

Title: Machine Learning Lecture

Speakers: Mohamed Hibat Allah

Collection: Machine Learning 2023/24

Date: April 23, 2024 - 11:30 AM

URL: <https://pirsa.org/24040052>

Lecture 10: Generative modeling

* Review

① Supervised learning

Gradient descent optimizers

Overfitting

Linear regression

Logistic regression

Feed-forward
Neural Networks

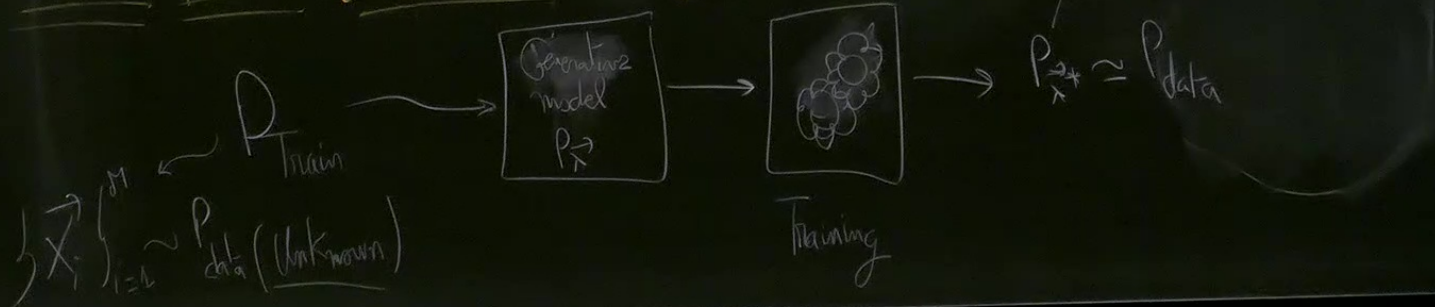
Convolutional
Neural Networks

② Unsupervised learning

PCA tSNE
Dimensional reduction

Generative modeling

* General picture of Generative modeling



Learning P_{data} \rightarrow Optimization

$$\text{Likelihood } L_{\vec{x}} = \prod_{i=1}^M P_{\vec{x}}(\vec{x}_i)$$

$-\log$

$$\text{Negative log likelihood} = -\frac{1}{M} \sum_{i=1}^M \log(P_{\vec{x}}(\vec{x}_i))$$

NLL

$$KL(P_{\text{data}} || P_{\vec{x}}) = \text{const} + \text{NLL}$$

\swarrow
KL divergence

Chat-GPT

M Explain generative modeling to five years-old kid



Generative modeling is like playing pretend with your friends. Imagine you have a box of toys, and you want to make a new toy that looks like the ones in the box. You can use your imagination to think of what the new toy might look like, and then try to make it with the materials you have.

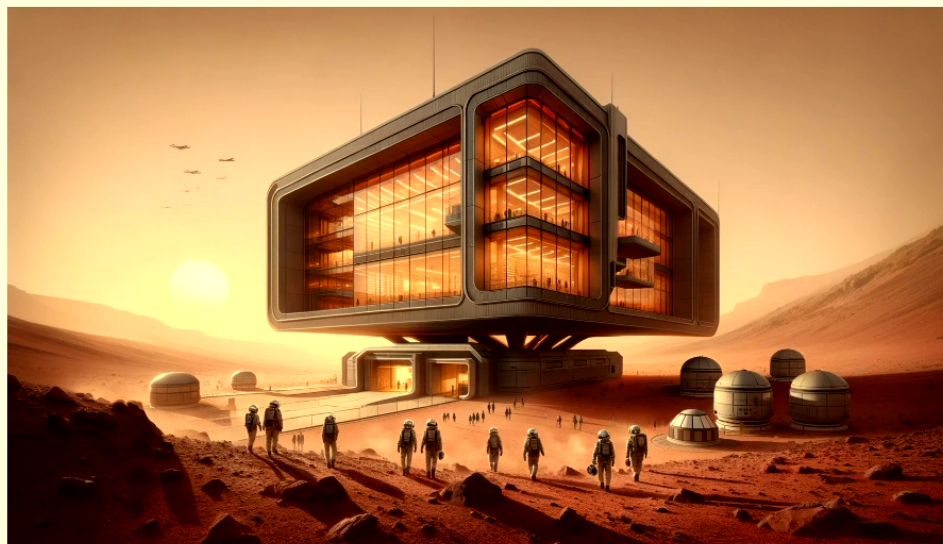
Generative modeling is kind of like that, but with a computer instead of toys. The computer tries to imagine new things based on what it has seen before, and then creates them. It's like the computer is using its imagination to make something new!

Dall-E

M You



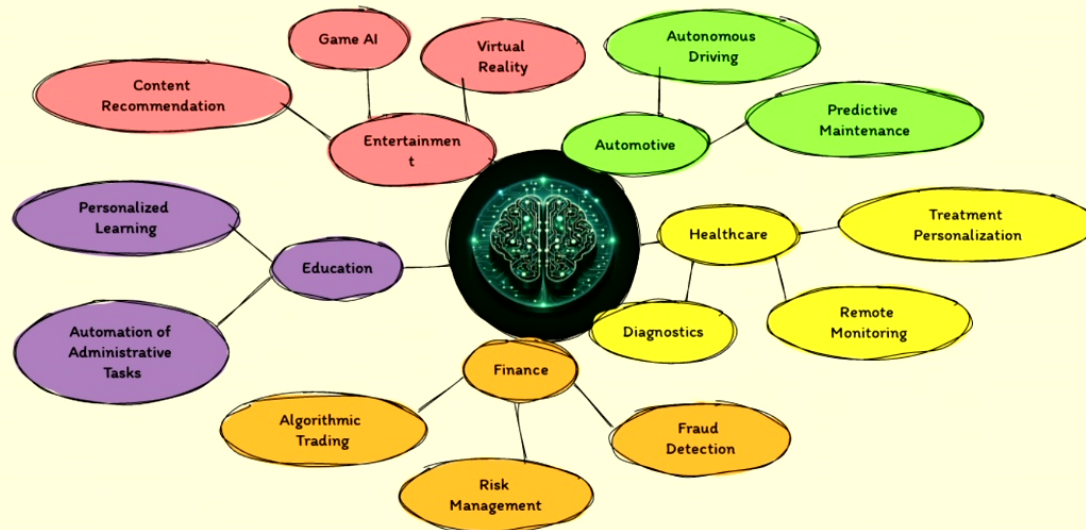
generate the picture of Perimeter Institute as if it was on Mars with examples of scientists entering the building



GPT-4

M You

🧠 Create a mindmap of AI application in real world + add an artificial brain icon in the center



Artificial Intelligence Virtual Artist



GPT-4 code copilot



You

How can I support GPU in my pytorch code?



