Data Science

2020/06/12 Course 3

Machine Learning II: PCA, regression, sequential data analysis, deep learning

Data Science Center, Naoaki ONO

Contents

1. Basic Statistics
2. Machine Learning: Classification, Clustering
3. **Machine Learning II: PCA, regression, sequential data analysis, deep learning**
4. Applications of Information Science and Data Science in Biology
5. Systems Biology
6. Descriptors in Material and Molecular Design
7. PCA・PLS (development of organic materials) and Pareto solutions (Catalyst design)

Contents

1. 基礎統計
2. 機械学習I:分類、クラスタリング
3. **機械学習II:PCA、回帰、系列データ学習、深層学習**
4. バイオサイエンスにおけるビッグデータ解析とデータサイエンス
5. システムバイオロジー
6. マテリアル・分子設計における記述子
7. PCAとPLS(有機材料の開発)、パレート最適解(触媒設計)

Preface

- **Environments for Data Science**
  - Editor

    Atom, Vim, Emacs
  - IDE

    Eclipse, Visual Studio, XCode, etc…
  - Interactive

    Jupyter (via Anaconda)
  - Cloud

    Google Colaboratory, MS Azure, AWS, Kaggle, etc…

Preface

- **Modeling and Learning**
  - Why we need models?
  - What is "learning" in data science?
  - How can we know which modeling is better?

Principal Component Analysis (PCA)
（主成分分析）

PCA

- **Principal Component Analysis**
  - Linear method to **reduce variable dimension.**
  - Pros

    Easy to apply, simple to interpret.
  - Cons

    Not suitable for noisy, high dimensional data.

  - Currently we have many other non-linear models for dimension reduction, though PCA is still useful for simple data.

PCA

- **Principal Components**
  - PC is a vector that maximize the variance of the given multidimensional variables.
  - Thus, the first PC is defined as below,

$$\mathbf{v_1} = \mathrm{argmax}_{|\mathbf{v}|=1} |X\mathbf{v}|^2$$

    where X is given variable set.

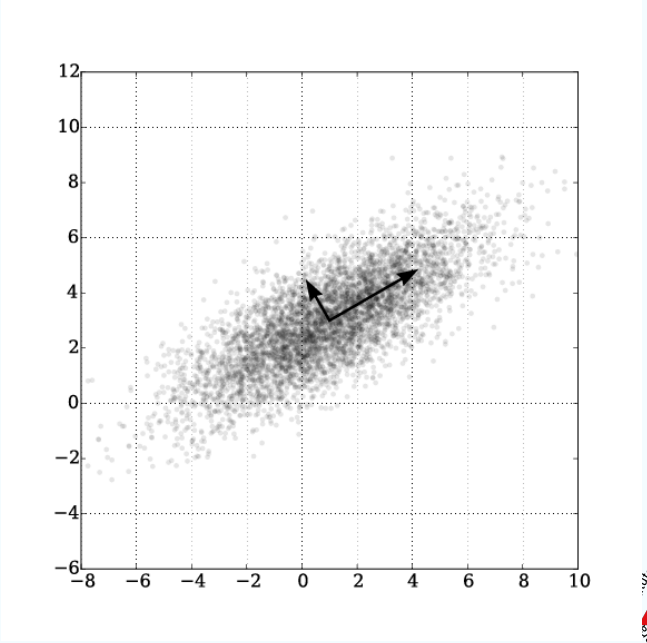
  - And subsequent components are,

$$X_{i+1} = X_i - X\mathbf{v}_i$$

$$\mathbf{v_i} = \mathrm{argmax}_{|\mathbf{v}|=1} |X_i\mathbf{v}|^2$$

PCA

- **Score**
  - Components of original variables X along with each PC.
- **Contribution**
  - Covariance of X with each PC divided by the total variance.
- **Rotation**
  - Transformation matrix from original to PC score.



PCA

- **Singular Value Decomposition (SVD)**
  - PCA can be regarded as one of the applications of singular value decomposition.

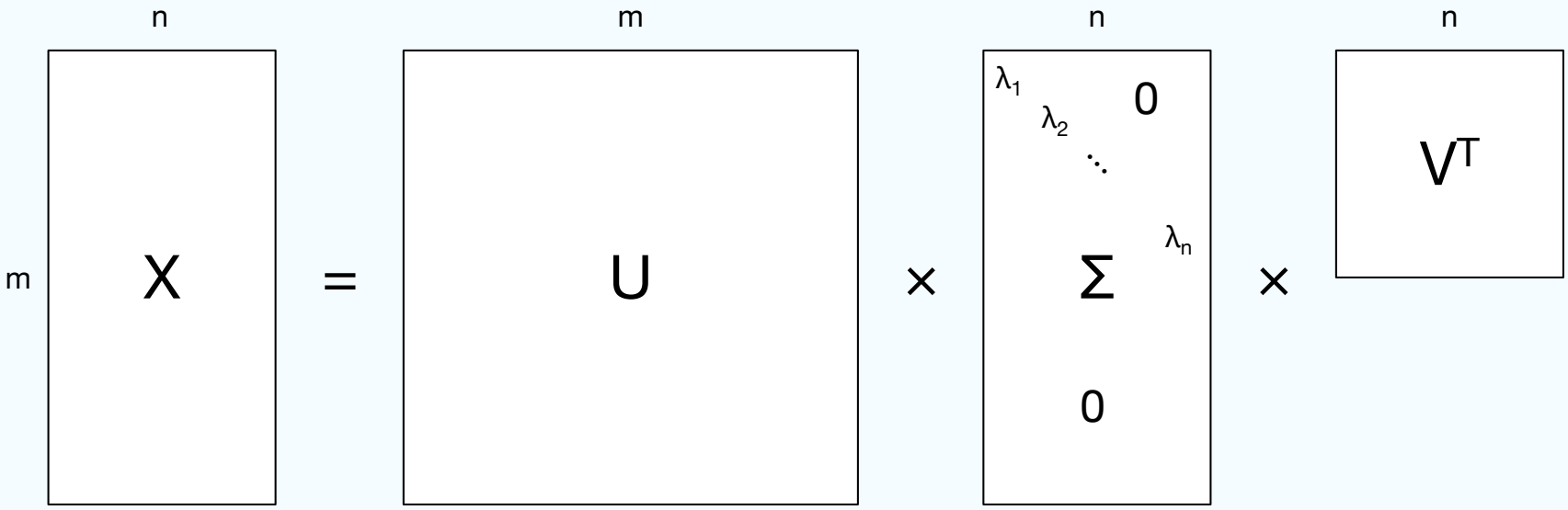
  - For any matrix X, there exists unitary matrixes U, V and Σ, which diagonal elements are non-negative and other elements are 0, that satisfies,

$$X = U\Sigma V^T$$

Unlike eigenvalue decomposition, X can be any $m \times n$ matrix.

PCA

- **Singular Value Decomposition (SVD)**



If $m < n$, Σ will be a horizontal matrix.

