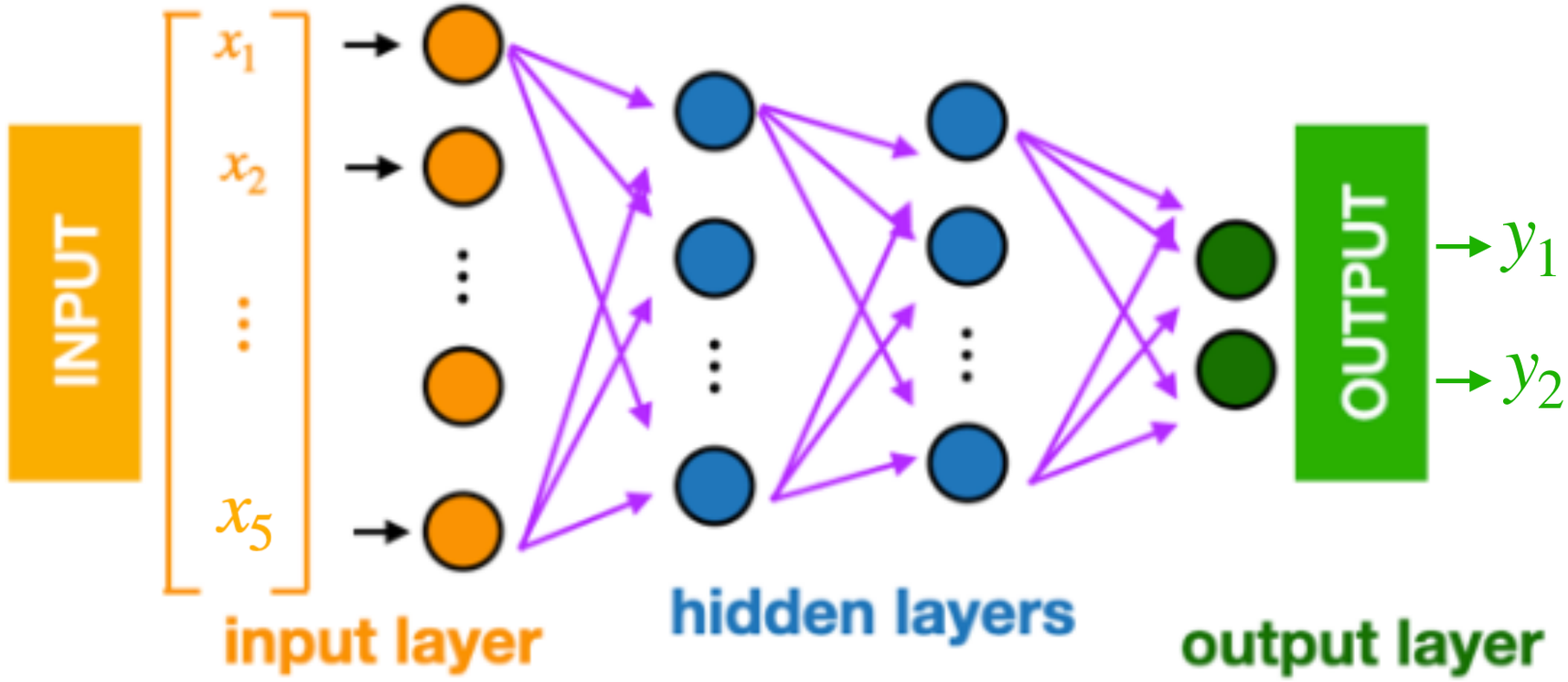


TensorFlow → Very popular deep learning library where tensors "flow" through models and loss functions.

Data:

	continuous		type (one-hot)			target
i	sqft	age	condo	house	town	price
0	1450	12	0	1	0	410.0
1	980	45	1	0	0	265.5
2	2100	3	0	1	0	620.0
3	1320	28	0	0	1	355.2

real-valued binary, exactly one 1 per row what we predict



Examples of tensor data in deep learning

Data and neural models

As we will see deep learning is largely a collection of different model architectures designed for different types of data

Record Data

ID	height (ft)	weight (lb)	adult/child
1	h_1	w_1	A
2	h_2	w_2	C
3	h_3	w_3	C
⋮	⋮	⋮	
N	h_N	w_N	A

Non-linear relationships →

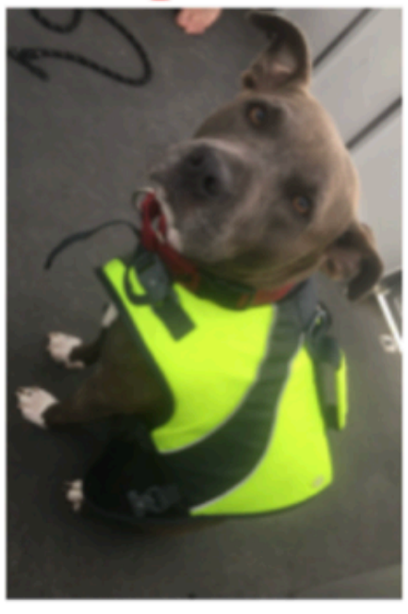
Simple images (e.g. MNIST)

Deep Feed Forward (DFF) (MLP)



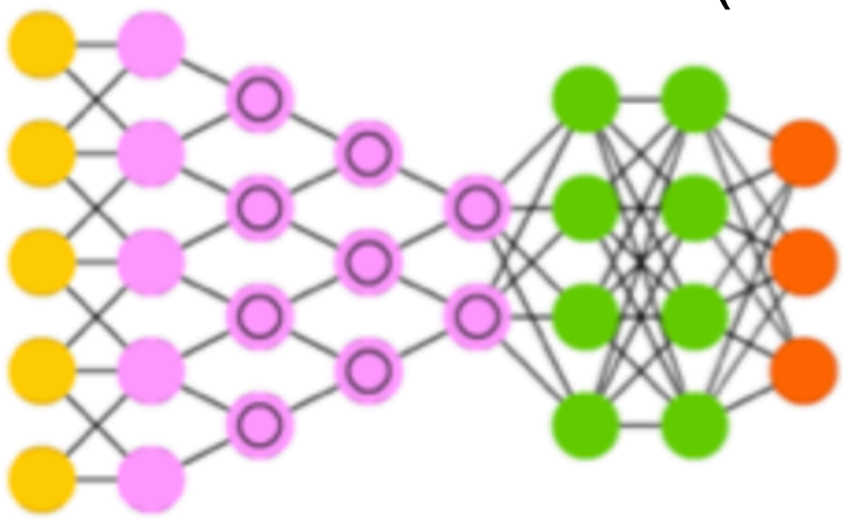
(Focus of today)

Image data



Computer vision →

Deep Convolutional Network (DCN) (CNN)



Your Data Dictates your model choice!!

Data and neural models

As we will see deep learning is largely a collection of model architectures designed for different types of data

Text data



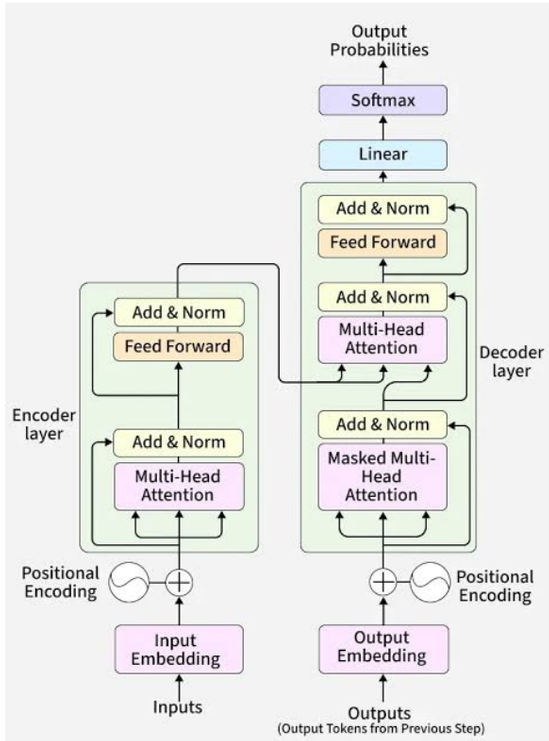
NLP →

(Focus of tomorrow)

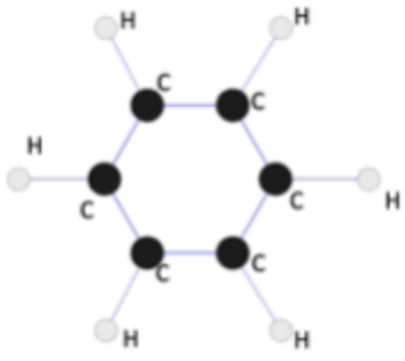
Recurrent neural networks (RNN)



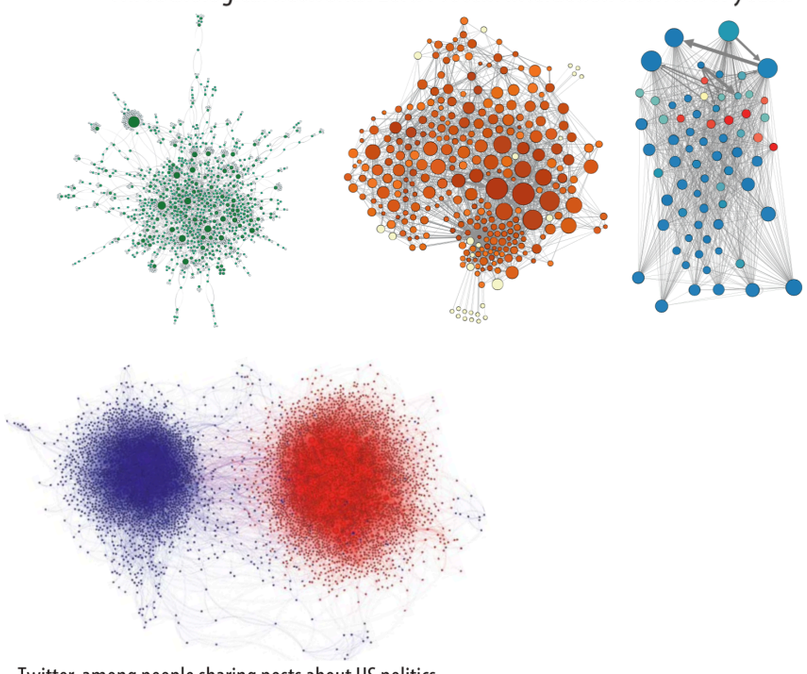
Transformers



Network data



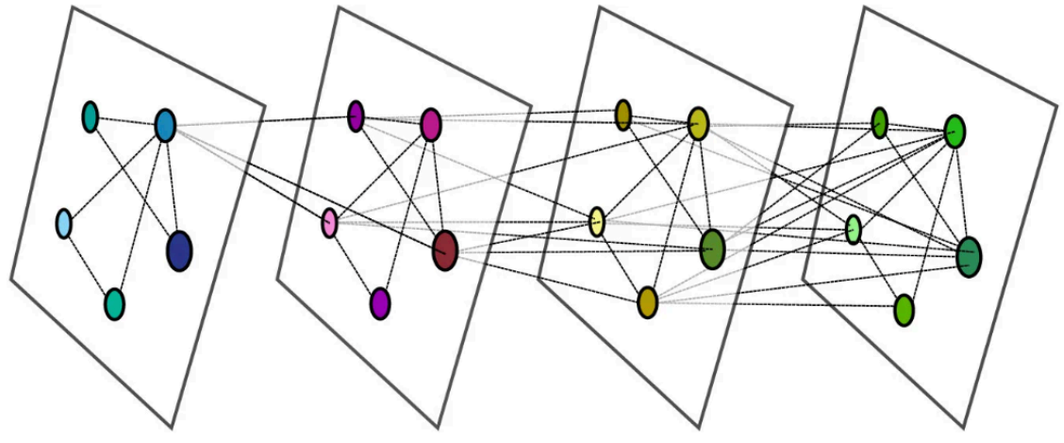
Three biological networks. Left: Protein interaction network of yeast.



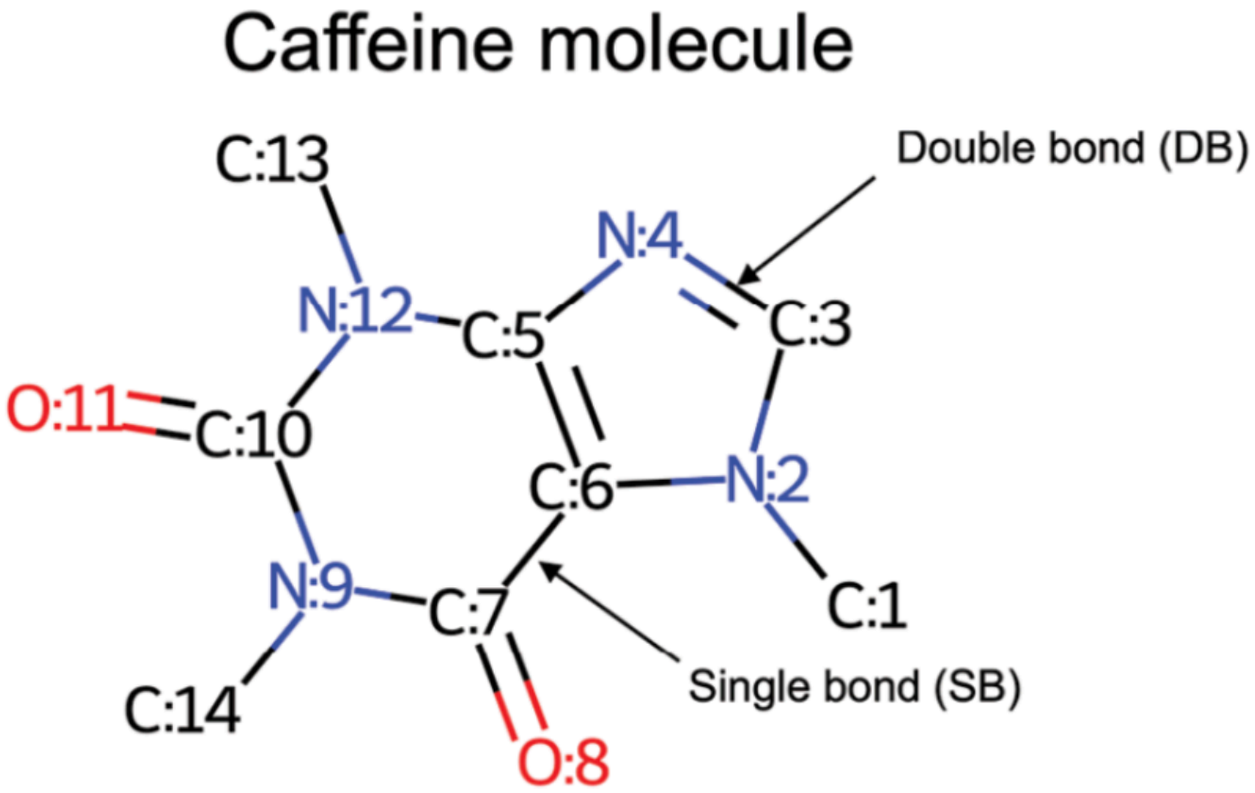
Twitter, among people sharing posts about US politics

→

(Graph Neural networks)(GNN)



Network (Graph) level data



$$A =$$

Adjacency matrix

$$\begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

One-hot encoding of the atom type

$$X =$$

Node label matrix

	C	N	O
	1	0	0
	0	1	0
	1	0	0
	0	1	0
	1	0	0
	1	0	0
	0	0	1
	0	1	0
	1	0	0
	0	0	1
	0	1	0
	1	0	0

One-hot encoding of the bond type

$$X_E =$$

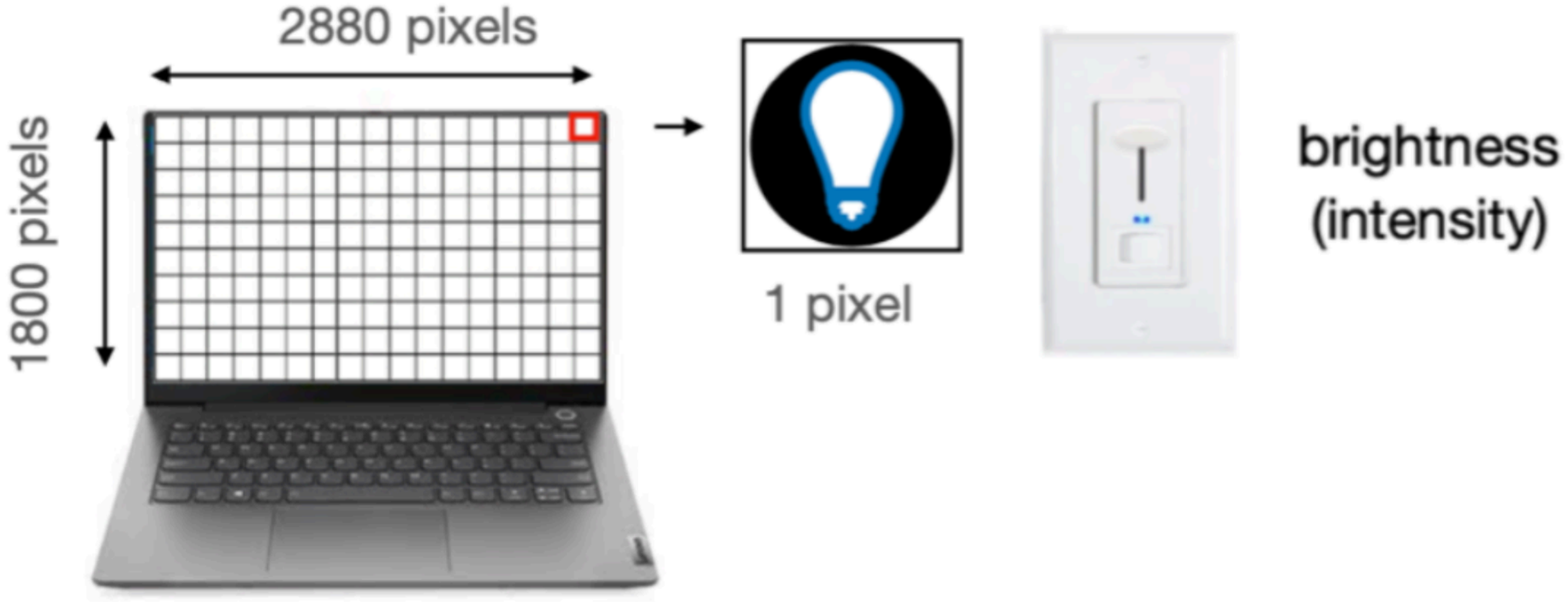
Edge label matrix

	SB	DB
(1,2)	1	0
(2,3)	1	0
(2,6)	1	0
(3,4)	0	1
(4,5)	1	0
(5,6)	0	1
(5,12)	1	0
(6,7)	1	0
(7,8)	0	1
(7,9)	1	0
(9,10)	1	0
(9,14)	1	0
(10,11)	0	1
(10,12)	1	0
(12,13)	1	0

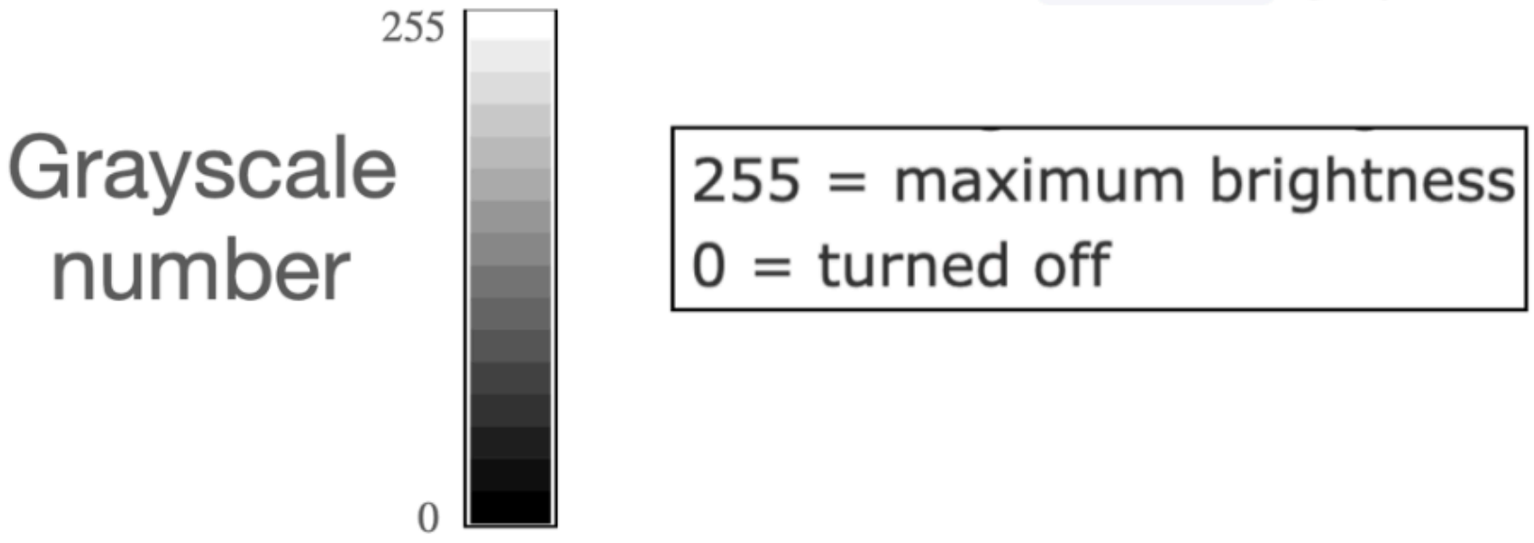
Image data

Black and white screens

- Some basic concepts about screens can be useful for understanding computer vision
- In a black and white screen, pixels are variable light bulbs outputting *white (mixed) light*

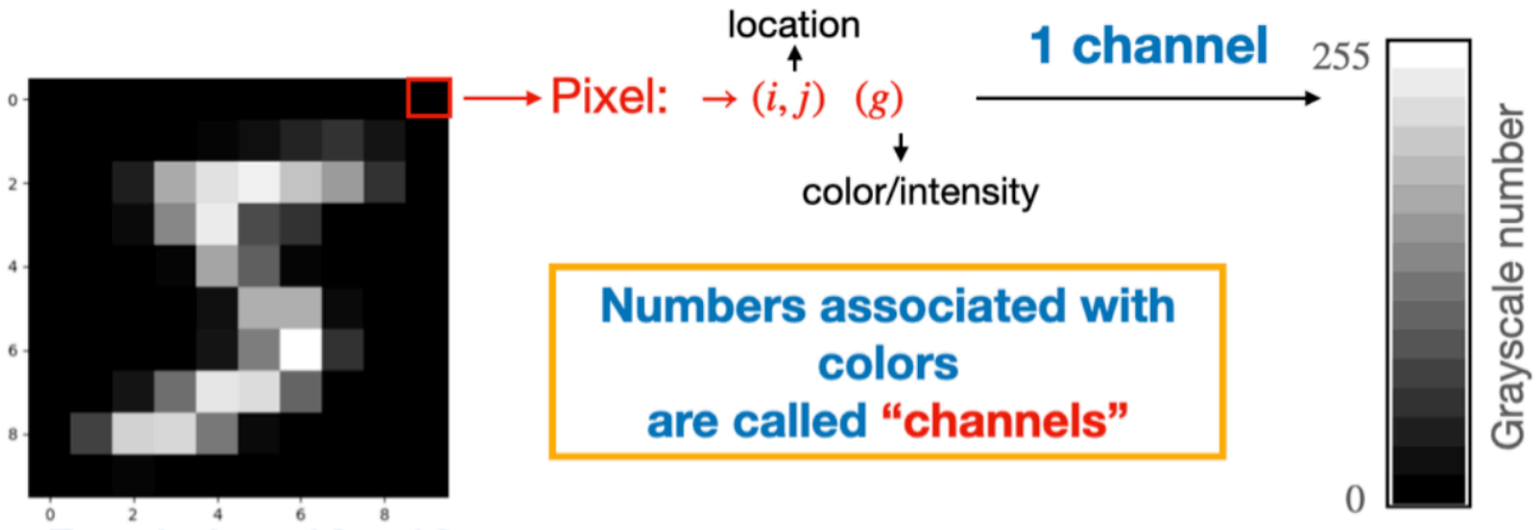


- The intensity of a pixel's light is control by the *gray-scale* number.
- Numbers associated with colors are called `channels`, gray-scale has one channel



Grayscale images

- Black and white images are stored by specifying the gray-scale number for all the pixels
- The pixel location is determined by its location in the matrix.



- `Note`: The gray-scale image can also be expressed as a rank-3 tensor instead of a matrix
  - `[height, width]` → `[height, width, channels]`, e.g. `[10, 10]` → `[10, 10, 1]`

MNIST data-set

- The full MNIST data set has 60,000 images, well balanced across the classes



```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Training Inputs:

```
------------------------
INFO
------------------------
SHAPE: (60000, 28, 28)
MIN: 0
MAX: 255
TYPE: <class 'numpy.ndarray'>
DTYPE: uint8
```

Training Outputs (labels):

```
------------------------
INFO
------------------------
SHAPE: (60000,)
MIN: 0
MAX: 9
TYPE: <class 'numpy.ndarray'>
DTYPE: uint8
```

Test Inputs:

```
------------------------
INFO
------------------------
SHAPE: (10000, 28, 28)
MIN: 0
MAX: 255
TYPE: <class 'numpy.ndarray'>
DTYPE: uint8
```

Test labels:

```
------------------------
INFO
------------------------
SHAPE: (10000,)
MIN: 0
MAX: 9
TYPE: <class 'numpy.ndarray'>
DTYPE: uint8
```

Source and recommended reading: Chollet p27-p31



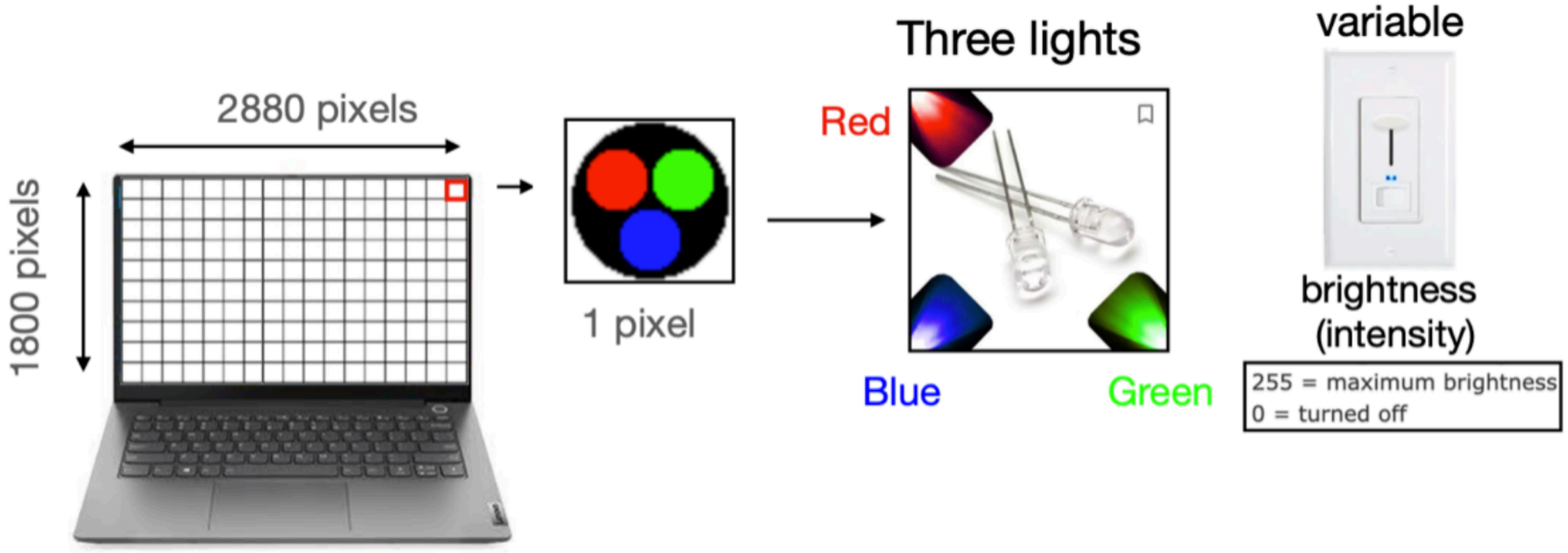
Number of Instances of each label type

