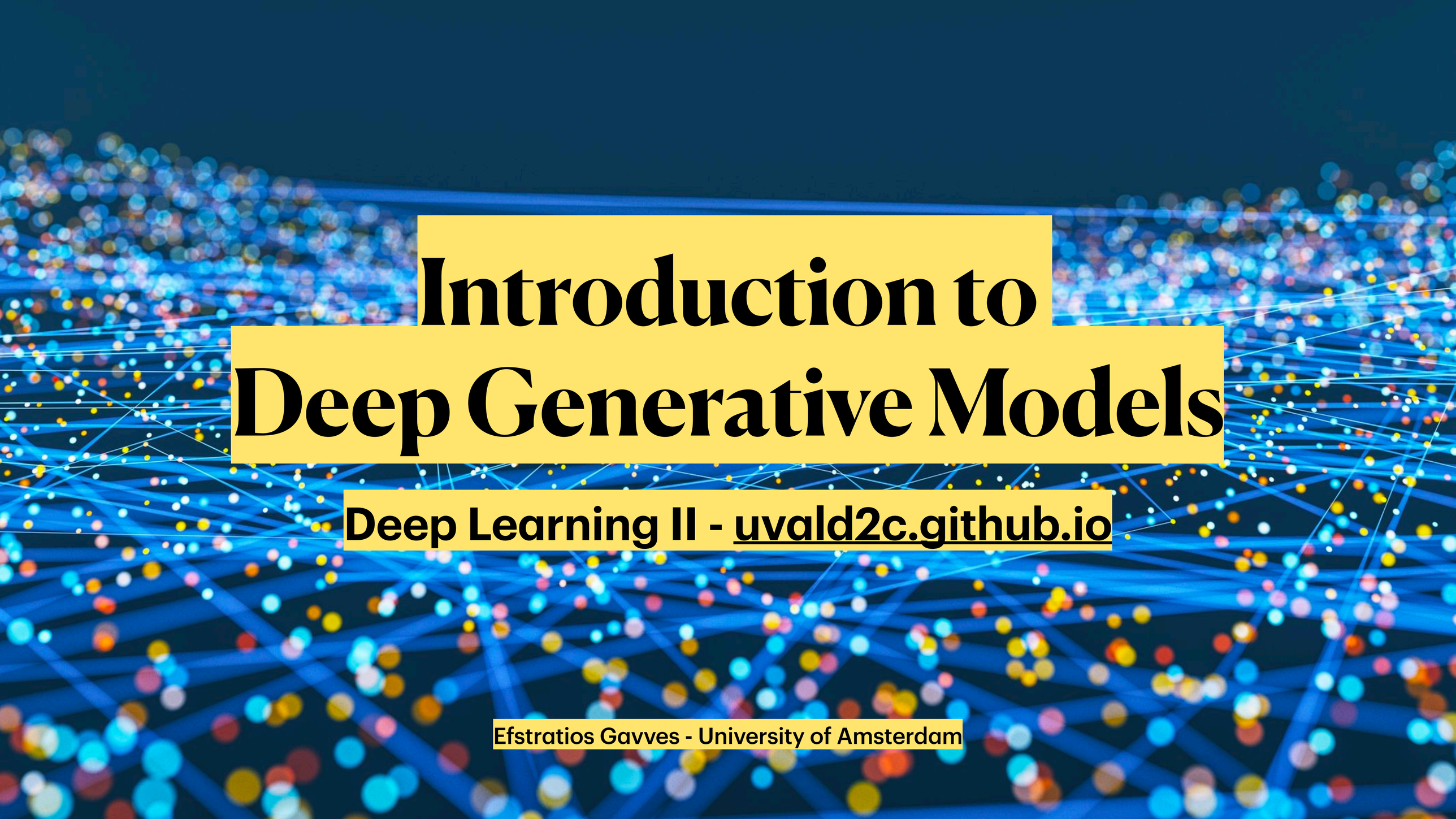


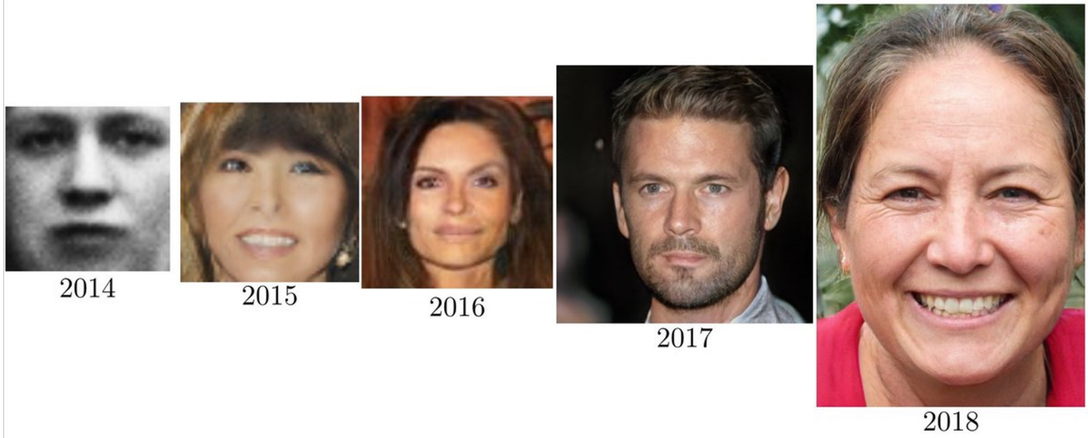
Introduction to Deep Generative Models

Deep Learning II - uvald2c.github.io

Efstratios Gavves - University of Amsterdam

Module overview

- Intro to generative models
- Deep Autoregressive Models
- Latent Variable Models
- Normalising Flows
- Energy-Based Models
- Score-based & Diffusion models