PCA

- For any matrix X there exists U, V and Σ, such that,

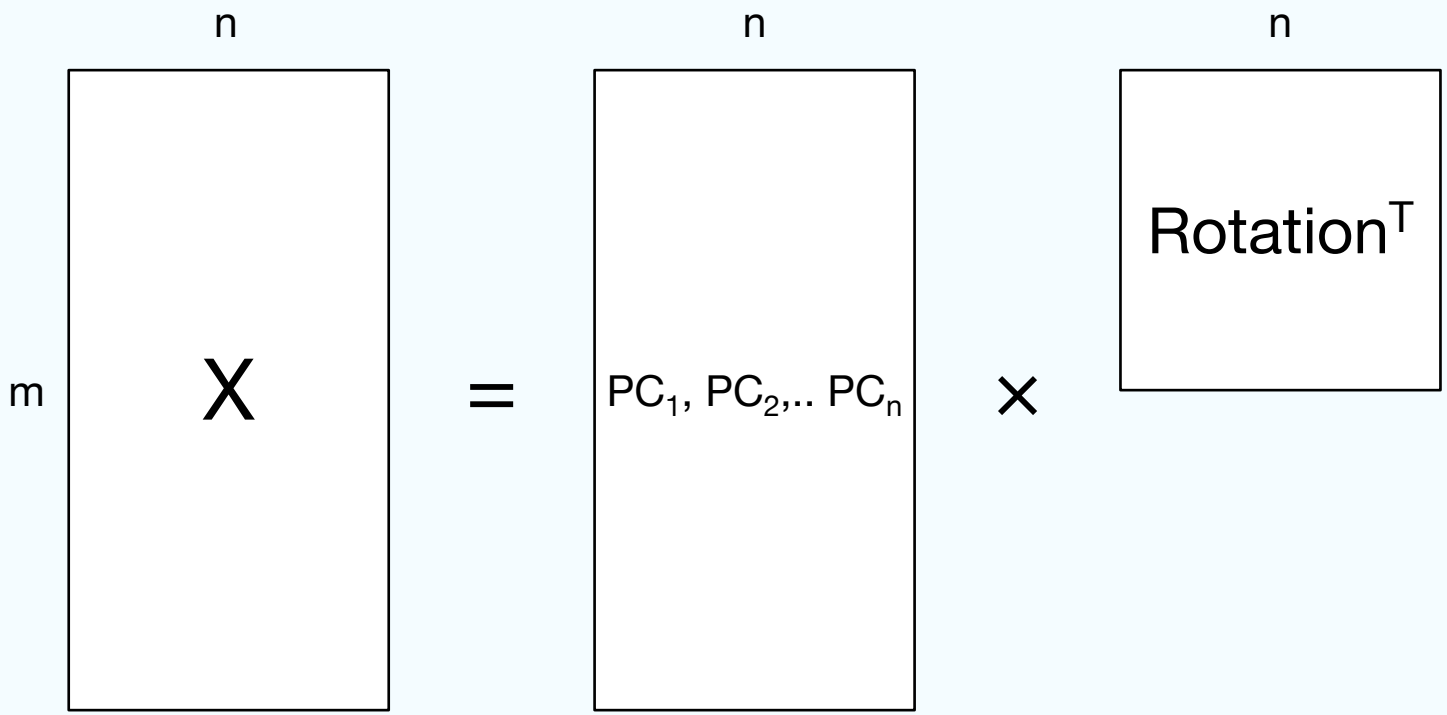
$$X = U\Sigma V^T$$

- The diagonal elements of Σ (i.e., λ_i) are **the square root of the eigenvalues** of $X^T X$, and columns of V are its **eigenvectors**.

- **PCA**
  - **The PCA score of X is $U\Sigma$, the rotation is V.** Contributions of each PC are equal to the relative proportion of λ_i.

PCA

- **PCA**
  - **The PCA score of X is $U\Sigma$, the rotation is V.**



PCA

- **Dimension reduction**

$$X = U\Sigma V^T$$

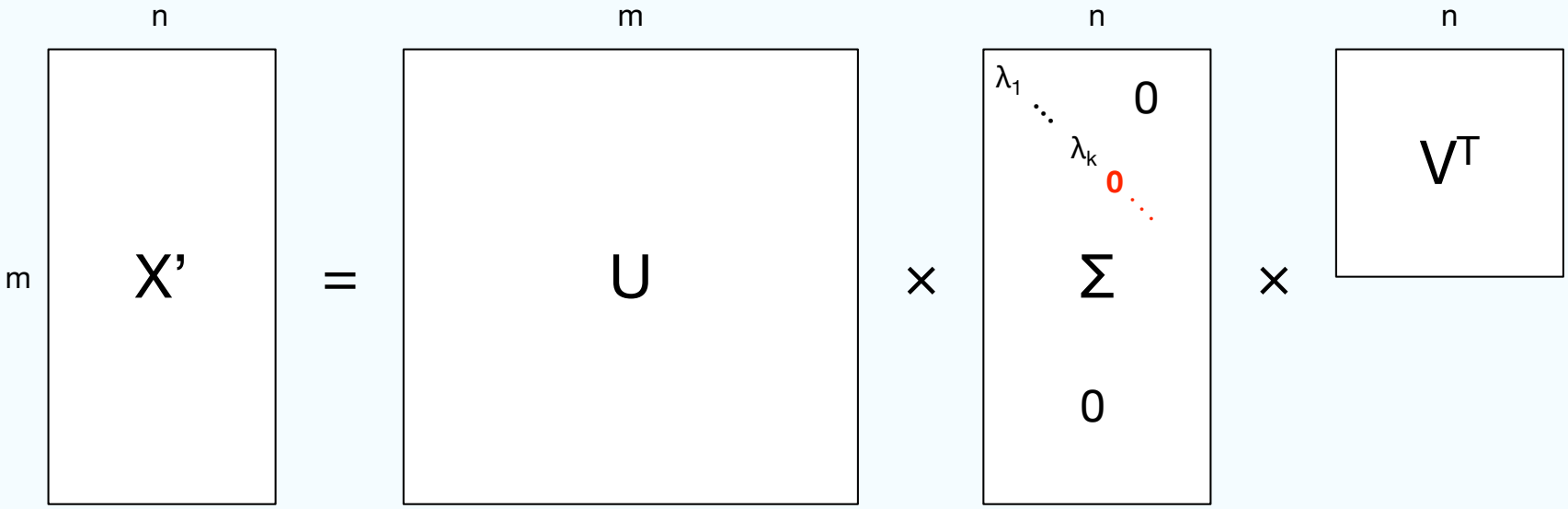
  - The diagonal elements of Σ is given by λ_i. and assume that they are in decreasing order.
  - Consider to approximate X by a lower dimension $k < n$, the approximation X' can be given by

$$X' = U\Sigma' V^T$$

    where Σ' is the same as Σ except $\sigma_{ii} = 0$ if $i > k$.

PCA

- **Dimension reduction**
  - By definition, all $\lambda_i > 0$. Let λ_i be in decreasing order.
  - Ignoring smaller λ_i $(i > k)$, X' will be an approximation of X.



Sequential Data Analysis

（系列データ解析）

SDA

- **Modeling time evolution**
  - Filters
  - Fourier Transform, Wavelet
  - Gaussian Process
  - Markov Chain
  - Ordinal differential equations, Stochastic differential equations
  - Data assimilation etc…