```
----- TRAINING ------
label = 0     count = 5923
label = 1     count = 6742
label = 2     count = 5958
label = 3     count = 6131
label = 4     count = 5842
label = 5     count = 5421
label = 6     count = 5918
label = 7     count = 6265
label = 8     count = 5851
label = 9     count = 5949
TOTAL = 60000
-----   TEST   ------
label = 0     count = 980
label = 1     count = 1135
label = 2     count = 1032
label = 3     count = 1010
label = 4     count = 982
label = 5     count = 892
label = 6     count = 958
label = 7     count = 1028
label = 8     count = 974
label = 9     count = 1009
TOTAL = 10000
```

(Good feature balancing)

Color screens & pixels

- In a color screen the pixels are variable light bulbs outputting *red, green, or blue* light.

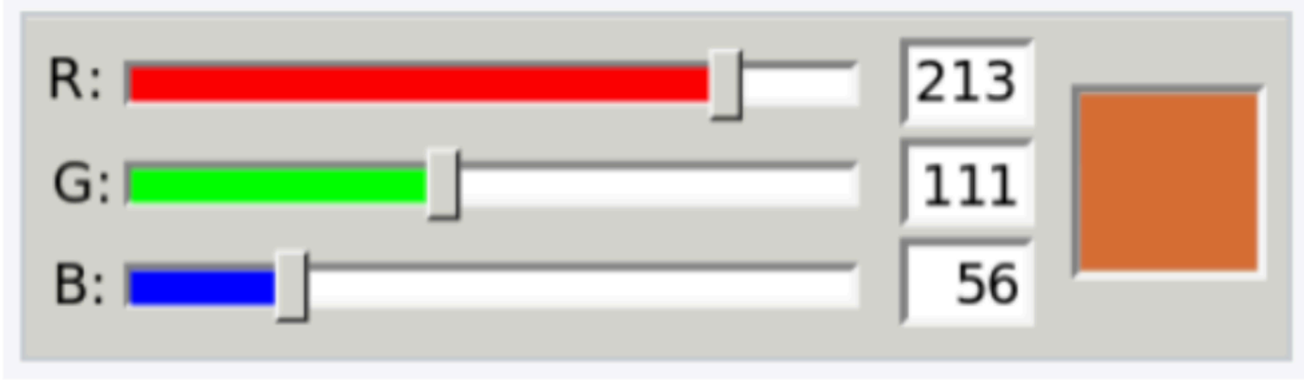


- We need *three numbers* (R,G,B) for each pixel to control the brightness of the 3 lights
- These three numbers allow us to form different colors through color-mixing

$$R \in [0,255]$$

$$G \in [0,255]$$

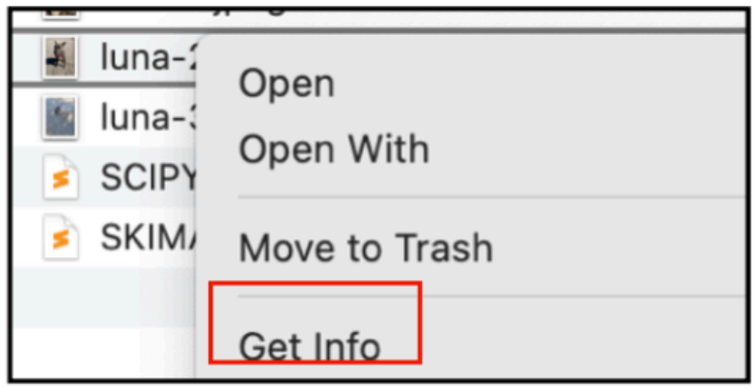
$$B \in [0,255]$$



- There are $255 \times 255 \times 255 = 16,581,375$ possible colors on the computer screen

Example: Color image





Meta-data

Last opened: Thursday, September 30, 2021 at 3:47 PM
Dimensions: 3024×4032
Device make: Apple
Device model: iPhone SE (2nd generation)
Color space: RGB
Color profile: Display P3
Focal length: 3.99 mm
Alpha channel: No
Red eye: No
Metering mode: Pattern
F number: f/1.8
Exposure program: Normal
Exposure time: 1/2,288

RESOLUTION: 3024 x 4032 → 12,192,768 PIXELS WITH 3 COLOR CHANNELS

My dog luna as a mathematical surface

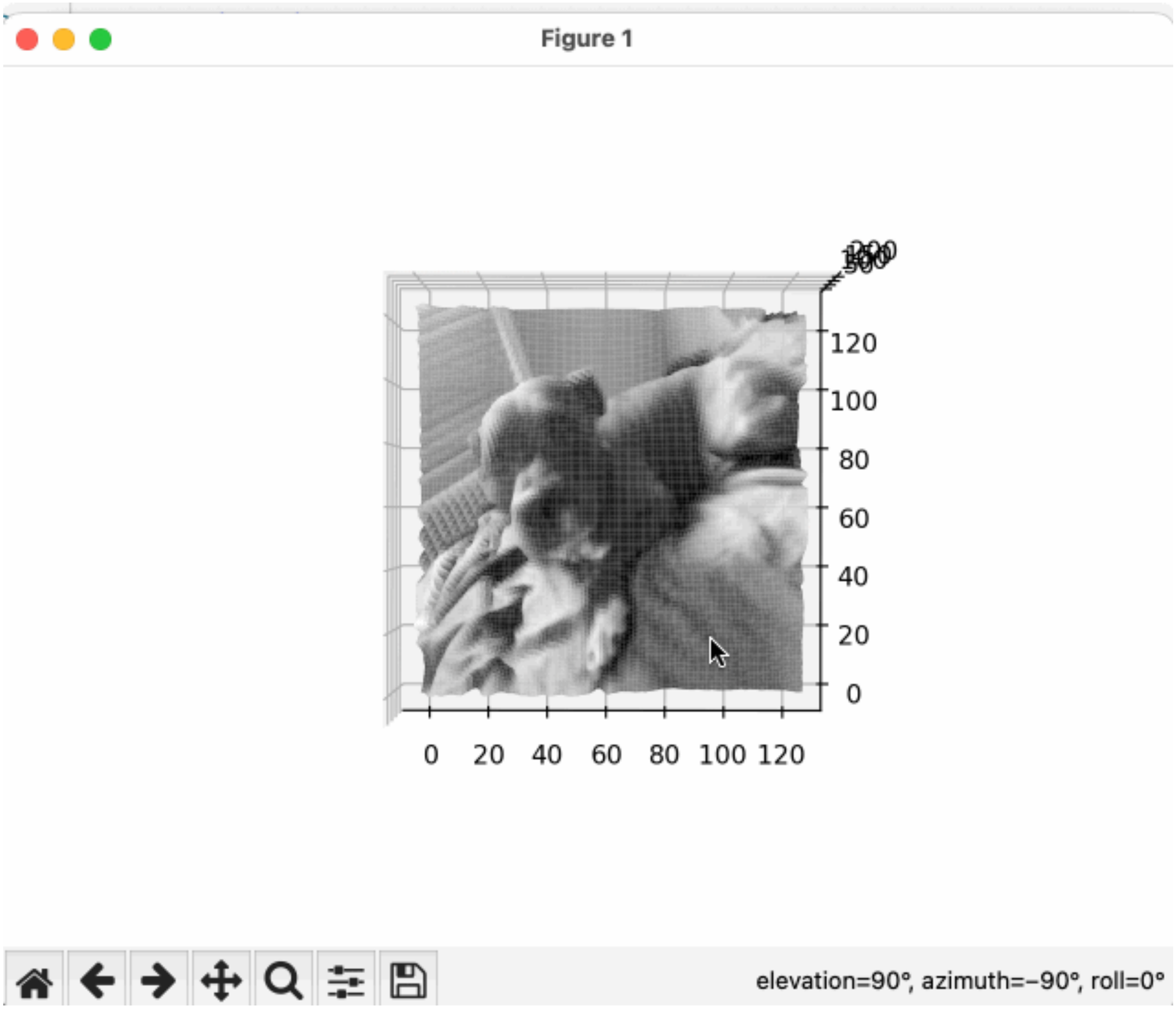
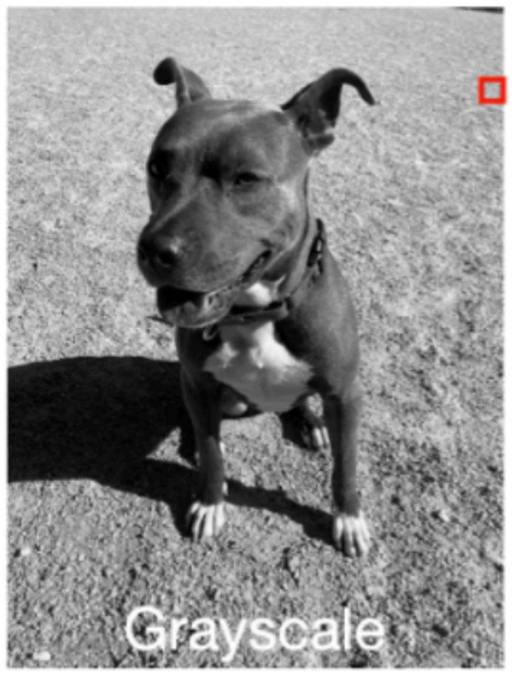
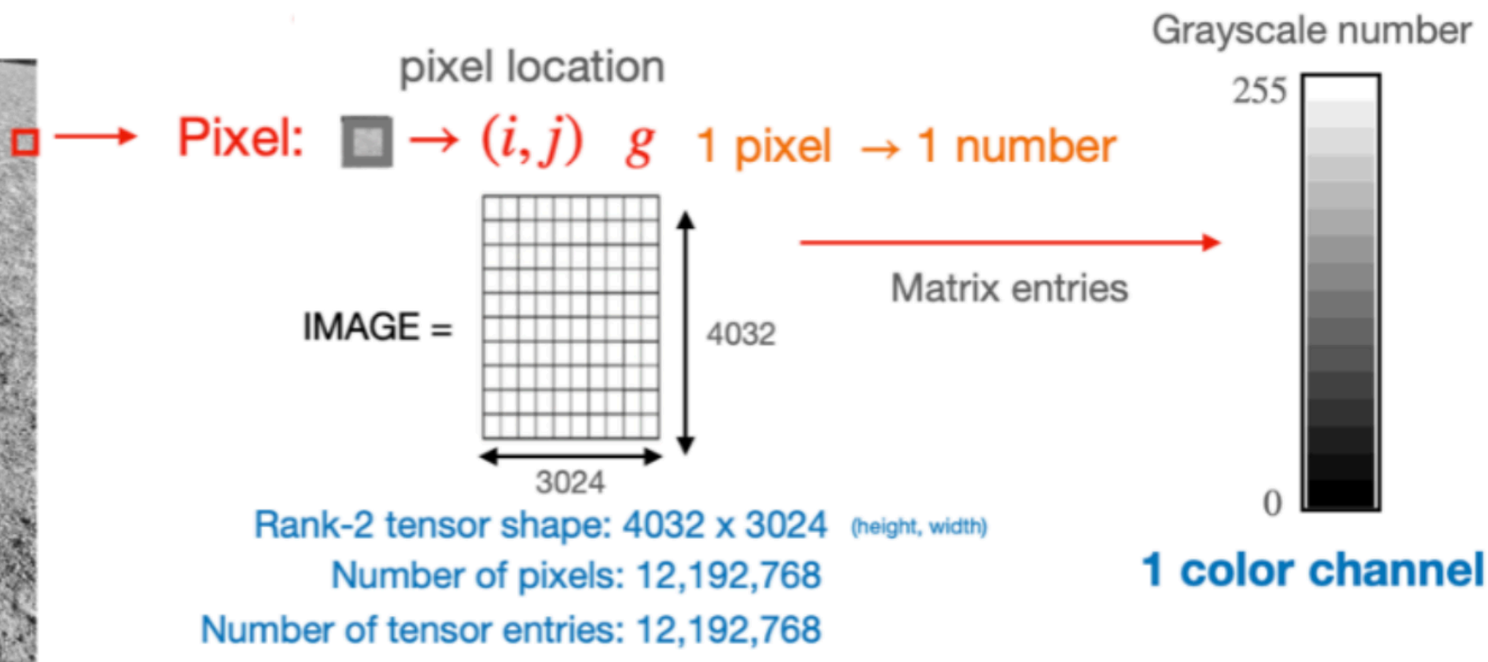


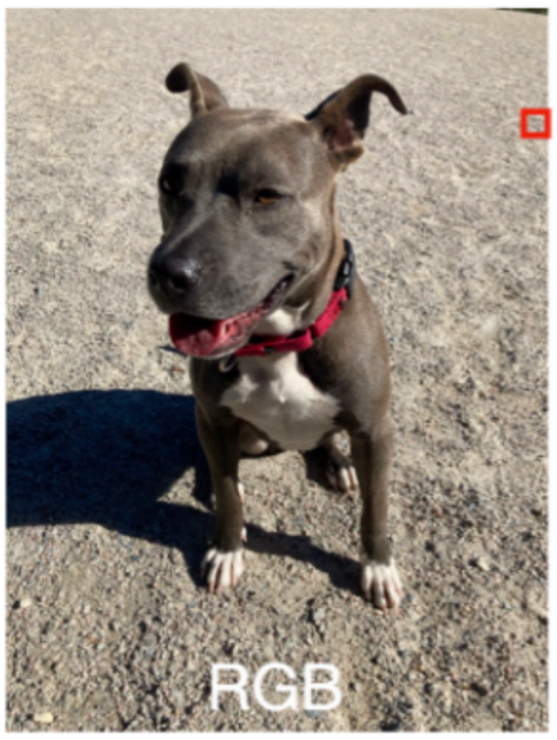
Image data

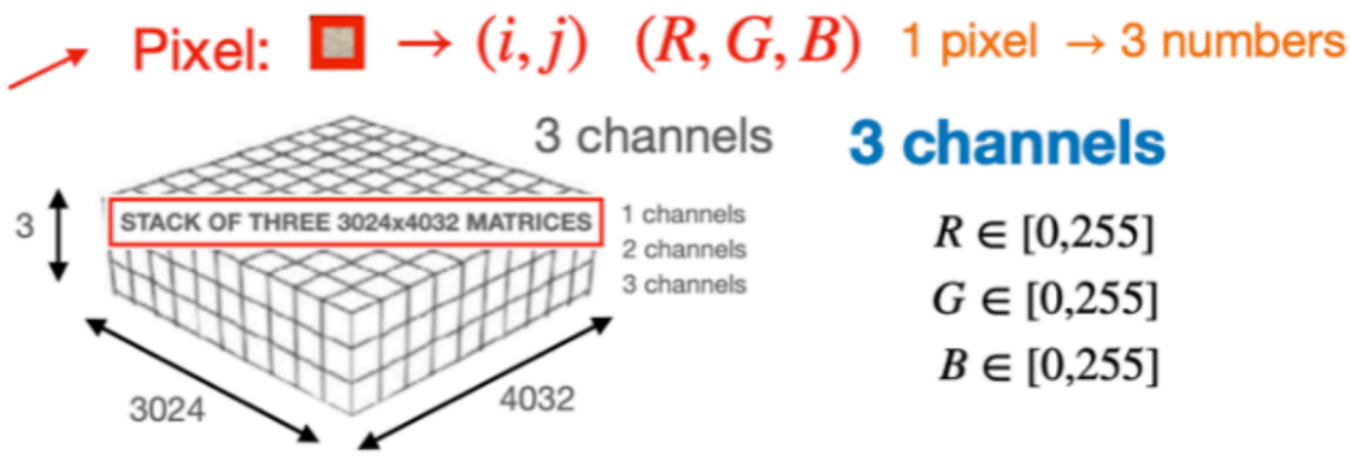
- **Grayscale: matrices** (e.g. `np.shape=(320,120) or np.shape=(320,120,1)`).





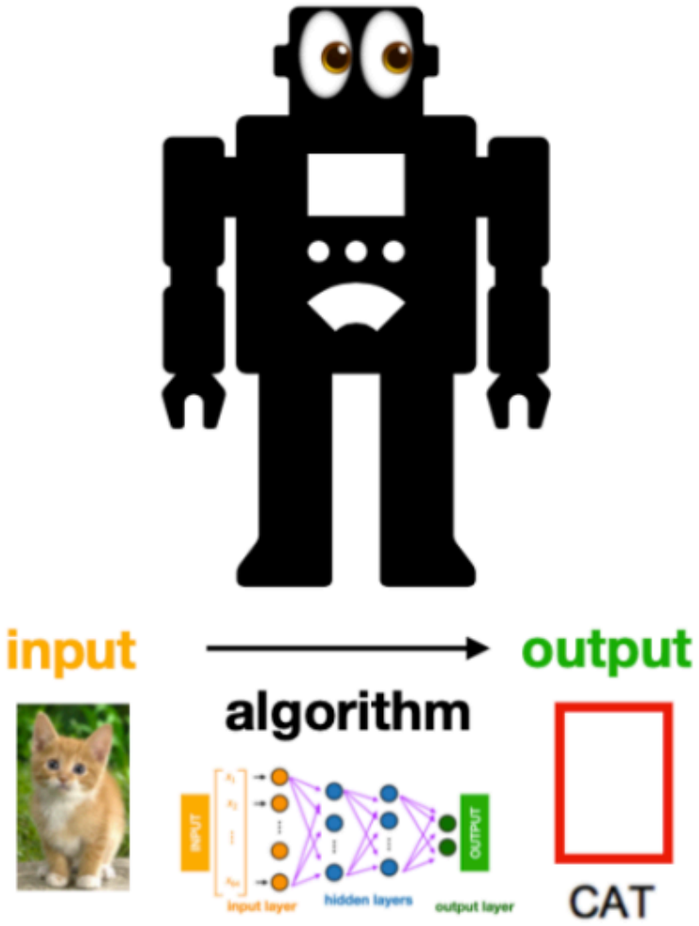
- **RGB: Rank-3 tensors** (stack of matrices)(e.g. `np.shape=(320,120,3)`).

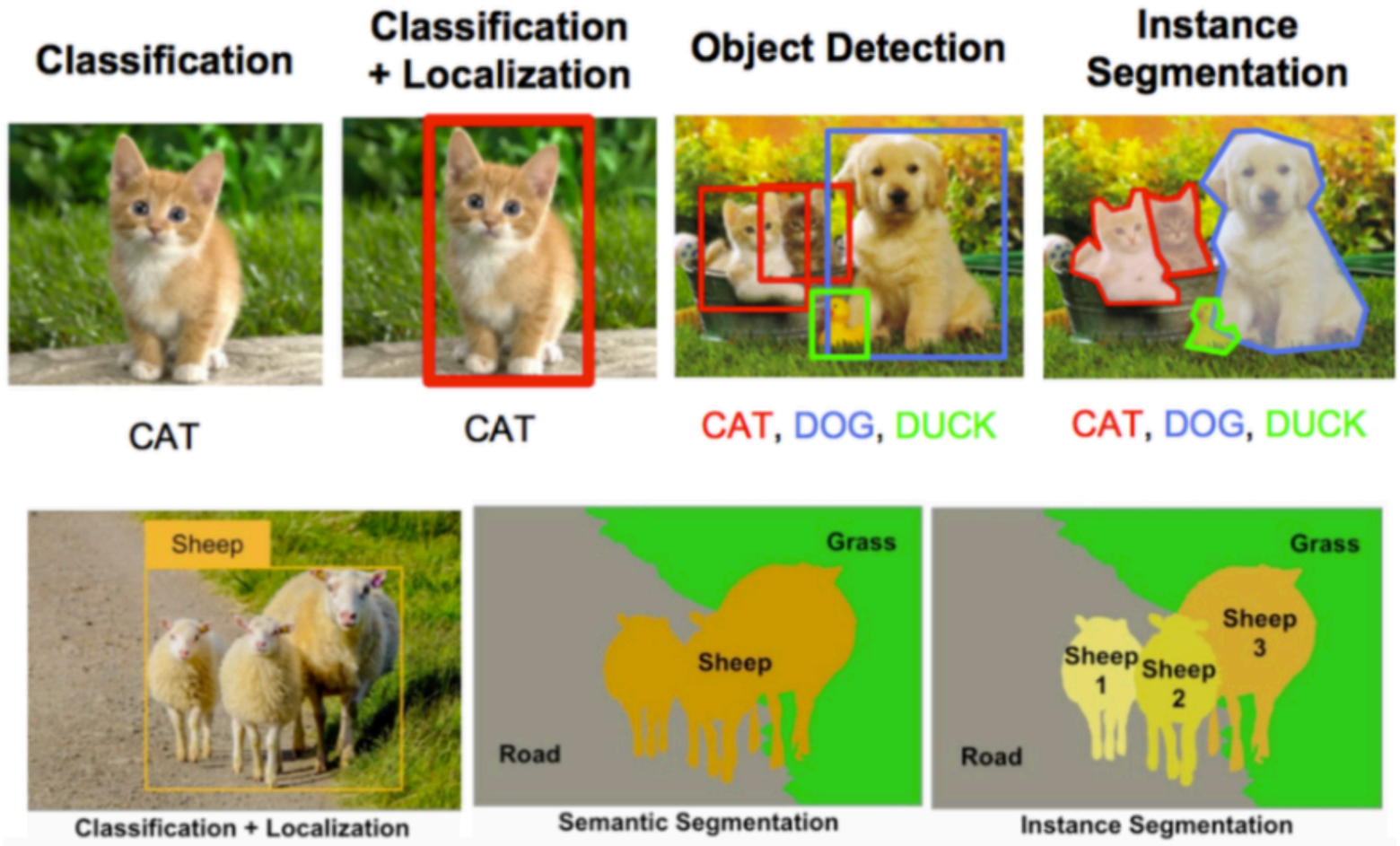




Traditional computer vision learning tasks

- **Images are data!!** Understanding image data is the first step in understanding computer vision and other advanced applications





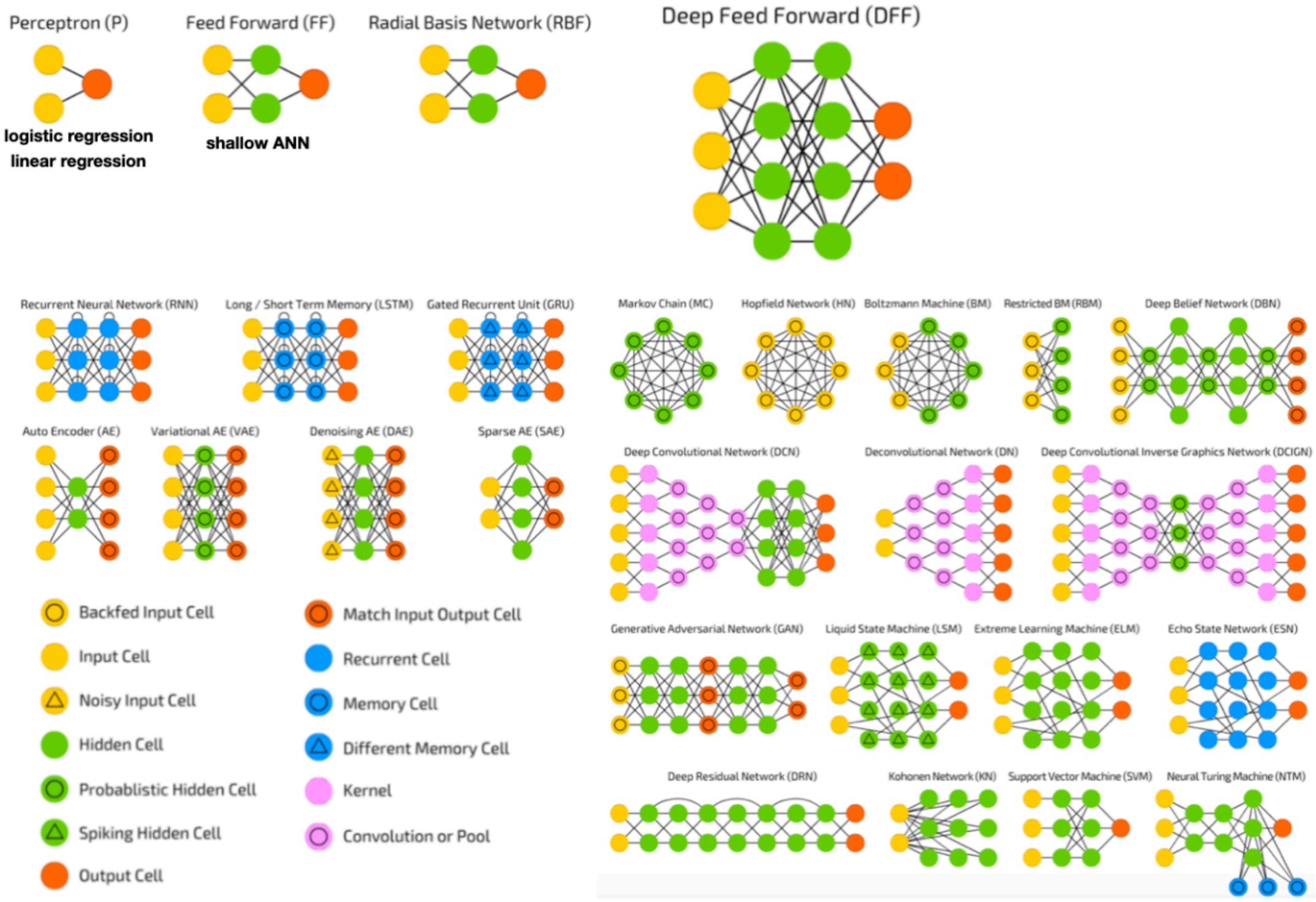
https://www.r-bloggers.com/2020/10/platypus-r-package-for-object-detection-and-image-segmentation/

- This becomes very important with respect to `Convolutional Neural networks`

Generative modeling https://thispersondoesnotexist.com/

Common Neural network architecture

- The type of data you have dictates which architecture you should use
- General classifier/regressor (MLP), image data (CNN), text data (LSTM or Transformer).



SOURCE: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

The deep learning training workflow

As illustrated with the simple case of linear regression

If you understand linear regression deeply you are well on your way to understanding deep learning

The “canonical” deep learning training workflow

- **Data** → collect, clean, **supervise**, split (train/val/test), augment
  - Garbage in, garbage out: this is where most projects live or die
- **Model Architecture** → choose a model that well suited to the data
  - MLP (simple data), CNN (grids/Images), RNN/Transformer (sequences), GNN (graphs)
- **Objective** (prediction quality scoring mechanism) → define a loss $\mathcal{L}(\theta)$ that encodes "good"
  - Regression: Mean square error MSE
  - Binary classification: Binary Cross-entropy
  - Multi-class classification: Categorical Cross-entropy
- **Optimization** (learning) → back-prop + SGD/Adam, LR schedule, regularization
  - We are just rolling downhill on a very high-dimensional surface
- **Evaluation** → validation metrics, error analysis, ablations
  - Tune on val, touch test *once*
- **Deployment** → export, serve, monitor drift, iterate

Aside: "Canonical" means conforming to a standard, authoritative, or generally accepted form the version of something that serves as the reference point against which variants are judged (e.g., the canonical form of a matrix, or the canonical version of a text).

Raw Data

Collect data on houses

Resultant cleaned Pandas data-frame

s a p → Temporary variable name

Sample (batch) dimension

i	sqft 1000 ft²	age years	price $1000s
0	1.45	12	365.4
1	0.98	45	219.8
2	2.10	3	498.6
3	1.32	28	308.1
4	2.85	8	625.7
5	1.70	35	366.2
⋮			

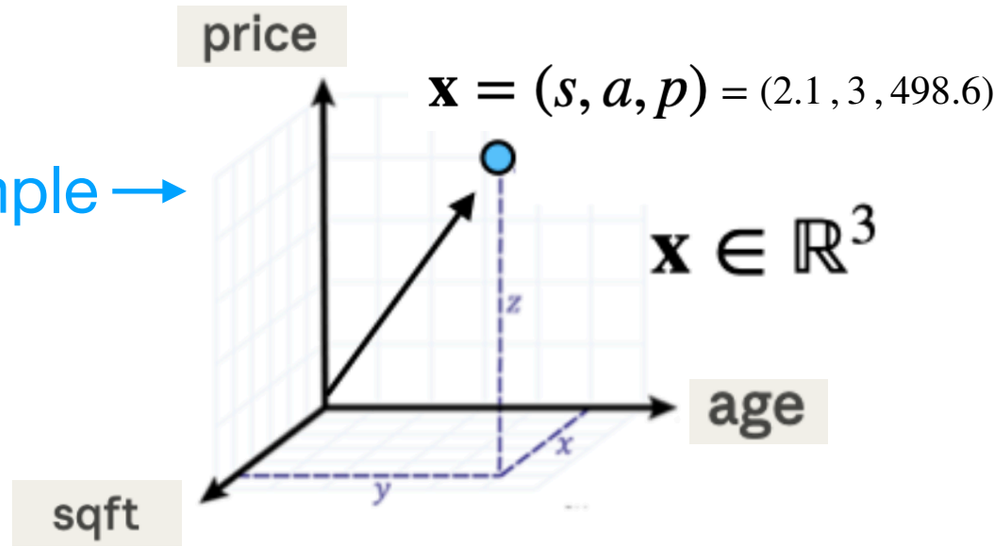
i^{th} sample →

Unit of observation = house

Let's assume there are 100 rows

Full Data shape: (100, 3)

Data Space



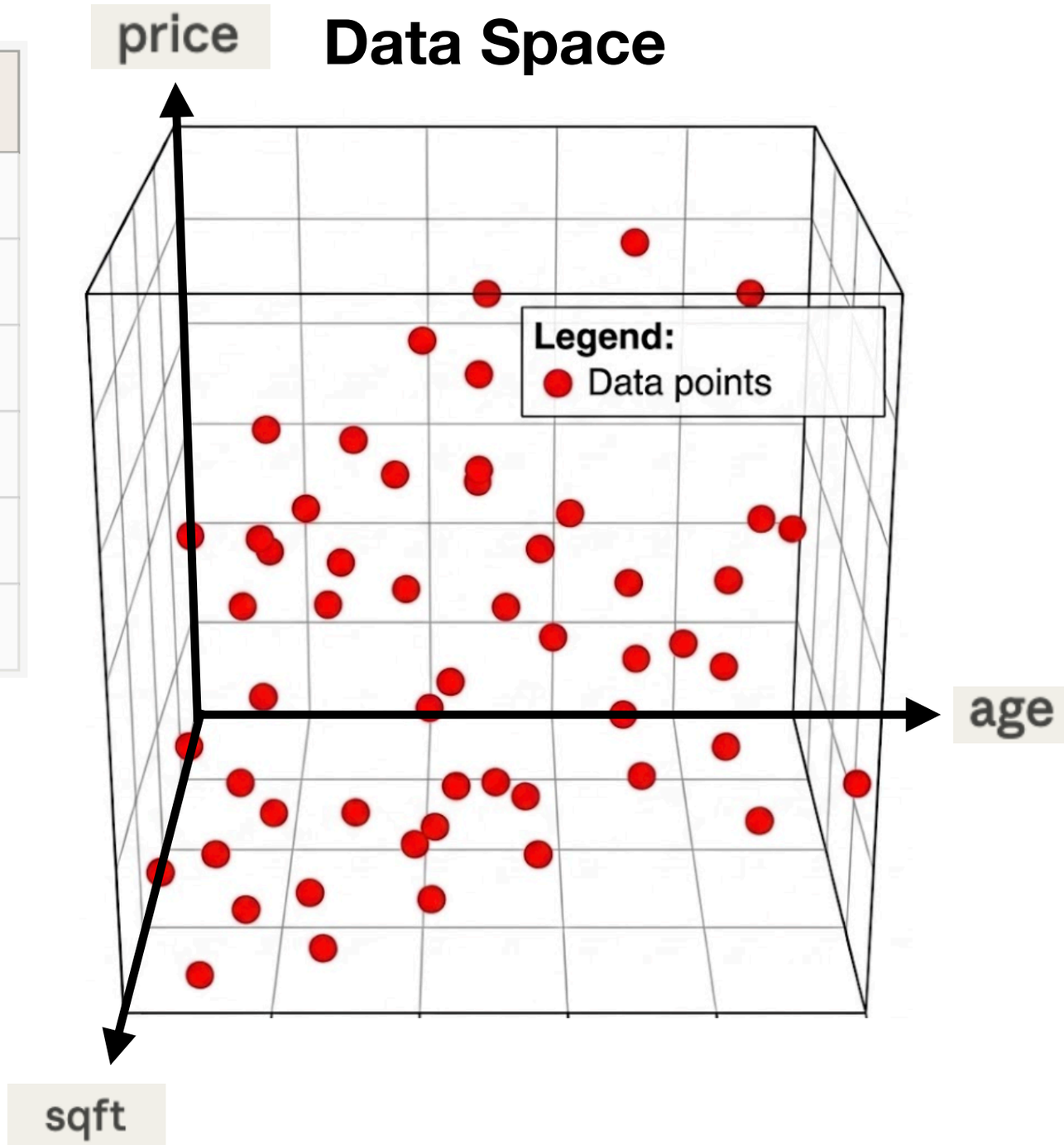
Full data-set lives in $\mathbb{R}^3$

Basic Exploratory data analysis (EDA)

Since the data is so simplistic, there's not much to do here other than visualizing a standard point cloud.

Sample (batch) dimension

i	sqft 1000 ft²	age years	price $1000s
0	1.45	12	365.4
1	0.98	45	219.8
2	2.10	3	498.6
3	1.32	28	308.1
4	2.85	8	625.7
5	1.70	35	366.2

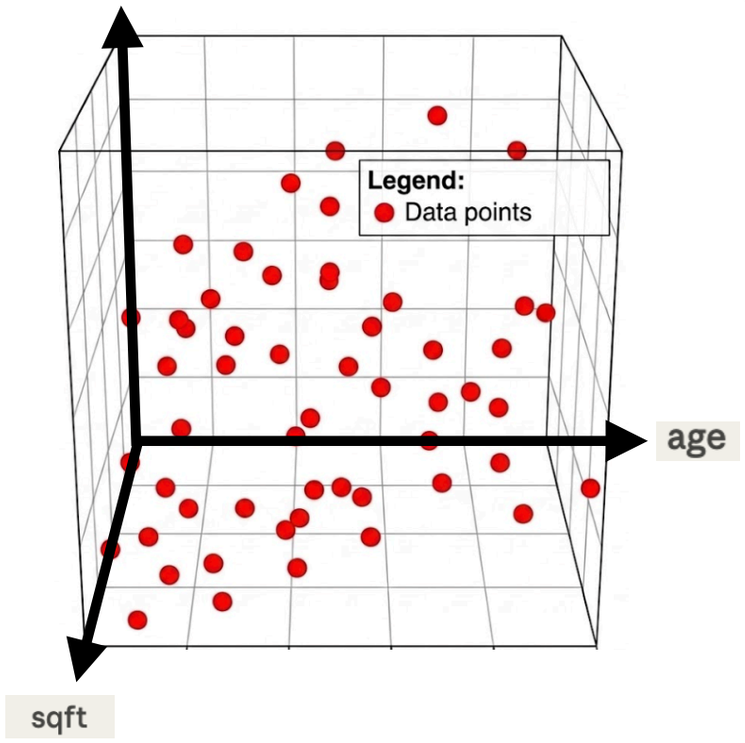


Clearly there is a numeric relationship between the variables.

Aside: Normalization

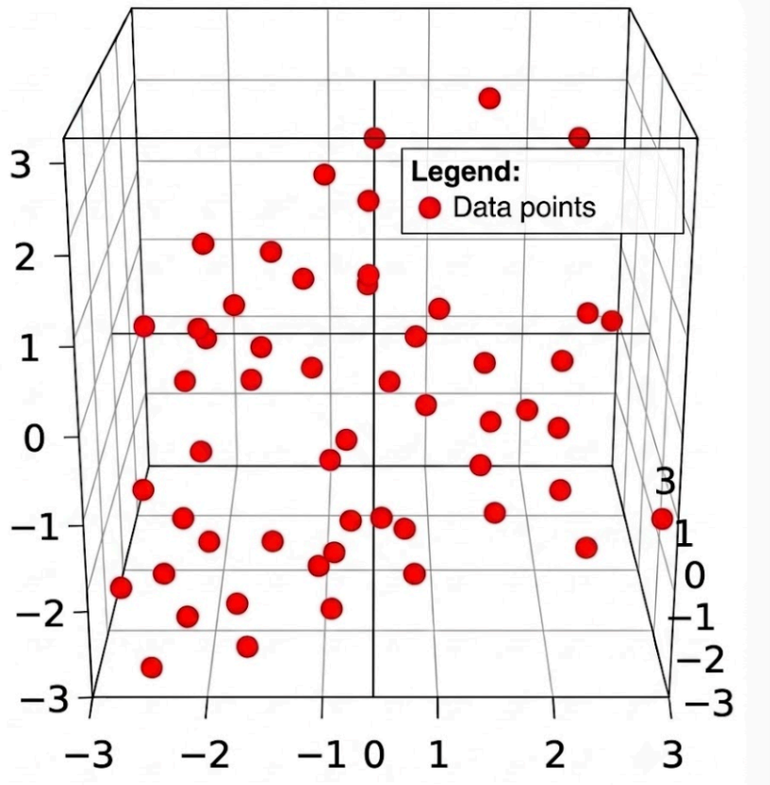
Normalization is very important in deep learning. It is a general procedure. It makes the models train more efficiently. They tend to work better with numbers on the order of negative ten to ten centered around the origin.

Data with real units (e.g. years)



$$z = \frac{x - \mu}{\sigma}$$

Data on standarized scale



i	sqft 1000 ft²	age years	price $1000s
0	1.45	12	365.4
1	0.98	45	219.8
2	2.10	3	498.6
3	1.32	28	308.1
4	2.85	8	625.7
5	1.70	35	366.2

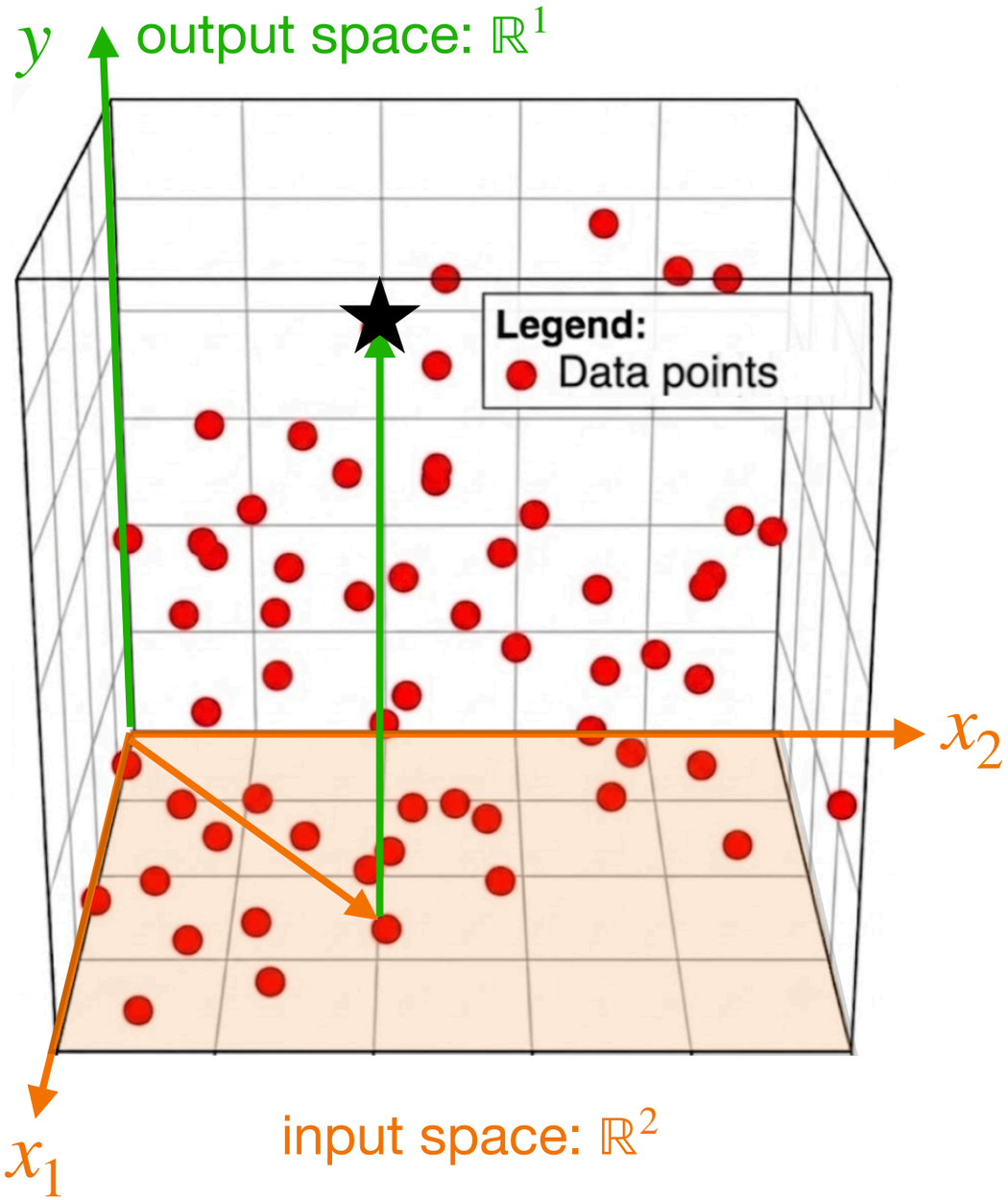
i	sqft z units	age z units	price z units
0	−0.47	−0.65	−0.24
1	−1.24	1.52	−1.35
2	0.61	−1.24	0.77
3	−0.68	0.41	−0.68
4	1.84	−0.91	1.74
5	−0.05	0.86	−0.24

μ	1.733	21.83	397.30
σ	0.606	15.23	131.58

below the column mean above the column mean

Supervision

In this case the supervision lies in the act of establishing or hypothesizing the relationship between the input space and the output space.



input space: x_1 x_2 — output space: y

	input space: $\mathbb{R}^2$		output space: $\mathbb{R}^1$
i	sqft $x_1 \cdot 1000\ ft^2$	age $x_2 \cdot$ years	price $y \cdot \$1000s$
0	1.45	12	365.4
1	0.98	45	219.8
2	2.10	3	498.6
3	1.32	28	308.1
4	2.85	8	625.7
5	1.70	35	366.2

★

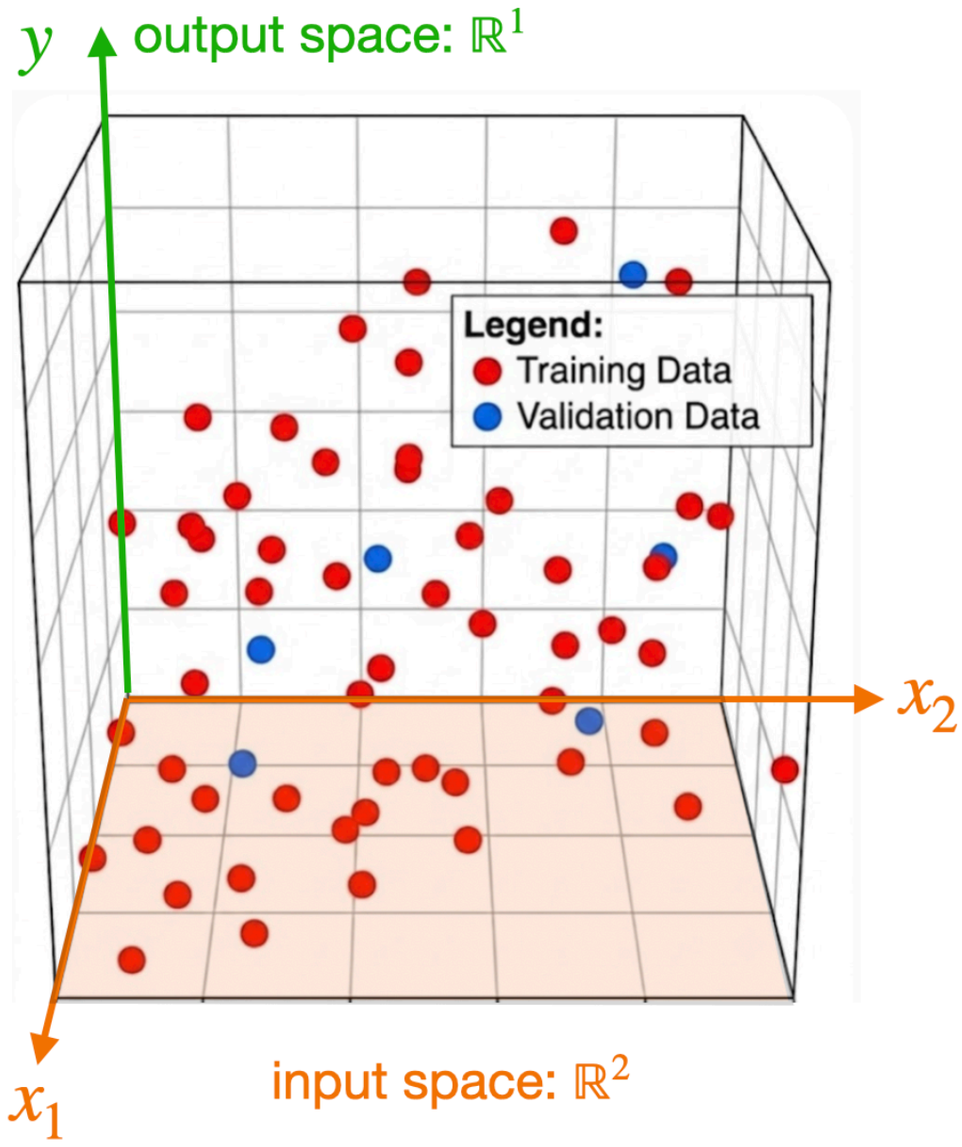
position on the floor — height above the floor

$$\mathbf{x} = (x_1, x_2) \xrightarrow{\text{functional relationship}} y$$

Note that in reality the data is generally much higher dimensional and the existence of the relationship cannot be inferred visually and instead hypothesized and verified during training.

Data split

Deep learning models are very prone to overfitting, so Splitting our data into tra tra tra tra tra tra tra tra tra tra tra training and validation or training test and validation is a non negotiable practice.



x_1 x_2 y

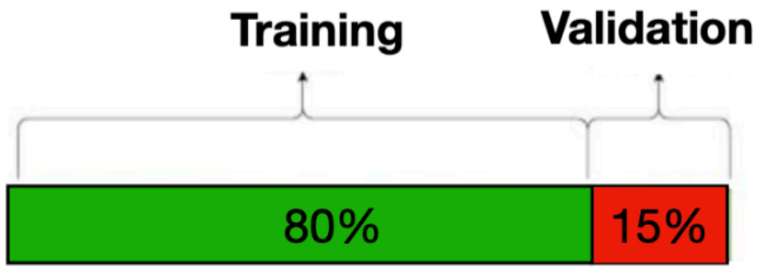
	input space: $\mathbb{R}^2$		output space: $\mathbb{R}^1$	not a feature
i	sqft x_1 · 1000 ft²	age x_2 · years	price y · $1000s	split
0	1.45	12	365.4	train
1	0.98	45	219.8	train
2	2.10	3	498.6	validation
3	1.32	28	308.1	train
4	2.85	8	625.7	train
5	1.70	35	366.2	validation

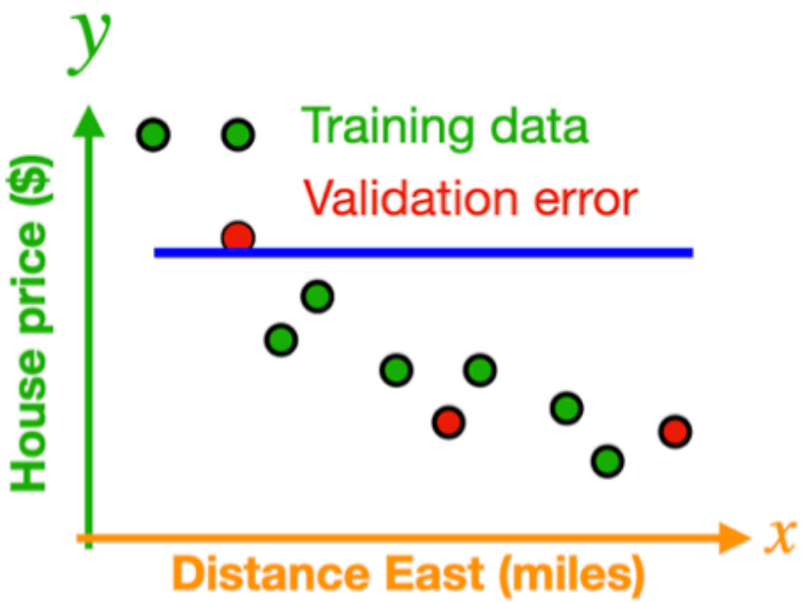
position on the floor height above the floor red / blue dots in the figure

Note that in reality the data is generally much higher dimensional and the existence of the relationship cannot be inferred visually and instead hypothesized and established during training.

Recall why we need the validation set!

monitor and avoid overfitting!



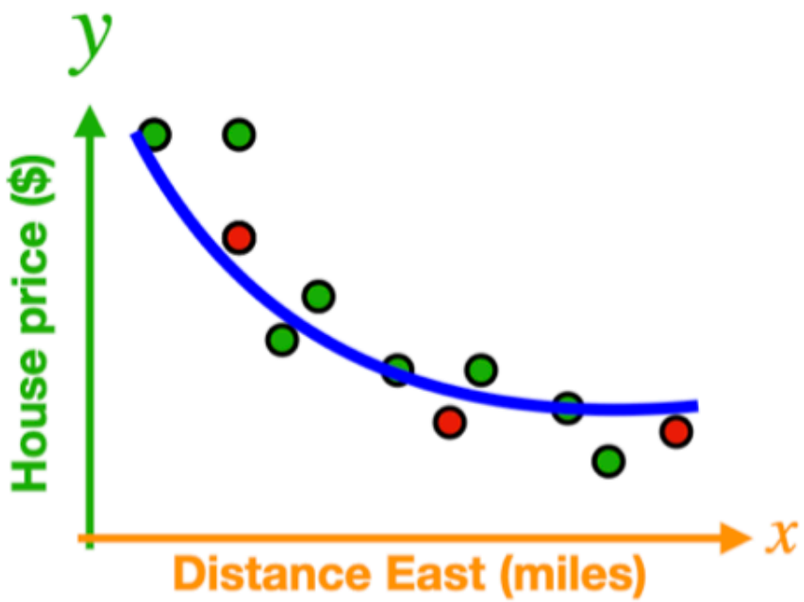


under-fitting
model complexity is too low

High training error
High Validation error

high bias

low variance

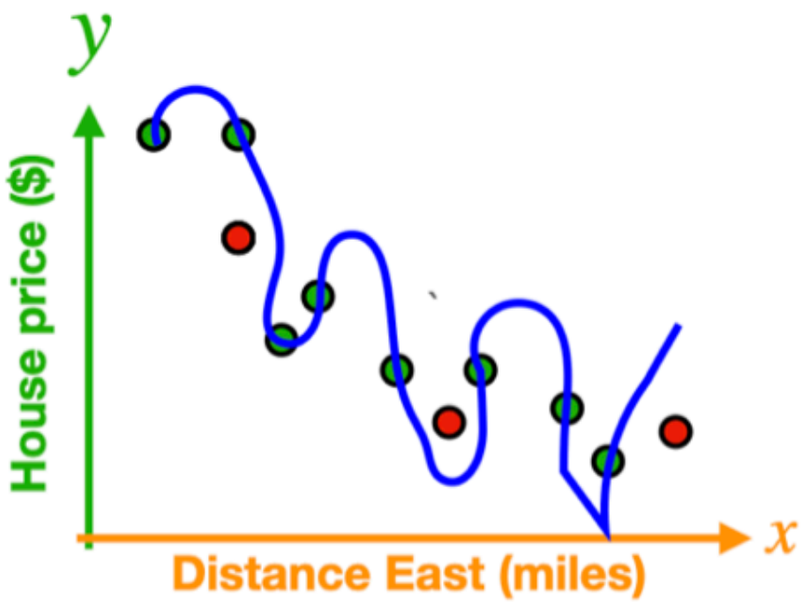


optimal
correct level of model complexity

Training and Validation error
are equally low

low bias

low variance



over-fitting
model complexity is too high

Very Low training error
Very high Validation error

low bias

high variance

The “canonical” deep learning training workflow

- **Data** → collect, clean, **supervise**, split (train/val/test), augment
  - Garbage in, garbage out: this is where most projects live or die
- **Model Architecture** → choose a model that well suited to the data
  - MLP (simple data), CNN (grids/Images), RNN/Transformer (sequences), GNN (graphs)
- **Objective** (prediction quality scoring mechanism) → define a loss $\mathcal{L}(\theta)$ that encodes "good"
  - Regression: Mean square error MSE
  - Binary classification: Binary Cross-entropy
  - Multi-class classification: Categorical Cross-entropy
- **Optimization** (learning) → back-prop + SGD/Adam, LR schedule, regularization
  - We are just rolling downhill on a very high-dimensional surface
- **Evaluation** → validation metrics, error analysis, ablations
  - Tune on val, touch test *once*
- **Deployment** → export, serve, monitor drift, iterate

Aside: "Canonical" means conforming to a standard, authoritative, or generally accepted form the version of something that serves as the reference point against which variants are judged (e.g., the canonical form of a matrix, or the canonical version of a text).

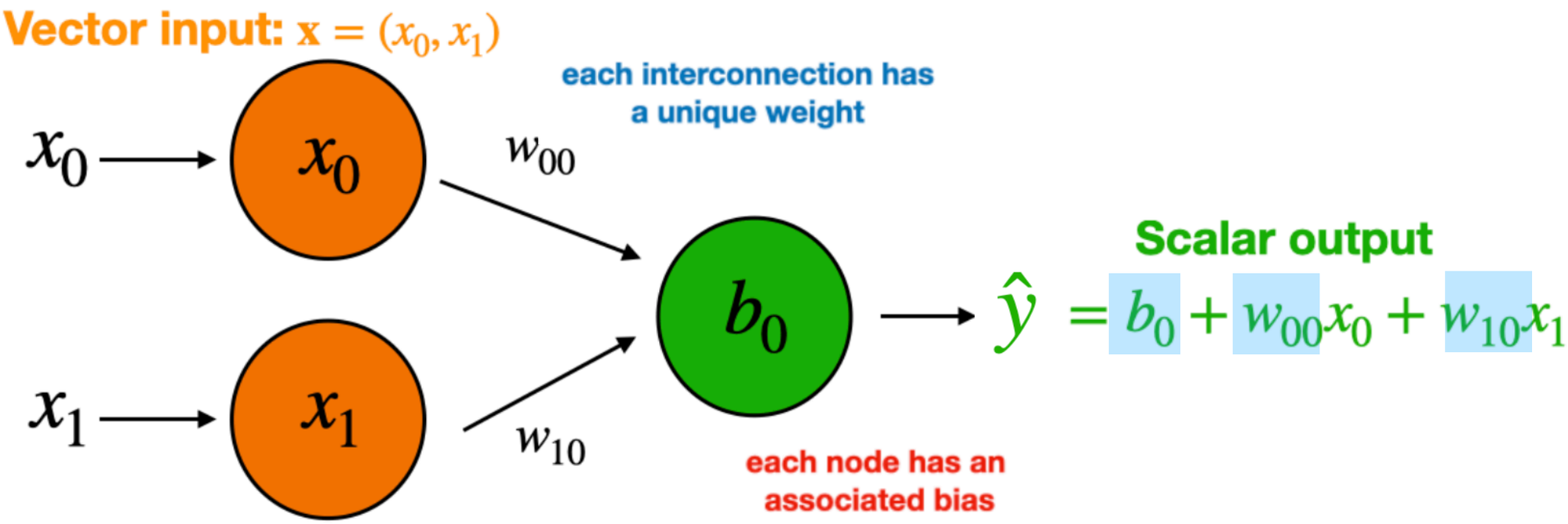
Choosing a model: For now consider a linear model

model=prediction machine: $\hat{y} = M(\mathbf{x}|\mathbf{w})$

- For example, consider a **two** variable $\mathbf{x} = (x_0, x_1)$ linear scalar regression model

$$\hat{y} = M(\mathbf{x}|\mathbf{w}) = b_0 + w_0 x_0 + w_1 x_1$$

- We can represent this as a "computational graph"



- **Problem Type:** Scalar-regression
- **Mapping:** $\mathbb{R}^2 \rightarrow \mathbb{R}^1$ (input layer) (output layer)
- **Hidden layers**: None
- **Hidden-layer activation**: N/A
- **Output-layer activation**: linear
- **Number of fitting parameters**: 3

Bivariate linear regression = Our first "neural network" model

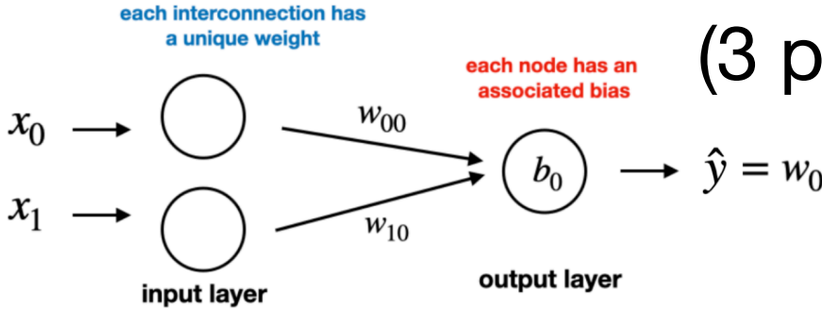
For educational purposes let's drop a feature to simplify the math

	input space: $\mathbb{R}^2$		output space: $\mathbb{R}^1$	not a feature
i	sqft x_1 · 1000 ft²	age x_2 · years	price y · $1000s	split
0	1.45	12	365.4	train
1	0.98	45	219.8	train
2	2.10	3	498.6	validation
3	1.32	28	308.1	train
4	2.85	8	625.7	train
5	1.70	35	366.2	validation

position on the floor height above the floor red / blue dots in the figure

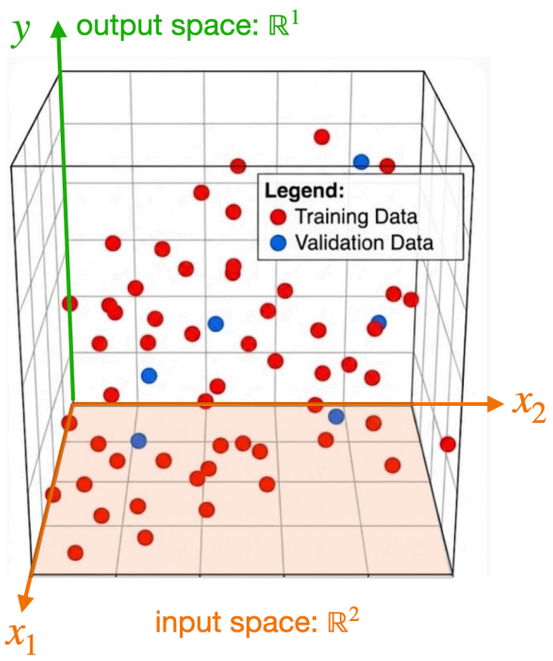
x_1 y

i	sqft x_1 · 1000 ft²	price y · $1000s	split
0	1.45	365.4	train
1	0.98	219.8	train
2	2.10	498.6	validation
3	1.32	308.1	train
4	2.85	625.7	train
5	1.70	366.2	validation

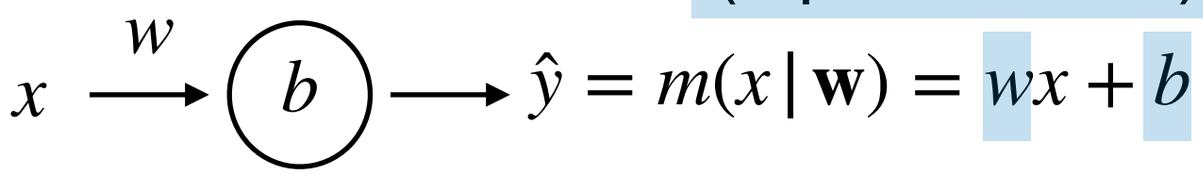


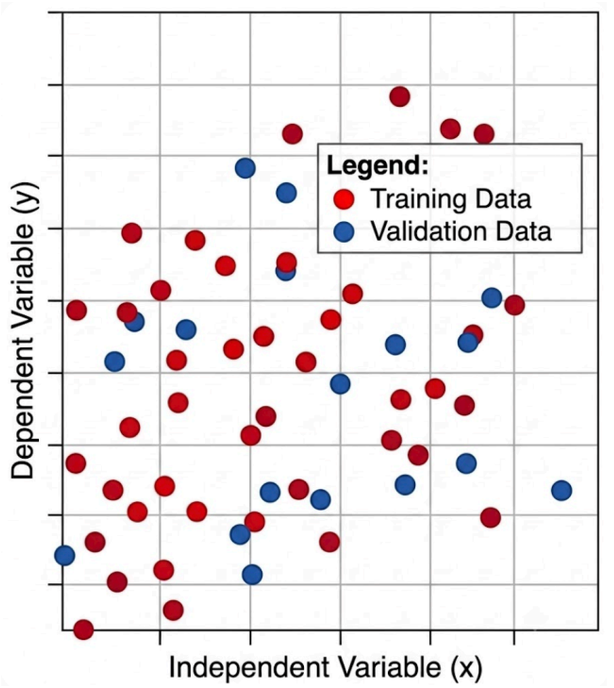
(3 parameters)

$$\hat{y} = w_{00}x_0 + w_{10}x_1 + b_0$$



(2 parameters)





The “canonical” deep learning training workflow

- **Data** → collect, clean, **supervise**, split (train/val/test), augment
  - Garbage in, garbage out: this is where most projects live or die
- **Model Architecture** → choose a model that well suited to the data
  - MLP (simple data), CNN (grids/Images), RNN/Transformer (sequences), GNN (graphs)
- **Objective** (prediction quality scoring mechanism) → define a loss $\mathcal{L}(\theta)$ that encodes "good"
  - Regression: Mean square error MSE
  - Binary classification: Binary Cross-entropy
  - Multi-class classification: Categorical Cross-entropy
- **Optimization** (learning) → back-prop + SGD/Adam, LR schedule, regularization
  - We are just rolling downhill on a very high-dimensional surface
- **Evaluation** → validation metrics, error analysis, ablations
  - Tune on val, touch test *once*
- **Deployment** → export, serve, monitor drift, iterate

Aside: "Canonical" means conforming to a standard, authoritative, or generally accepted form the version of something that serves as the reference point against which variants are judged (e.g., the canonical form of a matrix, or the canonical version of a text).

Parameters are everything in deep learning!!

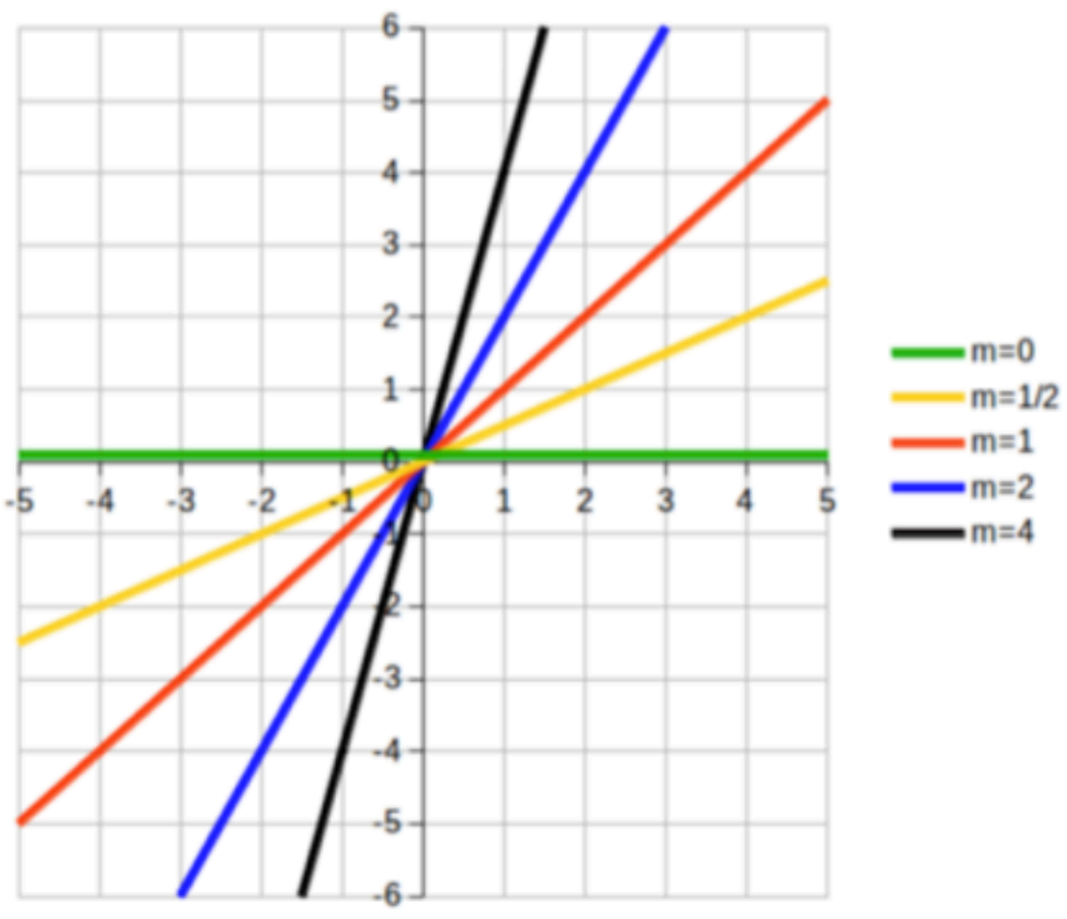
There are an infinite number of possible parameterizations of this plane

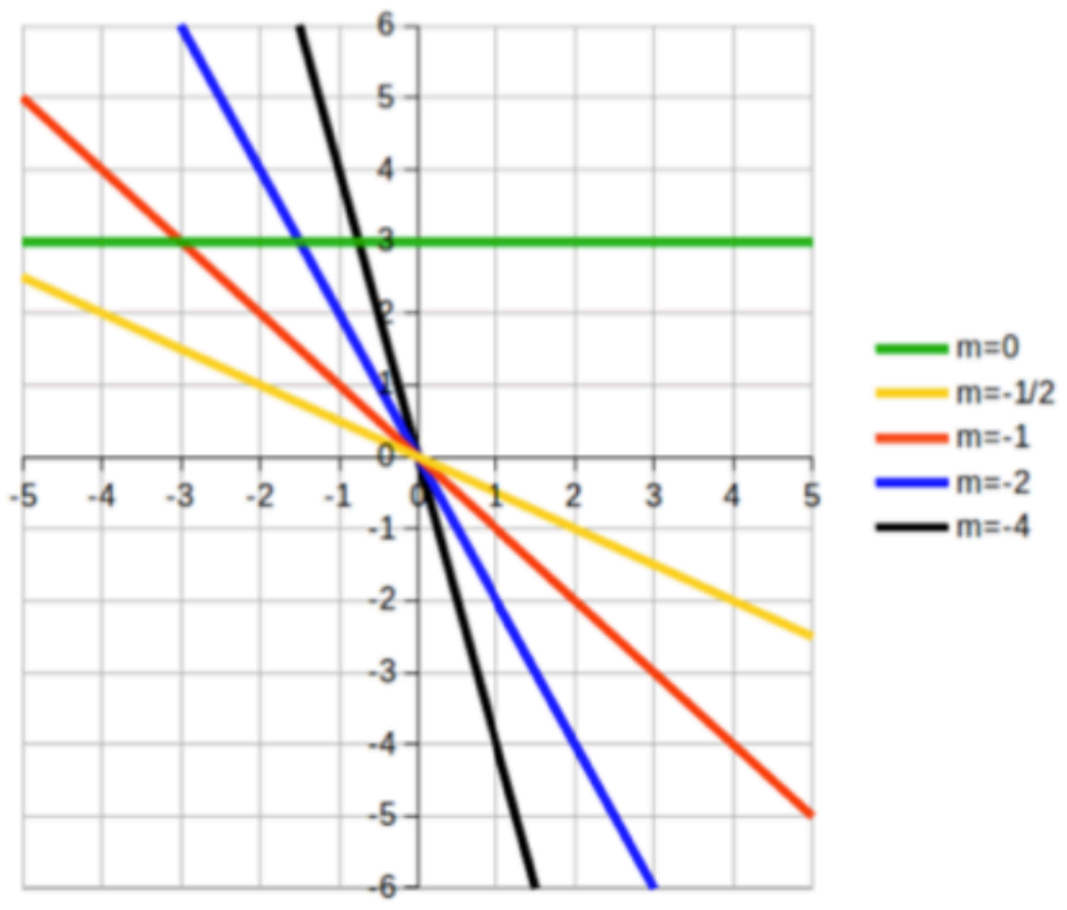
$$x \xrightarrow{w} \boxed{b} \longrightarrow \hat{y} = m(x \mid \mathbf{w}) = w_1 x + w_0 = mx + b$$

(2 parameters)

This model represents all possible lines in a two dimensional plane.

A particular choice of parameters instantiates a particular line.



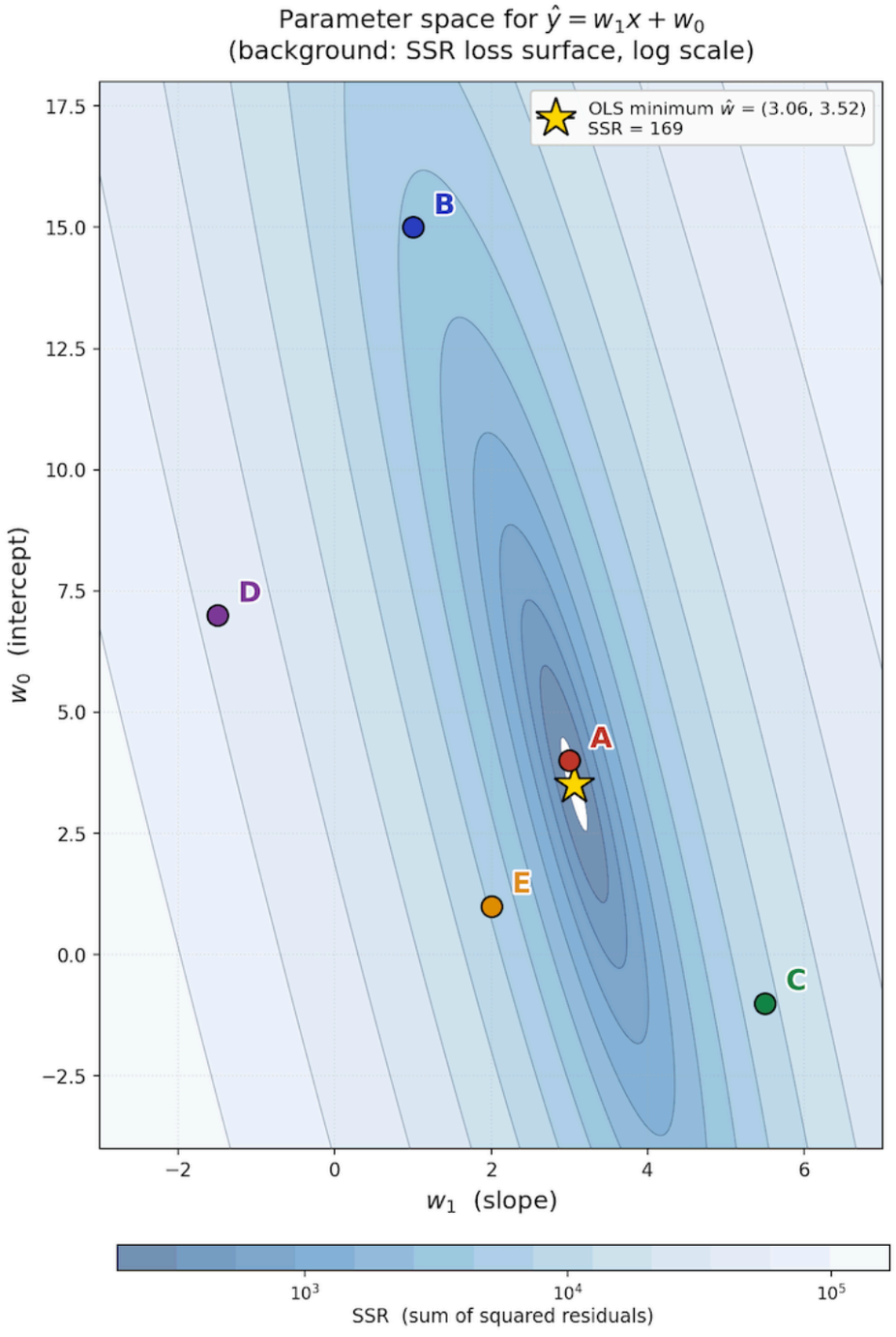


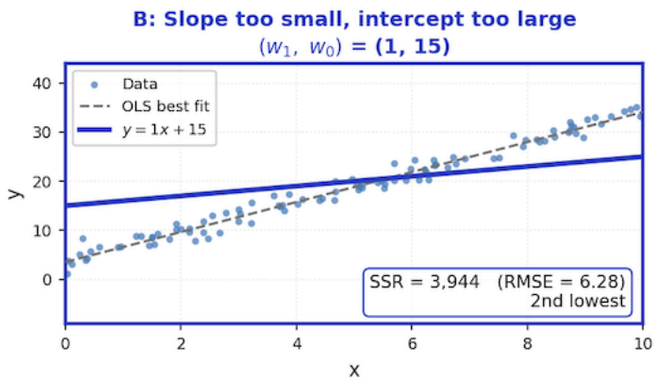
Parameters are what we learn in deep learning

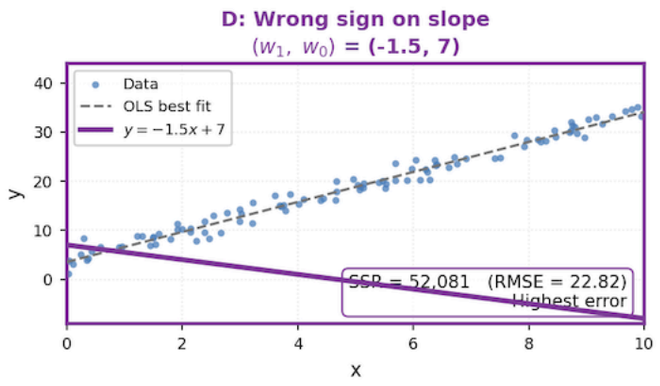
Parameter space

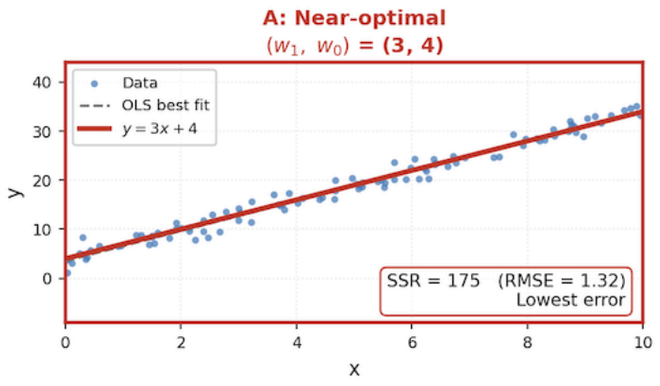
Data space

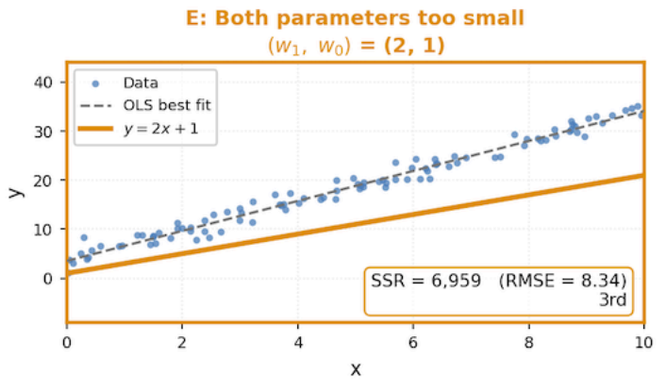
Parameter space ↔ data space: every (w_1, w_0) pair is one candidate line

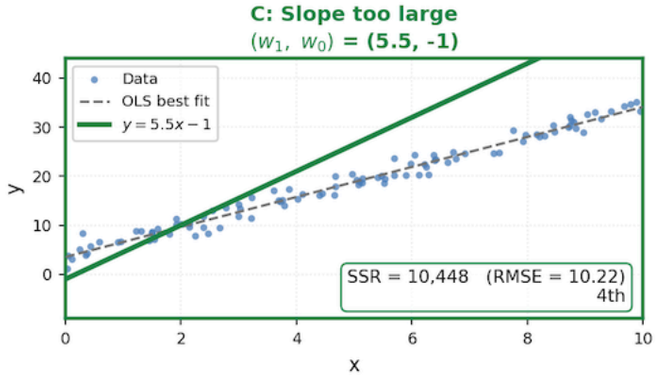













