Machine Learning techniques for numerical solvers

Day 2: supervised learning

Maria Han Veiga

Mini-course SUSTech

11.03 - 14.03

THE OHIO STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

Schedule

- **Monday:** Introduction to Machine Learning (1h) 17:00-18:00
- **Tuesday:** Computational Framework + Supervised learning: integrating data-driven methods within a numerical solver + Hands-on session (2h) 14:00-16:00
- **Wednesday:** Unsupervised learning: Physics informed neural networks (1h) 11:00-12:00
- **Thursday:** Reinforcement Learning ? (1h) 11:00-12:00

Computational concerns

Outline

- Computational stack
- Reproducibility

Computational stack

Python:

- Scikit-learn: traditional machine learning tools
  - https://scikit-learn.org/
  - Cross-validation, datasets, feature transformation, many hypothesis classes (including neural networks)
- PyTorch: deep learning
  - https://pytorch.org/
  - Similar functionality as tensorflow, keras
  - Personal opinion: more explicit and easy to develop with, ML (theory) researchers seem to like it more

Reproducibility

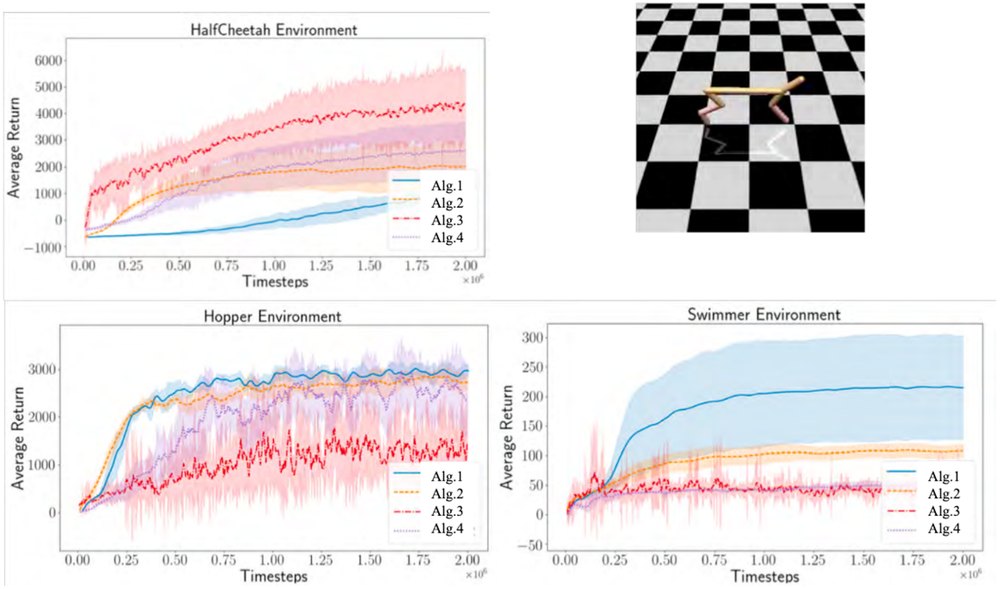
- *Experimental* Machine Learning can be thought of as an experimental science
- There are many sources of randomness:
  - sampling, optimisation, parameter initialisation, parallelism, etc…
- Other sources of *non-uniqueness*:
  - Development environment (e.g. different software versions, floating point accuracy, etc…)
- Experimental results are only as good as we can reproduce them

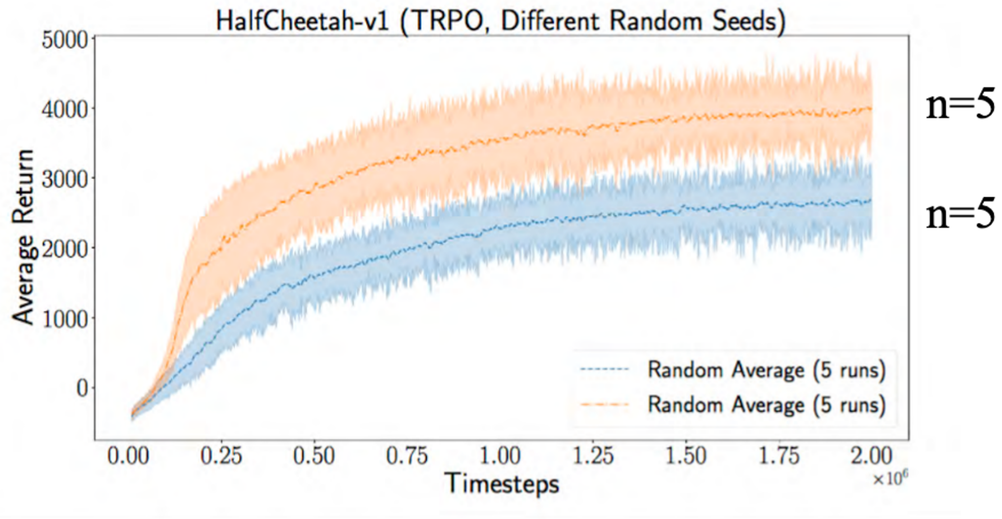
Reproducibility

A case study (Pineau et al. 2020)

- They studied the behaviour of common baseline algorithms that are used to compare to new algorithms

Consider Mujoco simulator:





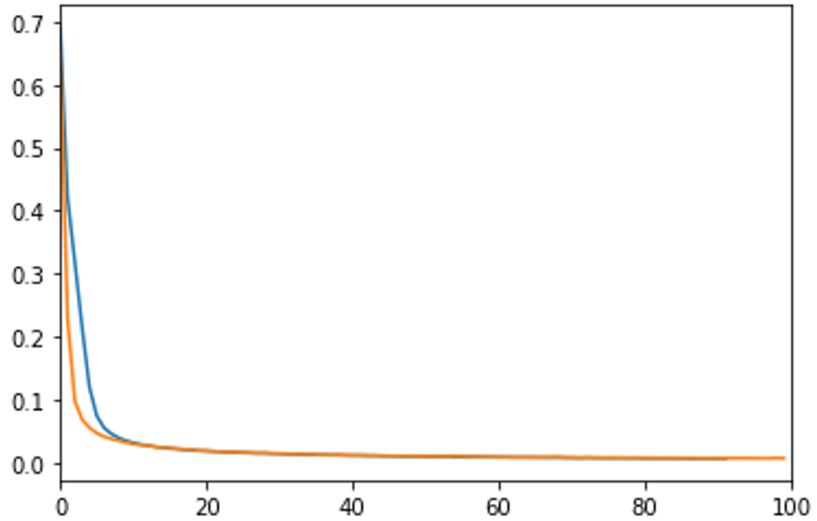
Reproducibility

```
# Define model (MLP) with sklearn
from sklearn.neural_network import MLPRegressor
modelSK = MLPRegressor(hidden_layer_sizes=[24,24,24])
# Fit model
modelSK.fit(scaled_x_train, scaled_y_train)
```

```
import keras
from keras.models import Sequential
from keras.layers import Input, Dense

inputs = Input(shape=(input_size,))
f = Dense(24,activation='relu')(inputs)
f = Dense(24,activation='relu')(f)
f = Dense(24,activation='relu')(f)
outputs = Dense(1)(f)
model = keras.Model(inputs, outputs)
model.compile('Adam', loss="mse", metrics="mse")
```

