A brief introduction to the mathematics of deep learning and its application in generative models

Niklas Walter

QFIPC

LMU Munich

22th June 2023

Let's see through the mist of buzzwords

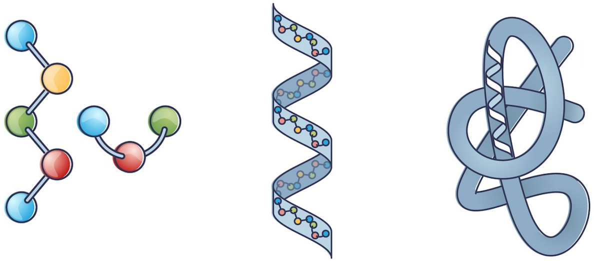
Depending on the definition one can distinguish four major approaches within machine learning:

1. Supervised learning (neural network approximation, regression, labelled classification, ...)
2. Unsupervised learning (clustering, dimension reduction, anomaly detection, ...)
3. Semi-supervised learning (generative models, e.g. GANs...)
4. Reinforcement learning (learning how to play games, ...)

Examples.

Example I: AlphaFold

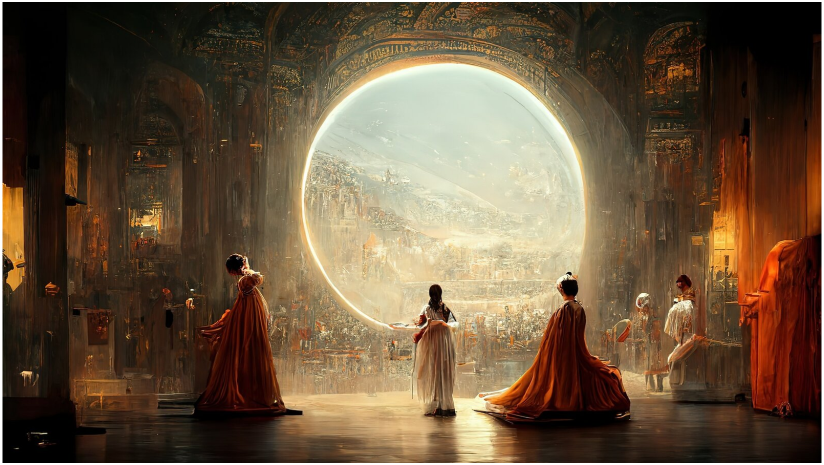
- Proteins perform numerous functions, from storing oxygen in tissues or transporting it in a blood to muscle contraction and relaxation or cell mobility.
- They come in three-four structures:



Example I: AlphaFold (cont.)

- The problem of protein folding:
  - Consider a protein sequence of 100 amino acids.
  - Each amino acid can adopt, at a minimum, about 10 different conformations due to the angles it can attain.
  - So the total number of conformations is given by 10^{100}.
  - Assume that the rate of conformational sampling is about 10^{14} conformations per second.
  - Then finding the teritary structre by a random search algorithm might take about 10^{18} years, which is more than the age of the universe.
- AlphaFold uses transformers (a special architecture of neural networks) to predict the tertiary structure of proteins based on their amino acid structure.

Who knows this picture?



Example II: Midjourney

- Midjourney is a model to generate picture based on input text.
- It uses the so called diffusion technique, which starts with a prior noise distribution and is creating an image by solving a backward stochastic differential equation (BSDE).
- It is free to use and accessible via a Discord bot.

Example II: Midjourney (cont.)

- Latest version only accessible with a premium account.

Supervised Learning.

What means supervised learning mathematically?

Consider an input space $\mathcal{X}$ and a output space $\mathcal{Y}$. Moreover, assume that there exists a unknown function

$$\mathcal{G} \ni f : \mathcal{X} \to \mathcal{Y},$$

where $\mathcal{G}$ is a function space, also called hypothesis space. Then based on some given pairs

$$\Pi := \{(x_1, f(x_1)), \dots (x_N, f(x_N))\} \in (\mathcal{X} \times \mathcal{Y})^N$$

supervised learning aims to approximate the function f based on Π.

