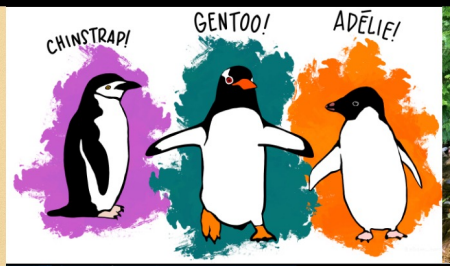
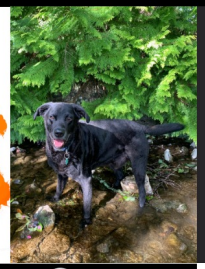
Neural Network Building Blocks

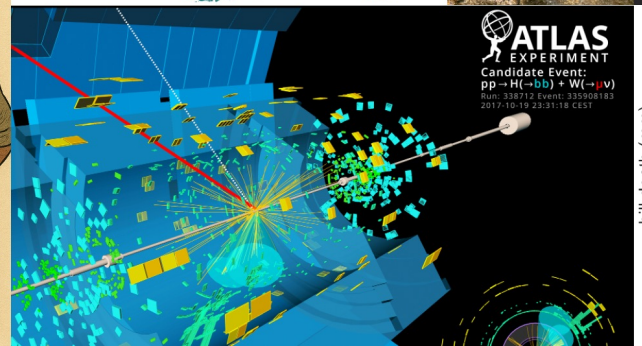
Deep Learning School “Basic Concepts”,
5th Edition, Aachen

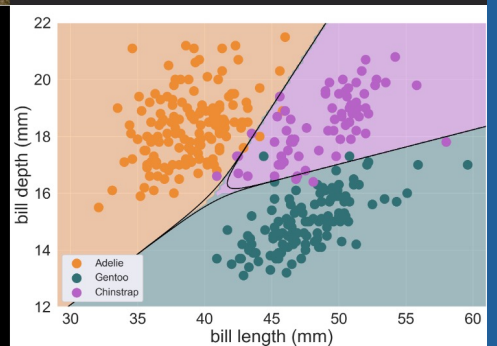
Introduction to Machine Learning for Physicists





```
import torch.nn as nn
N_features = len(input_features)
model = nn.Sequential(
    nn.Linear(N_features, 100),
    nn.ReLU(),
    nn.Linear(50, 1),
    nn.Sigmoid()
)
import torch.optim as optim
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.25)
```

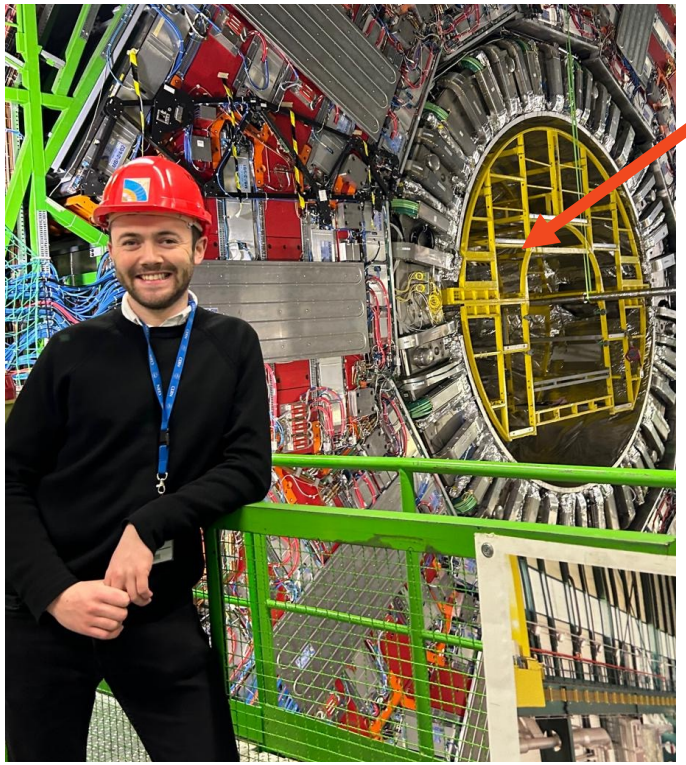




Ethan Simpson,
University of Manchester

Mikhail Romodin,
University of Tübingen

Persons



Not the ATLAS Experiment

Ethan,
University of Manchester +
ATLAS Experiment

Mikhail,
University of Tübingen

Contents

1. ML in the physical sciences
2. Software and computing aspects
3. Learning tasks
4. Linear models
5. Deep neural networks: theoretical and practical

GitHub

GitHub here

Aachen_Intro2NN Public

Pin | Unwatch 1 | Fork 0 | Star 1

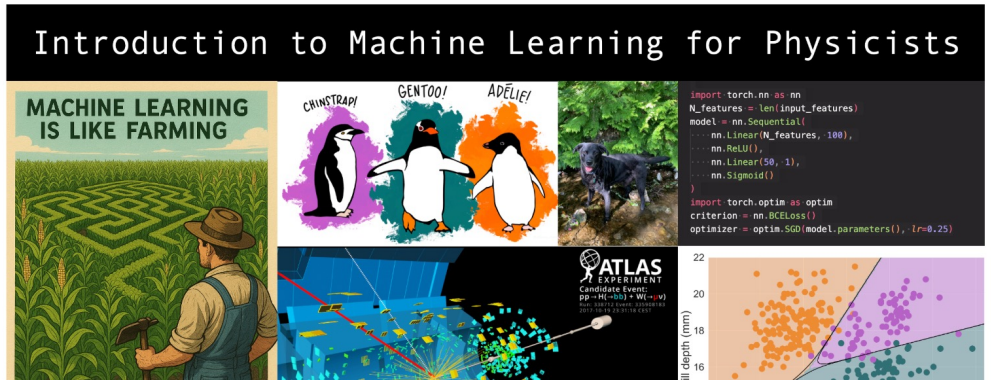
main | 3 Branches | 0 Tags | Go to file t | Add file | Code

els285 Update README.md — 8e78482 · 2 hours ago — 50 Commits

Exercises	Created using Colab	2 hours ago
data	Adding final task dataset	3 hours ago
images	Image of penguin on scales	3 hours ago
DNN4HEP_combined.ipynb	Adding nn.Module	3 days ago
DNN4HEP_combined_exercise.ipynb	Remove solutions and add instructions	3 days ago
README.md	Update README.md	2 hours ago

README

Neural Network Building Blocks - 2025 Deep Learning School "Basic Concepts"



About

No description, website, or topics provided.

Readme
Activity
1 star
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2

els285 Ethan Simpson
HerrHorizontal Maximilian Horzela

Languages

Jupyter Notebook 100.0%

Exercises

Exercises are iPython notebooks.

You can run them on Google Colab or download to your local machine.

Please try your preferred setup now…

Resources

PyTorch tutorials

HSF Deep Learning

Post questions on Google Docs

“Are you sitting comfortably? Then we’ll begin…”

Introduction

AI vs ML vs DL

Mat Velloso 🇺🇦
@matvelloso

Difference between machine learning and AI:

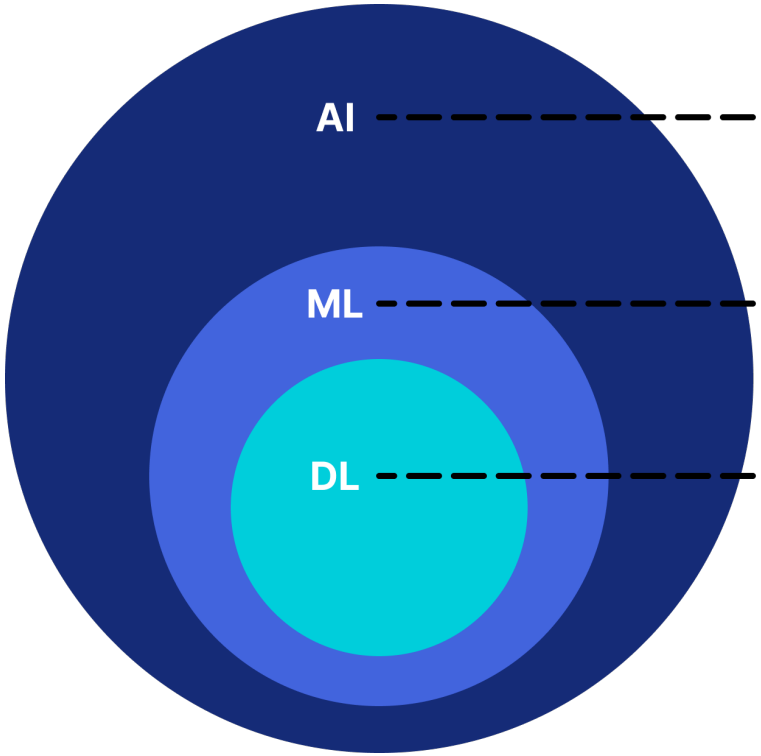
If it is written in Python, it's probably machine learning

If it is written in PowerPoint, it's probably AI

3:25 AM · Nov 23, 2018 · Twitter Web Client

8,264 Retweets **911** Quote Tweets **23.8K** Likes

AI vs ML vs DL



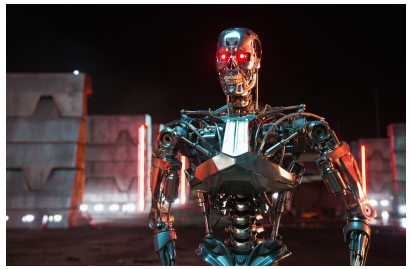
Artificial Intelligence
Engineering of making intelligent machines and programs.

Machine Learning
Ability to learn without being explicitly programmed.

Deep Learning
Learning based on deep neural networks.

neurosnap.ai

EXPECTATION



REALITY

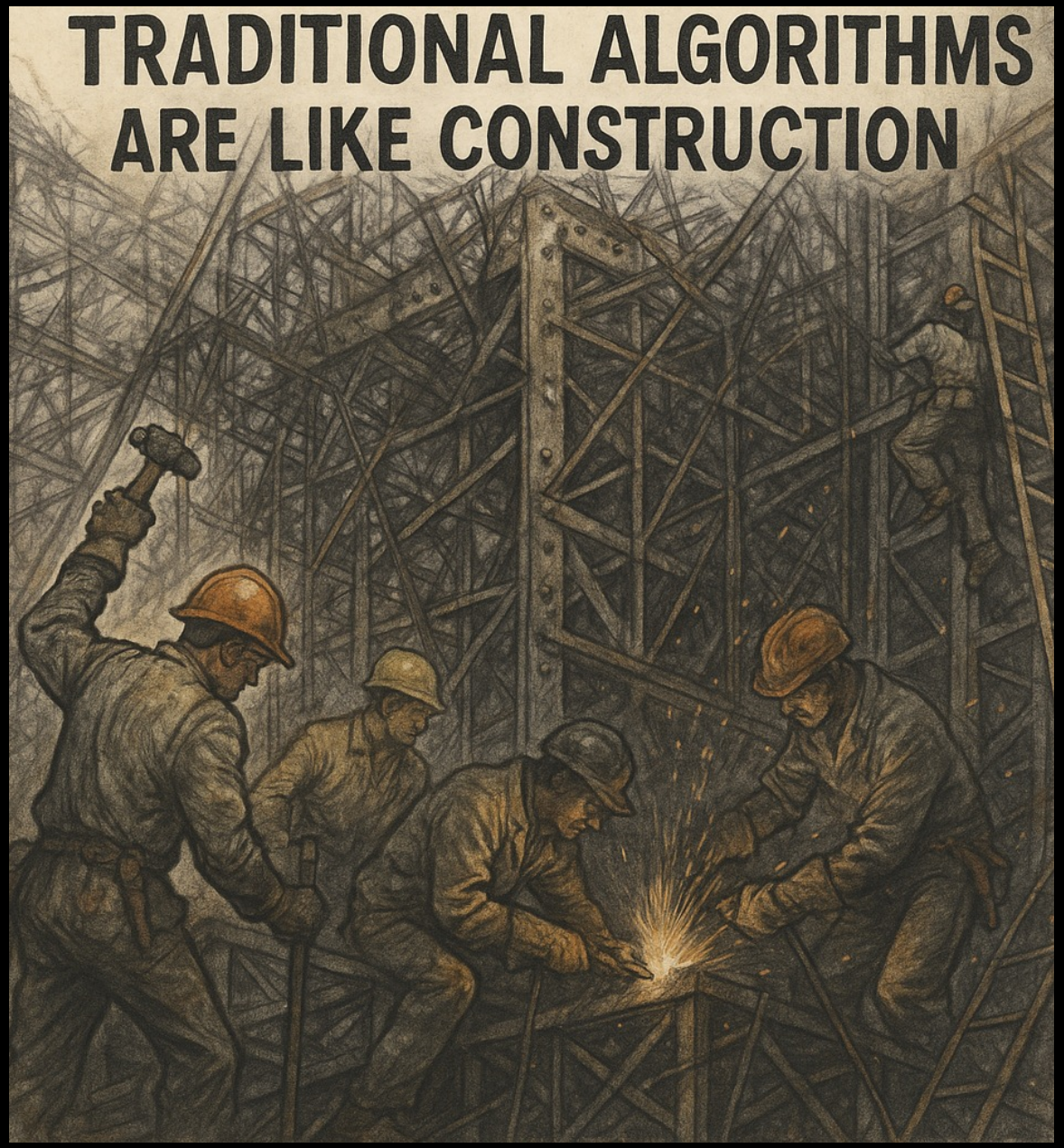
```
from torch import nn
```

Premise

Build a tool (algorithm) to achieve a particular goal.

Specifics of the algorithms are learned from examining data

In contrast, traditional algorithms are built by hand – in many cases ML outperforms “the old way”


TRADITIONAL ALGORITHMS
ARE LIKE CONSTRUCTION


MACHINE LEARNING
IS LIKE FARMING

Maths

ML algorithms are parameterized functions from an input space to an output space.

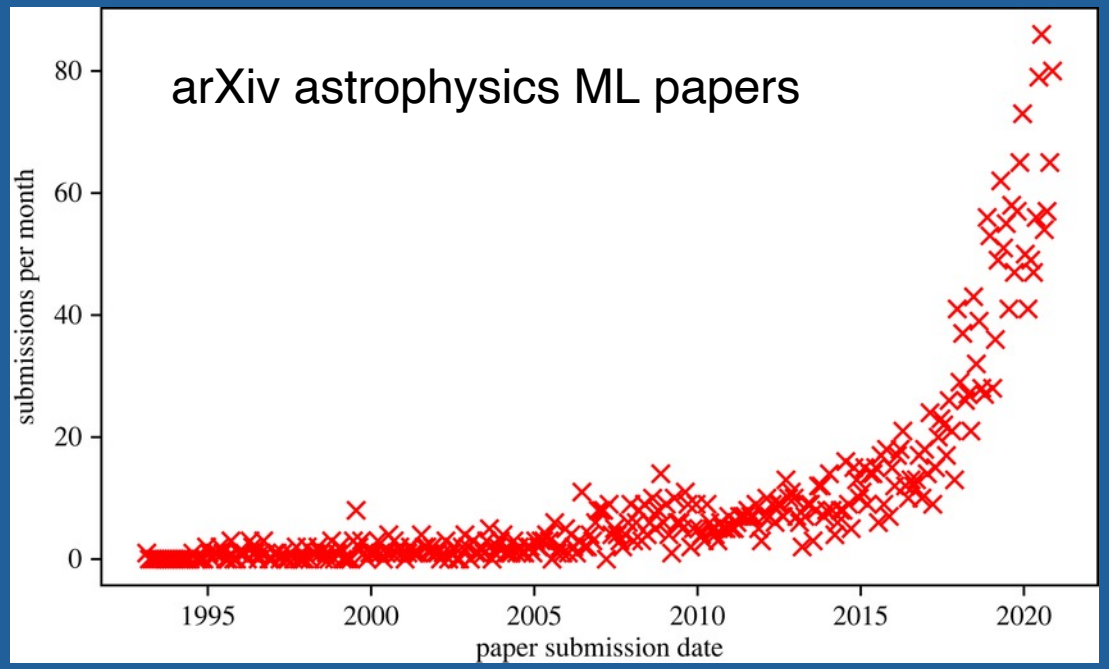
$$f_\theta : \mathcal{X} \to \mathcal{Y}$$

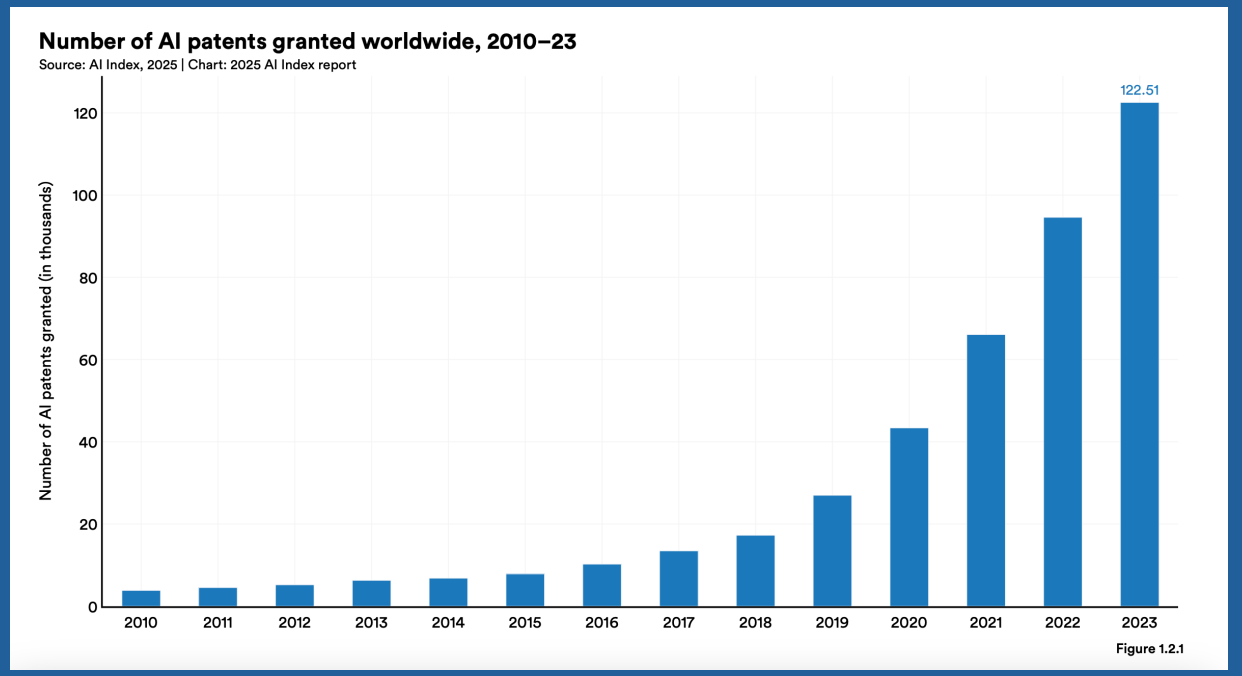
We learn optimal parameters which best align predictions with examples in a dataset.

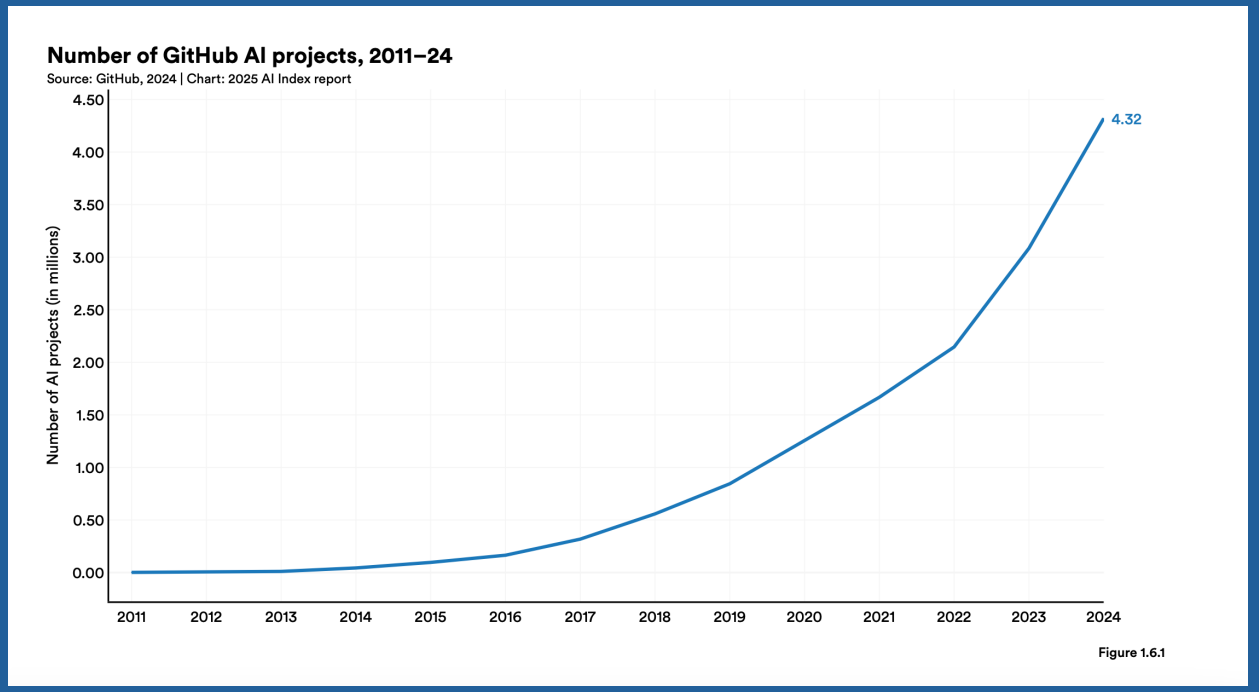
$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{C}(f_\theta, \mathcal{D})$$


NEURAL NETWORK

Why ML now?


arXiv astrophysics ML papers
submissions per month
0
20
40
60
80
1995
2000
2005
2010
2015
2020
paper submission date

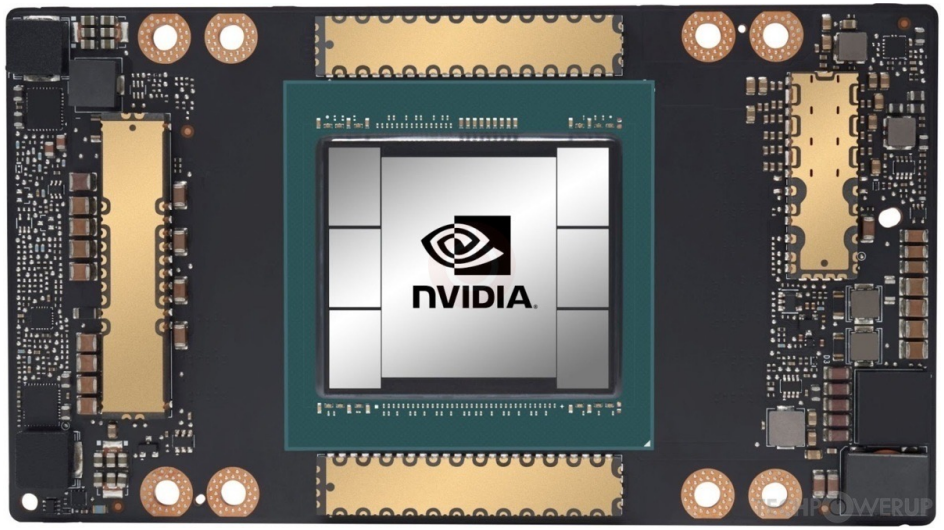

Number of AI patents granted worldwide, 2010–23
Source: AI Index, 2025 | Chart: 2025 AI Index report
Number of AI patents granted (in thousands)
0
20
40
60
80
100
120
122.51
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Figure 1.2.1


Number of GitHub AI projects, 2011–24
Source: GitHub, 2024 | Chart: 2025 AI Index report
Number of AI projects (in millions)
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
4.50
4.32
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
Figure 1.6.1

Why ML now?