- Same training / test set
- Same optimizer
- Architecture



	SK-Learn	Keras
R2 score	0.9789	0.9844

Tools that can be helpful

- **Dataset:**
  - Dryad https://datadryad.org/stash/
  - Zenodo https://zenodo.org/
  - University-hosted repositories
- **Experiments tracking:**
  - MLFlow https://mlflow.org/
  - TensorBoard https://www.tensorflow.org/tensorboard
  - Weights and biases https://wandb.ai/site
  - Or your own way to store trained models and experimental set-up
- **Presenting results and sharing code:**
  - Google colab
  - Github
  - Development environment, e.g. environment.yml

The Machine Learning Reproducibility Checklist (v2.0, Apr.7 2020)

For all **models** and **algorithms** presented, check if you include:

- ❑ A clear description of the mathematical setting, algorithm, and/or model.
- ❑ A clear explanation of any assumptions.
- ❑ An analysis of the complexity (time, space, sample size) of any algorithm.

For any **theoretical claim**, check if you include:

- ❑ A clear statement of the claim.
- ❑ A complete proof of the claim.

For all **datasets** used, check if you include:

- ❑ The relevant statistics, such as number of examples.
- ❑ The details of train / validation / test splits.
- ❑ An explanation of any data that were excluded, and all pre-processing step.
- ❑ A link to a downloadable version of the dataset or simulation environment.
- ❑ For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control.

For all shared **code** related to this work, check if you include:

- ❑ Specification of dependencies.
- ❑ Training code.
- ❑ Evaluation code.
- ❑ (Pre-)trained model(s).
- ❑ README file includes table of results accompanied by precise command to run to produce those results.

For all reported **experimental results**, check if you include:

- ❑ The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results.
- ❑ The exact number of training and evaluation runs.
- ❑ A clear definition of the specific measure or statistics used to report results.
- ❑ A description of results with central tendency (e.g. mean) & variation (e.g. error bars).
- ❑ The average runtime for each result, or estimated energy cost.
- ❑ A description of the computing infrastructure used.

Reproduced from: *www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist-v2.0.pdf*

- Proposed by Dr. Pineau
- Adopted by NeurIPS

A shorter checklist

- *environment.yml* to recreate runtime environment
- Versioned dataset (if size allows it, otherwise use timestamps)
- Full code used to preprocess data
- Full code used to train and evaluate model(s)
- Hyperparameters used for training (model, optimiser)
- Final trained models in some universally readable format
- Strict version control of GitHub repo (for example)

Day 2: Supervised learning

Outline

- Examples of supervised learning approaches in numerics
- General considerations
  - Identifying problem
  - Dataset sampling
  - Training/testing
  - Integration with existing software
- Pitfalls, concerns, and other things

- Hands-on session:
  - Machine learning tricks of the trade
  - Training a shock detector for 1D problems

with a 10 min break
somewhere in between

Supervised learning

Task: Learn function $f : \mathcal{X} \to \mathcal{Y}$ when given dataset $S = \{(x_i, y_i) : i = 1,...,m\}$ where $y_i = f(x_i)$.

Supervised learning

Task: Learn function $f : \mathcal{X} \to \mathcal{Y}$ when given dataset $S = \{(x_i, y_i) : i = 1,...,m\}$ where $y_i = f(x_i)$.

- Shock detection: given a solution, is there a shock there? (Ray et al. 2018, Han Veiga et al. 2019, Zheng et al. 2020 and others)
- Shock capturing: given a solution, return a limited solution. (Kurz et al. 2022, Bruno et al. 2022, and others)
- Given a solution, return an integrated quantity. (Lye et al. 2020 and others)
- Given left and right data, solve the Riemann problem. (Tokareva et al. 2019, Ruggeri et al. 2022 and others)

General considerations

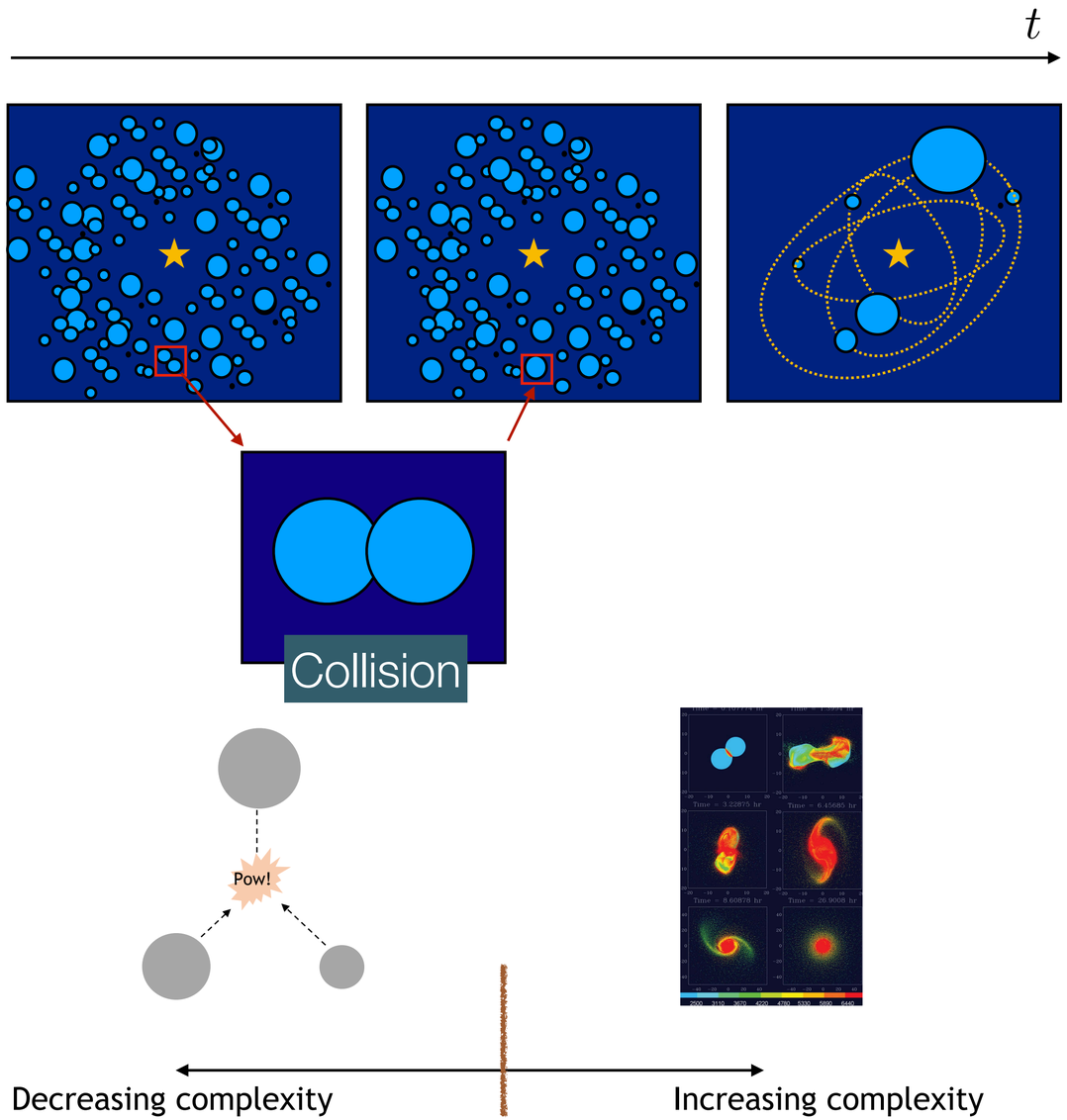
Wish to learn some map $f : \mathcal{X} \to \mathcal{Y}$, when we have available $S = \{(x_i, y_i) : i = 1,...,m\}$.