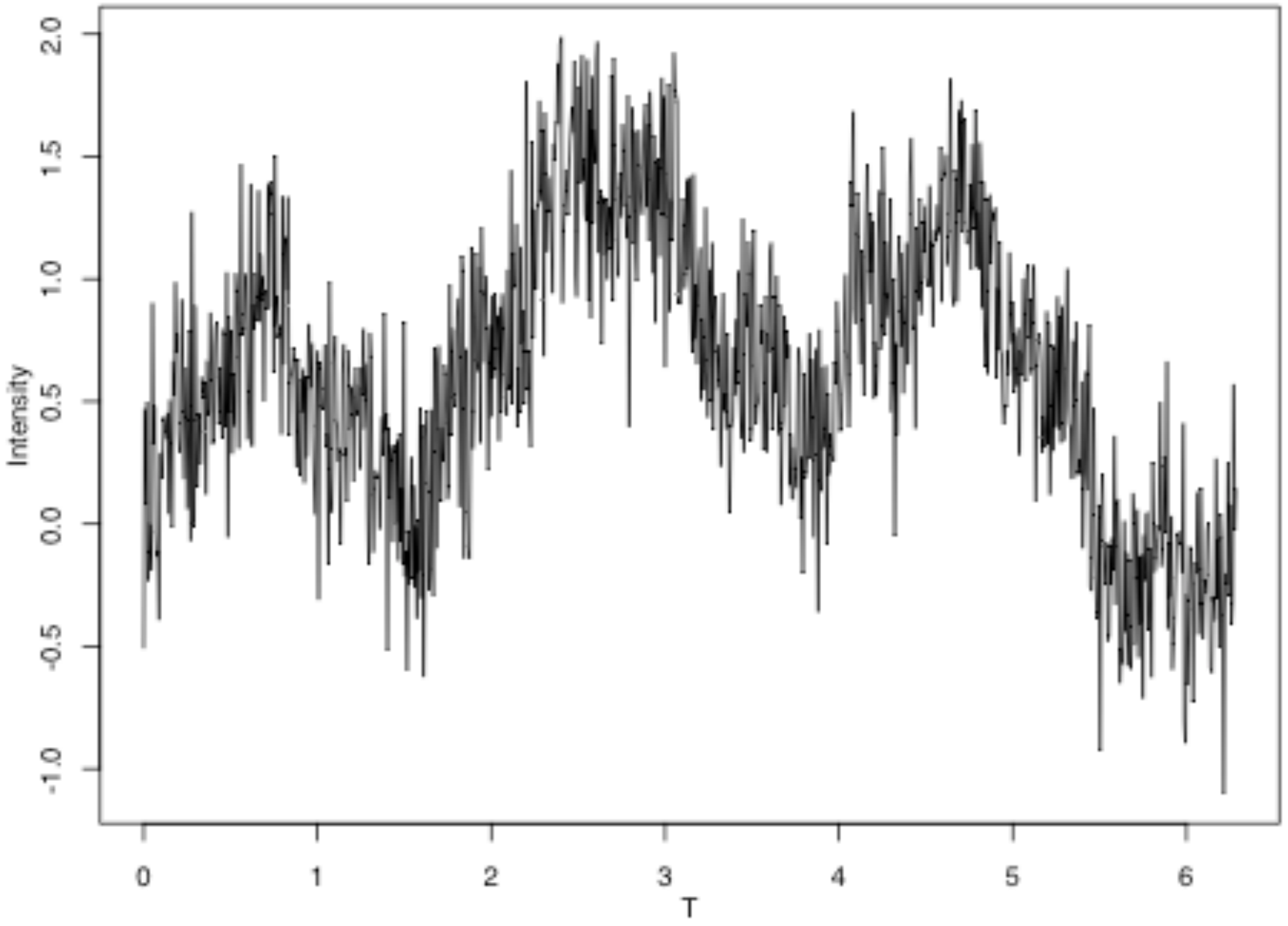
SDA

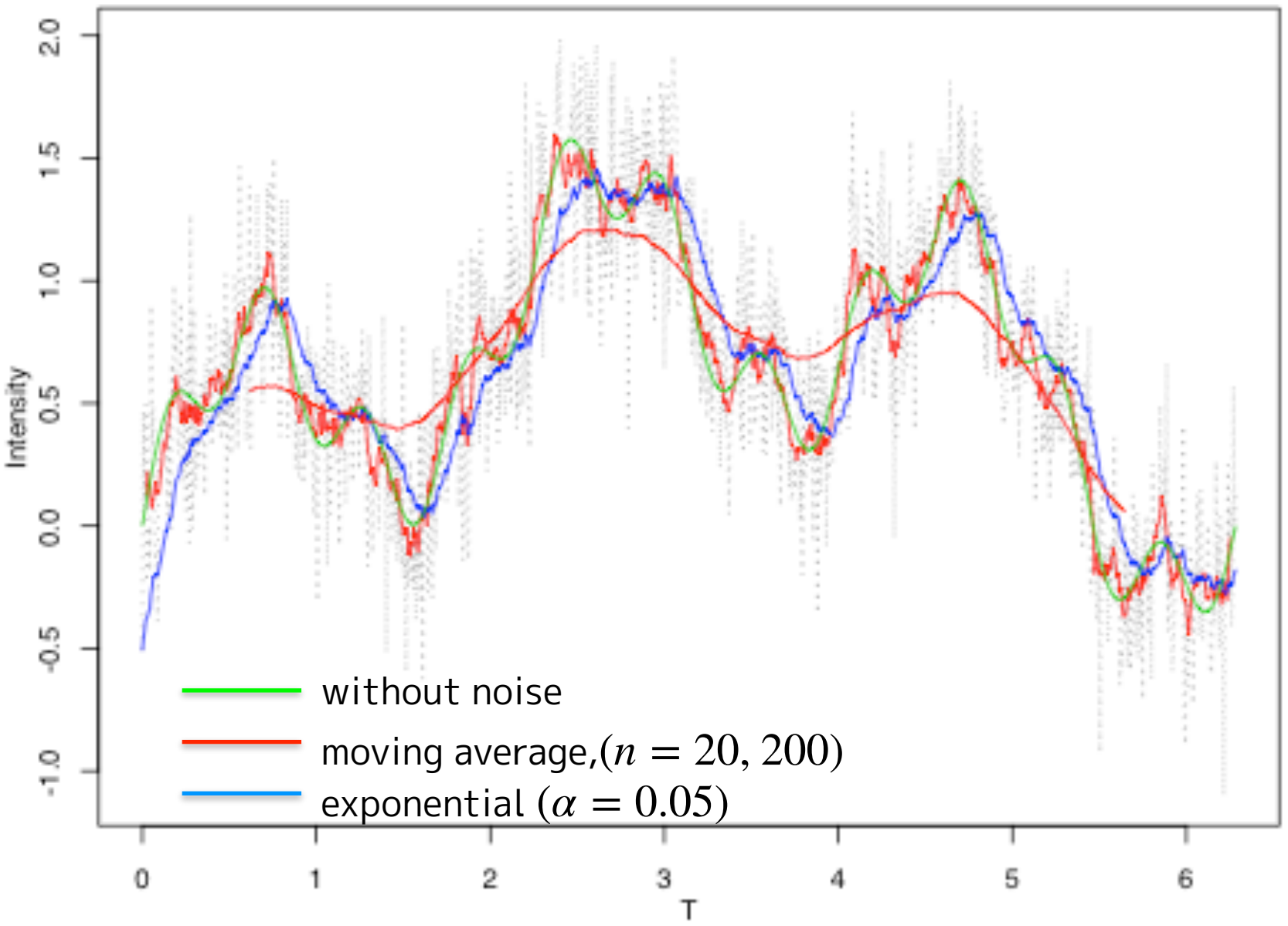
- **Moving Average Filter**
  - When the time series is noisy, moving average would reduce the effect of random fluctuation.

$$x_{MA}(t) = \frac{1}{n}\sum_{\tau=0}^{n-1} x(t-\tau)$$

- **Exponential Average Filter**

$$x_{EA}(t) = \lambda x(t) + (1-\lambda)x_{EA}(t-1)$$


Intensity
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
0
1
2
3
4
5
6
T

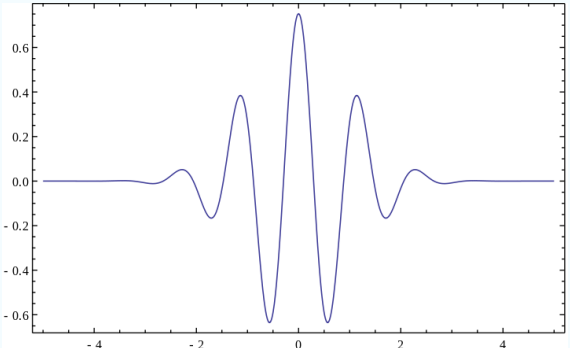

Intensity
2.0
1.5
1.0
0.5
0.0
-0.5
-1.0
0
1
2
3
4
5
6
T
without noise
moving average,($n = 20, 200$)
exponential ($\alpha = 0.05$)

