Generative machine learning models for scenario simulation

Vadim Strizhov

Geneva, 2023

Generate a scenario with a probabilistic model

Terms:

- scenario is a time series realization of a reconstructed stochastic process,
- machine learning selects a reconstruction model to fit data,
- model generates data of same distribution.

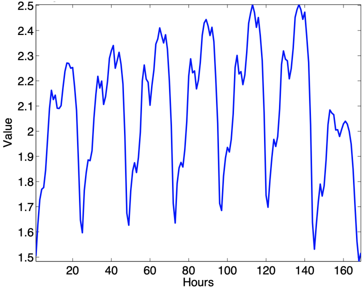
The goal: to reconstruct true distribution of data

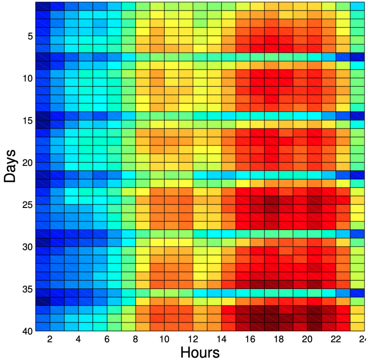
The method: Principal Component Analysis (as autoencoder)

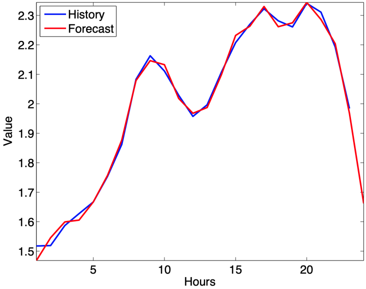
Assumptions:

1. time series are relatively short,
2. variance of each time series is high,
3. covariance of some time series is high.

One-state scenario forecasting model







The design matrix is

$$\begin{array}{c|c} \underset{1\times 1}{y_{t+1}} & \underset{1\times n}{\boldsymbol{x}_t} \\ \hline \underset{t\times 1}{\boldsymbol{y}} & \underset{t\times n}{\boldsymbol{X}} \end{array},$$

the forecasting model $\hat{\boldsymbol{y}} = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{X}$,
the forecast $\hat{y}_{t+1} = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_t$.

Singular Spectrum Analysis and state space

1. Construct the Hankel matrix with time series,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{x}_3 \\ \vdots \\ \boldsymbol{x}_t \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \\ x_2 & x_3 & x_4 & \dots & x_{n+1} \\ x_3 & x_4 & x_5 & \dots & x_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_t & x_{t+1} & x_{t+2} & \dots & x_{t+n} \end{bmatrix}.$$

2. Decompose the matrix and take the k first components,

$$\boldsymbol{X} = \boldsymbol{X}_1 + \dots + \boldsymbol{X}_n = \boldsymbol{U}\Lambda\boldsymbol{V}^\mathsf{T} = \sum_{k=1}^{n} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^\mathsf{T}.$$

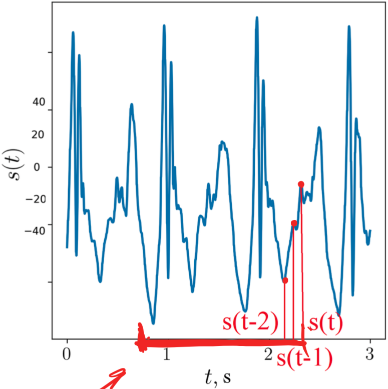
3. Reconstruct the time series by anti-diagonal average

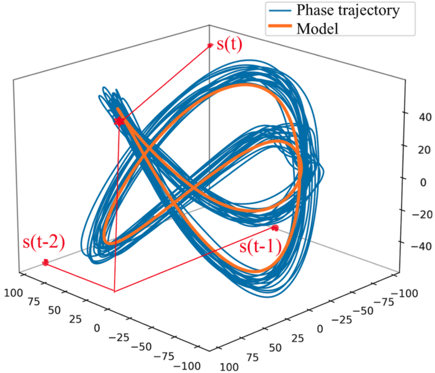
$$\hat{x}_t = \text{mean}\boldsymbol{X}(i,j), \quad i+j=t-1.$$

State space models describe the change of state x in time $\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(\boldsymbol{z})$

Define the state space by vectors $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_t\}$, the phase trajectory $\boldsymbol{X}$.

Model of the phase trajectory in the state space





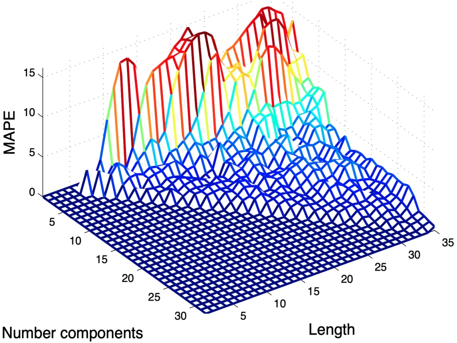
t - 1000

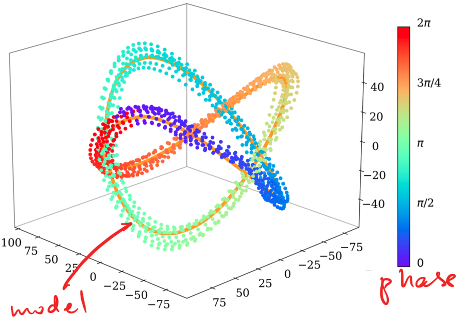
$\dim(s) \approx 1000 \longrightarrow \dim(x) = 4$

