~/warwick-wake-ml

session 1: from linear models to machine learning

Thomas Killestein

WarwickAstro/WAKE_workshops

Workshop format

Session 1 (today)

> From linear models to ML (fundamentals)

Session 2 (tomorrow)

> Machine learning in modern astronomy (applications)

Session 3 (Thursday)

> Hackathon-style session: bring your own dataset/ML/issues from the course

\+ additional time to work on notebooks through the week

Ethos

> Teach **concepts** not **code** - notebooks have some minimal level of coding required to follow, but more interested in grasping the bigger picture.

> Notebooks come in `_full.ipynb` and `_workbook.ipynb` `_workbook.ipynb` has elements missing for you to fill in.

> It's ok **(encouraged)** to refer to documentation/slides/ask for help/chat to your fellow participants - this is unfamiliar territory for most of us, and learning to use all resources available is a central part of ML.

> Slides: high-level overview, notebooks: implementation details.

JAX: our framework of choice

JAX is a differentiable linear algebra library that compiles your numpy-like code using XLA - this enables JAX to do quite a few neat things.

- Automatic differentiation with primitives like `grad`, `value_and_grad`, `jvp`, etc.
- JIT compilation [=speed!]
- Seamless computation on GPU/TPU [=more speed!]

Numpy-like syntax -> in the notebooks replace `np` with `jnp`, with the vast majority of routines implemented.

A few gotchas - see next slide.

JAX gotchas

- Randomness doesn't work like numpy - instead we manually handle advancing the RNG with `jax.random.split(rng_key)`. See notebook for examples, but this is intentional given parallelism.
- JAX ndarrays are immutable (can't be changed in place) - this means you can't change values without re-writing the array.

- The more painful gotchas have been taken care of by me in the notebooks :)
- **Ask** if you don't understand any error messages!

./machine-learning

background

machine learning jargon

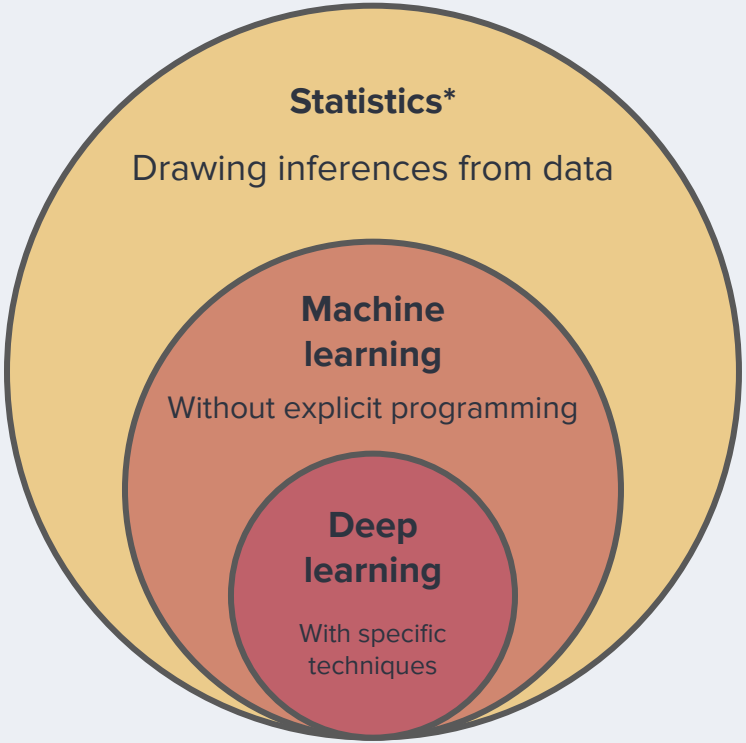
prep content for notebooks

What is machine learning?