```
HOW TO READ THIS FIGURE

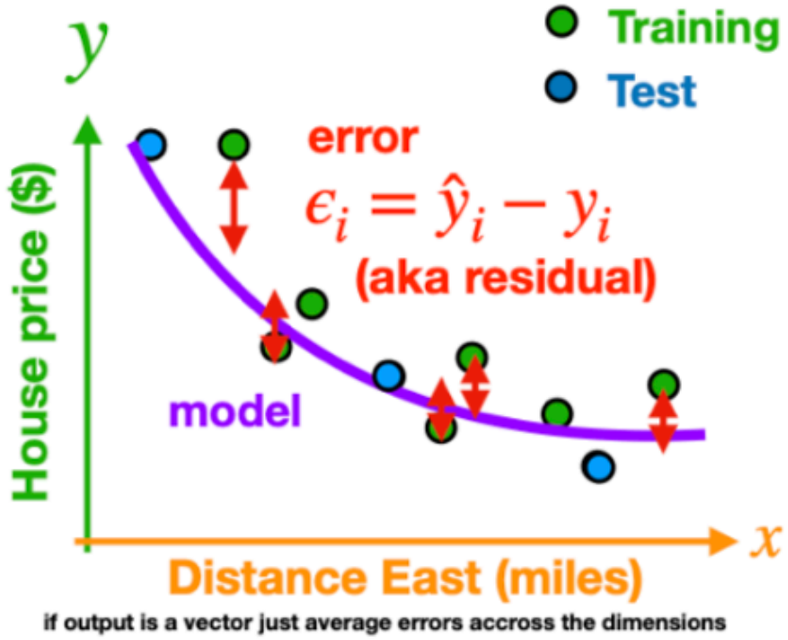
One point in parameter space (left)
= one candidate line in data space.

The contours are the SSR loss surface:
the farther a point sits from the gold
star (the OLS minimum), the larger its
SSR and the worse its line fits.