- What is $\mathcal{X}$ and $\mathcal{Y}$?
- How to generate S? Simulate, observe, sample?
- What is a suitable structure for h to approximate f?
- How efficient does h have to be? What is the computational cost of training and running h?

General considerations: a case study

- Study the evolution of a planetary system, from proto-planets to planets.
  - Over millions of years, proto-planets collide and eventually form the planets for a planetary system
  - Each collision is a complicated process
  - Model this evolution as a two-scale process

General considerations: a case study



Modeling:

1. System of particles $\{x_i\}_{i=1}^{N}$, with positions r_i and velocities v_i, that interact under gravity, governed by the system of equations:

$$\frac{dr_i}{dt} = v_i \qquad \text{for i} = 1, \cdots, N$$

$$\frac{dv_i}{dt} = -G \sum_{j \neq i}^{N} m_j \frac{r_i - r_j}{|r_i - r_j|^3}$$

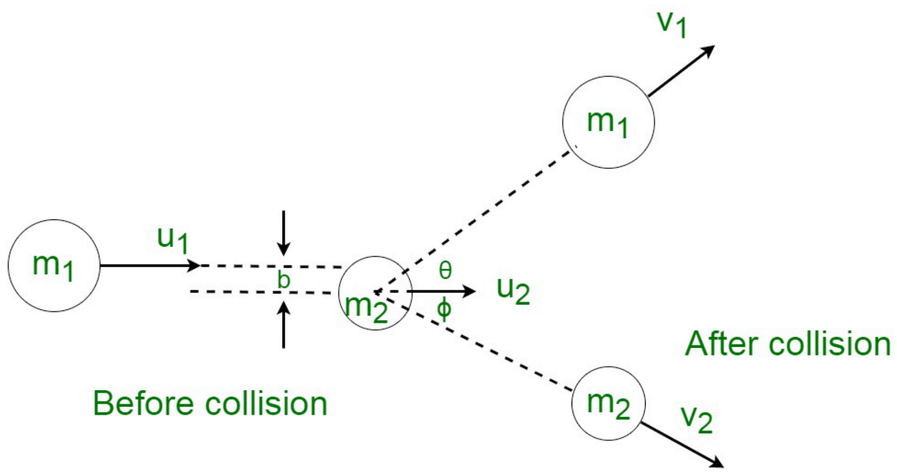
2. Model for two body interaction

- Subgrid/analytical models
- Modelling the collision with CFD
- Something in between?

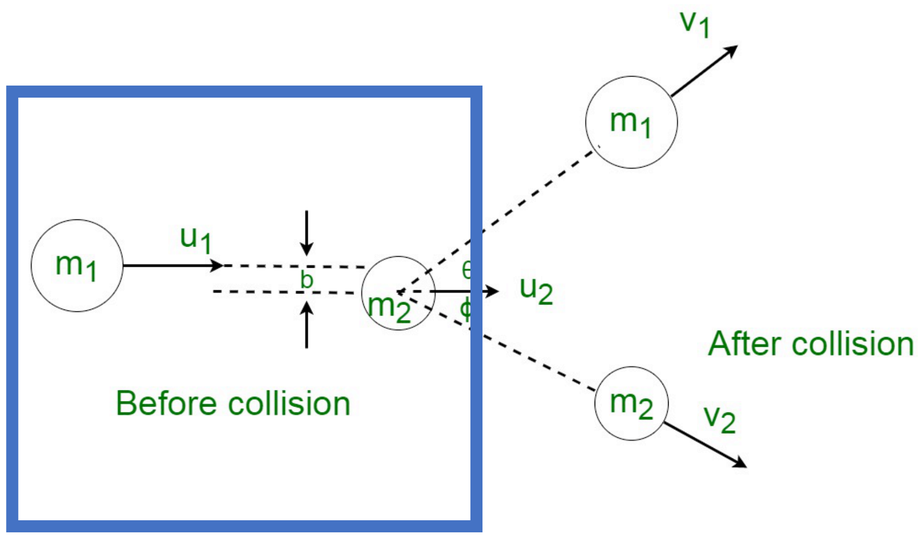
Learning problem: case study

- What is the outcome of a two-body collision?

Learning problem: case study



Learning problem: case study



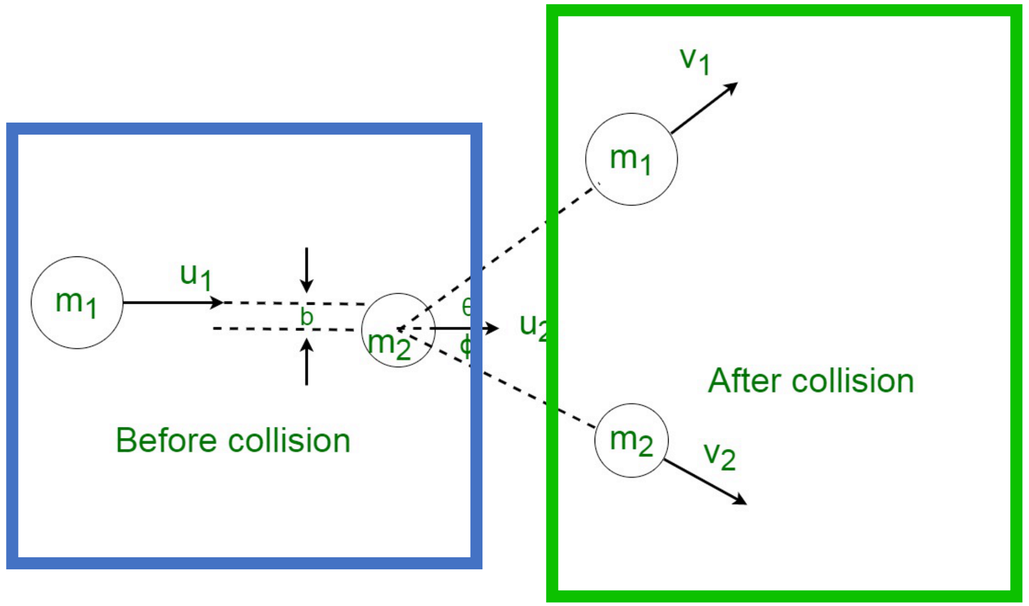
Input quantities:

- Total mass of system
- Mass ratio
- Impact parameters
- Target (m2) and projectile (m1) properties

Defines the collision initial conditions

$$\vec{x} = [x_1, \cdots, x_k] \in \mathcal{X} \subset \mathbb{R}^{12}$$

Learning problem: case study



Input quantities:

