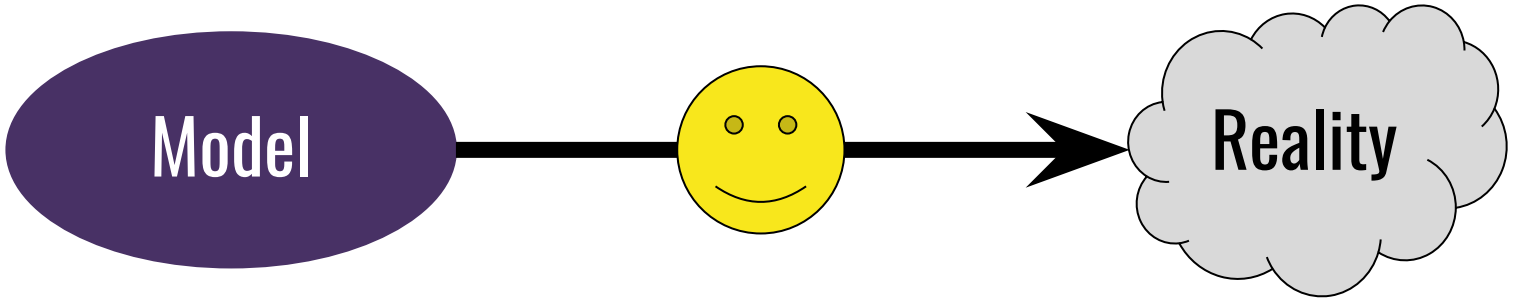
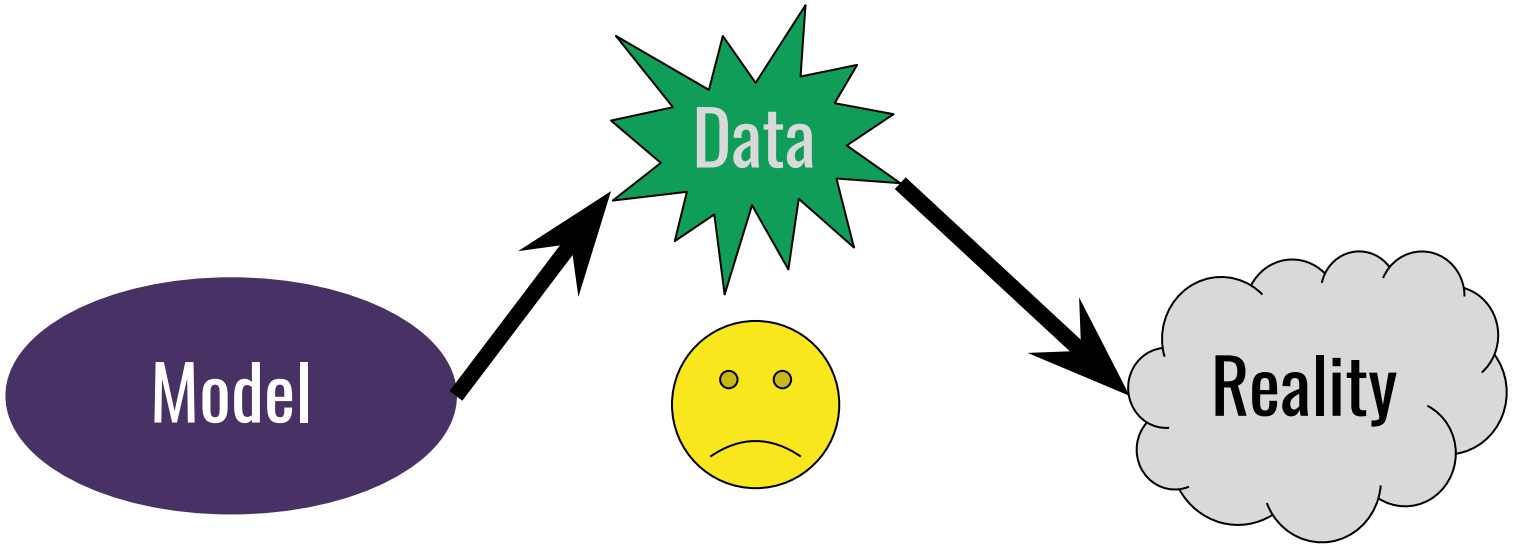
Fitting Markov Processes to Data with Neural Networks