SDA

- **Feature Analysis**
  - Fourier Transform

    Compute power dependance on wave length, i.e., to find **periodic patterns** by convolution with triangular functions.
  - Wavelet Transform

    Like FT, but consider dependance on both wave length and position using **wavelet mother functions.**

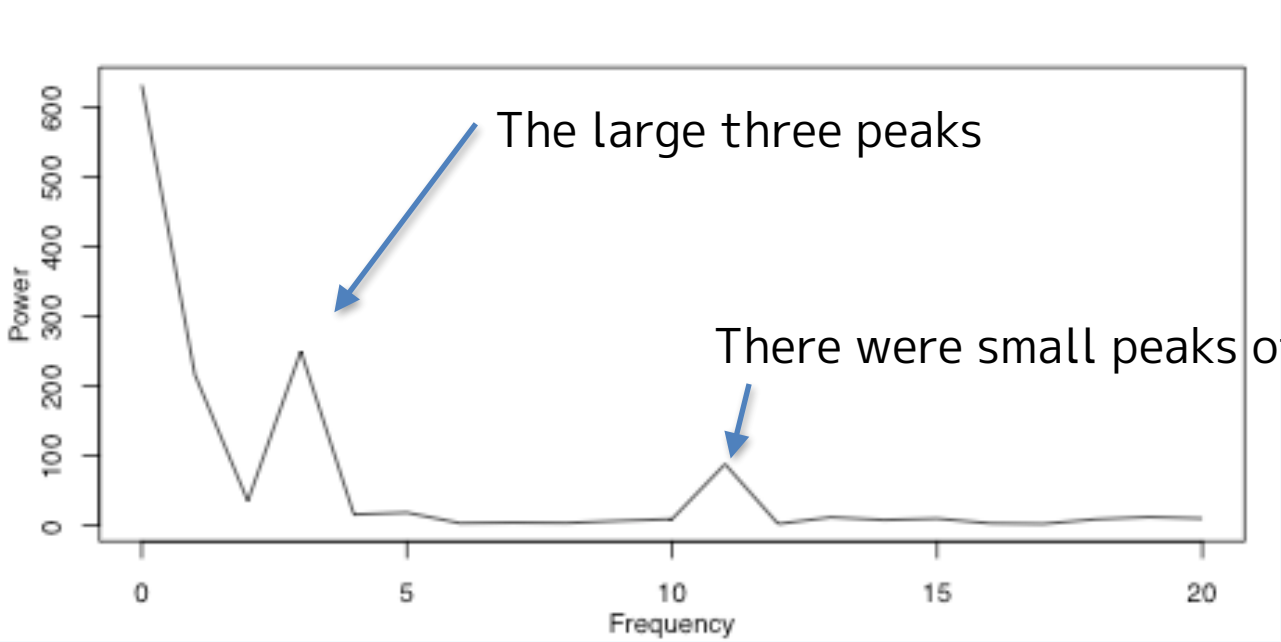


SDA

- **Fourier transform**

Continuous : $F(k) = \int f(x) e^{-2\pi i k x} dx$

Discrete : $F_k = \frac{1}{N} \sum f_j e^{-2\pi i j k / N}$

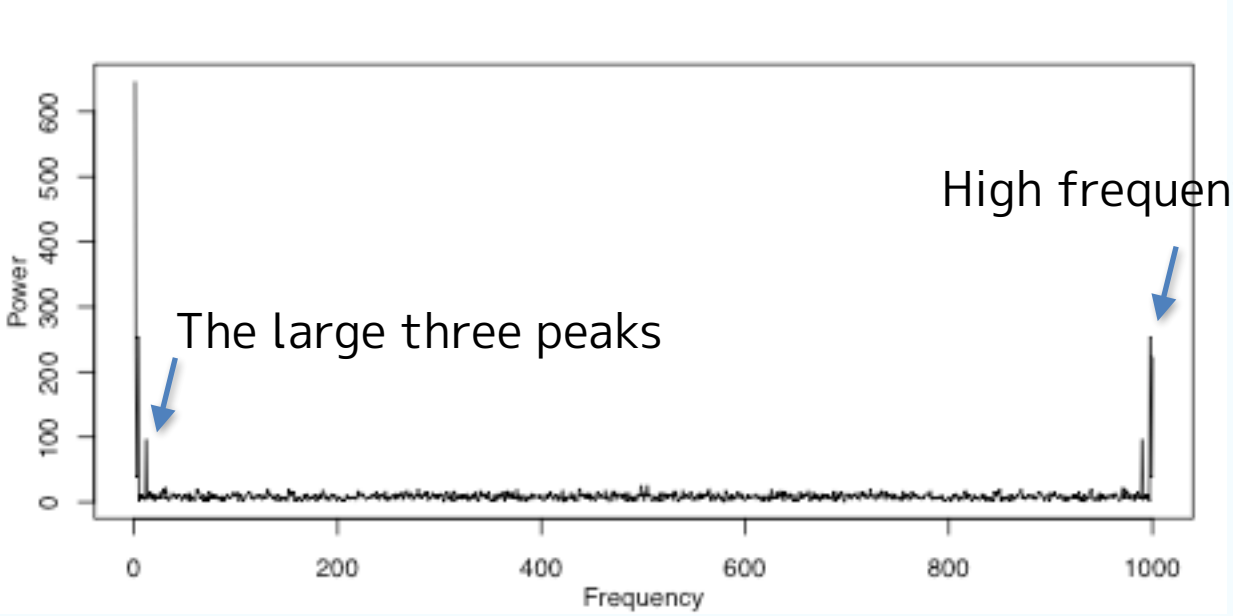


SDA

- **Fourier transform**

$$\text{Continuous} : F(k) = \int f(x) e^{-2\pi i k x} dx$$

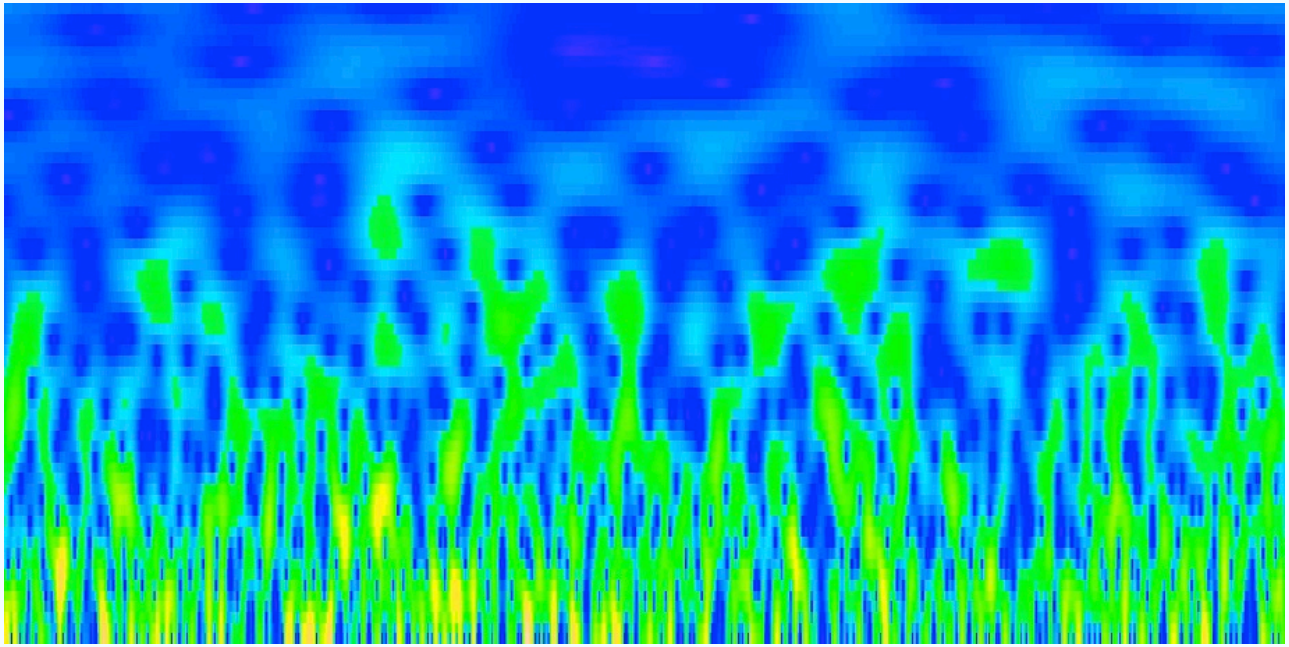
$$\text{Discrete} : F_k = \frac{1}{N} \sum f_j e^{-2\pi i j k / N}$$