Reduce dimensionality with the principal component analysis, autoencoder $\boldsymbol{z} = \boldsymbol{W}^\mathsf{T}\boldsymbol{x}$ where $\boldsymbol{W}$ is an orthogonal (rotation) matrix. The first principal components are given by Singular Values Decomposition

$$\sqrt{\lambda_k}\boldsymbol{V_k} = \boldsymbol{X}^\mathsf{T}\boldsymbol{U}_k \quad \text{the SVD is} \quad \boldsymbol{X} = \boldsymbol{U}\Lambda\boldsymbol{V}^\mathsf{T}$$

The model complexity and the phase trajectory



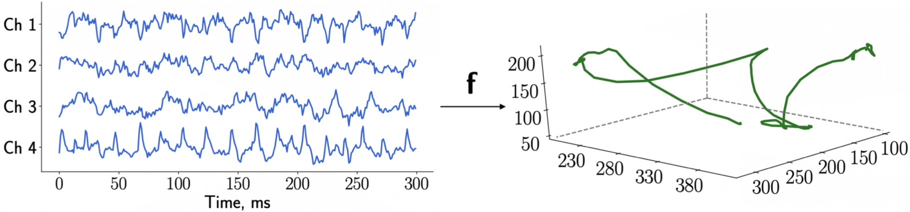


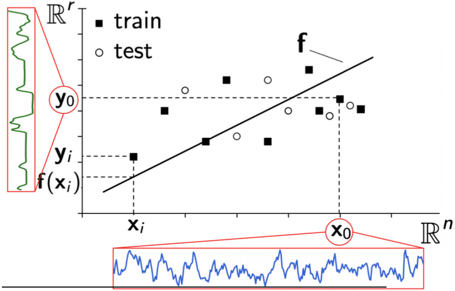
The length is n, here a point of the phase trajectory $\boldsymbol{x}_t \in \mathbb{R}^n$, the complexity is k. The encoder is

$$\underset{1\times k}{\boldsymbol{z}} = \underset{k\times n}{\boldsymbol{W}^{\mathsf{T}}}\,\underset{n\times 1}{\boldsymbol{x}}$$

Canonical correlation analysis

To control the scenario, it reconstructs dependencies in design space, target space, and align in-between. The forecasting model is $\boldsymbol{f} = \boldsymbol{W}\Lambda\boldsymbol{Q}$.





$$\begin{tikzcd}
\mathbf{x} \in \mathbb{R}^n \arrow[rr, "\mathbf{f}"] \arrow[dr, bend left, "\mathbf{W}"'] & & \mathbf{y} \in \mathbb{R}^r \arrow[dl, bend left, "\mathbf{C}"] \\
& \mathbf{t}, \mathbf{u} \in \mathbb{R}^\ell \arrow[ul, bend left, "\mathbf{P}"'] \arrow[ur, bend left, "\mathbf{Q}"] &
\end{tikzcd}$$

$$\mathbf{x} = \mathbf{P}\mathbf{t} + \mathbf{e}_{\mathbf{x}}$$

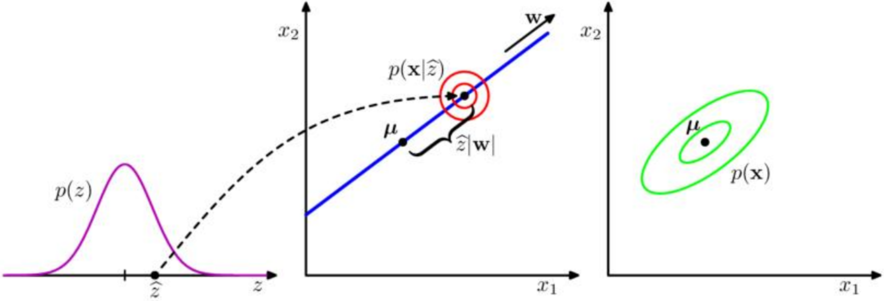
$$\mathbf{y} = \mathbf{Q}\mathbf{u} + \mathbf{e}_{\mathbf{y}}$$

$$\operatorname{cov}(\mathbf{t}, \mathbf{u}) \to \max_{\mathbf{P}, \mathbf{Q}}$$

Quadratic programming feature selection for multicorrelated signal decoding by R. Isachenko and V. Strijov, 2022, Expert Systems

The simplest generative model

Probabilistic principal component analysis: to reconstruct the initial data, sample from the normal distribution

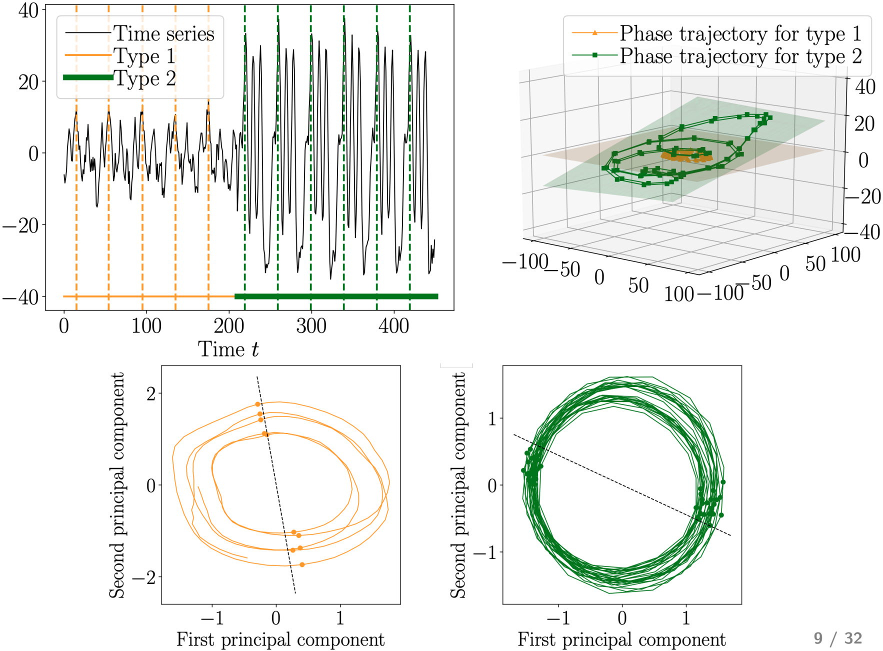


$$\boldsymbol{x} = \underbrace{\boldsymbol{W}^{\mathsf{T}}\boldsymbol{z} + \boldsymbol{\mu}}_{\text{deterministic}} + \underbrace{\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})}_{\text{stochastic}}.$$

Denote by $p(\boldsymbol{z})$ distribution in latent space, and $p(\boldsymbol{x} \mid \boldsymbol{z})$ in the data space given the latent variable $\boldsymbol{z}$.