- Total mass of system
- Mass ratio
- Impact parameters
- Target (m2) and projectile (m1) properties

Defines the collision initial conditions

$$\vec{x} = [x_1, \cdots, x_k] \in \mathcal{X} \subset \mathbb{R}^{12}$$

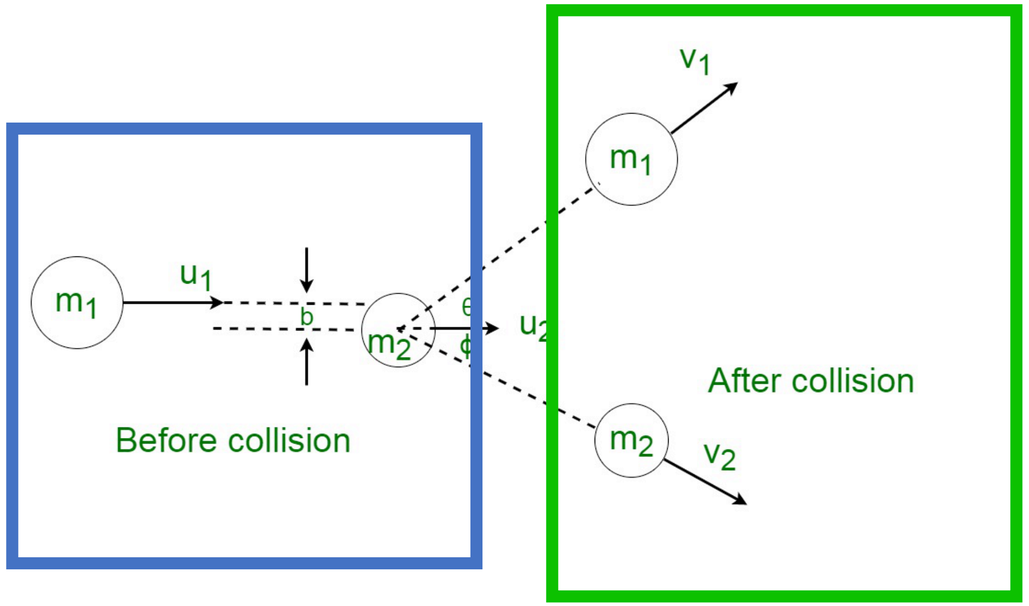
Output quantities:

- Properties of largest remnant (mass, velocity, rotation…)
- Properties of second largest remnant
- Properties of debris

Quantities of interest for the n-body code

$$\vec{y} = (y_1, \cdots, y_p) \in \mathcal{Y}$$

Learning problem: case study



Input quantities:

- Total mass of system
- Mass ratio
- Impact parameters
- Target (m2) and projectile (m1) properties

Defines the collision initial conditions

$$\vec{x} = [x_1, \cdots, x_k] \in \mathcal{X} \subset \mathbb{R}^{12}$$

Output quantities:

- Properties of largest remnant (mass, velocity, rotation…)
- Properties of second largest remnant
- Properties of debris

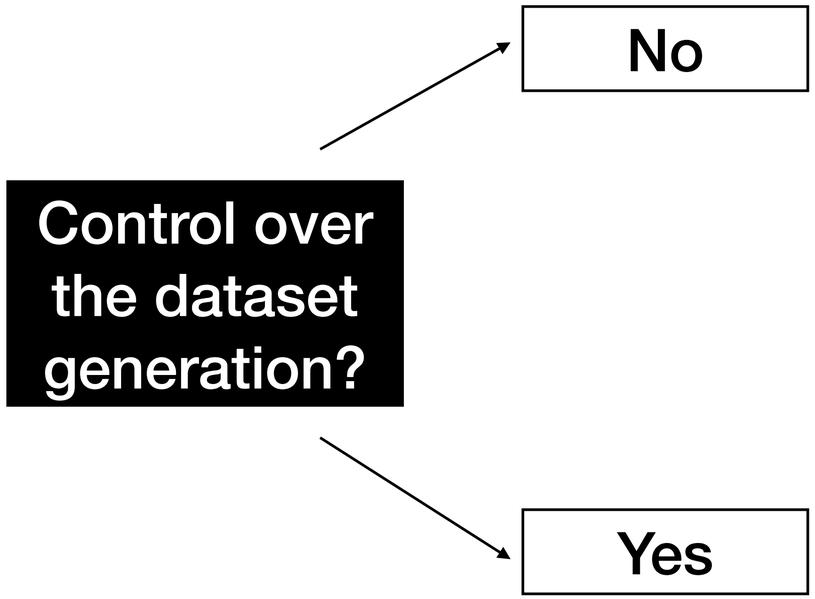
Quantities of interest for the n-body code

$$\vec{y} = (y_1, \cdots, y_p) \in \mathcal{Y}$$

Given $\mathcal{H}$ and $\mathcal{M}_\theta \in \mathcal{H}$ where $\mathcal{M}_\theta : \mathcal{X} \to \mathcal{Y}$, find parameters θ that solve:

$$\min_\theta \sum_{i=1}^{M} \ell(\mathcal{M}_\theta(x_i), y_i)$$

Dataset generation



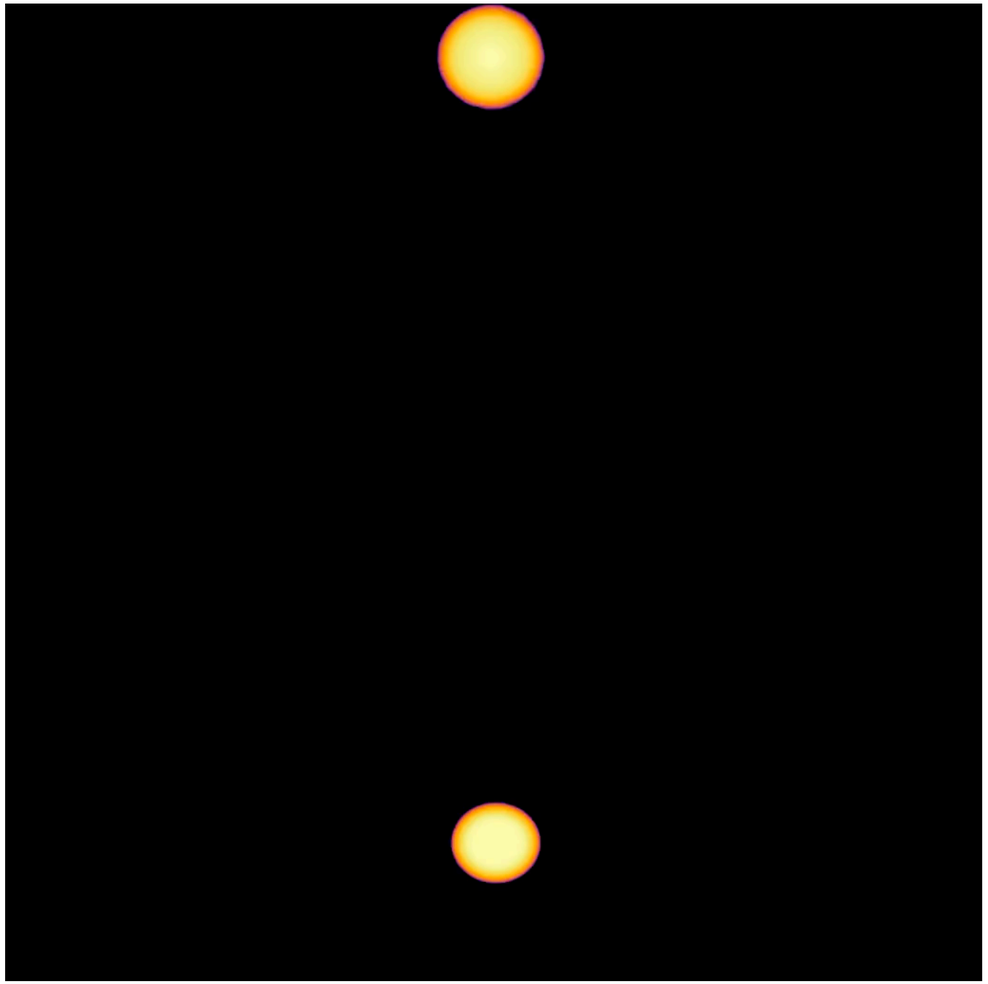
- Computational budget?
- Sampling the input / output space?