SDA

- **Wavelet Transform**

$$\Phi(k, x') = \int \frac{1}{\sqrt{k}} f(x) \phi(\frac{x - x'}{k}) dx$$



The X and Y axis denote position of waves and wave length.
The hue from blue to white represents the power at (k, T)

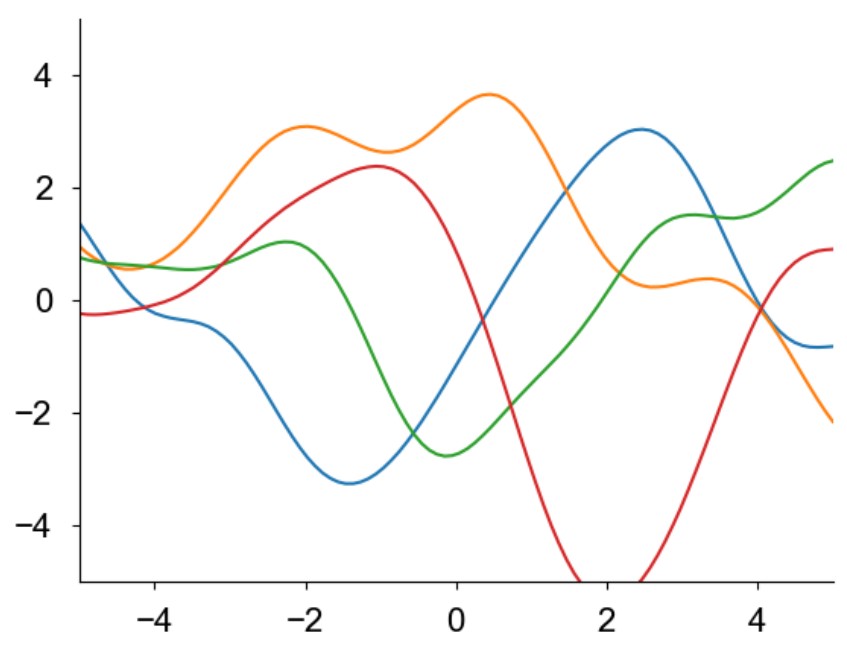
SDA

- **Gaussian Process (GP)**
  - Consider a generalized linear regression using Radial Basis Function (RBF), i.e., Gaussian.

$$f(x) = \int \phi(x') e^{-\frac{(x - x')^2}{\sigma^2}} dx'$$

GP is a probabilistic model of $f(x)$.

Assuming a prior distribution of $\phi(x)$, we can model a wide range of samples of $f(x)$ that belongs to the GP.



Samples from the GP. The smoothness of the curves depends on the σ.

SDA

- **Gaussian Process Regression**
  - Given a set of observation X, Y, we can evaluate posterior distribution of the GP using a kernel trick.
  - The posterior of y at a given x' is a Gaussian, as follows

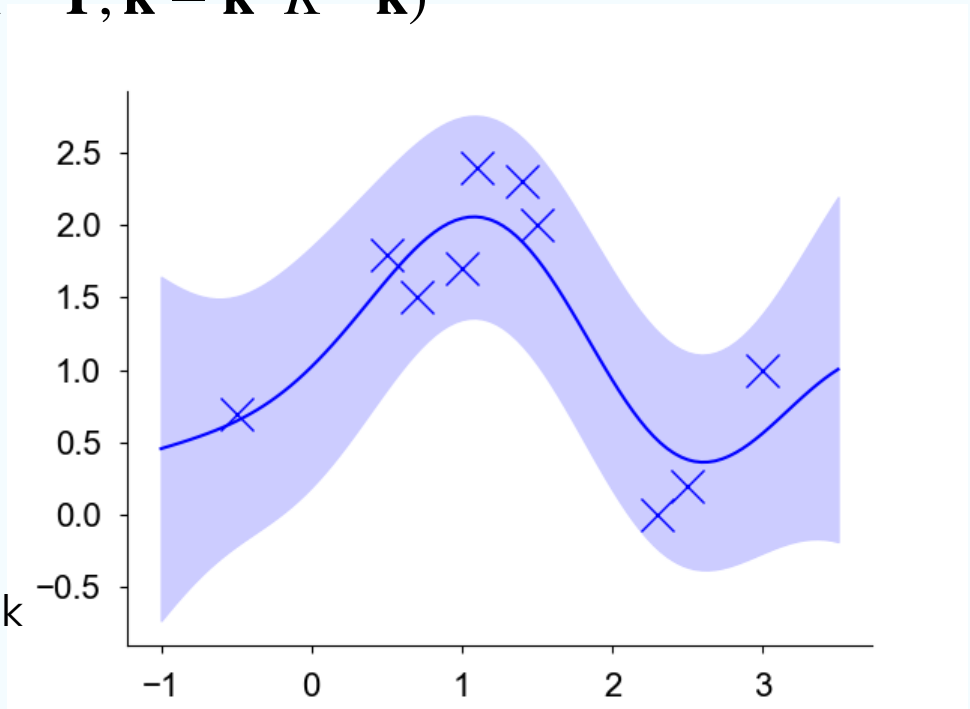
$$p(y(x \mid x') \mid \mathbf{X}, \mathbf{Y}, \theta) \sim N(\mathbf{k}^T K^{-1} \mathbf{Y}, \mathbf{k} - \mathbf{k}^T K^{-1} \mathbf{k})$$

where

$$k(x_1, x_2) = e^{-\frac{(x_1 - x_2)^2}{\theta^2}}$$

$$K = k(\mathbf{X}, \mathbf{X})$$

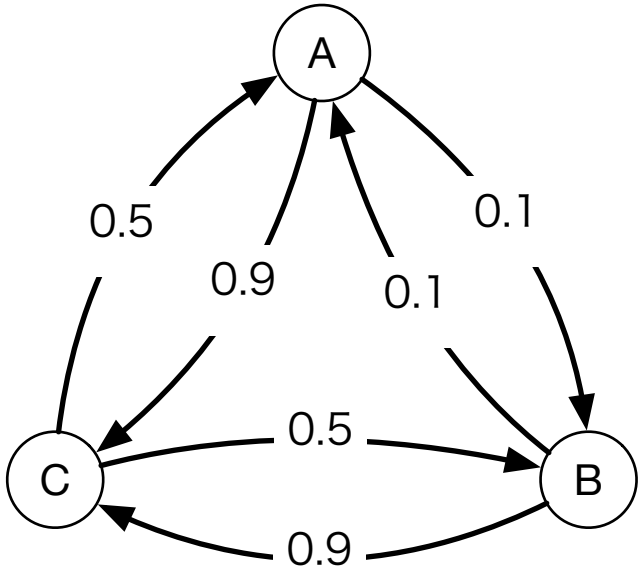
$$\mathbf{k} = k(x', \mathbf{X})$$



An example of GP regression. The blue line is the average and the dark region shows SD of the prediction