Liam Hodgkinson

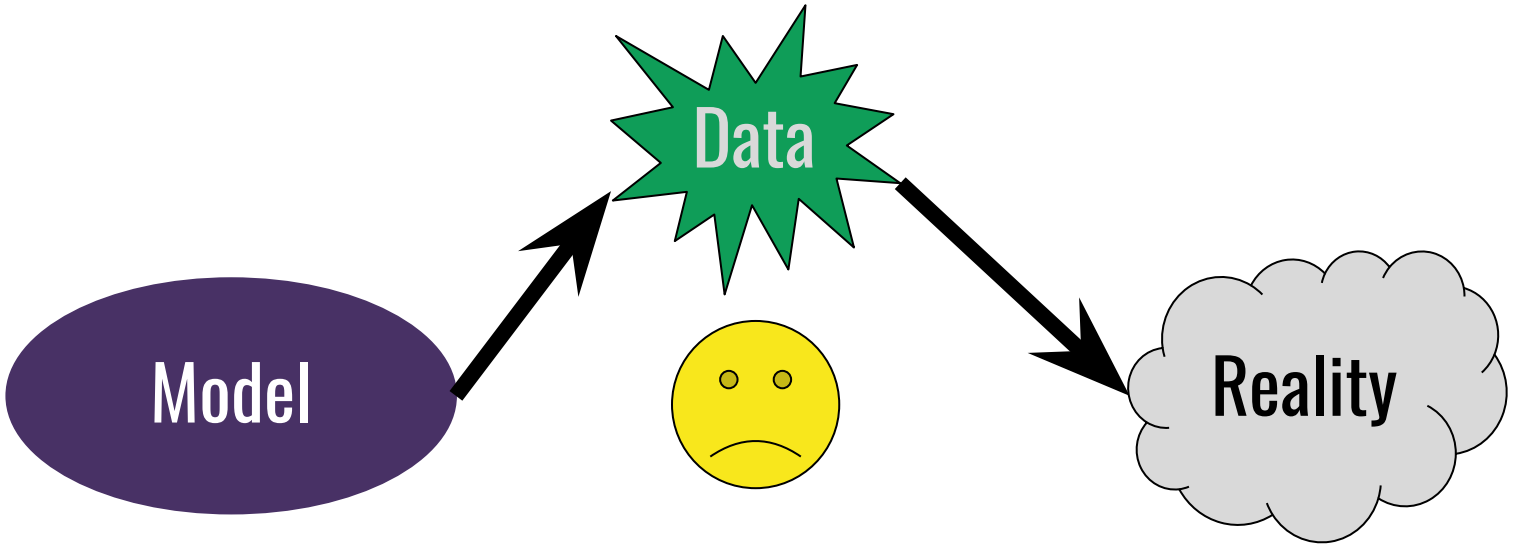
The Challenge



The Challenge



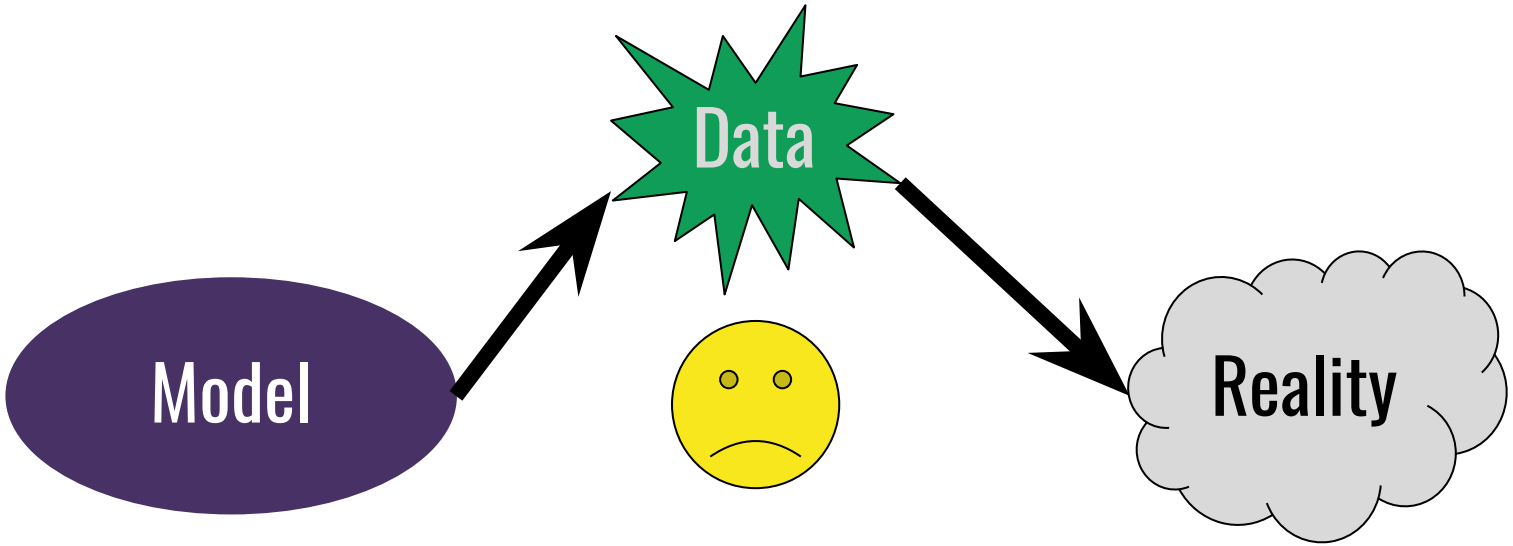
The Challenge



Maximum Likelihood?

MCMC?

The Challenge



~~Maximum Likelihood?~~ ~~MCMC?~~

The model can often be too complex to easily do these.

Approximate Bayesian Computation (ABC)

Starting with initial distribution (prior $p(\theta)$) on parameters:

1. Randomly sample parameters (from $\theta \sim p(\theta)$)
2. Simulate process from parameters ($Y \sim p(Y|\theta)$)
3. If the simulation is "close" to the data, accept θ; otherwise, reject θ.
4. Repeat 1–3 many times, obtain posterior distribution of parameters $p(\theta|Y)$

Improvement: **SMC-ABC**

Available in PyMC

We can now cheaply compute gradients of *anything.*

Optimisation is *virtually free* (for neural networks).



SciML

Open Source Software for Scientific Machine Learning



https://sciml.ai

Level 1: Training a Model

The Adjoint Method

Goal: Estimate parameters θ in an ODE $\dot{x} = f(x, \theta, t)$ given available data by minimizing a loss function / likelihood, e.g.