Data:  n = 100,  y = 3x + 4 + N(0, 1.5²)
SSR floor ≈ nσ² = 225

SSR ranking:
   A: SSR =      175
   B: SSR =    3,944
   E: SSR =    6,959
   C: SSR =   10,448
   D: SSR =   52,081
```

Regression cost functions



if output is a vector just average errors accross the dimensions

$$\underset{data}{\mathbf{y}} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \quad \underset{predictions}{\mathbf{m}(x \mid \mathbf{p}) = \hat{\mathbf{y}}} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_N \end{pmatrix} \quad \underset{errors}{\mathbf{y} - \hat{\mathbf{y}}} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix}$$

Sum of square error (SSE)

$$SSE = \sum_i^N \epsilon_i^2 = \sum_i^N (y_i - \hat{y}_i)^2$$

no $\frac{1}{N}$

Mean square error (MSE) (variance of error distribution)

$$MSE = \frac{1}{N} \sum_i^N \epsilon_i^2 = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$$

Root mean square error (RMSE) (standard deviation of error distribution)

$$RMSE = \left(\frac{1}{N} \sum_i^N \epsilon_i^2 \right)^{\frac{1}{2}} = \left(\frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2 \right)^{\frac{1}{2}}$$

Mean absolute error (MAE) Less sensitive to outliers

$$MAE = \frac{1}{N} \sum_i^N |\epsilon_i| = \frac{1}{N} \sum_i^N |y_i - \hat{y}_i|$$

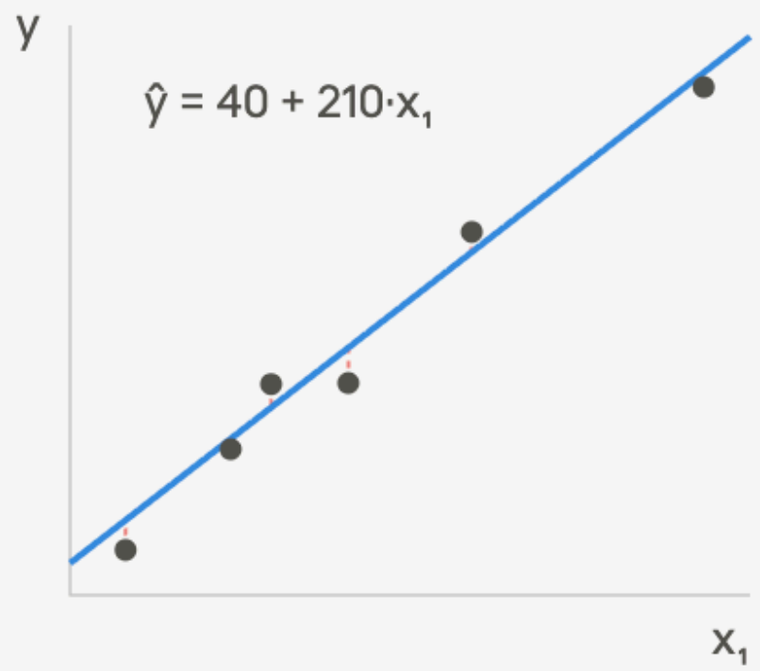
Scoring the quality of fit: SSE

$$\hat{y} = w_0 + w_1 \cdot x_1 \text{ with } p = (40, 210)$$

everything to the right of the dashed line depends on p

i	x_1 sqft	y price	$\hat{y}$ predicted	ε residual	ε^2 sq. error	split
0	1.45	365.4	344.5	+20.9	436.8	● train
1	0.98	219.8	245.8	−26.0	676.0	● train
2	2.10	498.6	481.0	+17.6	309.8	○ validation
3	1.32	308.1	317.2	−9.1	82.8	● train
4	2.85	625.7	638.5	−12.8	163.8	● train
5	1.70	366.2	397.0	−30.8	948.6	○ validation

given data | computed from p



436.8
676.0
309.8
82.8
163.8
948.6

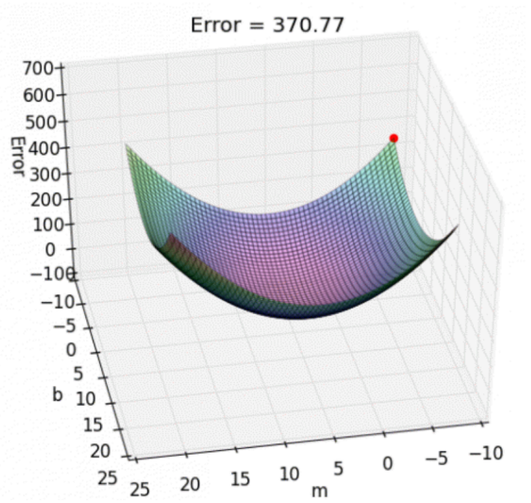
Σ → L(p) **2617.8**

one number per choice of p

Loss surface: Ordinary Least squares (OLS)

- Assume we have supervised training data $x_i \in (x_1, x_2 \dots x_N) \rightarrow y_i \in (y_1, y_2 \dots y_N)$
- Lets use a single variable linear model $y_{pred} = M(x|\mathbf{w}) = M(x|m, b) = mx + b$
- Let's also use mean square error (MSE) as our loss function:

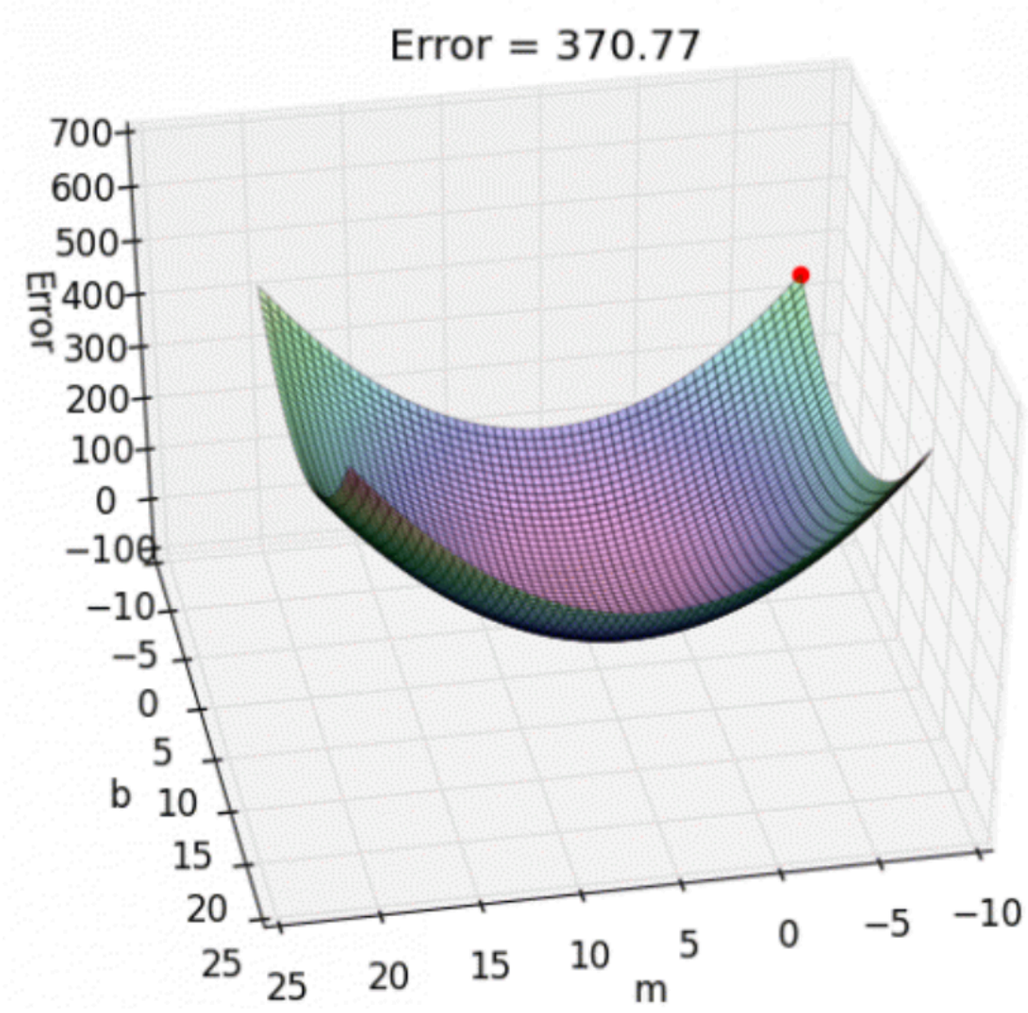
$$
\begin{aligned}
L(m,b) &= \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad (MSE) \\
&= \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}(y_i - mx_i - b)(y_i - mx_i - b) \\
&= \frac{1}{N}\sum_{i=1}^{N} y_i^2 - mx_iy_i - y_ib - mx_iy_i + (mx_i)^2 + mx_ib - by_i + bmx_i + b^2 \\
&= \frac{1}{N}\sum_{i=1}^{N} y_i^2 - 2x_iy_im - 2y_ib + 2mx_ib + (mx_i)^2 + b^2 \\
&= \overline{y^2} + \overline{x^2}m^2 - 2\overline{xy}m + 2\overline{x}mb - 2\overline{y}b + b^2
\end{aligned}
$$

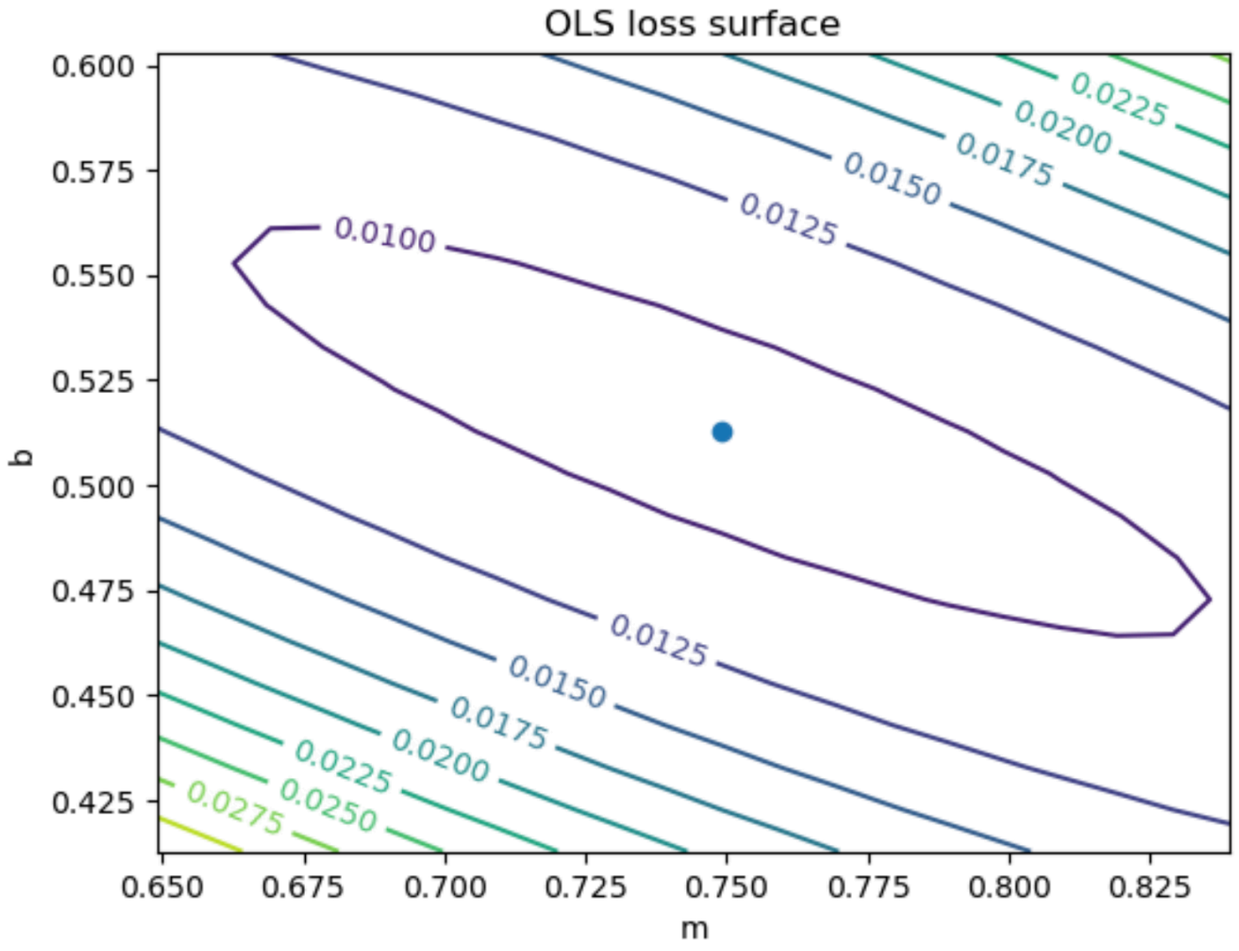


- $\boxed{L(m,b) = \overline{y^2} + \overline{x^2}m^2 - 2\overline{xy}m + 2\overline{x}mb - 2\overline{y}b + b^2}$ (where bar denotes the average)

Loss surface: Ordinary Least squares (OLS)

$$L(m,b) = \overline{y^2} + \overline{x^2}m^2 - 2\overline{xy}m + 2\overline{x}mb - 2\overline{y}b + b^2$$





The “canonical” deep learning training workflow

- **Data** → collect, clean, **supervise**, split (train/val/test), augment
  - Garbage in, garbage out: this is where most projects live or die
- **Model Architecture** → choose a model that well suited to the data
  - MLP (simple data), CNN (grids/Images), RNN/Transformer (sequences), GNN (graphs)
- **Objective** (prediction quality scoring mechanism) → define a loss $\mathcal{L}(\theta)$ that encodes "good"
  - Regression: Mean square error MSE
  - Binary classification: Binary Cross-entropy
  - Multi-class classification: Categorical Cross-entropy
- **Optimization** (learning) → back-prop + SGD/Adam, LR schedule, regularization
  - We are just rolling downhill on a very high-dimensional surface
- **Evaluation** → validation metrics, error analysis, ablations
  - Tune on val, touch test *once*
- **Deployment** → export, serve, monitor drift, iterate

Aside: "Canonical" means conforming to a standard, authoritative, or generally accepted form the version of something that serves as the reference point against which variants are judged (e.g., the canonical form of a matrix, or the canonical version of a text).

Ordinary least squares: closed form

Stack the n observations into a **design matrix** with a column of ones for the intercept:

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \in \mathbb{R}^{n \times 2}, \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \qquad \mathbf{y} \in \mathbb{R}^n$$

so that the model is simply $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$.

Loss (squared error, a quadratic form in $\mathbf{w}$):

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Gradient $\rightarrow$ set to zero $\rightarrow$ **normal equations**:

$$\nabla_{\mathbf{w}} \mathcal{L} = -2\,\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) = \mathbf{0} \quad \Longrightarrow \quad \mathbf{X}^\top \mathbf{X}\,\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

$$\boxed{\hat{\mathbf{w}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}}$$

Analytic solution

- Using the loss surface above, we can analytically obtain the solution to the OLS problem
- OLS is simply enough that we can apply analytic optimization techniques, similar to the previous slide
- The global minima of $L(m, b) = \overline{y^2} + \overline{x^2}m^2 - 2\overline{xy}m + 2\overline{x}mb - 2\overline{y}b + b^2$ satisfy the condition

$$\nabla L(\mathbf{w}) = \left(\frac{\partial L}{\partial m}, \frac{\partial L}{\partial b} \right) = (0, 0)$$

- gonna do the derivation here because it is part of your homework but it is quite straightforward
- This forms a system of two equations which can be solved algebraically
- Which yields the closed form analytical solution for the best fit parameters

$$m_{opt} = \frac{Cov(x_i, y_i)}{var(x_i)} \qquad b_{opt} = \overline{y} - m_{opt}\overline{x} = \overline{y} - \overline{x}\frac{Cov(x_i, y_i)}{var(x_i)}$$

Looking ahead:

$$y_{pred} = M(x|\mathbf{w}) = M(x|m,b) = mx + b$$

new value = old value – step

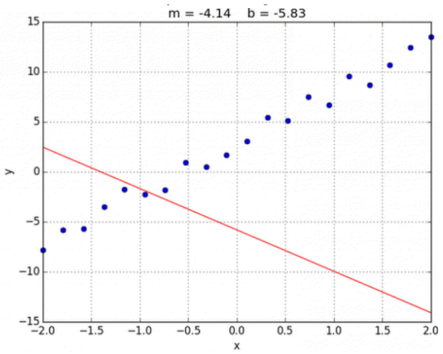
$$\mathbf{w}_{\text{next}} \leftarrow \mathbf{w}_{\text{current}} - \eta \nabla_{\mathbf{w}} L$$

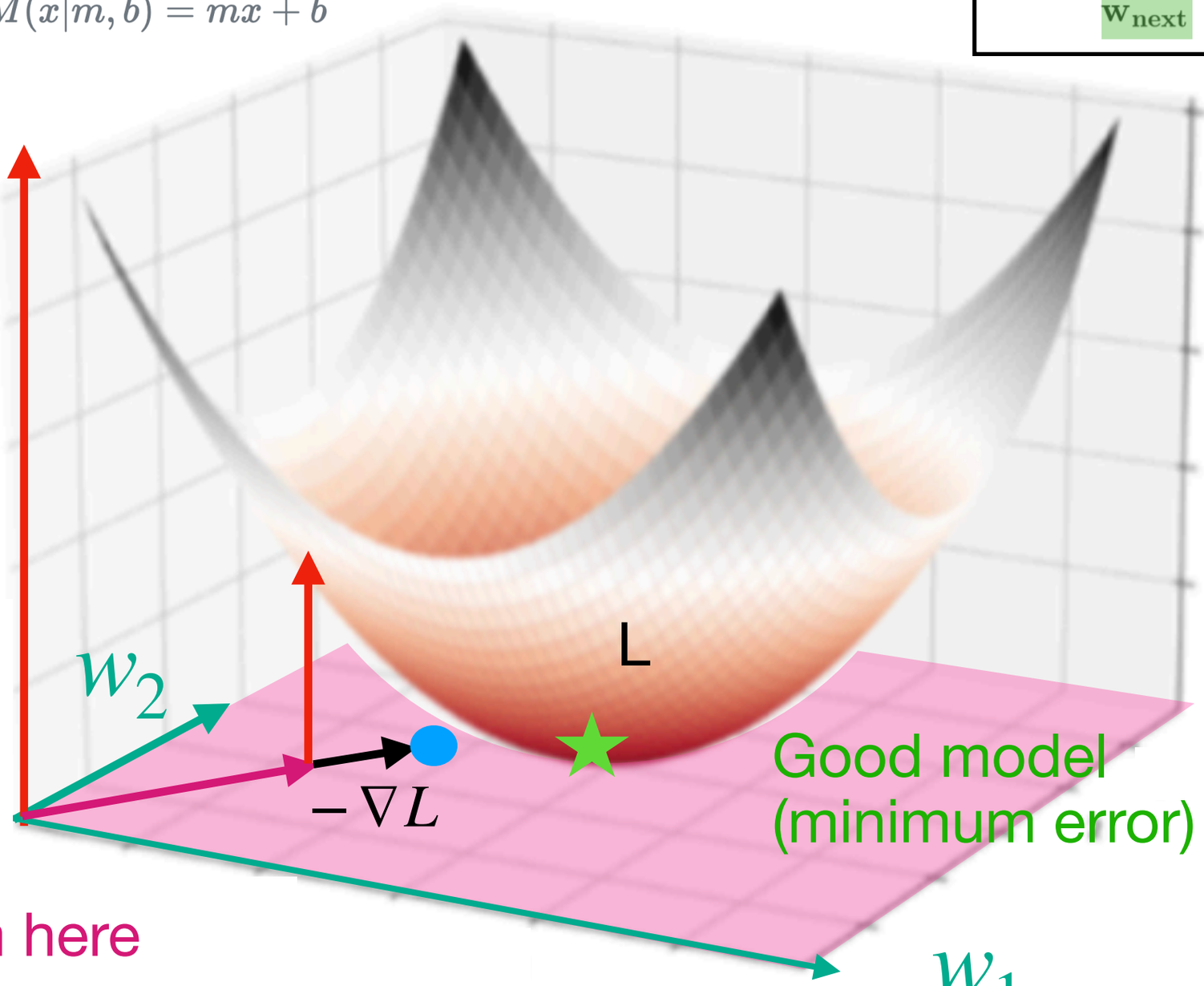


Model error

$l = l(w_1, w_2)$

(Loss)



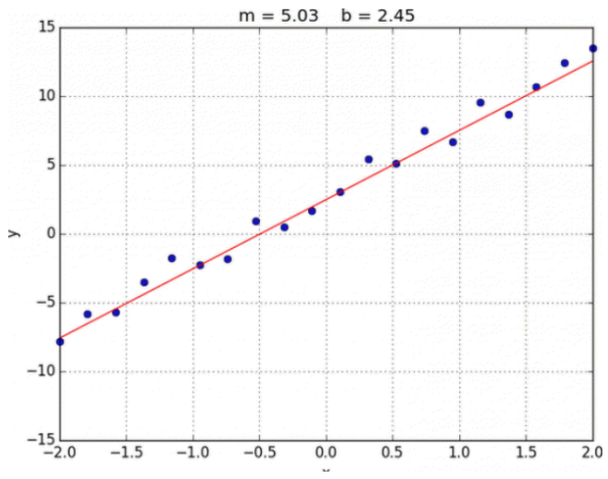


Imagine I'm here

$\mathbf{w} = (w_1, w_2) = (-4.1, -5.8)$

Model parameter space

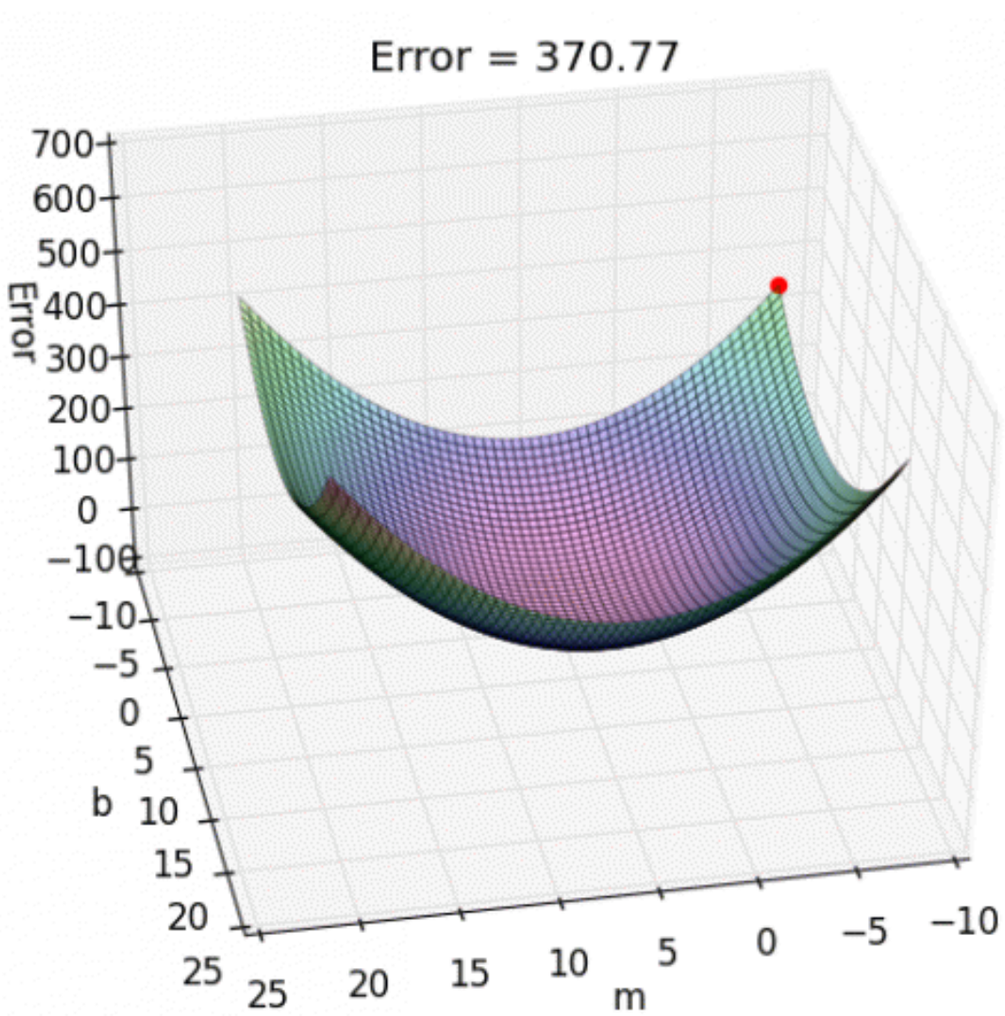
$\mathbf{w}^* = (5\ , 2.5)$

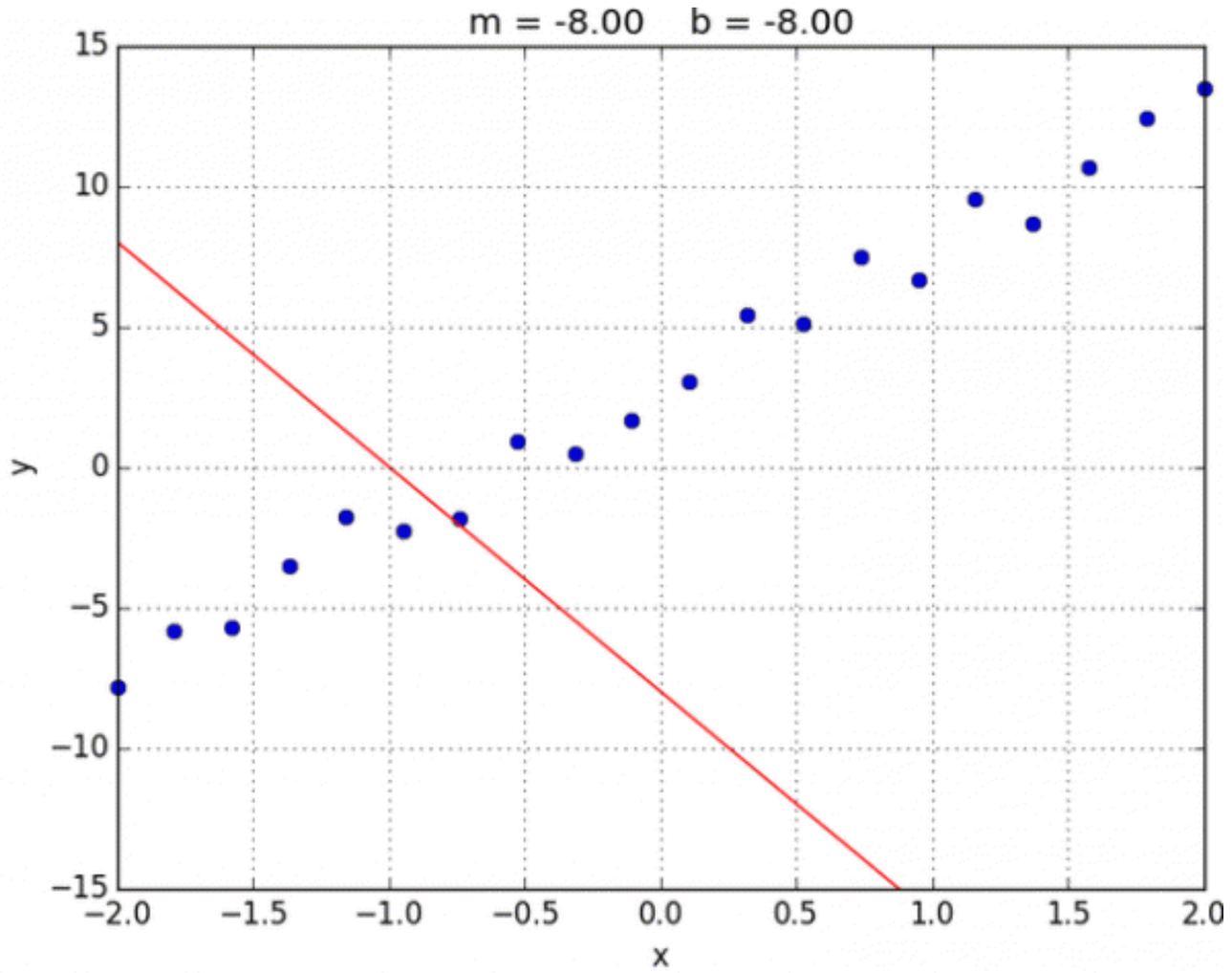


Computer calculates gradient vector for us via back-propagation (don't ask)

$$-\nabla L(\mathbf{w}) = -(-0.5, -0.5) = (0.5, 0.5)$$

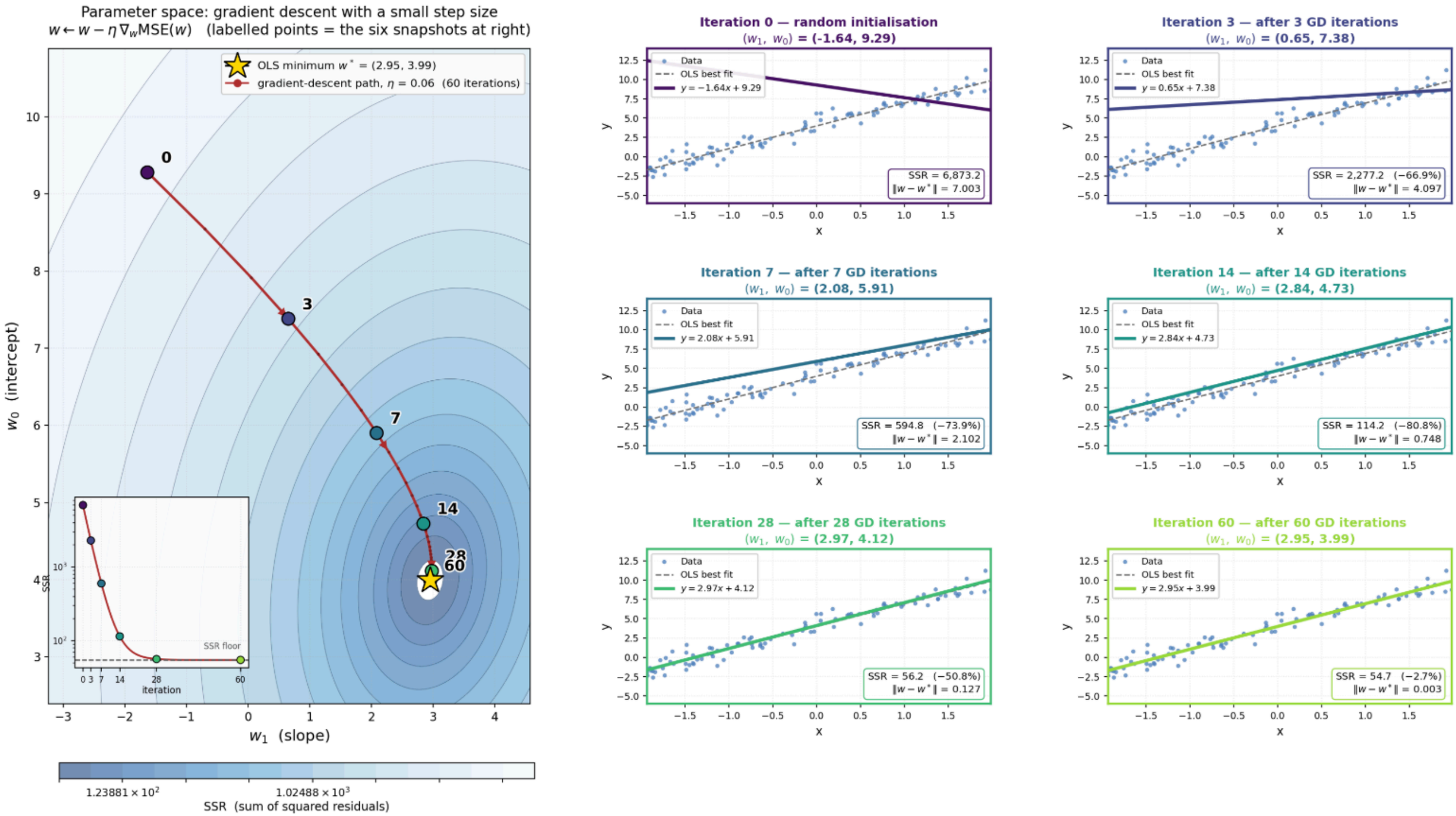
Gradient descent applied to linear regressor:



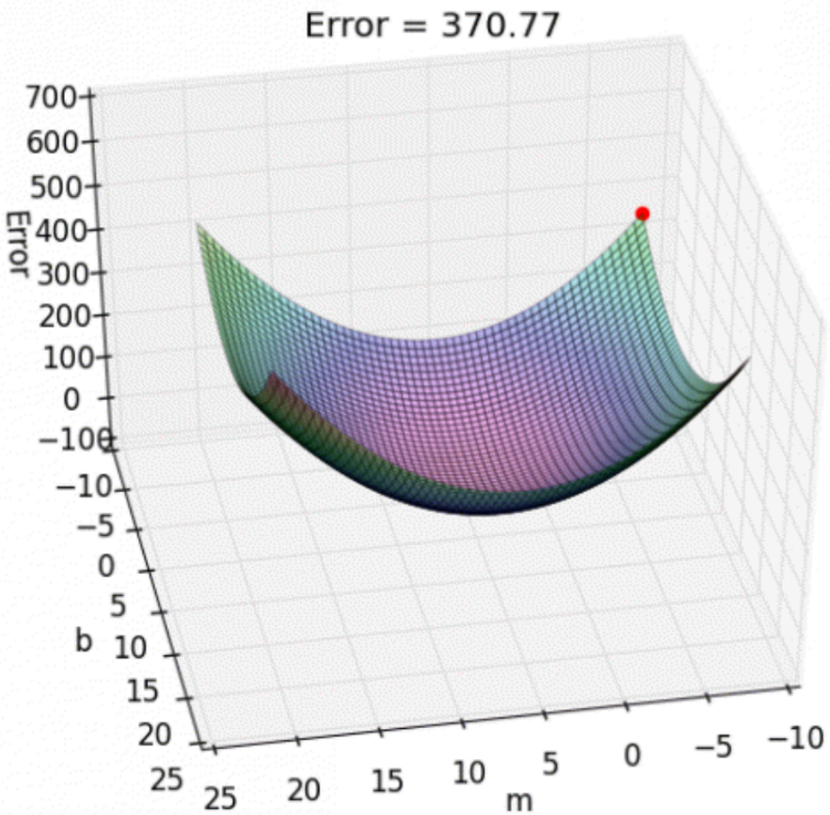


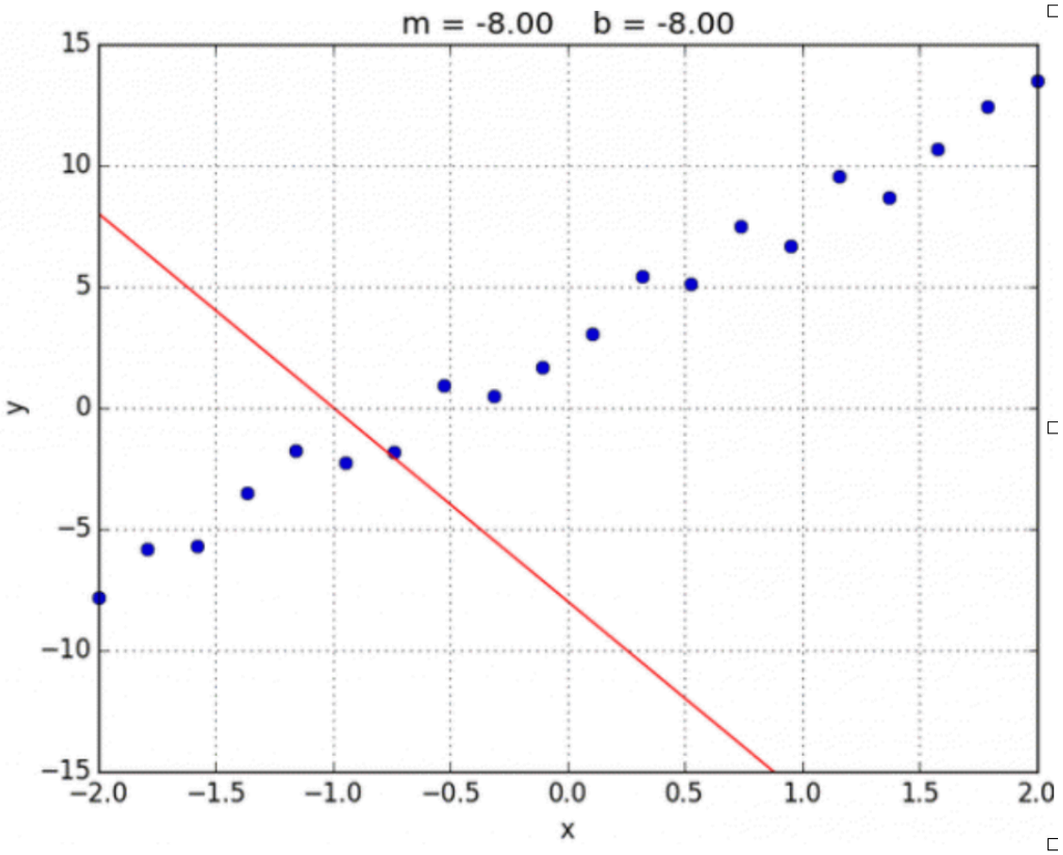
Parameter space makes a search space that allows us to systematically search for better and better models (better parameterizations)

Gradient descent applied to linear regressor:

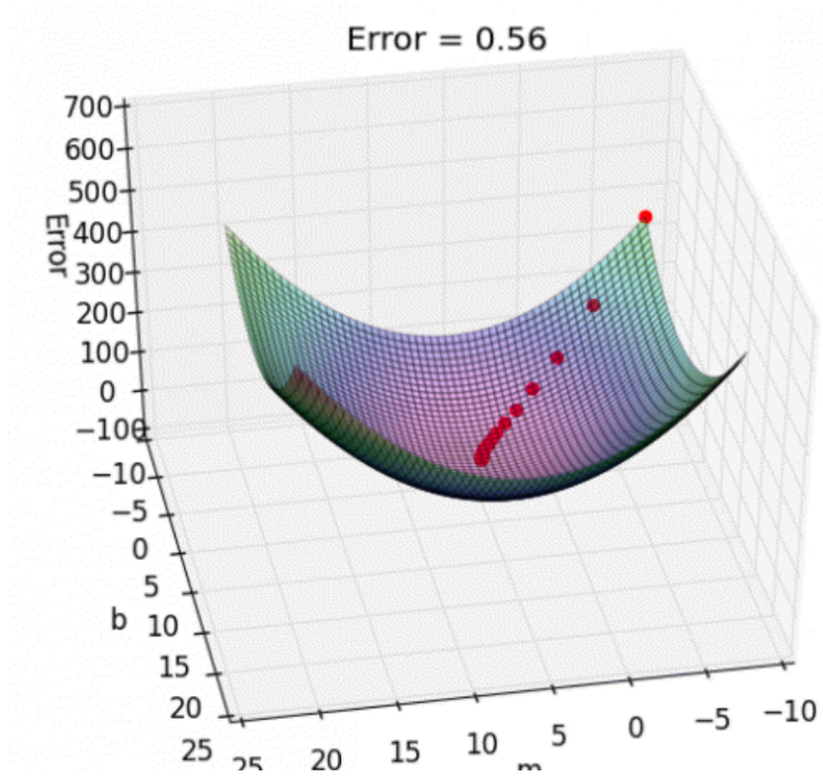


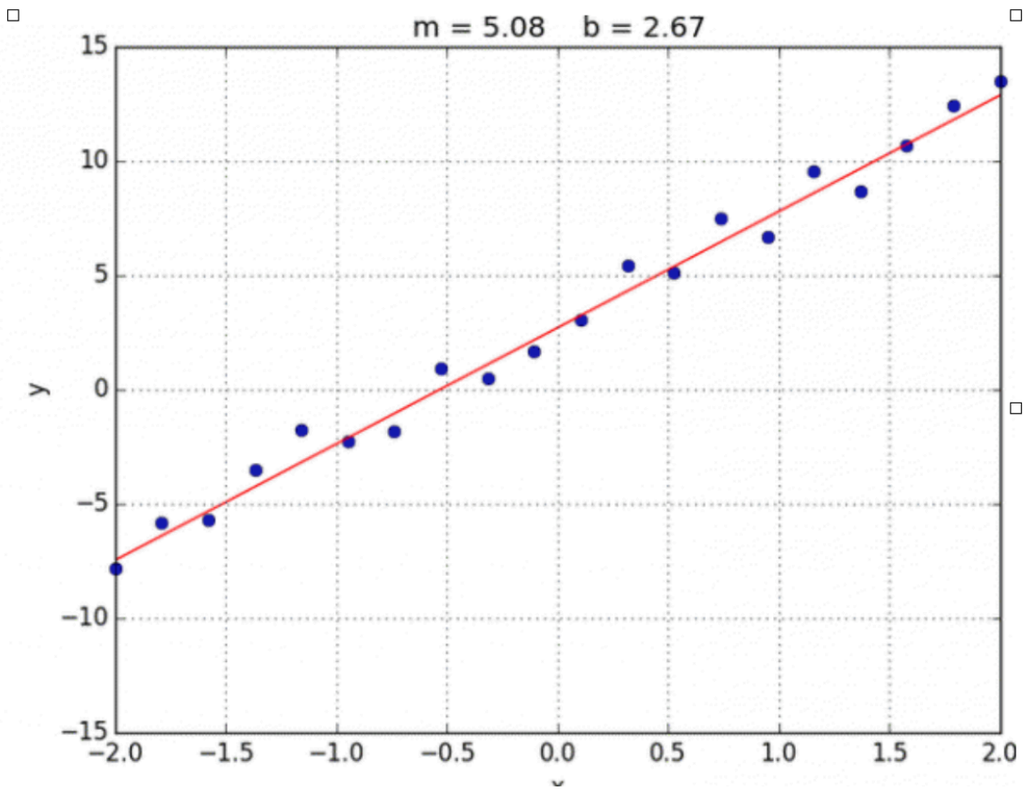
Random initial parameter guess:



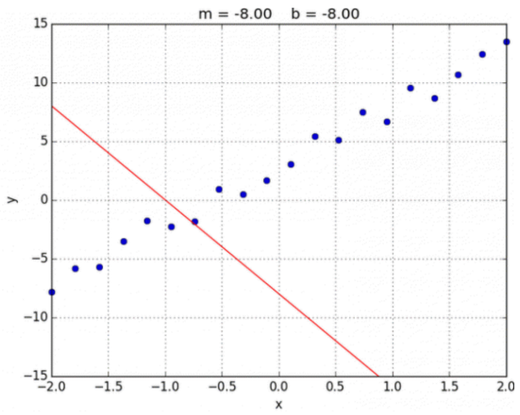


Final solution after optimization





Training loop for a model