SDA

- **Markov Chain**
  - Memoryless discrete random process between finite states.
  - Defined by a transition probability matrix.



$$P = \begin{bmatrix} 0 & 0.1 & 0.9 \\ 0.1 & 0 & 0.9 \\ 0.5 & 0.5 & 0 \end{bmatrix}$$

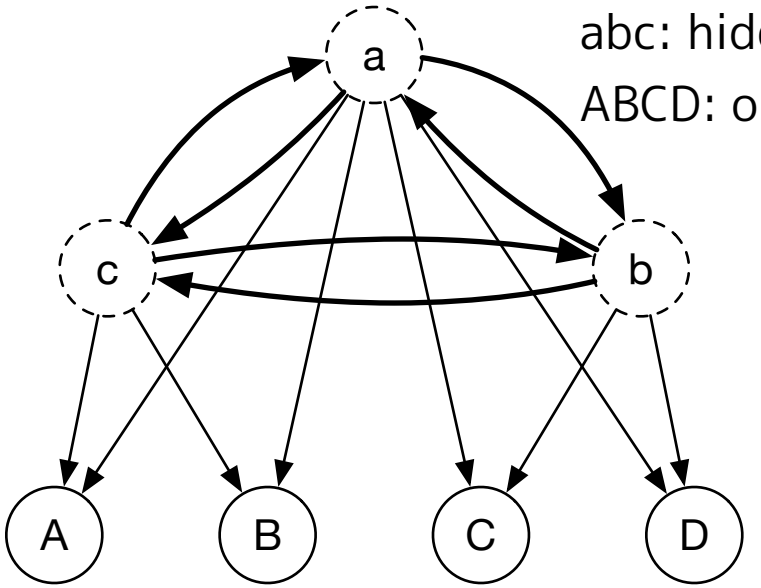
Sample State Sequence: A,C,A,B,C,B,C,B,C,B,...

SDA

- **Hidden Markov Model**
  - Markov models with unobservable ("hidden") states.
  - Observations Y depends on the hidden states X.

$$P(Y) = \sum_{X} p(Y|X)p(X)$$

Hidden states and their transition probability matrix can be estimated from the sequence of observations using Bayesian inference.



abc: hidden states

ABCD: observations

Deep Learning （深層学習）

Deep Learning

- DL is a one of the architectures of artificial neural networks.

- **Artificial Neural Networks**
  - Perceptron
  - Three Layer ANN
  - Deep Neural Networks
  - Convolutional Neural Networks
  - Recurrent Neural Networks

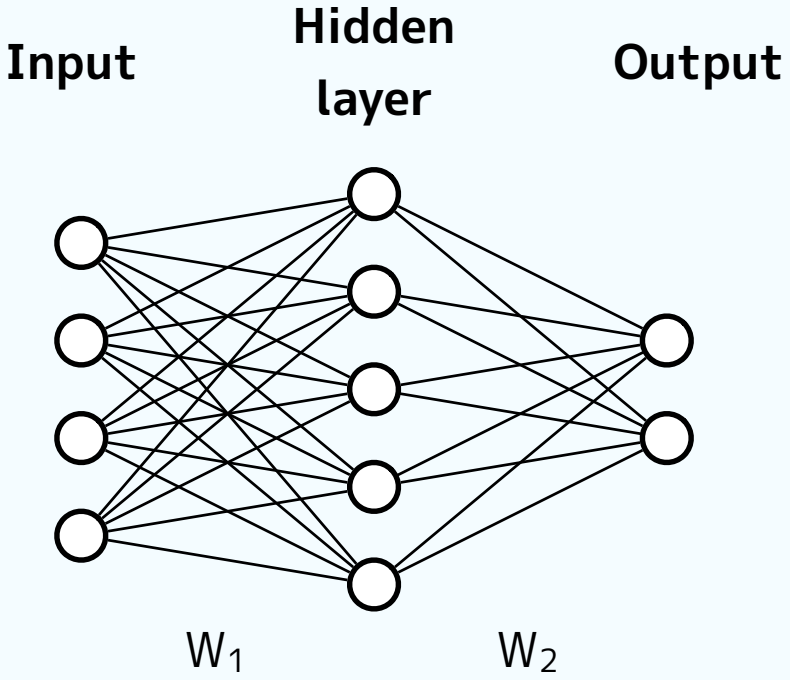
Deep Learning

- **Perceptron**
  - The output of a layer in perceptron is given as follows,
  - $\mathbf{x}_{l+1} = W_l \mathbf{x}_l + \mathbf{b}_l$

    $\mathbf{x}_l$ is the input, W_l is the weight matrix, and $\mathbf{b}_l$ is the bias.
- **Artificial Neural Networks**
  - $\mathbf{x}_{l+1} = f\left(W_l \mathbf{x}_l + b_l\right)$
  - Generally some non-linear activation function is applied to the output of ANN.

Deep Learning

- **Artificial Neural Networks**
  - A simplest architecture of an artificial neural network is three layer neural networks,



x_1: Input

$v_2 = W_1 x_1 + b_1$

$x_2 = f(v_2)$: Hidden

$v_3 = W_2 x_2 + b_2$

$x_3 = f(v_3)$: Output

Deep Learning

- **Training**
  - We need to optimize the weight parameters to obtain desirable models.
  - Minimize "loss" function, for example,

    Mean squared error: $L_2(\mathbf{y}, \mathbf{y}') = \frac{1}{N} |\mathbf{y}' - \mathbf{y}|^2$

    Cross entropy: $L_X(\mathbf{t}, \mathbf{p}) = -\sum t_i log(p_i)$
  - Gradient descent

    The most general stepwise optimization method.

    $W^{t+1} = W^t - \lambda \nabla L(\mathbf{X}; W^t)$

Deep Learning

- **Back propagation**
  - To optimize all W_l, we can compute their derivative using the **chain rule**, though it looks complicated …

$$\frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial \mathbf{y}_n}\frac{\partial \mathbf{y}_n}{\partial W_n} = \frac{\partial L}{\partial \mathbf{y}_n}\frac{\partial \mathbf{y}_n}{\partial \mathbf{x}_n}\frac{\partial \mathbf{x}_n}{\partial W_n} = L'(y_n)f'(x_n)W_n(\mathbf{x}_{n-1})$$

$$\frac{\partial L}{\partial W_{n-1}} = \frac{\partial L}{\partial \mathbf{y}_n}\frac{\partial \mathbf{y}_n}{\partial \mathbf{x}_n}\frac{\partial \mathbf{x}_n}{\partial \mathbf{y}_{n-1}}\frac{\partial \mathbf{y}_{n-1}}{\partial \mathbf{x}_{n-1}}\frac{\partial \mathbf{x}_{n-1}}{\partial W_{n-1}} = \sum_i L'(y_n)f'(x_n)W_n x_{n-1,i} f'(x_{n-1})W_{n-1}(x_{n-2})$$