Code Copilot

To support GPU usage in your PyTorch code, you need to ensure that your tensors and models are moved to the GPU. Here's a step-by-step guide to do that:

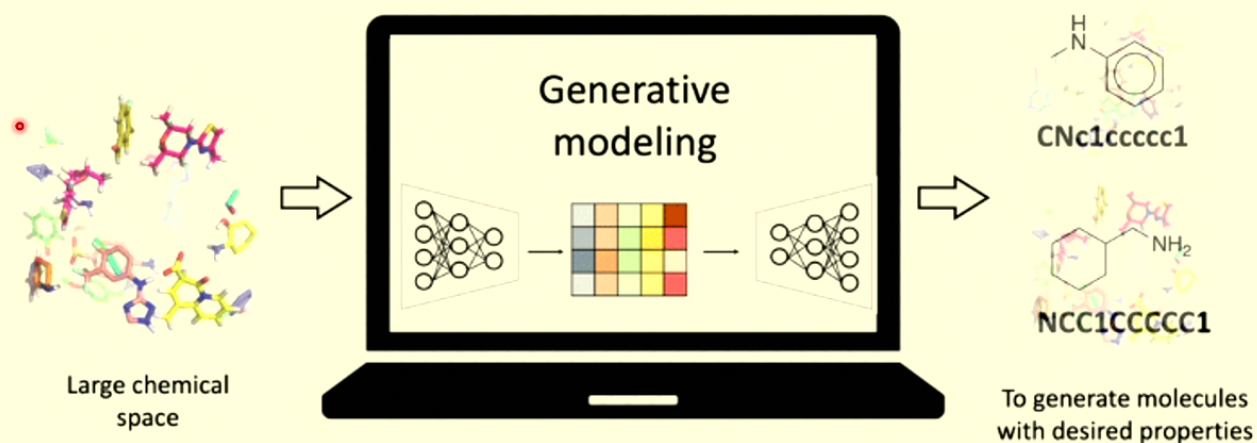
Step 1: Check GPU Availability

First, you should check if CUDA (NVIDIA's parallel computing platform and API) is available on your system. PyTorch uses CUDA to interact with the GPU.

Step 2: Transfer Tensors to GPU

To utilize the GPU, you need to transfer your data (tensors) and models to the GPU.

Drug discovery



Generative chemistry: drug discovery with deep learning generative models

Review | Published: 04 February 2021

Volume 27, article number 71, (2021) [Cite this article](#)

Cosmology

CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks

Mustafa Mustafa^{1*}, Deborah Bard¹, Wahid Bhimji¹, Zarija Lukić¹, Rami Al-Rfou² and Jan M. Kratochvil³

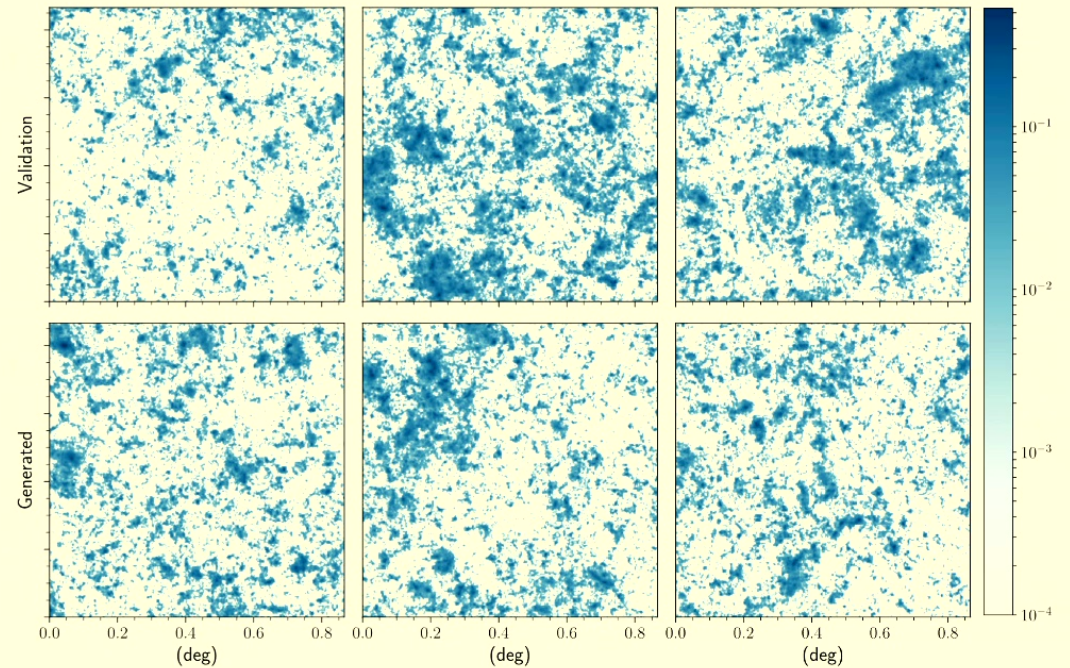


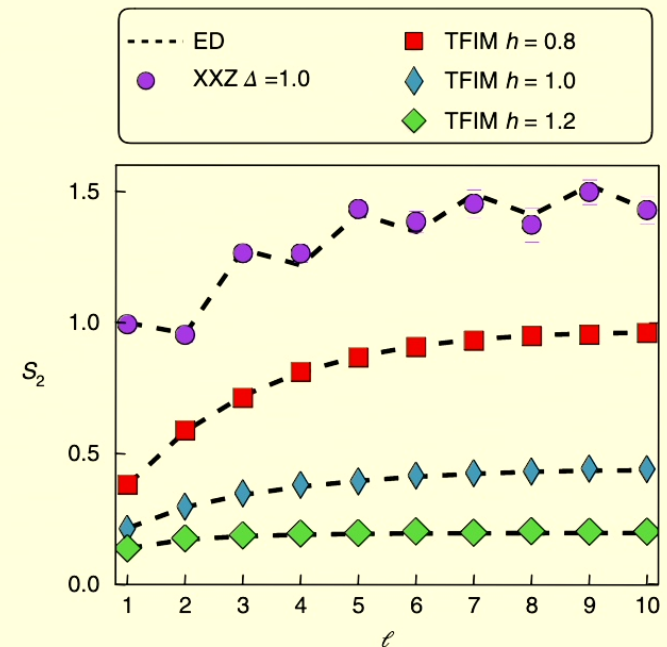
Figure 1 Weak lensing convergence maps for our Λ CDM cosmological model. Randomly selected maps from validation dataset (top) and GAN generated examples (bottom)

Quantum state tomography

Neural-network quantum state tomography

Giacomo Torlai^{1,2}, Guglielmo Mazzola³, Juan Carrasquilla^{4,5}, Matthias Troyer^{3,6}, Roger Melko^{1,2} and Giuseppe Carleo^{3,7*}

Starting from a **dataset in the z basis**, we can obtain **observables in x and y basis**, and also obtain challenging quantities such as the **entanglement entropy**.