Dataset generation: case study

Input space:

Parameter	Range	Unit	Description
M_{tot}	$0.1 - 2$	$\mathrm{M}_{\oplus}$	Total mass ($M_{targ} + M_{proj}$)
γ	$0.1 - 1$	-	Mass ratio ($M_{proj} \div M_{targ}$)
b_{∞}	$0 - 1$	$\mathrm{R}_{\mathrm{grav}}$	Asymptotic impact parameter
v_{∞}	$0.1 - 10$	$\mathrm{v}_{\mathrm{esc}}$	Asymptotic impact velocity
F_{targ}^{core}	$0.1 - 0.9$	-	Target core mass fraction
Ω_{targ}	$0 - 0.9$	Ω_{crit}	Target rotation rate
θ_{targ}	$0 - 180$	deg	Target obliquity
ϕ_{targ}	$0 - 360$	deg	Target azimuth
F_{proj}^{core}	$0.1 - 0.9$	-	Projectile core mass fraction
Ω_{proj}	$0 - 0.9$	Ω_{crit}	Projectile rotation rate
θ_{proj}	$0 - 180$	deg	Projectile obliquity
ϕ_{proj}	$0 - 360$	deg	Projectile azimuth

Hypercube of dimension 12



Sampling dataset

Suppose we want to sample from a hypercube $[0,1]^d \subset \mathbb{R}^d$.

- Uniform sampling: Sample uniformly s values per dimension. This yields the number of samples:

$$|S| = s^d.$$

Grows extremely fast with the dimension.

Sampling dataset

Suppose we want to sample from a hypercube $[0,1]^d \subset \mathbb{R}^d$.

- Uniform sampling: Sample uniformly s values per dimension. This yields the number of samples:

$$|S| = s^d.$$

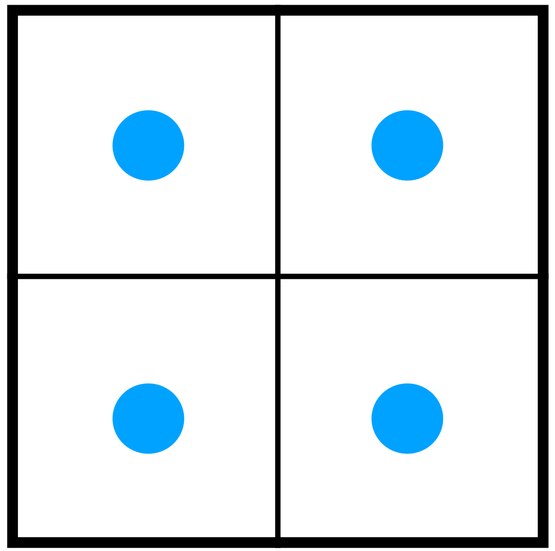
Grows extremely fast with the dimension.

Alternatively, given a computational budget for N data points, randomly sample the input space N times.

Sampling dataset

In between randomly sampling and uniformly sampling: **Latin hypercube sampling** (McKay et al. 1979).

- Stratified sampling.
- Example: $d = 2, N = 4$

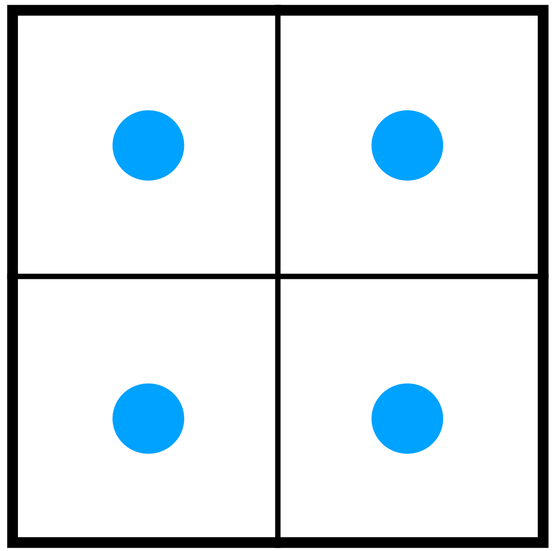


Uniform sampling

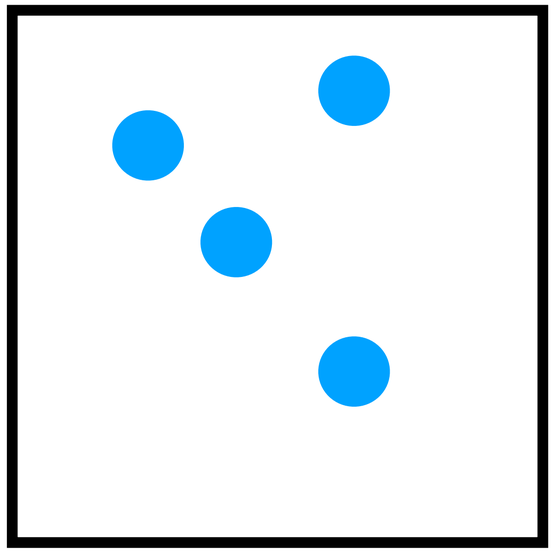
Sampling dataset

In between randomly sampling and uniformly sampling: **Latin hypercube sampling** (McKay et al. 1979).