$$\vdots$$

Deep Learning

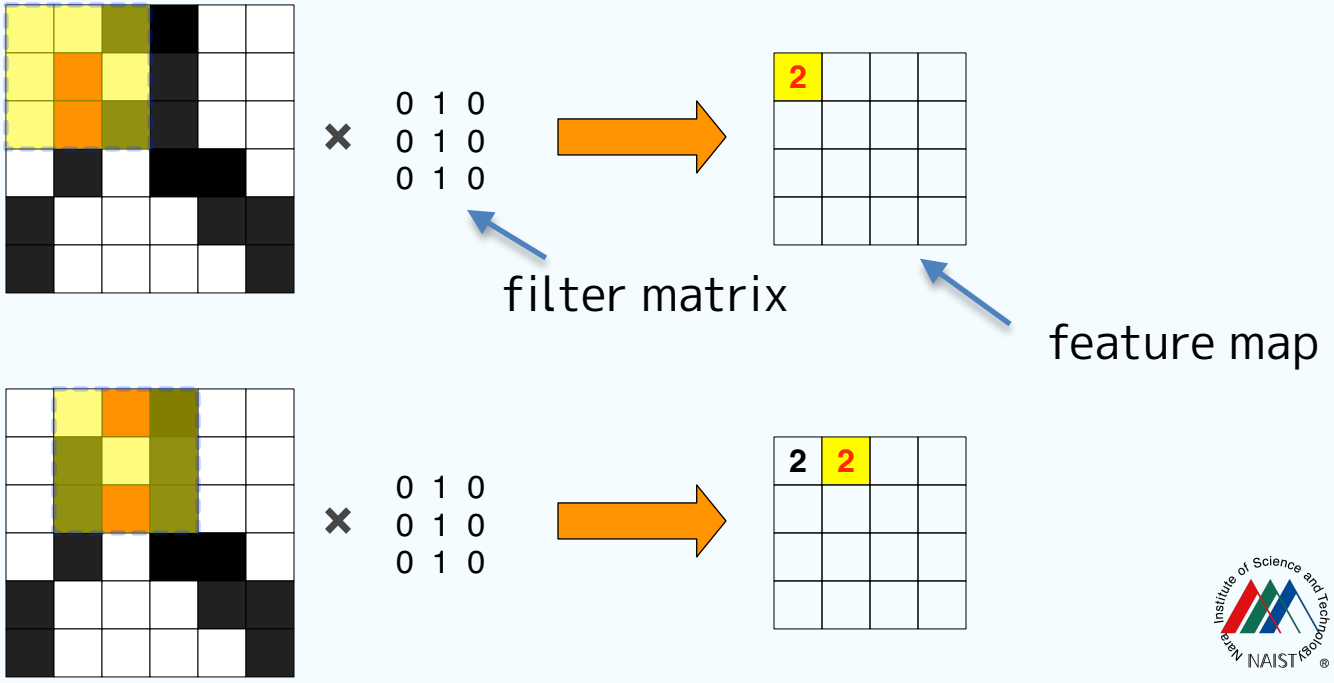
- **Three layers architecture of ANN**
  - Many layers ANN implies smaller gradients.

    > optimization will become difficult and slower.
  - Theoretically, three-layers ANN can compute arbitrary boundaries. (if the number of nodes are large enough)

    > Deeper ANN had not been studied so much.
- **Deep Learning**
  - Recently, "Big Data" and higher computational power allow us to train more complex ANNs.

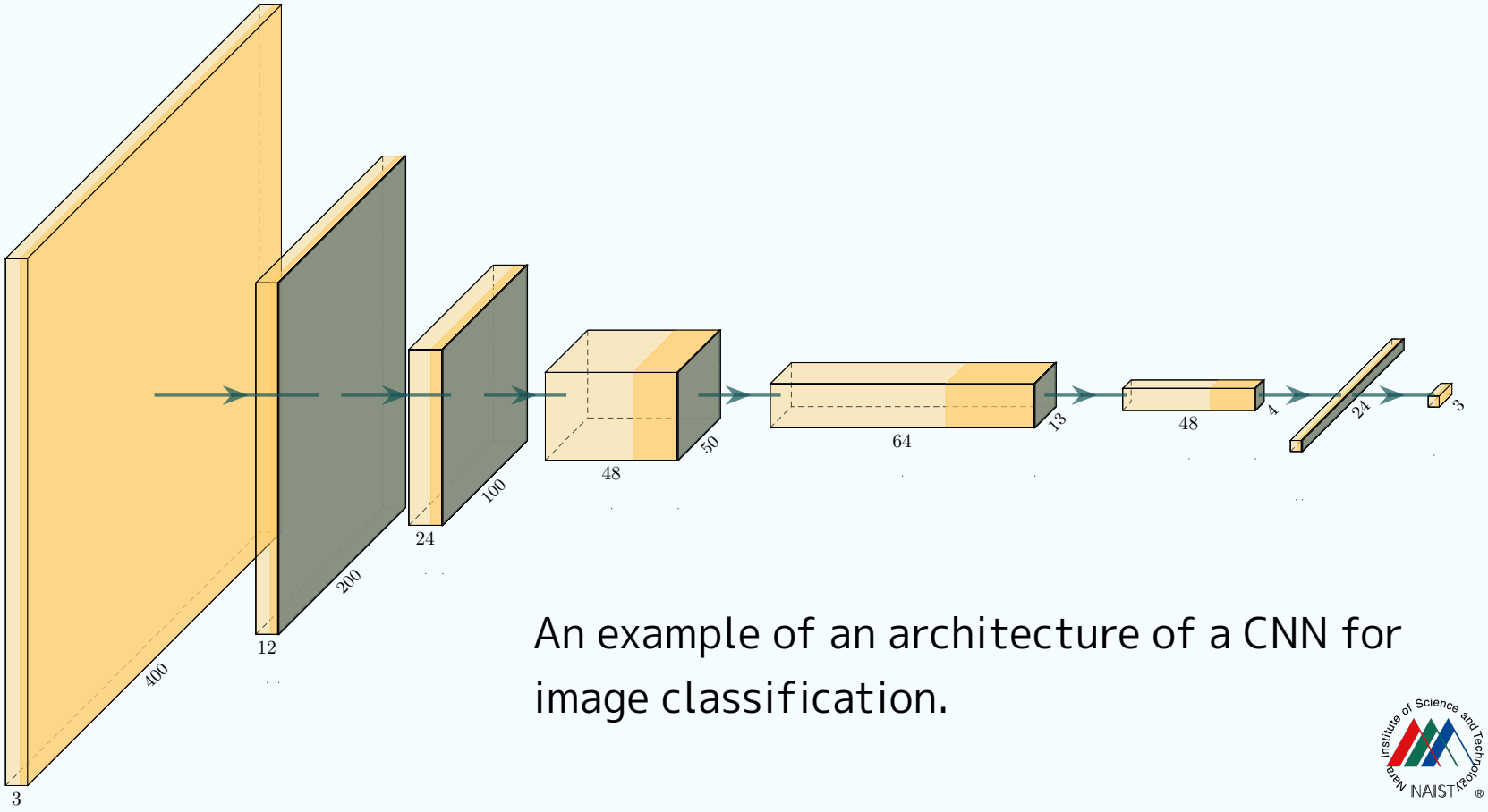
Deep Learning

- **Convolutional Neural Networks**
  - Like as wavelet, convolution using weighted "filter" extracts local feature from given images.
  - Optimizing recursive filters allows us to obtain more complex features.