Primary

- Access to large datasets
- Increased computational power

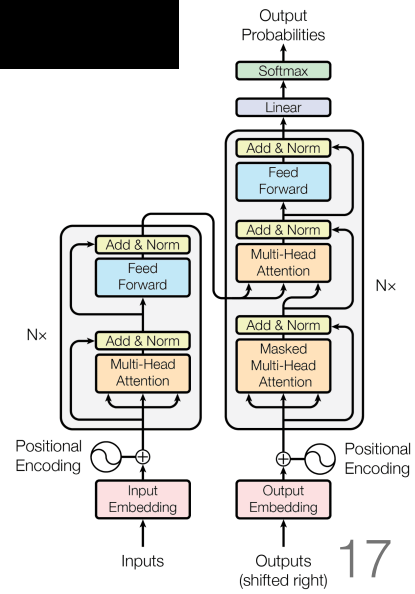


Secondary

- Breakthroughs in algorithms
- Open-source libraries
- Companies doing AI
- Economic and political motivation

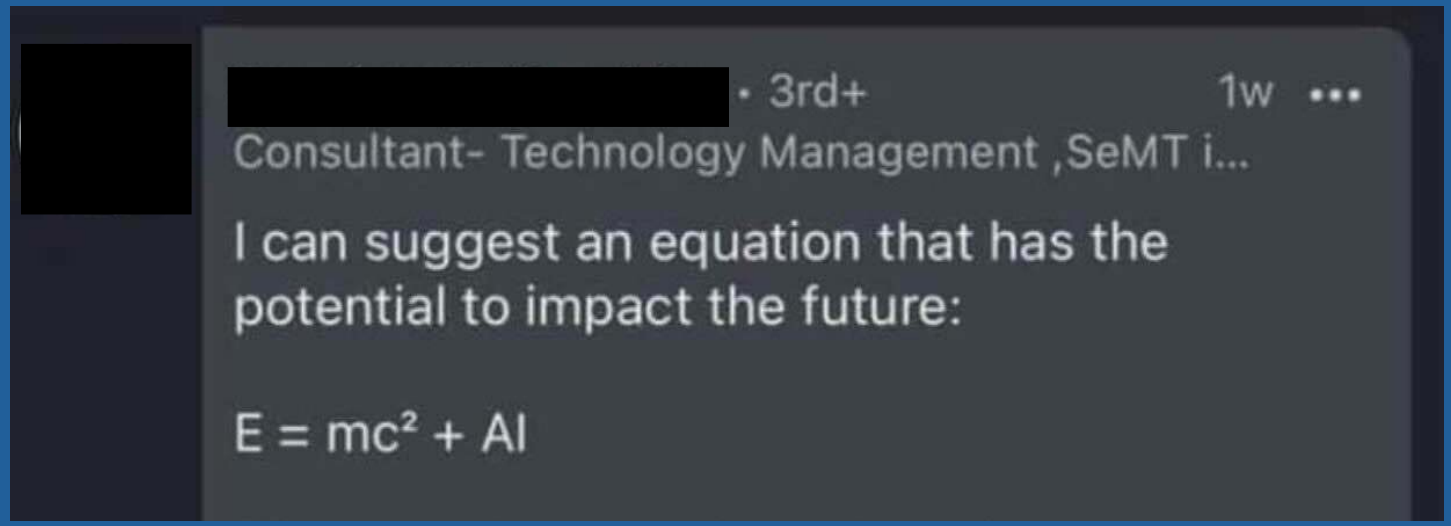
AI in the 21st Century

- ImageNet, 2009 – Fei-Fei Li creates dataset
- AlexNet, 2012 – Alex Krizhevsk's model for images uses deep learning (CNN)
- Tensorflow, 2015 – Made open-source
- AlphaGo, 2016 – Beats Lee Sidol 4-1
- Transformer, 2017 – "*Attention is all you need*"
- AlphaFold2, 2020 – Wins CASP competition
- ChatGPT, 2022 – Commercial LLM

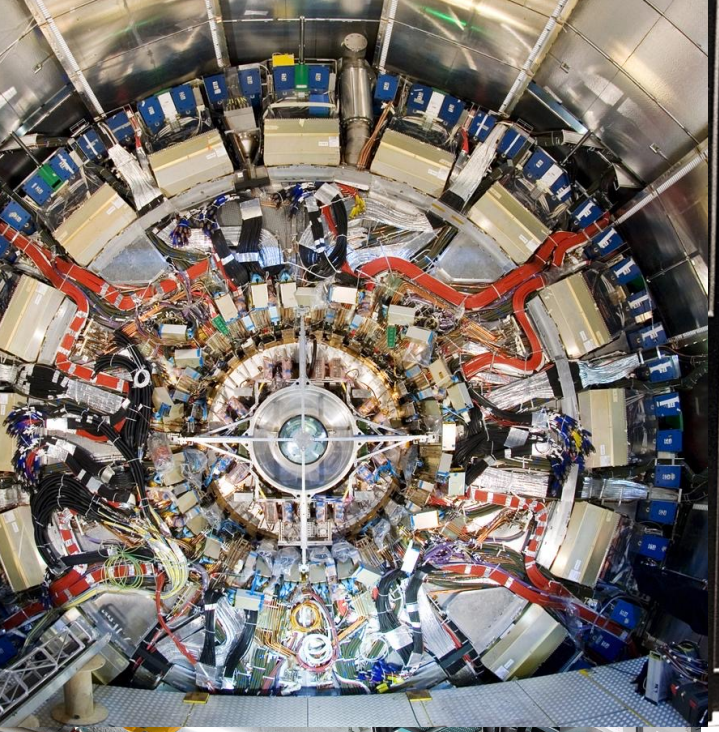


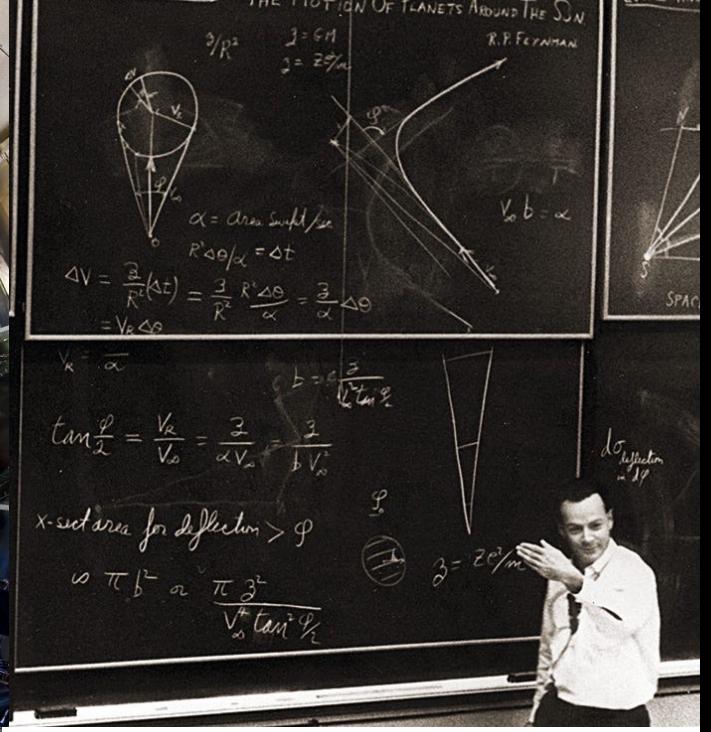
ML in the Physical Sciences

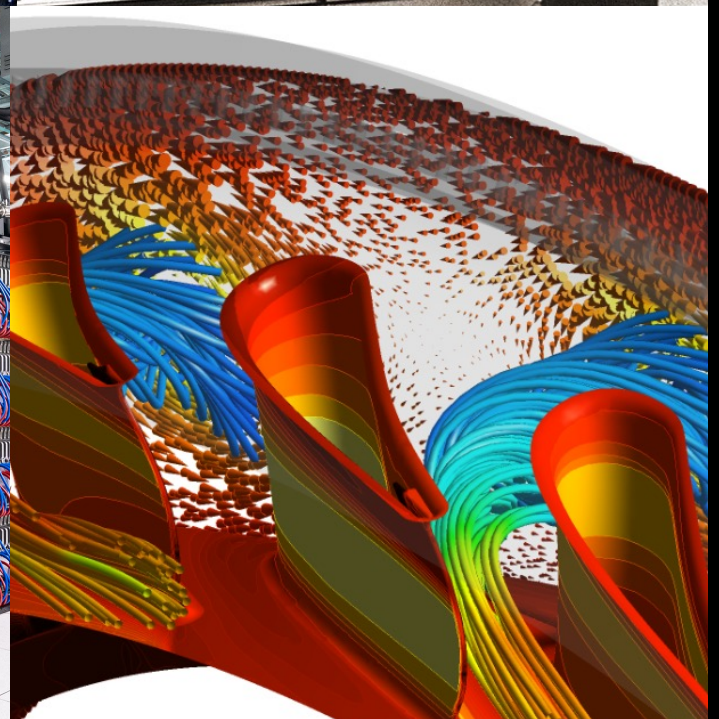
ML


[redacted] · 3rd+ 1w
Consultant- Technology Management ,SeMT i...
I can suggest an equation that has the potential to impact the future:
E = mc² + AI

ces



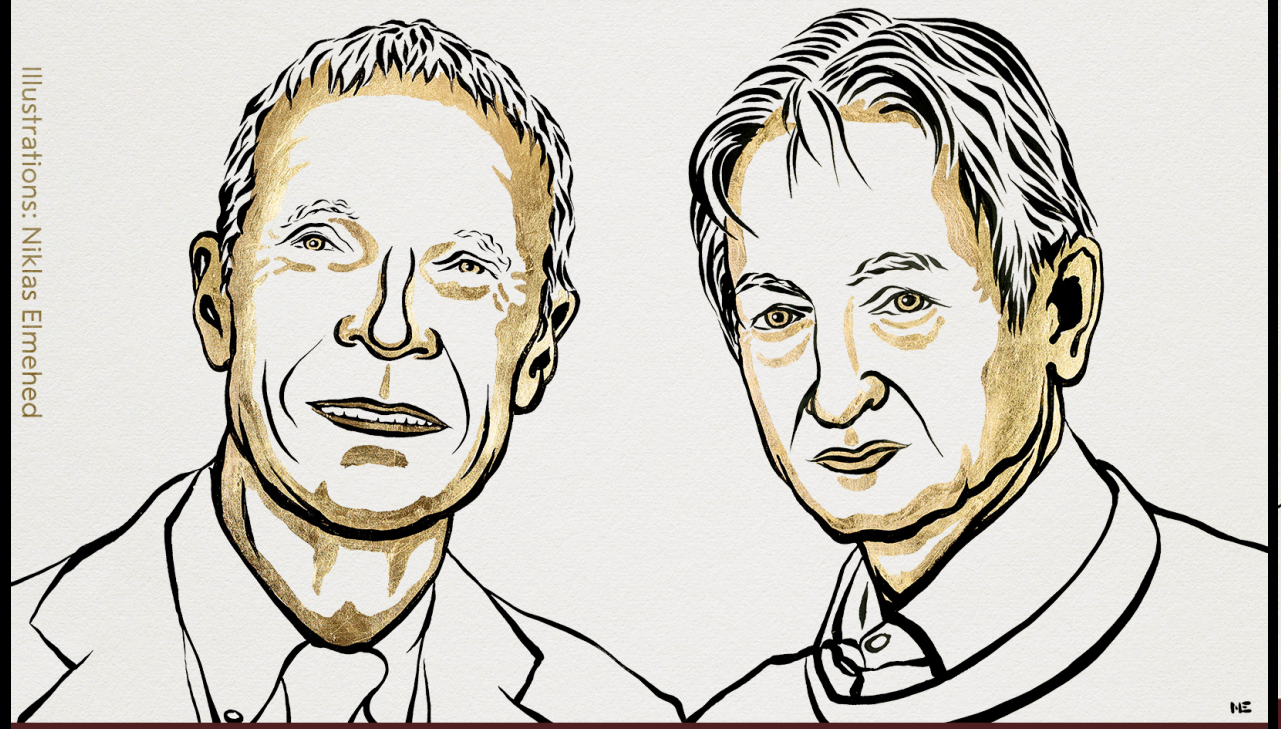




Machine Learning ❤️ Physical Science

- Big experimental data
- Data obeys laws
- Know how to generate
- Historical symbioses with computing

THE NOBEL PRIZE IN PHYSICS 2024



Illustrations: Niklas Elmehed

John J. Hopfield Geoffrey E. Hinton

"for foundational discoveries and inventions that enable machine learning with artificial neural networks"

THE ROYAL SWEDISH ACADEMY OF SCIENCES

THE NOBEL PRIZE IN CHEMISTRY 2024

Illustrations: Niklas Elmehed

David Baker

"for computational protein design"

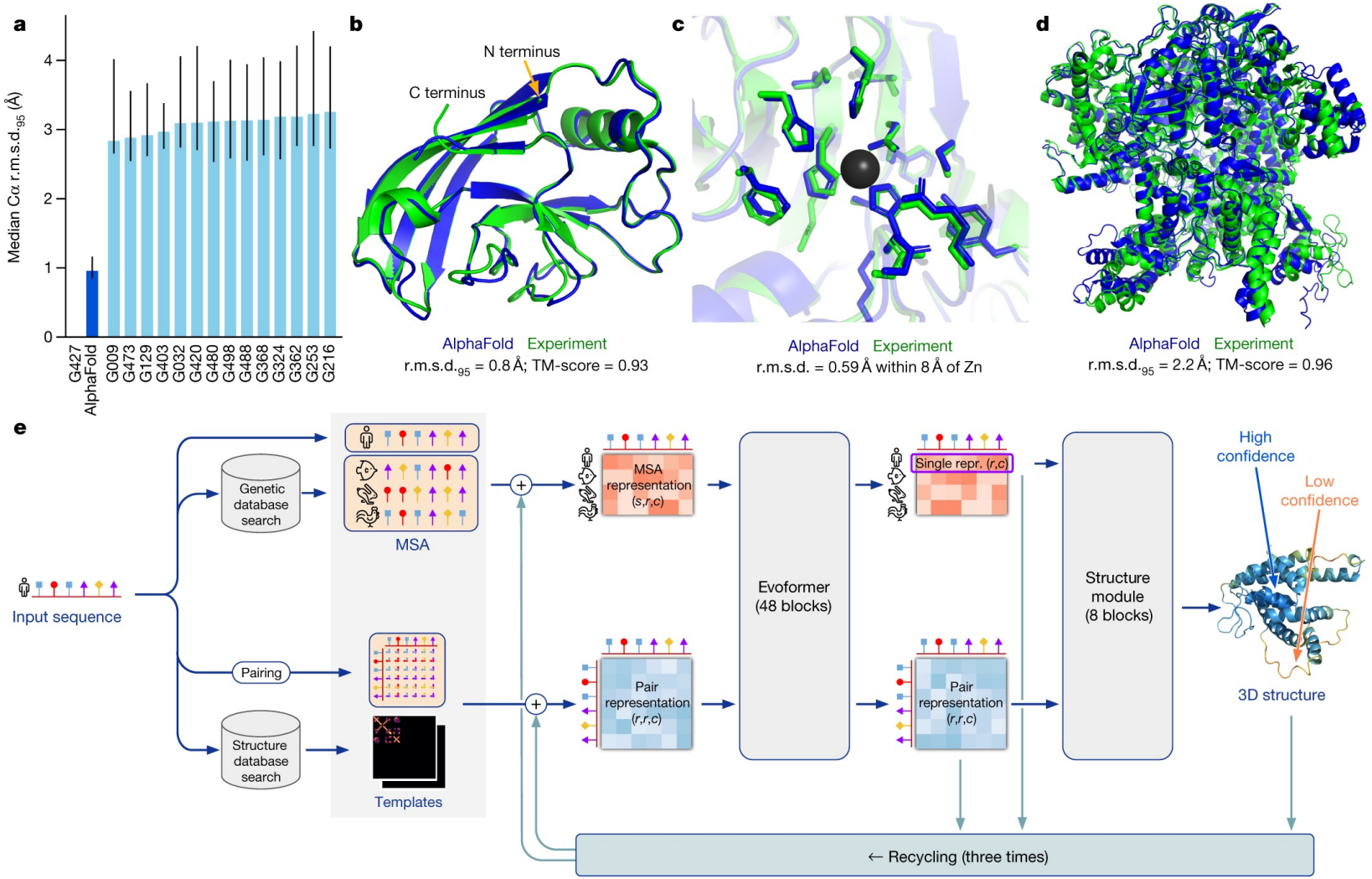
Demis Hassabis John M. Jumper

"for protein structure prediction"

THE ROYAL SWEDISH ACADEMY OF SCIENCES

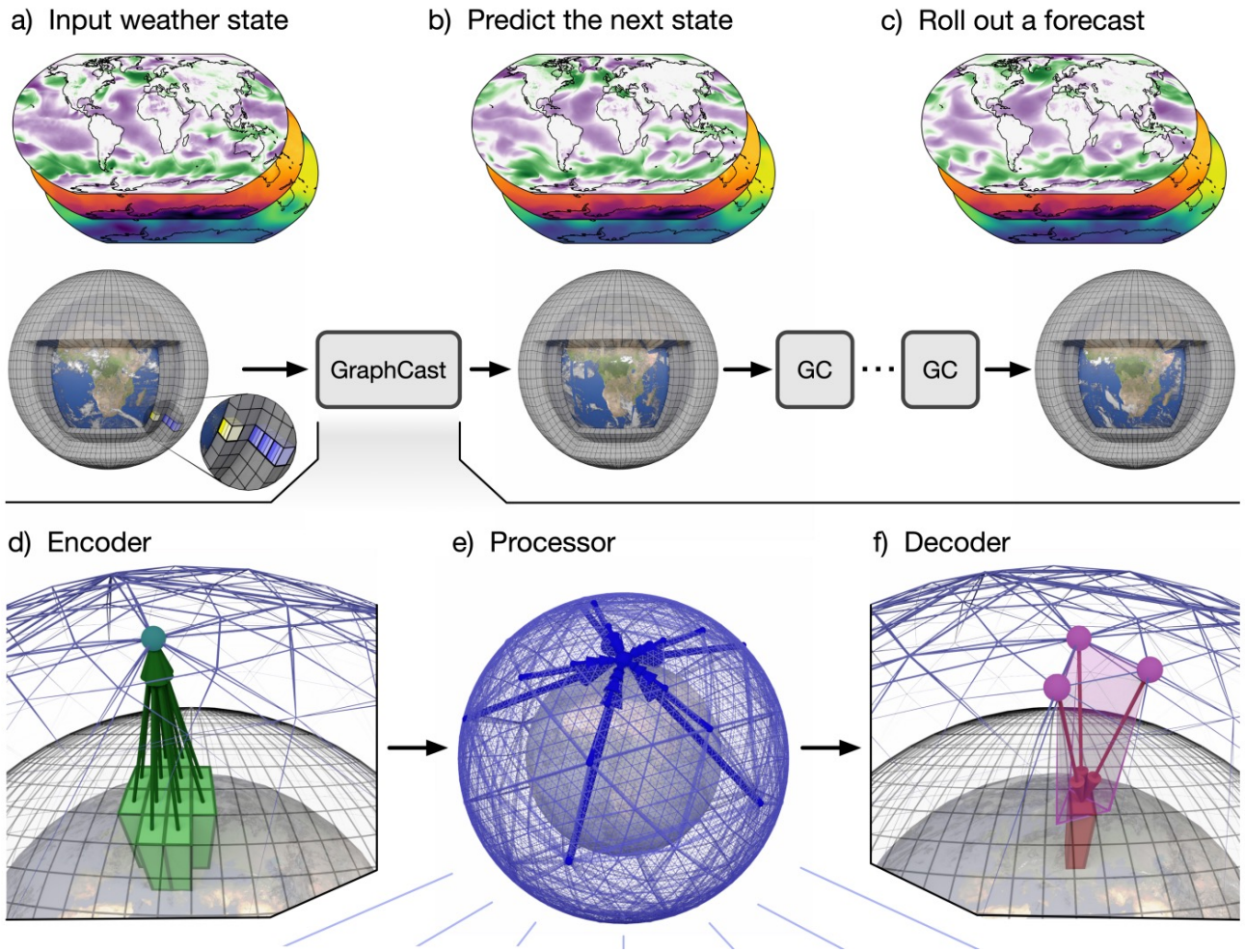
AlphaFold

Nobel Prize-winning model for predicting 3D protein structure



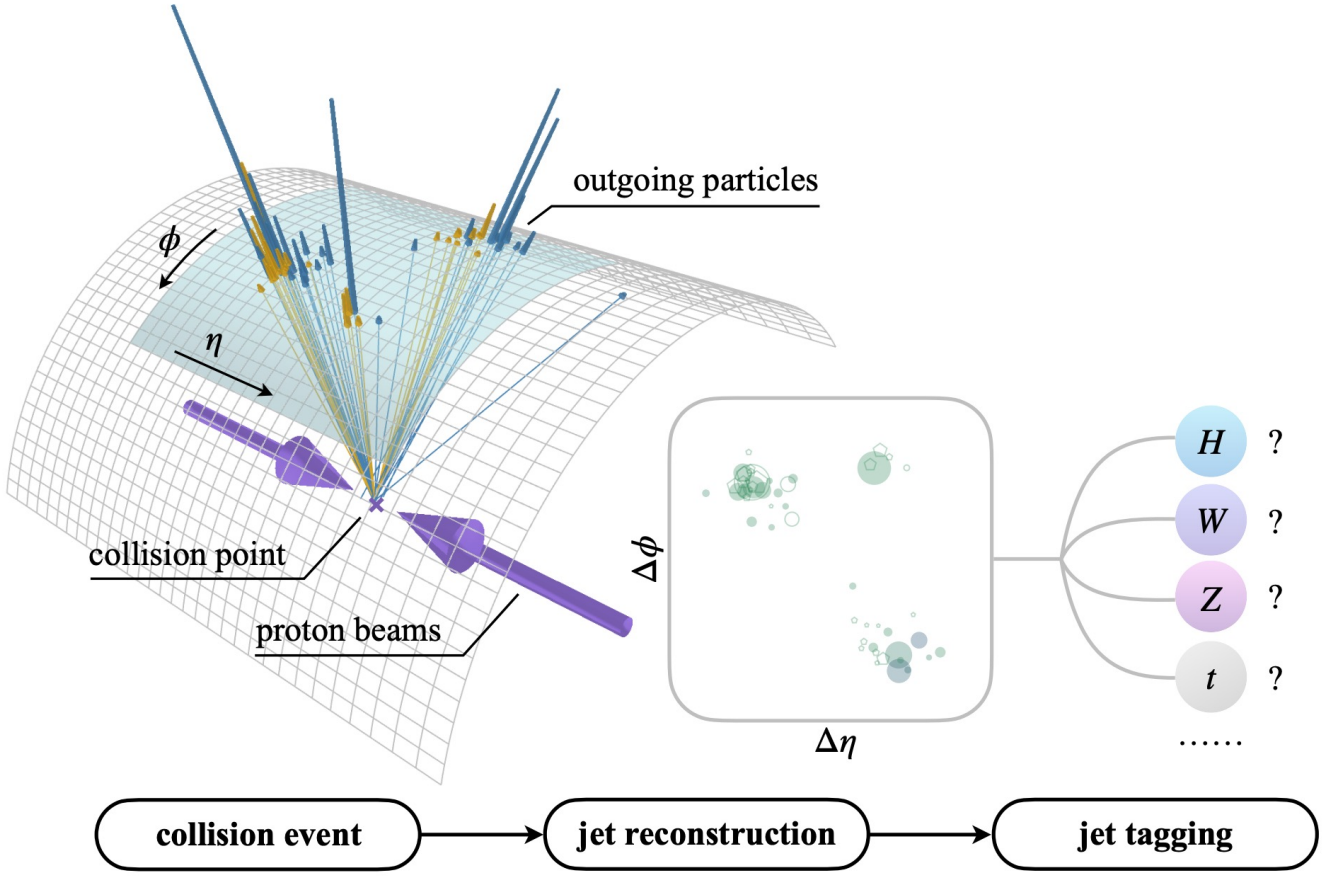
GraphCast

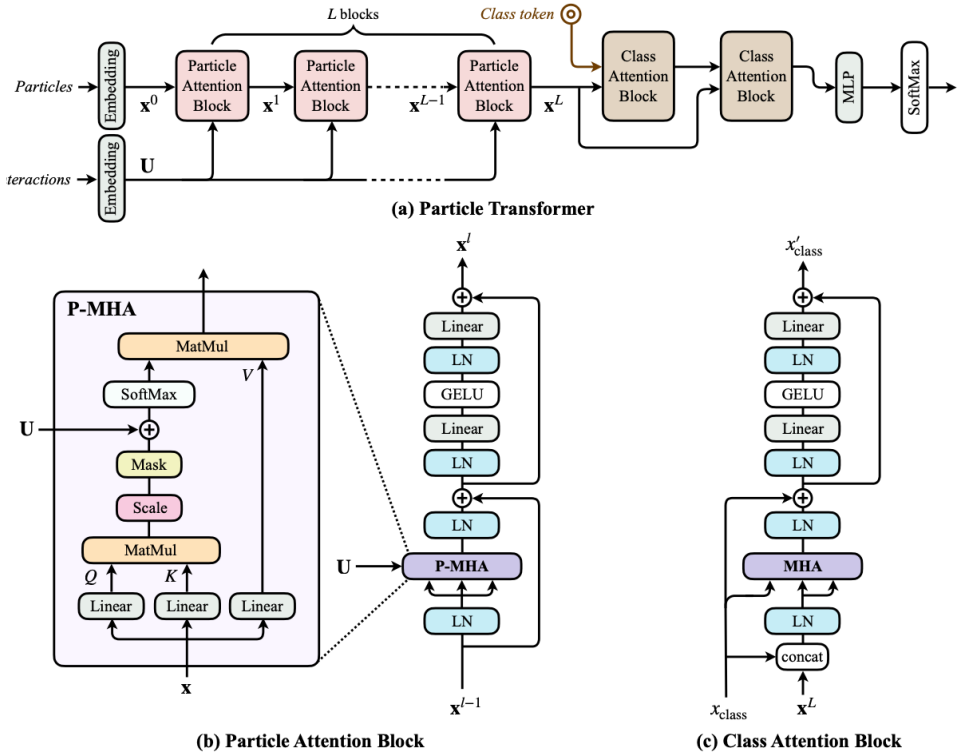
Graph learning model for advanced weather prediction