Examples for $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{G}$

Normally in the real world, the data we can observe/measure lies in the Euclidean space, i.e. $\mathcal{X} = \mathbb{R}^n$ or $\mathcal{Y} = \mathbb{R}^m$. For examples:

1. a d-dimensional timeseries with N observations lies in $\mathbb{R}^{d \times N}$
2. a RGB coded picture lies in $\mathbb{R}^{h \times w \times 3}$
3. an audio signal recorded on N timepoints lies in $\mathbb{R}^N$

The hypothesis space $\mathcal{G}$ could vary more and might be based on our assumptions. So for example:

1. $\mathcal{G} = \{f : \mathcal{X} \to \mathcal{Y} \text{ linear}\}$ refers to linear regression
2. $\mathcal{G} = \{f : \mathcal{X} \to \mathcal{Y},\ f(x) = \sum_{i=1}^n a_i \sin(b_i x) + c_i \cos(d_i x)\}$ could make sense knowing the data is periodic
3. $\mathcal{G} = \mathcal{C}^k(\mathcal{X}, \mathcal{Y})$ makes sense if the data looks continuous/smooth

In the following, we will focus on $\mathcal{X} = \mathbb{R}^n$, $\mathcal{Y} = \mathbb{R}^m$, $\mathcal{G} = \mathcal{C}^k(\mathcal{X}, \mathcal{Y})$.

Deep Neural Networks.

What is a (feedforward) deep neural network?

Definition 1

Let $n, m, L \in \mathbb{N}$. A function $\psi : \mathbb{R}^n \to \mathbb{R}^m$ is a (feedforward) deep neural network (DNN) with $L \geq 3$ hidden layers and single activation function ϱ if it can be written as

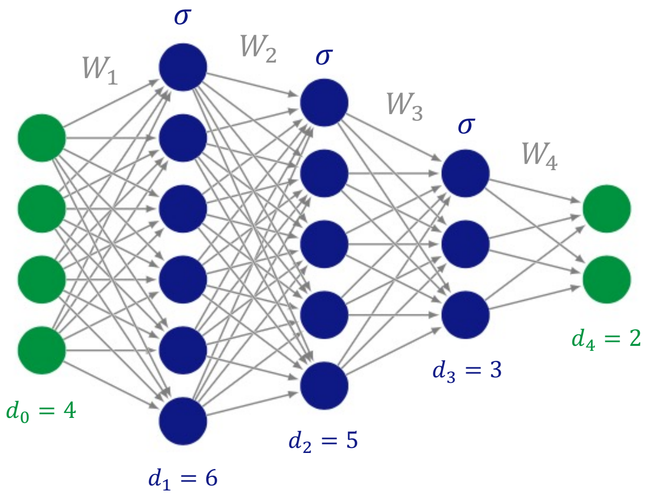
$$\psi = W_L \circ (\varrho \circ W_{L-1}) \circ ... \circ (\varrho \circ W_1), \tag{1}$$

where $W_l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ for $l = 1, ..., L$ is an affine function

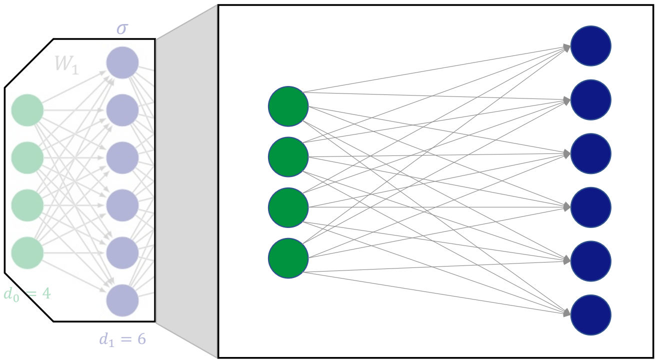
$$W_l(x) := A^l x + b^l,\ x \in \mathbb{R}^{d_{l-1}} \tag{2}$$

defined by weight matrices $A^l \in \mathbb{R}^{d_l \times d_{l-1}}$ and bias vectors $b^l \in \mathbb{R}^{d_l}$ for $d_0 := n$ and $d_L := m$.