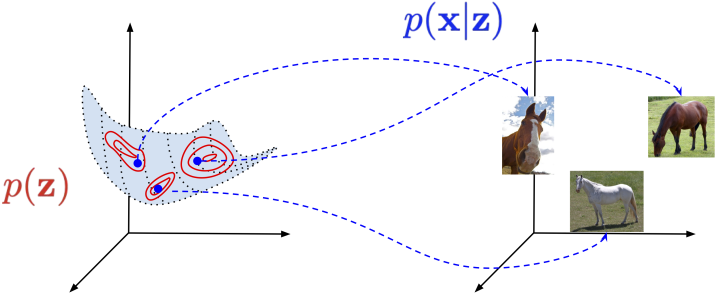
A latent space-based estimation of distribution algorithm for large-scale global optimization by W. Dong et al., 2018, Soft Computing

Select an optimal manifold, given a mixture



The simplest generative model, a mixture

Each data cluster has its own mean and variance.



PCA reveals manifolds and reduces data dimensionality. Shall we use a deterministic or a probabilistic manifold?

Latent Variable Models by J.M. Tomczak, 2021, Chapter

Generative versus discriminative models

The variable $\boldsymbol{x}$ is either probabilistic or deterministic.

- Discriminative Model
- Generative Model

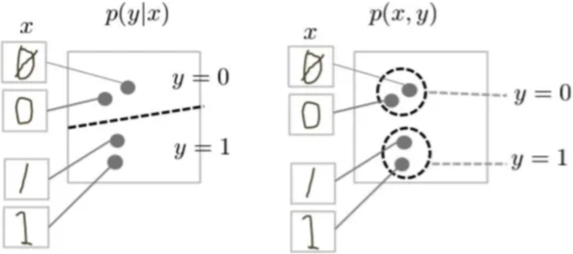


Fig 1. From a Machine Learning course by Google

The Bayes' rule

$$p(y \mid \boldsymbol{x}) = \frac{\overbrace{p(\boldsymbol{x} \mid y)p(y)}^{p(\boldsymbol{x},y)}}{p(\boldsymbol{x})}$$

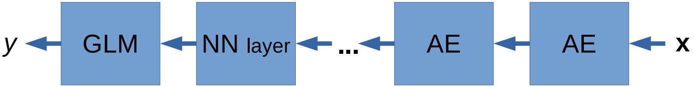
Discriminative: in the logistic regression $\boldsymbol{x}$ is not a random variable,

$$p(y \mid \boldsymbol{x}) = \left(1 + \exp(-\boldsymbol{w}^\mathsf{T}\boldsymbol{x})\right)^{-1}.$$

Generative: in the naive Bayesian $\boldsymbol{x}$ is a random variable, here it is normally distributed,

$$p(\boldsymbol{x} \mid y_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp -\frac{1}{2\sigma_k^2}(\boldsymbol{x} - \boldsymbol{c}_k)^2.$$

Neural network with stack of autoencoders



$$f = \underset{1\times1_k}{\boldsymbol{w}^{\mathsf{T}}}\boldsymbol{z}_{k-1} \circ \boldsymbol{W}_{k-1}^{\mathsf{T}}\boldsymbol{z}_{k-2} \circ \cdots \circ \underset{n_2\times1}{\boldsymbol{W}_2^{\mathsf{T}}\boldsymbol{z}_1} \circ \underset{n_1\times n}{\boldsymbol{W}_1^{\mathsf{T}}}\,\underset{n\times1}{\boldsymbol{x}}$$

Neural network error
$$E_y = \big(y_i - f(\boldsymbol{x})\big)^2$$

Autoencoder reconstruction error
$$E_{\mathbf{x}} = \|\boldsymbol{x} - \boldsymbol{r}(\boldsymbol{z})\|^2$$

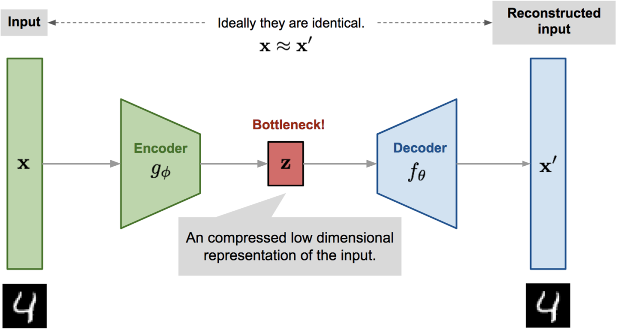
Types of autoencoders

PCA	skip block	metric	multi-linear
$\boldsymbol{W}^{\mathsf{T}}\boldsymbol{W} = \boldsymbol{I}_n$	$\boldsymbol{W} = \boldsymbol{I}_n$	$\boldsymbol{x}^{\mathsf{T}}\boldsymbol{W}\boldsymbol{x} \geqslant 0$	$\underline{\boldsymbol{W}}\boldsymbol{X}$

Autoencoder transform: $\boldsymbol{z} = \big(1 + \exp(-\boldsymbol{W}^{\mathsf{T}}\boldsymbol{x} + \boldsymbol{b})\big)^{-1}$

Reconstruction decoder: $\hat{\boldsymbol{x}} = \boldsymbol{r}\big(\boldsymbol{z}(\boldsymbol{x})\big)$

Autoencoder: probabilistic or deterministic?



Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

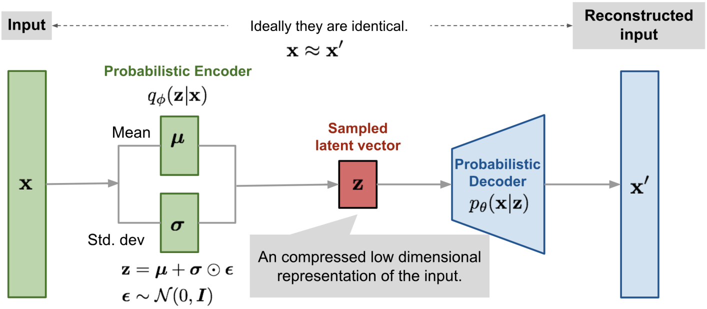
Encoder g_ϕ translates the original high-dimensional $\boldsymbol{x}$ to the low-dimensional latent $\boldsymbol{z}$.

Decoder f_θ reconstructs original $\hat{\boldsymbol{x}} \sim \boldsymbol{x}$ with the loss function

$$\left(\boldsymbol{x} - f_\theta \quad \underbrace{(g_\phi(\boldsymbol{x}))}_{\text{low-dim latent } \boldsymbol{z}} \right)^2$$

From Autoencoder to Beta-VAE by L. Weng, 2018, GitHub

Variational autoencoder

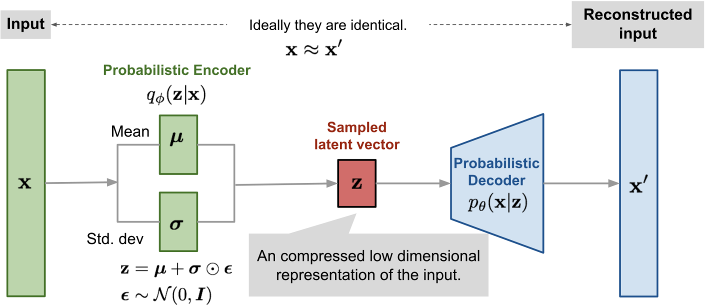

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
Mean
$\boldsymbol{\mu}$
$\boldsymbol{\sigma}$
Std. dev
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Encoder $q(\boldsymbol{z} \mid \boldsymbol{x}) = \mathrm{NN}_{\mathrm{enc}}(\boldsymbol{x}, \phi)$ outputs $\boldsymbol{\mu}_\phi(\boldsymbol{x})$ and $\boldsymbol{\sigma}_p hi(\boldsymbol{x})$ Decoder $p\boldsymbol{x} \mid \boldsymbol{z}, \theta) = \mathrm{NN}_{\mathrm{dec}}$ outputs $\boldsymbol{x}$ of similar distribution.

It is probabilistic decoder: conditional probability $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ defines a generative model, similar to decoder $f_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ and probabilistic encoder: the approximation function $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ similar to $g_\phi(\boldsymbol{z} \mid \boldsymbol{x})$.

From Autoencoder to Beta-VAE by L. Weng, 2018, GitHub

Variational autoencoder


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
Probabilistic Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{x}$
Mean
$\boldsymbol{\mu}$
Std. dev
$\boldsymbol{\sigma}$
$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$
$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$
Sampled latent vector
$\mathbf{z}$
An compressed low dimensional representation of the input.
Probabilistic Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

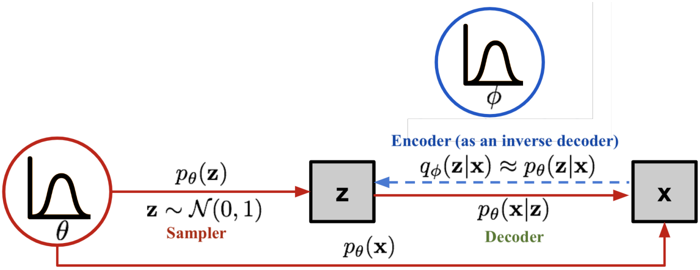
Pobability to generate real-data samples $\hat{\theta} = \arg\max \sum_{i=1}^{m} \log p_\theta(\boldsymbol{x}_i)$ the data generation procedure uses encoding vector $p_\theta(\boldsymbol{x}_i) = \int p_\theta(\boldsymbol{x}_i \mid \boldsymbol{z}) p_\theta(\boldsymbol{z}) d\boldsymbol{z}$ Approximate it with $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$

To generate a sample $\boldsymbol{x}_i$ that looks like real data

1. sample $\boldsymbol{z}_i$ from the prior distribution $p_{\hat{\theta}}(\boldsymbol{z})$
2. then generate $\boldsymbol{x}_i$ from the conditional distribution $p_{\hat{\theta}}(\boldsymbol{x} \mid \boldsymbol{z} = \boldsymbol{z}_i)$

From Autoencoder to Beta-VAE by L. Weng, 2018, GitHub

Graphical model of the Variational autoencoder



ϕ
Encoder (as an inverse decoder)
$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$
θ
$p_\theta(\mathbf{z})$
$\mathbf{z} \sim \mathcal{N}(0,1)$
Sampler
$\mathbf{z}$
$\mathbf{x}$
$p_\theta(\mathbf{x}|\mathbf{z})$
Decoder
$p_\theta(\mathbf{x})$