Lecture overview

- What are generative models
- Use and applications of generative models
- Step-by-step ‘minimum viable definition’ for generative models
- Overview of types of families of models

Disclaimer

- Generative models may sound hard hard
- I will try a ‘first principles approach’
- A first ‘principles approach’ will hopefully make most challenges feel natural
- Ask when things become too entangled
- Once you understand the challenges, you understand the intuition behind solutions

Generative models: Overview

- Can we have an algorithm ‘paint’ new bicycles based on few training examples?
- Rephrase: can we learn a function that returns **completely** new bicycles?
- Usually, for this function we use the probability density function $p(\mathbf{x})$
- We call them generative because once we know $p(\mathbf{x})$, we can sample from it

Training examples of bicycles




Newly generated bicycle

Discriminative models: Overview

- Basically, make predictions given input $p(y \mid \mathbf{x})$
  - Predict the correct class label → Classification
  - Predict the correct response → Regression
- Examples: ImageNet classification, translation, captioning, AlphaGo, ..

$p(y = \text{'bicycle'} \mid$  $)$

$p(\text{'IMDB Score'} \mid$ 

$)$

Generative vs Discriminative

- Generative models learn the joint distribution $p(\mathbf{x}, y) = p(y \mid \mathbf{x})p(\mathbf{x})$
- Discriminative models learn only learn the conditional $p(y \mid \mathbf{x})$
- Basically, we do not learn the (often much more complex) prior $p(\mathbf{x})$