$\mathbf{w}_{t=0} \rightarrow$ e.g. np.rand() $w = 0.1, b = 0.4$

Single training run:

Random parameters guess

Input X
(Training data features)

Parametric Model

$x \xrightarrow{w} (b) \longrightarrow \hat{y} = m(x \mid \mathbf{w}) = w_1 x + w_0 = mx + b$ (2 parameters)

model prediction at iteration t for i^{th} training sample
for parameterization $\mathbf{w}_t$ at t^{th} iteration of GD

prediction $= \hat{y}_t^i = m(\mathbf{x}_i \mid \mathbf{w}_t)$ evaluate

(Repeat)

True targets Y
(Training data targets)

Predictions $\hat{y}_t$

Exit criteria
(convergence or early stopping)

Trained model
$\mathbf{w}^*$ = optimal model parameters

Sum of square error (SSE)

$$SSE = \sum_i^N \epsilon_i^2 = \sum_i^N (y_i - \hat{y}_i)^2$$

Loss function
Compare targets and predictions

$\mathbf{w}_{t+1}$

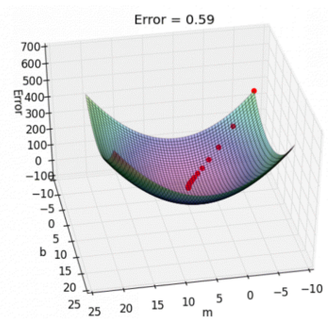
Updated Parameters

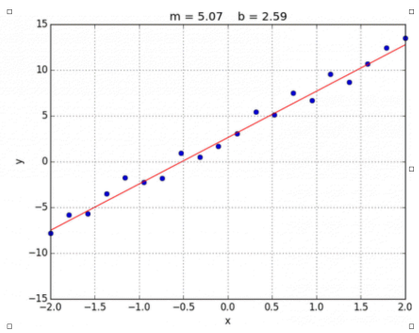
(*sum or mean*) reduction

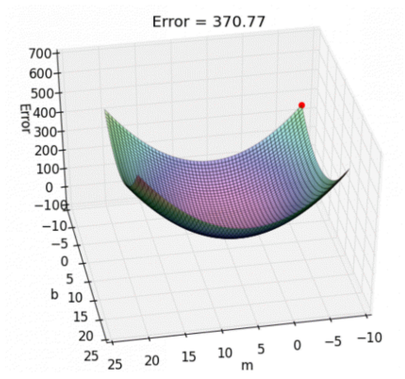
Training loop
(iterations of gradient descent)

Monitor

Loss $L_t = L(\mathbf{w}_t)$







Loss score
(AKA loss surface)

Optimizer

The gradient vector $\nabla_w L\big|_{\mathbf{w}_t}$,at the t^{th} time-step, is computed numerically via back-propagation (i.e. automatic differentiation) using specialized libraries such as Tensor-flow or PyTorch. Alternatively the gradient can also be approximated via finite difference in limited simplistic cases

Gradient descent update rule

Vector equation (updates all fitting parameters)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \nabla_w L \Big|_{\mathbf{w}_t}$$

λ = learning rate

(For the j^{th} fitting parameter) $\longrightarrow$

$$w_{t+1}^j = w_t^j - \lambda \frac{\partial L}{\partial w^j}\Big|_{w_t^j}$$

$\longrightarrow$ Read vertical line as "Evaluated at"

Deep learning software tools

MOST POPULAR TOOLKITS

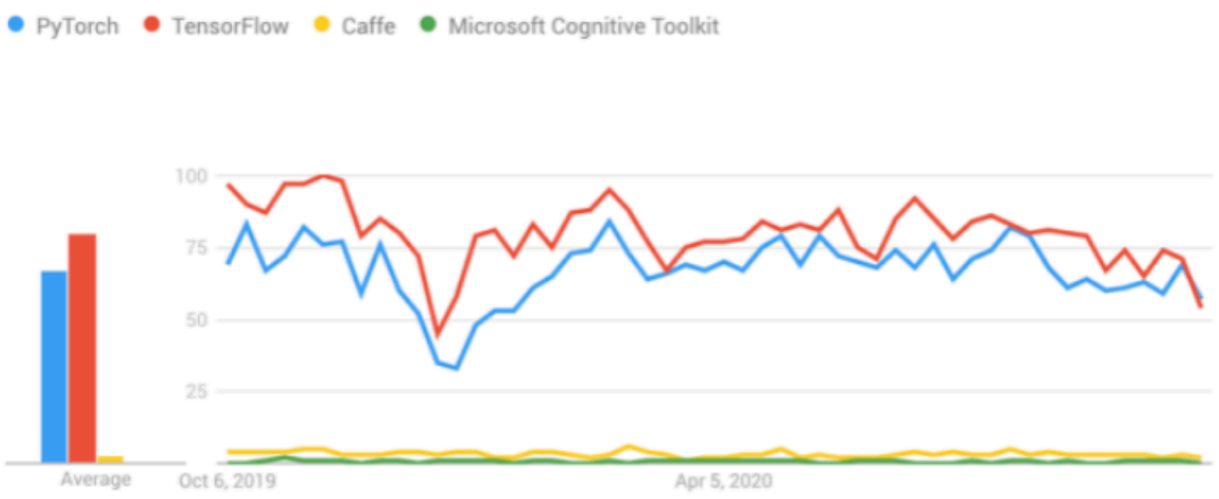
- **TENSOR-FLOW**
- **PYTORCH**
- **THEANO (MILA U-Montral)**
- **COGNITIVE TOOLKIT CNTK**
  - **(MICROSOFT)**

Keras

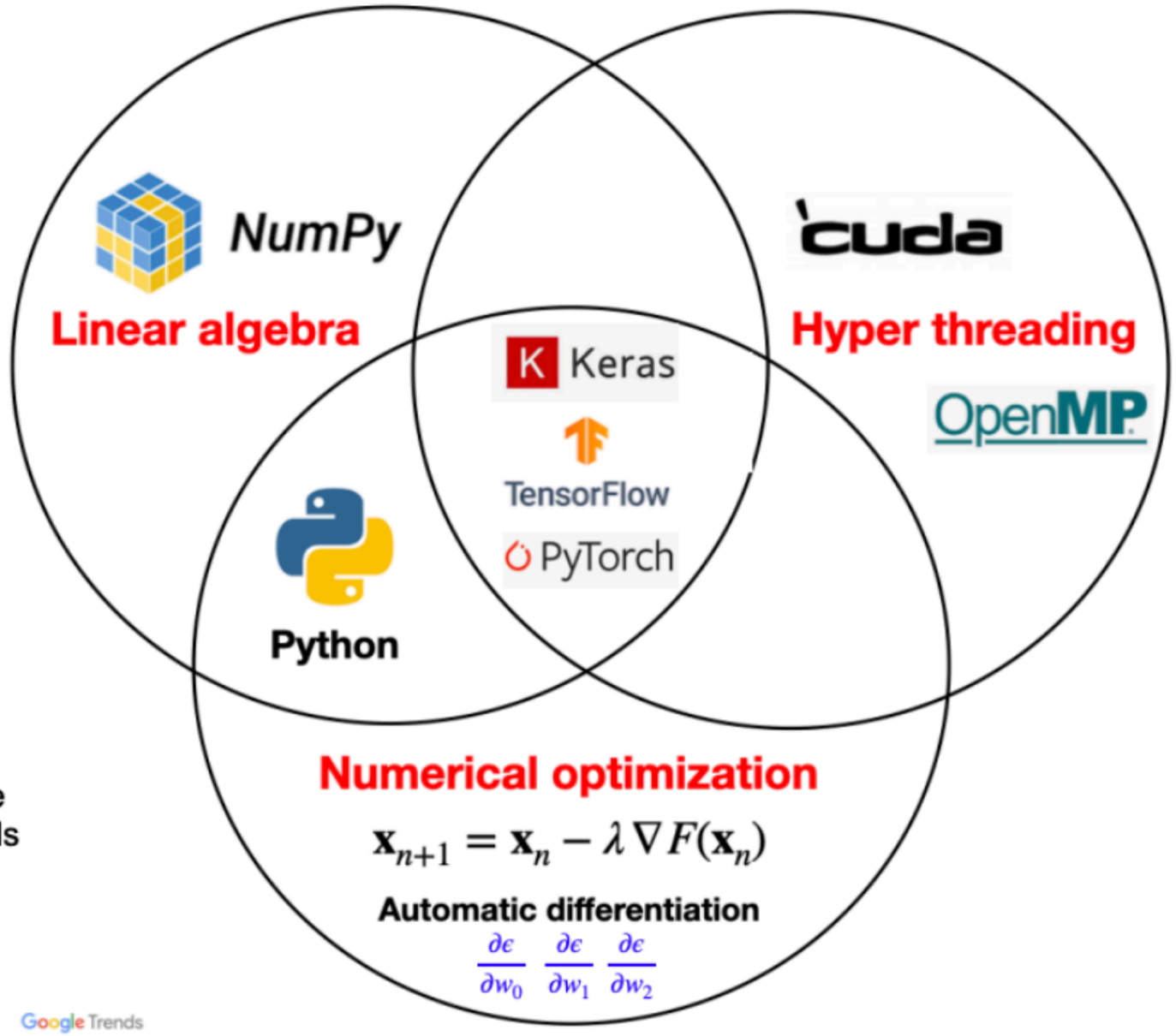
- Keras is an **open-source** software library
- **High-level API**: Application Programming Interface
- Provides a **python interface** for deep learning tools
- **Runs on top of TensorFlow,**
  - or Microsoft CNTK, or Theano
- **Easy to use**

Interest over time

Google Trends



United States. Past 12 months. Web Search. https://www.tooploox.com/blog/pytorch-vs-tensorflow-a-detailed-comparison

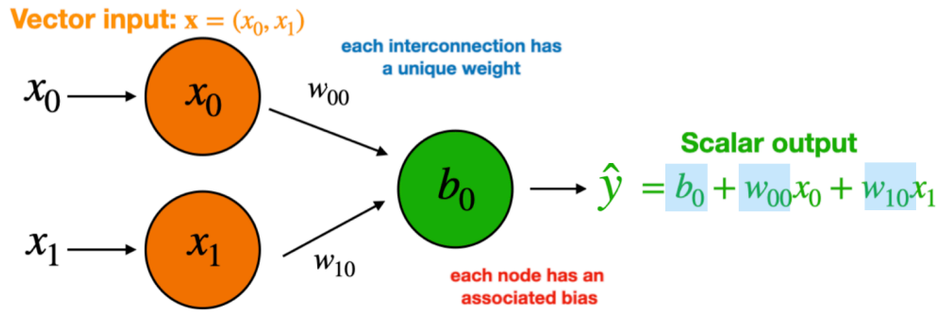


Keras + TensorFlow

PyTorch

- provide exposure
- Add extra line to vour resume

Dimensionality of the optimization space = number of fitting parameters

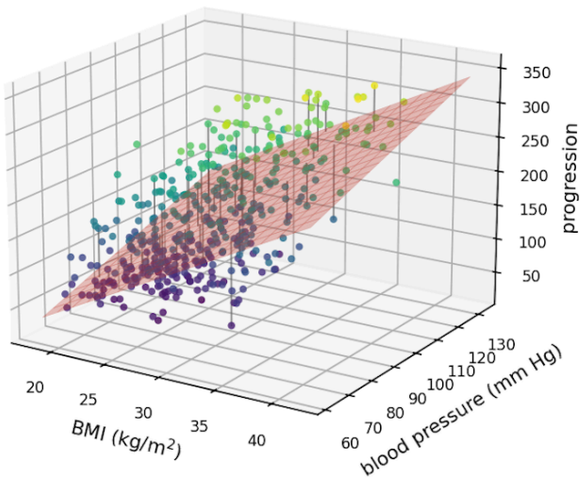


- **Number of fitting parameters: 3**

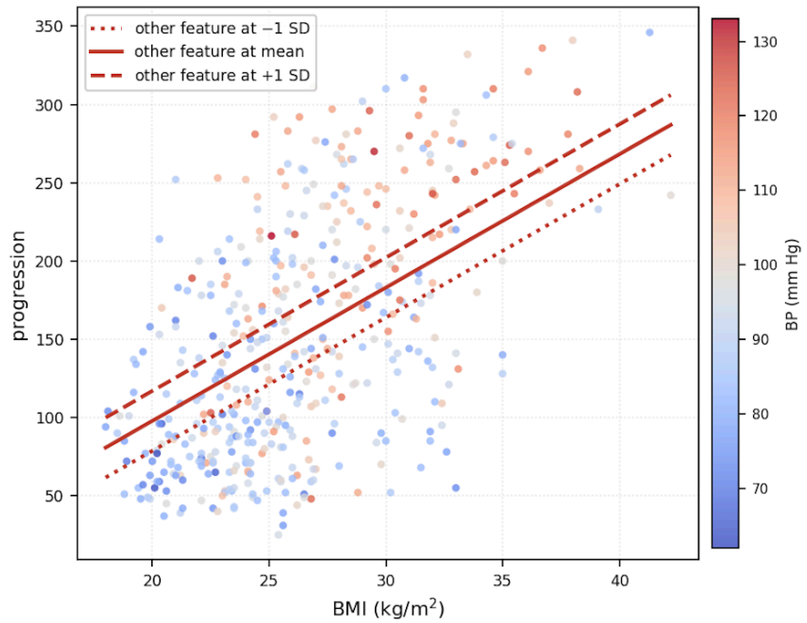
A three-parameter loss space: progression $\approx w_0 + w_1 \cdot \mathrm{BMI}_z + w_2 \cdot \mathrm{BP}_z$ (diabetes study, n = 442)

SSR(w^*) = 1,583,105 | R^2 = 0.396 | w^* = (152.1, 37.6, 19.1) | standard errors = (2.9, 3.1, 3.1) | corr(w_1, w_2) = -0.40

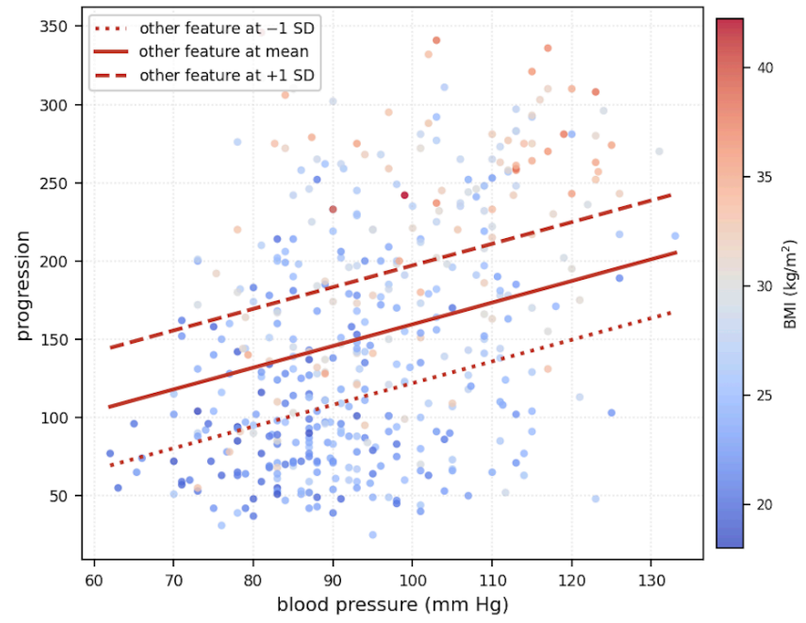
(a) Data cloud + fitted plane
n = 442; grey sticks = residuals, SSR sums their squares



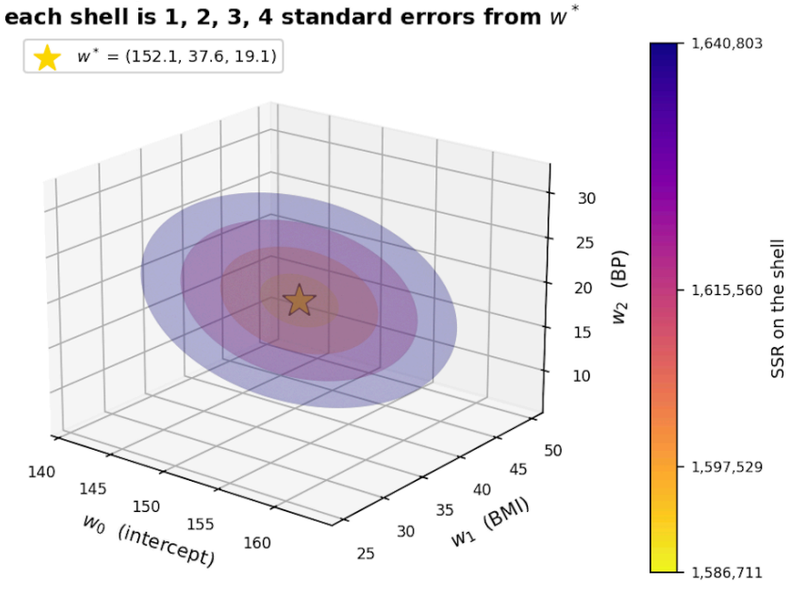
(b) The same plane, sliced along BMI
colour = the other feature



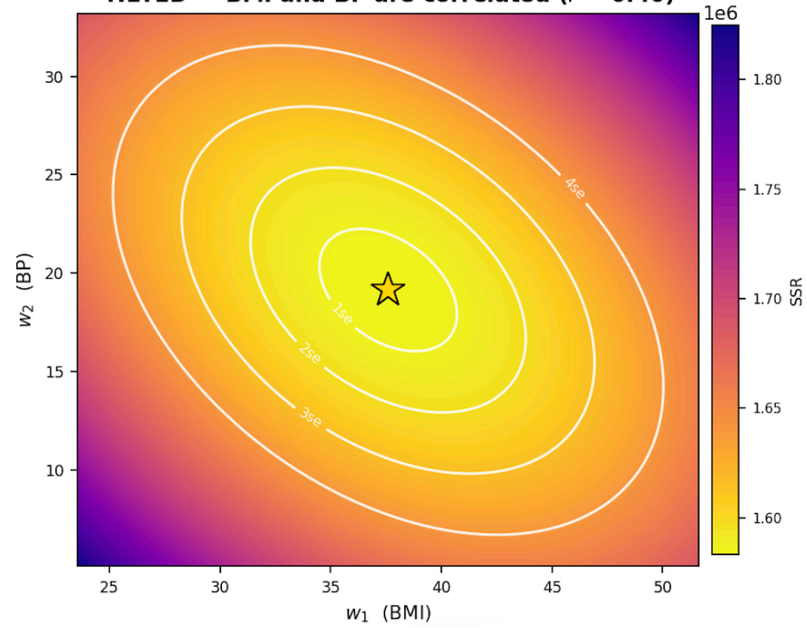
(c) The same plane, sliced along blood pressure
colour = the other feature



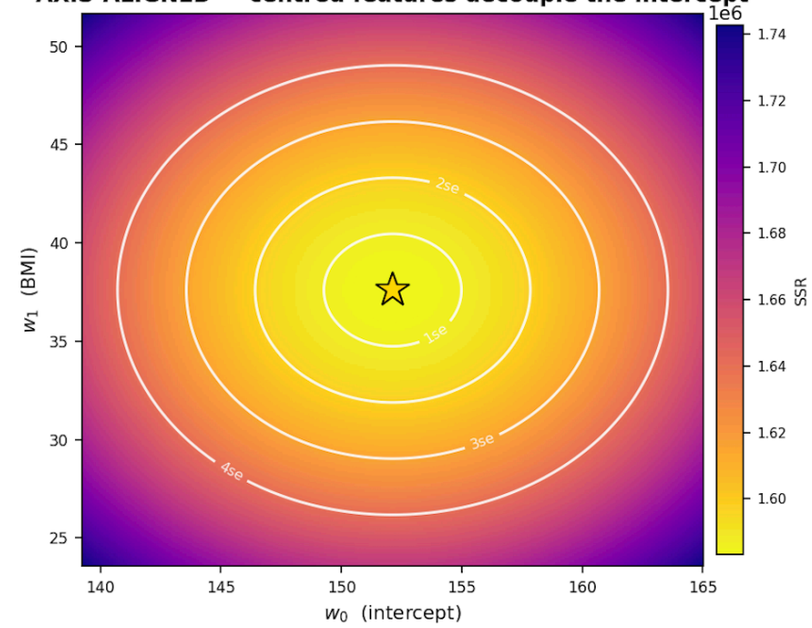
(d) Loss space is 3-D: nested SSR isosurfaces
each shell is 1, 2, 3, 4 standard errors from w^*



(e) Slice at $w_0 = w_0^*$: the (w_1, w_2) plane
TILTED — BMI and BP are correlated (r = 0.40)



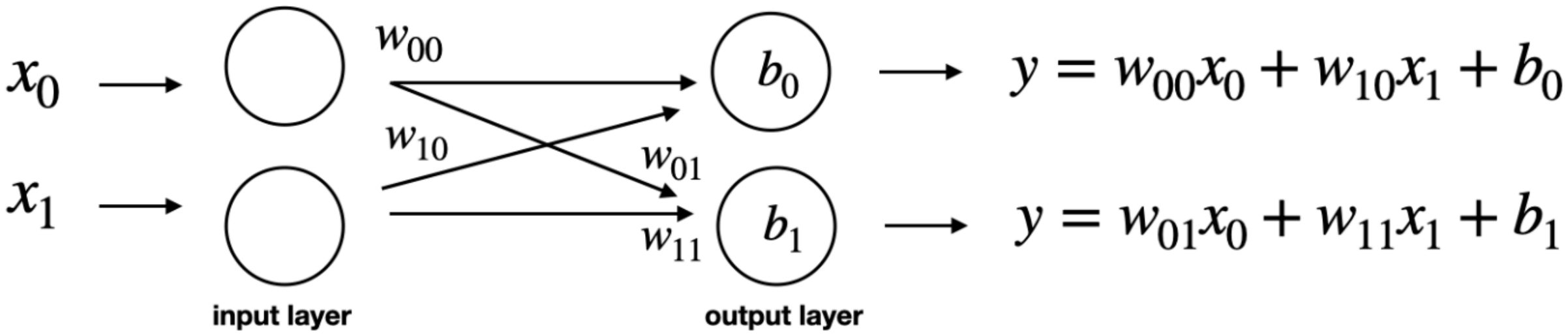
(f) Slice at $w_2 = w_2^*$: the (w_0, w_1) plane
AXIS-ALIGNED — centred features decouple the intercept



Linear vector regression:

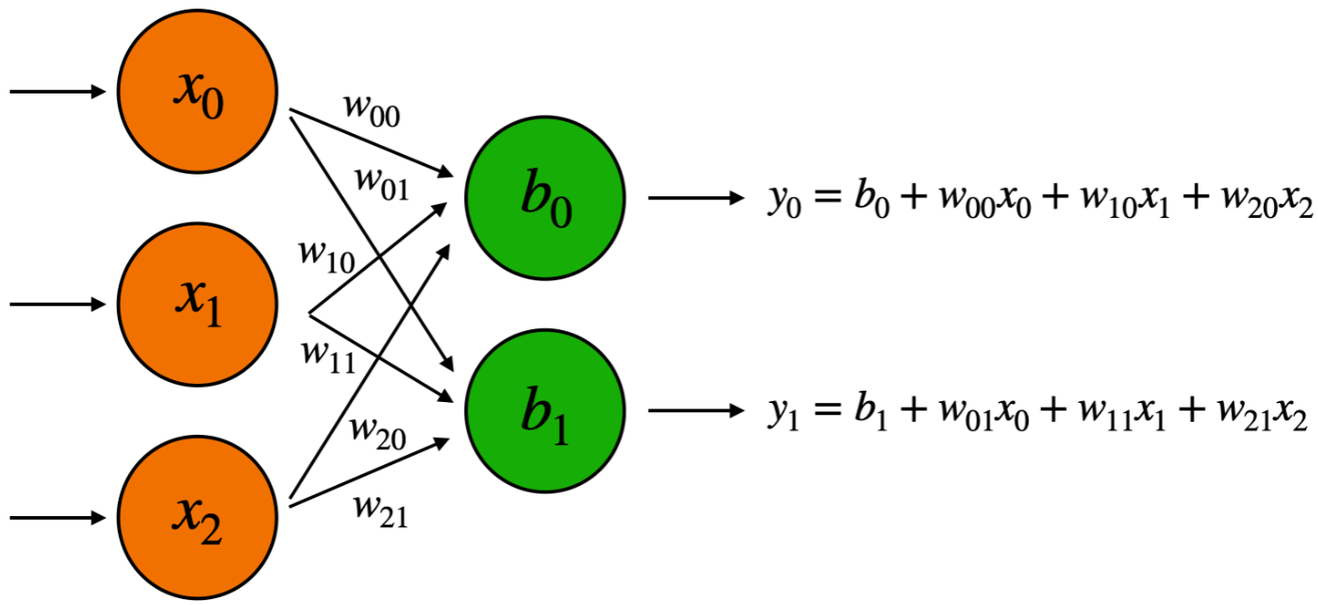
- **Problem Type:** Vector-regression
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^2}$
- **Hidden layers**: None
- **Hidden-layer activation**: N/A
- **Output-layer activation**: linear
- **Number of fitting parameters**: 6

Computational graph



Scaling up: We can easily extend the problem by considering more dimensions.

	input space: $\mathbb{R}^3$			output space: $\mathbb{R}^2$	
i	**sqft** $x_1 \cdot 1000\ ft^2$	**age** $x_2 \cdot years$	**beds** $x_3 \cdot count$	**price** $y_1 \cdot \$1000s$	**rent** $y_2 \cdot \$/month$
0	1.45	12	3	365.4	2150
1	0.98	45	2	219.8	1480
2	2.10	3	4	498.6	2890
3	1.32	28	3	308.1	1975
4	2.85	8	5	625.7	3420
5	1.70	35	3	366.2	2240



For a single input feature vector $\mathbf{x}^T = (x_0, x_1, x_2)$ the model is

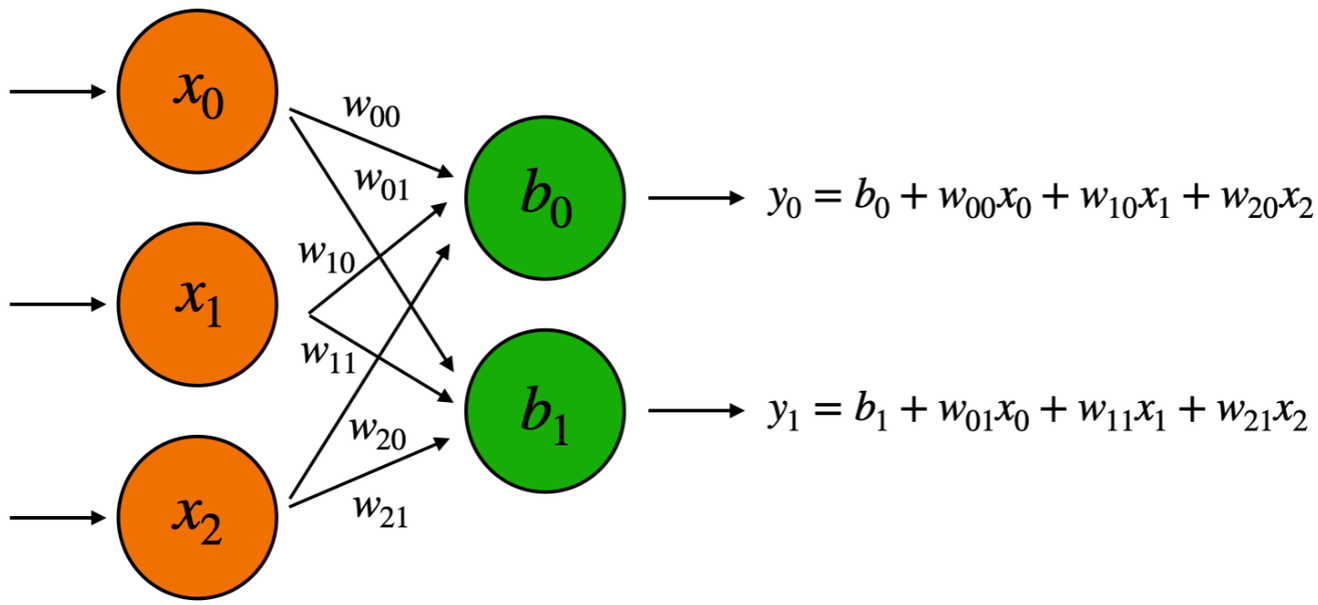
$$\mathbf{y}_{pred} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}_{pred} = \mathbf{m}(\mathbf{x}|\mathbf{w}, \mathbf{b}) = \mathbf{b} + \mathbf{w}\mathbf{x} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} + \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

In which case the model can be rewritten in a mathematically identical form

$$\mathbf{y}_{pred} = \mathbf{m}(\tilde{\mathbf{x}}|\tilde{\mathbf{w}}) = \tilde{\mathbf{w}}\tilde{\mathbf{x}} = \begin{pmatrix} b_0 & w_{00} & w_{01} & w_{02} \\ b_1 & w_{10} & w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

Scaling up: We can easily extend the problem by considering more dimensions.

	input space: $\mathbb{R}^3$			output space: $\mathbb{R}^2$	
i	**sqft** $x_1 \cdot 1000\ ft^2$	**age** $x_2 \cdot years$	**beds** $x_3 \cdot count$	**price** $y_1 \cdot \$1000s$	**rent** $y_2 \cdot \$/month$
0	1.45	12	3	365.4	2150
1	0.98	45	2	219.8	1480
2	2.10	3	4	498.6	2890
3	1.32	28	3	308.1	1975
4	2.85	8	5	625.7	3420
5	1.70	35	3	366.2	2240



For a single input feature vector $\mathbf{x}^T = (x_0, x_1, x_2)$ the model is

$$\mathbf{y}_{pred} = \begin{pmatrix} y_0 \\ y_1 \end{pmatrix}_{pred} = \mathbf{m}(\mathbf{x}|\mathbf{w}, \mathbf{b}) = \mathbf{b} + \mathbf{w}\mathbf{x} = \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} + \begin{pmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

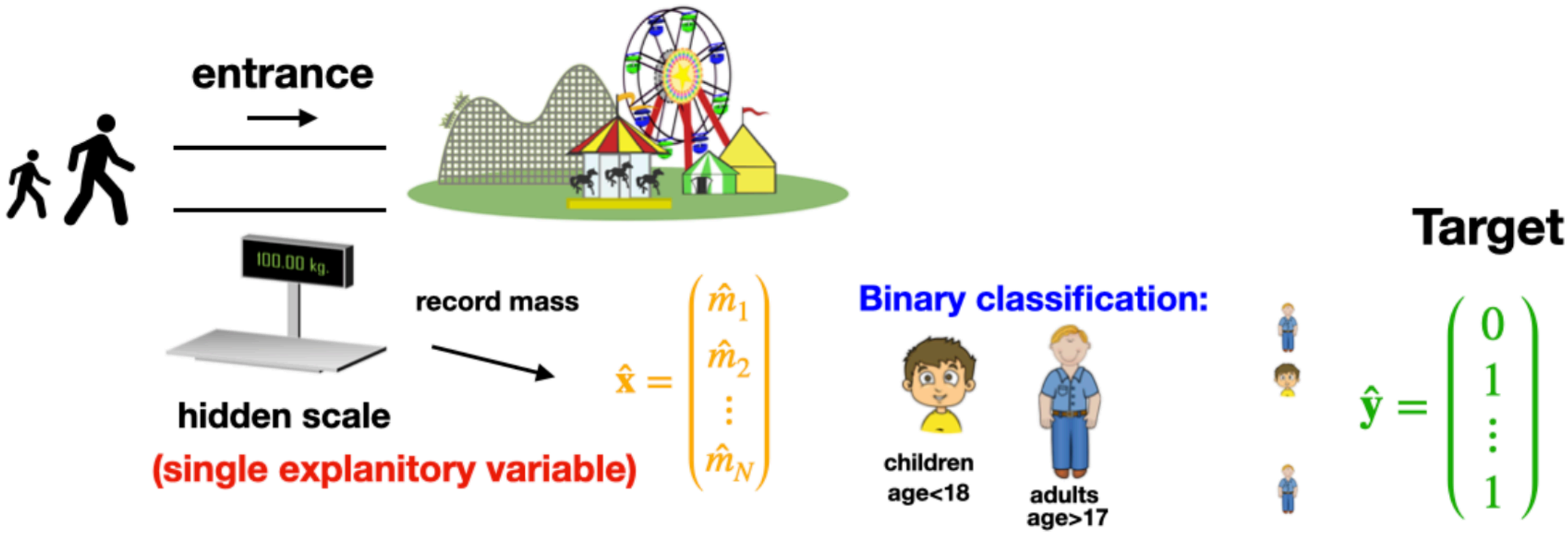
In which case the model can be rewritten in a mathematically identical form

$$\mathbf{y}_{pred} = \mathbf{m}(\tilde{\mathbf{x}}|\tilde{\mathbf{w}}) = \tilde{\mathbf{w}}\tilde{\mathbf{x}} = \begin{pmatrix} b_0 & w_{00} & w_{01} & w_{02} \\ b_1 & w_{10} & w_{11} & w_{12} \end{pmatrix} \begin{pmatrix} 1 \\ x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

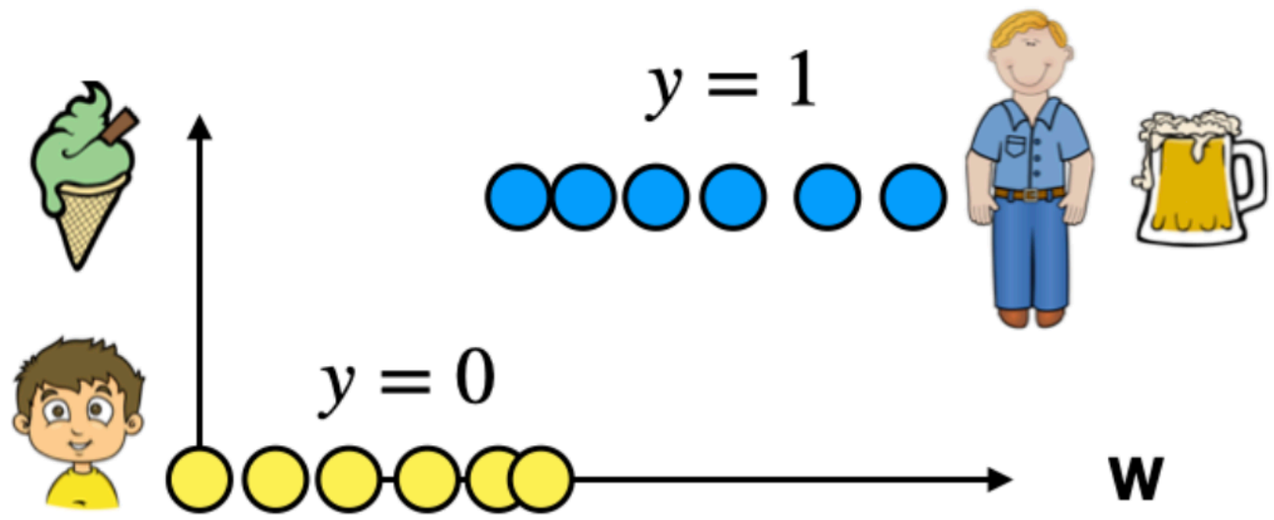
Generalizing the model as a classifier (Logistic regression)

Example: Dataset

- Consider the following scenario at an amusement park gathering data.
- **Goal:** Classify qualitative age y (0=child vs 1=adult) based on weight x descriptor
  - This data could be used to help predict inventory prediction (ice-cream vs beer).

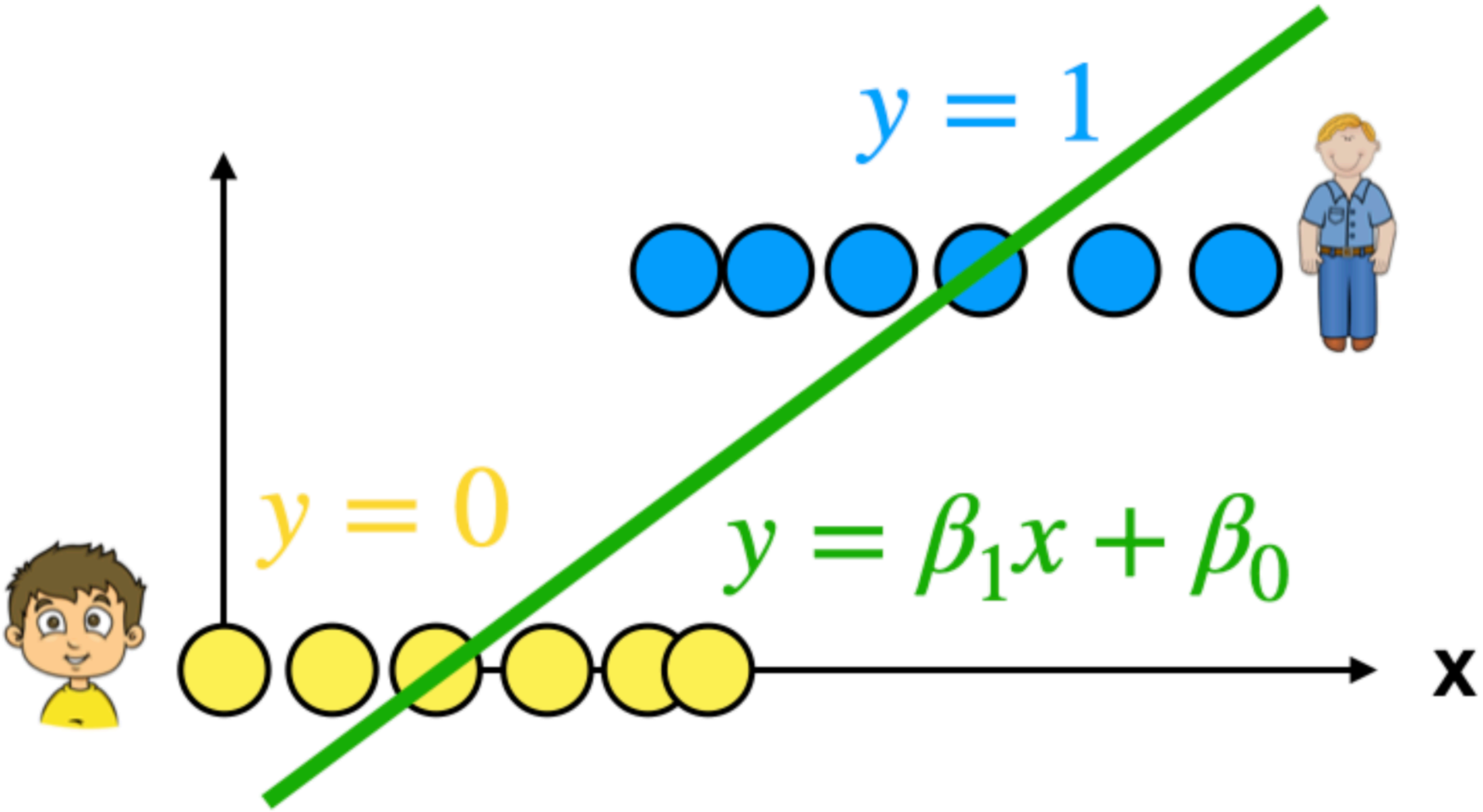


Make prediction: given an arbitrary mass, is it adult or child?



Linear regression

- A logical **question** is ... can we simply perform a linear regression of Y on X and classify as Yes if $\hat{Y} > 0.5$?



- Since in the population $E(Y \mid X = x) = \Pr(Y = 1 \mid X = x)$, we might think that regression is perfect for this task.
- However, linear regression might produce probabilities less than zero or bigger than one. Also, the transition region is clearly non-linear

Logistic function

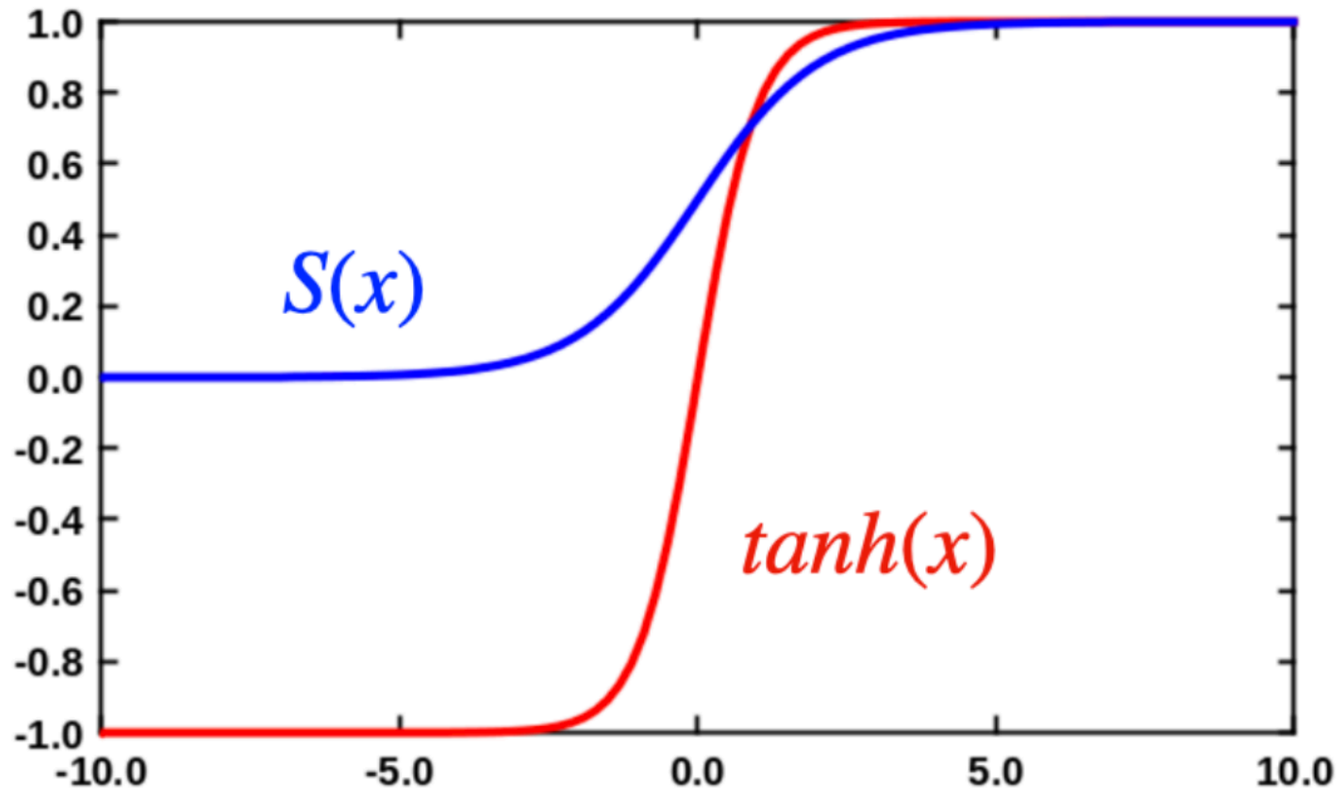