Loss function to teach the network parameters

$$L_{\mathsf{VAE}}(\phi, \boldsymbol{\theta}) = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) + \underbrace{D_{\mathsf{KL}}\left(q_{\phi}(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})\right)}_{\text{distributions look similar}} =$$

$$-\mathsf{E}_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) + D_{\mathsf{KL}}\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \parallel p_{\boldsymbol{\theta}}(\boldsymbol{z})\right),$$

$$\hat{\phi}, \hat{\boldsymbol{\theta}} = \arg\min L_{\mathsf{VAE}}$$

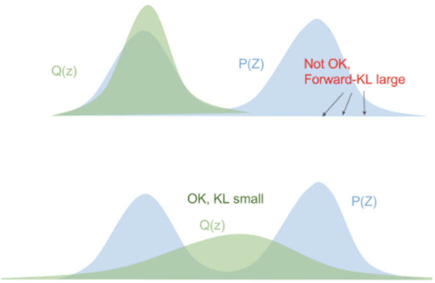
Design of Variational Autoencoder for Generation of/ by A,. Das et al., 2021, Innovation in Power Eng.

Multi-modal distribution of data and uni-modal prior

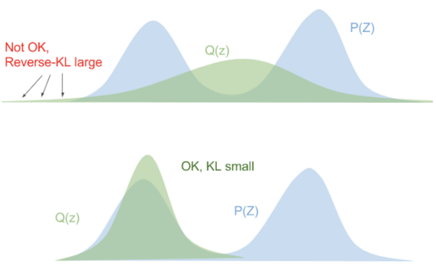
Forward and reversed KL divergence

$$D_{\mathrm{KL}}(p \,\|\, q) = \int_{-\infty}^{\infty} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, d\boldsymbol{x}$$

matches two distributions in different ways.



Forward KL: $D_{\mathrm{KL}}(P\|Q)$



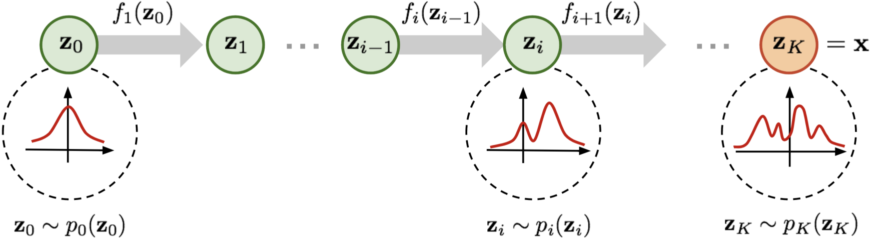
Reversed KL: $D_{\mathrm{KL}}(Q\|P)$

Variational Bayes by E. Jang, 2016, source

The normalizing flow is a superposition

The flow $f_1, \ldots, f_K$ must be

1. easily invertible,
2. its Jacobian determinant is easy to compute.

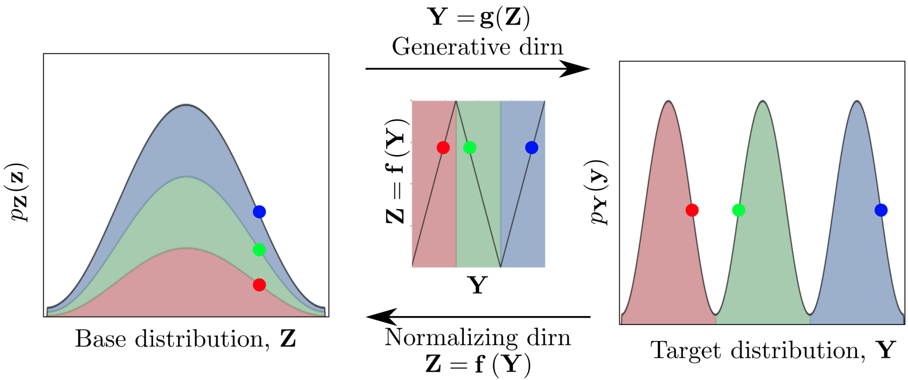


The target distribution $\log p_K(\boldsymbol{z}_K) = \log p_0(\boldsymbol{z}_0) - \sum_{i=1}^{K} \log \left(\det \frac{df_i(\boldsymbol{z}_{i-1})}{d\boldsymbol{z}_{i-1}}\right)$.

Let $\boldsymbol{z}_0 = f_1^{-1}(\boldsymbol{z}_1)$. Change variables in the pdf so $p_1(\boldsymbol{z}_1) = p_0(\boldsymbol{z}_0) \left(\det \frac{df_1^{-1}(\boldsymbol{z}_1)}{d\boldsymbol{z}_1}\right)$.

Flow-based Deep Generative Models by L. Weng, 2018, GitHub

An example of the flow: piecewise bijective

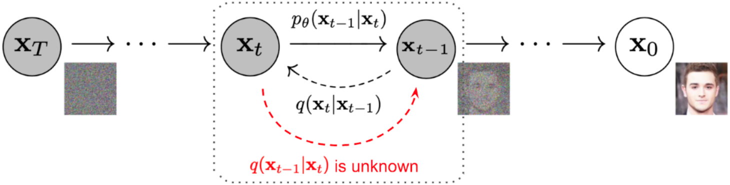

$\mathbf{Y} = \mathbf{g}(\mathbf{Z})$
Generative dirn
$p_{\mathbf{Z}}(\mathbf{z})$
$\mathbf{Z} = \mathbf{f}(\mathbf{Y})$
$\mathbf{Y}$
$p_{\mathbf{Y}}(\mathbf{y})$
Base distribution, $\mathbf{Z}$
Normalizing dirn
$\mathbf{Z} = \mathbf{f}(\mathbf{Y})$
Target distribution, $\mathbf{Y}$

A monotone function maps sections of data domain to the base distribution.

To invert the function, sample the base distribution with a gating network. Use a mixture of experts network.