Discriminative model

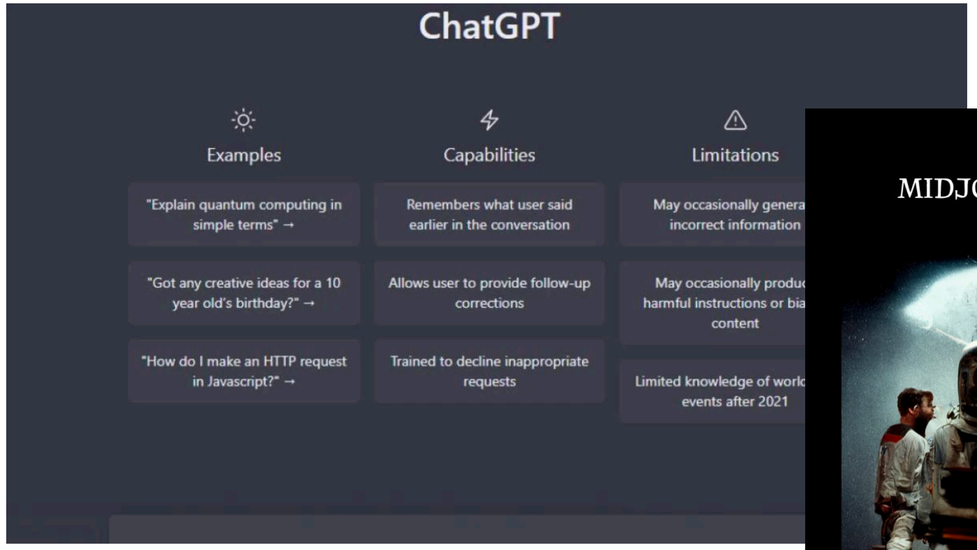
$p(y = \text{'bike'} \mid \mathbf{x})$

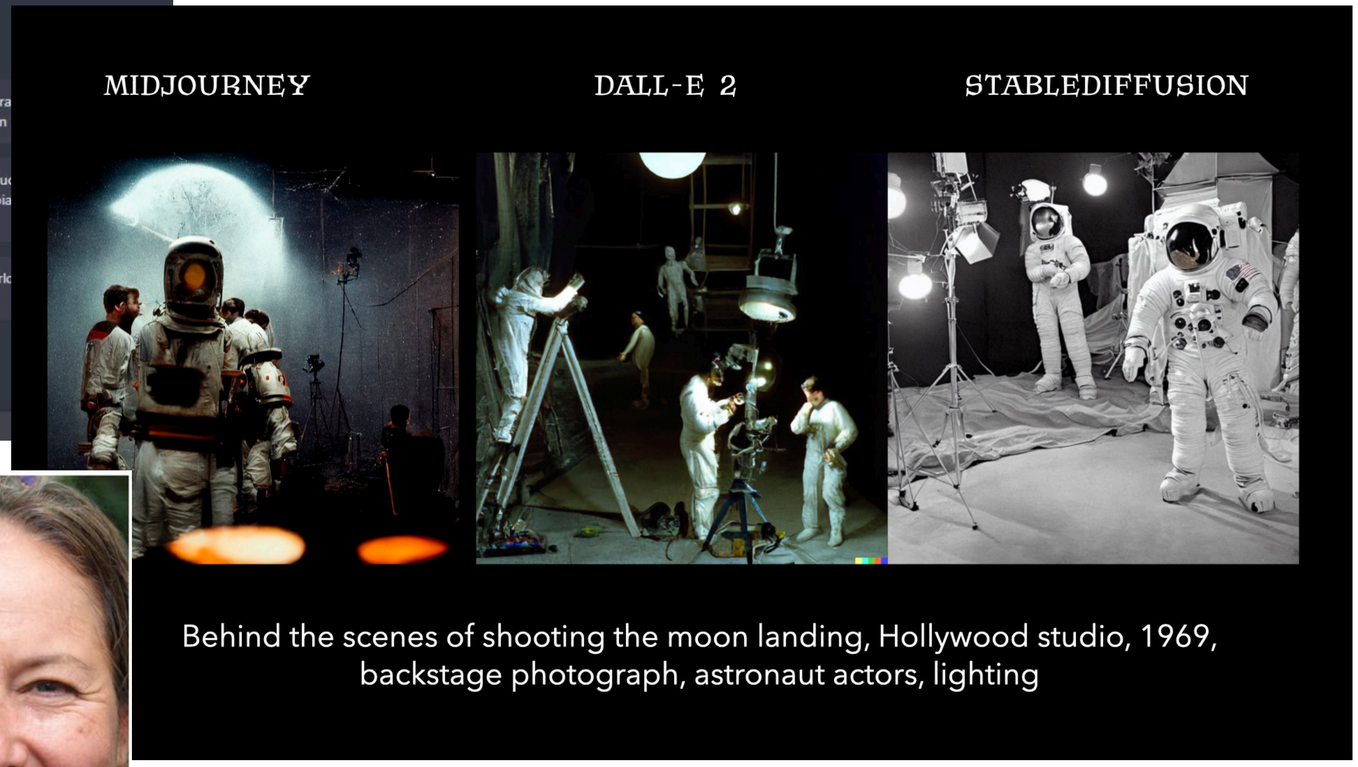
Generative model

$p(\mathbf{x}, y = \text{'bicycle'}) = 0.83$

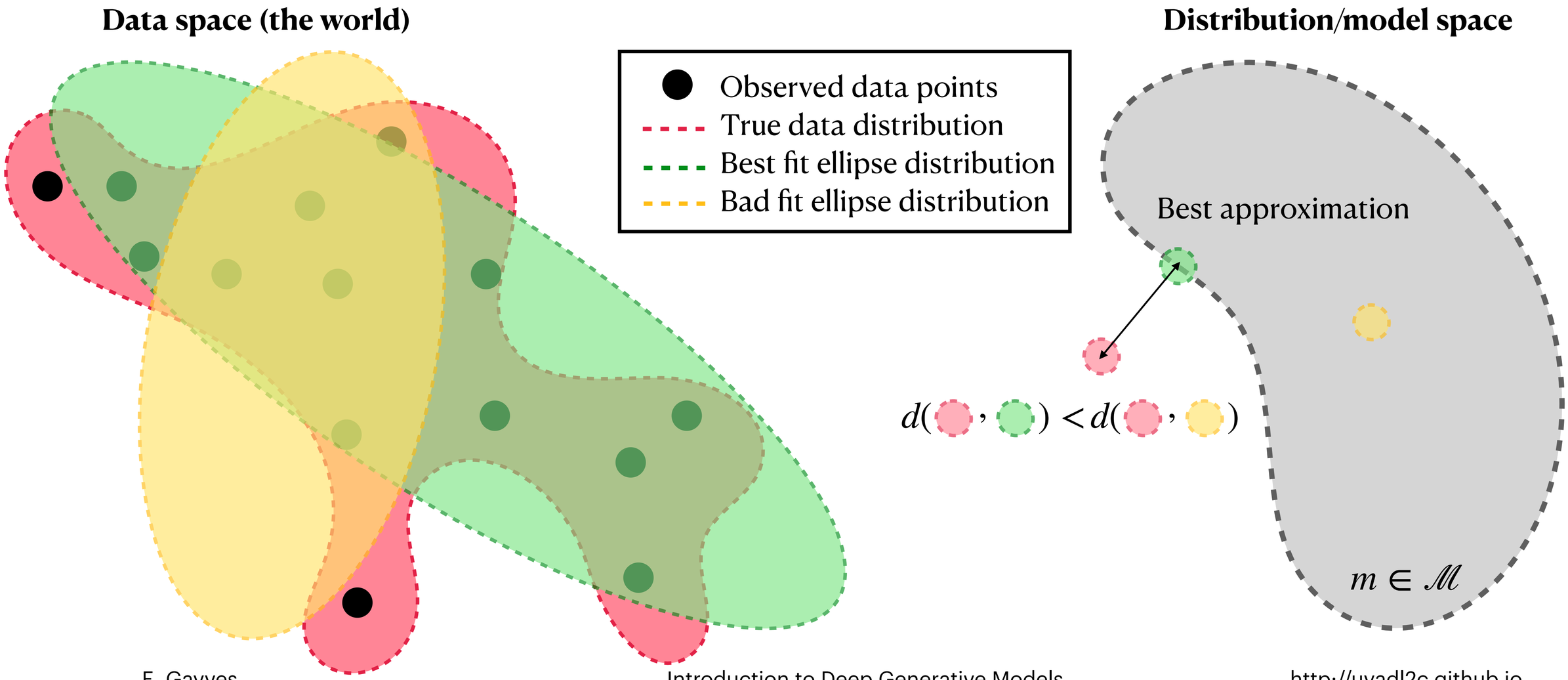
$p(\mathbf{x}, y = \text{'bicycle'}) = 0.01$

ChatGPT - Midjourney/DALLE - Stable Diffusion



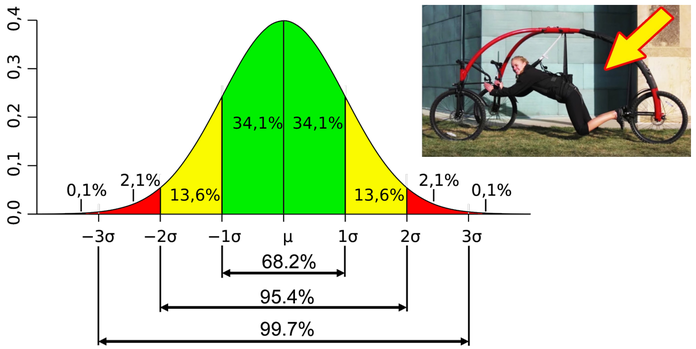


The world as a distribution: A duality



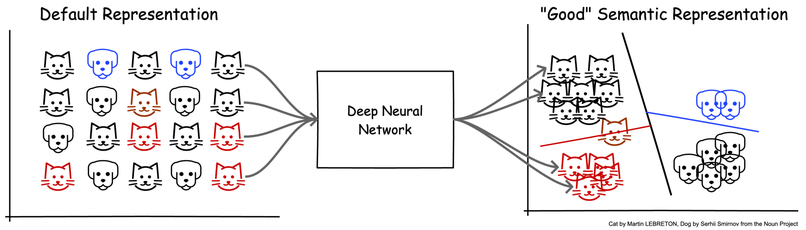
Generative models: what for?