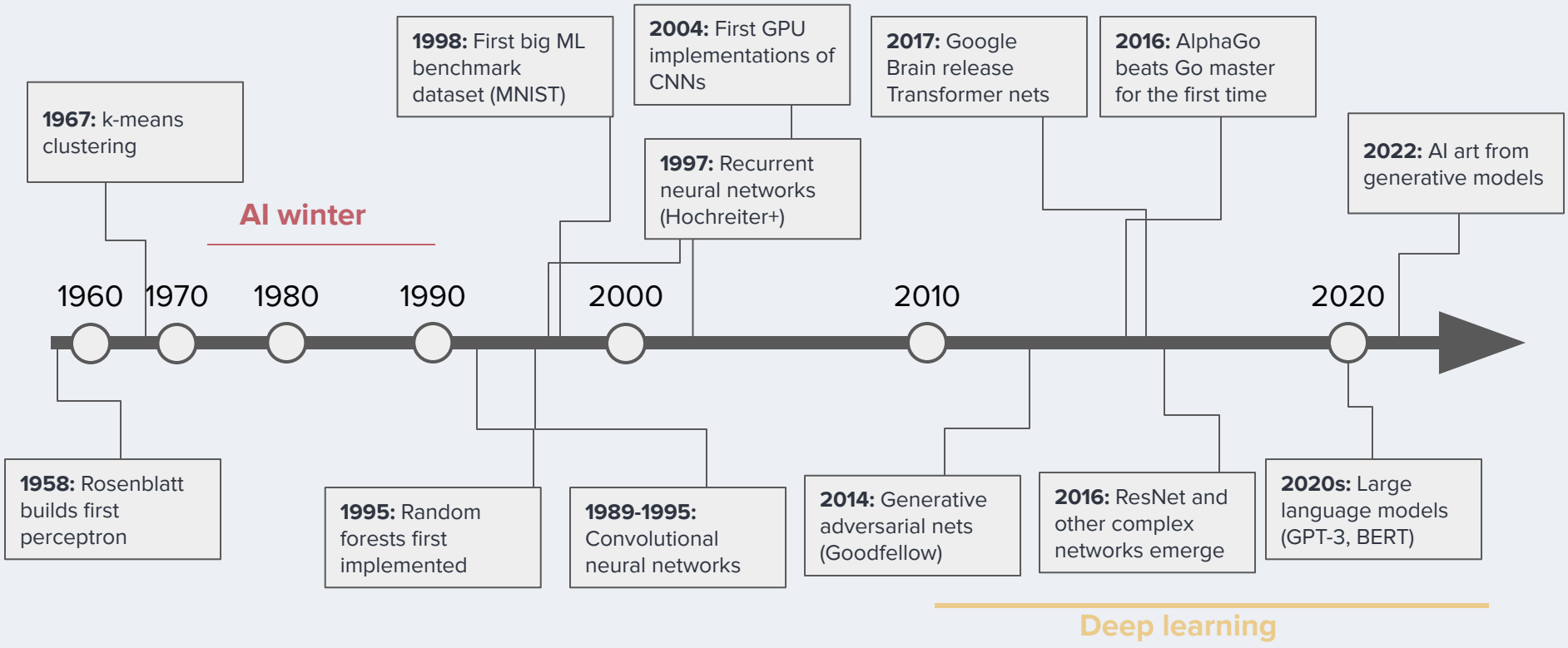
* Some put AI here, but this is widely regarded as a buzzword

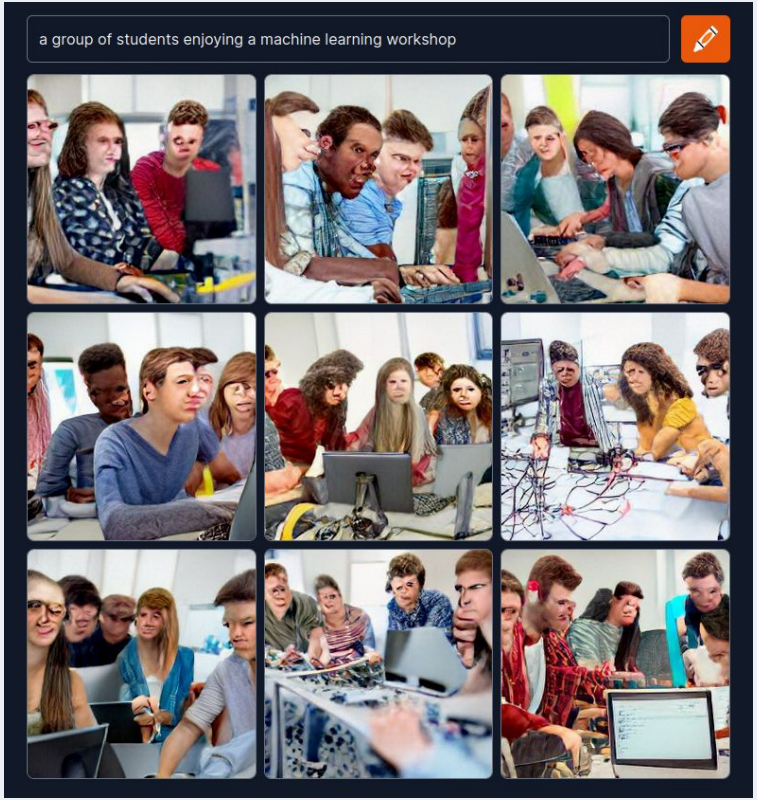
Machine learning is defined by algorithms that draw inferences from data without explicit human coding - they leverage data to improve their performance independently.