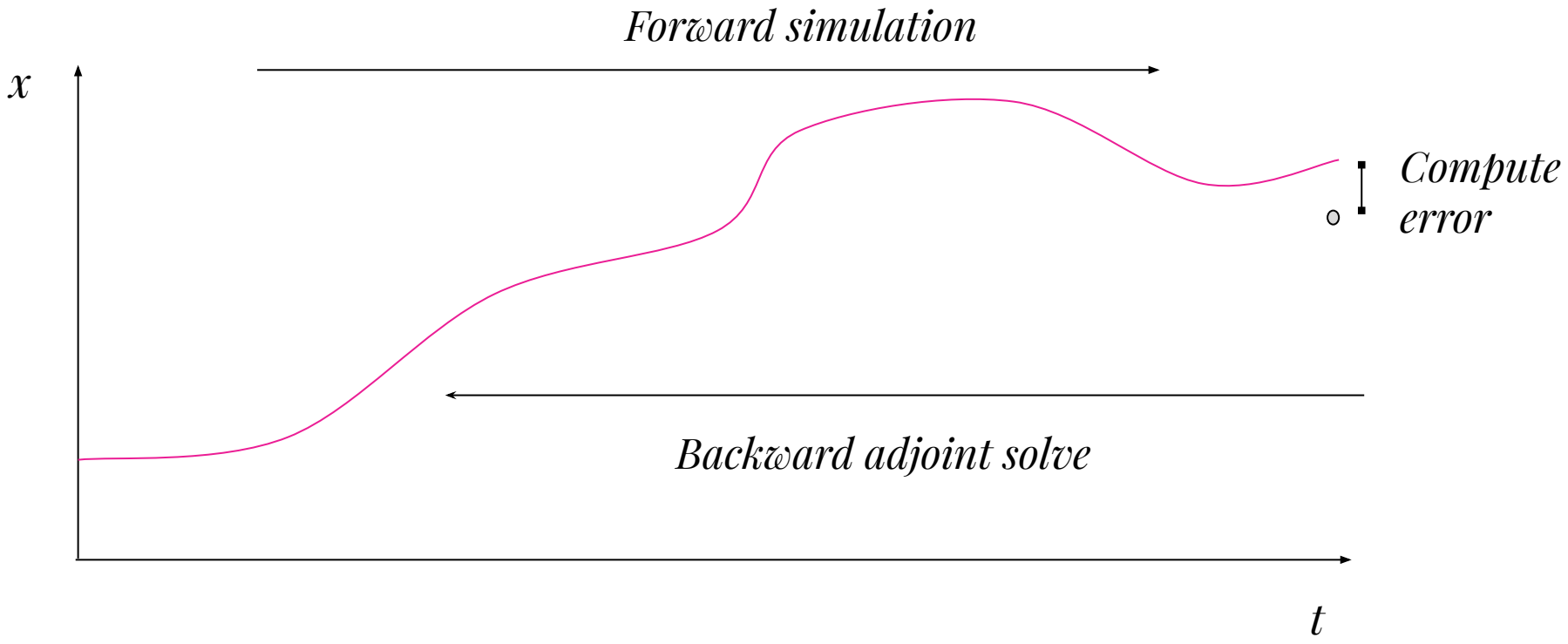
$$L(\theta) = \sum_{i=1}^{n} \|x(t_i) - x_{\text{obs}}(t_i)\|^2$$

Problem: Gradients using chain rule requires derivatives of all outputs in every coordinate and can be very expensive!

Solution: Pontryagin adjoint method. Consider $a(t) = \dfrac{\partial L}{\partial x(t)}$ which is known at terminal values. Then

$$-\dot{a} = \left(\frac{\partial f}{\partial x}\right)^{\top} a \qquad \text{and} \qquad \nabla_\theta L = \int_0^T \left(\frac{\partial f}{\partial \theta}\right)^{\top} a\, dt$$

The Adjoint Method



Neural Ordinary Differential Equations

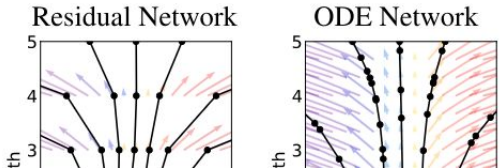
Ricky T. Q. Chen*, **Yulia Rubanova***, **Jesse Bettencourt***, **David Duvenaud**
University of Toronto, Vector Institute
{rtqichen, rubanova, jessebett, duvenaud}@cs.toronto.edu

Abstract

We introduce a new family of deep neural network models. Instead of specifying a discrete sequence of hidden layers, we parameterize the derivative of the hidden state using a neural network. The output of the network is computed using a black-box differential equation solver. These continuous-depth models have constant memory cost, adapt their evaluation strategy to each input, and can explicitly trade numerical precision for speed. We demonstrate these properties in continuous-depth residual networks and continuous-time latent variable models. We also construct continuous normalizing flows, a generative model that can train by maximum likelihood, without partitioning or ordering the data dimensions. For training, we show how to scalably backpropagate through any ODE solver, without access to its internal operations. This allows end-to-end training of ODEs within larger models.

1 Introduction

Models such as residual networks, recurrent neural network decoders, and normalizing flows build complicated transformations by composing a sequence of transformations to a hidden state:



Extensions to SDEs

Hodgkinson, L., van der Heide, C., Roosta, F., & Mahoney, M. W. (2021, December). Stochastic continuous normalizing flows: training SDEs as ODEs. In *Uncertainty in Artificial Intelligence* (pp. 1130-1140). PMLR.