Quantum state tomography

Attention-based quantum tomography

Peter Cha^{6,1}, Paul Ginsparg², Felix Wu², Juan Carrasquilla^{3,4}, Peter L McMahon⁵ and Eun-Ah Kim¹

Published 23 November 2021 • © 2021 The Author(s). Published by IOP Publishing Ltd

[Machine Learning: Science and Technology, Volume 3, Number 1](#)

Citation Peter Cha et al 2022 *Mach. Learn.: Sci. Technol.* 3 01LT01

DOI 10.1088/2632-2153/ac362b

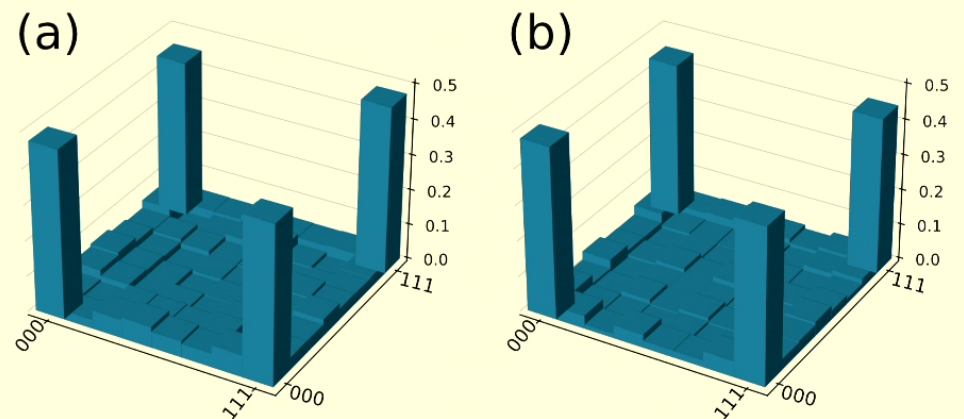
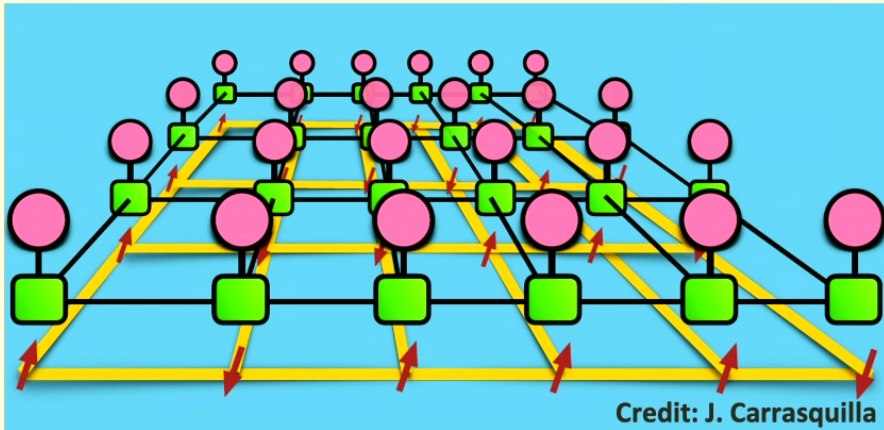


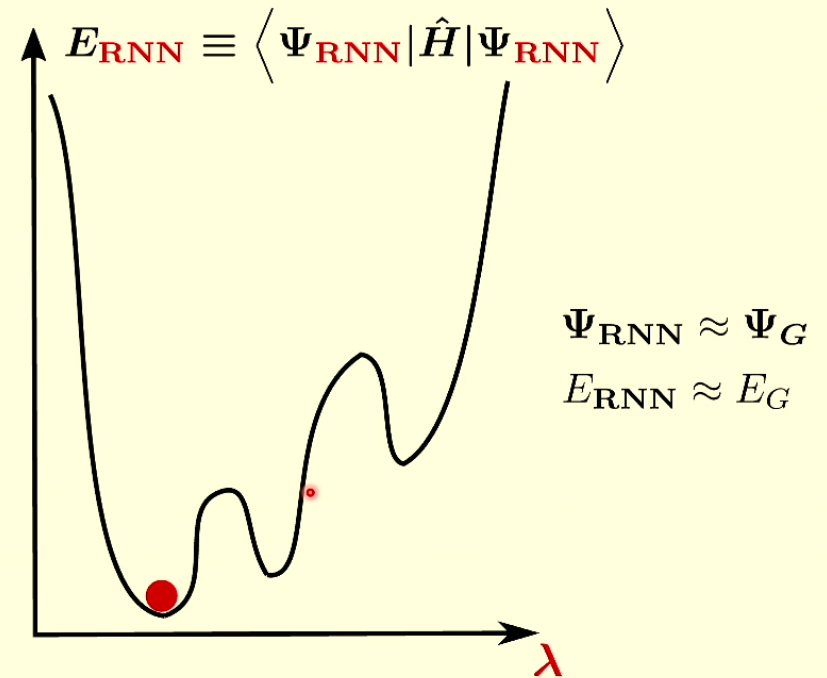
Figure 3. Benchmarking AQT (a) to MLE tomography offered by IBM's Qiskit library (b) for a noisy 3-qubit GHZ state data generated on the IBMQ_OURENSE quantum computer. Each bar represents the absolute value of a density matrix (DM) element.

Generative language modeling meets quantum physics

Ground states search using RNN wave functions



M.H. M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, Recurrent Neural Network Wave Functions, PRResearch, 2020.



Topic covered in the minicourse from May 21st to May 24th

*)
Outline: Classification of generative models

① Exact likelihood models

② Approximate likelihood models

③ Implicit likelihood models

J. Goodfellow NeurIPS 2016

① Exact Likelihood models

↳ Likelihood (or NLL) can be computed efficiently

1-1. Autoregressive models

↳ A prediction of a new variable is based on the previous variables.

$\vec{X} = (X_1, \dots, X_M)$ ← train
data (unknown)

Training

$$P_{\vec{X}}(\vec{X}) = P_X(X_1) \times P_X(X_2|X_1) \times P_X(X_3|X_2, X_1) \times \dots \times P_X(X_N|X_1, \dots, X_{N-1}) \text{ Chain rule of probability}$$

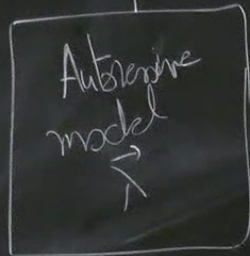
$$\vec{X} = (X_1, X_2, \dots, X_N)$$

\vec{X} = sentence of words. → word → dictionary size 10K-1M

= Pixels in a picture → Pixel → 256 x 3 (RGB)

= Configuration of spins → Spin → 2 ↑, ↓

$$\vec{Y} = P(\cdot | X_1, \dots, X_n) = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix}$$