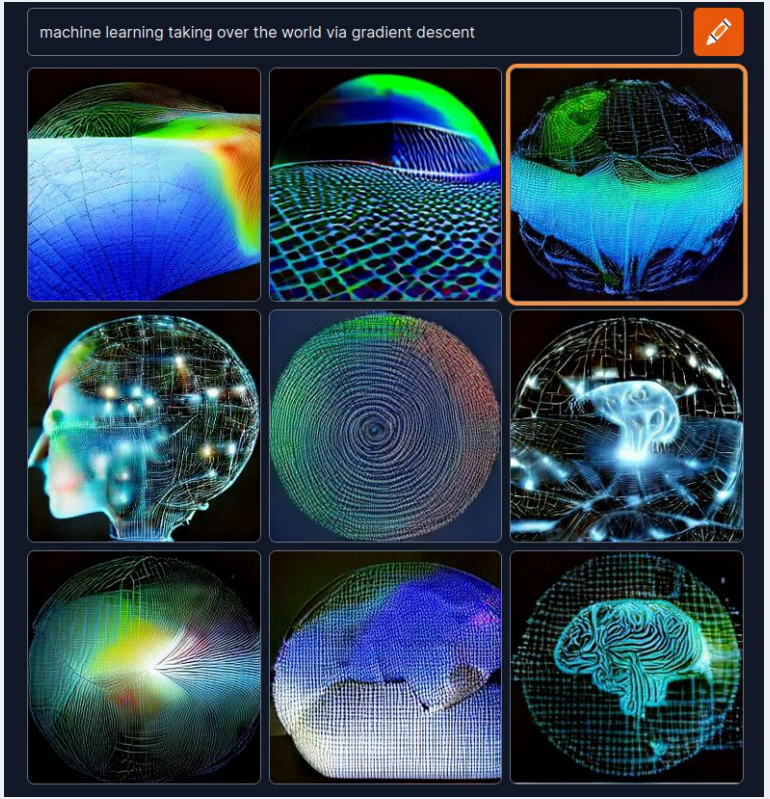
[strictly speaking, any algorithm that learns from data to make empirical predictions is a machine learning algorithm!]



Historical landscape of ML (log scale!)







craiyon.com – DALL-E mini

Why use machine learning?

- **Non-parametric:** you don’t need to know (or assume a functional form) for your data.
- **Data-driven:** predictive power benefits from large amounts of data
- **Fast:** ML is perfect for accelerating slow tasks/minimising human effort

Supervised: we have data, and want to predict labels/values for some quantity.

Unsupervised: we want the algorithm to learn some low-dimensional representation (hopefully informative), we have no labels to use.

Semi/self-supervised: bit of both, we have a large unlabelled dataset and a smaller labelled one

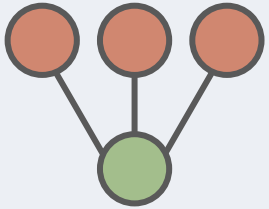
When not to use machine learning

- **For ‘simple’ problems:** often a lot of tasks that ML is used for could be replaced with something simpler, or a pre-existing algorithm with better guarantees about robustness.
- **For ill-defined tasks:** if you can’t easily specify what you want to get out, the results from ML are not likely to align with what you want.
- **For ‘small’ datasets:** generalisation performance is likely to be poor (with caveats for e.g. few-shot learning)
- **For datasets with the potential to cause societal harms:** ML by definition amplifies biases and overlooks minority groups (even when we try our best to compensate)