- A much better option, is to model the class-probability of y (0=child vs 1=adult) as a non-linear logistic function

Sigmoid (aka. logistic) function

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

Hyperbolic Tangent function

$$\tanh(x) = 2\left(S(2x) - 0.5\right)$$



Parameterized

$$y = S(x\,|\,\mathbf{p}) = \frac{1}{1+e^{-(wx+b)}}$$

$$\mathbf{p} = (w, b)$$

Logistic regression

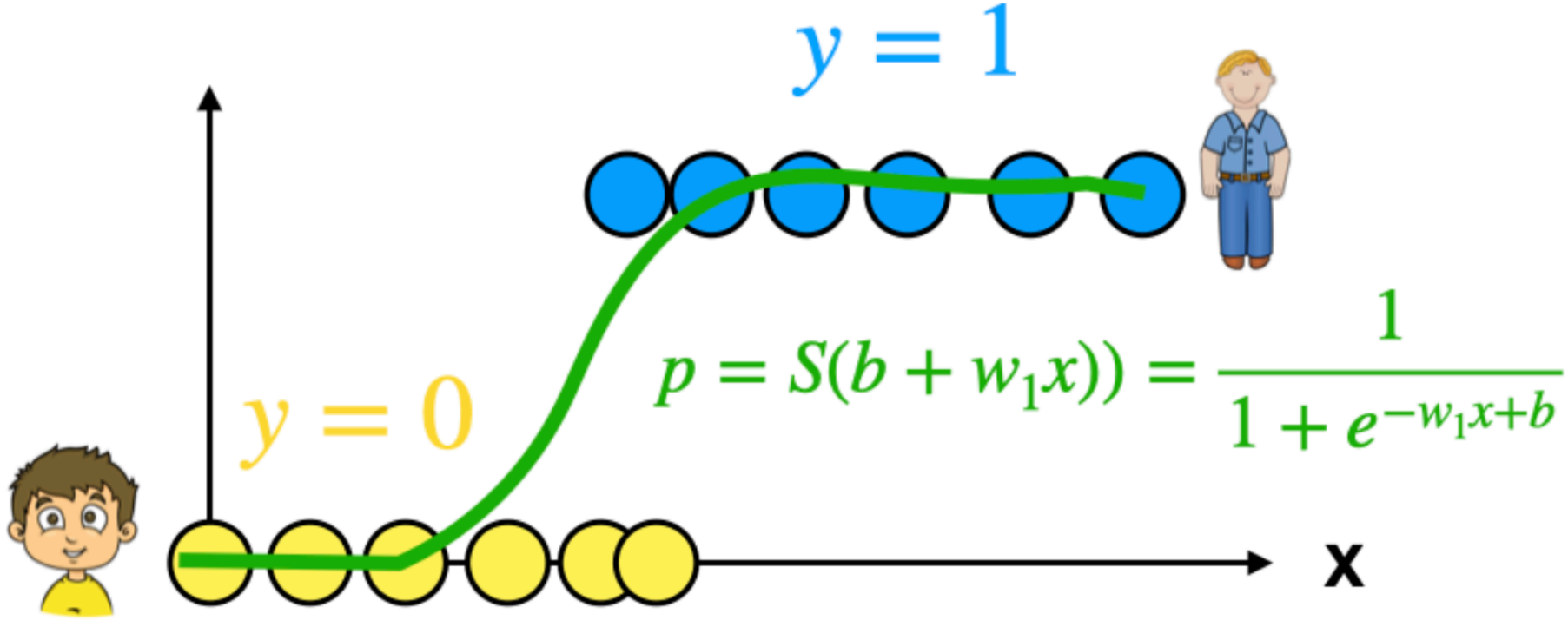
- It is common to assume a `linear logit function` in binary logistic regression .
- In this case the class-probability obeys a sigmoid (logistic function)

$$\ell = \log_e \frac{p}{1-p} = b + w_1 x$$ log-odds (also called logit)

$$\text{odds} \quad \frac{p}{1-p} = e^{b+w_1 x} \longrightarrow$$

probablility that outcome=true (Y=1)

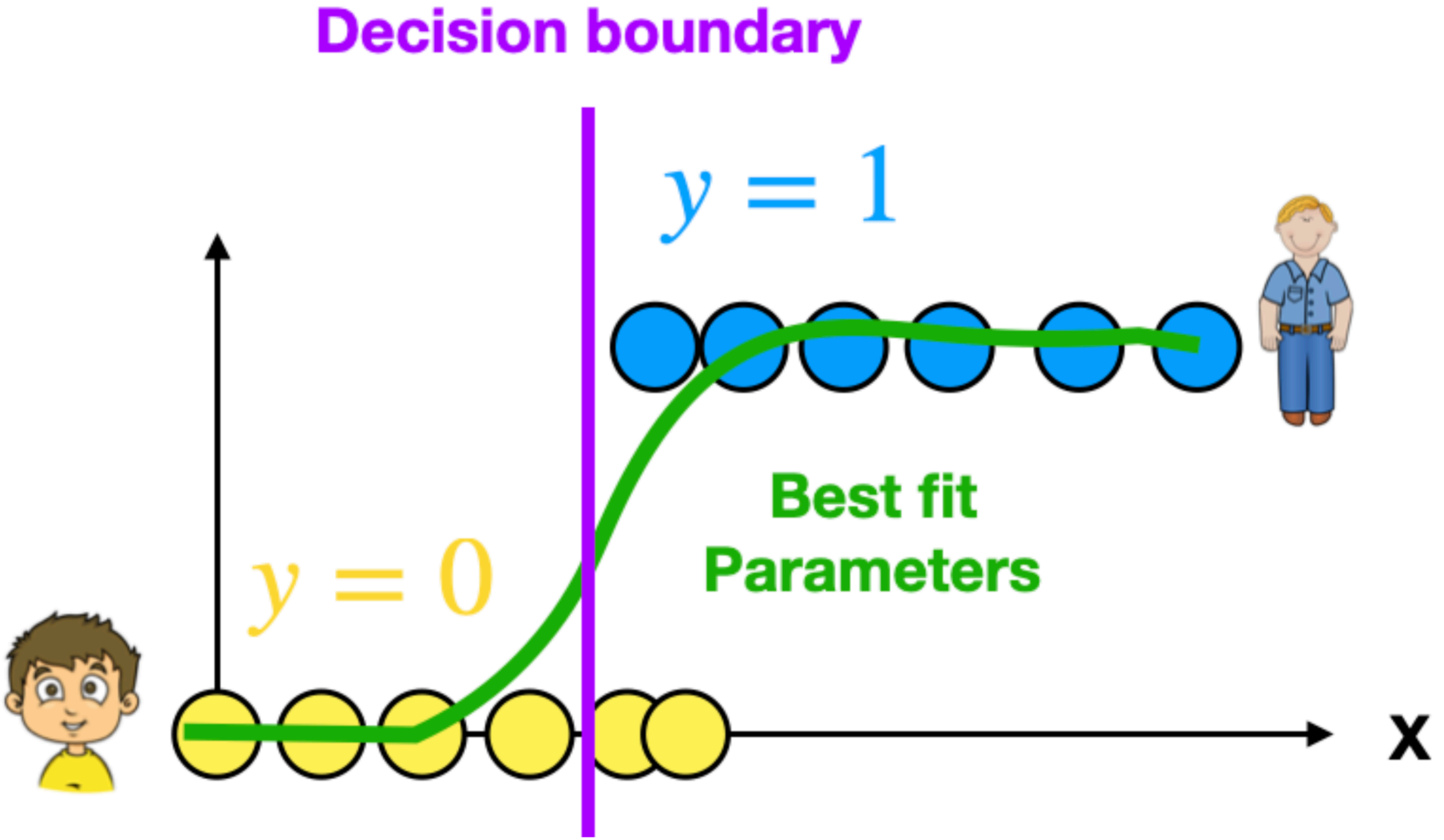
$$p = S(b + w_1 x)) = \frac{1}{1 + e^{-w_1 x + b}}$$



- **Important**: Even though probability $p(x)$ is non-linear, we still call this a `linear logistic regression model`, because the is assumed to be linear $logit = w_1 x + b$

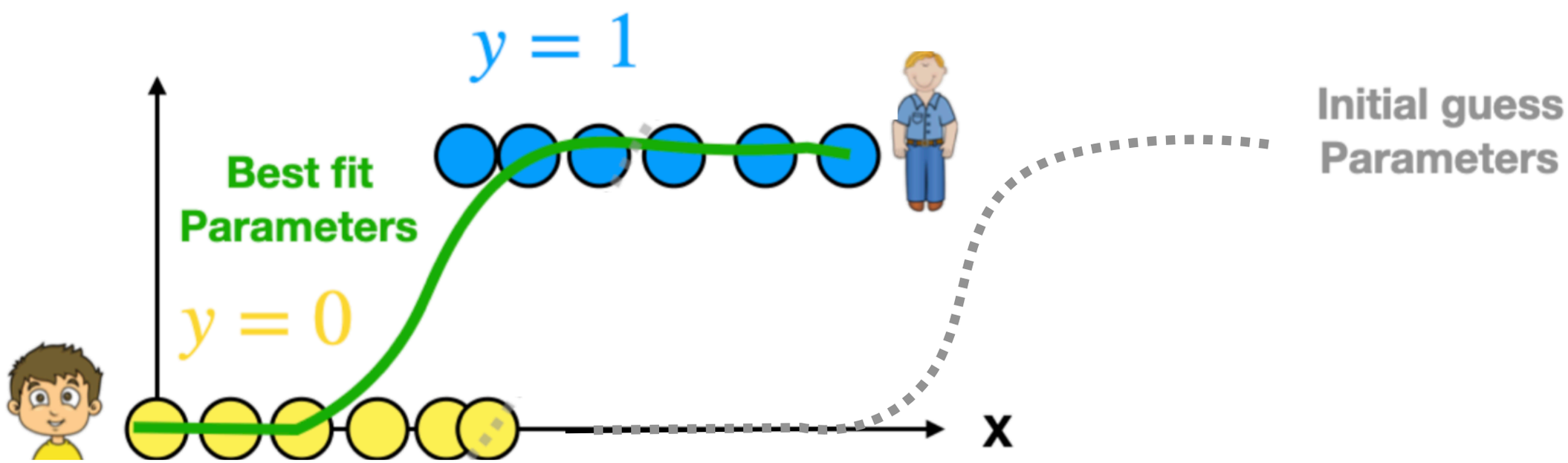
Decision boundary

- A decision boundary is a line or surface that separates data points of different classes in a classification problem.
- In `linear logistic regression model`, we have a linear decision boundary,
- This is typically defined where $p = 0.5$



Model fitting-1

- In theory, we could fit this using numerical optimization, similar to in module-1.
- The model is simple enough that we can do it analytically (computationally faster)



(weight and bias)

Model: $M(x \mid \mathbf{w}) = \sigma(w_1 x + w_0)$ $\mathbf{w} = (w_1,\ w_0)$ $\sigma(z) = \frac{1}{1+e^{-z}},$ $z = w_1 x + w_0$

Minimize binary cross-entropy = maximize log-likelihood

Loss: ~~$\sum_{i=1}^{N} \epsilon_i^2$~~

$$L(\mathbf{w}) = -N^{-1}\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$$

$$\hat{y}_i = M(x_i \mid \mathbf{w}) = \Pr(y_i = 1 \mid x_i; \mathbf{w})$$

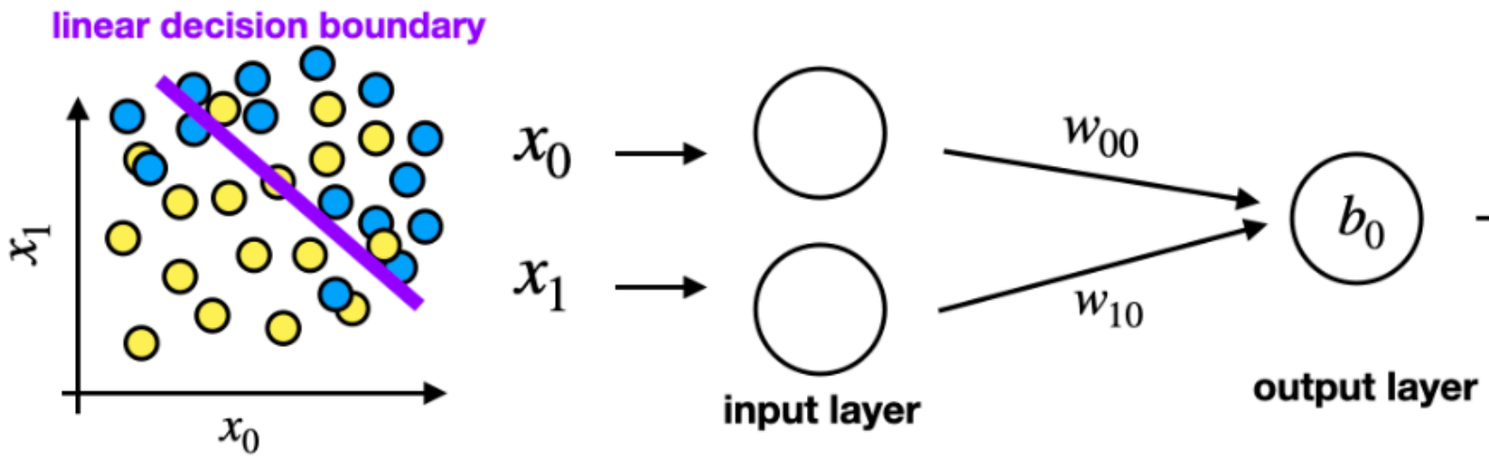
squared error (regression)

Numerical Optimization: $\nabla L(\mathbf{w}) = 0 = \left(\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_0}\right)$ $\mathbf{w}_{n+1} = \mathbf{w}_n - \eta \nabla L(\mathbf{w}_n)$

logistic regression:

- **Problem Type:** Binary-classification
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^1}$
- **Hidden layers**: None
- **Hidden-layer activation**: N/A
- **Output-layer activation**: Sigmoidal (logistic)
- **Number of fitting parameters**: 3

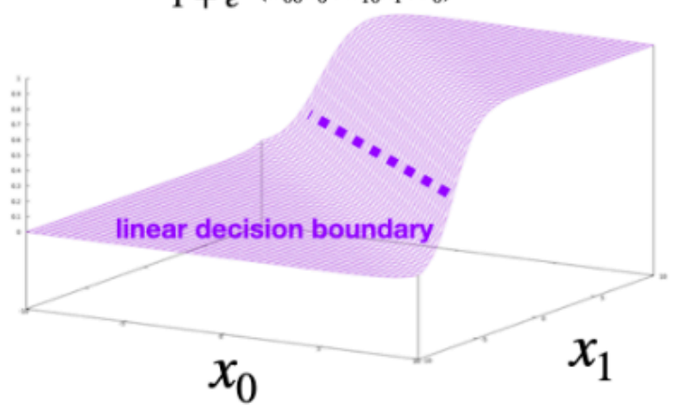
Computational graph



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

$$\rightarrow S(w_{00}x_0 + w_{10}x_1 + b_0)$$

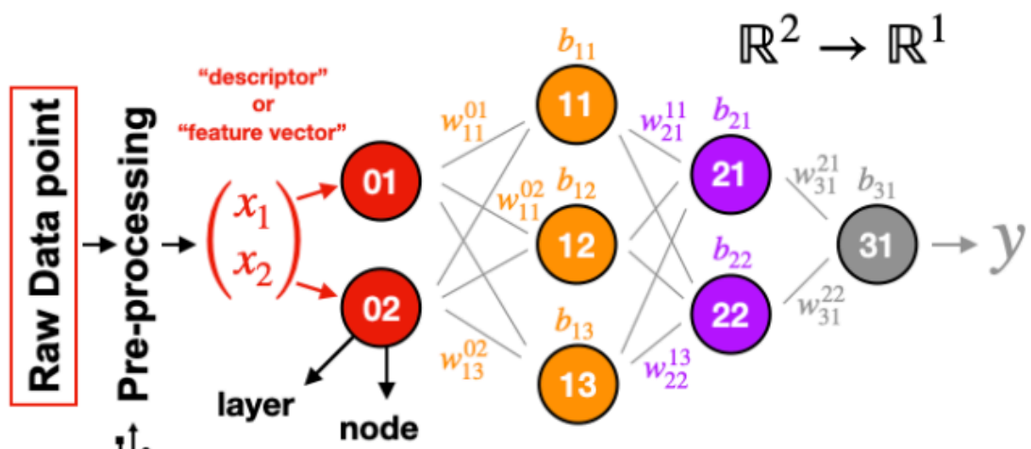
probablitly: $p = \frac{1}{1+e^{-(w_{00}x_0+w_{10}x_1+b_0)}} = S(w_{00}x_0 + w_{10}x_1 + b_0)$



The multi-layer perceptron model

ANN model: introductory example

- **Problem Type:** Scalar regression (un-constrained output)
- **Mapping:** $\mathbb{R}^2 \rightarrow \mathbb{R}^1$ (input layer) (output layer)
- **Hidden layers:** $N_{hidden} = 2 \rightarrow [3,2]$
- **Hidden-layer activation**: Sigmoid
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 20



NN architecture: 2-3-2-1

Number of fitting parameters:

$$N_{fit} = (3 \times 2 + 3) + (2 \times 3 + 2) + (2 \times 1 + 1) = 20$$

$$\begin{pmatrix} w_{11}^{01} \\ w_{12}^{01} \\ w_{13}^{01} \\ w_{11}^{02} \\ w_{12}^{02} \\ w_{13}^{02} \\ b_{11} \\ b_{12} \\ b_{13} \\ w_{21}^{11} \\ w_{22}^{11} \\ w_{21}^{12} \\ w_{22}^{12} \\ w_{21}^{13} \\ w_{22}^{13} \\ b_{21} \\ b_{22} \\ w_{31}^{21} \\ w_{31}^{22} \\ b_{33} \end{pmatrix}$$

Input Layer $\rightarrow A(x) \rightarrow A(x) \rightarrow$ **Output layer**

Activation function:

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

apply activation function (non-linearity) Apply component wise

Hidden layer-1 output

$$\begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} = A\left(\begin{pmatrix} w_{11}^{01} & w_{11}^{02} \\ w_{12}^{01} & w_{12}^{02} \\ w_{13}^{01} & w_{13}^{02} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \end{pmatrix} \right)$$

Hidden layer-2 output

$$\begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} = A\left(\begin{pmatrix} w_{21}^{11} & w_{21}^{12} & w_{21}^{13} \\ w_{22}^{11} & w_{22}^{12} & w_{22}^{13} \end{pmatrix} \begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} + \begin{pmatrix} b_{21} \\ b_{22} \end{pmatrix} \right)$$

Neural network output

$$y = (O_{31}) = \begin{pmatrix} w_{31}^{21} & w_{31}^{22} \end{pmatrix} \begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} + (b_{31})$$

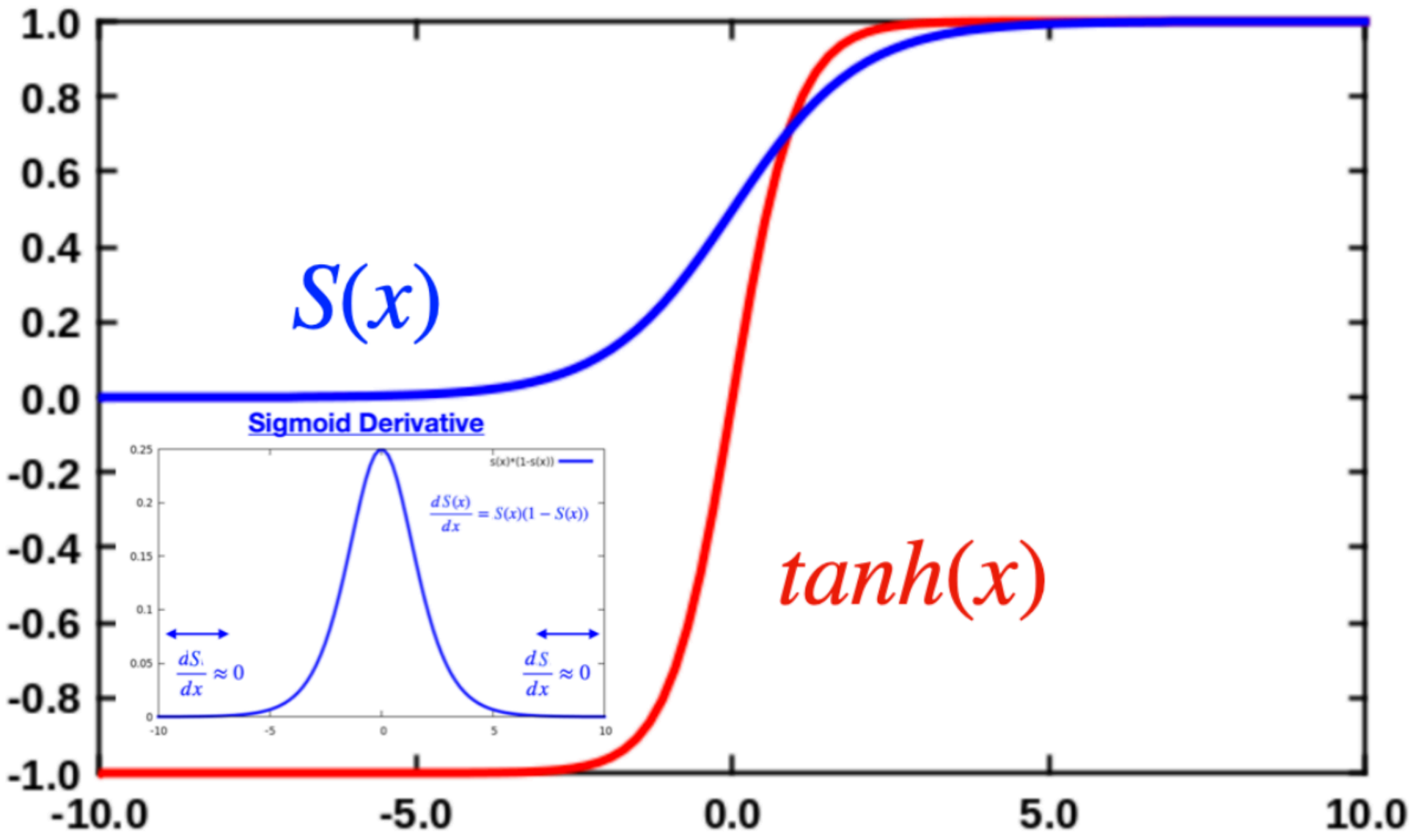
Don't apply activation function on output layer

Matrix form:

$$y = \mathbf{w}_{23} A\left(\mathbf{w}_{12} A\left(\mathbf{w}_{01}\mathbf{x} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right) + \mathbf{b}_3$$

Activation: Sigmoid (i.e. logistic) and Tanh

- Activation functions, aka. non-linearities, make the ANN a non-linear interpolator
- Historically, these were very popular choices for activation functions.



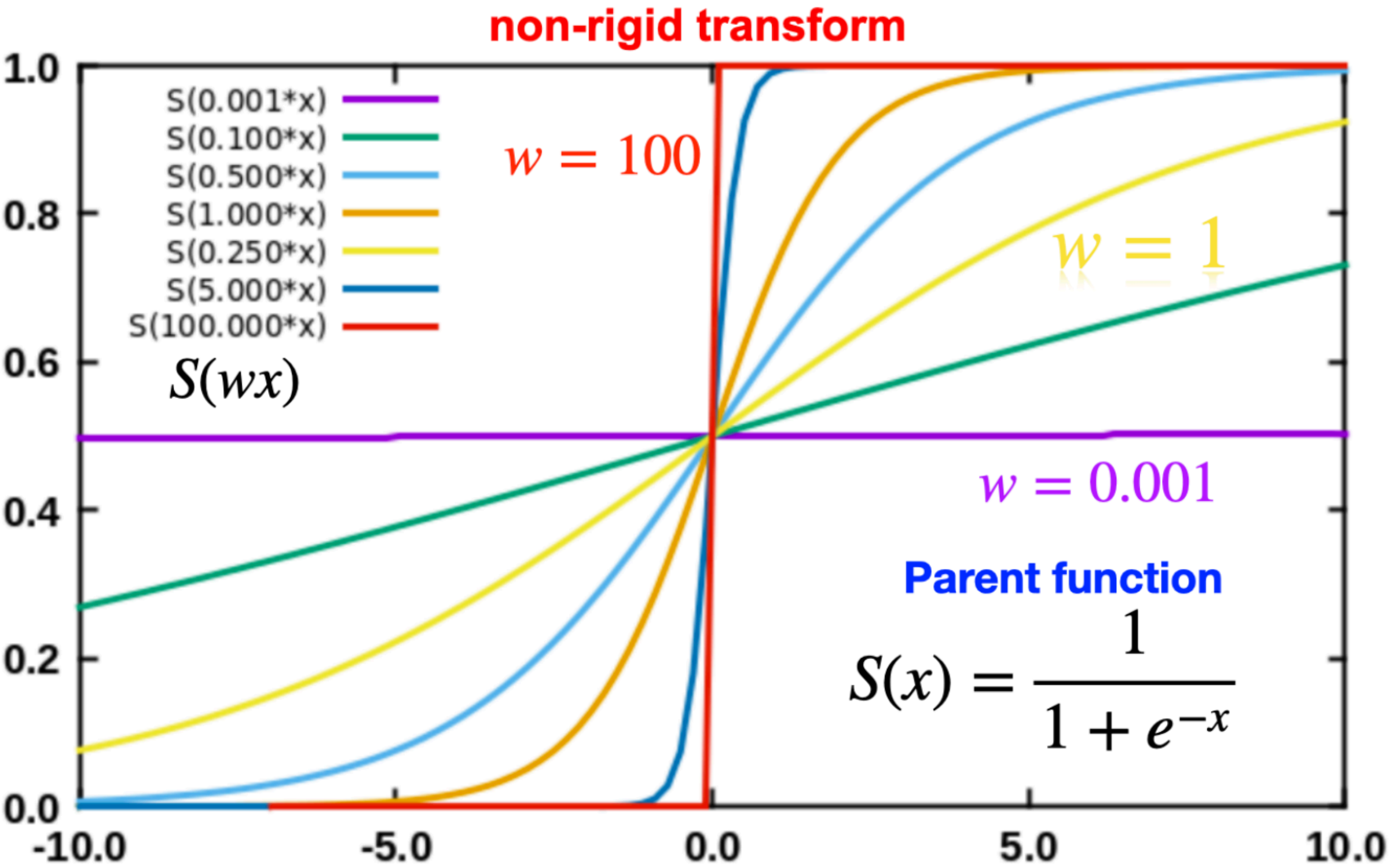
Sigmoid function:

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

Hyperbolic Tangent function

$$\tanh(x) = 2\left(S(2x) - 0.5\right)$$

Logistic function: Width parameter



Small weights —> less overfitting

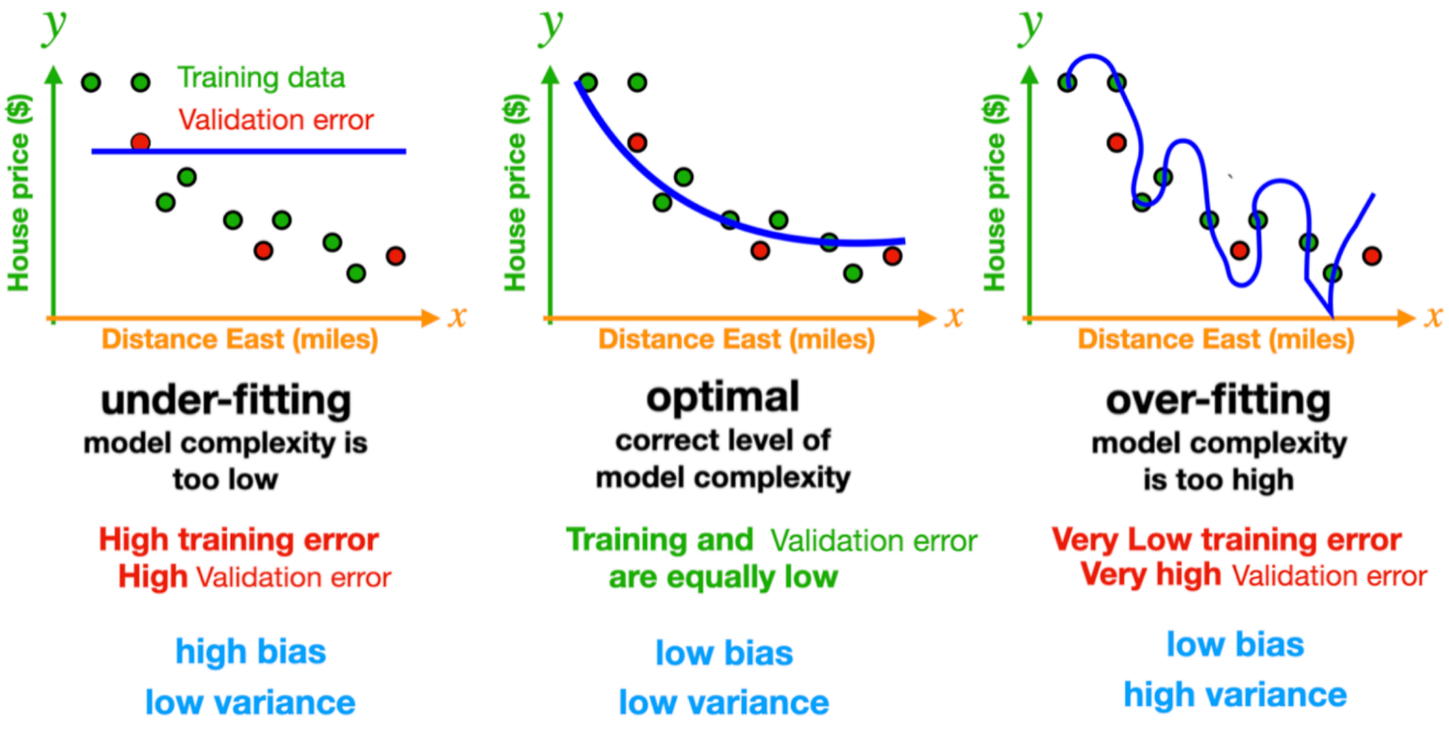
Regularization: constrain model flexibility

Deep nets have enough capacity to memorize the training set.

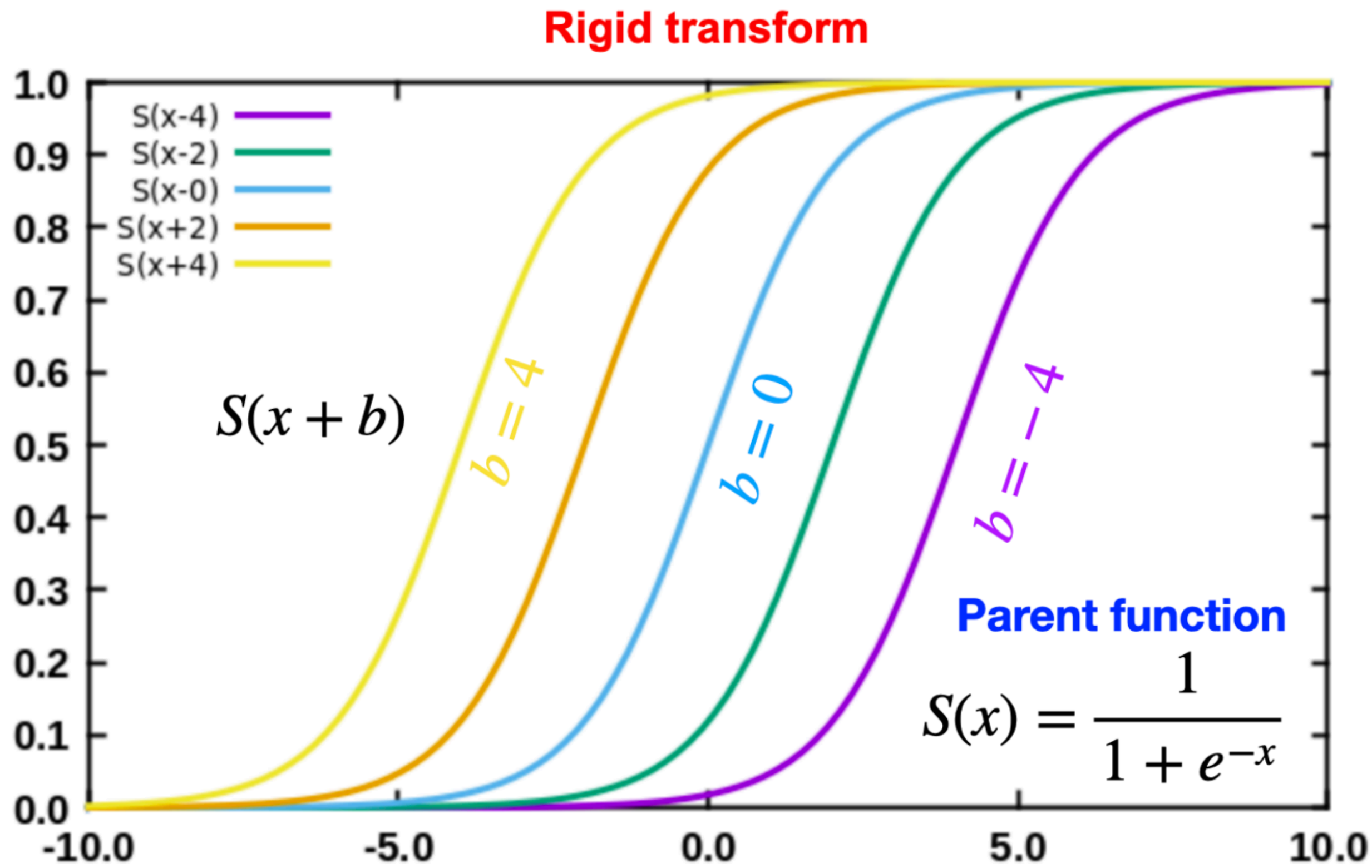
Regularization methods = tricks to PREVENT OVER FITTING

Method	Mechanism	Keras
L2 / weight decay	Penalize $\|w\|_2^2 \rightarrow$ shrink weights toward 0	`kernel_regularizer=l2(1e-4)`
L1	Penalize $\|w\|_1 \rightarrow$ drives weights *to* 0 (sparse)	`kernel_regularizer=l1(1e-5)`
Dropout	Randomly zero units each step → ensemble of subnets	`layers.Dropout(0.5)`
Early stopping	Halt when val loss turns up	`callbacks.EarlyStopping()`
Data augmentation	Enlarge the effective training set	`layers.RandomFlip()`, etc.
Batch norm	Noisy batch statistics act as a regularizer	`layers.BatchNormalization()`

All of these say the same thing in different dialects: **prefer the simpler function** when the data can't tell two models apart.

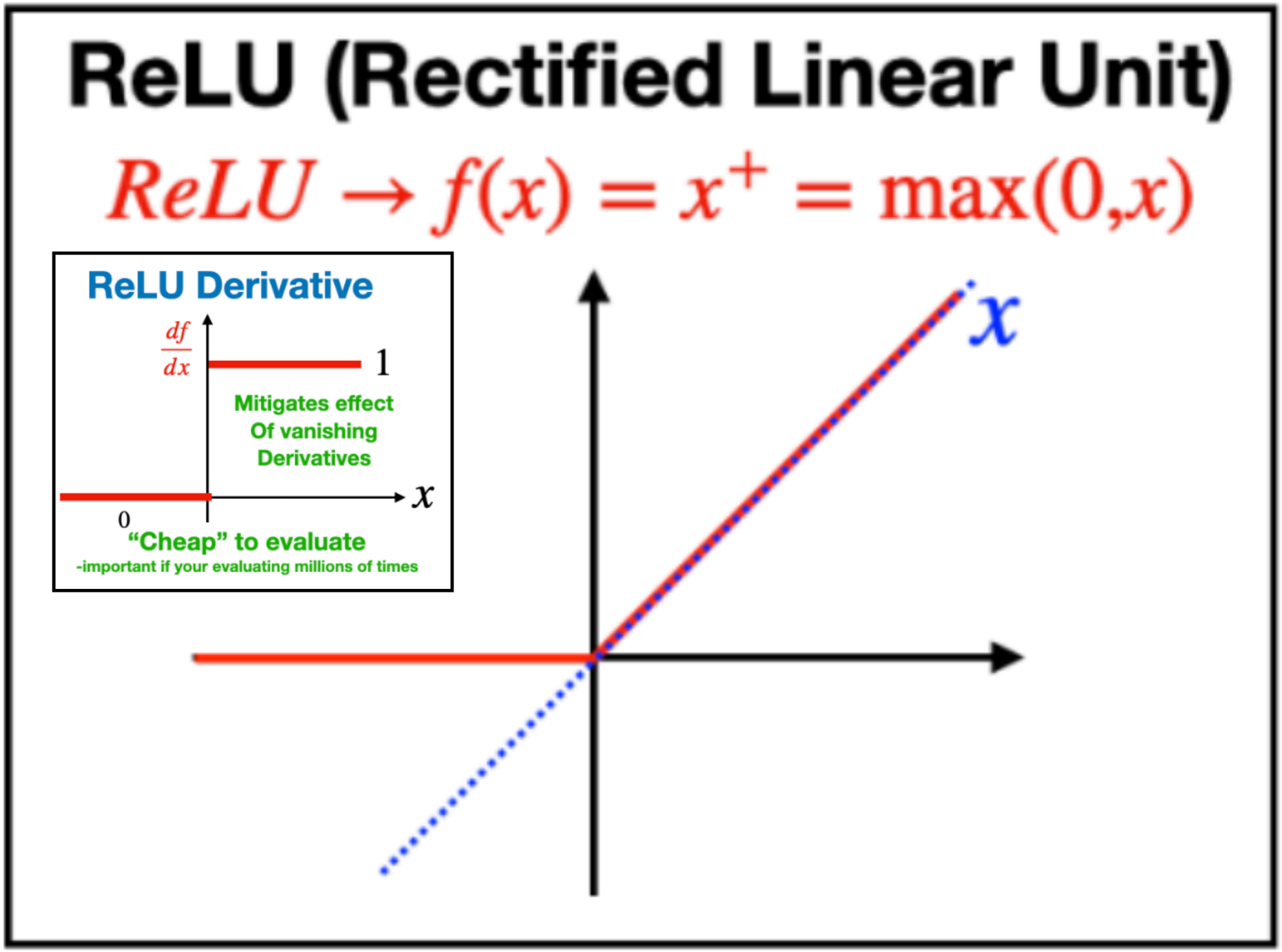


Logistic function: Shift parameter



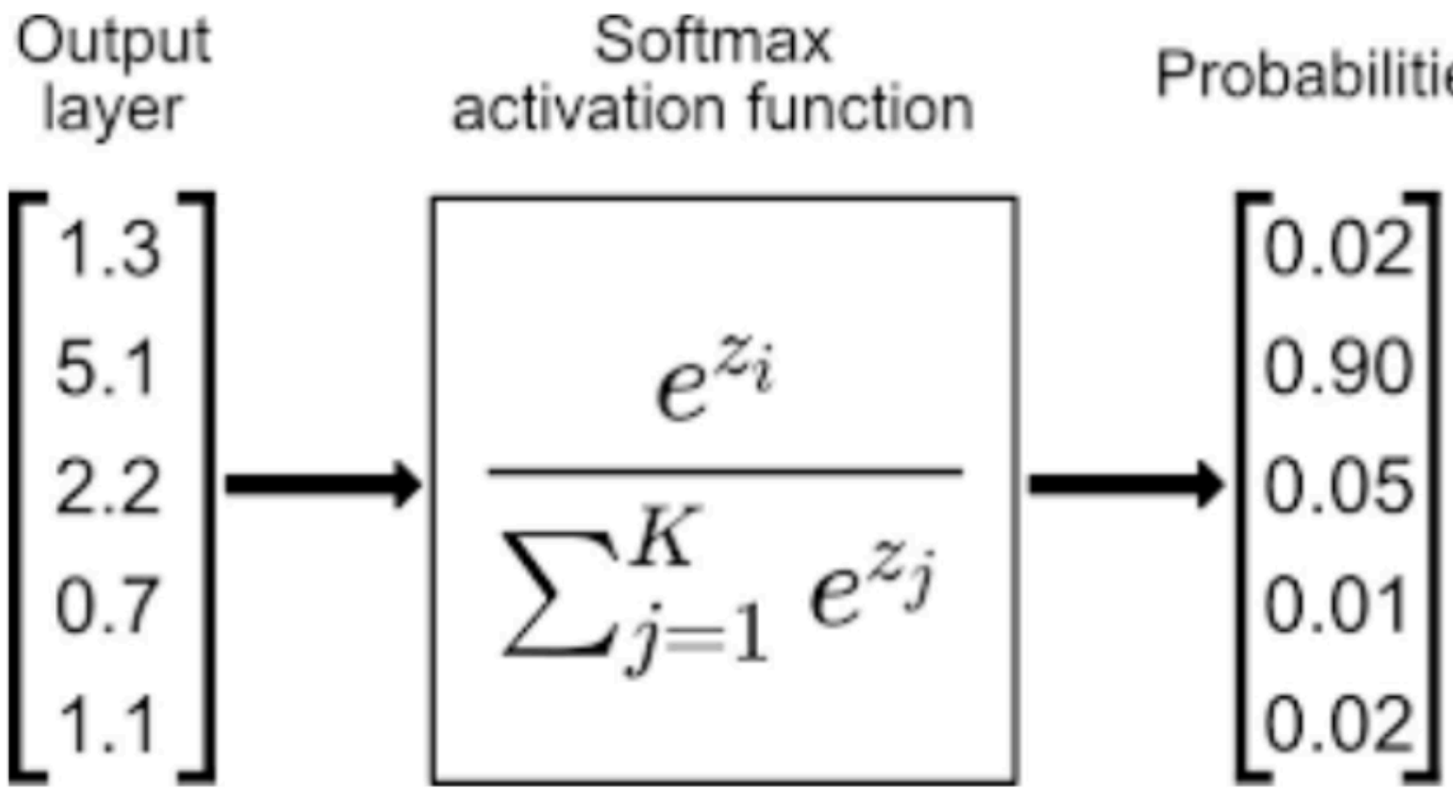
Activation: RELU

- A rectified linear unit (ReLU) is an activation function that introduces non-linearity
- Arguably, the popular common activation function, since it mitigates the vanishing gradient effect.



Activation: Softmax

- The `softmax function` converts a vector of numbers into a probability distribution by exponentiating each element and dividing by the sum of exponentiated values.
    - The division normalizes the vector so that the sum is equal to one, since $\sum_i p_i = 1$.
- Softmax returns an array of K probability scores (summing to 1). Each score is the probability that the sample belongs to one of our K classes.
- The standard (unit) soft-max function is defined by the formula



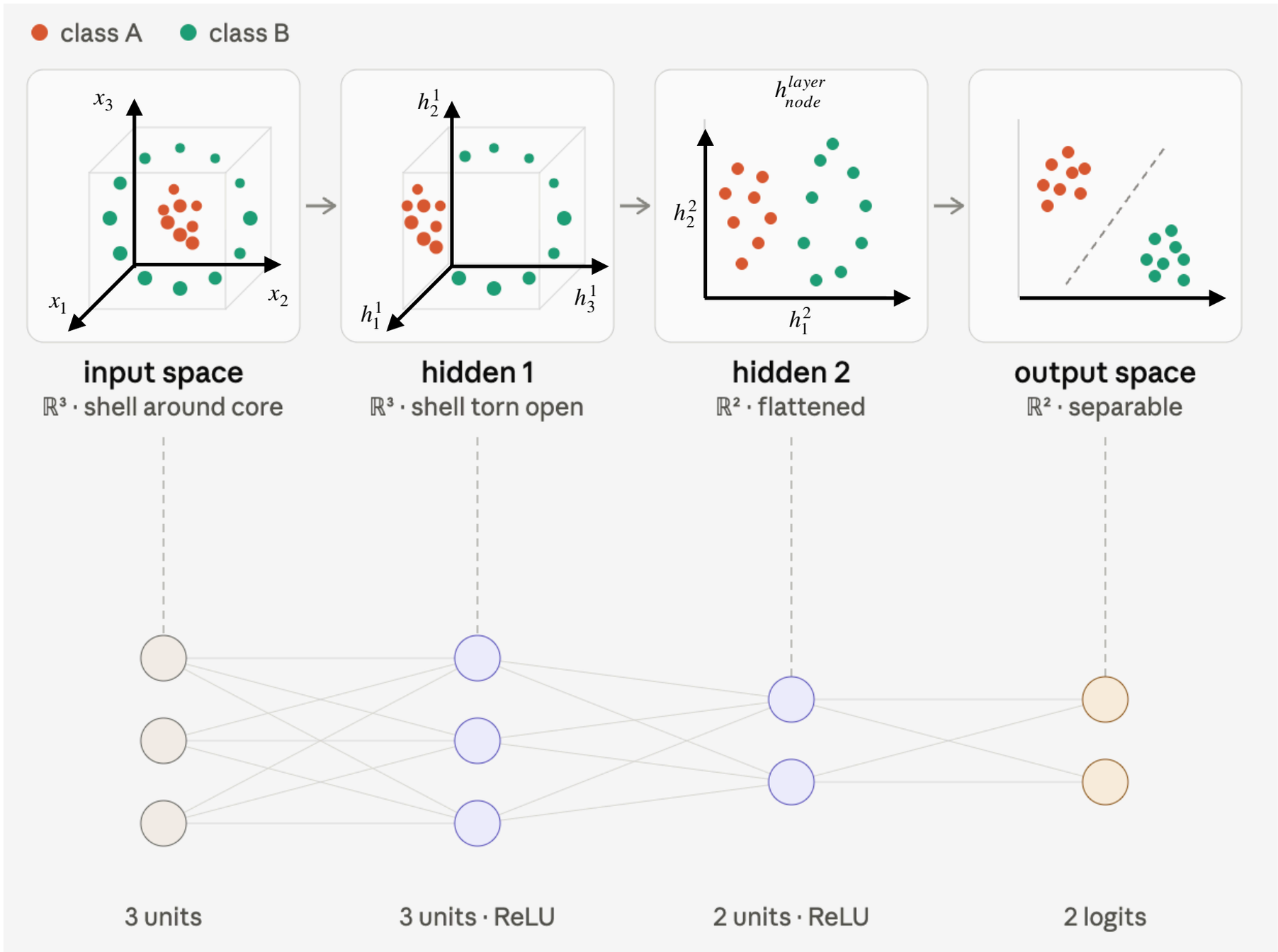
- This is very useful for `Multi-class classification`

Representation learning: Core idea in deep learning

Deep learning models repeatedly transform data into better and better representations spaces.