X_i

Outline: Classification models

- ① Exact Likelihood
- ② Approximate Likelihood
- ③ Implicit Likelihood

J. Goodfellow Neural

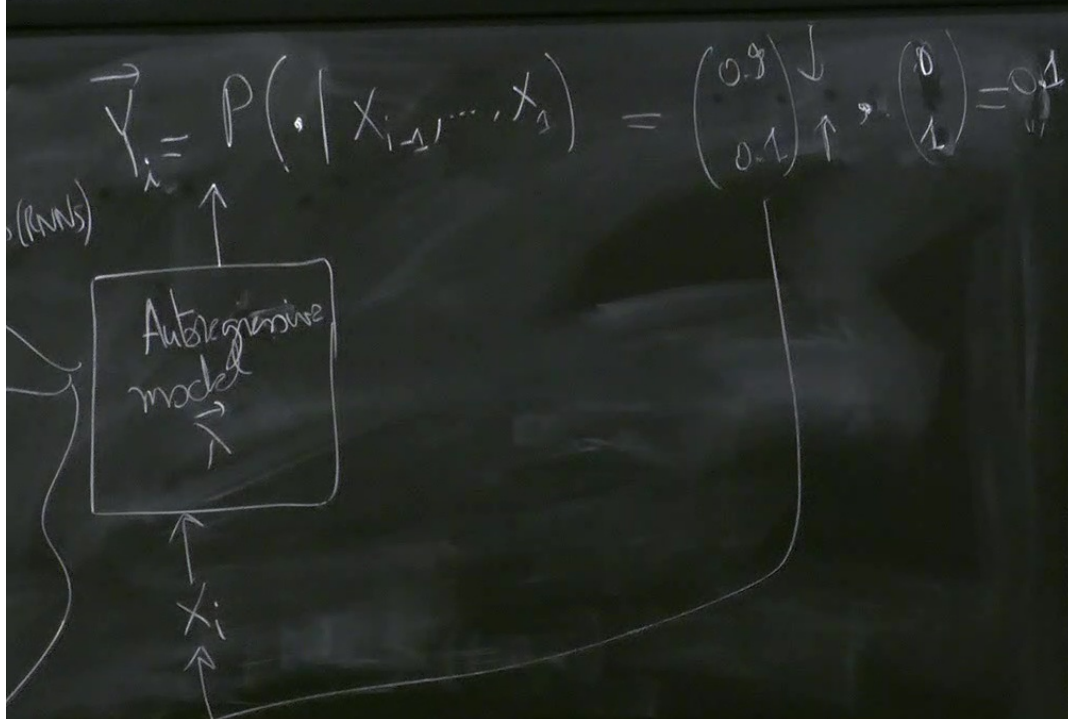
Recurrent Neural Networks (RNNs)

Transformers
(building of GPT)
Large language models
(LLMs)

MADE, MADE

$$\vec{y} = P(\cdot | x_{i-1}, \dots, x_1) = \begin{pmatrix} 0.9 \downarrow \\ 0.1 \uparrow \end{pmatrix}$$



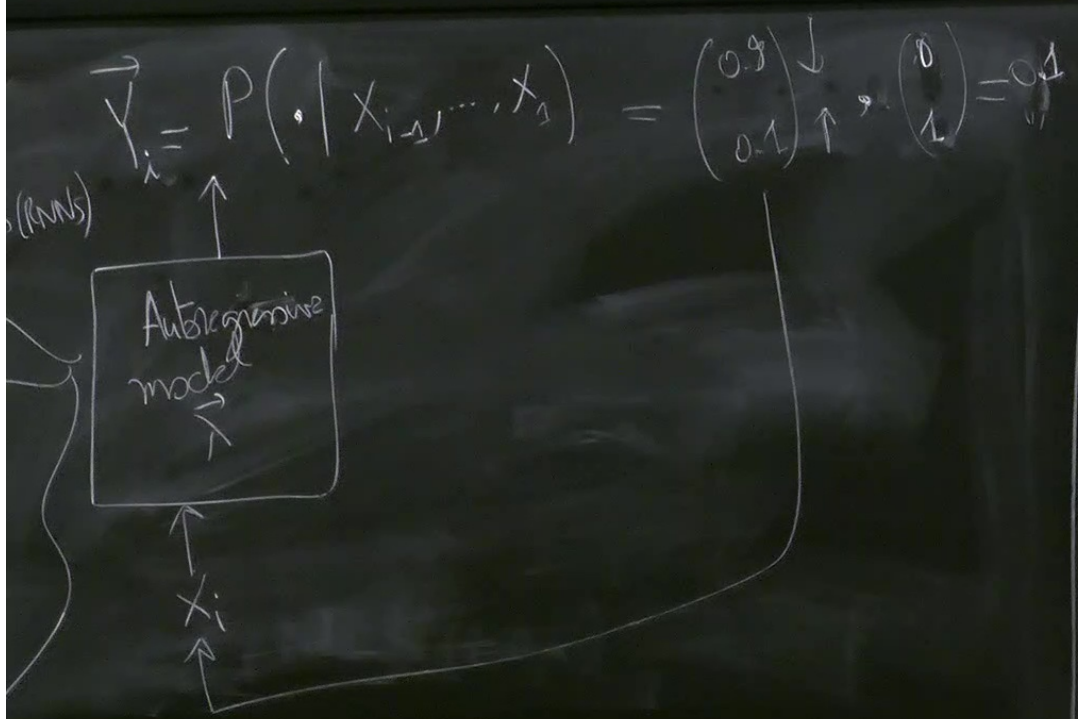


→ What if we want to compute $P_{\vec{Y}}(\vec{X})$?

Inference ← Known

one-hot encoding of X

$$P_{\vec{Y}}(X_i | X_{<i}) = \vec{Y}_i \cdot X_i$$



→ What if we want to compute $P_{\vec{Y}}(\vec{X})$?

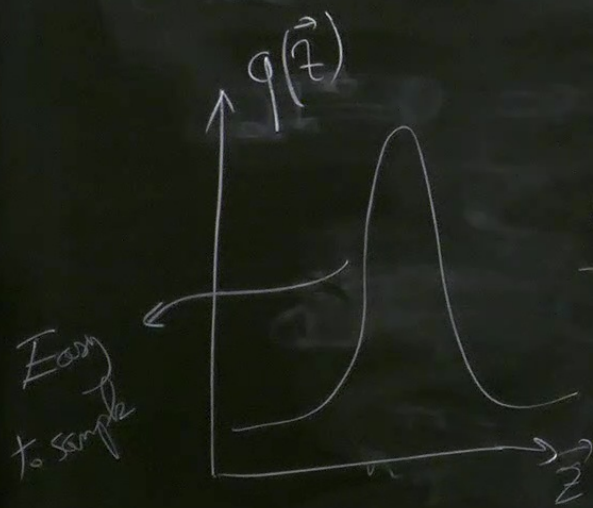
Inference ← Known

one-hot encoding of X

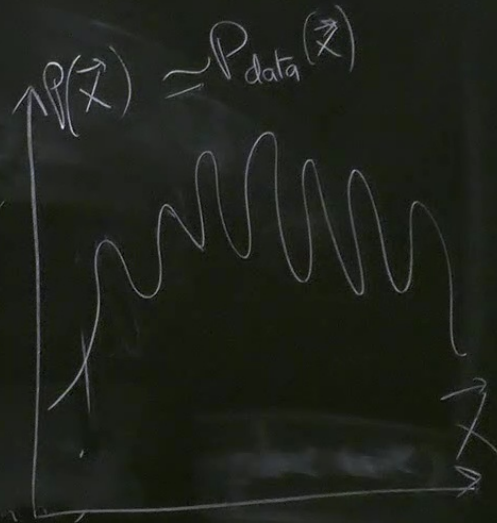
$$P_{\vec{Y}}(X_i | X_{<i}) = \vec{Y}_i \cdot X_i$$

→ Autoregressive models suitable to discrete data.