Normalizing Flows: review by I. Kobyzin et al., 2020, IEEE

Diffusion models: learn slowly by adding noise



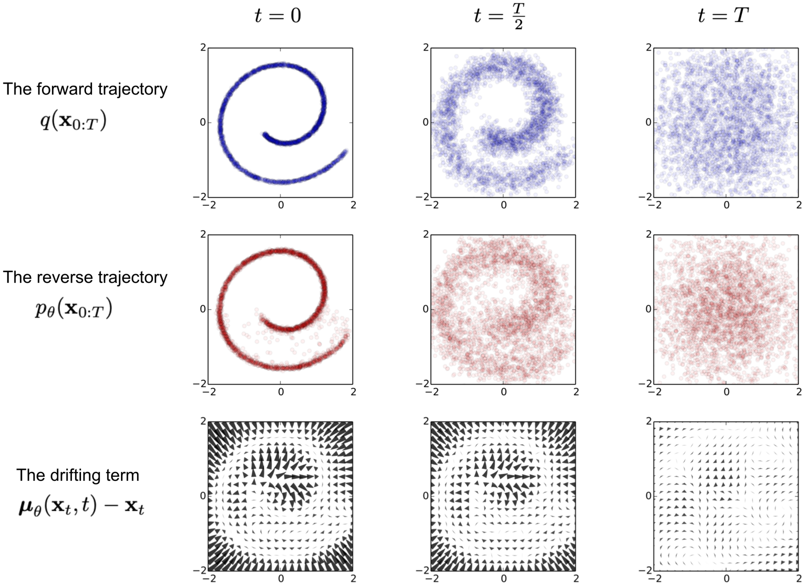
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$
$q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown

Given the data distribution $\boldsymbol{x}_0 \sim q(\boldsymbol{x})$ set:

1. forward diffusion process $\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{z}_t$, sampling i.i.d. $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_T \sim \mathcal{N}(0, \boldsymbol{I})$,
2. sampling $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$ and learning parameters $\boldsymbol{\theta}$ of U-Net $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$,
3. reverse diffusion process $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{0:T}) = p(\boldsymbol{x}_T)\prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$,
4. slow learning gives $p_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \Sigma_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t))$.

Denoising diffusion probabilistic models by J. Ho, 2020 ArXiv

An example of training a diffusion model



$t = 0$ $t = \frac{T}{2}$ $t = T$

The forward trajectory $q(\mathbf{x}_{0:T})$

The reverse trajectory $p_\theta(\mathbf{x}_{0:T})$

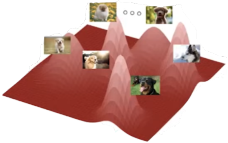
The drifting term $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) - \mathbf{x}_t$

Deep unsupervised learning by J. Sohl-Dickstein et al., 2015, ArXiv

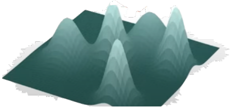
Building complex generative models

The main challenge is to estimate the normalizing constant

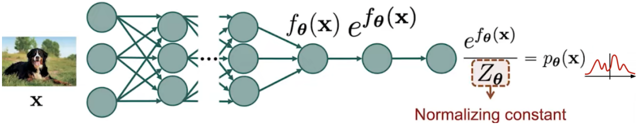
$$Z_{\boldsymbol{\theta}} = \int \exp\big(f_{\boldsymbol{\theta}}(\boldsymbol{x})\big)\, d\boldsymbol{x}$$



Unknown data distribution



Model distribution



$\mathbf{x}$
$f_{\boldsymbol{\theta}}(\mathbf{x})$
$e^{f_{\boldsymbol{\theta}}(\mathbf{x})}$
$\frac{e^{f_{\boldsymbol{\theta}}(\mathbf{x})}}{Z_{\boldsymbol{\theta}}} = p_{\boldsymbol{\theta}}(\mathbf{x})$
Normalizing constant

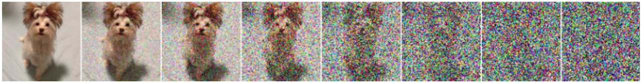
Approximate unknown true distribution with a neural network

Normalizing for Gaussian distribution $Z_{\boldsymbol{\mu}} = 2\pi^{-\frac{d}{2}}$

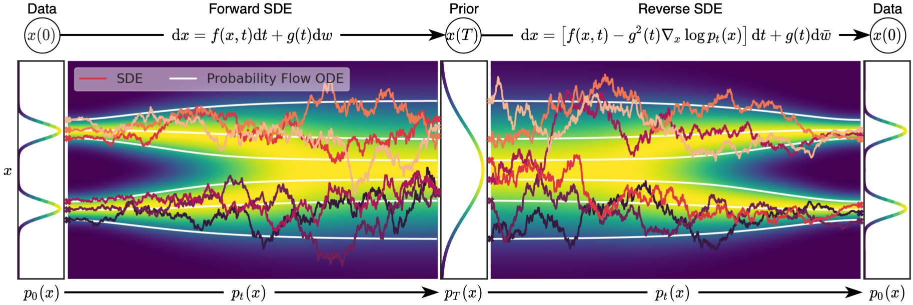
Score-based generative modeling through SDEs by Yang Song et al., 2020, ArXiv

Score-based generative model via Neural SDEs

SDE smoothly transforms a complex data distribution to a known prior distribution by slowly injecting noise.



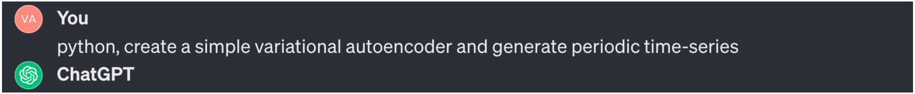
To reverse the SDE compute the score $\nabla_{\mathrm{x}\log p_t(\mathrm{x})}$ of the distribution at each t.