You can envision this as the data flowing through the network resulting in various intermediate representation spaces.

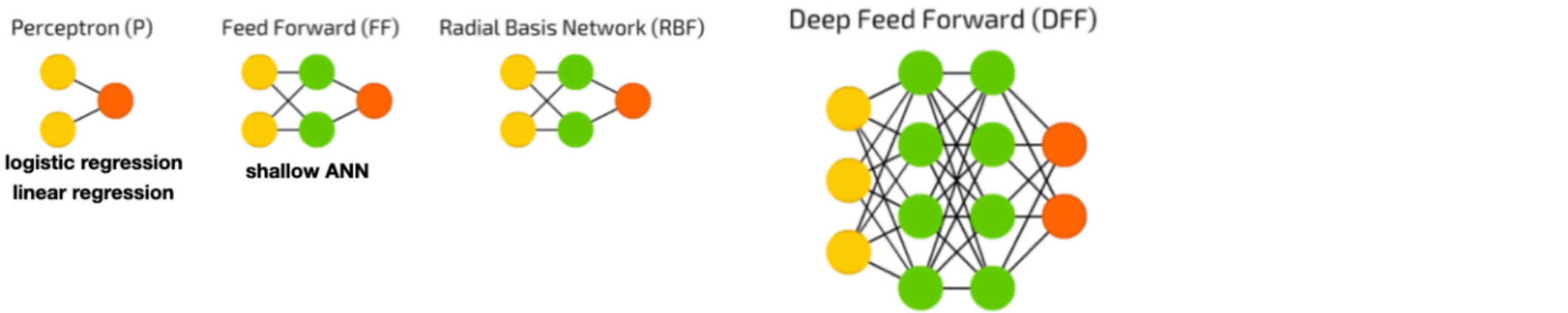
learned data transformation



What is better: In this context better means more suitable for a particular machine learning task.

Common Neural network architecture

- The type of data you have dictates which architecture you should use
- General classifier/regressor (MLP), image data (CNN), text data (LSTM or Transformer).



SOURCE: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

Common Loss Functions

Regression

- **MSE (L2)** — default; smooth gradients, but heavily penalizes outliers.
  $\mathcal{L} = \frac{1}{N}\sum_i (\hat{y}_i - y_i)^2$
- **MAE (L1)** — robust to outliers; constant-magnitude gradient.
  $\mathcal{L} = \frac{1}{N}\sum_i |\hat{y}_i - y_i|$
- **Huber** — L2 near zero, L1 in the tails; best of both when outliers exist but you want smooth gradients.

Binary Classification

- **Binary cross-entropy** — $\hat{y} \in (0, 1)$ from a **sigmoid**; the standard for one-vs-rest and multi-label.
  $\mathcal{L} = -\frac{1}{N}\sum_i \left[y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i) \right]$

Multi-Class Classification

- **Categorical cross-entropy** — $\hat{\mathbf{y}}$ from a **softmax** over C classes; $\mathbf{y}$ is one-hot.
  $\mathcal{L} = -\frac{1}{N}\sum_i \sum_{c=1}^{C} y_{ic} \log \hat{y}_{ic}$
- **Sparse categorical CE** — identical math, integer labels instead of one-hot (memory saver).

Training jargon: what actually counts as a "step"

- **Iteration** (*step*): one forward pass, one backward pass, **one gradient update**.
- **Gradient update**: the optimizer touching the weights once: $w \leftarrow w - \eta\,\hat{g}$.
  Synonymous with iteration; the two are 1:1.
- **Epoch**: one full sweep over all N training examples.

$$\text{iterations per epoch} = \lceil N/B \rceil$$

The three regimes

Regime	batch size B.	Updates / epoch	Gradient $\hat{g}$
Full batch	N	1	stable ∇L (Smoother training)
Mini-batch	$1 < B < N$	$\lceil N/B \rceil$	Unbiased estimate, some noise
Stochastic (SGD)	1	N	Unbiased, very noisy

Worked example — MNIST

$N = 60{,}000$, `batch_size=128`, `epochs=10`

- $\lceil 60000/128 \rceil = \mathbf{469}$ iterations per epoch (last batch holds 96)
- $469 \times 10 = \mathbf{4{,}690}$ gradient updates total
- Full batch would give **10** updates. Same data, ~470× less learning.

Noise can be good isn't a defect to be minimized: it's what lets mini-batch escape sharp minima and saddle points. Full batch is *more accurate per step* and *worse per epoch*.

Keras

1. **Load** — `keras.datasets.mnist.load_data()`
2. **Explore** — shapes, ranges, class balance, sample images
3. **Preprocess** — flatten to 784, rescale to [0, 1]
4. **Split** — train / validation / (locked) test
5. **Build & compile** — `Sequential` + `Dense` layers; loss, optimizer, metrics
6. **Train** — `fit()`, watching validation each epoch
7. **Evaluate** — training curves, test accuracy, confusion matrix, error inspection

Deep learning software tools

MOST POPULAR TOOLKITS

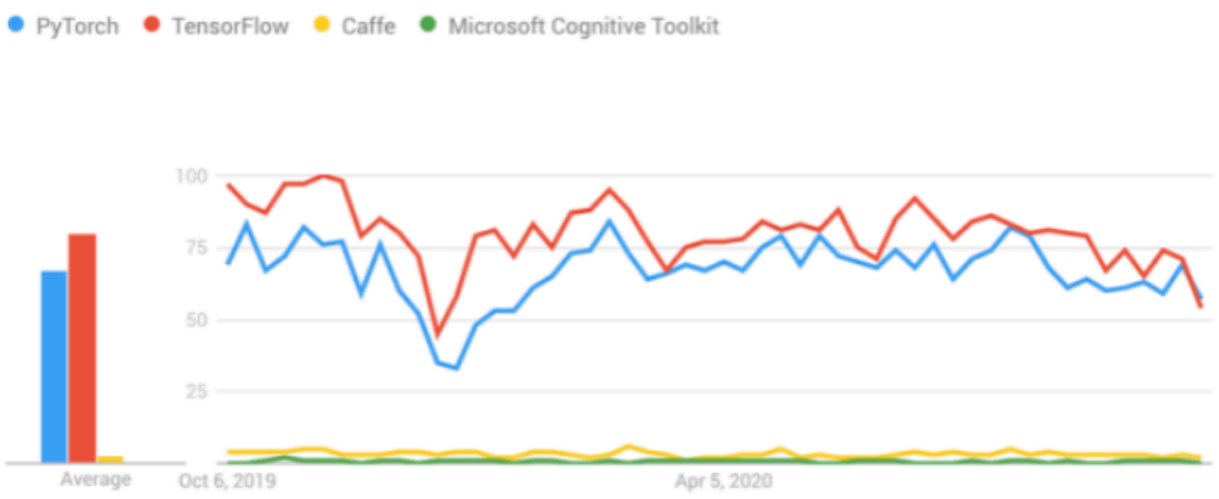
- **TENSOR-FLOW**
- **PYTORCH**
- **THEANO (MILA U-Montral)**
- **COGNITIVE TOOLKIT CNTK**
  - **(MICROSOFT)**

Keras

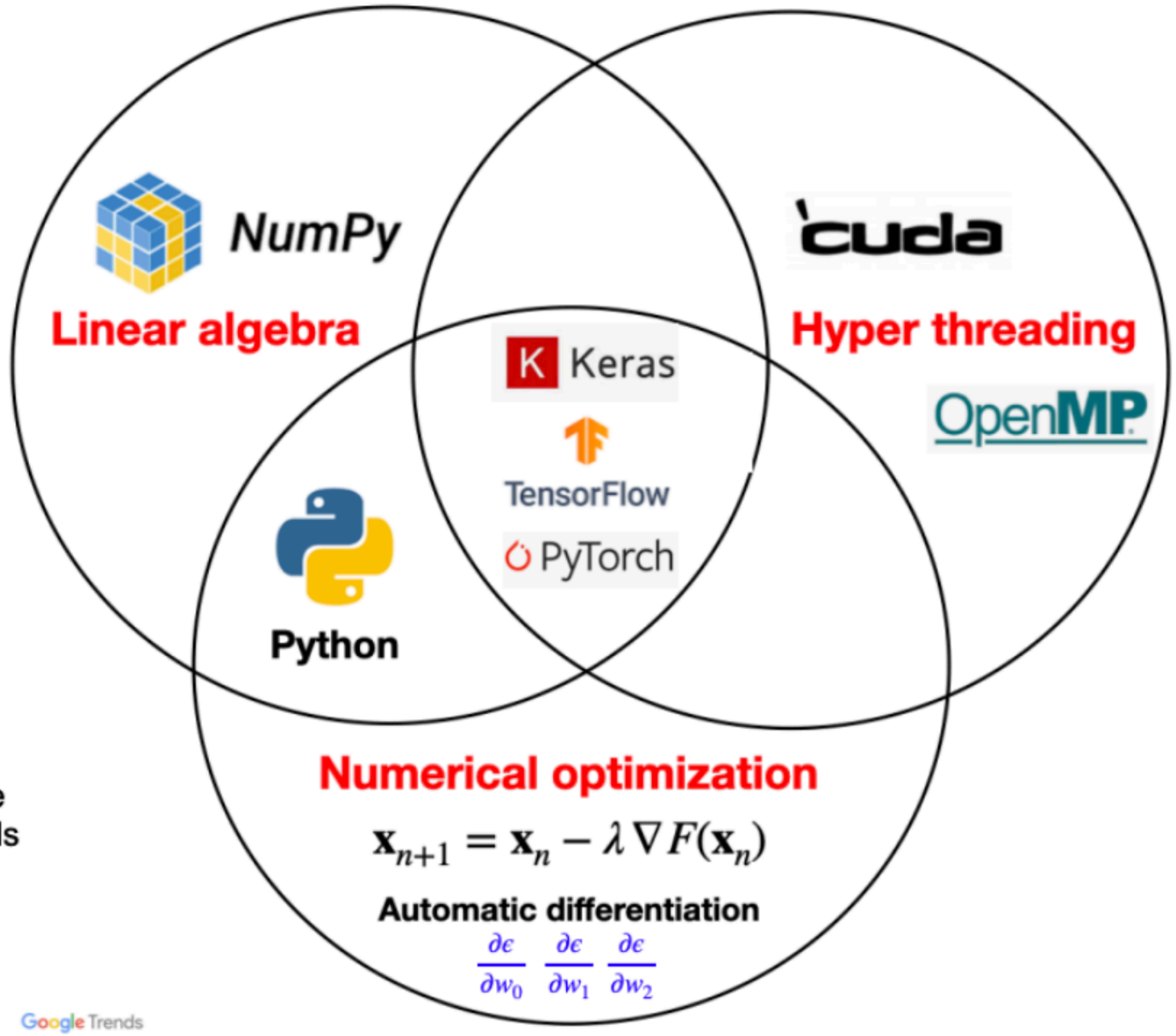
- Keras is an **open-source** software library
- **High-level API**: Application Programming Interface
- Provides a **python interface** for deep learning tools
- **Runs on top of TensorFlow,**
  - or Microsoft CNTK, or Theano
- **Easy to use**

Interest over time

Google Trends



United States. Past 12 months. Web Search. https://www.tooploox.com/blog/pytorch-vs-tensorflow-a-detailed-comparison



Keras + TensorFlow

PyTorch

- provide exposure
- Add extra line to vour resume

KERAS SUMMARY

Keras API reference

Keras

- Models API
- Layers API
- Callbacks API
- Data preprocessing
- Optimizers
- Metrics
- Losses
- Built-in small datasets
- Keras Applications
- Utilities
- Keras Tuner

The Model class

- Model class
- summary method
- get_layer method

The Sequential class

- Sequential class
- add method
- pop method

Model training APIs

- compile method
- fit method
- evaluate method
- predict method
- train_on_batch method
- test_on_batch method
- predict_on_batch method
- run_eagerly property

Keras layers API

- Creating custom layers
- Layers API overview

The base Layer class
Layer activations
Layer weight initializers
Layer weight regularizers
Layer weight constraints
Core layers
Convolution layers
Pooling layers
Recurrent layers
Preprocessing layers
Normalization layers
Regularization layers
Attention layers
Reshaping layers
Merging layers
Locally-connected layers
Activation layers

Optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Model saving & serialization APIs

- save method
- save_model function
- load_model function
- get_weights method
- set_weights method
- save_weights method
- load_weights method
- get_config method
- from_config method
- model_from_config function
- to_json method
- model_from_json function
- clone_model function

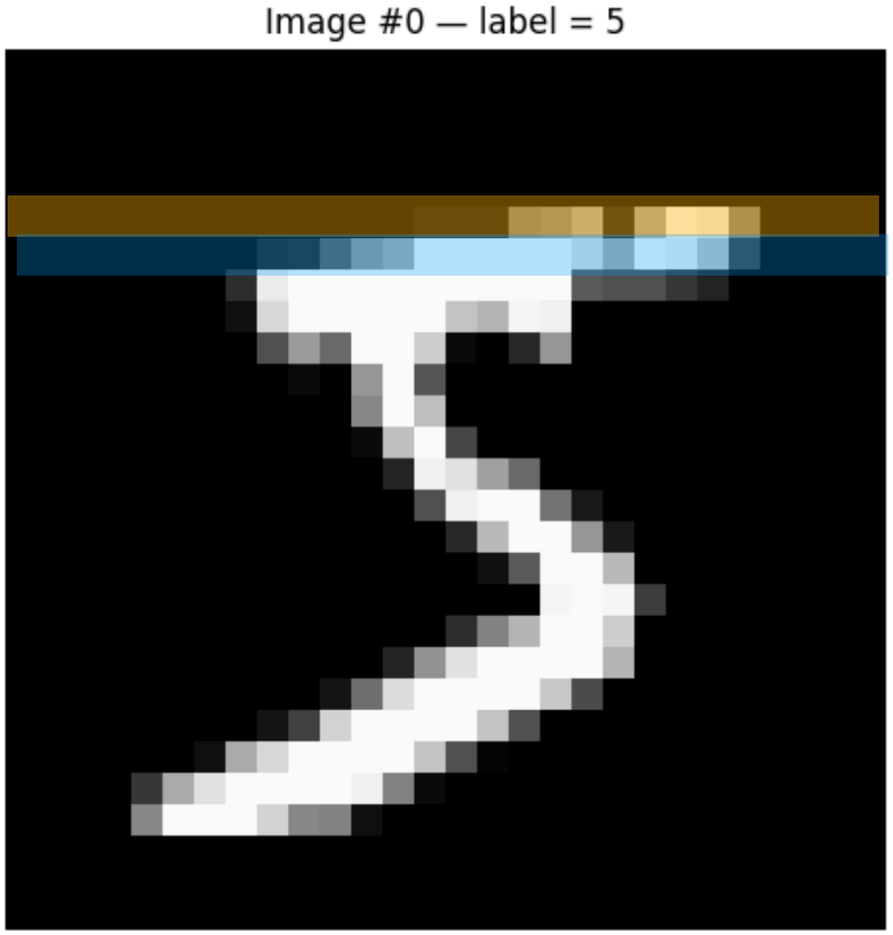
https://keras.io/api/models/

The “canonical” deep learning training workflow

- **Data** → collect, clean, **supervise**, split (train/val/test), augment
  - Garbage in, garbage out: this is where most projects live or die
- **Model Architecture** → choose a model that well suited to the data
  - MLP (simple data), CNN (grids/Images), RNN/Transformer (sequences), GNN (graphs)
- **Objective** (prediction quality scoring mechanism) → define a loss $\mathcal{L}(\theta)$ that encodes "good"
  - Regression: Mean square error MSE
  - Binary classification: Binary Cross-entropy
  - Multi-class classification: Categorical Cross-entropy
- **Optimization** (learning) → back-prop + SGD/Adam, LR schedule, regularization
  - We are just rolling downhill on a very high-dimensional surface
- **Evaluation** → validation metrics, error analysis, ablations
  - Tune on val, touch test *once*
- **Deployment** → export, serve, monitor drift, iterate

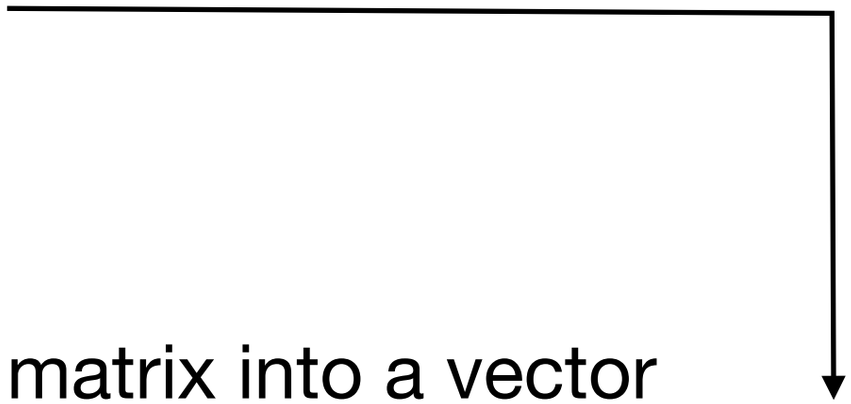
Aside: "Canonical" means conforming to a standard, authoritative, or generally accepted form the version of something that serves as the reference point against which variants are judged (e.g., the canonical form of a matrix, or the canonical version of a text).

Data: MNIST



28x28

Flatten



Flatten: unwraps (reshapes) matrix into a vector



$x \rightarrow (1,784)$

Keras Models:

Keras →

`Model` class

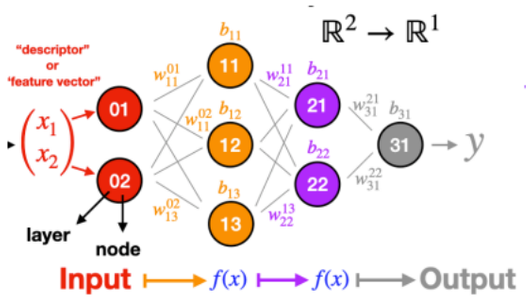
```
tf.keras.Model()
```

`Model` groups layers into an object with training and inference features.

Arguments

- **inputs**: The input(s) of the model: a `keras.Input` object or list of `keras.Input` objects.
- **outputs**: The output(s) of the model. See Functional API example below.
- **name**: String, the name of the model.

Neural network



- *Layers*, which are combined into a *network* (or *model*)
- The *input data* and corresponding *targets*
- The *loss function*, which defines the feedback signal used for learning
- The *optimizer*, which determines how learning proceeds

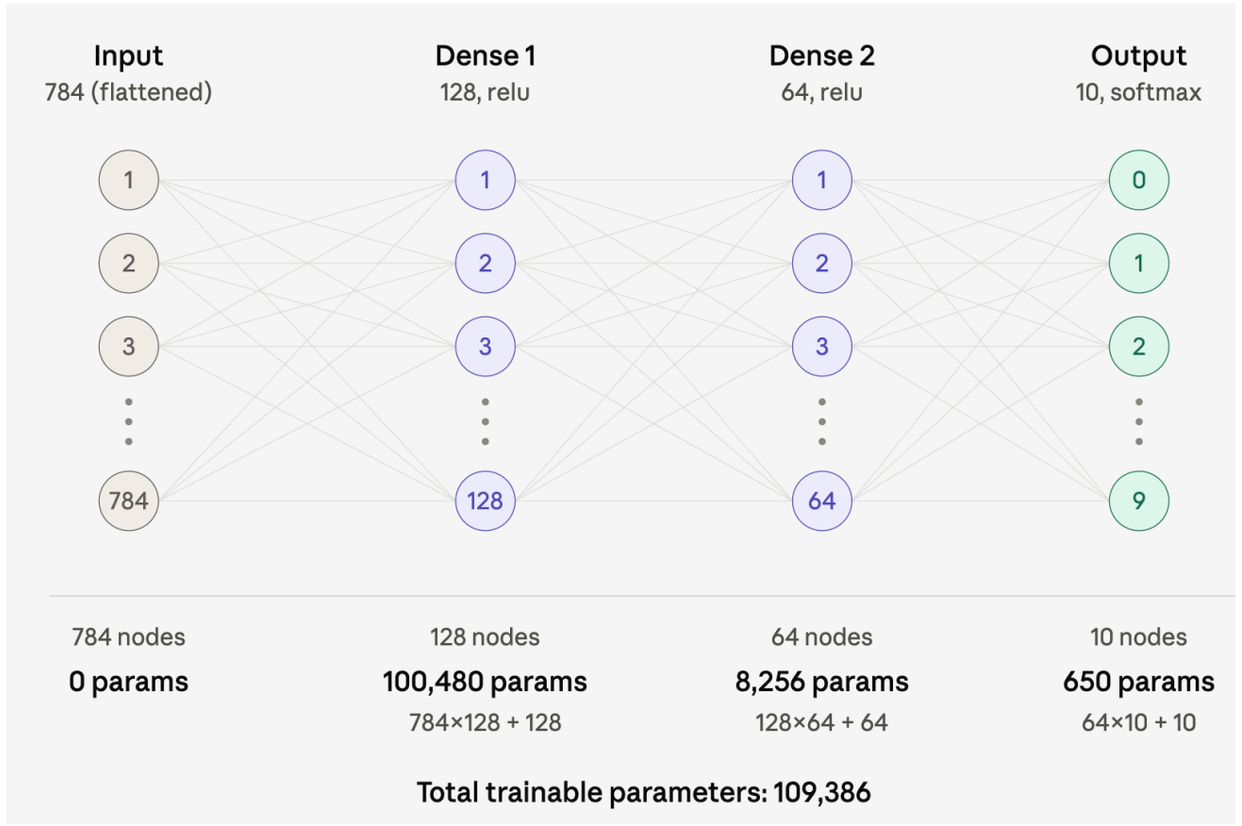
```
model = keras.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(128, activation="relu"),
    layers.Dense(64,  activation="relu"),
    layers.Dense(10,  activation="softmax"),
])