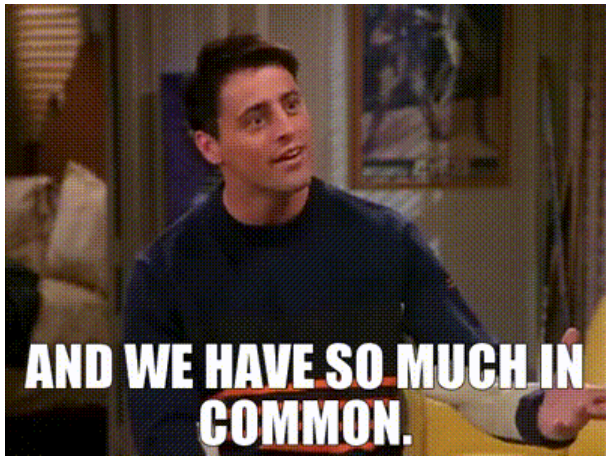
Particle Transformer

SOTA transformer model for large jet tagging at the LHC



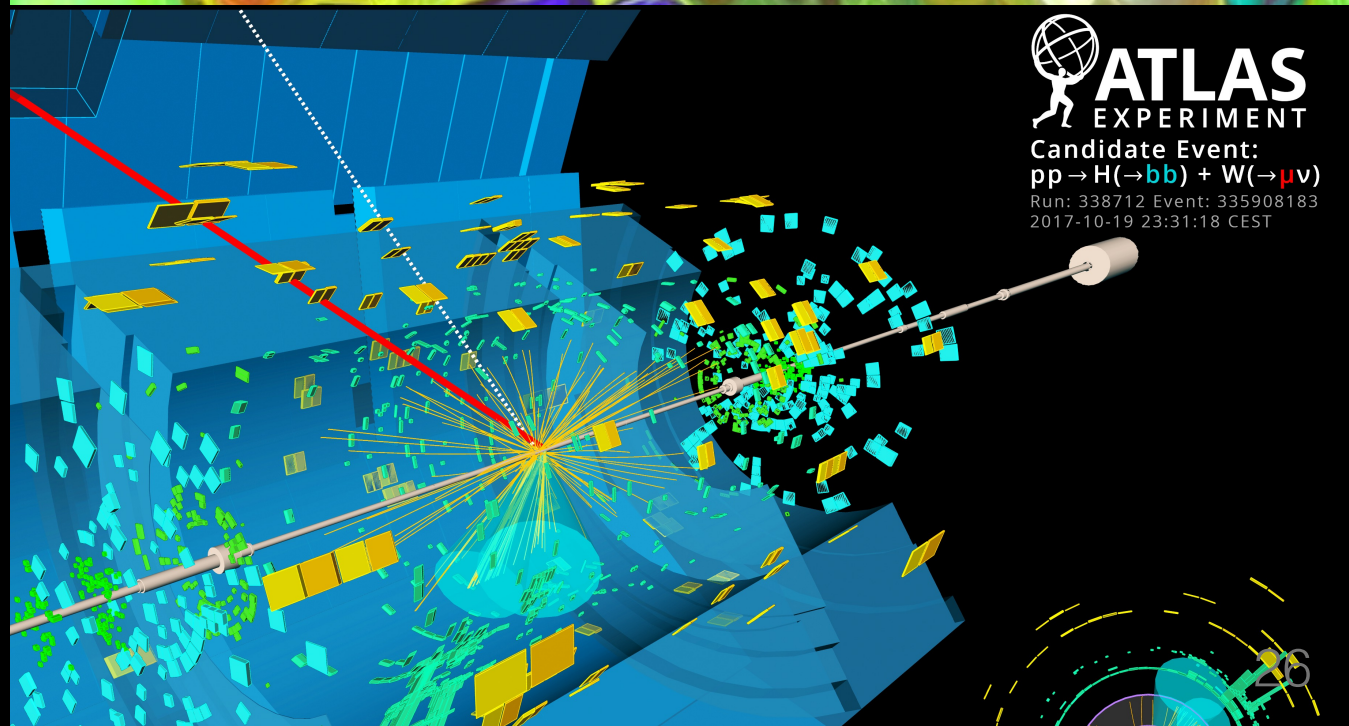


Commonalities



- A clearly defined task
- Trained on huge, realistic dataset
- Large number of trainable parameters
- Encodes physics into abstract representation

Physical Science
Data



Physical Science

Data

Large

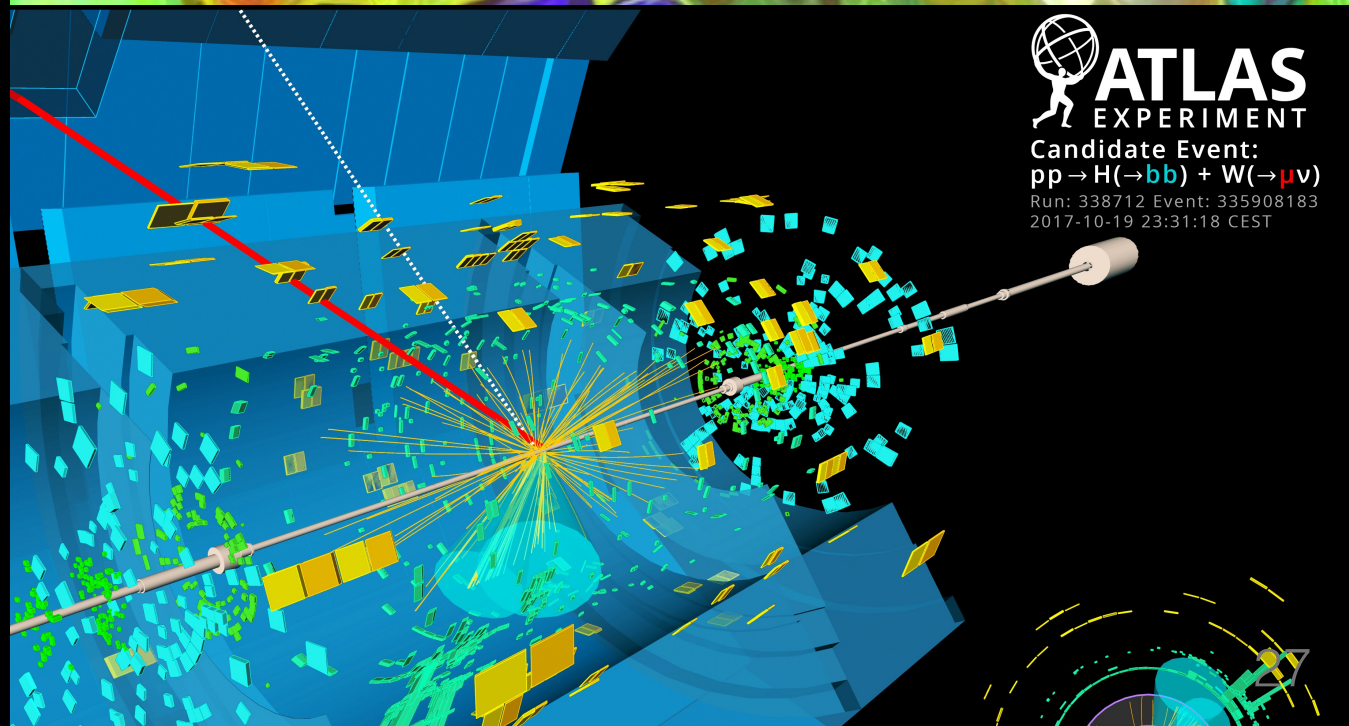
Complex

High-dimensional

Structured

Correlated

Empirical



Software and computing

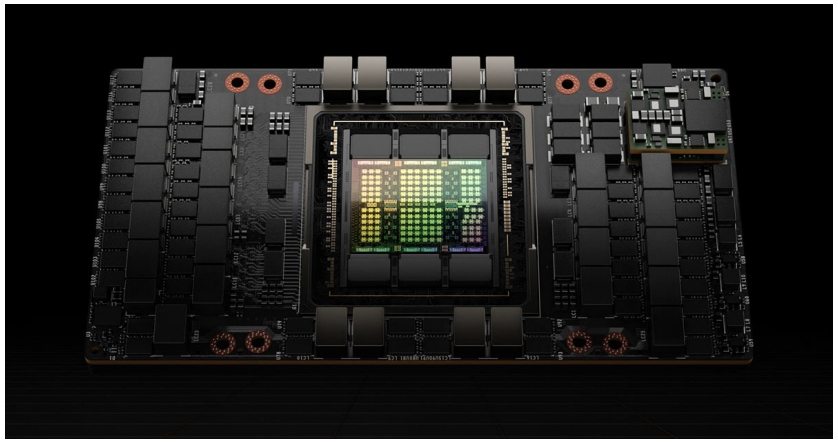
GPUs

(GEMM)

Highly efficient at parallel processing of matrix operations

CPU – small number of very powerful processor cores

GPU – Tens of thousands of smaller cores



Python Frameworks

- Best for classic ML models
- Simple interface
- Great for learning
- Useful data processing functions

TensorFlow

- Built by Google (TF1 2015; TF2 2019)
- Complex
- Simple Keras API for NNs
- Most popular in industry
- Popularity weining?

Python Frameworks

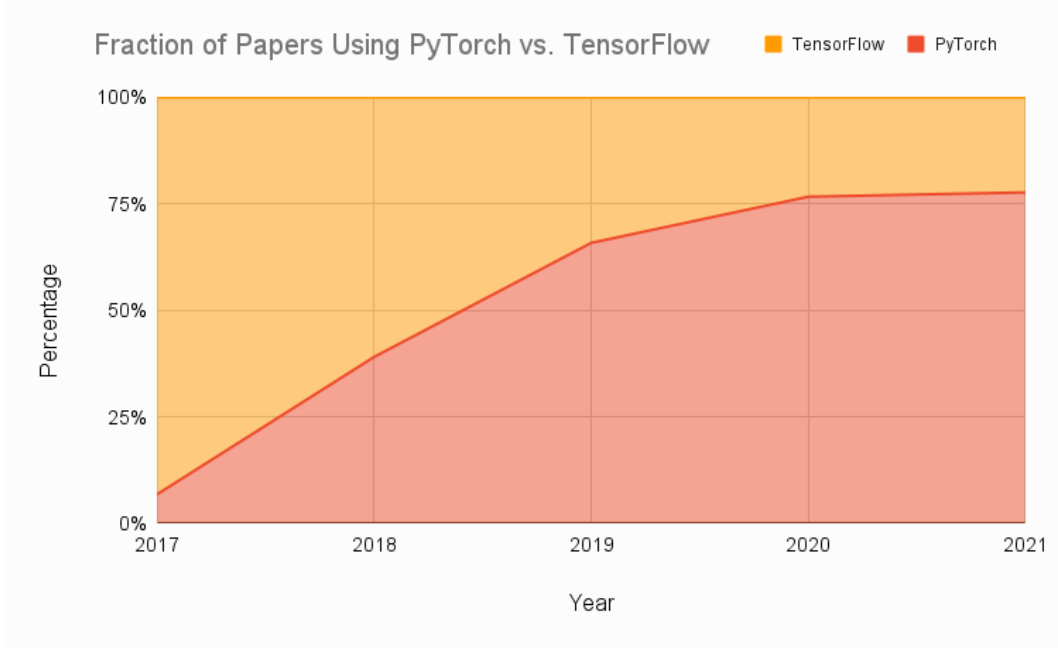
PyTorch

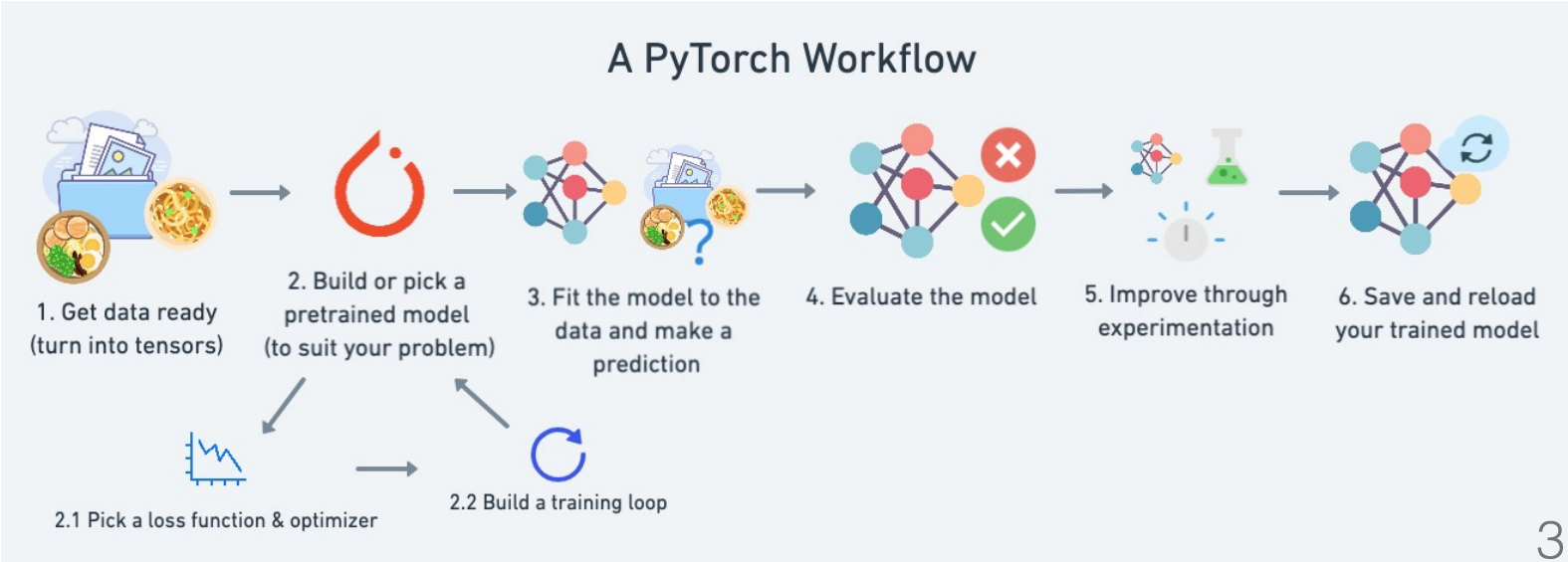
- Built by Meta and Microsoft
- Powerful and easy to use
- Most popular in research
- Dynamic computation graph
- Balance between ergonomic and power

- Built by Google
- Focused more on arrays and automatic differentiation
- Different approach to other tools
- Used in specific research groups (Google mostly)

PyTorch

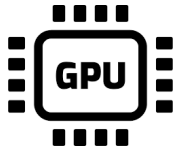
- Tensors compatible with GPUs
- Autograd for automatic differentiation
- Dynamic computation graph



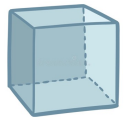


Tensors

Tensors are just arrays

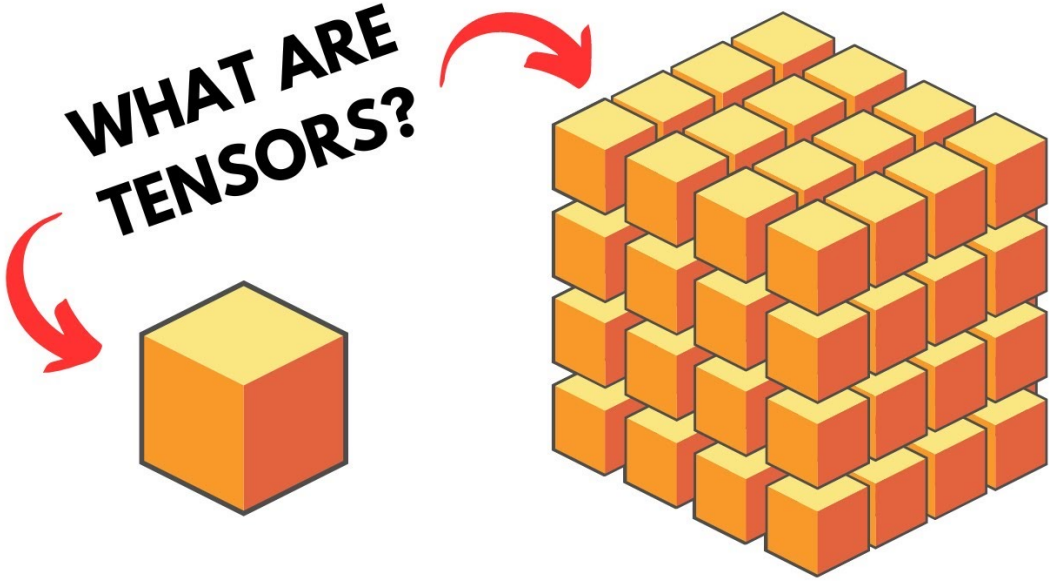


Can reside on GPUs



Have arbitrary dimension

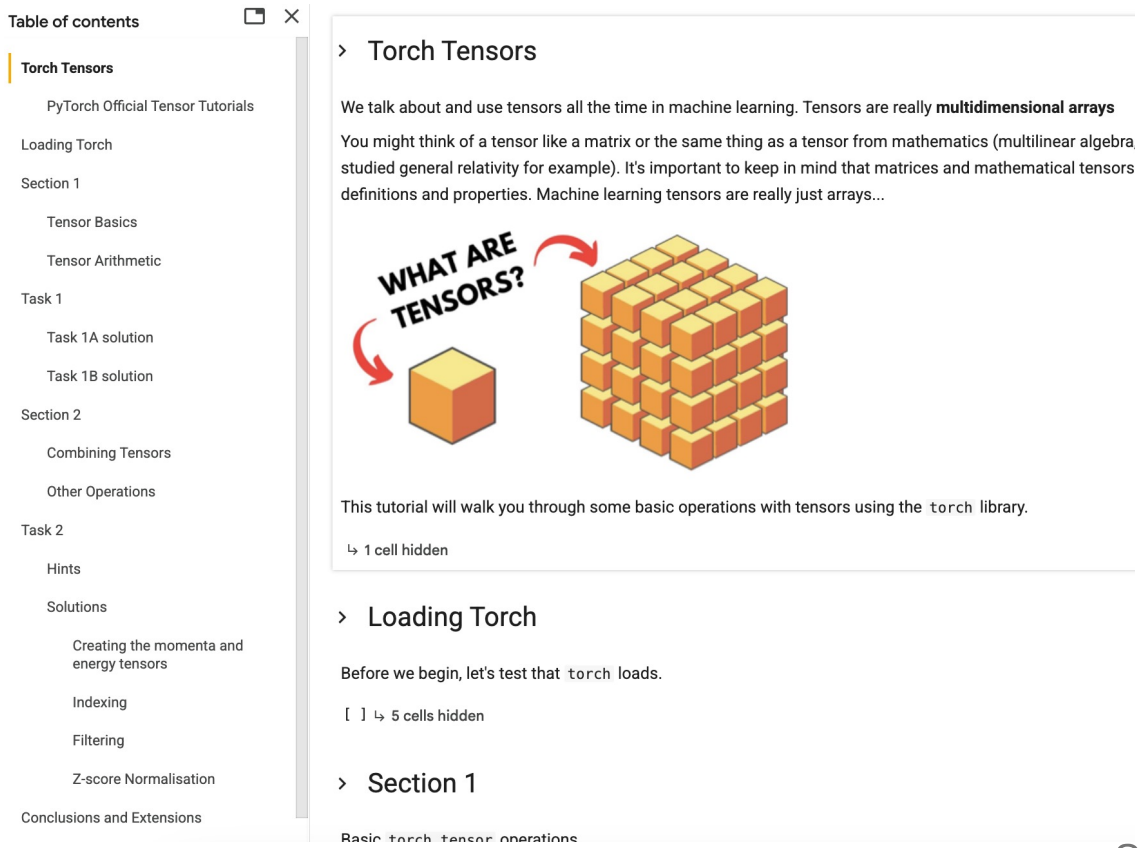
Optimised for mathematical operations



Activity 1 Torch Tensors

Tasks:

- Two sections on basics of tensors
- Two exercises to let you think about what you've learned
- This is designed to take maybe 25 minutes

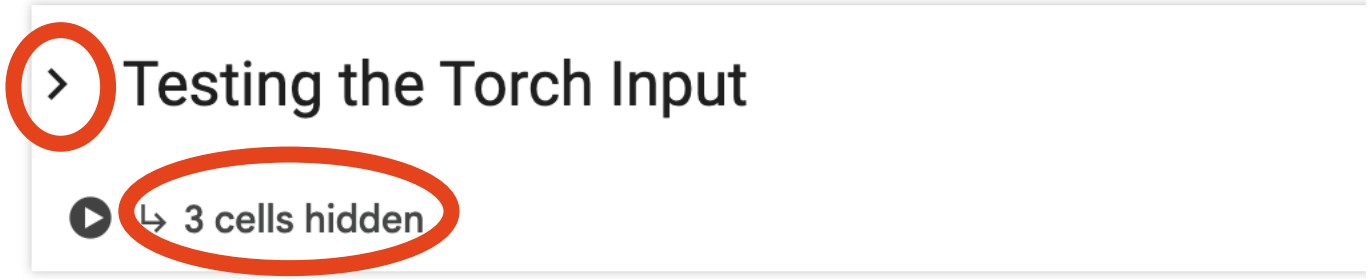


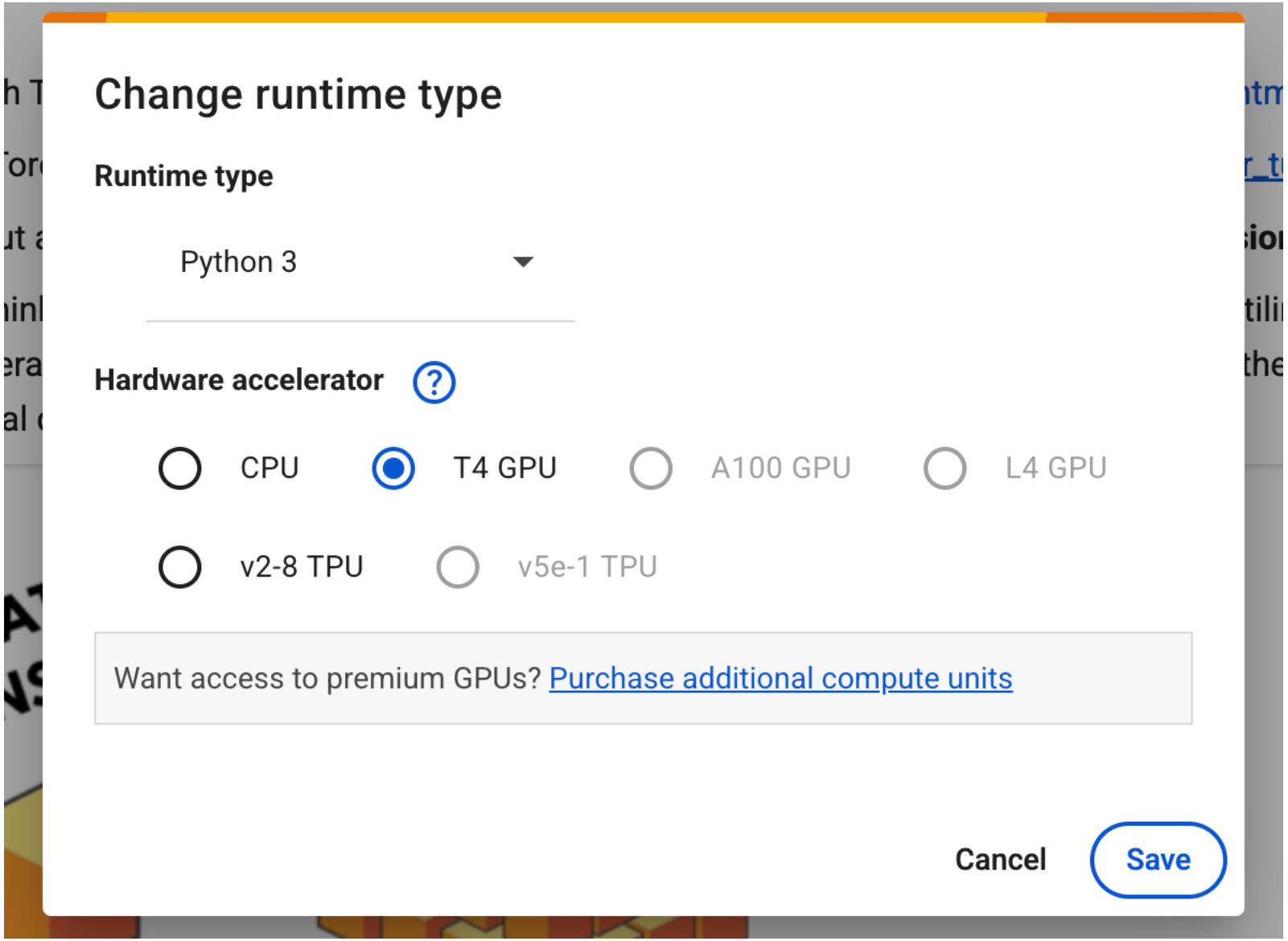
Tutorial Tips

Run the notebooks in Google Colab.

Alternatively, run in VSCode: you may not have access to a GPU.

We have by default suppressed all sections to keep the notebooks neat. You can expand each section by clicking either:




Change runtime type
Runtime type
Python 3
Hardware accelerator
CPU
T4 GPU
A100 GPU
L4 GPU
v2-8 TPU
v5e-1 TPU
Want access to premium GPUs? Purchase additional compute units
Cancel
Save

Time check – 1 hour in?