Consider an (Ito) **stochastic differential equation**

$$dX_t = \mu_t(X_t)dt + \sigma_t(X_t)dB_t$$

This needs to be converted to a **Stratonovich** SDE:

$$dX_t^i = \mu_t^i(X_t) - \tfrac{1}{2}\sum_{j,k}\sigma_t^{jk}(X_t)\frac{\partial \sigma_t^{ik}}{\partial x^j}(X_t)dt + \sum_k \sigma_t^{ik}(X_t)\circ dB_t^k$$

Proper adjoint procedures exist, but are *very complicated*. An easier alternative (H. et al, 2021) is to **approximate the Brownian motion** and treat as an ODE.

$$B_t \approx \sqrt{2}\sum_{k=1}^{M} Z_k \frac{\sin((k-\frac{1}{2})\pi t)}{(k-\frac{1}{2})\pi}$$

Markov Jump Processes

For a Markov jump process X_t satisfying

$$\mathbb{P}(X_{t+h} = j \mid X_t = i) = q_{ij}(\theta)h + o(h)$$

consider the **Kolmogorov forward equation** for transition probabilities $i \to j$

$$\frac{dp_{ij}(t)}{dt} = \sum_{k \neq j} p_{ik}(t) q_{kj}(\theta) - p_{ij}(t) \sum_{k \neq j} q_{jk}(\theta)$$

The **likelihood** can only be explicitly evaluated for exact data.

Markov Jump Processes

With noisy observations (Y), the underlying Markov process (Z) is hidden.

Variational inference: the *ELBO* (evidence lower bound)

$$\log p(Y) \geq \mathbb{E}[\log p(Y|Z)] - D_{\mathrm{KL}}(q_\phi(Z|Y)\|p_\theta(Z))$$

$p_\theta(Z)$ is the likelihood from the Markov jump process Z

$p(Y \mid Z)$ is the distribution of the noise in the observations (normal)

$q_\phi(Z \mid Y)$ is the unknown posterior, approximated by a neural network

Adjoint Methods for All Processes

- Neural Ordinary Differential Equations (Chen et al., 2018)
- Neural Controlled Differential Equations (Kidger et al., 2020)
- Neural Stochastic Differential Equations (Li et al., 2020)
- Neural Hawkes Processes (Mei & Eisner, 2017)
- Neural Jump Stochastic Differential Equations (Jia & Benson, 2019)
- Neural Temporal Point Processes (Omi et al., 2019)
- Neural Markov Jump Processes (Seifner & Sanchez, 2023)
- Neural Jump-Diffusion Temporal Point Processes (Zhang et al., 2024)
- Neural Levy Processes (Ramirez-Gonzalez & Sun, 2026)
- Neural Fourier Operator (Li et al., 2020)
- Neural Stochastic PDEs (Salvi, Lemercier & Gerasimovics, 2022)

Part 2: Training Inference

Neural Bayes Estimators

Suppose that we can simulate model outputs $Y \sim p(Y|\theta)$.

Lemma. The function g that minimizes

$$\mathcal{L}(g) = \mathbb{E}_{\theta \sim p(\theta), Y \sim p(Y|\theta)} \|g(Y) - \theta\|^2$$

is the *posterior mean* $g(Y) = \mathbb{E}[\theta \mid Y]$

Idea. Parameterize g as a neural network (*DeepSets* architecture for permutation invariance over inputs) and minimize $\mathcal{L}(g)$ estimated over samples $\{(\theta_i, Y_i)\}_{i=1}^n$.

Other posterior estimators can be constructed in similar ways.

Sainsbury-Dale, M., Zammit-Mangion, A., & Huser, R. (2024). Likelihood-free parameter estimation with neural Bayes estimators. *The American Statistician*, *78*(1), 1-14.

So Why Use Neural Networks?