- Stratified sampling.
- Example: $d = 2, N = 4$



Uniform sampling

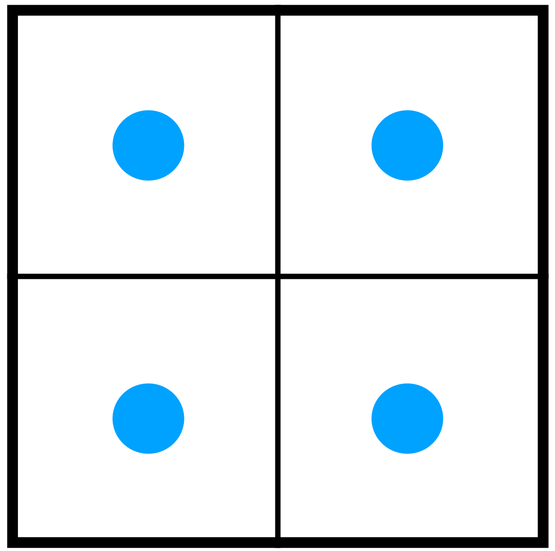


Random sampling

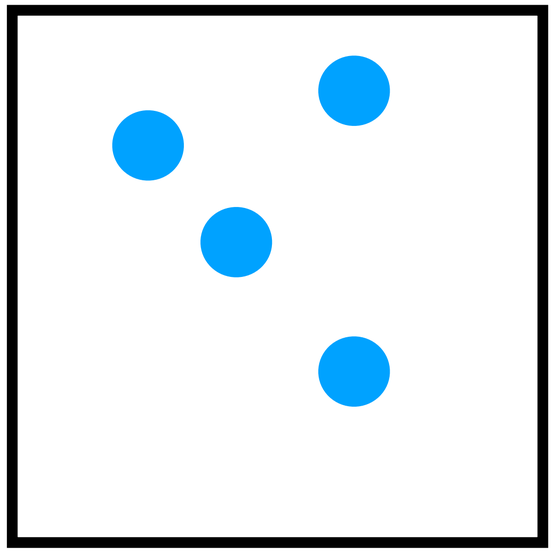
Sampling dataset

In between randomly sampling and uniformly sampling: **Latin hypercube sampling** (McKay et al. 1979).

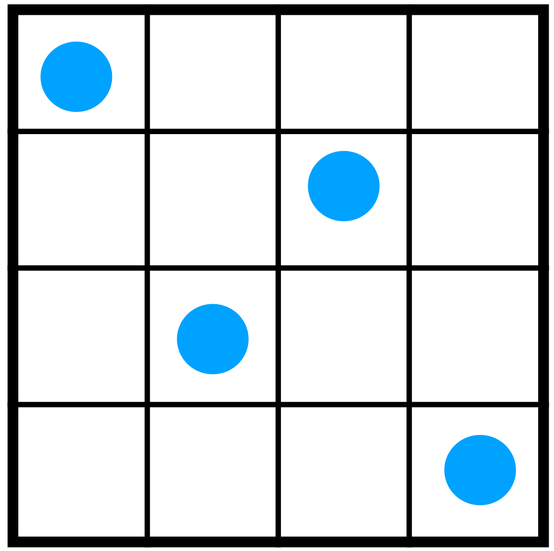
- Stratified sampling.
- Example: $d = 2, N = 4$



Uniform sampling



Random sampling

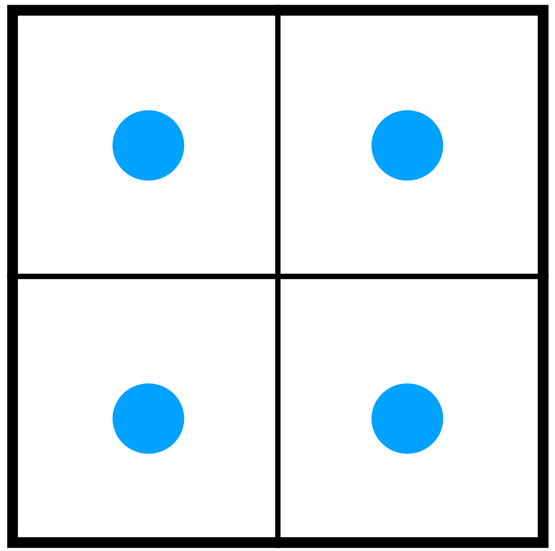


Latin hypercube sampling

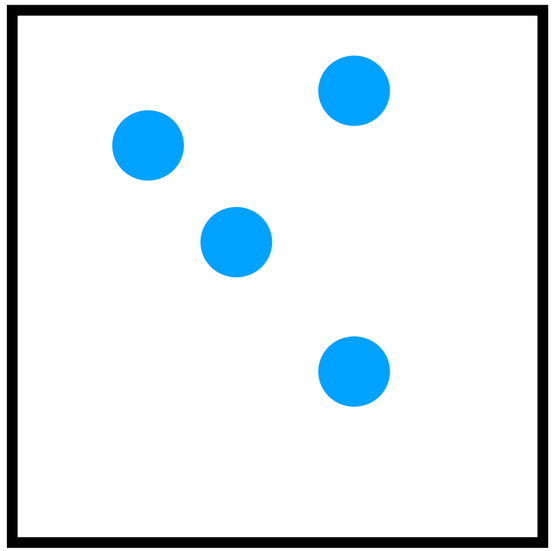
Sampling dataset

In between randomly sampling and uniformly sampling: **Latin hypercube sampling** (McKay et al. 1979).

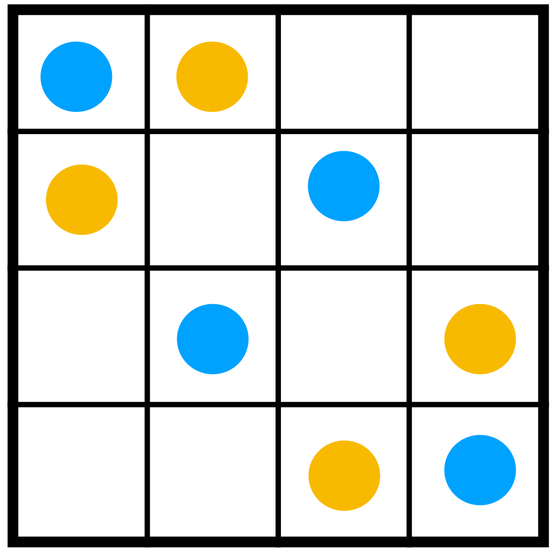
- Stratified sampling.
- Example: $d = 2, N = 4$



Uniform sampling



Random sampling



Latin hypercube sampling

Sampling dataset

In between randomly sampling and uniformly sampling: **Latin hypercube sampling** (McKay et al. 1979).

- Stratified sampling.

Definition: The Latin hypercube sampling method is as follows: choose randomly N different multi-integers $j \in \{1,...,N\}^d$ with the constraint that for any two different multi-indices j, j' we have

$$\forall i, \quad j_i \neq j'_i.$$

Sampling dataset

Example: Computing the median of $X_1 + X_2$. The true value is 5.