Learning tasks

Learning Task Taxonomy

Supervised

Data is labelled

- Classifying collider events
- Image recognition

Semi-supervised

Sparse labelling

- Protein structure prediction
- Text document classification

Unsupervised

Data is labelled

- Anomaly detection
- Clustering data

Reinforcement

Interacting, rewarded agent

- Controlling tokomak plasma
- Autonomous driving

Supervised Learning

A labelled dataset

CAT

DOG

CAT

DOG

Latent Concept

🧠 There is an abstract, latent idea of what a dog and a cat are…

(Is this a product of high-level thinking?)

← Our data are individual realisations of this concept

🤖 Our ML model aims to learn an analogue of this concept

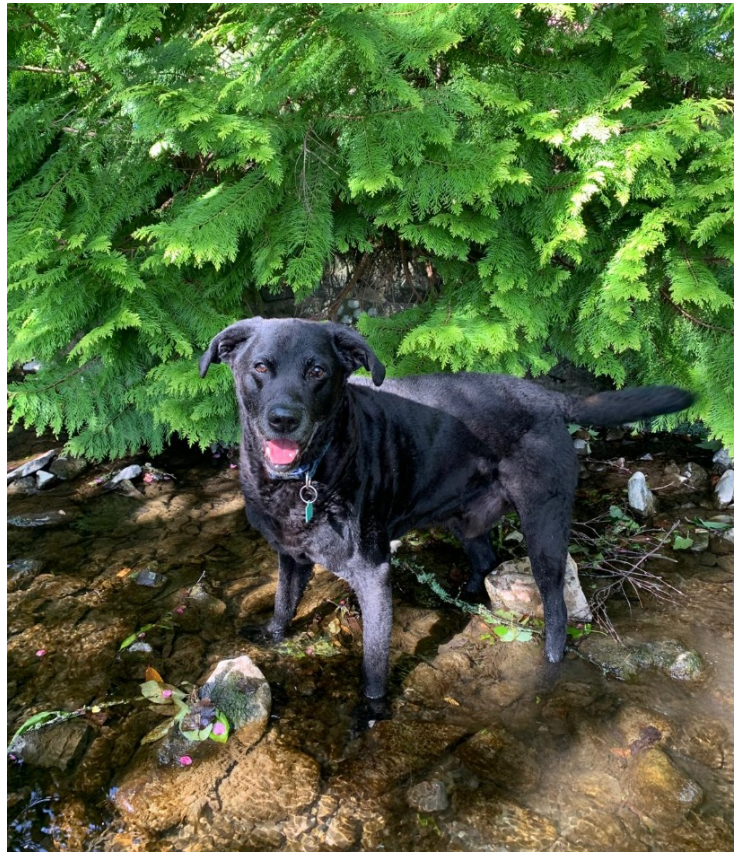
Latent Concept

A model which achieves this will generalize better to unseen data

The model does not understand "what a dog is" in the sense that a human does…

It learns a statistical representation, which correlates with our concept

Unseen data



Latent Concept

You are one of the most complex deep learning models. Do you think you have any idea of these latent concepts in the human-semantic sense, or have you learned very complex statistical representations?

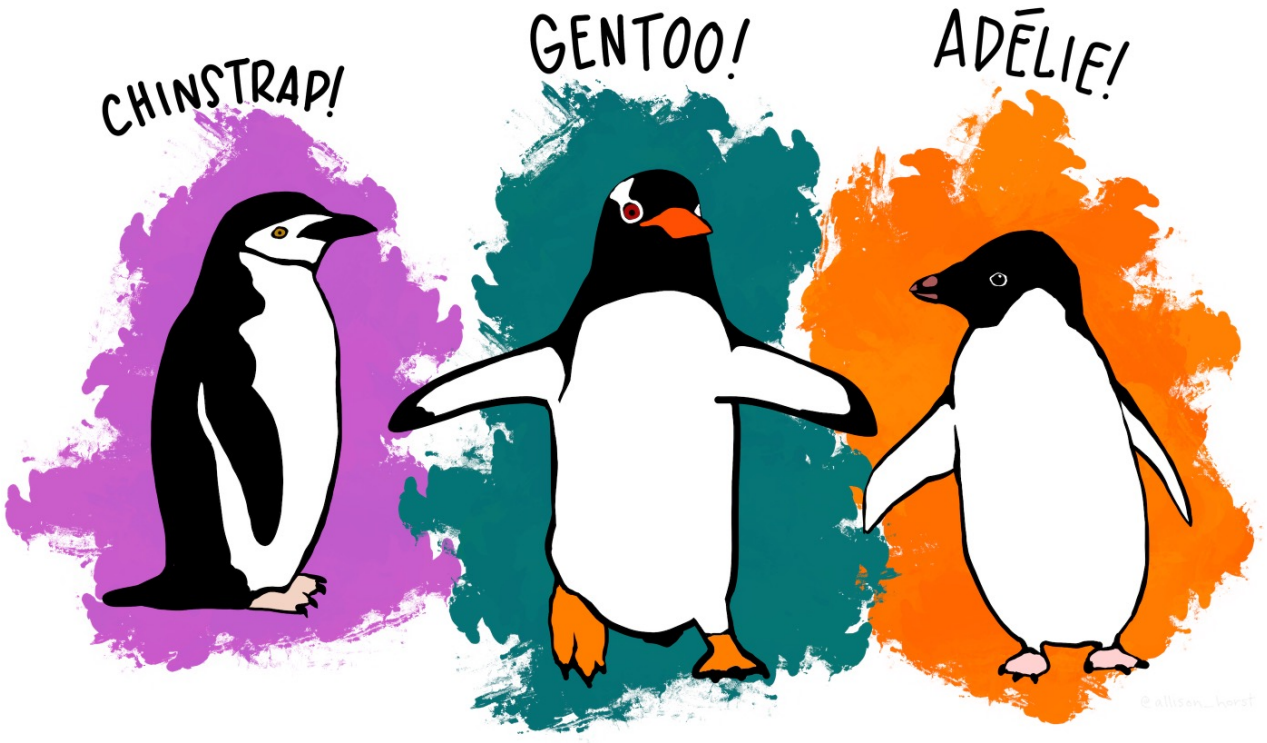
This is a foundational and profound question—and one that gets at the core of what models like me *are*.

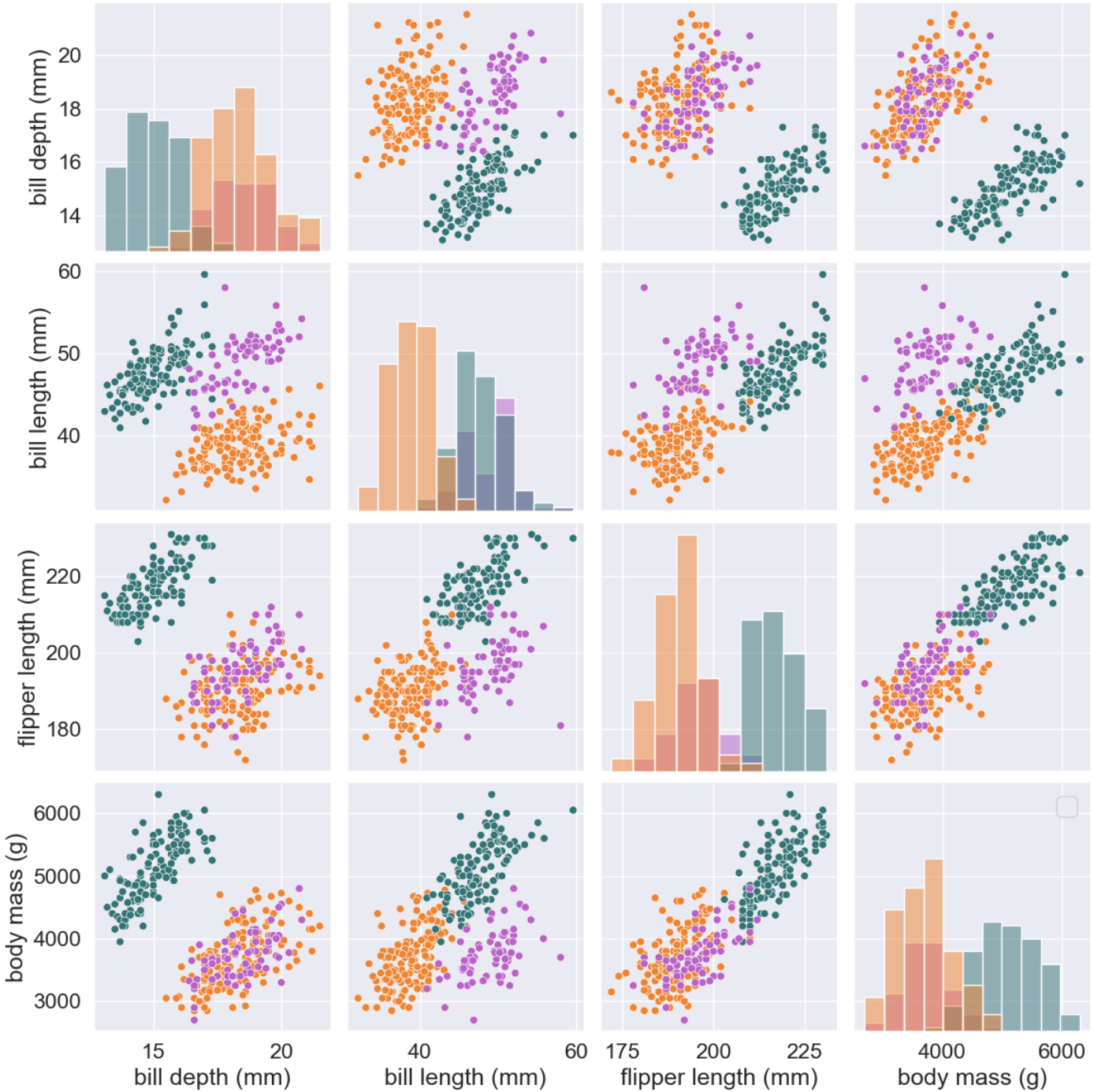
🧠 Short answer:

> I do **not** possess human-semantic concepts. What I have are **very high-dimensional, complex statistical representations** that *align* with human concepts—often very closely, but only from the outside.

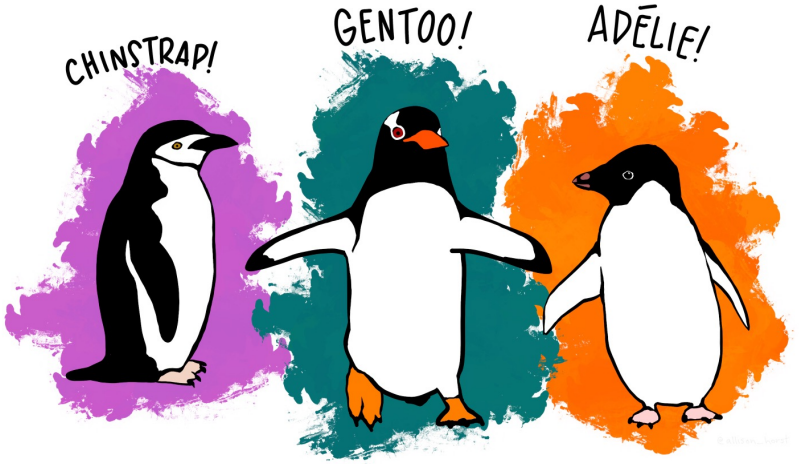
Latent Concept

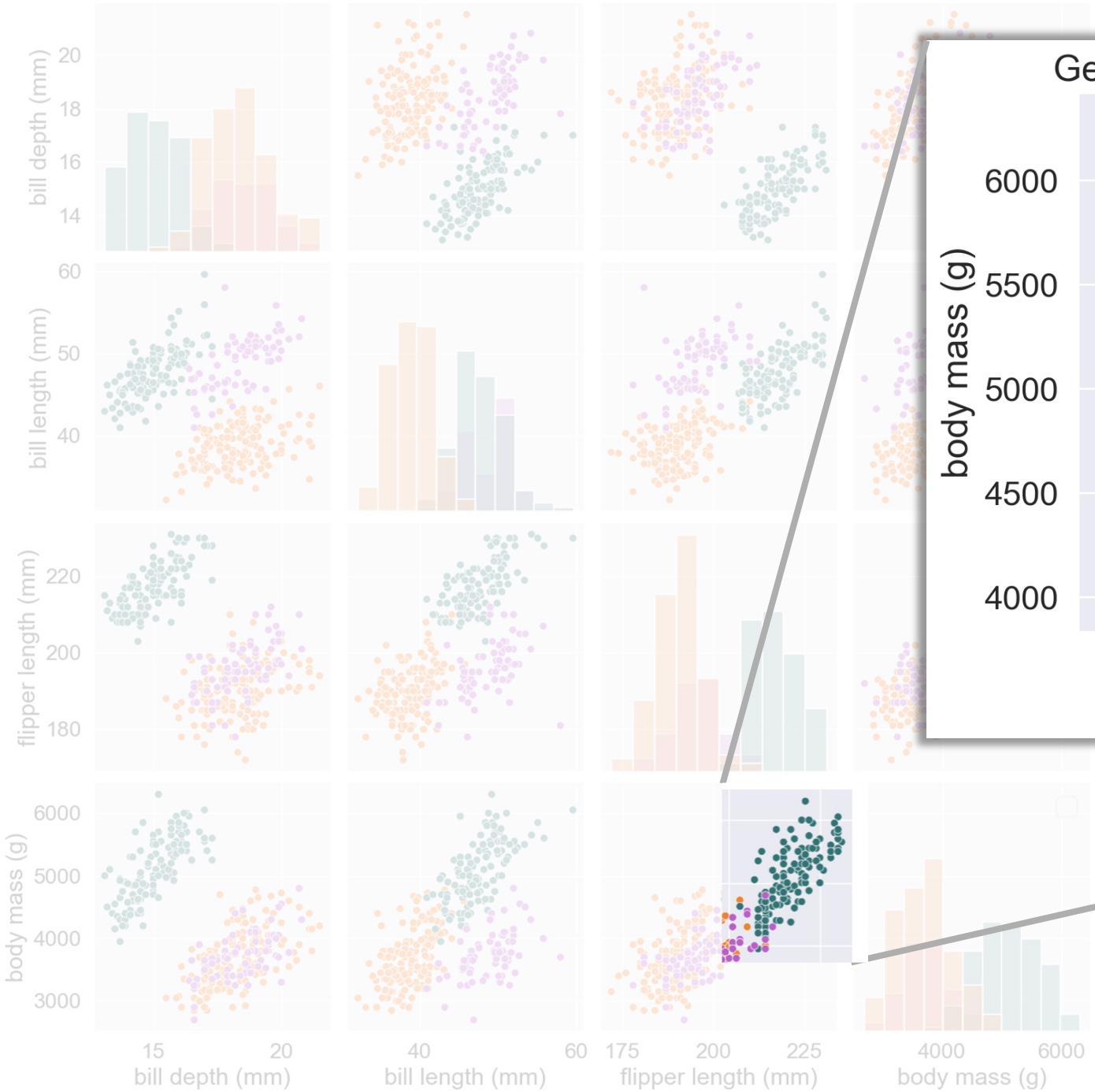
Example dataset

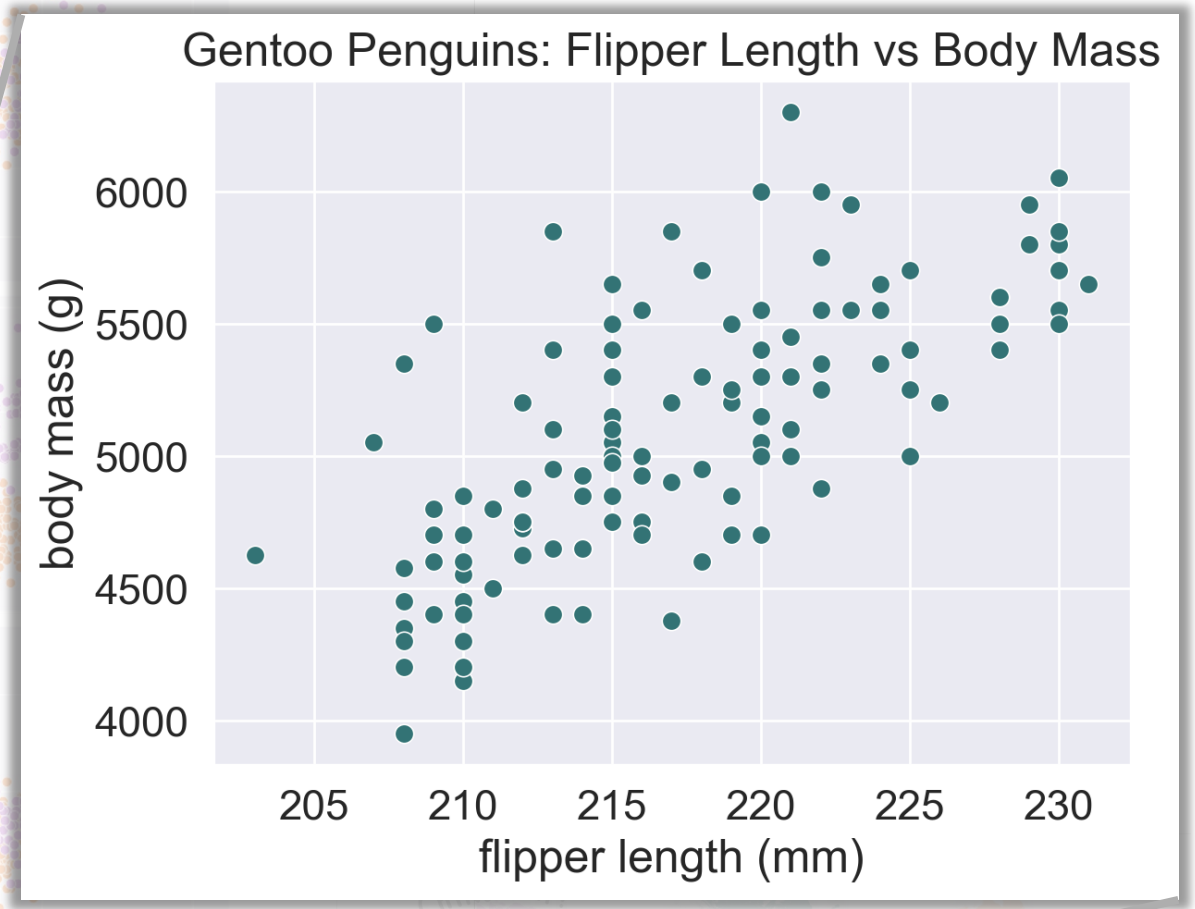




species
- Adelie
- Gentoo
- Chinstrap



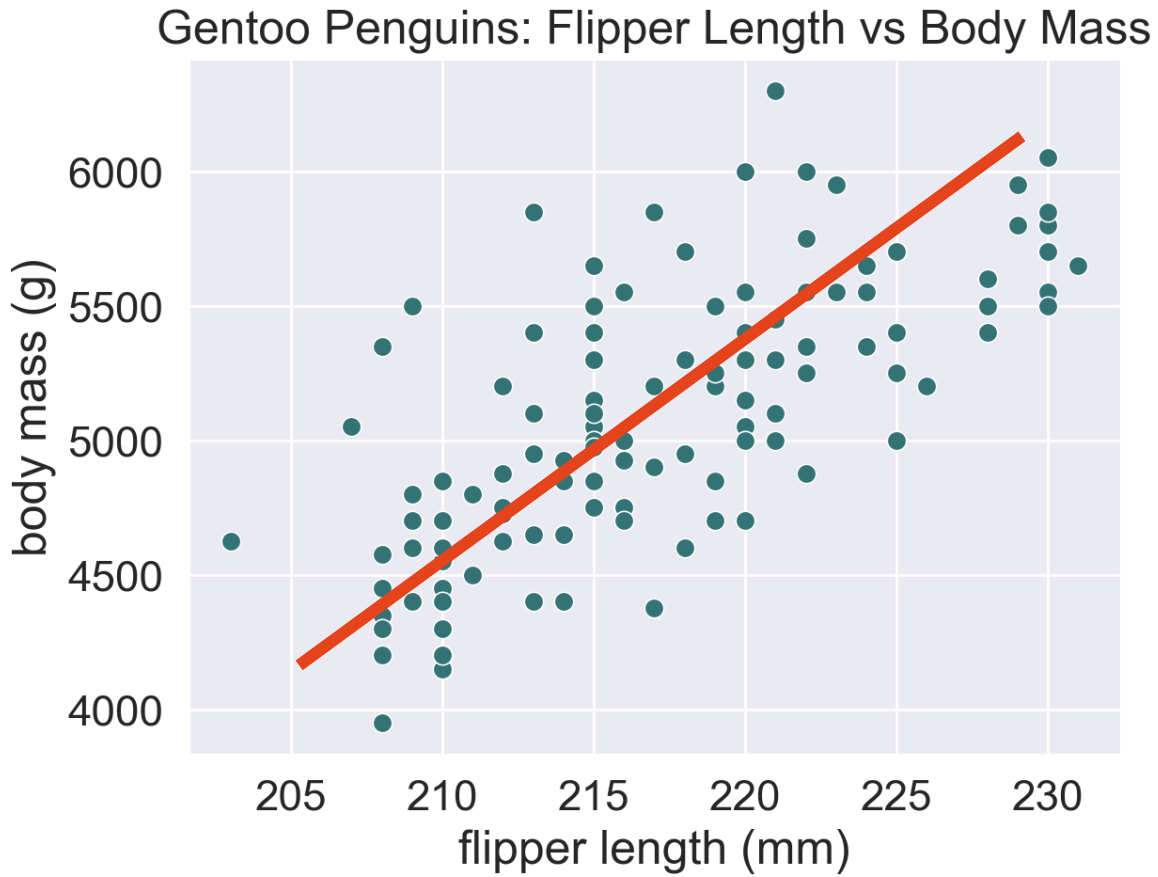

bill depth (mm)
bill length (mm)
flipper length (mm)
body mass (g)
15
20
40
60
175
200
225
4000
6000


Gentoo Penguins: Flipper Length vs Body Mass
body mass (g)
6000
5500
5000
4500
4000
205
210
215
220
225
230
flipper length (mm)

Regression Predicting numerical data

How heavy is a penguin with this length of flipper?