Statistical inferences

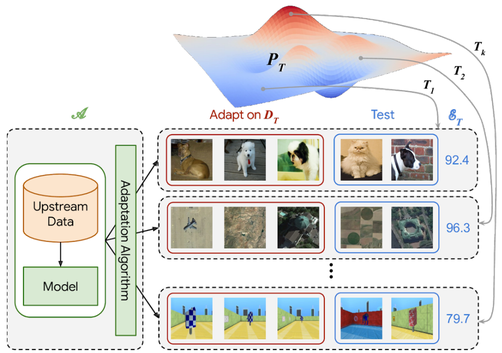


Sampling/Generation

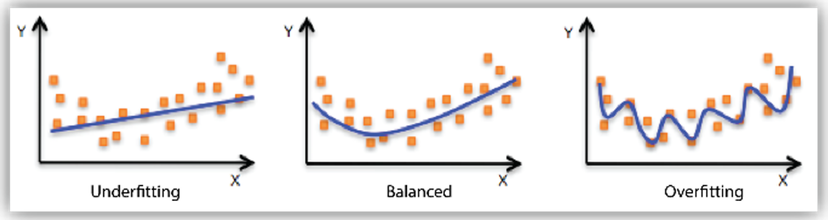
Representation Learning



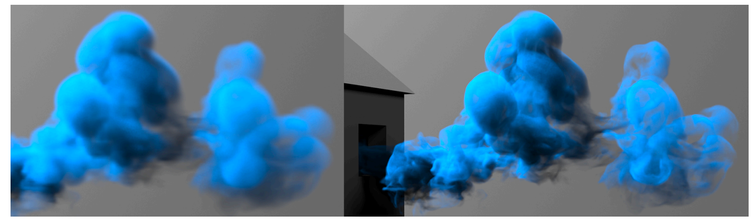
Pre-training



Generalisation



Simulations



Why not to learn distributions?

- *“One should solve the [classification] problem directly and never solve a more general [and harder] problem as an intermediate step.”* , V. Vapnik, father of SVMs.
- Learning how data looks like everywhere in the data space can be very complex and even intractable
- Do you want probabilities for inference? Want to learn in an unsupervised manner? Generalize better and avoid overfitting? Want to be able to generate new samples?

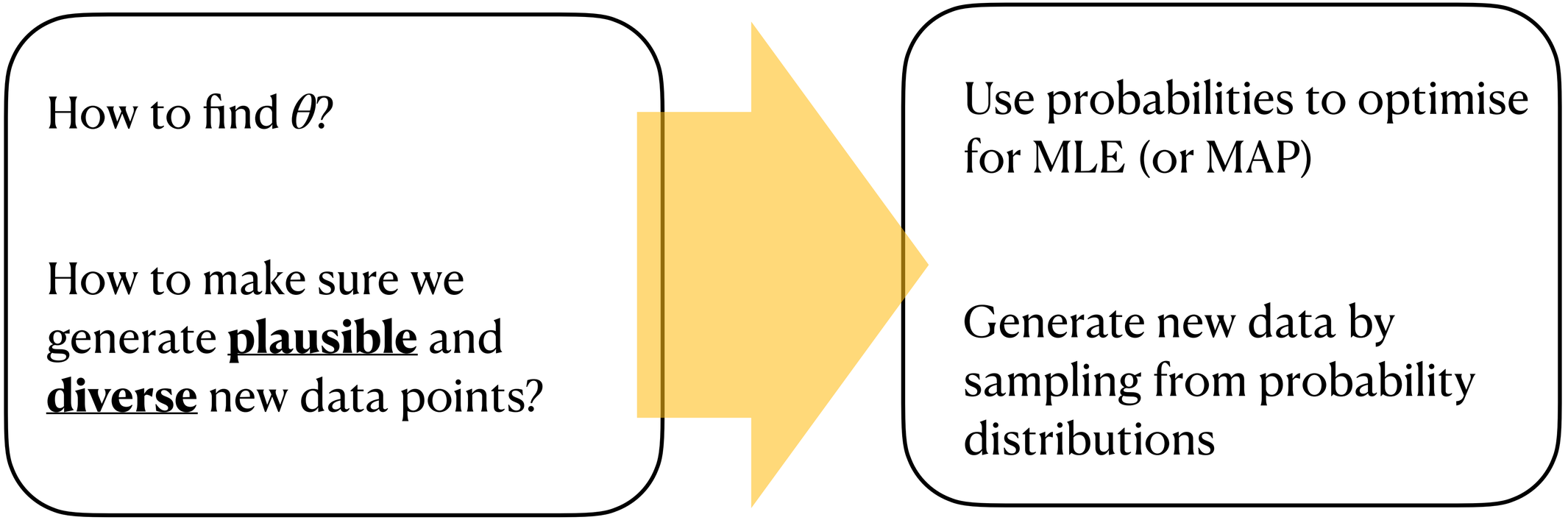
  → **Go generative**
- You have a specific prediction task, enough data, want to get the job done