- The popular cited reason is that neural networks are universal approximators

  This is true, *but so are many other things…*

- Neural networks are excellent at capturing real-world behavior because of a **natural simplicity bias.**

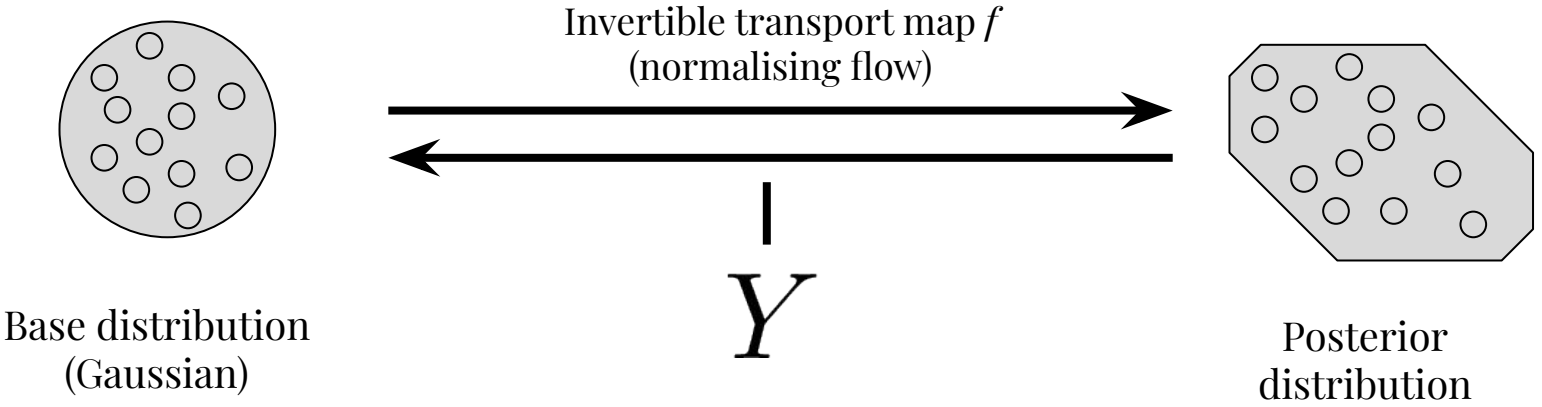
- They are also very easy to train now!



Neural Posterior Estimation

Suppose that we can simulate model outputs $Y \sim p(Y|\theta)$. **We will estimate the posterior** $p(\theta|Y)$ by the distribution $q(\theta|Y)$ of $f(Y, Z)$ where $Z \sim \mathcal{N}(0, I)$.

If the transport map f has Jacobian in z that is lower-triangular, then q is easy to compute by change of variables.



Neural Posterior Estimation



Suppose that we simulate model outputs Y_i from $\theta_i \sim p(\theta)$.

Lemma. The posterior $p(\theta|Y)$ is the maximizer over q of

$$\mathbb{E}_{\theta \sim p(\theta), Y \sim p(Y|\theta)} \log q(\theta|Y)$$

NPE. Maximize estimates of this log-density over samples

$$\{(\theta_i, Y_i)\}_{i=1}^{N}$$

and return the posterior as the density q.