The labels and outputs are continuous in R^D



Linear Regression

A simple form of regression

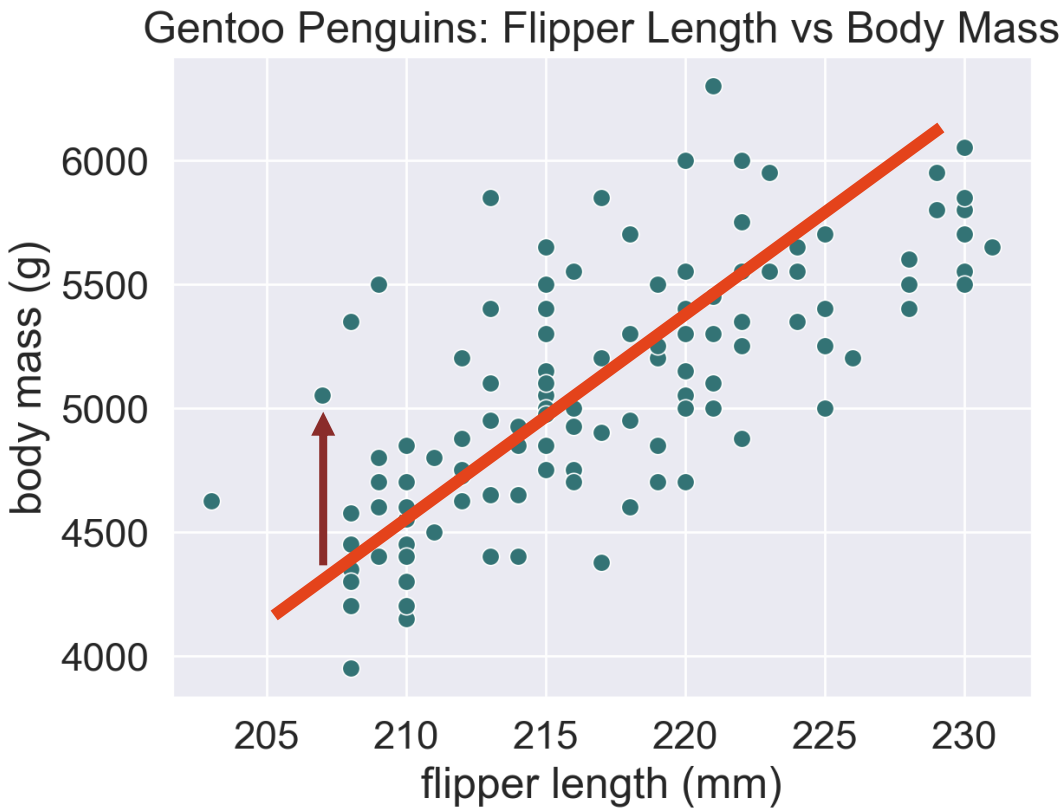
Vector of target data

Vector of input data

Vector of errors

$$\vec{\mathbf{y}} = \omega\vec{\mathbf{x}} + b + \vec{\boldsymbol{\epsilon}}$$

Parameters to find (learn)



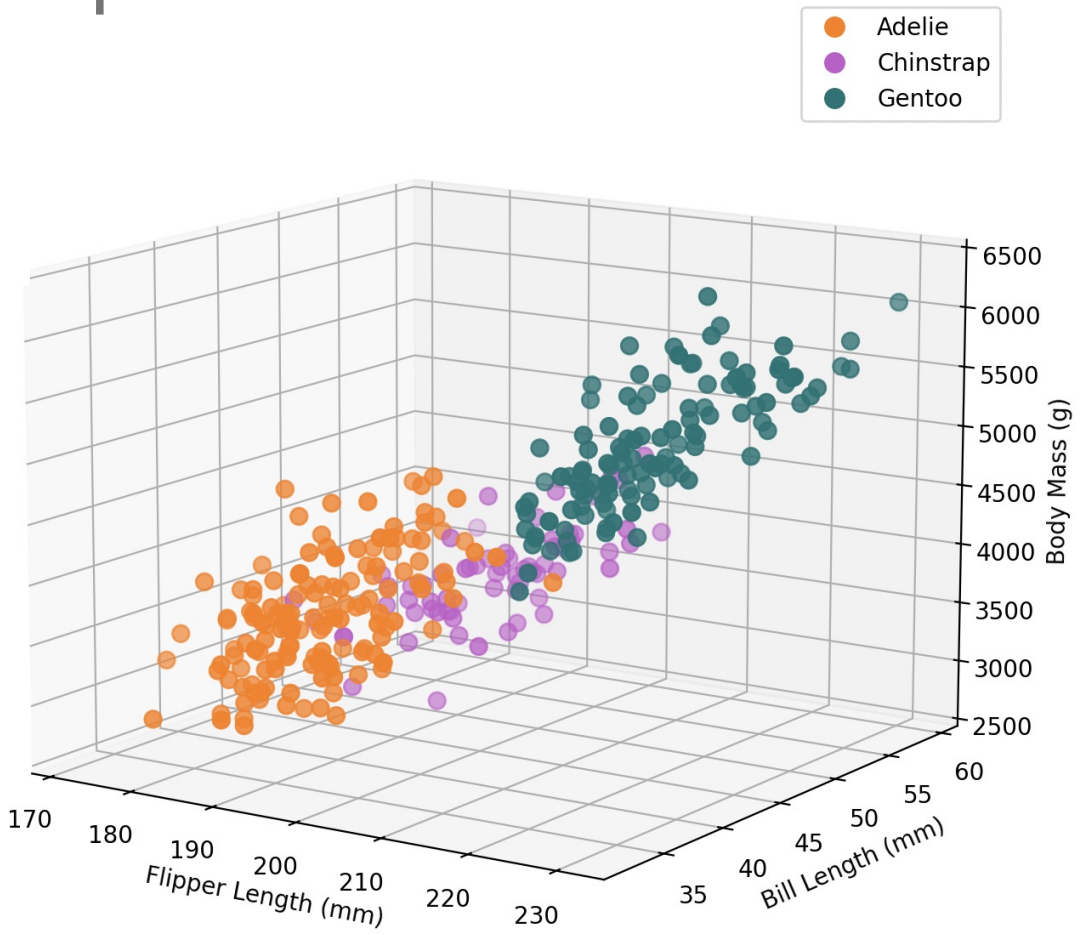
Multiple Linear Regression

Multiple inputs regress a single output

$$\vec{\mathbf{y}} = \omega_1 \vec{\mathbf{x}}_1 + \omega_2 \vec{\mathbf{x}}_2 + b + \vec{\boldsymbol{\epsilon}}$$

For N inputs:

$$\vec{\mathbf{y}} = \sum_{i=1}^{N} \omega_i \vec{\mathbf{x}}_i + b + \vec{\boldsymbol{\epsilon}}$$



Linear Regression Analytic Solution

$$\hat{\mathbf{y}} = \sum_{i=1}^{N} \omega_i \mathbf{x}_i + b$$

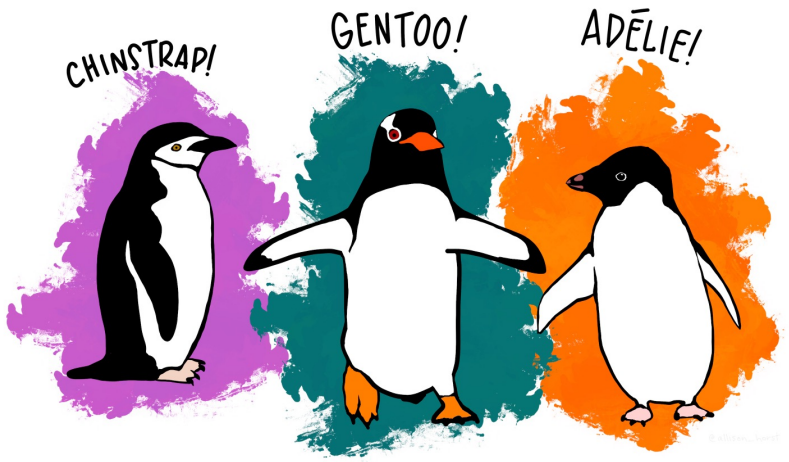
Optional slide

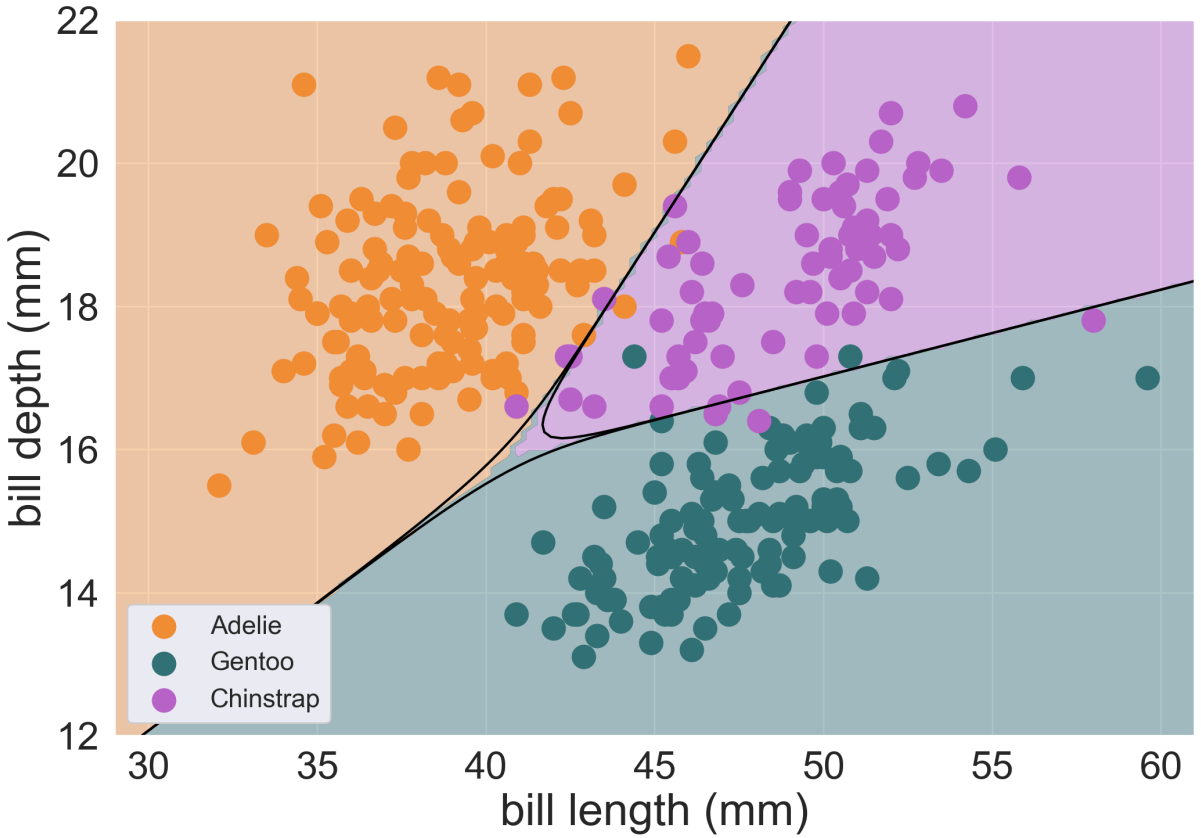
Ethan write about Ordinary Least Squares in his spare time

Classification

Predicting categorical data

Given these features, what species of penguin is this?



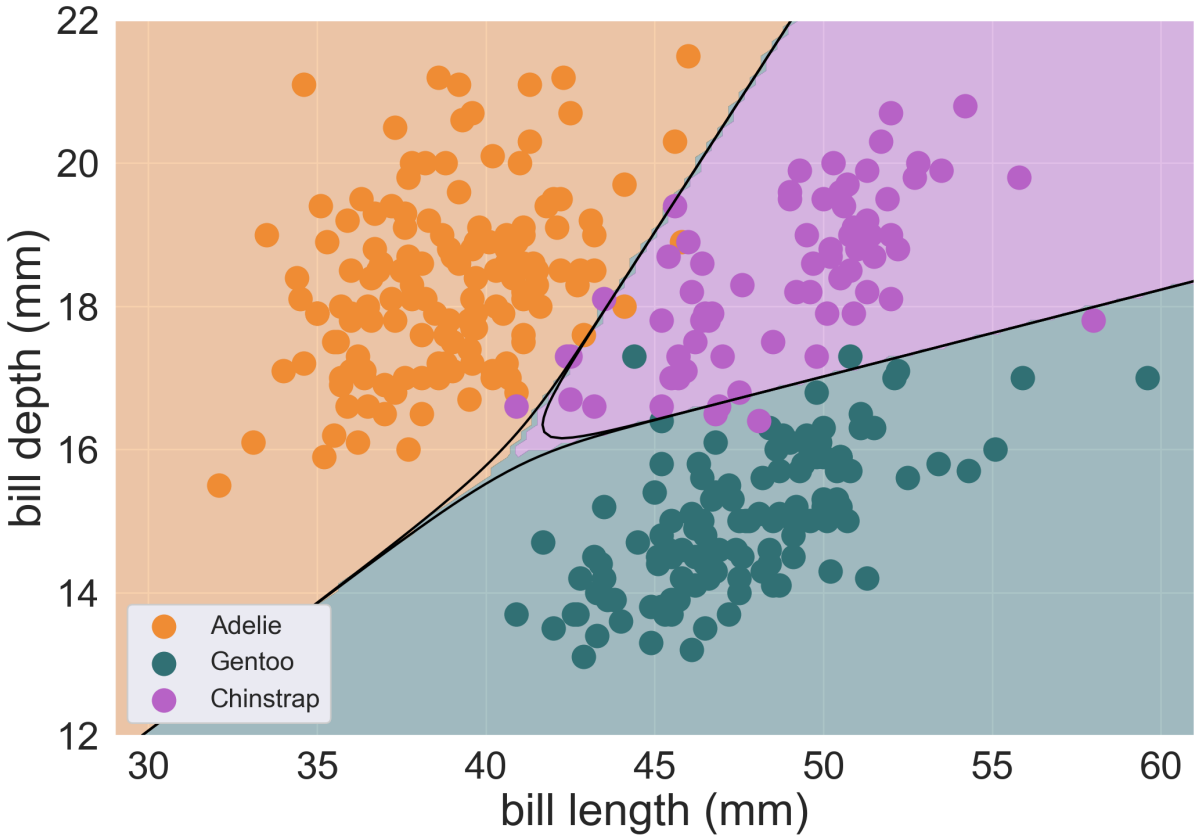


Classification

Predicting categorical data

Given these features, what species of penguin is this?

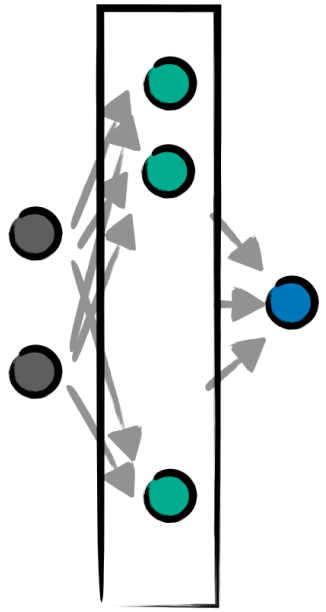
- The outputs and labels categorical
- Categorial data represented numerically e.g integer, one-hot encoding



Machine learning

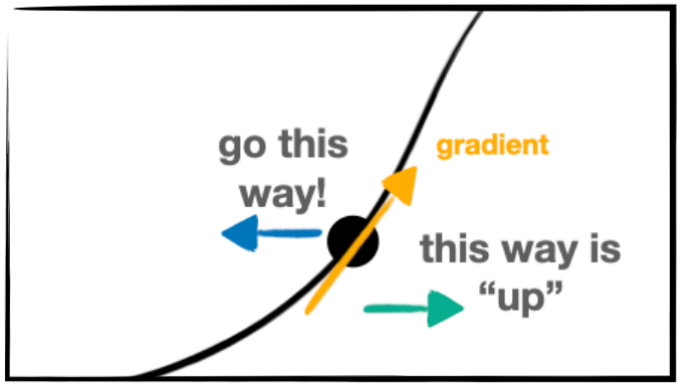
ML Structure

1. Model makes prediction



2. Compare prediction to target

3. Update the model intelligently



Model

Maps inputs to outputs

A specific architecture performs a specific set of mathematical operations

```python
class CNNModel(nn.Module):
    def __init__(self, input_channels=3, num_classes=3):
        super(CNNModel, self).__init__()

        # Convolutional Layers (6 hidden layers)
        self.conv1 = nn.Conv2d(input_channels, 32, kernel_size=3,  padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3,  padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3,  padding=1)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3,  padding=1)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=3,  padding=1)
        self.conv6 = nn.Conv2d(512, 512, kernel_size=3,  padding=1)

        # Max Pooling Layers
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)

        # Fully Connected Layer (for final classification)
        self.fc1 = nn.Linear(512 * 7 * 7, 4096)  #
        self.fc2 = nn.Linear(4096, num_classes)
        # Dropout (to prevent overfitting)
        self.dropout = nn.Dropout(0.5)
```

$$\begin{pmatrix} 0.14 \\ 0.81 \\ 0.05 \end{pmatrix}$$

Linear Models

Prediction depends linearly on the parameters

$$\hat{\mathbf{y}} = \sum_{i=1}^{N} \omega_i \mathbf{x}_i + b$$

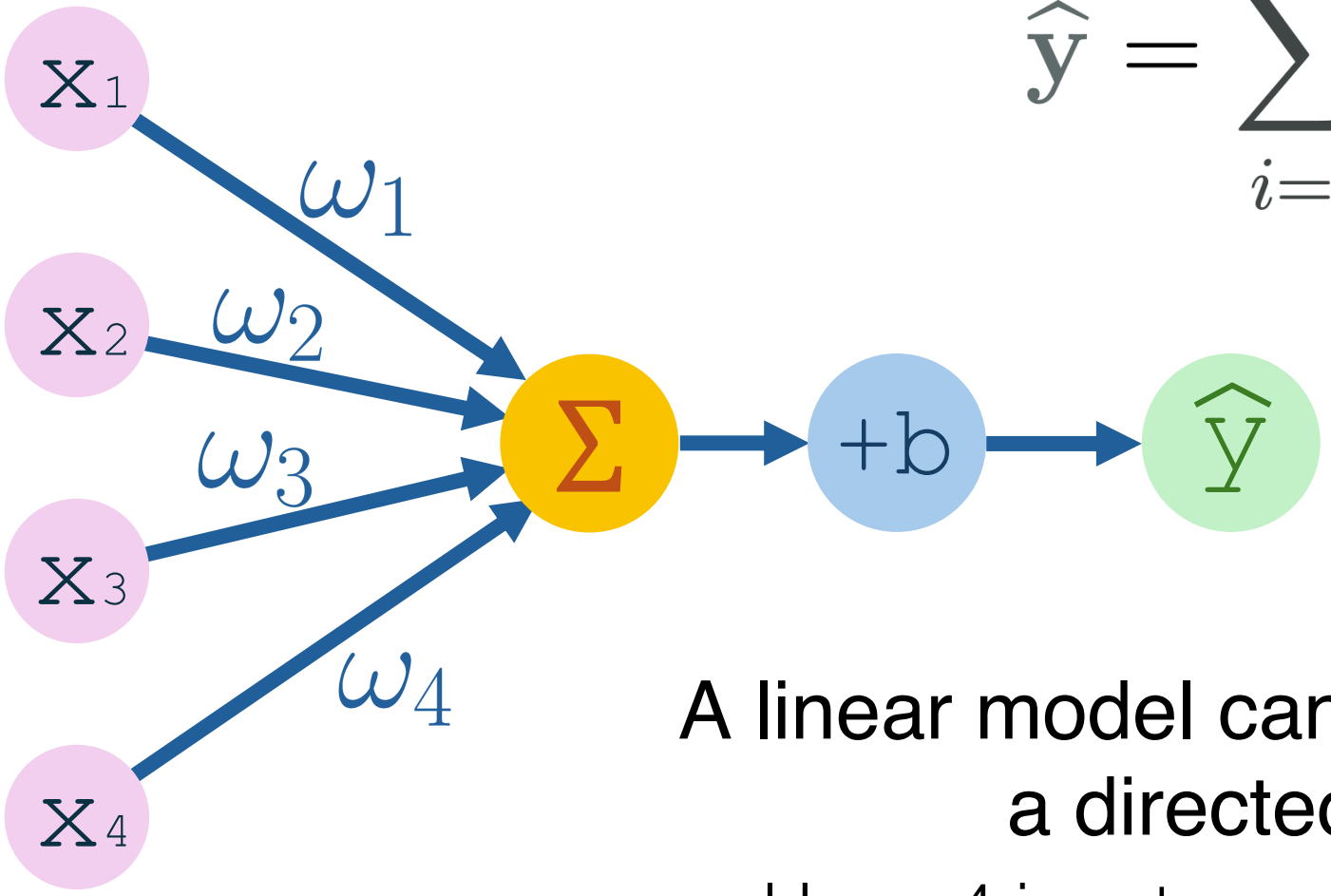
In ML contexts, the parameters are learnable: refined over learning epochs

Simple to use and train!

Scikit-Learn has many linear models: see here

Linear Models

$$\hat{\mathbf{y}} = \sum_{i=1}^{N} \omega_i \mathbf{x}_i + b$$

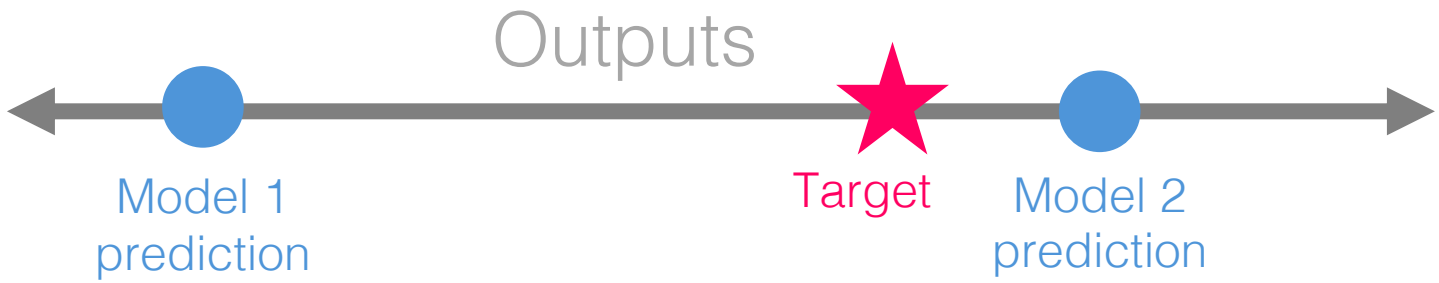


A linear model can be imagined as a directed graph

Here, 4 inputs are mapped to one output

Objective Function (Loss)

How close is my model's prediction to the target?



Function quantifies "closeness" of prediction and target

$$\mathrm{MSE}(\mathbf{y}, \widehat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \widehat{\mathbf{y}}_i)^2$$

Average squared difference between target and predicted data points

Optimisers

Improve the model based on its current and past performance.

Optimisers search the parameter space to find an optimal set of parameters

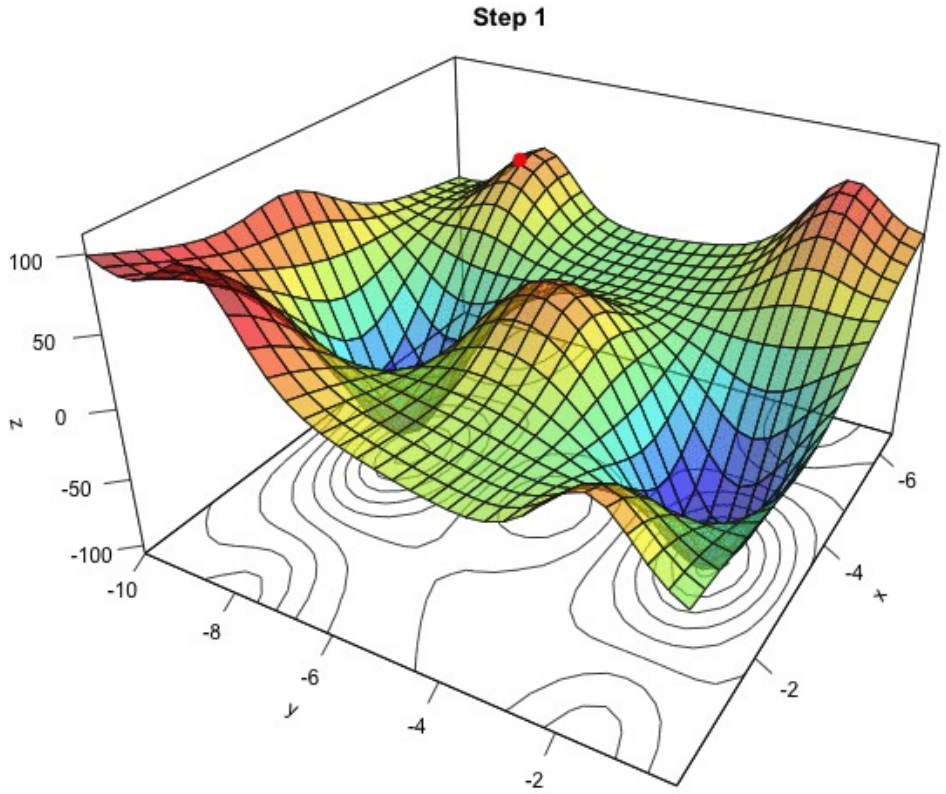
Optimising an objective function is a very general problem in applied mathematics. Most popular method is gradient descent

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Gradient Descent

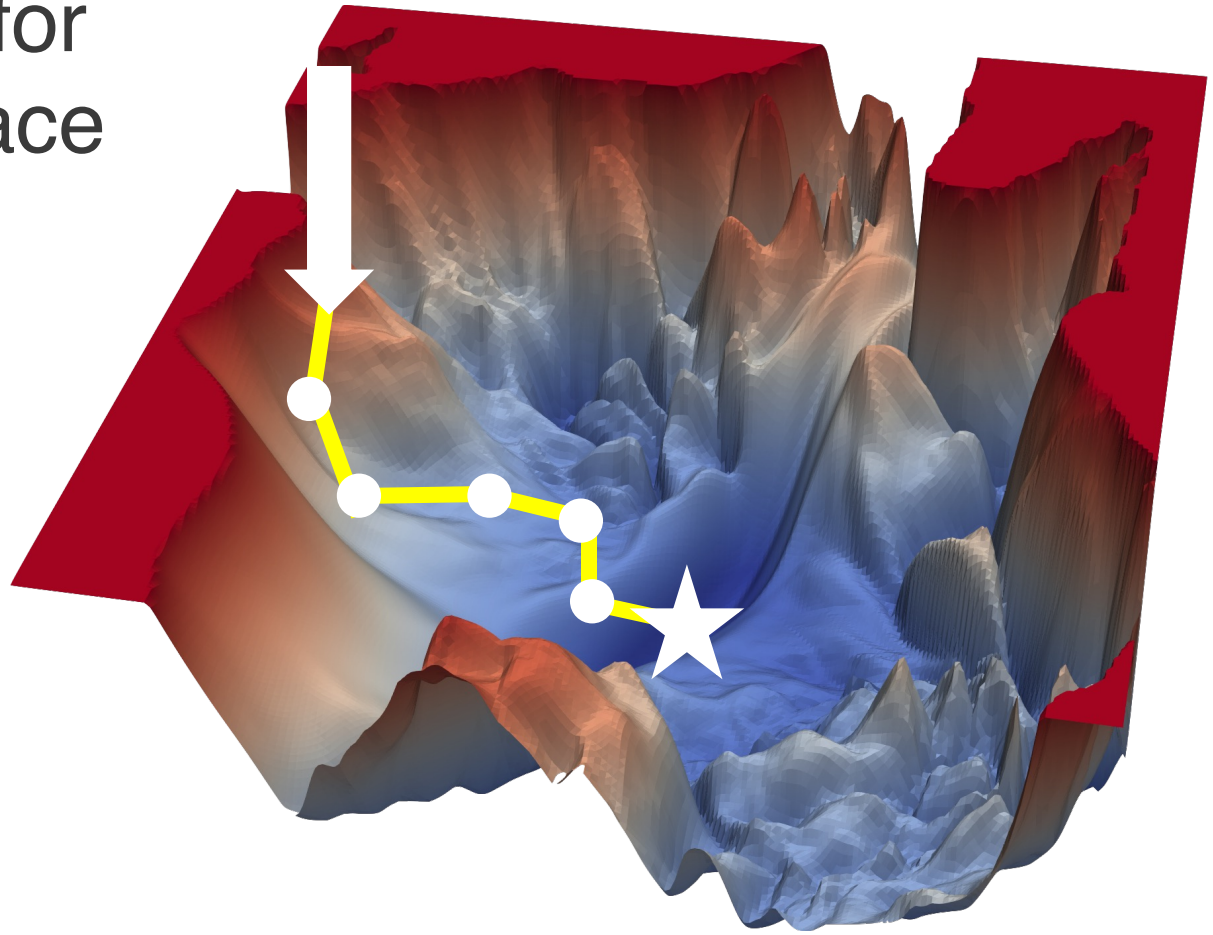
Use local gradients to search for minimum in the parameter space