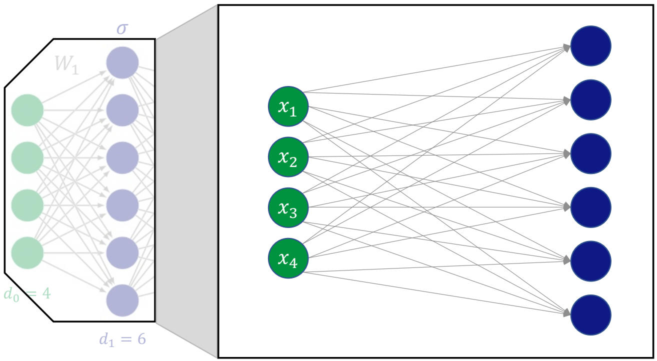
How do they work? - They are quite easy objects!



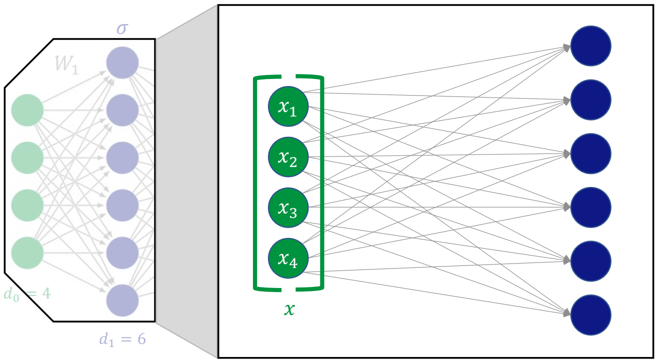
How do they work? - They are quite easy objects!



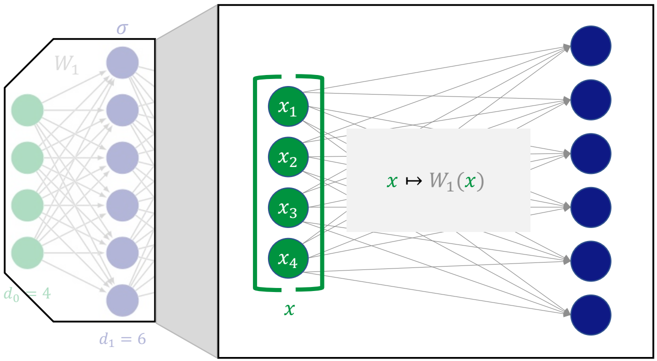
How do they work? - They are quite easy objects!



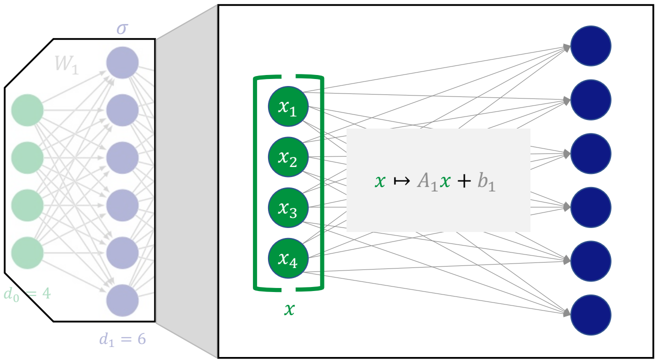
How do they work? - They are quite easy objects!