Three key ingredients of any ML algorithm

Model

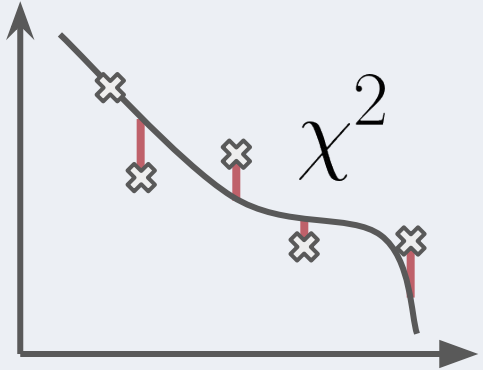
Mathematical function that generates our predictions



$$f(x) = ax^2 + bx + c$$

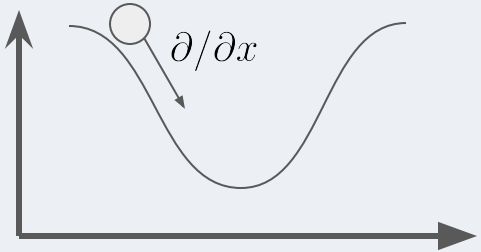
Loss function

Tells us how well we're learning the dataset based on our outputs



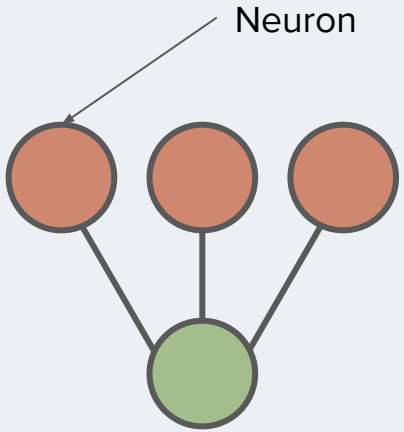
Optimiser

Changes model parameters to minimise the loss function



Model: neural network

Neural networks are a popular choice for ML - they are fast, end-to-end differentiable, and are universal function approximators.



Each neuron takes in inputs, multiplies them by some weight, adds a bias factor and then applies a non-linear activation.

$$y_{out} = f(y_{in} \cdot w + b)$$

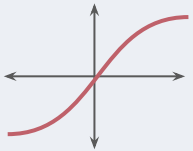
By feeding neurons into each other, and optimising the weights and biases with gradient descent, we can learn arbitrarily complex functions.

The simplest possible neural network

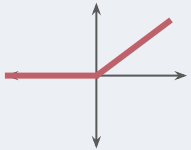
Is just linear regression with a twist!

Add an activation function to rectify output -> **generalised linear model**

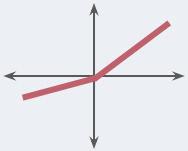
Sigmoid
(probability)



ReLU
(non-negative)



More complex



Loss function: classification/regression

Given a set of outputs, the loss function tells us how well we're fitting the data, given the predictions from our model. This is not an exhaustive list!

Mean-squared error (L2 loss) (regression)

~= chi-squared

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))^2$$

Mean-absolute error (L1 loss) (regression)

Less sensitive to outliers

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

Cross-entropy (classification)

Measures 'distance' between probability distribution and true values

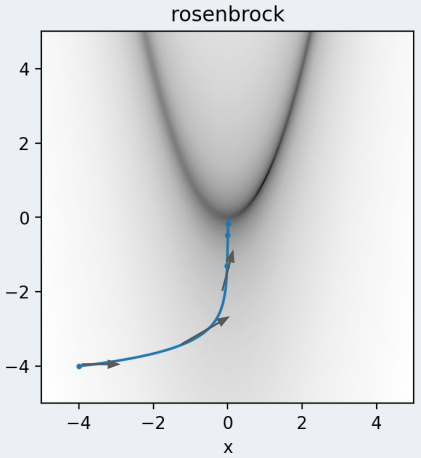
$$\frac{1}{N}\sum_{i=1}^{N} y_i \log_2 (f(x_i))$$

Optimiser: (stochastic) gradient descent

We can iteratively optimise our model via gradient descent – given some initial parameters, and the gradient of our loss function with respect to the parameters, we can take small steps in the direction that minimises loss and eventually converge on a local minimum.

This is known as *gradient descent*, and underpins ML.

$$x_{n+1} = x_n - \eta \nabla \mathcal{L}(x_n)$$



./over to you!

```
~> git clone git@github.com:WarwickAstro/WAKE_workshops.git
~> git checkout stable
~> pip install -r WAKE_workshops/requirements.txt
~> jupyter lab
```

https://mybinder.org/v2/gh/WarwickAstro/WAKE_workshops/stable