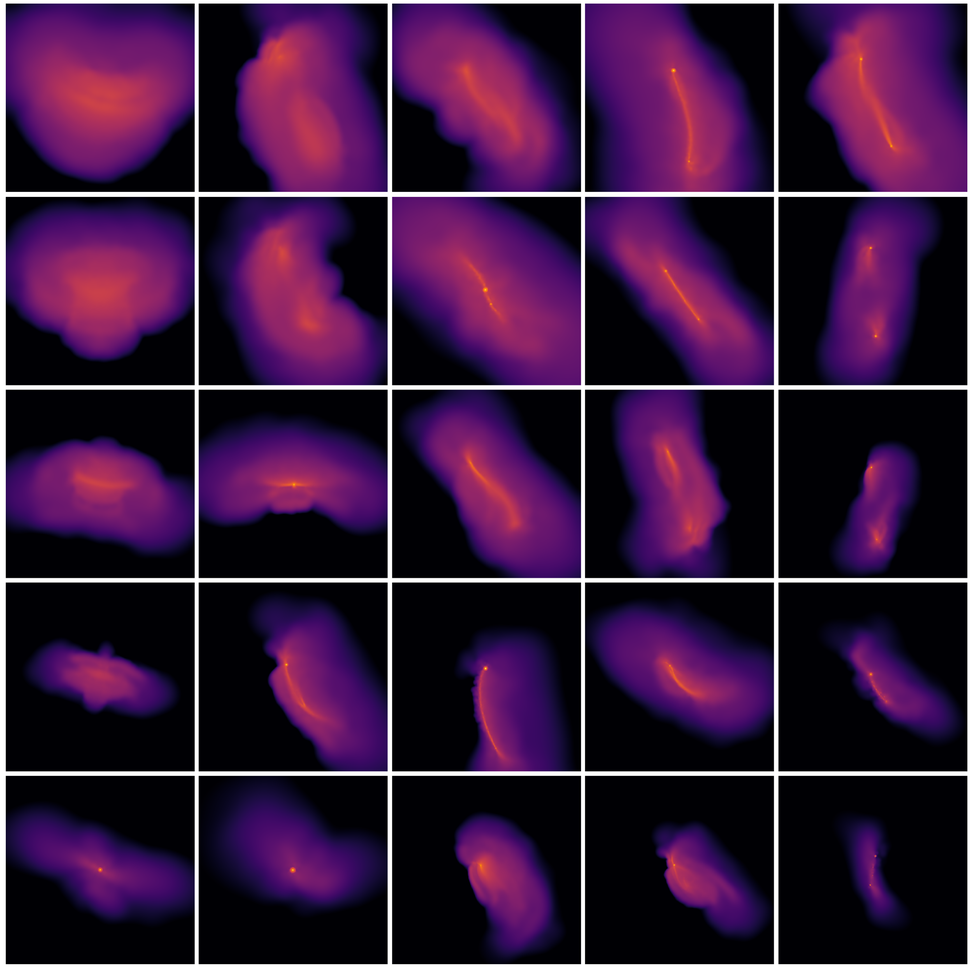
Source: https://datasciencegenie.com/latin-hypercube-sampling-vs-monte-carlo-sampling/

Empirically:

- Less variability in the estimates
- Faster convergence
- In some tasks, there's not much advantage, but not worse than random

Sampling dataset: case study

- Sampling a 12-dimensional input space
  - Latin hypercube sampling (McKay et al. 1979)
- Each sampled point is used to generate a pairwise collision simulation
- 10,500 simulations
  - After each, we compute the quantities of interest (e.g.: mass of the resulting body)



Dataset: Miles Timpe, MHV, Mischa Knabenhans, Joachim Stadel, Stefano Marelli (2020), Simulations of planetary-scale collisions between rotating, differentiated bodies, Dryad, Dataset, https://doi.org/10.5061/dryad.j6q573n94

Data preparation

- Scaling data: **e.g:** $X \rightarrow \frac{X - \bar{X}}{\sigma_X}$

Data preparation

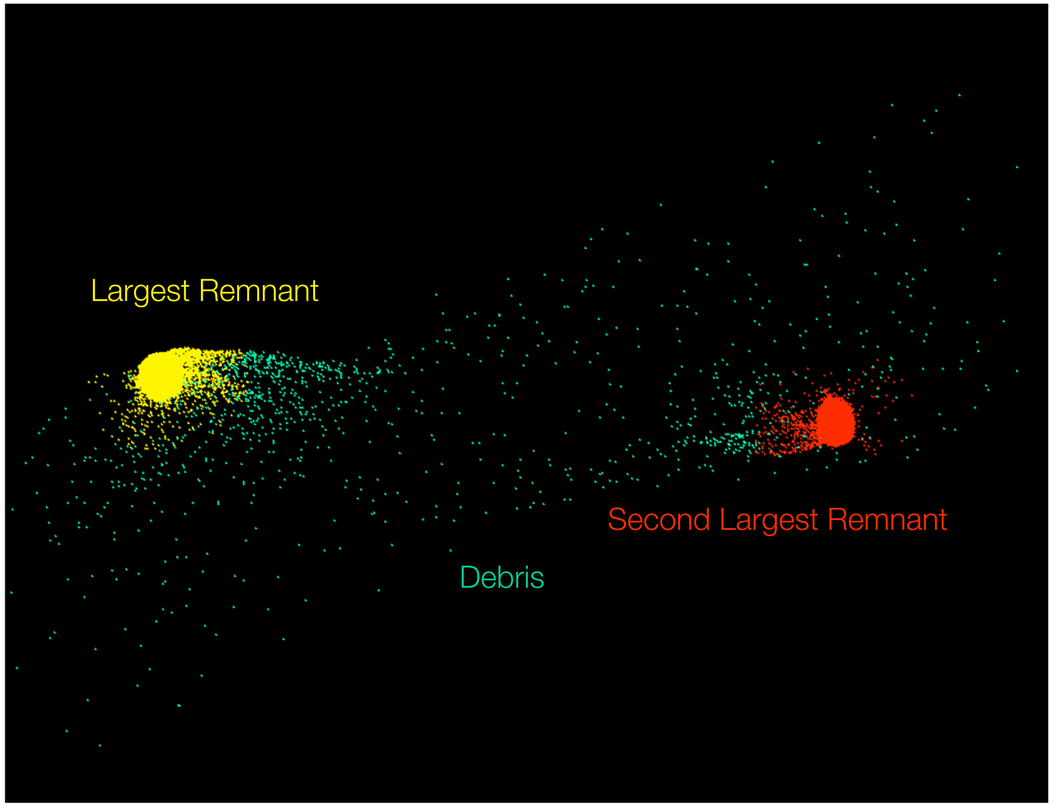
- Scaling data: **e.g:** $X \rightarrow \frac{X - \bar{X}}{\sigma_X}$
- Simplifying data: extracting relevant features
  - Domain knowledge

Data preparation

- Scaling data: **e.g:** $X \rightarrow \dfrac{X - \bar{X}}{\sigma_X}$

- Simplifying data: extracting relevant features

  - Domain knowledge



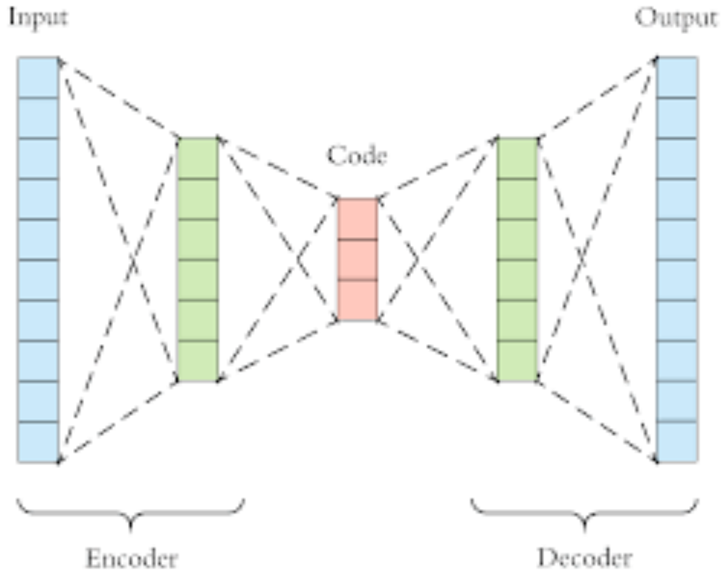
Data preparation

- Scaling data: **e.g:** $X \rightarrow \frac{X - \bar{X}}{\sigma_X}$

- Simplifying data: extracting relevant features

  - Domain knowledge

  - Encoders



Optimises reconstruction loss:

e.g. $\ell(X, X_{out}) = ||X - X_{out}||^2$

Protein representation, image representation, etc…

Training and testing: finding best model

- Cross-validation for different types of features, hypothesis classes, etc.
- Select best set-up under cross-validation
- Retrain and estimate generalisation error with unseen test set for that set-up
- **At the end:** trained model h that approximates map f
  - Evaluate new samples $x \in \mathcal{X}$