1.2 Normalizing flows → Suitable for continuous data.



Coordinate Transformation
 $\vec{x} = g(\vec{z})$



QuickTime Player Animation Normalizing Flow (Invertible Neural Network) - Florian Marquardt.mp4

Home Insert Draw Design Transitions Animations Slides

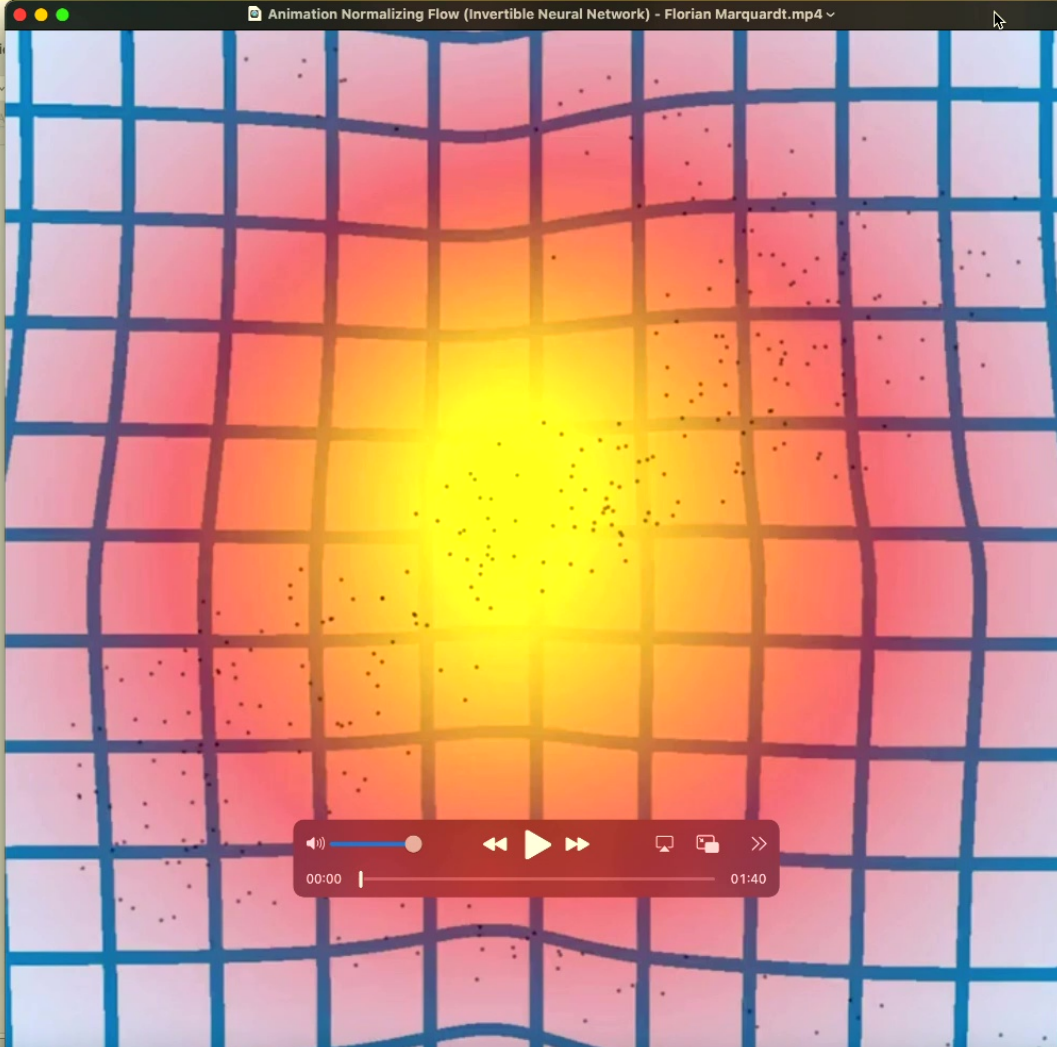
Paste Cut Copy Format New Slide Layout Reset Section Calibri

Slide 11 of 11 English (United States) Accessibility: Investigate

00:00 01:40

11

Comments 118%



$NN \approx \Psi G$
 $NN \approx E G$

QuickTime Player File Edit View Window Help

AutoSave Home Undo Redo

Home Insert Draw Design Transitions Animations Slides

Paste Cut Copy Format New Slide Layout Reset Section Calibri

2 3 4 5 6 7 8 9 10 11



Search (Cmd + Ctrl + U) Comments Share

and Share Slido PDF

$NN \approx \Psi G$
 $NN \approx E G$

11

Slide 11 of 11 English (United States) Accessibility: Investigate

Comments 118%



QuickTime Player File Edit View Window Help

AutoSave Home Undo Redo ...

Home Insert Draw Design Transitions Animations Slides

Paste Cut Copy Format New Slide Layout Reset Section Calibri B I U ...

Slide 11 of 11 English (United States) Accessibility: Investigate

Search (Cmd + Ctrl + U) Comments Share

Slido

$$NN \approx \Psi G$$
$$NN \approx E_G$$

11

Comments 118%

Firefox File Edit View History Bookmarks Tools Window Help

GAN Lab: Play with Gen X GAN Lab: Play with Gen X Visualizing Autoencode X ex4sperans/variational X LLM Visualization X RNN Explainer X The Illustrated Transformer X Tutorial 8: Deep Energy X Machine Learning for M X +

https://github.com/ex4sperans/variational-inference-with-normalizing-flows/tree/master?tab=readme-ov-file

Flow length: 4
Samples on iteration #9900

Flow length: 16
Samples on iteration #9900

https://github.com/ex4sperans/variational-inference-with-normalizing-flows/blob/master/assets/flow_4.png

The image displays a web browser window showing a GitHub repository page for 'variational-inference-with-normalizing-flows'. The main content area contains four plots arranged in a 2x2 grid. The top-left plot is a heatmap showing a ring-shaped distribution in a 2D space. The top-right plot is a scatter plot of blue points forming a ring. The bottom-left plot is a scatter plot of blue points forming a ring, labeled 'Flow length: 4 Samples on iteration #9900'. The bottom-right plot is a scatter plot of blue points forming a ring, labeled 'Flow length: 16 Samples on iteration #9900'. The browser's address bar shows the URL 'https://github.com/ex4sperans/variational-inference-with-normalizing-flows/tree/master?tab=readme-ov-file'. The browser's taskbar at the bottom shows various application icons.