  → **Go discriminative**, no need to make things too complex

Distributions

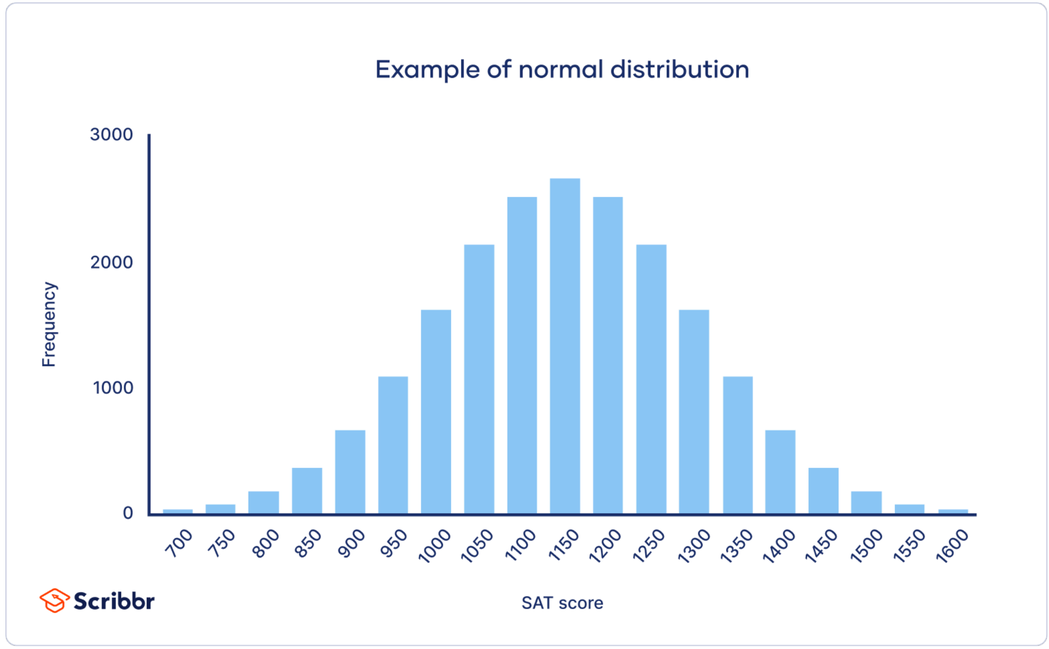
Why even consider probability distributions?

- Goal: find optimal parameters of a function f_θ that generates plausible new inputs

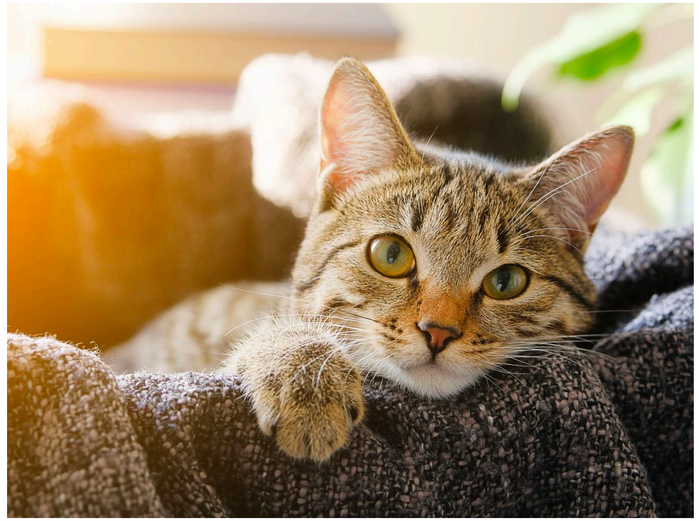


Non-canonical distributions

- For low-dimensional data (e.g. SAT scores) we can often find a good intuitive distribution fit, e.g., Gaussian
- Compute mean/variance → Sample!



- Are images (or other complex n-dimensional data) necessarily Gaussian?
- More flexibility to learn beyond non-canonical distributions is needed

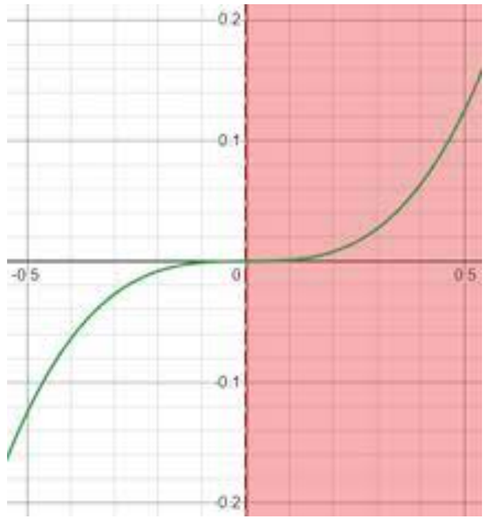


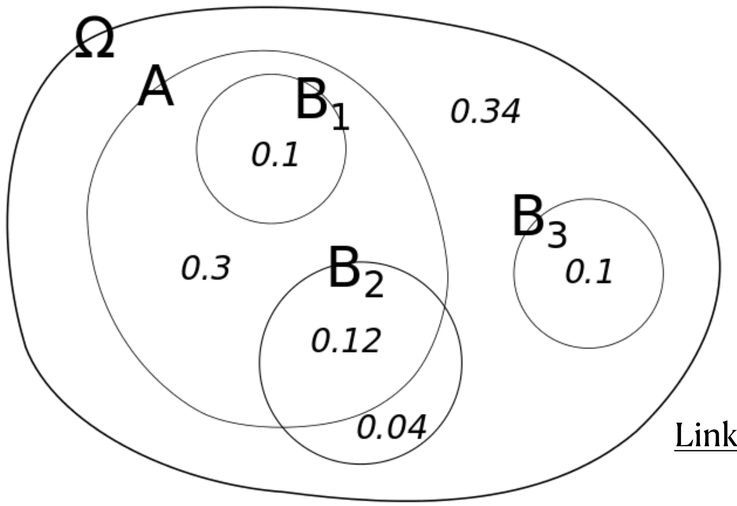
Beyond canonical distributions

Kolmogorov axioms

- Valid probabilities → Our function must satisfy the Kolmogorov axioms

$$1.\ P(X \in E) \geq 0, \forall E \in \mathscr{A} \qquad 2.\ P(\Omega) = 1 \qquad 3.\ P(X \in \cup_i E_i) = \sum_i P(X \in E_i)$$





Link to image

Satisfying Kolmogorov by normalisation

Setting an always positive function is easy

- Just pick a family of functions that are positive

$$f(x) = x^2$$

$$f(x) = \exp(x)$$

...