Integrating with existing codes

- Once model has finished training, you have a static model (mostly).
  - e.g. Trained neural network is defined by it's weights and biases and activation functions.

Integrating with existing codes

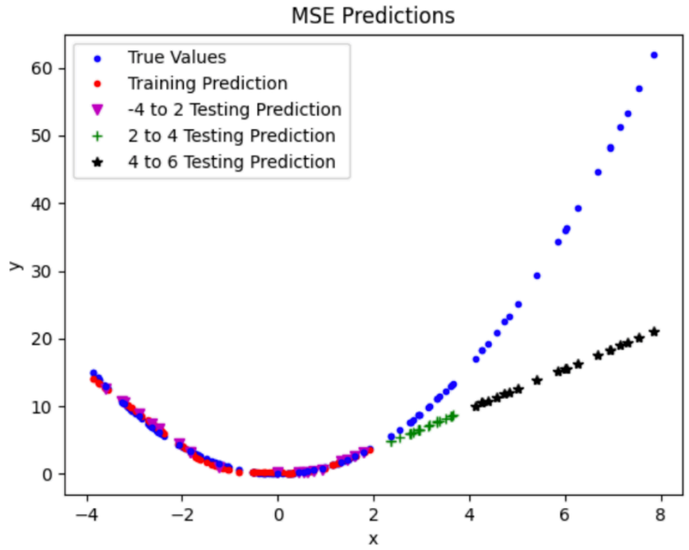
- Once model has finished training, you have a static model (mostly).
  - e.g. Trained neural network is defined by it's weights and biases and activation functions.
- To use in an existing code, implement two functions:
  - **Feature generation:** takes whatever data and generates x
  - **Prediction:** loads trained model and evaluates x

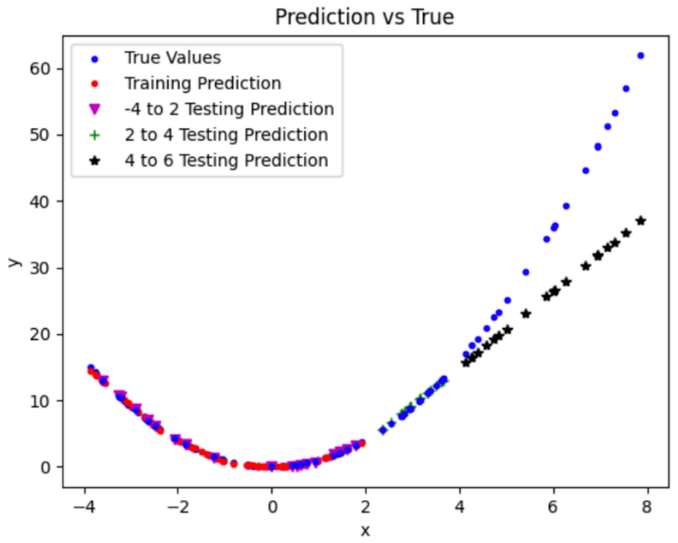
Pitfalls, concerns, etc…

- Structure preserving/awareness:
  - f is symmetric, invariant to some transformations, positive everywhere, bounded, etc…
  - Want h to have those same properties

Pitfalls, concerns, etc…

- Structure preserving/awareness:
  - **Example:** Learn a symmetric function, e.g.: $f(x) = x^2$
    Standard loss function: $\ell(y, h_\theta(x)) = (y - h_\theta(x))^2$
    Constraint guided loss function: $\ell(y, \hat{y}, \vec{x}) = (y - h_\theta(x))^2 + \lambda(h_\theta(x) - h_\theta(-x))^2$





Pitfalls, concerns, etc…

- Structure preserving/awareness:
  - Adding a penalty in the loss function:
    - $\ell(y, h_\theta(x)) = (y - h_\theta(x))^2 + \lambda Constraint(h_\theta(x), x)$
  - Built in structure into the hypothesis class
    - Special tricks
    - Ad hoc

More pitfalls, concerns, etc…

- Structure-preserving data-driven models
- Error estimates beyond generalisation error
- Uncertainty estimates on prediction
- Interpretability

10 min break
Restart at:

Tricks of the trade
(for neural networks)

Model initialisation

- The way the initial weights of an untrained neural network are set can influence the training
  - Zero initialisation: $W_{ij}^{(n)} = 0 \implies \dfrac{\partial C}{\partial W_{ij}^{(n)}} = 0$
  - Constant initialisation: $W_{ij}^{(n)} = c \implies \dfrac{\partial C}{\partial W_{ij}^{(n)}} = \beta c^{(n)}$
  - Both case, training does not occur

Model initialisation

- The way the initial weights of an untrained neural network are set can influence the training
  - Randomly initialise the weights
  - Xavier: $w_{ij}^{n} \sim Uniform(0, 1/\sqrt{d_{n-1}})$
  - Le Cun: $w_{ij}^{n} \sim \mathcal{N}(0, 1/d_{n-1})$
  - He: $w_{ij}^{n} \sim \mathcal{N}(0, 2/d_{n-1})$

Controls the initial variance of the weights so the variance across different layers is constant and training is stable in the first step of GD

Input normalisation

- Common to normalise the input such that:
  - Average of each input over the training set should be close to zero
  - Scale input variables so that their variance is about the same size
  - **E.g.** Suppose all components of an input vector are positive. The gradient update for any w_{ij}^n is
  - $w_{ij}^{n,t+1} = w_{ij}^{n,t} - \eta \frac{\partial L(w_{ij}^{n,t})}{\partial w_{ij}^n}$ where $\frac{\partial L(w_{ij}^{n,t})}{\partial w_{ij}^n} = \alpha_j^{n-1} \frac{\partial L}{\partial \tilde{\alpha}_i^n}$
  - First layer, $n = 1$: if all entries of input vector are positive, the updates of the weights that feed into node i will have the same sign: $sign\left(\frac{\partial L}{\partial \tilde{\alpha}_i^1}\right)$.
  - Weights can only increase or decrease together for a given input

Other practices

- Preventing overfitting:
  - Regularisation: $\ell(y, h_\theta(x)) = (y - h_\theta(x))^2 + \lambda R(\theta)$
  - Early stopping
  - Drop-out
- Stabilising gradients:
  - Gradient clipping
  - Batch normalisation

1D shock capturing