In 1D:

$$\int q(z) dz = 1$$

$$z = g^{-1}(x)$$

$$\int q(g^{-1}(x)) \left| \frac{dz}{dx} \right| dx = 1$$

P(x)

$$P(\vec{x}) = q(g^{-1}(\vec{x})) \left| \det \left(\frac{\partial \vec{z}}{\partial \vec{x}} \right) \right|$$

Jacobian

$$\frac{\partial g^{-1}(\vec{x})}{\partial \vec{x}} = \begin{pmatrix} * & & (0) \\ * & * & \\ (* & & * \\ & & \dots \\ & & * \end{pmatrix}$$

Example:

$$\begin{cases} x_1 = z_1 \\ x_2 = z_2 \exp(\dots) \end{cases}$$

Jacobian

Example:

$$g \begin{cases} x_1 = z_1 \\ x_2 = z_2 \exp(s(z_1)) + t(z_1) \end{cases}$$

$$g^{-1} \begin{cases} z_1 = x_1 \\ z_2 = (x_2 - t(x_1)) \exp(-s(x_1)) \end{cases}$$



$$\exp(s(z_1)) + t(z_1)$$

$$(x_2 - t(x_1)) \exp(-s(x_1))$$

$$J = \frac{\partial \vec{z}}{\partial \vec{x}} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 & \\ * & \exp(-s(x_1)) \end{pmatrix}$$

$$J = J_1 \times J_2 \times \dots \times J_{N_f}$$

$$\det(J) = \det(J_1) \times \dots \times \det(J_{N_f})$$

Hyperparameter

$$f(z_1) + h(z_1)$$

$$f(x_1) \exp(-s(x_1))$$

$$J_1 = \frac{\partial z}{\partial x} = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \frac{\partial z_1}{\partial x_2} \\ \frac{\partial z_2}{\partial x_1} & \frac{\partial z_2}{\partial x_2} \\ 0 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ * & \exp(-s(x_1)) \\ 0 & 0 \end{pmatrix}$$

← learnable

② Approximate likelihood models

$Z_{\vec{x}}$ cannot be computed efficiently

2.1 Energy-based models

↔ 80's Hopfield networks

$$P_{\vec{x}}(\vec{x}) = \frac{\exp(-E_{\vec{x}}(\vec{x}))}{Z_{\vec{x}}}$$

$$E_{\vec{x}} = \sum_{i,j} w_{ij} x_i x_j - \sum_i b_i x_i$$

② Approximate likelihood models

$Z_{\vec{x}}$ cannot be computed efficiently

2.1 Energy-based models

← 80's Hopfield networks

$$P_{\text{data}}(\vec{x}) \approx P_{\vec{x}}(\vec{x}) = \frac{\exp(-E_{\vec{x}}(\vec{x}))}{Z_{\vec{x}}}$$

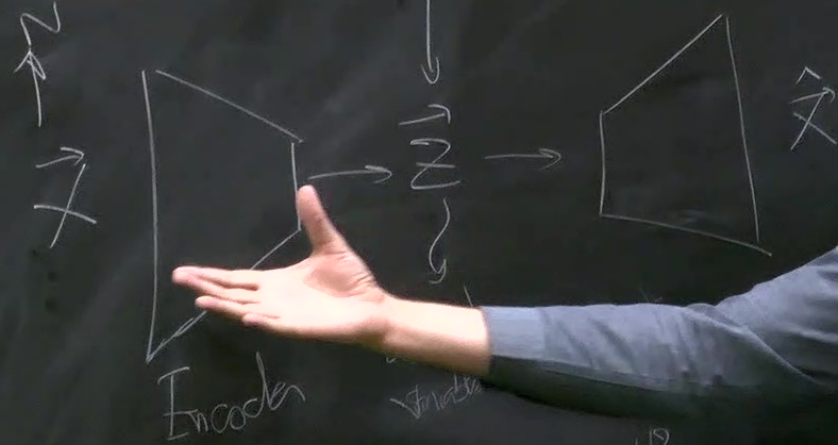
$$E_{\vec{x}} = \sum_{i,j} w_{ij} x_i x_j - \sum_i b_i x_i$$

2N ← $Z_{\vec{x}} = \sum_{\vec{x}} \exp(-E_{\vec{x}}(\vec{x}))$

$$\frac{P_{\vec{x}}(\vec{x}_2)}{P_{\vec{x}}(\vec{x}_1)} = \frac{\exp(-E_{\vec{x}}(\vec{x}_2))}{\exp(-E_{\vec{x}}(\vec{x}_1))} \rightarrow \text{Metropolis sampling}$$

$b_i x_i$

2.9 Autoencoders



$$L = \frac{1}{2} \|X - \hat{X}\|^2$$

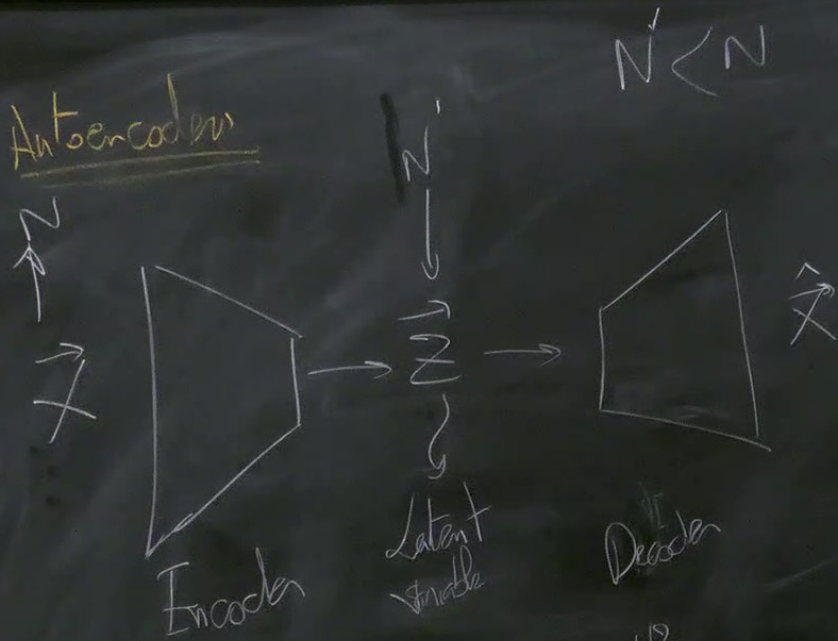
→ What if we want to

Inference

$$P(x_i | x_{-i}) =$$

some models

2.2 Autoencoder



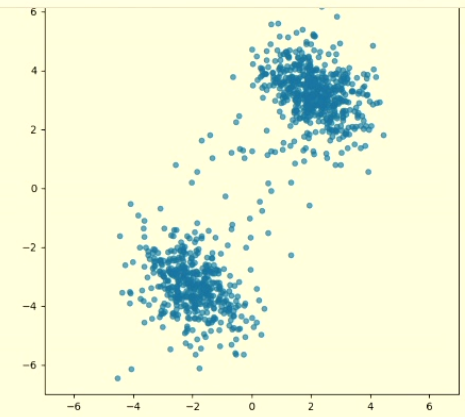
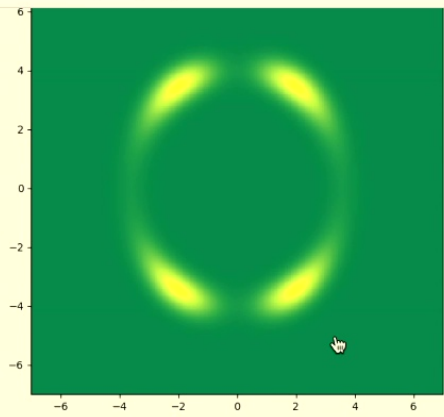
$$L = \| \hat{X} - X \|^2$$