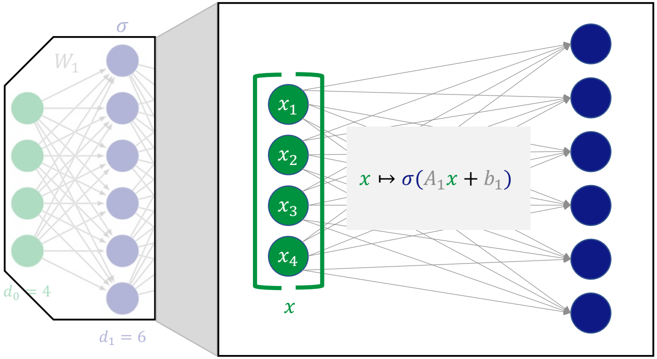
How do they work? - They are quite easy objects!



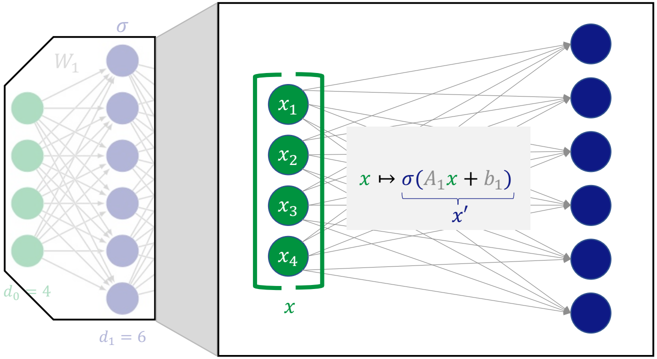
How do they work? - They are quite easy objects!



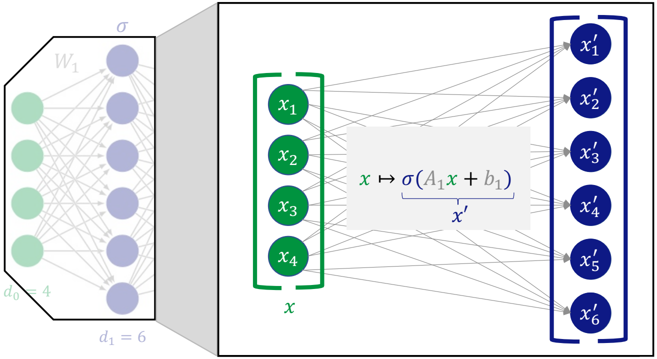
How do they work? - They are quite easy objects!



How do they work? - They are quite easy objects!



How do they work? - They are quite easy objects!



Why are those functions so interesting? - They are universal approximators.

Theorem 2 (Exemplary universal approximation theorem)

Consider the input space $\mathbb{R}^n$, output space $\mathbb{R}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ continuous and non-constant. Moreover, let $f \in \mathcal{C}^0(\mathbb{R}^n, \mathbb{R})$. Then for every $\varepsilon \in (0, 1)$ there exists a DNN $\psi : \mathbb{R}^n \to \mathbb{R}$ with L layers such that

$$\sup_x |f(x) - \psi(x)| < \varepsilon.$$

Why are those functions so interesting? - They are universal approximators.

Theorem 3 (Exemplary universal approximation theorem)

Consider the input space $\mathbb{R}^n$, output space $\mathbb{R}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ continuous and non-constant. Moreover, let $f \in \mathcal{C}^0(\mathbb{R}^n, \mathbb{R})$. Then for every $\varepsilon \in (0,1)$ there exists a DNN $\psi : \mathbb{R}^n \to \mathbb{R}$ with L layers such that

$$\sup_x |f(x) - \psi(x)| < \varepsilon.$$

This is a nice motivation to work with neural networks to approximate functions but is still quite "black-boxish". We do not know anything about their architecture.

Why are those functions so interesting? - They are universal approximators.

In the recent years so called "quantitative" universal approximation theorems were established.