Satisfying unit measure and additivity by normalisation

- Just some up for all possible outcomes and divide

$$p(x) = \frac{f(x)}{Z}$$

$$Z = \sum_j f(x_j) \text{ or } Z = \int f(x)dx$$

Normalisation constant

- Computing $Z = \int f(x)dx$ can be very complex or even intractable

Non-trivial integrals

- Technically, you have a new integral per dimension
- Solutions cannot be analytical

Curse of dimensionality

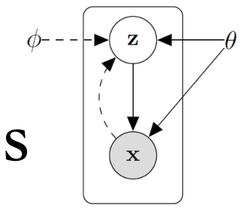
- Ok if you have 5 dimensions, but what about 1,000,000?

Find convenient ways to compute normalisation constant, like Boltzman or score-based models

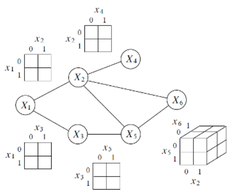
Or pick functions that satisfies all three axioms by construction, like autoregressive models, or flows, or latent variable models

Making intractable things tractable

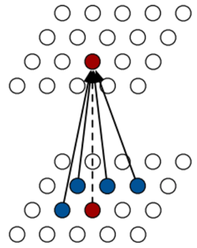
- Generative models are complex to solve because of intractabilities everywhere, either in the normalisation constant or familial quantities
  - ❖ In likelihood estimation, optimisation, sampling, inference
- Solve intractability by strong assumptions and inductive biases
  - VAE: tractable variational approximation
  - Autoregressive models: causal convolutions
  - Normalizing Flows: invertibility in layers



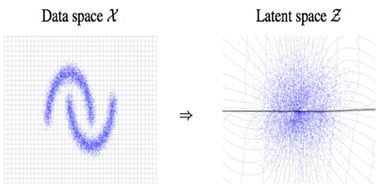
Bayesian networks (e.g., VAEs)



MRF



Autoregressive models

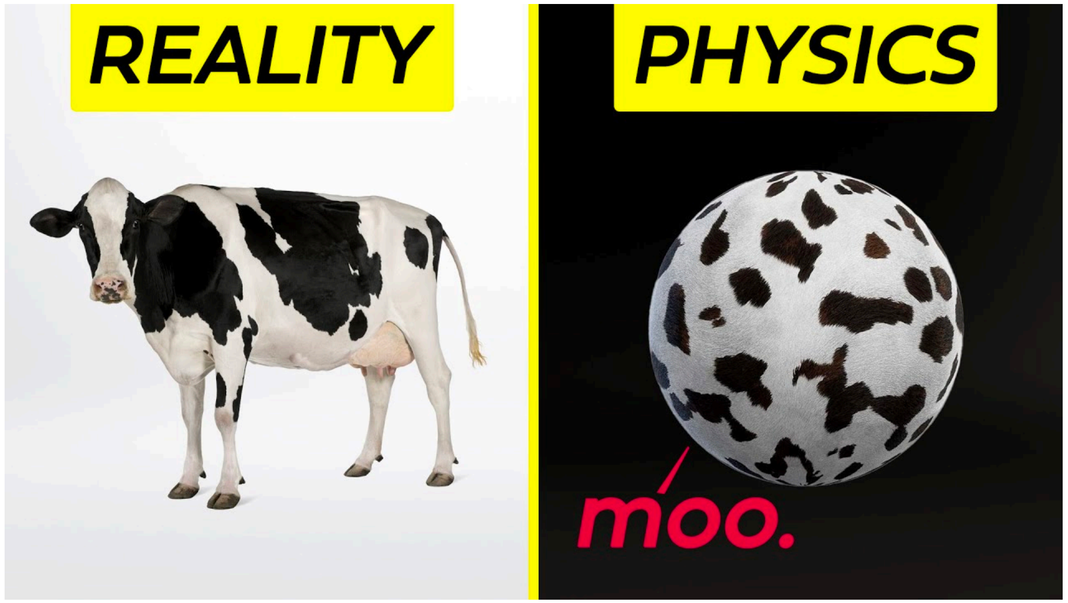


Flow models

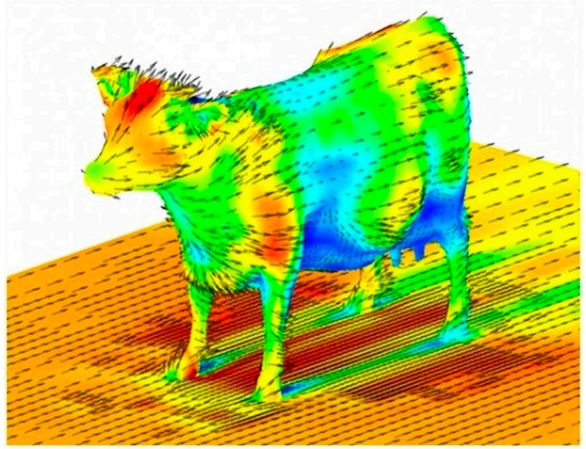
Tractability v. Flexibility

- Two opposing forces: tractability and flexibility
- Tractable models: usually analytically computable → easy to evaluate and fit
  - But usually not flexible enough to learn the true data structure
- Flexible models: fit arbitrary structures in data
  - But usually expensive to evaluate, fit, sample

Tractable but flexible?