Data
Forward SDE
Prior
Reverse SDE
Data
$x(0)$
$\mathrm{d}x = f(x,t)\mathrm{d}t + g(t)\mathrm{d}w$
$x(T)$
$\mathrm{d}x = \left[f(x,t) - g^2(t)\nabla_x \log p_t(x)\right]\mathrm{d}t + g(t)\mathrm{d}\bar{w}$
$x(0)$
SDE
Probability Flow ODE
x
$p_0(x)$
$p_t(x)$
$p_T(x)$
$p_t(x)$
$p_0(x)$

Score-based generative modeling through SDE by Y. Song, 2022, ArXiv

Create a test generative model

A 30-second project. Request for a code

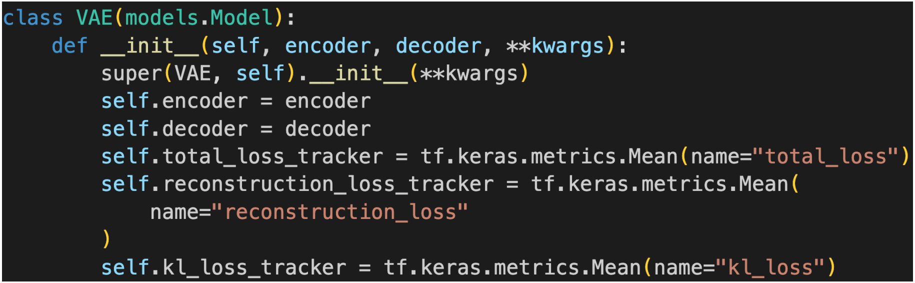


You

python, create a simple variational autoencoder and generate periodic time-series

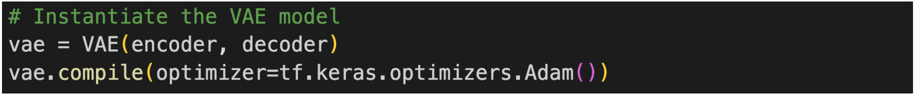
ChatGPT

Define the model



```
class VAE(models.Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder
        self.total_loss_tracker = tf.keras.metrics.Mean(name="total_loss")
        self.reconstruction_loss_tracker = tf.keras.metrics.Mean(
            name="reconstruction_loss"
        )
        self.kl_loss_tracker = tf.keras.metrics.Mean(name="kl_loss")
```

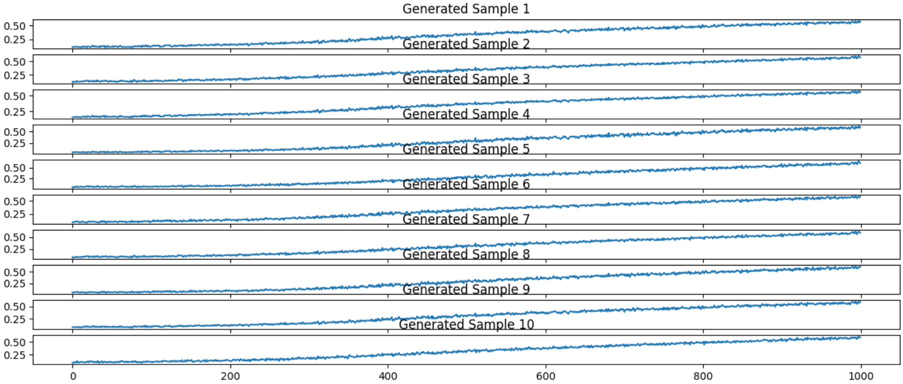
Run with a useful optimizer



```
# Instantiate the VAE model
vae = VAE(encoder, decoder)
vae.compile(optimizer=tf.keras.optimizers.Adam())
```

This code is generated by ChatGPT 3.5

The generated time series



This code is generated by ChatGPT 3.5

To program a simple project

Set an object to sample

describe a type of relations between time series (none, multi-linear, metric)

1. a point in a phase trajectory
2. set of points in trajectories
3. a dynamic graph in a graph trajectory
4. CCA source and target trajectories

Set a generation model to tune

put forward a hypothesis on data distribution; it makes optimization criteria to tune NN

1. variational auto-encoder
2. normalizing flow model
3. diffusion probabilistic model

Set an external utility function

it selects a type of model and structure of the neural network

Select performance measurement routine and dataset

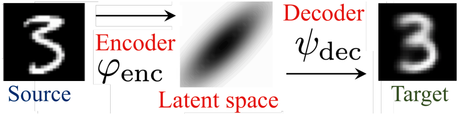
Generative model for Canonical Correlation Analysis

Source

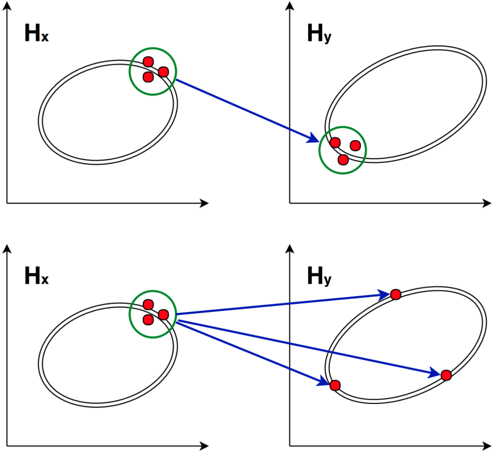
Target

$$\begin{tikzcd}
\underset{n\times d}{\mathbf{X}} \arrow[r, "\text{Forecast and}", "\text{portfolio selection}"'] \arrow[d, bend right, "\text{Encoder } \varphi_{\text{enc}}"'] & \underset{n\times s}{\mathbf{Y}} \arrow[d, bend left, "\psi_{\text{enc}}"] \\
\underset{n\times K}{\mathbf{T}} \arrow[u, "\varphi_{\text{dec}}"'] \arrow[r, leftrightarrow, "\text{Latent space of}", "\text{generative model}"'] & \underset{n\times K}{\mathbf{U}} \arrow[u, "\text{Decoder } \psi_{\text{dec}}"]
\end{tikzcd}$$