Theorem 4 (Exemplary universal approximation theorem 2)

Let $[0,1]^n$ be the input space and $\mathbb{R}$ the output space. Then there exists a constant $C > 0$ such that, for every $f \in \mathcal{C}^1([0,1]^n, \mathbb{R})$ with $\|f\|_{\mathcal{C}^1} \leq 1$ and every $\varepsilon \in (0, 1/2)$, there exists a DNN ψ such that

$$\max_x |f(x) - \psi(x)| < \varepsilon$$

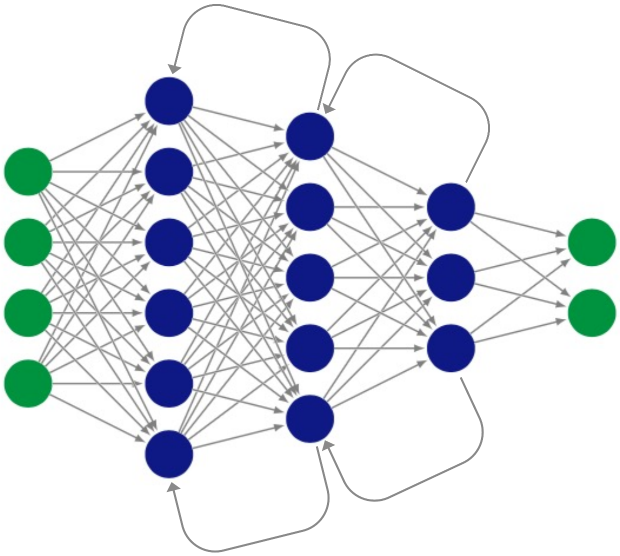
and

$$\#\text{ of non-zero matrix/vector entries } \leq C\varepsilon^{-2n}$$

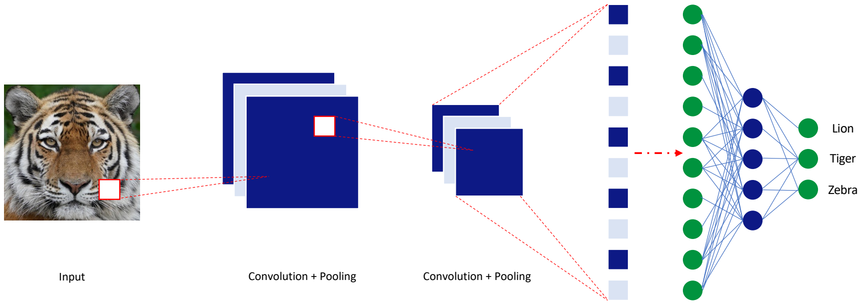
and

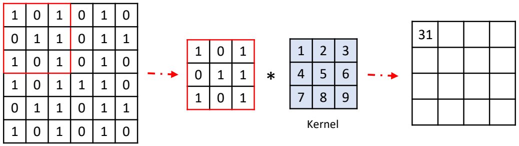
$$\#\text{ of layers } \leq C \log(1/\varepsilon).$$

Extension I: Recurrent neural networks (RNNs)



Extension II: Convolutional neural networks (CNNs)





$$\begin{array}{|c|c|c|c|c|c|}\hline 1&0&1&0&1&0\\\hline 0&1&1&0&1&1\\\hline 1&0&1&0&1&0\\\hline 1&0&1&1&1&0\\\hline 0&1&1&0&1&1\\\hline 1&0&1&0&1&0\\\hline\end{array} \dashrightarrow \begin{array}{|c|c|c|}\hline 1&0&1\\\hline 0&1&1\\\hline 1&0&1\\\hline\end{array} * \underbrace{\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4&5&6\\\hline 7&8&9\\\hline\end{array}}_{\text{Kernel}} \dashrightarrow \begin{array}{|c|c|c|c|}\hline 31&&&\\\hline &&&\\\hline &&&\\\hline &&&\\\hline\end{array}$$

Generative Models.