1. Compute the gradient where you are
2. Move in the direction of steepest descent
3. Repeat until happy



Gradient Descent

Use local gradients to search for minimum in the parameter space



Challenges:

- 9️⃣ Computing gradients in high-dimensional spaces
- 🚶 Knowing how far to step each iteration
- ⛰️ Knowing you're at a global minima

Linear Regression Example

Definitions

```
model = nn.Linear(1, 1)
loss_function = nn.MSELoss()
optimizer = optim.Rprop(model.parameters())
```

Training loop

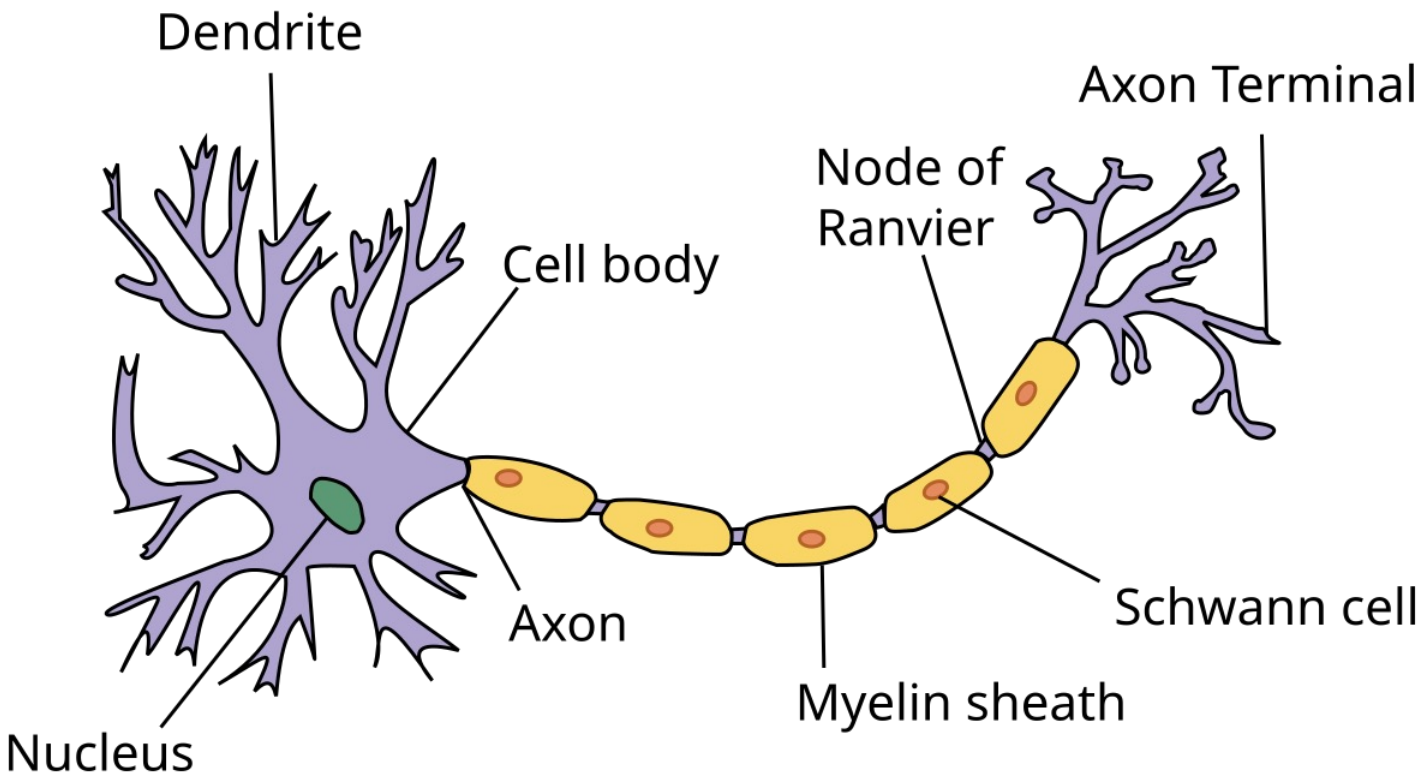
```
for epoch in range(1000):
    optimizer.zero_grad()
    predictions = model(input_data)
    loss = loss_function(predictions, target)
    loss.backward()
    optimizer.step()

y_out = model(input_data)
```

Classification and non-linearity

Neuron

➕ Collects inputs

💥 Activates if a threshold is exceeded

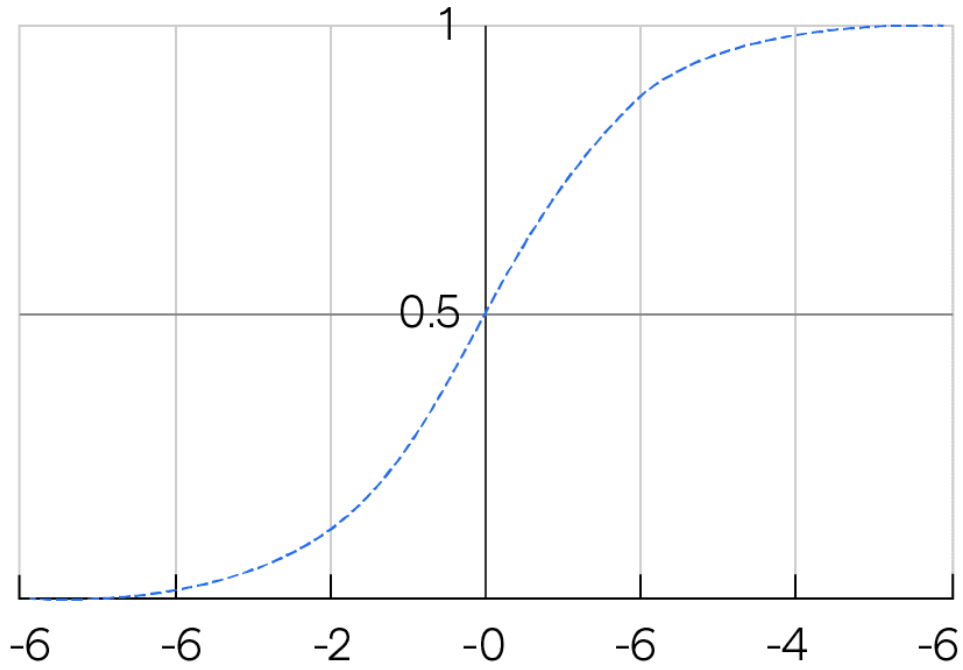


The output of a neuron is manifestly non-linear

Sigmoid Function

Maps any x value close to 0 or close to 1

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

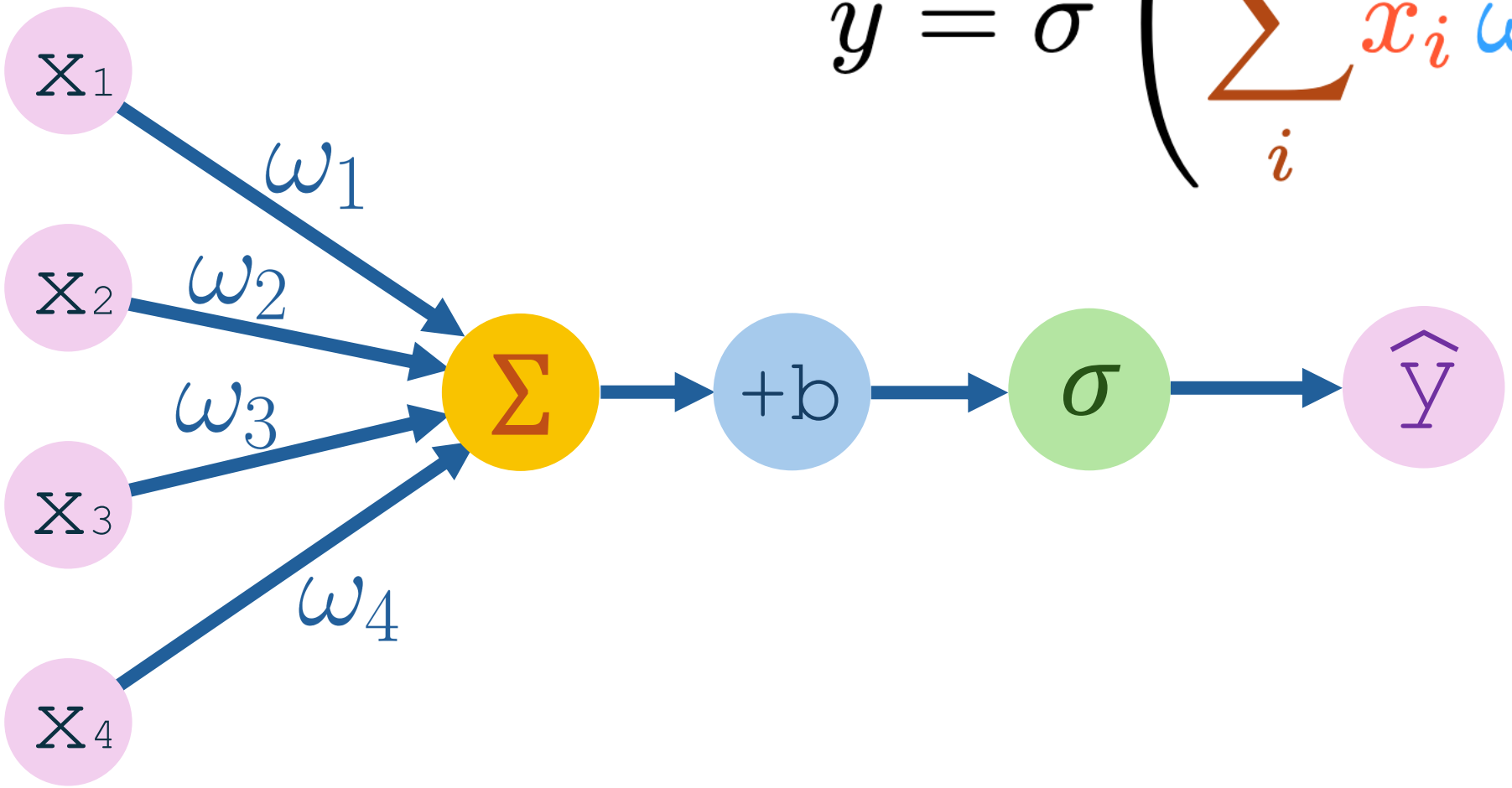


- Bounded outputs interpreted as probabilities
- Roughly separates by bunching values near 0 or 1

Linear + Sigmoid = Classifier Model

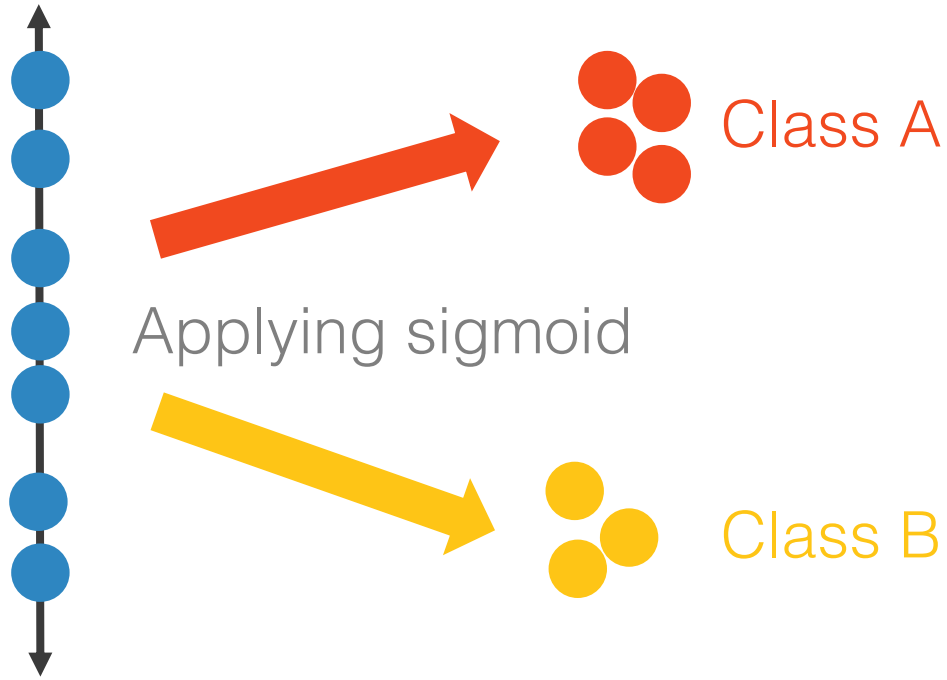
Artificial Neuron

$$y = \sigma\left(\sum_i x_i\, \omega_i + b\right)$$



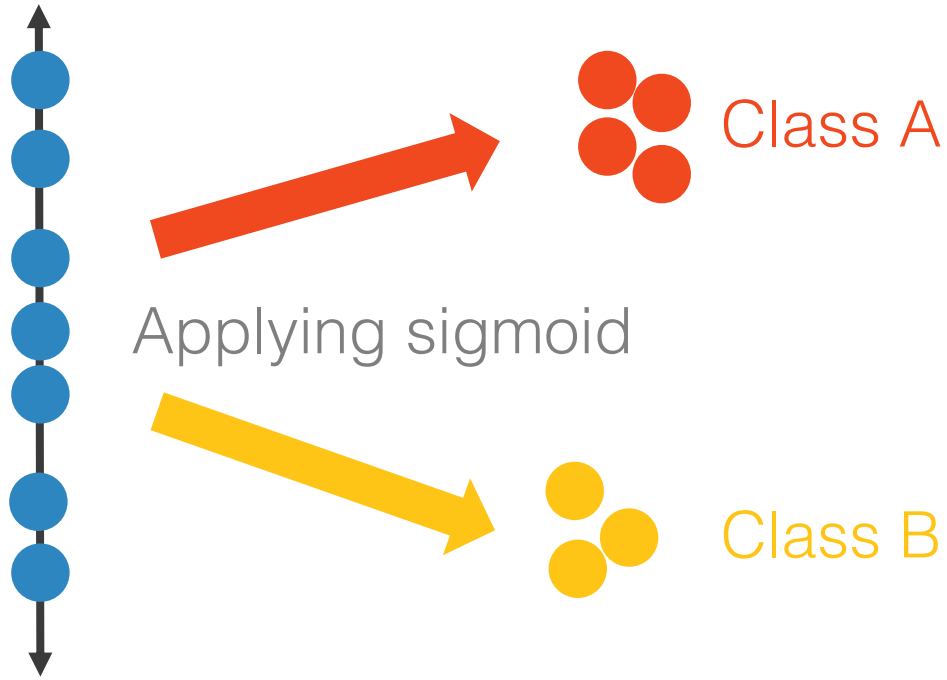
Classification

Apply output activation function to predict category from continuous input



Classification

Apply output activation function to predict category from continuous input



Logistic regression

Apply sigmoid to linear model

Multi-class prediction

Use softmax instead of sigmoid

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Alternative methods:

- Decision trees
- Random forests
- Support vector machines

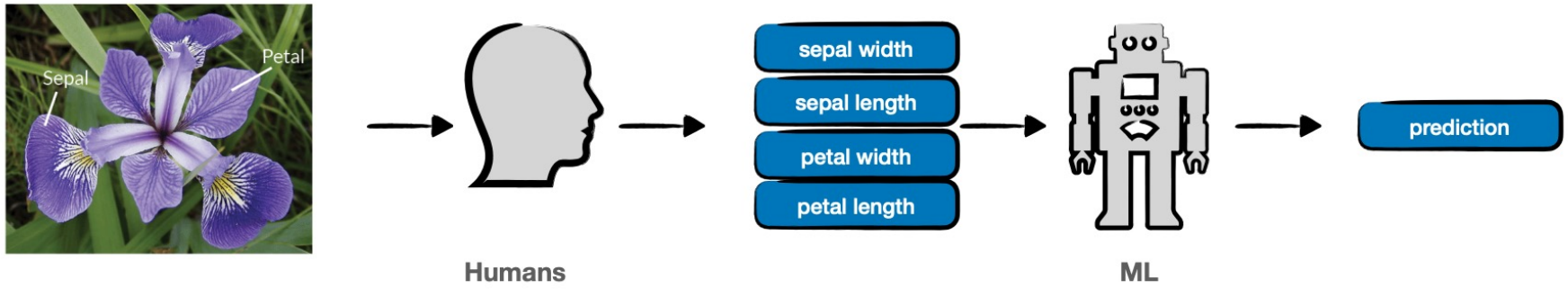
Summary

- We have used shallow techniques, amenable because our datasets were small and low-dimensional
- We defined feed-forward models to make predictions
- We used appropriate loss functions to evaluate the model's performance
- We used PyTorch's automatic tools for computing the gradient and updating the learning parameters
- A linear layer plus a sigmoid function gave a non-linear response

Deep Learning

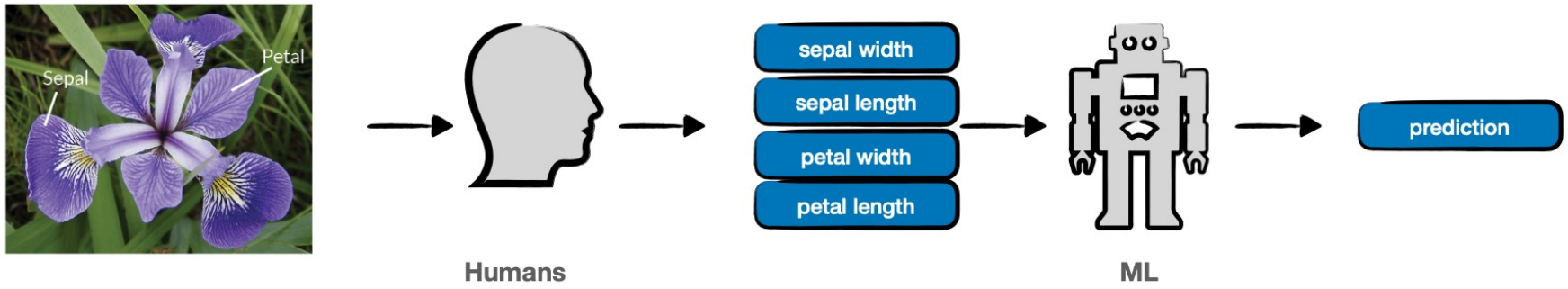
Shallow Networks

Classic ML relied on human-engineered features



Shallow Networks

Classic ML relied on human-engineered features



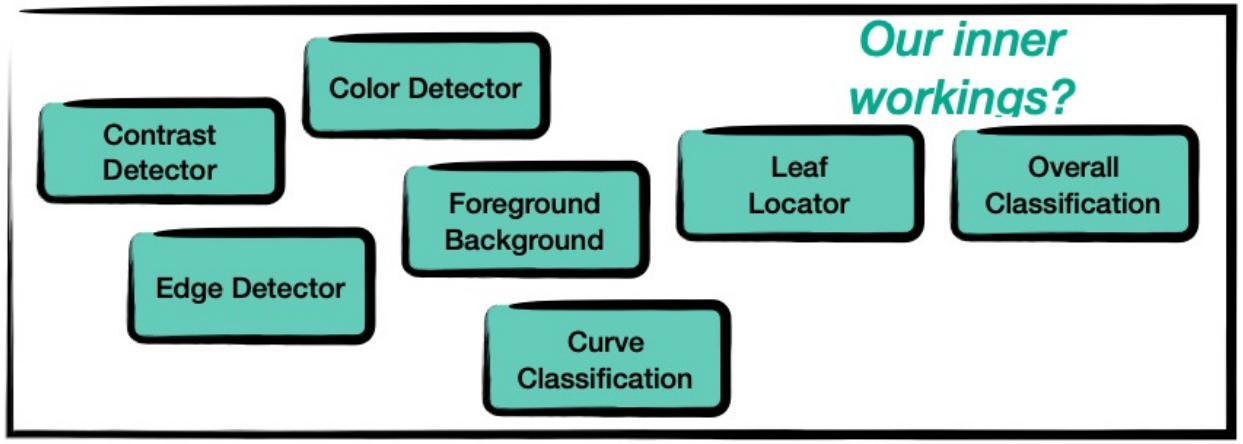
Limitations:

♾️ Single-layer networks require a lot of parameters

🐢 Inefficient to search universal space for optimal mapping

Deep Learning

Effective ML reasoning goes through layers of abstraction?



Setosa

Deep Learning

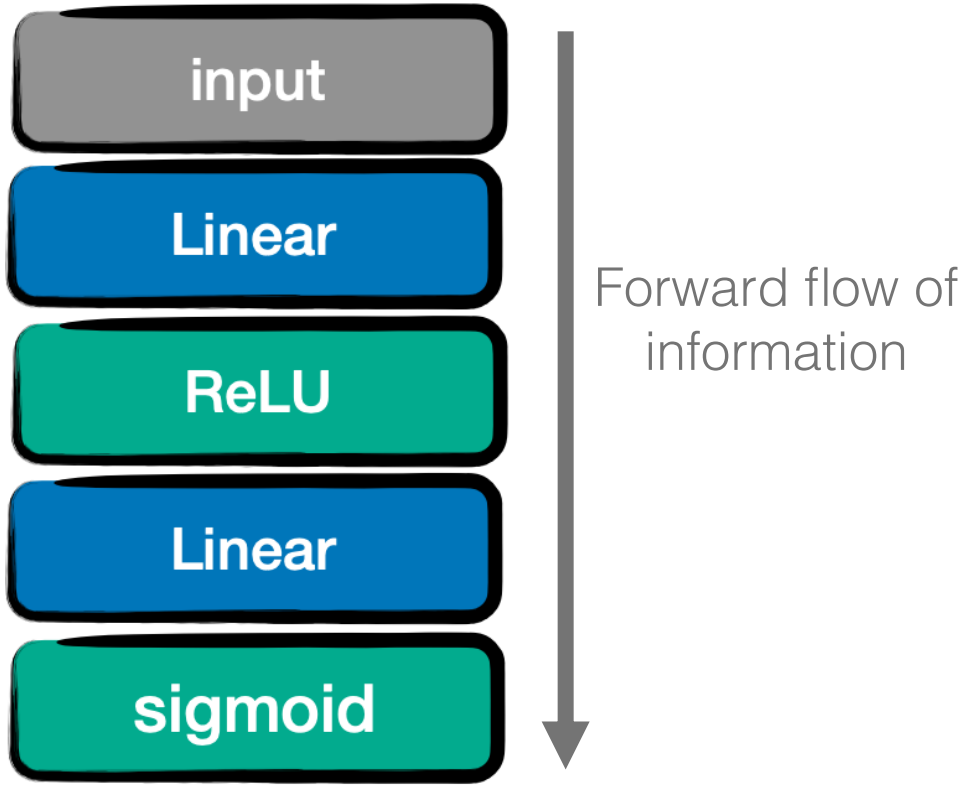
Effective ML reasoning goes through layers of abstraction?

Deep networks add multiple layers to emulate this

Why does deep and cheap learning work so well?*

Henry W. Lin, Max Tegmark, and David Rolnick
Dept. of Physics, Harvard University, Cambridge, MA 02138
Dept. of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139 and
Dept. of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139

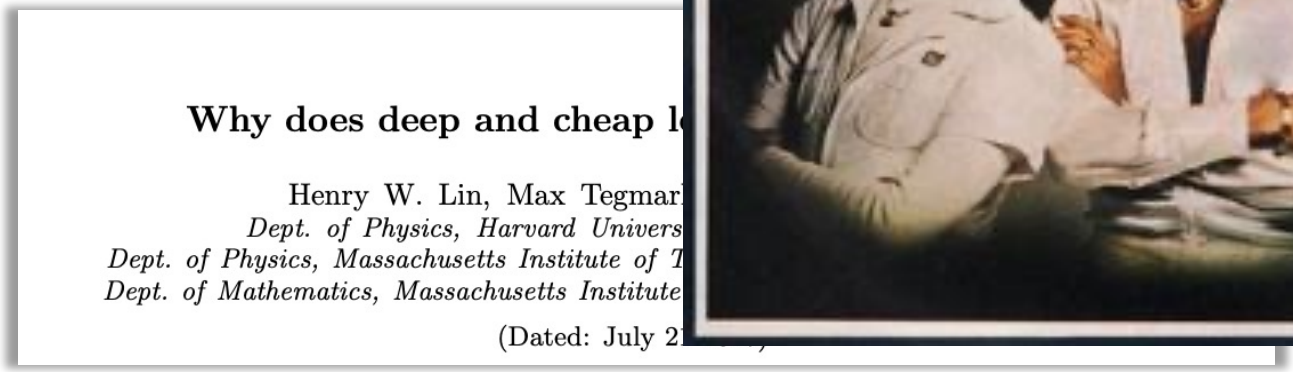
(Dated: July 21 2017)



Deep Learning

Effective ML rea of abstraction?