Flexible but tractable?



Minimum Viable Definition

Step-by-Step

What we need for MVP: Steps 1-2

- We start from the formal definition of the generative model: $\mathbf{x} \sim p_\theta(h_\omega(\mathbf{x}))$
- Step 1: What type of family do we pick for p_θ?
  - Autoregressive? Energy-based? Score-based?
- Step 2: How do we implement the function h?
  - Deep networks are a good function to learn features, but what exactly?
  - Parse all the image, sequentially, return binary outputs (Bernoulli distribution), return discrete (Categorical distribution), make layers invertible, ...?

MVP for $\mathbf{x} \sim p_\theta(h_\omega(\mathbf{x}))$: Steps 3

- Step 3: How to learn optimal parameters θ?
  - What is our loss function? Usually depends on the model we pick from step 1, what we optimise (MLE or something else), the type of output, ...
  - An often desired objective is to minimise difference of learned distribution from the true one, although obviously we do not have access to the true one

$$\arg_\theta \min \mathscr{L}(p^*(\mathbf{x}), p_\theta(\mathbf{x}))$$

  - Finding ways around gives rise to families of models, *e.g.* latent variable models

MVP for $\mathbf{x} \sim p_\theta(h_\omega(\mathbf{x}))$: Steps 4-6

- Step 4: How to optimize?
  - We must pick an optimisation algorithm. Better start with simple solutions, then more and more complex
  - Usually, ADAM with standard hyperparameters is best
- Step 5: Start training
  - Minimize loss, monitor on held out validation set, keep training
- Step 6: optimise hyperparameters ω with grid search, bayesian optimisation, ...

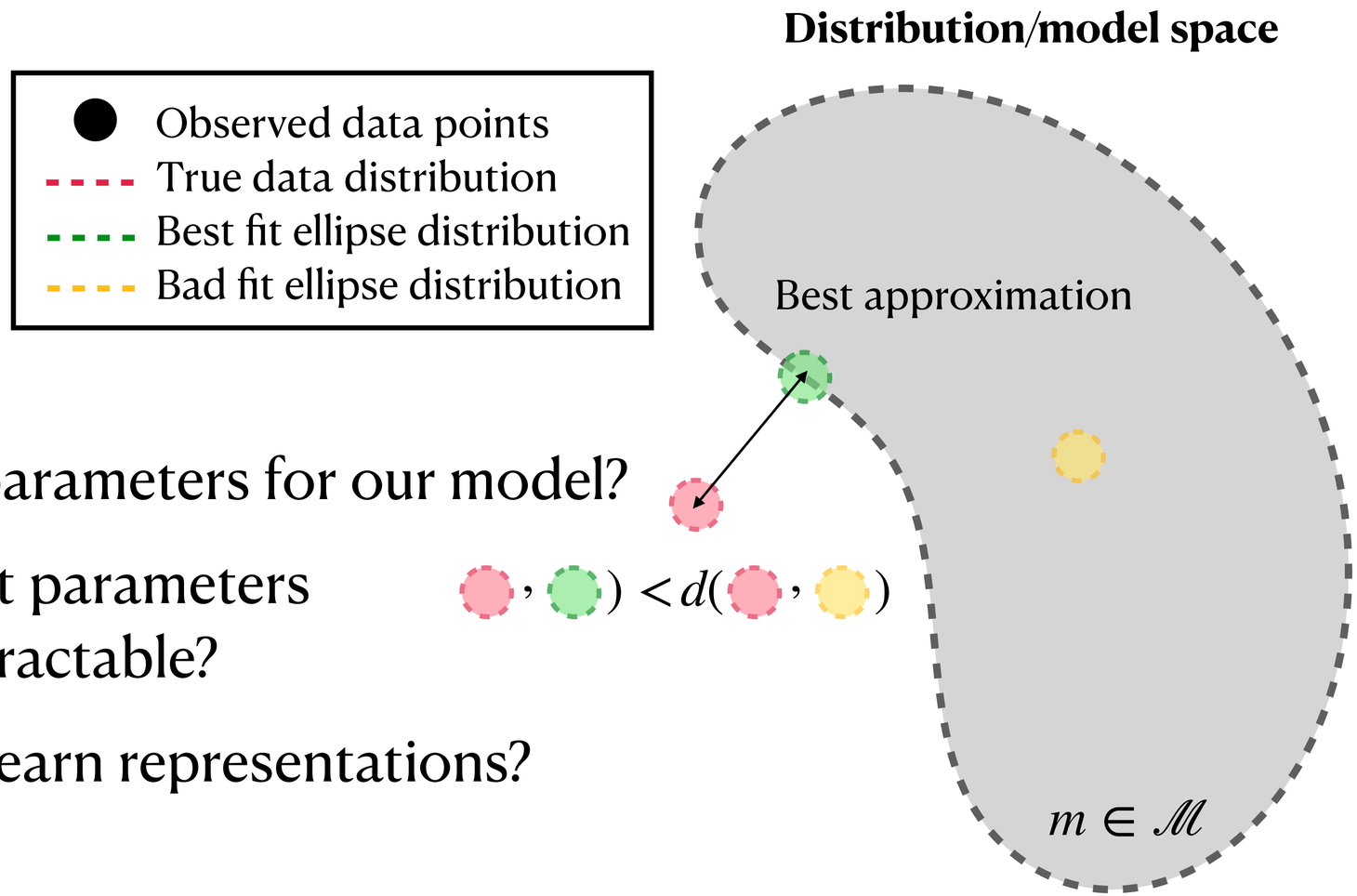
MVP for $\mathbf{x} \sim p_\theta(h_\omega(\mathbf{x}))$: Steps 6-7

- Step 6: Training is done. You can compute likelihoods for new inputs.
  - This is like the forward step
- Step 7: We can also sample $\sim$
  - Is our sampling efficient? E.g., autoregressive models are sequential → slow

- Per model we have more steps to check: how to do inference in latent variables, gradient computations *e.g.* with discrete variables, variance reduction, ...