→ What if we want to

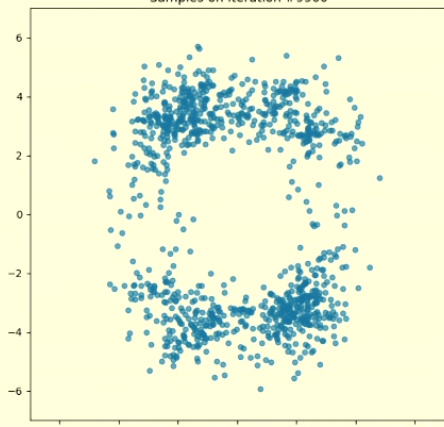
Inference

$$P_X(x_i | x_{<i}) =$$

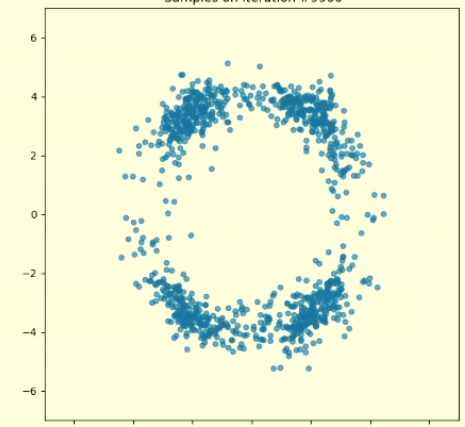
→ Autoencoder models



Flow length: 4
Samples on iteration #9900



Flow length: 16
Samples on iteration #9900



https://github.com/ex4sperans/variational-inference-with-normalizing-flows/blob/master/assets/density.png



input vector to a lower dimensionality, then project that low-dimensional vector back to the input vector shape in such a way as to produce an output vector that closely resembles the input vector. One can see a visual diagram of the autoencoder model architecture—and see how the autoencoder’s projections improve with training—by interacting with the figure below:



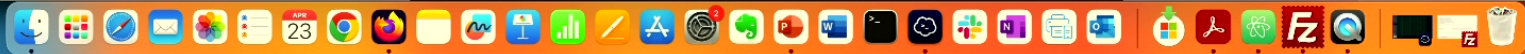
The figure above shows the model architecture of an autoencoder. Click the "Train" button to improve the autoencoder's reconstruction of input images, and click the "Sample" button to show how the model reconstructs a new sample image.

Having discussed the purpose and basic components of an autoencoder, let's now discuss how to create autoencoders using the Keras framework in Python.

Building Autoencoders with Keras

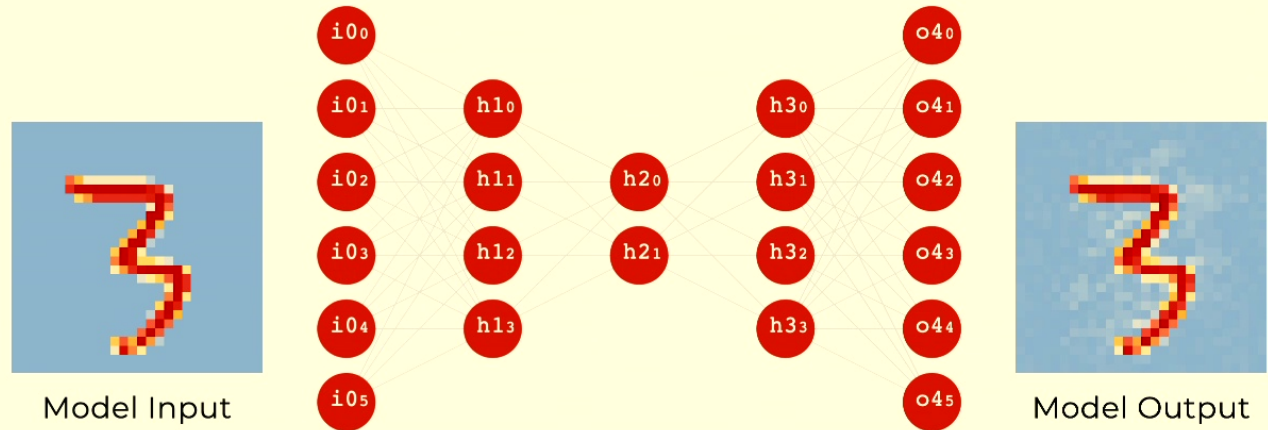
To build a custom autoencoder with the Keras framework, we'll want to start by collecting the data on which the model will be trained. To keep things simple and moderately interesting, we'll use a collection of images of celebrity faces known as the **CelebA dataset**. One can download and prepare to analyze 20,000 images from this dataset with the following:

```
from keras.preprocessing.image import load_img, img_to_array
```



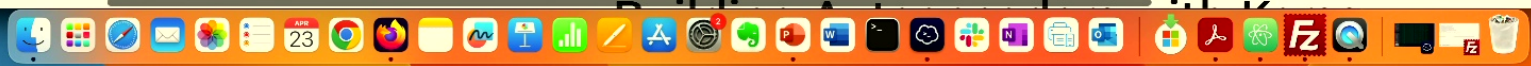
Train! Sample!