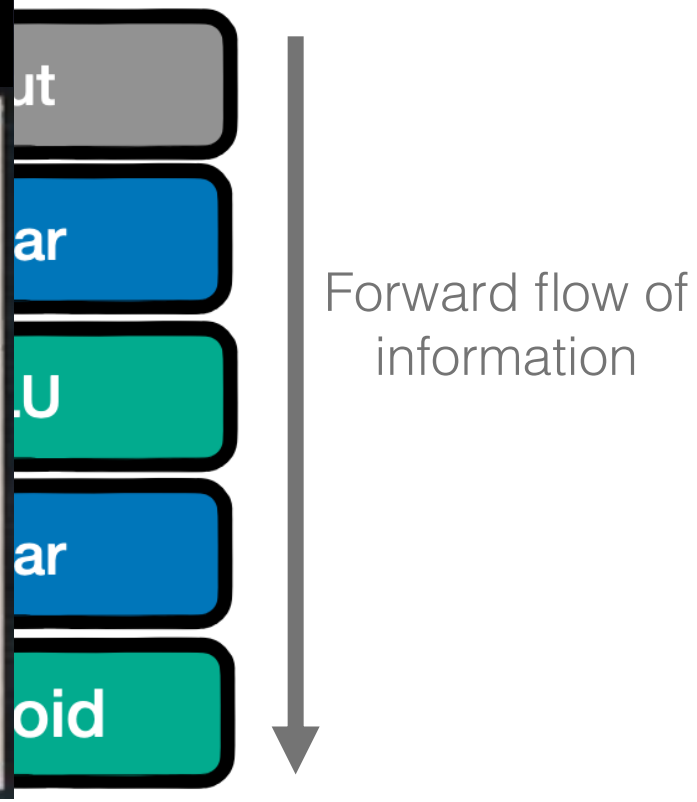
Deep netwo
multiple lay
emulate



Why does deep and cheap l

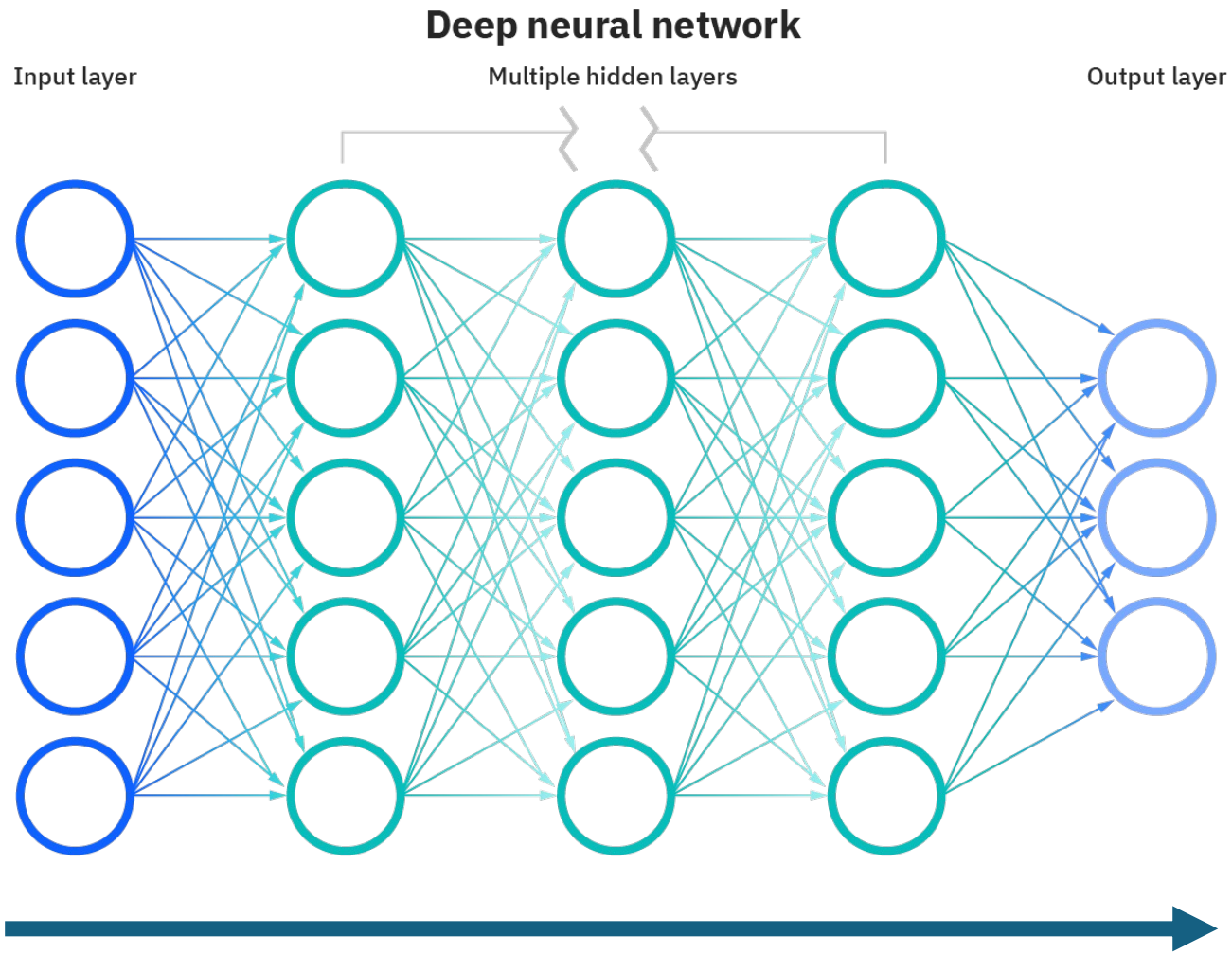
Henry W. Lin, Max Tegmar
Dept. of Physics, Harvard Univers
Dept. of Physics, Massachusetts Institute of T
Dept. of Mathematics, Massachusetts Institute

(Dated: July 2



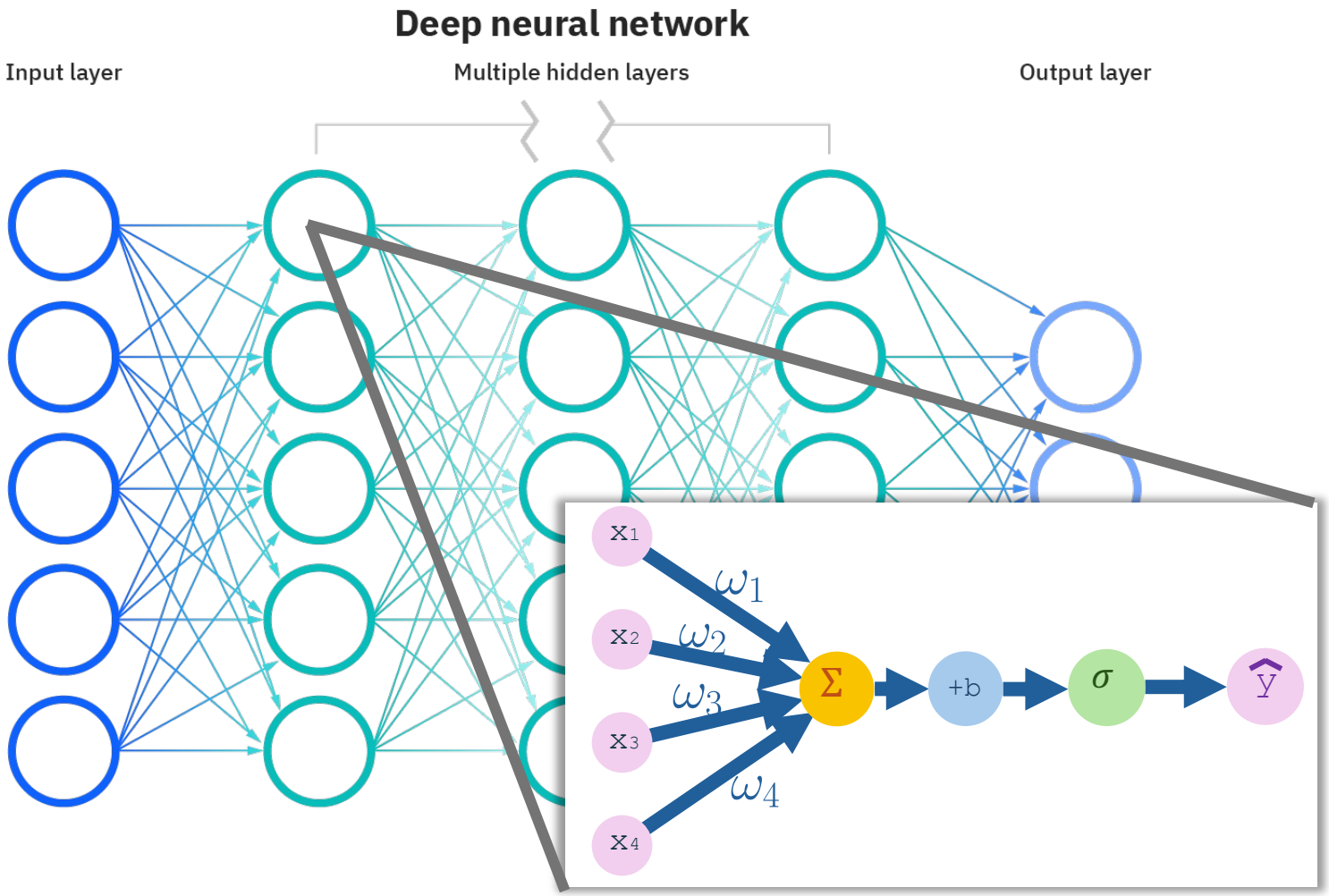
Forward flow of information

Multi-Layer Perceptrons



Many artificial neurons linked together

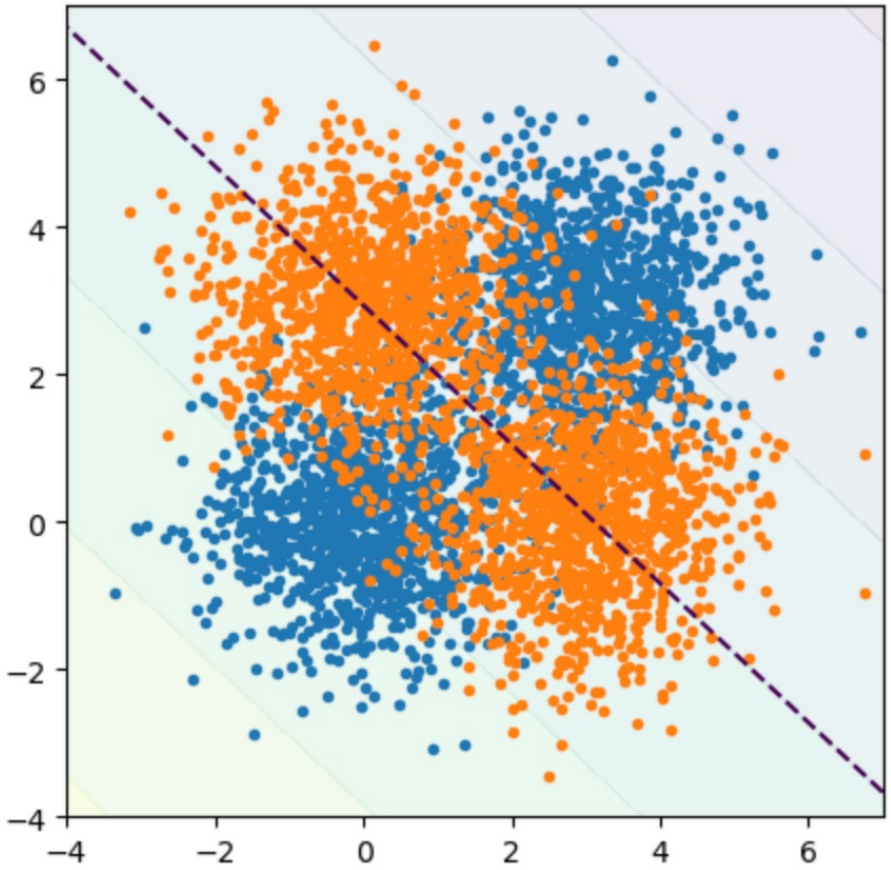
Multi-Layer Perceptrons



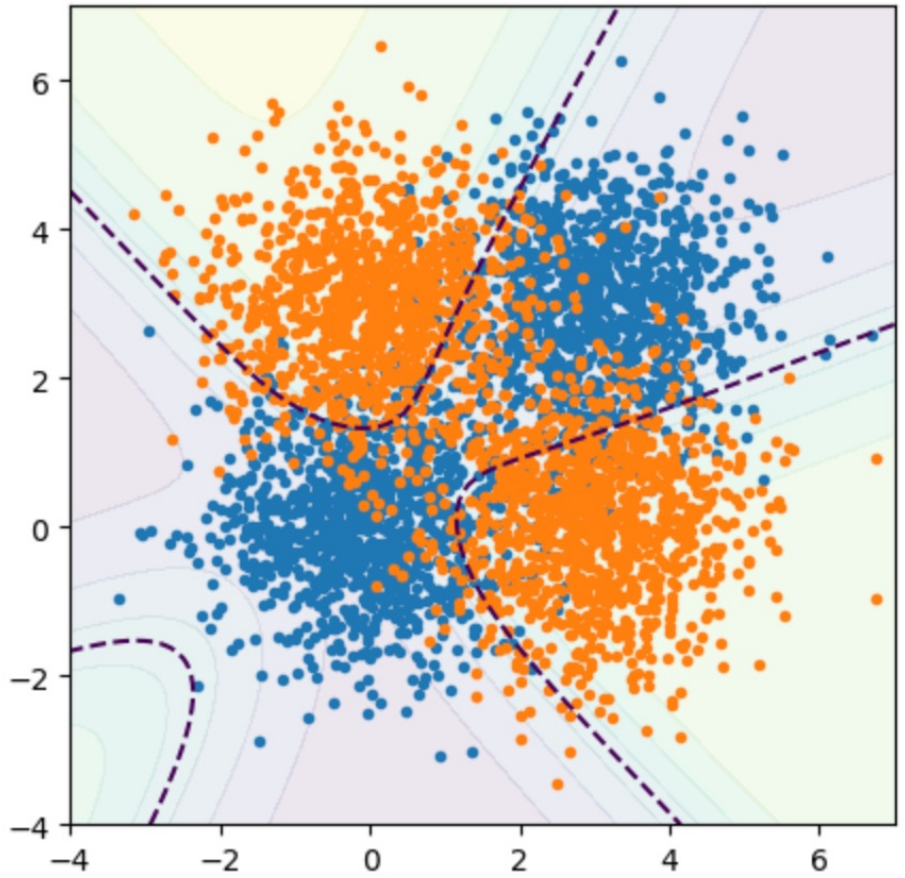
Many artificial neurons linked together

Successive affine transformations and activation functions

Proof



Logistic Regression



Deep Neural Network

Adaptive Basis Functions

A useful way of understanding the power of activation functions…

Series expansions are used everywhere in physics as approximators:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Taylor

$$f(x) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]$$

Fourier

These define a functional basis – but the bases function themselves are of fixed form!

Adaptive Basis Functions

A useful way of understanding the power of activation functions…

A powerful approach is to expand in terms of a function which is adaptive…

$$y = \sum_i^n c_i \psi(x; \alpha_i, \beta_i) = \sum_i^n c_i \frac{1}{1 + \exp\left((x - \alpha_i)/\beta_i\right)}$$

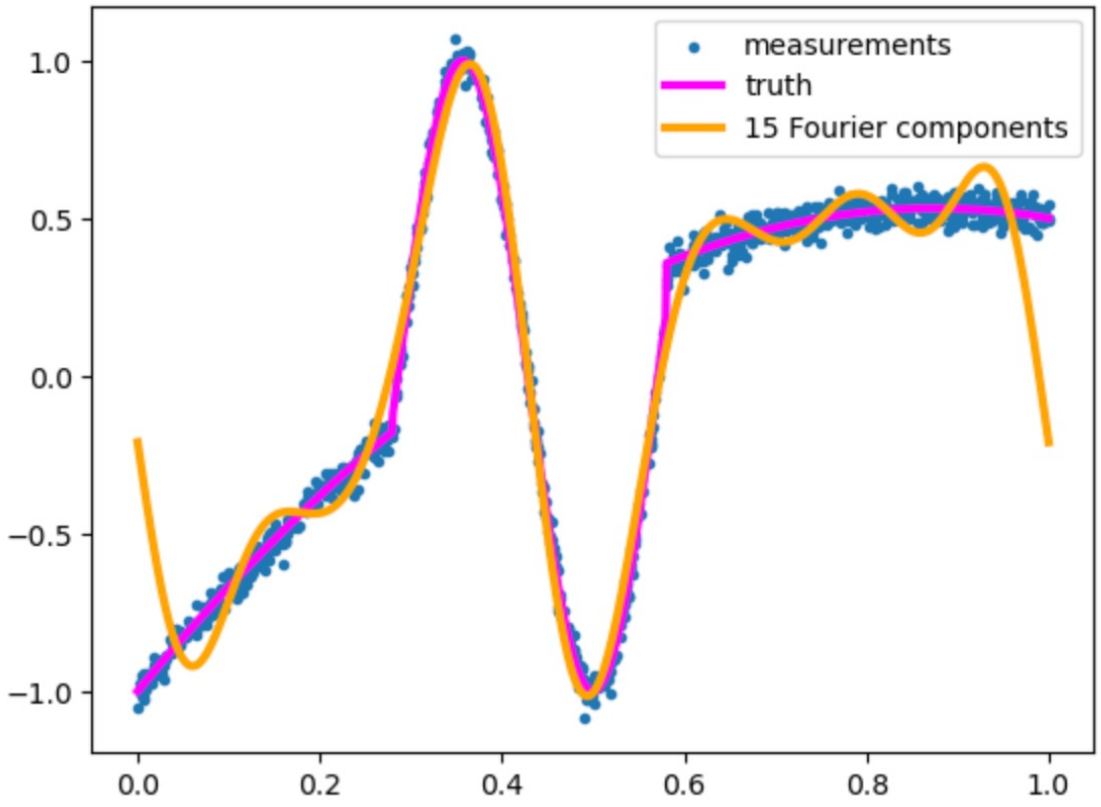
Linear transformation + sigmoid!!

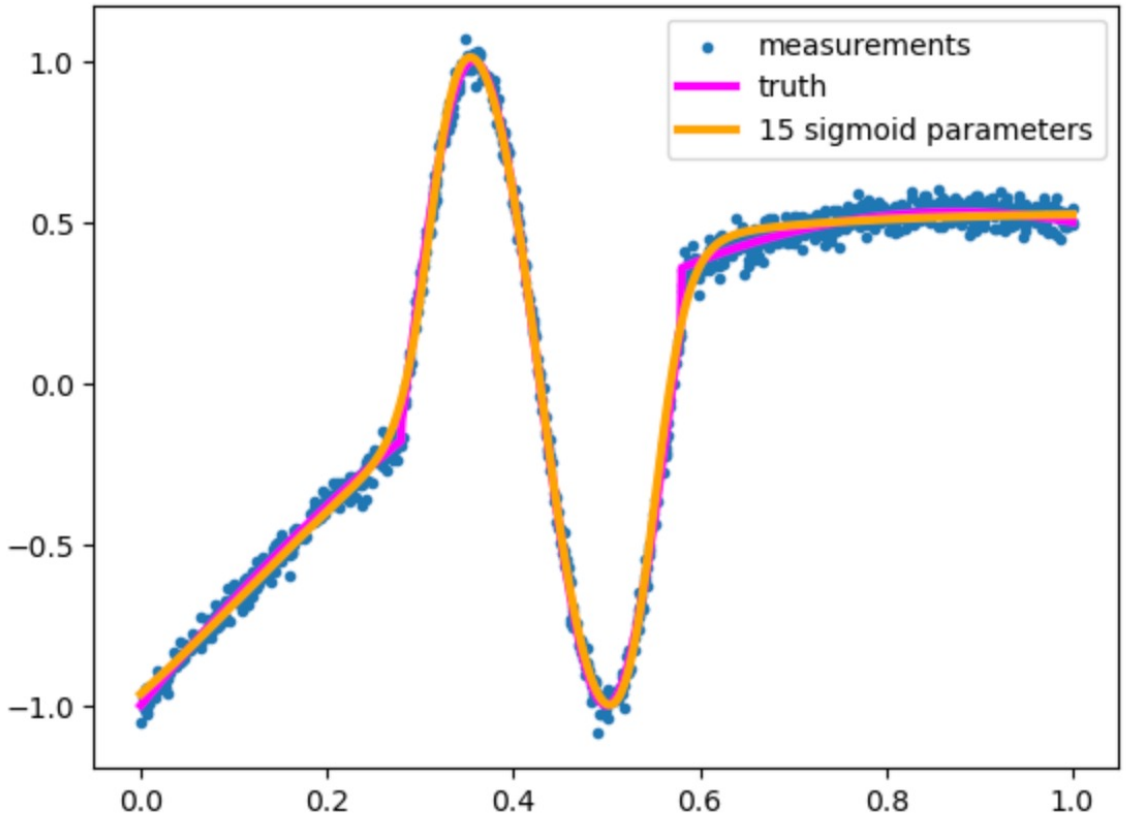
This is just a single layer neural network!

Detailed discussion on “adaptive basis functions” here

Adaptive Basis Functions

A useful way of understanding the power of activation functions…

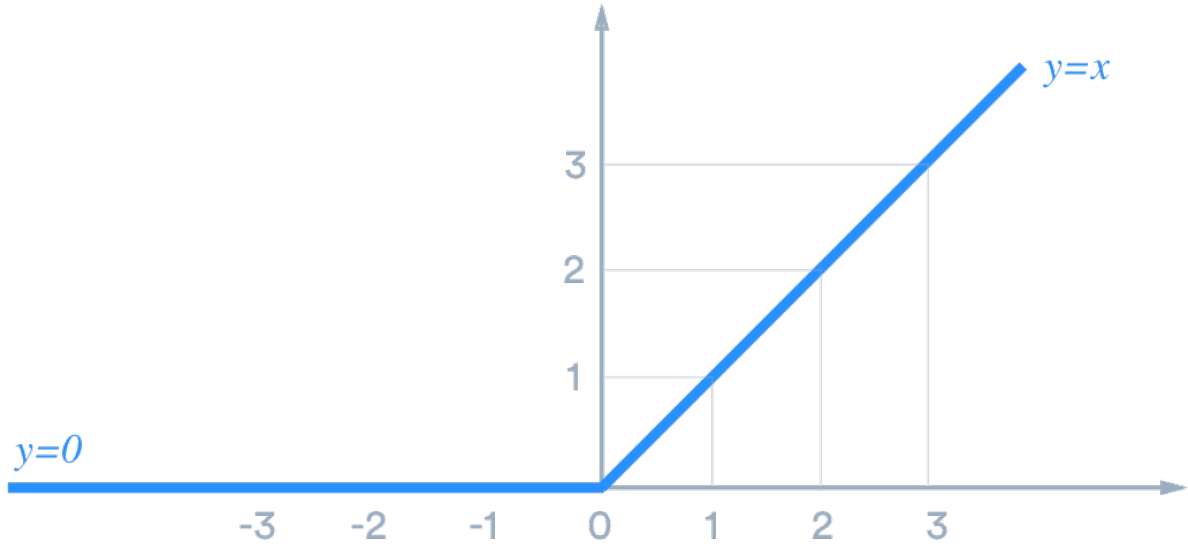




Activation Functions

Key ingredient to achieving non-linearity

Classes of functions which have a “turn on”-like behaviour – is a neuron activated or not?