model.summary()
```

Keras Models:

```
model = keras.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(128, activation="relu"),
    layers.Dense(64,  activation="relu"),
    layers.Dense(10,  activation="softmax"),
])

model.summary()
```




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

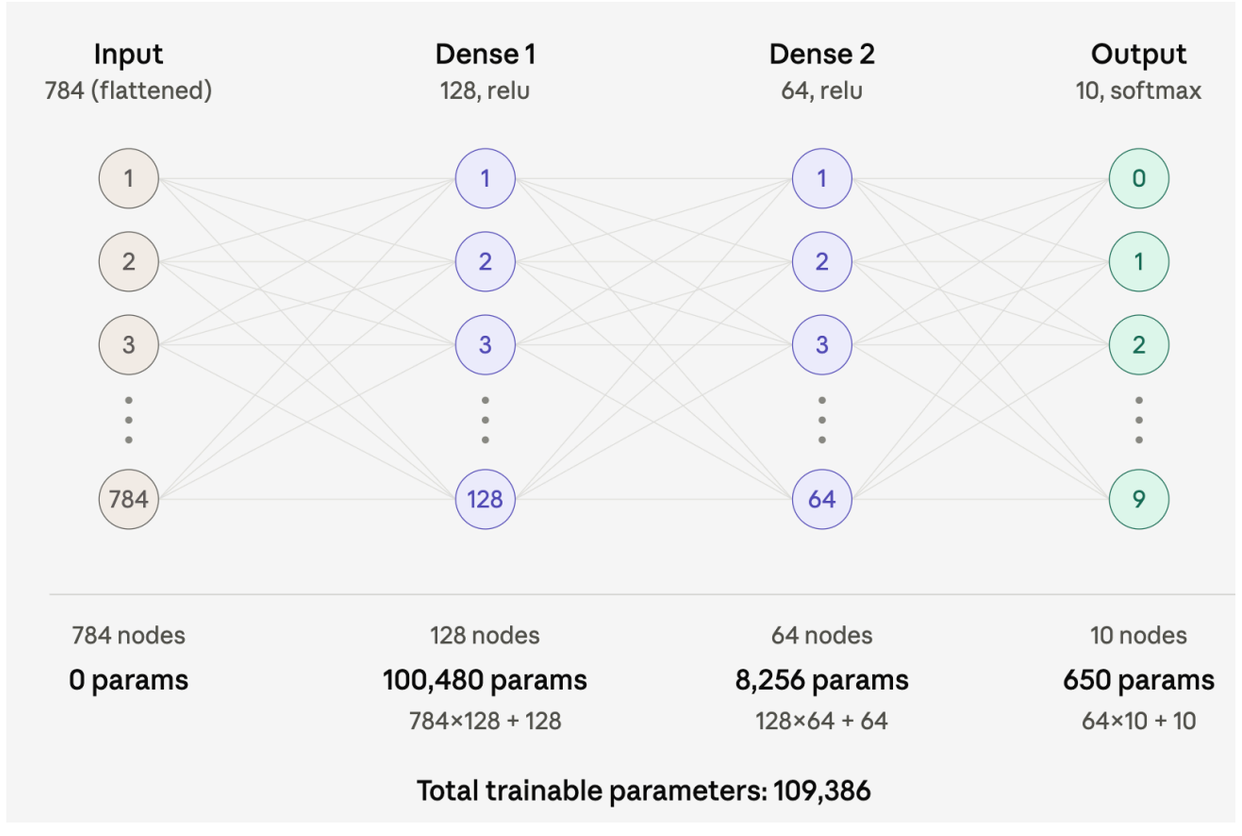
=================================================================
Total params: 109386 (427.29 KB)
Trainable params: 109386 (427.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

Keras Models:

Build model

```
model = keras.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(128, activation="relu"),
    layers.Dense(64,  activation="relu"),
    layers.Dense(10,  activation="softmax"),
])

model.summary()
```




```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109386 (427.29 KB)
Trainable params: 109386 (427.29 KB)
Non-trainable params: 0 (0.00 Byte)
```

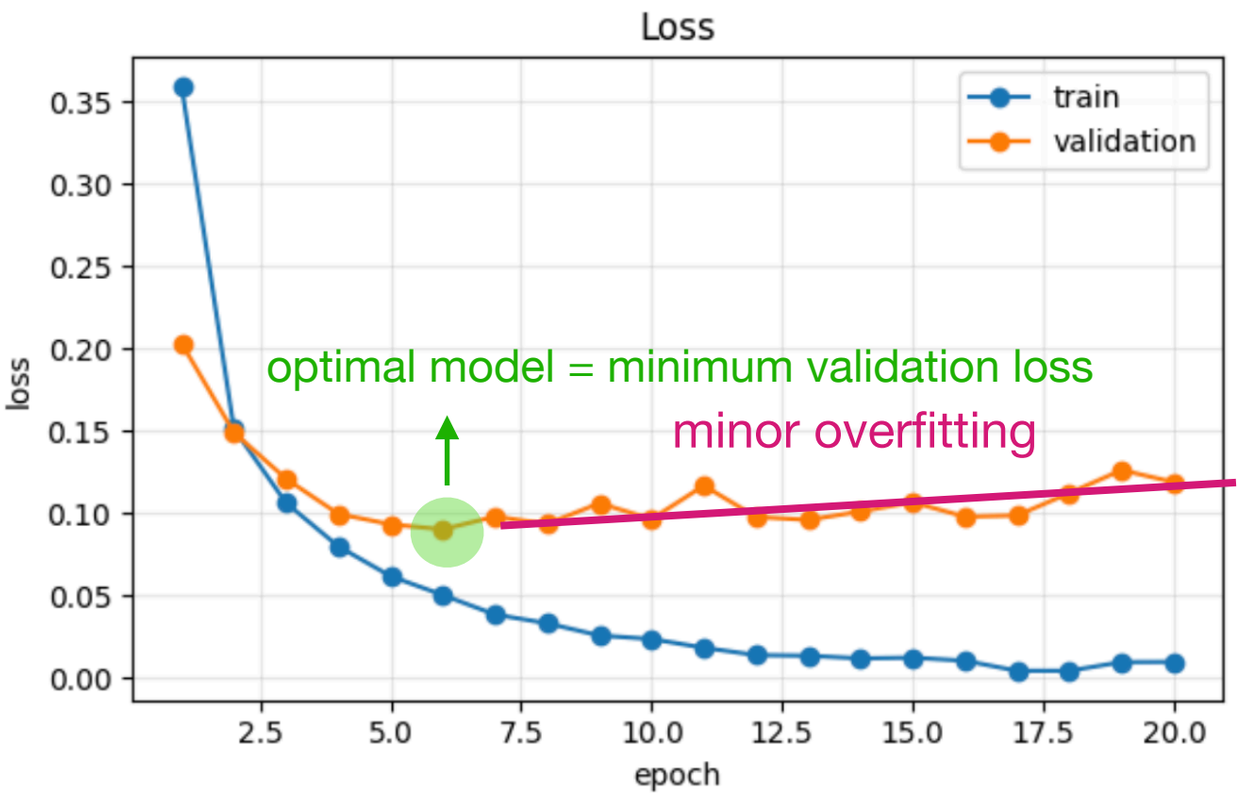
Compile model

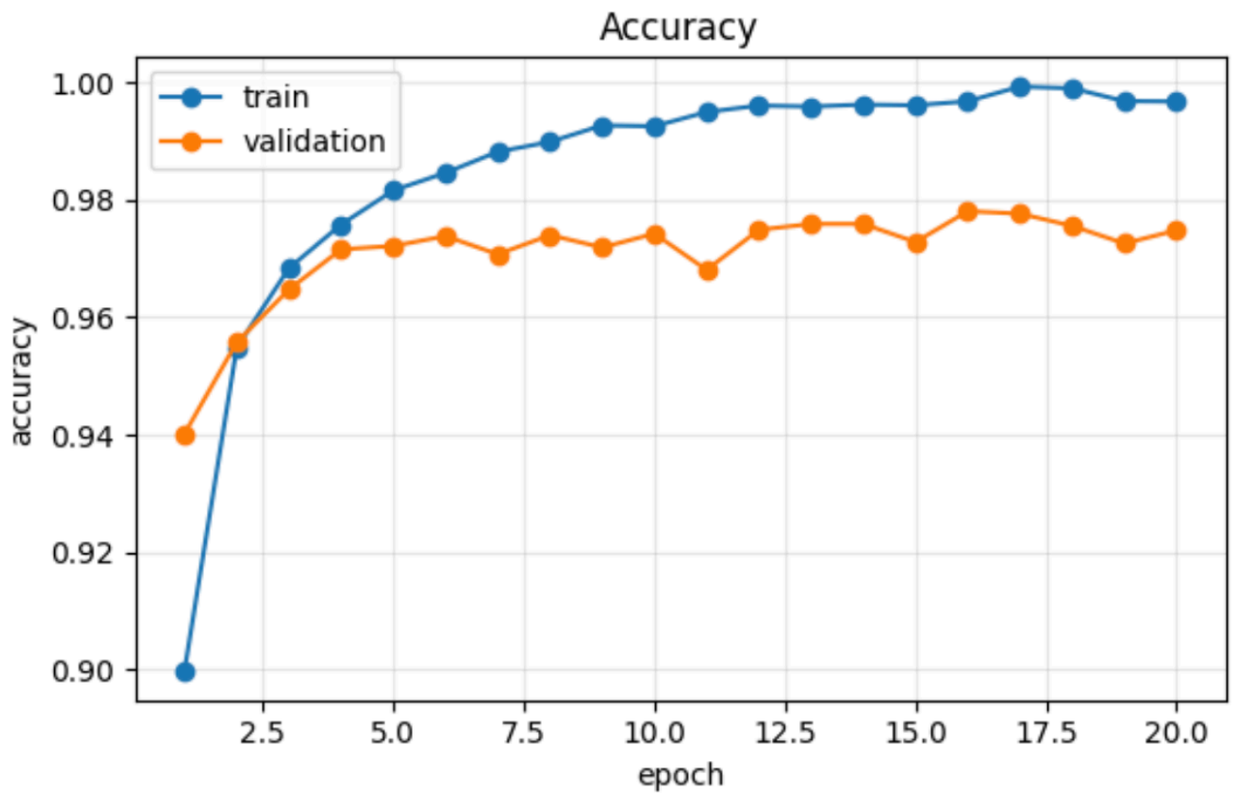
```
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)
```

Remember: Parameters are learned directly by the optimizer during training, **Hyper-parameters** (e.g. learning rate) are choices made by humans that **CONTROL** training

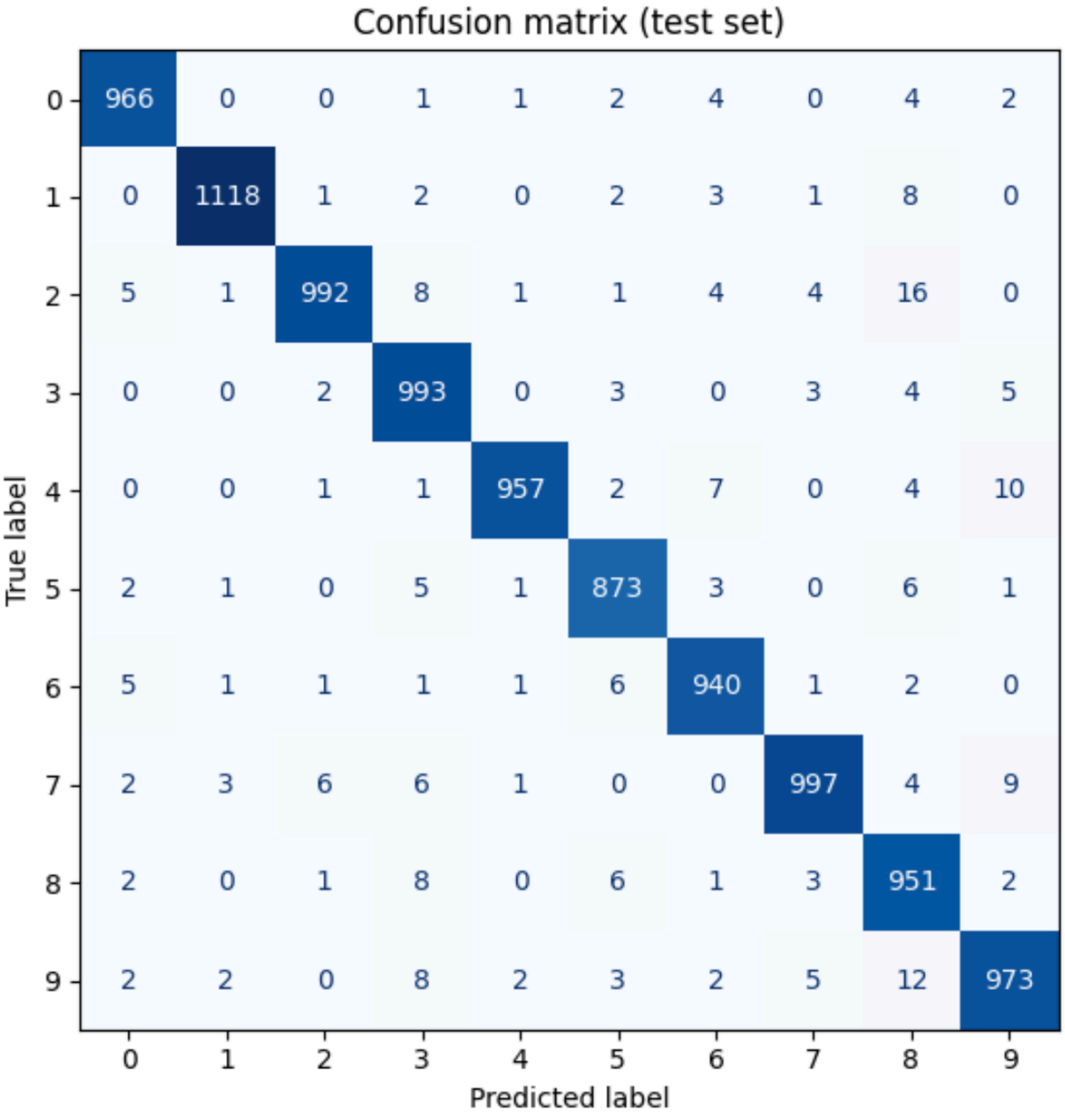
Train model

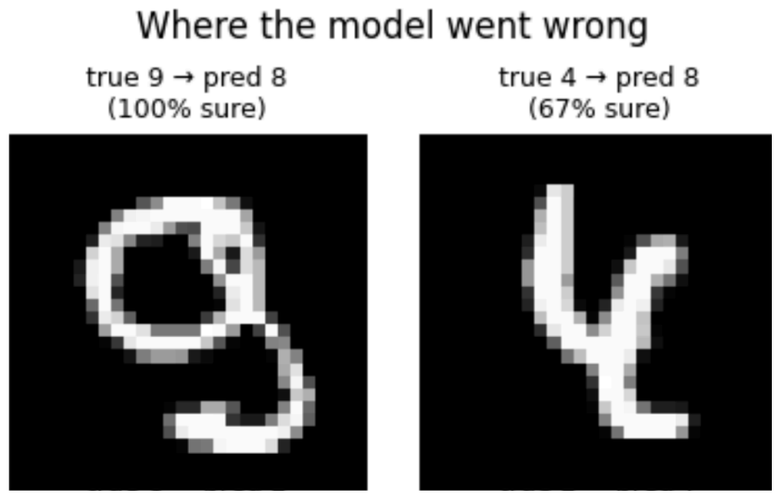
```
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=20,
    batch_size=128,
    verbose=2,
)
```





Results


Confusion matrix (test set)
True label
Predicted label
0 966 0 0 1 1 2 4 0 4 2
1 0 1118 1 2 0 2 3 1 8 0
2 5 1 992 8 1 1 4 4 16 0
3 0 0 2 993 0 3 0 3 4 5
4 0 0 1 1 957 2 7 0 4 10
5 2 1 0 5 1 873 3 0 6 1
6 5 1 1 1 1 6 940 1 2 0
7 2 3 6 6 1 0 0 997 4 9
8 2 0 1 8 0 6 1 3 951 2
9 2 2 0 8 2 3 2 5 12 973
0 1 2 3 4 5 6 7 8 9


Where the model went wrong
true 9 → pred 8
(100% sure)
true 4 → pred 8
(67% sure)

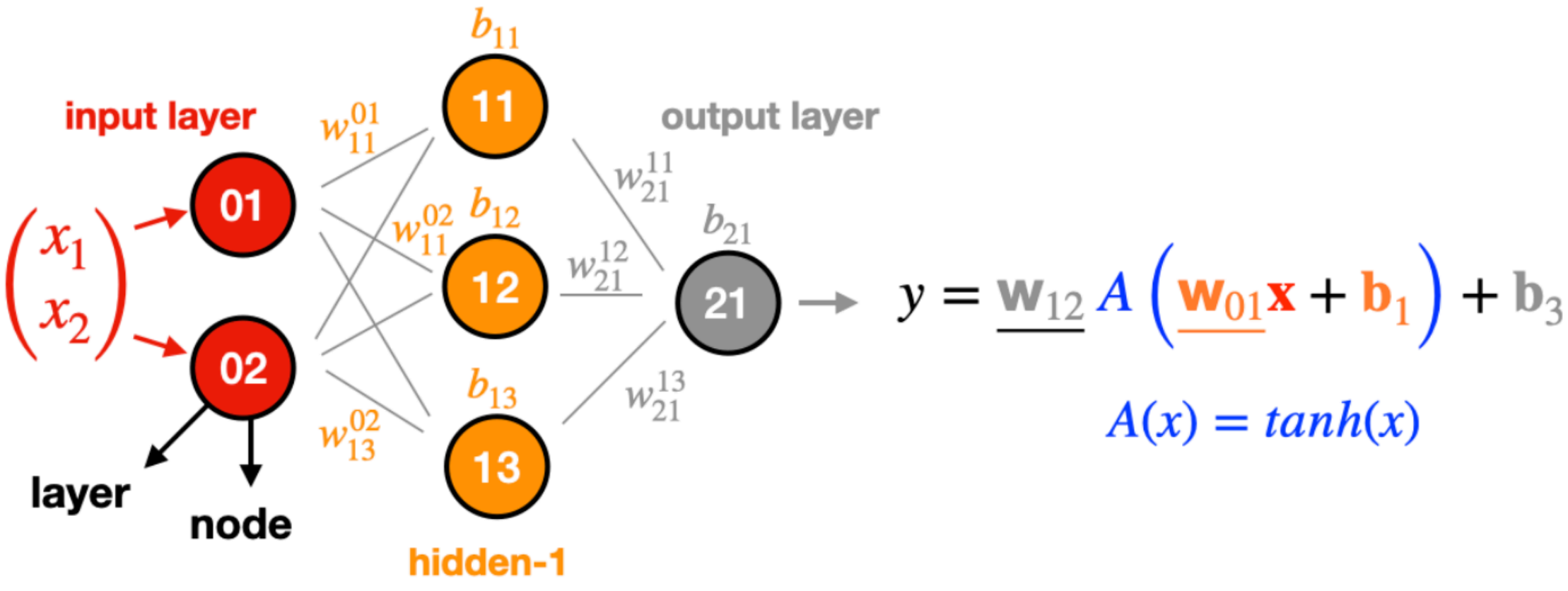
Summary of ANN type models

Important note: We use a 2D feature space, however, everything easily generalizes to higher dimensions and deeper ANNs

Shallow MLP: Scalar regression

- **Problem Type:** Scalar regression
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^1}$
- **Hidden layers**: $N_{hidden} = 1 \rightarrow [3]$
- **Hidden-layer activation**: Tanh
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 13

Computational graph

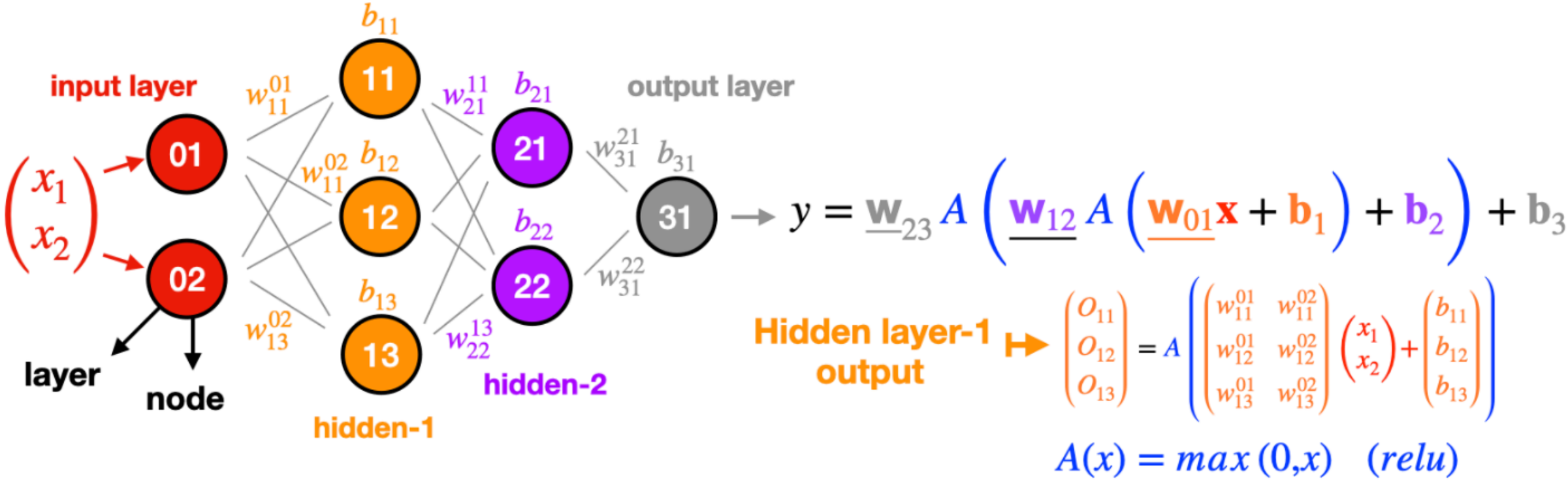


Note: We didn't have to use `tanh` on the *hidden layers*, other activations would also work

Deep MLP: Scalar regression

- **Problem Type:** Scalar regression
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^1}$
- **Hidden layers**: $N_{hidden} = 2 \rightarrow [3,2]$
- **Hidden-layer activation**: ReLu
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 20

Computational graph

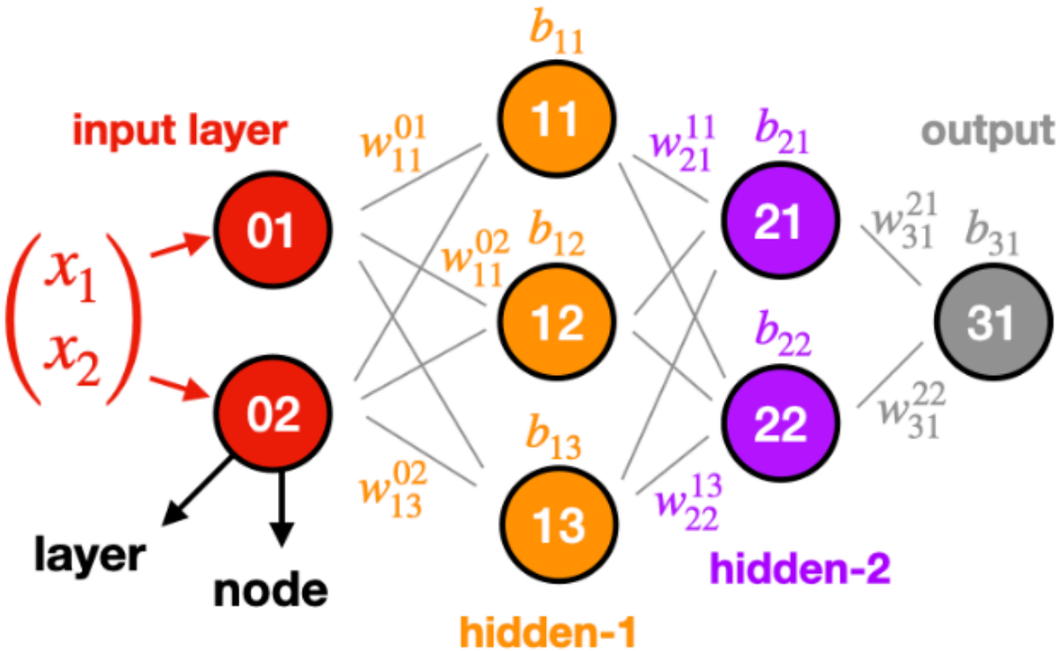


Note: (1) We didn't have to use `relu` on the *hidden layers*, other activations would also work (2) Instead of MLP, this is also referred to as a deep feed forward fully connected ANN

Deep MLP: Binary classification

- **Problem Type:** Binary classification
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^1}$
- **Hidden layers:** $N_{hidden} = 2 \rightarrow$ [3,2]
- **Hidden-layer activation**: Sigmoid
- **Output-layer activation**: Sigmoid
- **Number of fitting parameters**: 20

Computational graph



$$y = S\left(\underline{\mathbf{w}}_{23}\, S\left(\underline{\mathbf{w}_{12}}\, S\left(\underline{\mathbf{w}_{01}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3\right)$$

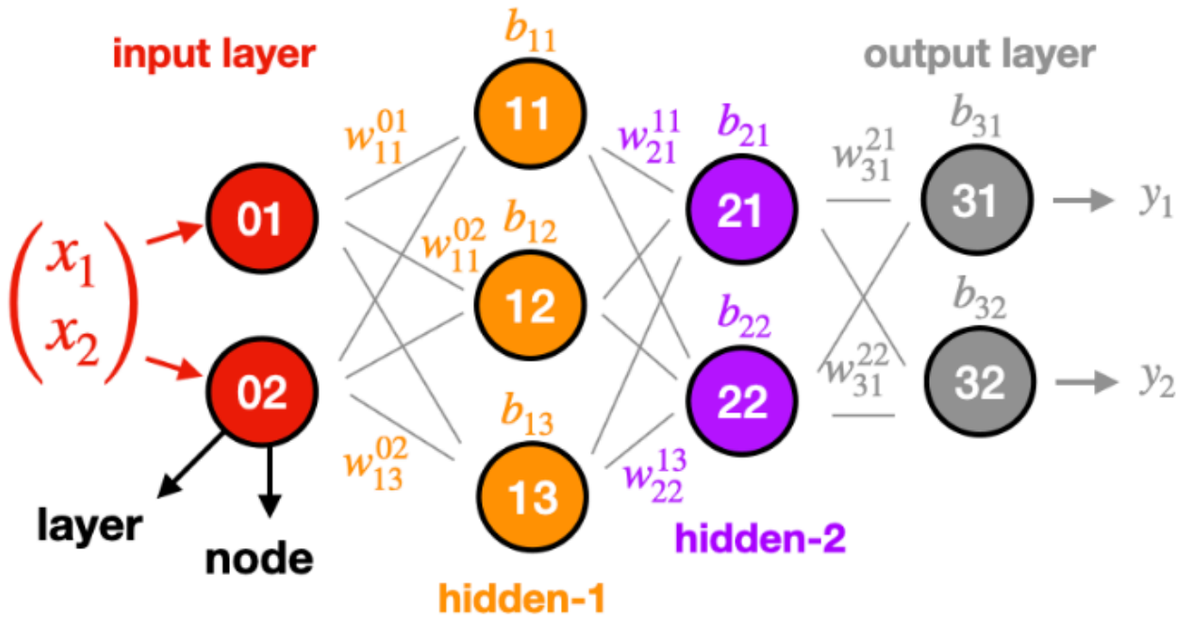
$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$

Note: (1) For the *hidden layers*, we didn't have to use `sigmoid`, other activations would also worked
(2) We only have one output node due to the binary constraint $1 = P_1 + P_2$

Deep MLP: Vector regression

- **Problem Type:** Vector regression
- **Mapping:** $\overset{\text{(input layer)}}{\mathbb{R}^2} \rightarrow \overset{\text{(output layer)}}{\mathbb{R}^2}$
- **Hidden layers:** $N_{hidden} = 2 \rightarrow [3,2]$
- **Hidden-layer activation**: ReLu
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 23

Computational graph



$$\mathbf{y} = (y_1, y_2) = \underline{\mathbf{w}}_{23}\, A\left(\underline{\mathbf{w}_{12}}\, A\left(\underline{\mathbf{w}_{01}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3$$

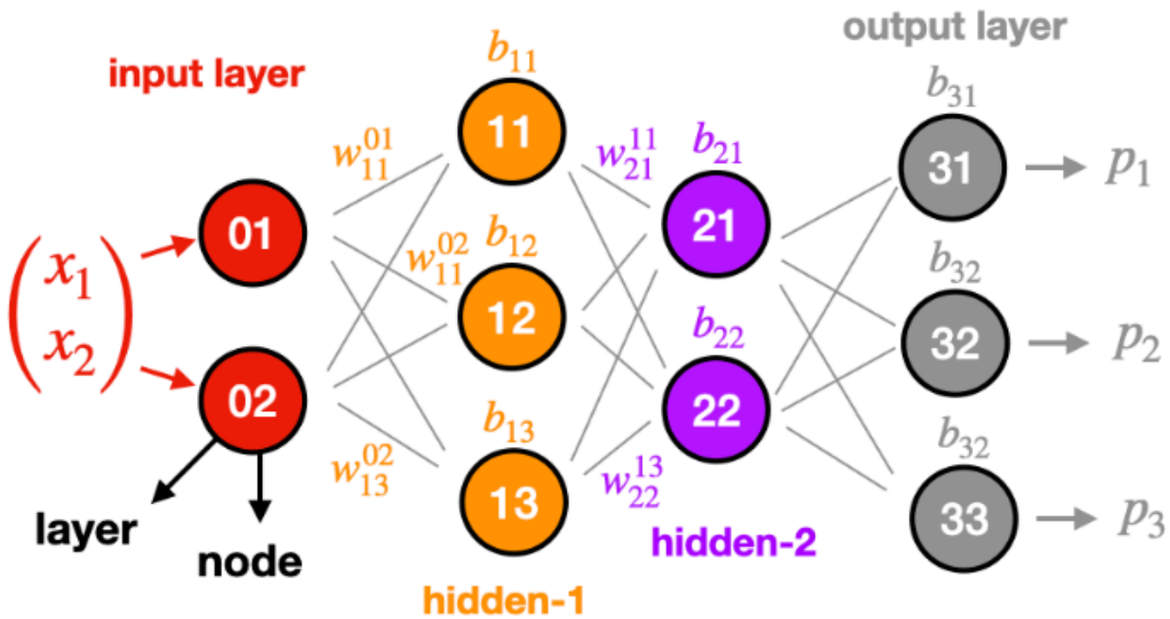
$$A(x) = max\,(0,x) \quad (relu)$$

Note: We didn't have to use `relu` on the *hidden layers*, other activations would also work

Deep MLP: Multi-class classification

- **Problem Type:** Multi-class $N_{class} = 3$
- **Mapping:** $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ (input layer) (output layer)
- **Hidden layers:** $N_{hidden} = 2 \rightarrow$ [3,2]
- **Hidden-layer activation**: ReLu
- **Output-layer activation**: Softmax
- **Number of fitting parameters**: 26

Computational graph



$$\mathbf{y} = (p_1, p_2, p_3) = A_2\left(\underline{\mathbf{w}_{23}}\, A_1\left(\underline{\mathbf{w}_{12}}\, A_1\left(\underline{\mathbf{w}_{01}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3\right)$$

$$A_1(x) = max\,(0,x) \quad (relu)$$

$$A_2(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_i}} \quad (softmax)$$

Note: We didn't have to use `relu` on the *hidden layers*, other activations would also work