Epochs: 800
Loss: 0.00968



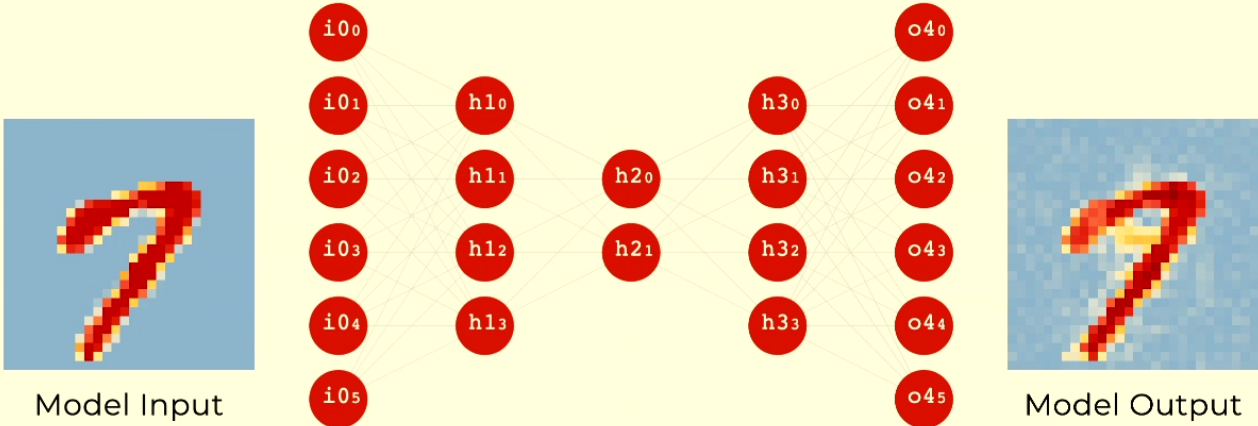
The figure above shows the model architecture of an autoencoder. Click the "Train" button to improve the autoencoder's reconstruction of input images, and click the "Sample" button to show how the model reconstructs a new sample image.

Having discussed the purpose and basic components of an autoencoder, let's now discuss how to create autoencoders using the Keras framework in Python.



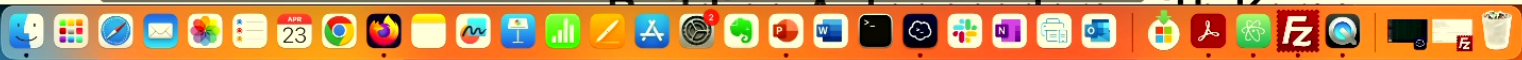
Train! Sample!

Epochs: 1400
Loss: 0.00414

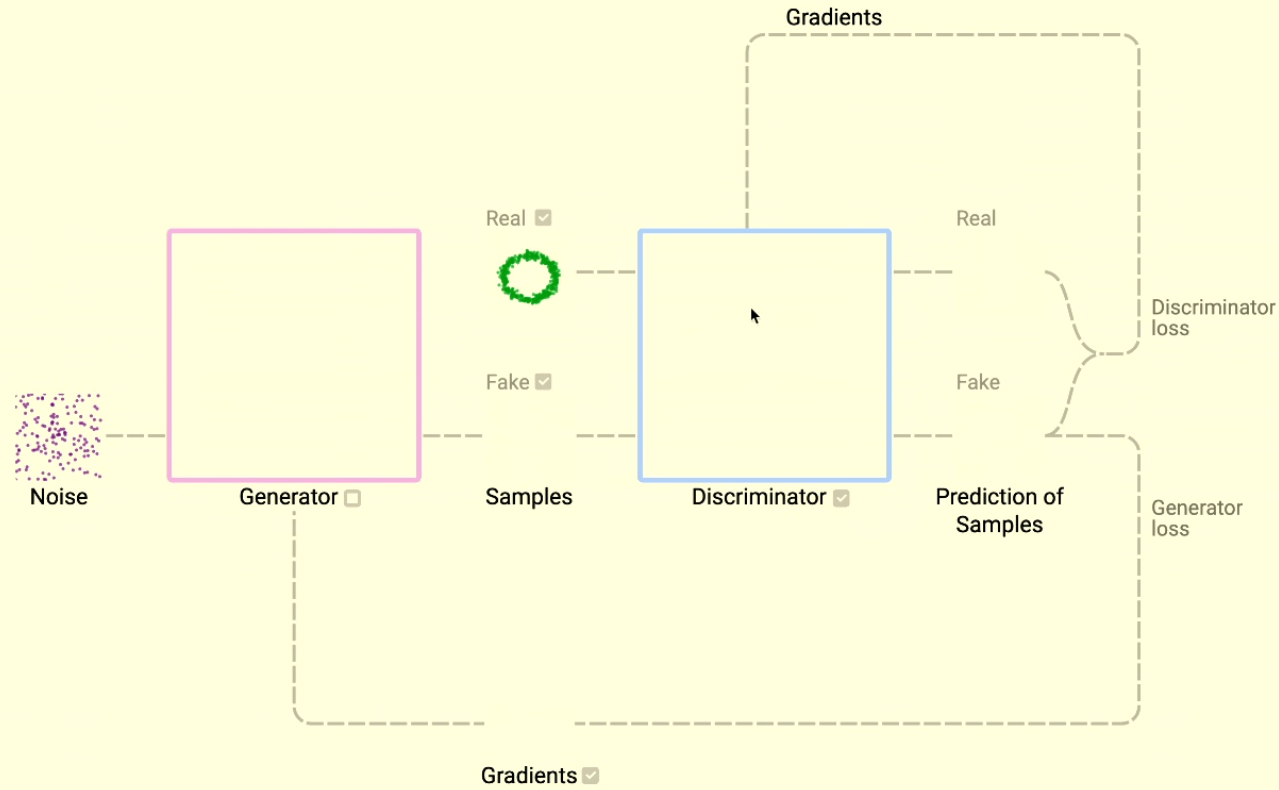


The figure above shows the model architecture of an autoencoder. Click the "Train" button to improve the autoencoder's reconstruction of input images, and click the "Sample" button to show how the model reconstructs a new sample image.

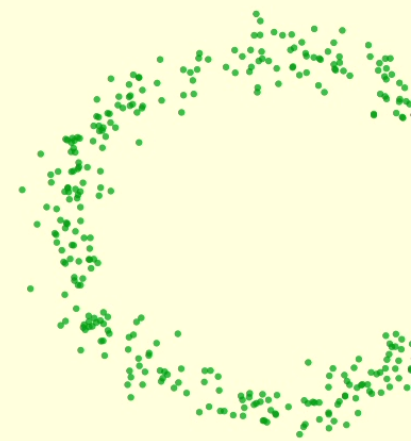
Having discussed the purpose and basic components of an autoencoder, let's now discuss how to create autoencoders using the Keras framework in Python.



MODEL OVERVIEW GRAPH



LAYERED DISTRIBUTIONS



Each dot is a 2D data sample: **real samples**; **fake sample**

Background colors of grid cells represent **discriminator's** prediction. Samples in **green regions** are likely to be real; those in **purple** regions are likely to be fake.



GAN Lab Data Distribution Use pre-trained model Epoch 000,798

MODEL OVERVIEW GRAPH

Shows the architecture of a model and data flow

LAYERED DISTRIBUTIONS

Each dot is a 2D data sample: **real samples**, **fake samples**.

Background colors of grid cells represent **discriminator's** classifications. Samples in **green regions** are likely to be real; those in **purple regions** likely fake.

Manifold represents **generator's** transformation results from noise space. Opacity encodes density: darker purple means more samples in smaller area.

Pink lines from fake samples represent **gradients** for generator.
 This sample needs to move upper right to decrease generator's loss.

METRICS

What is a GAN?