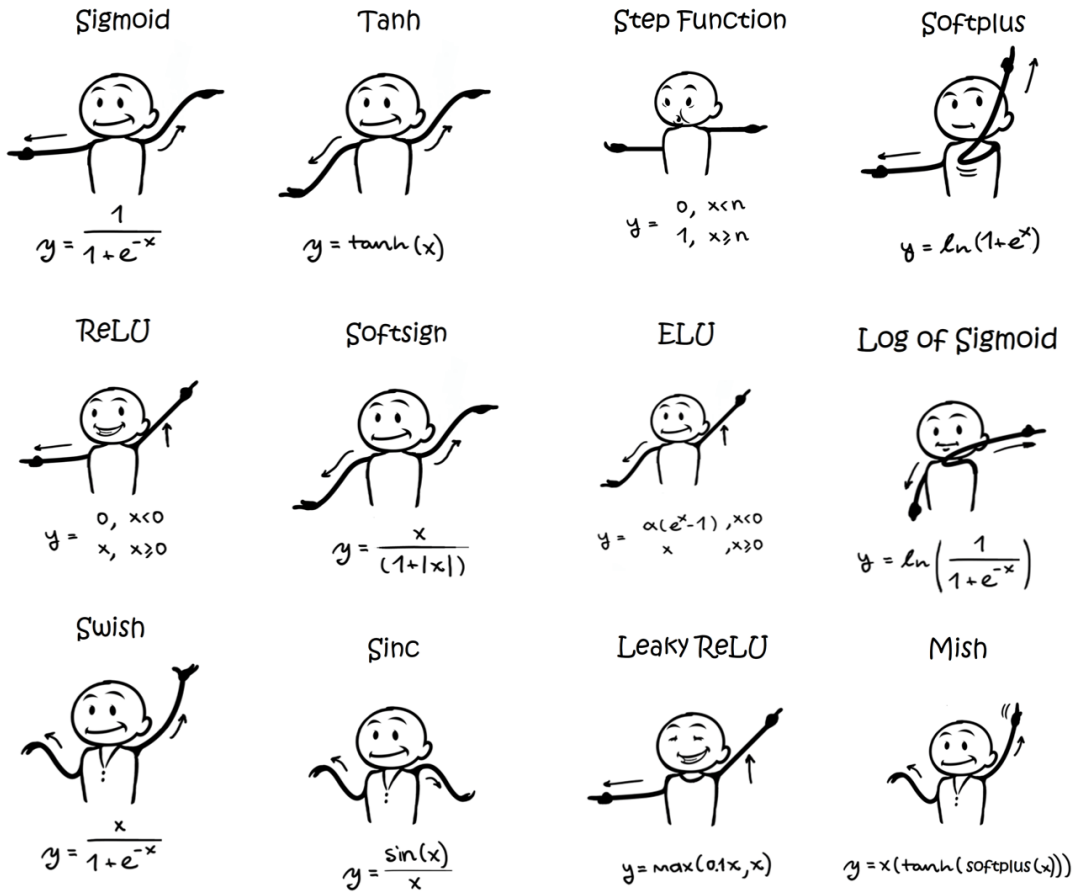
Universal Approximation Theorems

Loosely, neural networks can model any function.

A FFNN with a single layer, non-polynomial activations and a finite number of neurons can approximate a continuous function between Euclidean spaces to arbitrary precision

No guarantee that one can realistically find the optimal parameters: no prescription for finding it…

Activation Functions



Which function to choose?

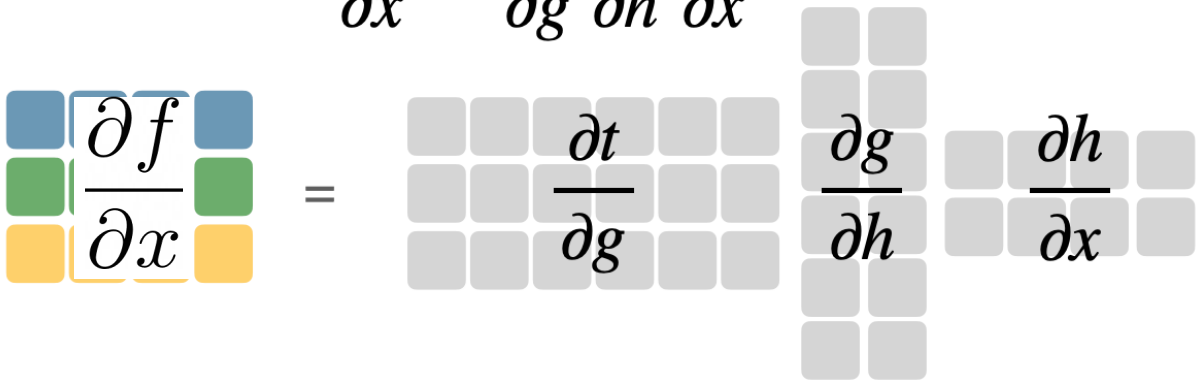
Let's leave that for Maximilian…

Gradients?

Derivates of complex, deeply-nested functions…

Naively using the chain rule scales poorly

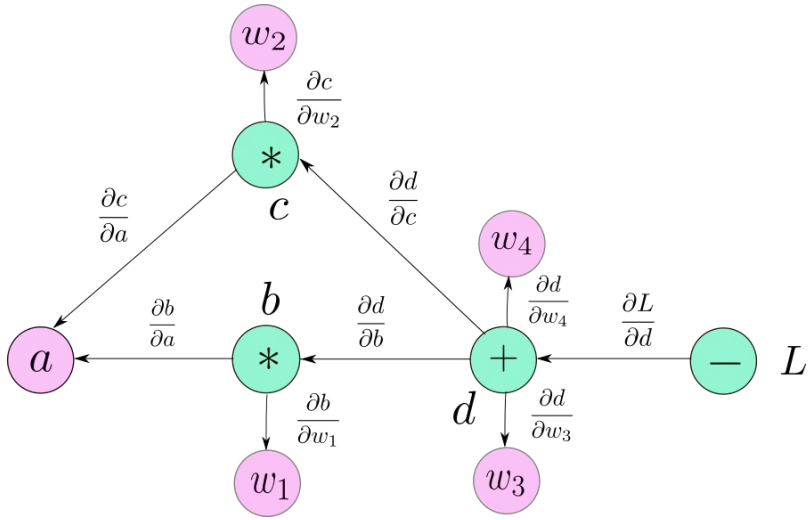
$$\frac{\partial f}{\partial x} = \frac{\partial t}{\partial g}\frac{\partial g}{\partial h}\frac{\partial h}{\partial x}$$



Derivatives of complex, nested functions is efficiently solved through **backpropagation**

Backpropagation

Compute gradients iteratively working backwards through the network



Key ideas:

- Cache things on the forward pass
- Compute vector-Jacobian products rather than full Jacobians

$$J^T \cdot \vec{v} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \begin{pmatrix} \frac{\partial l}{\partial y_1} \\ \vdots \\ \frac{\partial l}{\partial y_m} \end{pmatrix} = \begin{pmatrix} \frac{\partial l}{\partial x_1} \\ \vdots \\ \frac{\partial l}{\partial x_n} \end{pmatrix}$$

Optimisers

We have our gradients – still need to search high-dimensional parameter space effectively…

Maximilian will discuss various choices and how you should select an optimizer (but just use ADAM)

DNNs in PyTorch

Using DNNs in PyTorch follows exactly the structure we've already used.

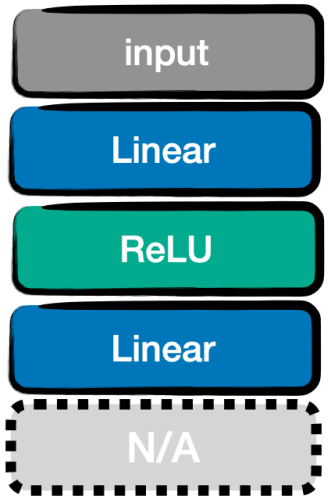
```
model = nn.Sequential(
    nn.Linear(10, 64),
    nn.ReLU(),
    nn.Linear(64, 32),
    nn.ReLU(),
    nn.Linear(32, 1),
    nn.Sigmoid()
)
```

```
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(10, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 1)


    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.sigmoid(self.fc3(x))
        return x
```

Key choices

Regression

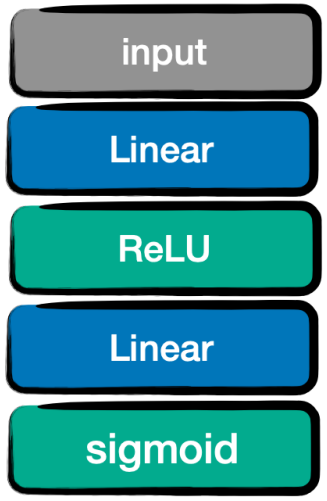
$\mu_\phi(x) \in \mathbb{R}$



No activation!

Binary Classification

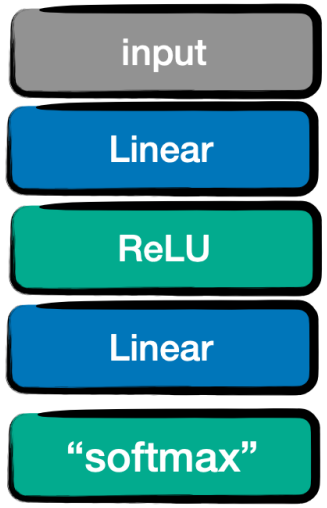
$\theta(x) \in [0,1]$



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

Multi-class Classification

$p_i(x) \geq 0 \text{ s.t. } \sum p_i = 1$



$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

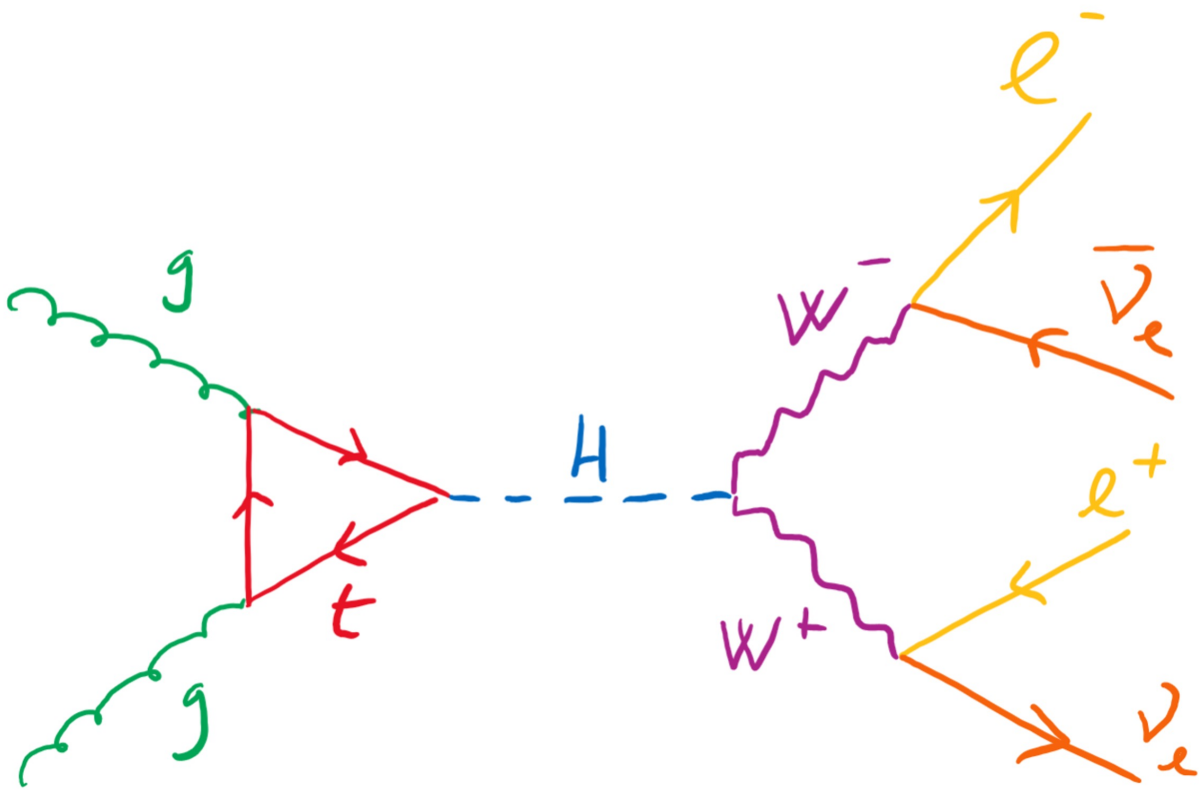
What I haven't covered...

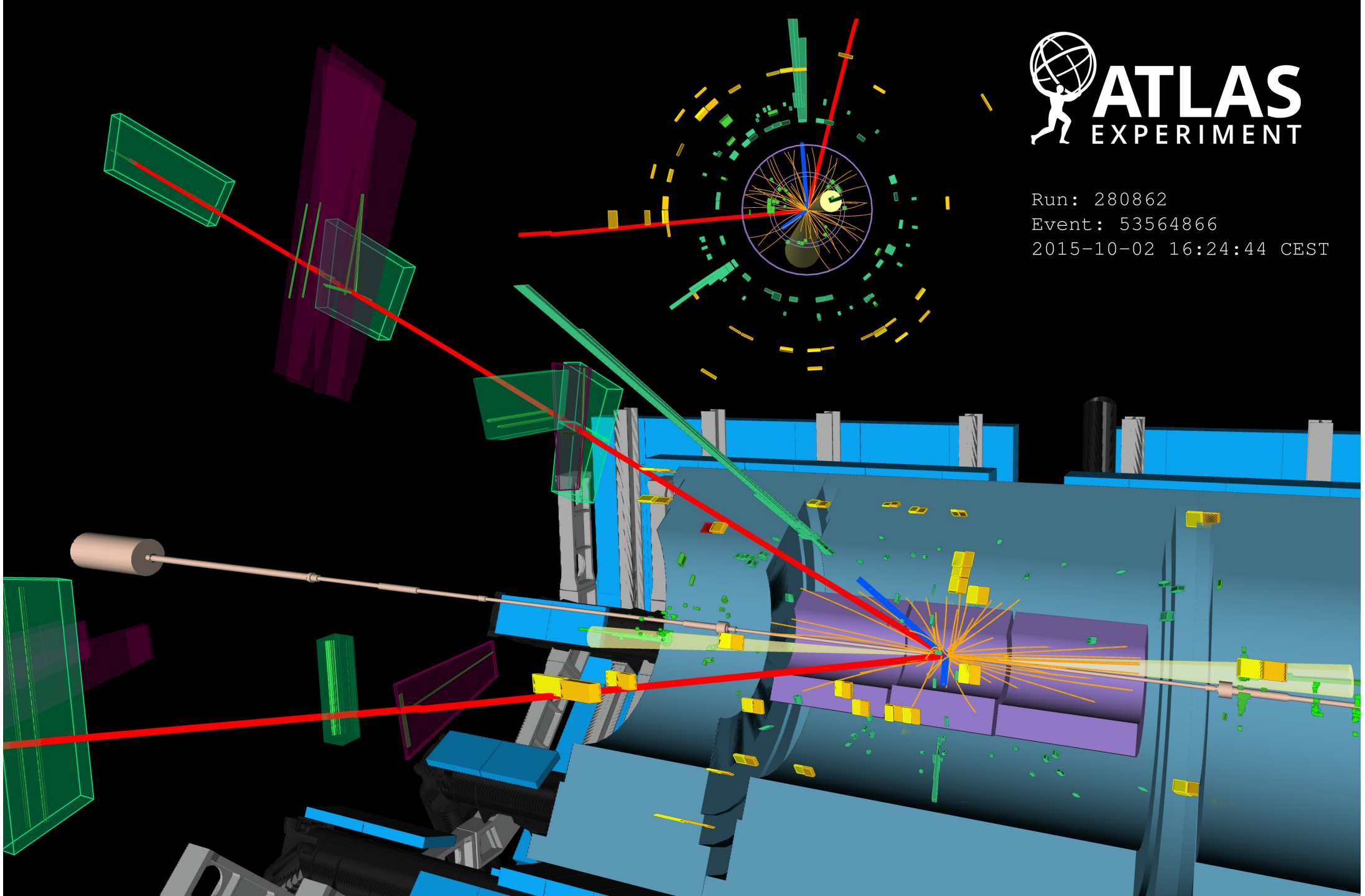
- Details of back-propagation
- Wider choices of loss functions?
- What if my data has some inherent structure I can utilize?
- Normalising the data
- Optimizing hyperparameters
- Reducing overfitting

DNN Challenge

- At the LHC, we collider protons together.
- This creates lots of different particles that we measure in detectors.
- Annoyingly, many different intermediate particles can be created which result in the same detected state

- For example, pp collisions can create a Higgs boson, which can decay to 2 leptons and things we call jets
- But pp collisions can also create two top quarks, which can decay to the same thing.


g
g
t
H
W⁻
W⁺
e⁻
ν̄e
e⁺
νe


ATLAS
EXPERIMENT
Run: 280862
Event: 53564866
2015-10-02 16:24:44 CEST

Beyond DNNs

A DNN with one hidden layer can in principle approximate anything…

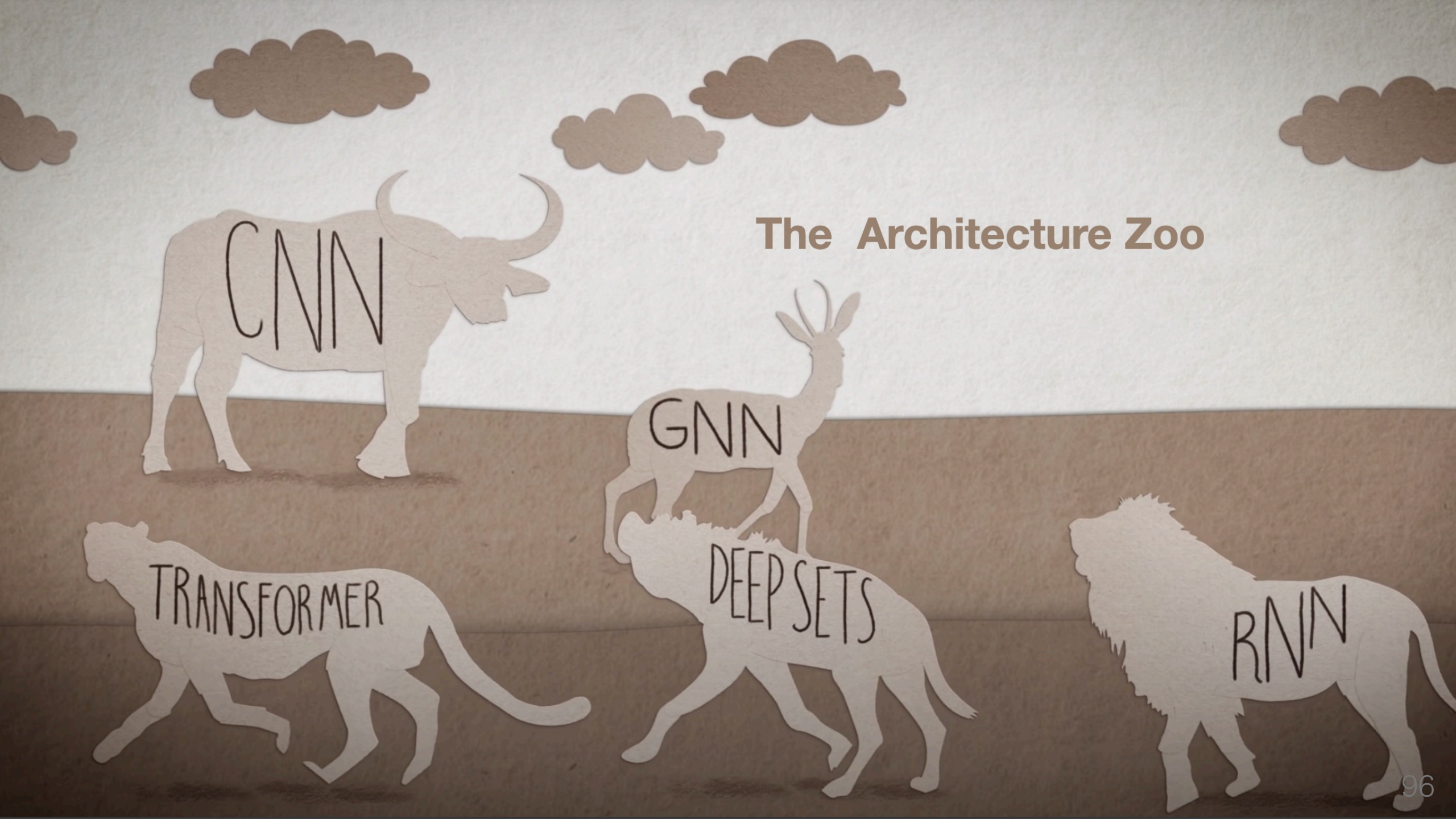
In reality, the solution space may be too large. We need to be smarter…

Inductive bias

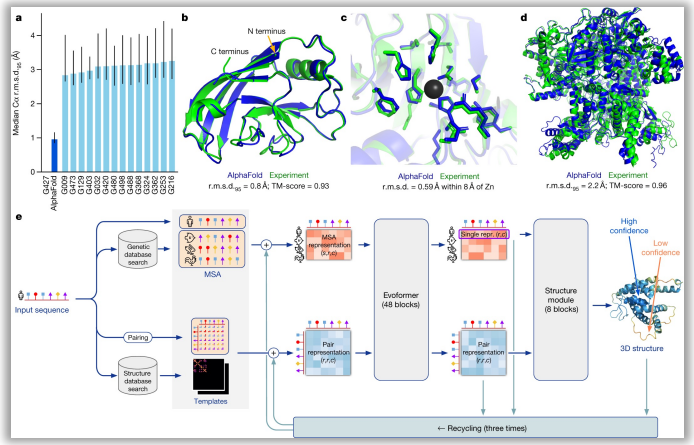
Use prior knowledge of the data to design architectures which constrain the solution space?

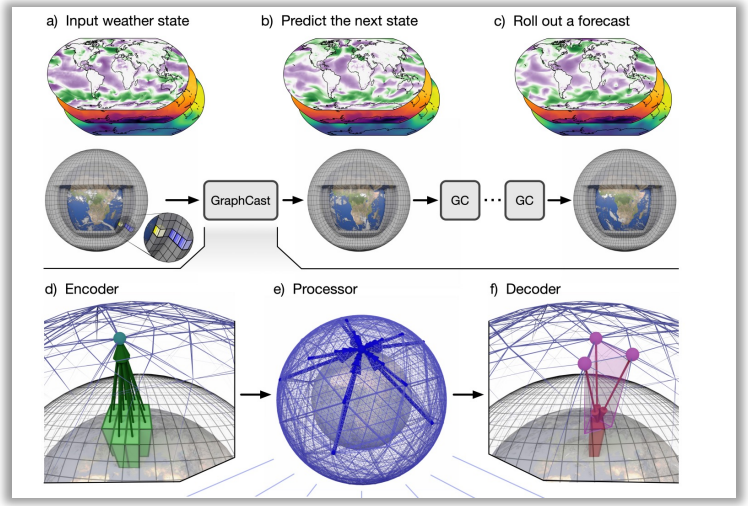
Data structure?

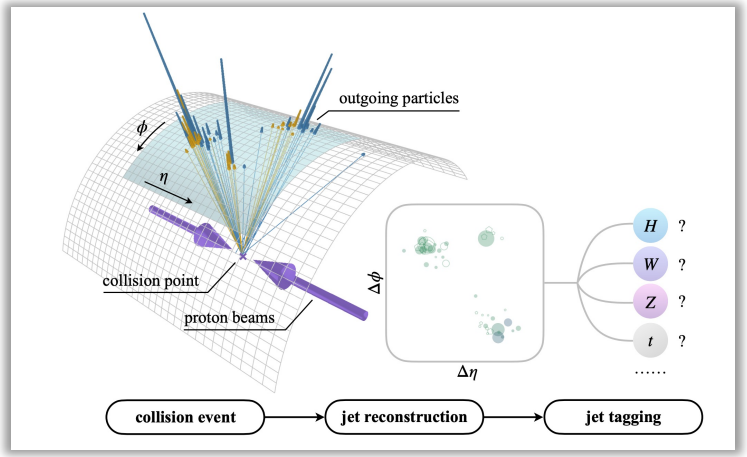
Laws of physics?


The Architecture Zoo
CNN
GNN
TRANSFORMER
DEEP SETS
RNN

SOTA Revisted





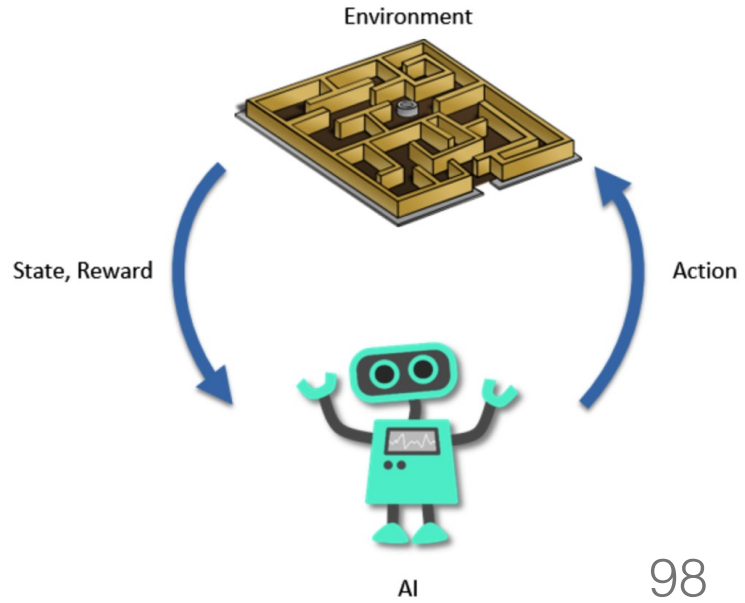


- Structured representations of data
- Inductive bias via symmetries (e.g. CNNs)
- Context-dependent interactions (e.g. attention)

ChatGPT

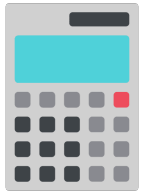
ALPHAGO

Gemini



Conclusions

ML is changing everything, in physical science too



DNNs are a generalization of simpler models – all based on same principles

Model → loss → optimiser → training loop

Easy to do in PyTorch!