Challenges for generative models

- How to make sure our model computations are tractable?
- How to find the right type of model for our data?
- How to optimise for the best parameters for our model?
- How to know what are the best parameters when things are technically intractable?
- How to sample, do inference, learn representations?

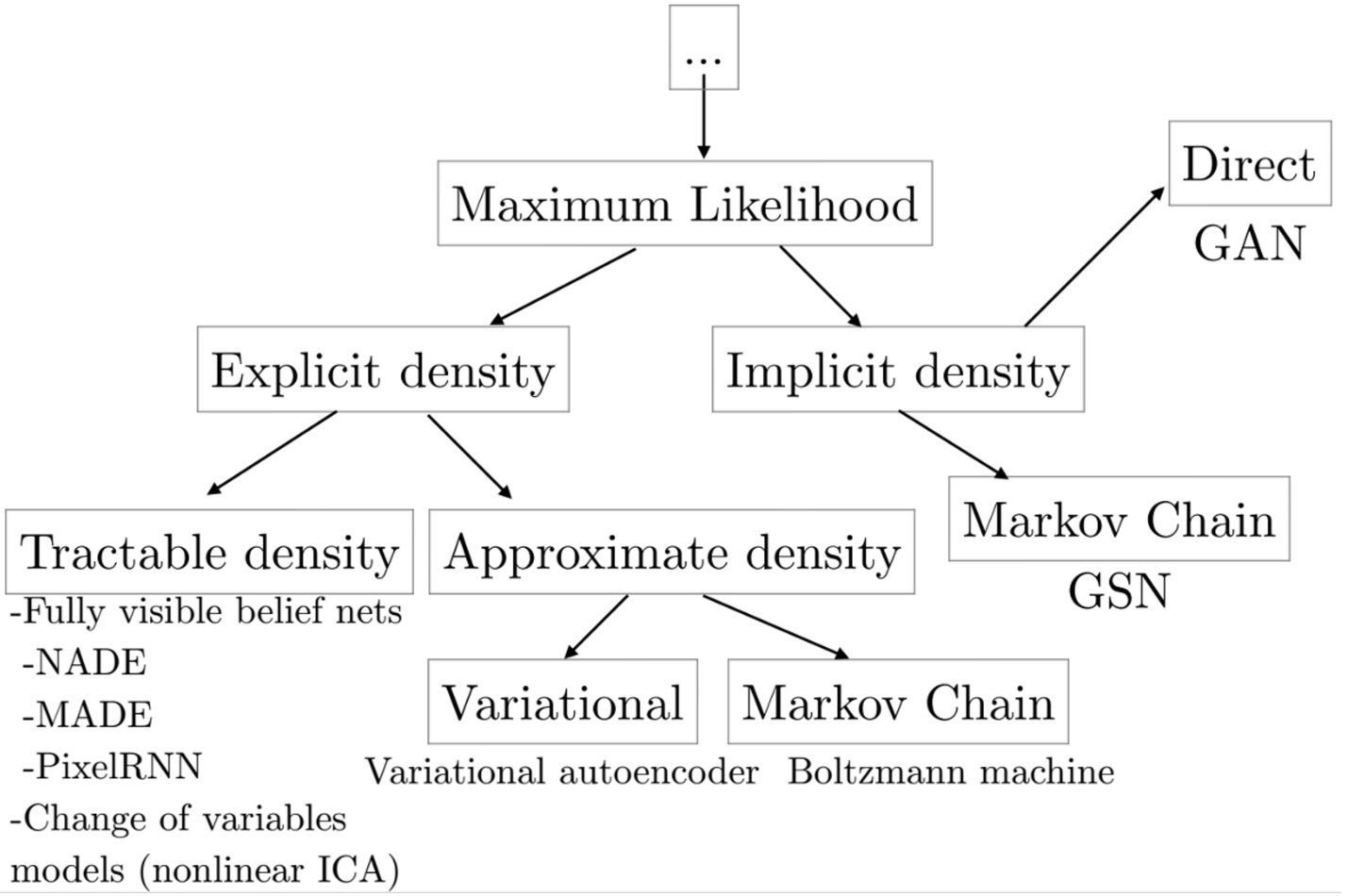


MVP for $\mathbf{x} \sim p_\theta(h_\omega(\mathbf{x}))$: Overview

- Forward and backward computations: how-to, complexity, efficiency
- Inference: how-to, complexity, efficiency
- Sampling: how-to, complexity, efficiency
- Learning objective and optimisation: $\arg_\theta \min \mathscr{L}(p^*(\mathbf{x}), p_\theta(\mathbf{x}))$
- Implementation of the neural network
- Do we learn representations? How to use them for downstream tasks?

Generative Model Families

A map of generative models



Generative model families: Pros and Cons

Table 1.1 A comparison of deep generative models

Generative models	Training	Likelihood	Sampling	Compression	Representation
Autoregressive models	Stable	Exact	Slow	Lossless	No
Flow-based models	Stable	Exact	Fast/slow	Lossless	Yes
Implicit models	Unstable	No	Fast	No	No
Prescribed models	Stable	Approximate	Fast	Lossy	Yes
Energy-based models	Stable	Unnormalized	Slow	Rather not	Yes

Book by J. Tomzchak

Wrapping up

What are generative models
Use and applications of generative models
Step-by-step ‘minimum viable definition’ for generative models
Overview of types of families of models

Further reading

- Stanford's course by S. Ermon
- Book by J. Tomzchak