Part 3: Foundation Models

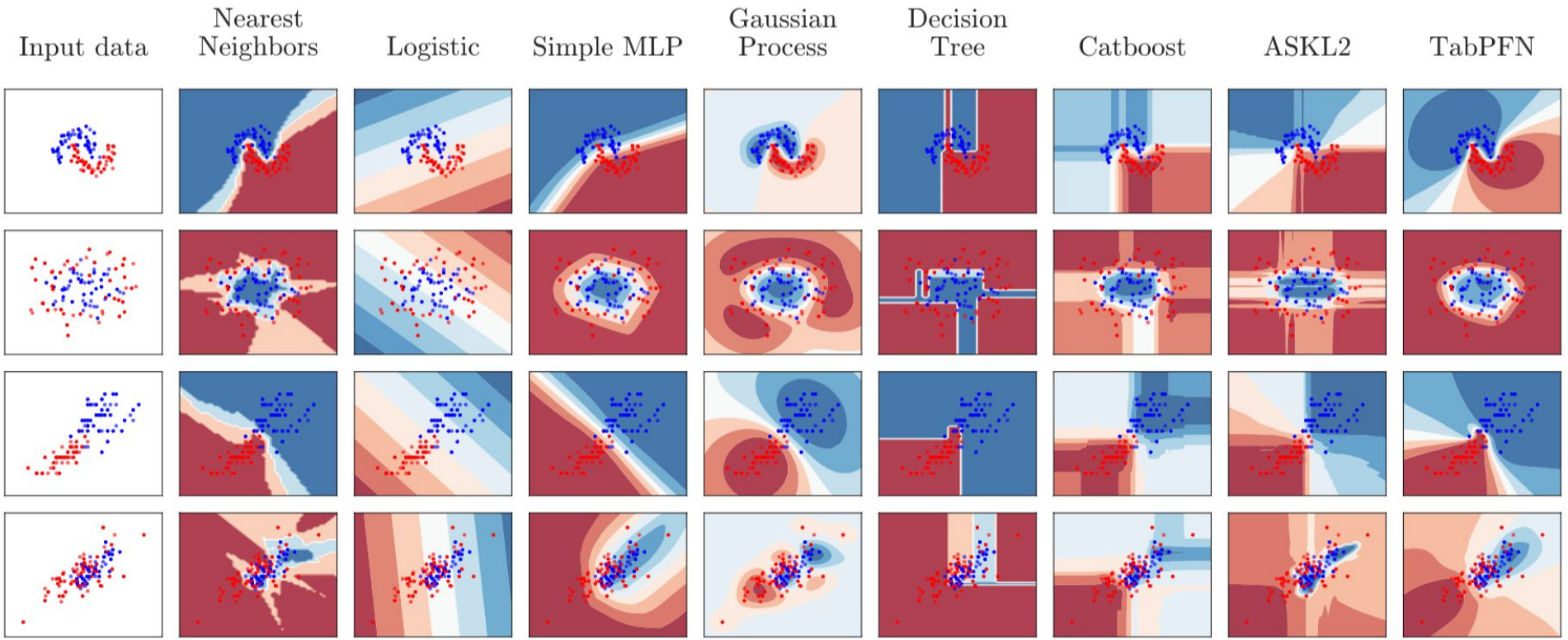
Tabular Prior-Data Fitted Networks (TabPFN)

Surprisingly, *we don't even need to train a model ourselves anymore for prediction.* This is called **zero-shot prediction**.

- Train a large transformer (neural network) model on an **extraordinary** amount of synthetic examples to achieve the following task:

$$(\{(X_i, Y_i)\}_{i=1}^{N}, X_{\text{obs}}) \xrightarrow{\text{foundation model}} \hat{p}(Y_{\text{obs}} | X_{\text{obs}}, X, Y)$$

- This can work **astonishingly well** in practice.



Hollmann, N., Müller, S., Eggensperger, K., & Hutter, F. (2022). TabPFN: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*.

Hoo, S. B., Müller, S., Salinas, D., & Hutter, F. (2024, December). The tabular foundation model TabPFN outperforms specialized time series forecasting models based on simple features. In *NeurIPS workshop on time series in the age of large models*.

NPE-PFN

Applying the same concept as TabPFN, but to posterior estimation instead.

$$\left(\{(\theta_i, Y_i)\}_{i=1}^{N}, Y_{\text{obs}}\right) \xrightarrow{\text{foundation model}} \hat{p}(\theta | Y_{\text{obs}})$$

This process requires virtually no effort at all outside of simulating the model.

Less trustworthy overall, but could give ***fantastic initial calibrations***.

Vetter, J., Gloeckler, M., Gedon, D., & Macke, J. H. (2026). Effortless, simulation-efficient Bayesian inference using tabular foundation models. *Advances in Neural Information Processing Systems*, *38*, 82685-82724.

Conclusion

Likelihood and likelihood-free approaches involving neural networks are *rapidly improving* in number and predictive capacity.

Please be aware of this space, especially the foundation model developments.

- A lot of existing approaches to efficiently fit a large variety of probabilistic models.
- Variational inference can deal with latents and noise very easily!
- Training-test splits give effective estimates of model accuracy *and inference*.
- Developed by computer scientists, so the packages are usually *very good*.