Problem Formulation and different approaches

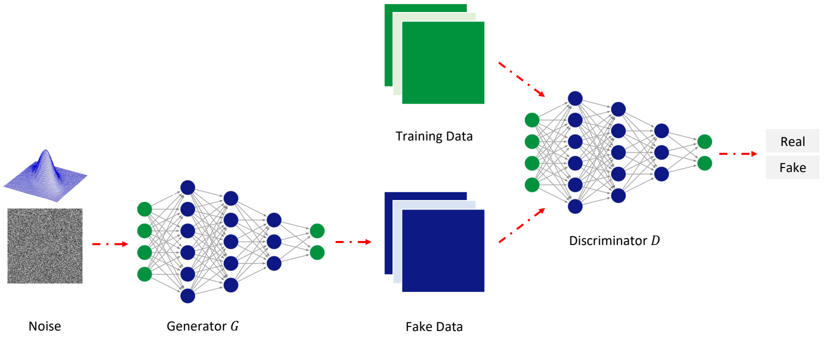
- Consider we are given a dataset Y^{real}, which is drawn from an unkown distribution $\mathbb{P}^{real}$.
- The aim of generative models is to learn the distribution $\mathbb{P}^{fake}$ so that $\mathbb{P}^{fake} \approx \mathbb{P}^{real}$. Hence, the aim is that if we sample $Y^{fake} \sim \mathbb{P}^{fake}$ then the sample should "look like" the original data.
- There are four main approaches to generative modelling:
  1. Generative Adversial Networks (GANs)
  2. Variational Autoencoder (VAE)
  3. Flow-based models
  4. Diffusion models

Generative Adversial Networks (GANs)

- Generative Adversarial Networks don't work with any explicit density estimation.
- It is based on a game theory approach with an objective to find Nash equilibrium between the two networks, Generator and Discriminator.
- **Idea:** We sample from a simple distribution like Gaussian and then learn to transform this noise to data distribution using universal function approximators (such as DNNs).
    - The generator G takes the noise as input and transforms it and passes it to the discriminator.
    - The discriminator D aims to distinguish between the data from the generator G and the original data distribution.

Generative Adversial Networks (GANs) (cont.)

- A GAN's structure can be displayed as follows



Generative Adversial Networks (GANs) (cont.)

- Mathematically, the game is defined on the probability space $(\Omega, \mathcal{F}, \mathbb{P}^{real})$.
- The generator's strategy set is given by

$$\{\mathbb{P}^{fake} \mid \mathbb{P}^{fake} \text{ probability measure on } (\Omega, \mathcal{F})\}.$$

- The discriminator's strategy set is given by

$$\{D \mid D : \Omega \to [0,1] \text{ measurable}\}.$$

- The optimisation problem is given by

$$\min_{\mathbb{P}^{fake}} \max_{D} \mathbb{E}_{x \sim \mathbb{P}^{real}}[\log(D(x))] + \mathbb{E}_{x \sim \mathbb{P}^{fake}}[1 - \log(D(x))].$$

Generative Adversial Networks (GANs) (cont.)

- In practice the measure $\mathbb{P}^{fake}$ is usually implemented as the pushforward $\mathbb{P}^{noise} \circ G^{-1}$, where $\mathbb{P}^{noise}$ is the distribution of the input noise.
- This gives rise to another formulation of the GAN game

$$\min_G \max_D \mathbb{E}_{x \sim \mathbb{P}^{real}}[\log(D(x))] + \mathbb{E}_{z \sim \mathbb{P}^{noise}}[1 - \log(D(G(z)))].$$

- The discriminator aims to attain the value 1 when evaluating data coming from the original data.
- The generator aims to pass data to the discriminator such that it attains 1.
- In conclusion, the discriminator is mainly a critic providing feedback for the generator, about "how far it is from perfection".

Wasserstein distance and WGAN

- So it might makes sense to think of a different metric to measure how far something is from perfection.
- One possibility is to consider the Wasserstein-1 metric which has the following dual representation

$$W_1(\mu,\nu) := \sup_{\|f\|_{Lip}\leq 1} \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)]$$

  for two probability meaures μ, ν.
- For the Wasserstein GAN (WGAN) the discriminator's strategy set is given by

$$\{D \mid D : \Omega \to [0,1] \text{ measurable, } \|D\|_{Lip} \leq 1\}.$$

Wasserstein distance and WGAN (cont.)