1. Approximates both spaces, design and target
2. Reduce the dimensionality of spaces, select the connected data
3. Select a subset of target time series



Convergent cross mapping as a distance function

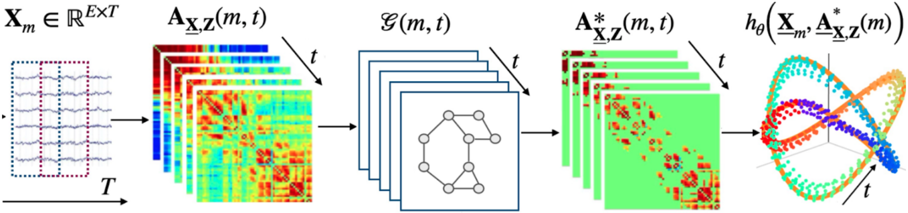


The time series $\boldsymbol{y}$ depends on the time series $\boldsymbol{x}$, if in the neighbourhood $(\boldsymbol{x}, \boldsymbol{x}') \in \boldsymbol{H_x}$ there exists a Lipschitz continuous map

$$\varphi \boldsymbol{H_x} \to \boldsymbol{H_y} \quad \text{such that} \quad \rho_{\boldsymbol{H_y}}\big(\varphi(\boldsymbol{x}, \boldsymbol{x}')\big) \geqslant L\rho_{\boldsymbol{H_x}}(\boldsymbol{x}, \boldsymbol{x}').$$

High variance and high co-variance in time series

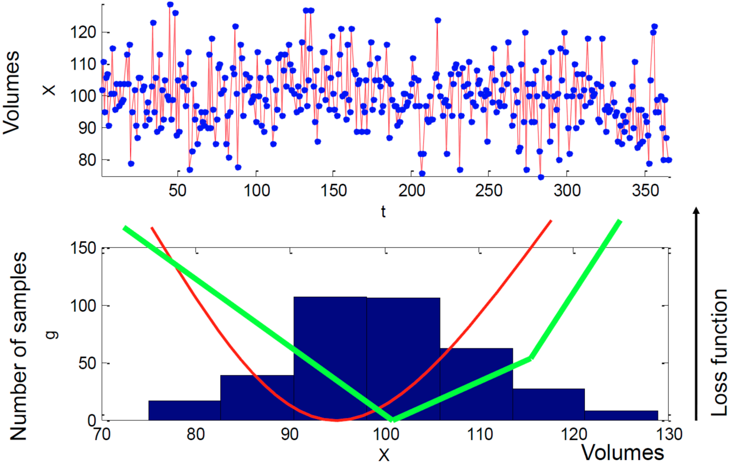
Dynamic graph reflects dependencies between the time series.


$\mathbf{X}_m \in \mathbb{R}^{E\times T}$
T
$\mathbf{A}_{\underline{\mathbf{X}},\mathbf{Z}}(m,t)$
t
$\mathcal{G}(m,t)$
t
$\mathbf{A}^*_{\underline{\mathbf{X}},\mathbf{Z}}(m,t)$
t
$h_\theta\Big(\underline{\mathbf{X}}_m, \mathbf{A}^*_{\underline{\mathbf{X}},\mathbf{Z}}(m)\Big)$
t

To reconstruct the dependencies

1. define distance between points of the phase trajectories,
2. make low-rank decomposition, prune the dependency graph,
3. reconstruct time series.

Convolution with an engineered utility function



There given the histogram $\{x_i, g_i\}$ and the utility function $L(z, x)$, for example, $|z - x|$ or $(z - x)^2$. Find the forecast $\hat{x}$ as

$$\hat{x} = \underset{z \in \{x_1, \dots, x_m\}}{\arg\min} \sum_{i=1}^{m} g_i L(z, x_i).$$

Tools to create generative models

General purpose

1. PyTorch, TensorFlow (Keras, TFP), and JAX
2. DCGAN, Torch-GAN and Conditional GAN
3. Google AutoML

Generative models and collections

1. Pytae: most common variational autoencoder models
2. UNet diffusion: denoising diffusion probabilistic model in PyTorch
3. PGMC: collection of generative models in PyTorch

CCA and Graph networks

1. DeepCCA: deep canonical correlation analysis
2. DGCCA: deep generalized canonical correlation analysis:
3. pyRiemann: Biosignals classification with Riemannian geometry

Articles to read

1. Introduction to Probabilistic Programming by A. Das, 2020, ayandas
2. Foundation of Variational Autoencoder (VAE) by A. Das, 2020, ayandas
3. From Autoencoder to Beta-VAE by L. Weng, 2018, GitHub
4. Flow-based Deep Generative Models by L. Weng, 2018, GitHub
5. Normalizing Flows: review by I. Kobyzin et al., 2020, IEEE
6. Variational Inference with Normalizing Flows by D.J. Rezende, S. Mohamed, 2015, ArXiv
7. Score-Based Generative Modeling through Stochastic Differential Equations by Y. Song et al., 2015, ArXiv
8. Denoising diffusion probabilistic models by J. Ho, 2020 ArXiv
9. Deep unsupervised learning using Nonequilibrium Thermodynamics by J. Sohl-Dickstein et al., 2015, ArXiv
10. An Intuitive Tutorial to Gaussian Processes Regression by J. Wang, 2020, ArXiv