- The WGAN optimisation problem is defined as

$$\min_{\mathbb{P}^{fake}} \max_{D} \mathbb{E}_{x \sim \mathbb{P}^{fake}}[D(x)] - \mathbb{E}_{x \sim \mathbb{P}^{real}}[D(x)]$$

- In fact, in the WGAN game the discriminator provides a better gradient than in the "normal" GAN game.

Market generators

Definition 5 (Market Generator)

We refer to a generative model in a financial time series context as a *Market Generator*. That is, to neural networks that are designed to approximate the underlying distribution of an underlying market from a data sample given in form of a time series, so as to generate new data variations of the learned distribution.

- Some reasons why one need to generate synthetic data:
  - **Lack of training data:** There are many reasons in practice why one might face a lack of available training data. Insufficient datasets may lead to a poor training of machine learning models and hence inconclusive results.

Market generators (cont.)

- **Risk management:** The risk management of portfolios often requires data points of bad scenarios to better estimate tail risk measures. However, historical data point from extreme events are normally rare and hence generated paths help a lot with assessing the underlying risk.
- **Backtesting:** In many applications such as the development of trading strategies it is crucial to run backtests to measure how the strategy would have performed in a realistic environment in the past. Synthetic datasets would enable robust backtesting with less risk for overfitting.

Market generators (cont.)

- The problem of generating realistic market data was normally solved through taking samples from a parametric model, e.g.

$$\begin{cases} S_0 = s, \\ \mathrm{d}S_t = \mu S_t \,\mathrm{d}t + \sigma S_t \,\mathrm{d}W_t. \end{cases}$$

- However, making assumptions on μ and σ is very strong. Therefore, we aim to build a non-parametric market generator.
- One possible approach could be to replace the drift and diffusion by DNNs, i.e.

$$\begin{cases} S_0 = \psi_0^\theta(V),\ V \sim \mathcal{N}(0,1), \\ \mathrm{d}S_t = \psi_1^\theta(S_t)\,\mathrm{d}t + \psi_2^\theta(S_t)\,\mathrm{d}W_t. \end{cases}$$

Market generators (cont.)

- So assume we want to generate 16 synthetic datapoints for an observable asset S^{real} and we build the generator G of the form

$$\begin{cases} S_0^{fake,\theta} = \psi_0^\theta(V), \ V \sim \mathcal{N}(0,1), \\ S_t^{fake,\theta} = S_{t-1}^{fake,\theta} + \psi_1^\theta(S_{t-1}^{fake,\theta}) + \psi_2^\theta(S_{t-1}^{fake,\theta})(W_t - W_{t-1}) \end{cases}$$

  for $t = 1, \dots, 16$.

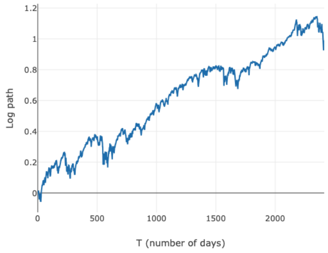
- Then the WGAN game the market generator needs to solve can be formulated as

$$\min_\theta \max_{\|f\|_{Lip} \leq 1} \mathbb{E}[f(S^{fake,\theta})] - \mathbb{E}[f(S^{real})].$$

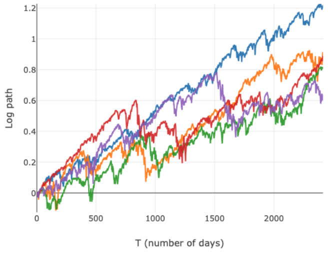
  for $S^{fake,\theta} = (S_0^{fake,\theta}, S_1^{fake,\theta}, \dots, S_{16}^{fake,\theta})$.

Market generators (cont.)

- Example of generated paths:



(a) S&P 500 log path



(b) Generated log paths of a Quant GAN

Questions?