CNN

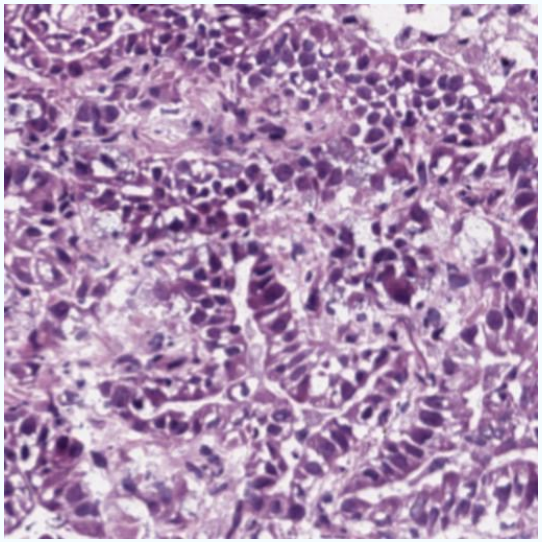
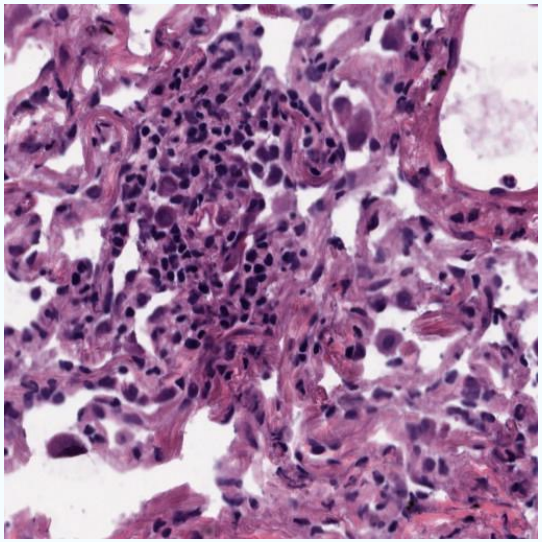
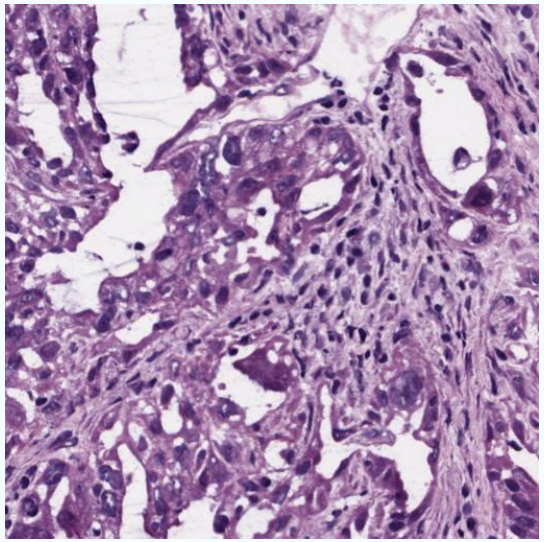
- **Image classification using CNN**
  - Extracting features from pixel matrix using layers of convolution and converged to the output nodes.



An example of an architecture of a CNN for image classification.

CNN

- **Image classification using CNN**
  - CNN can distinguish three sub-types of lung cancer from pathological images with 98.9% accuracy.

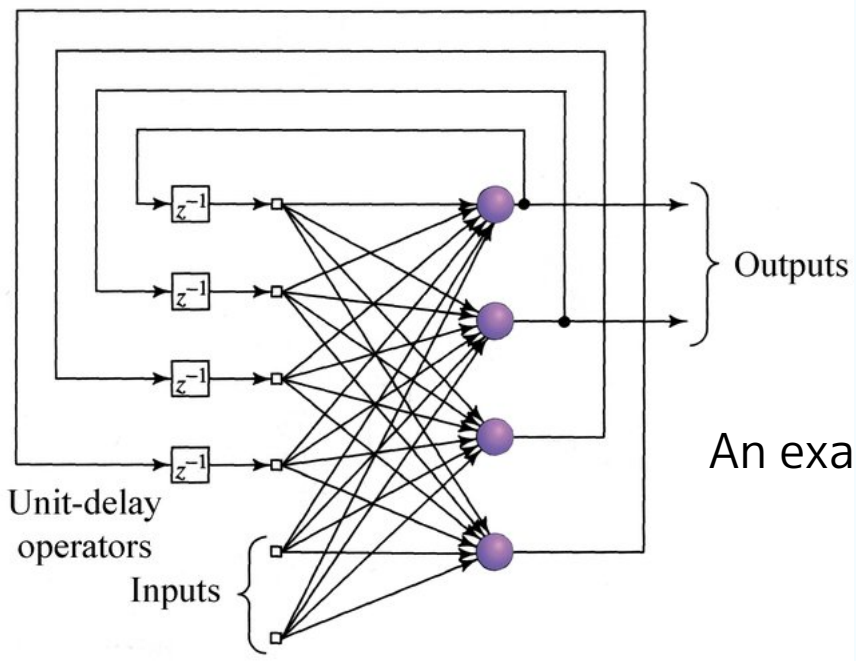




Sections of pathological images of thee subtypes of lung cancer.

[Antonio et al. IJCARS 2018]

Deep Learning

- **Recurrent Neural Networks**
  - Concatenating some output with the next input, an ANN can memorize its inner states.
    > Useful to extract features from sequential data.



An example of an architecture of an RNN

Deep Learning

- **Application of Deep Neural Networks**
  - Classifier

    Train to output discrete target variables.
  - Regressor

    Train to output continuous target variables.
  - AutoEncoder

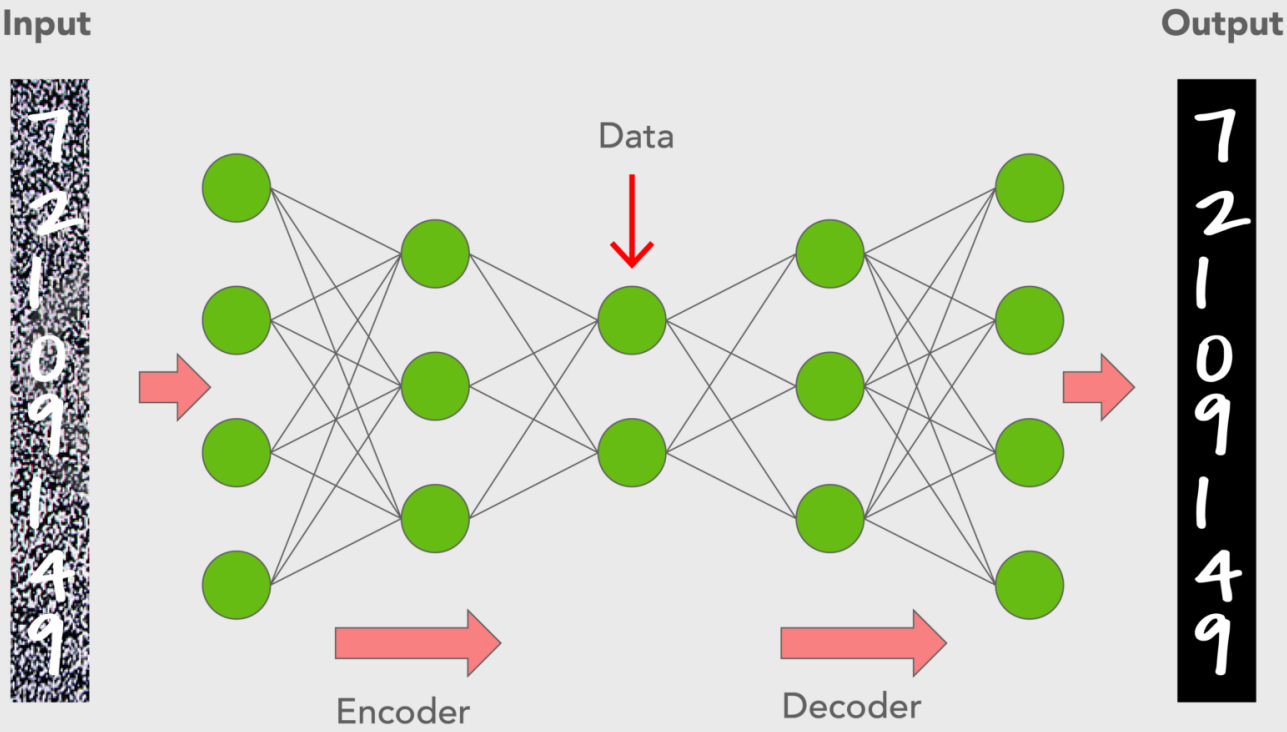
    Train to reconstruct the input using smaller features.
  - Converter

    Train to generate output from extracted features.
  - Generator

    Train to generate from some small parameters.

Deep Learning

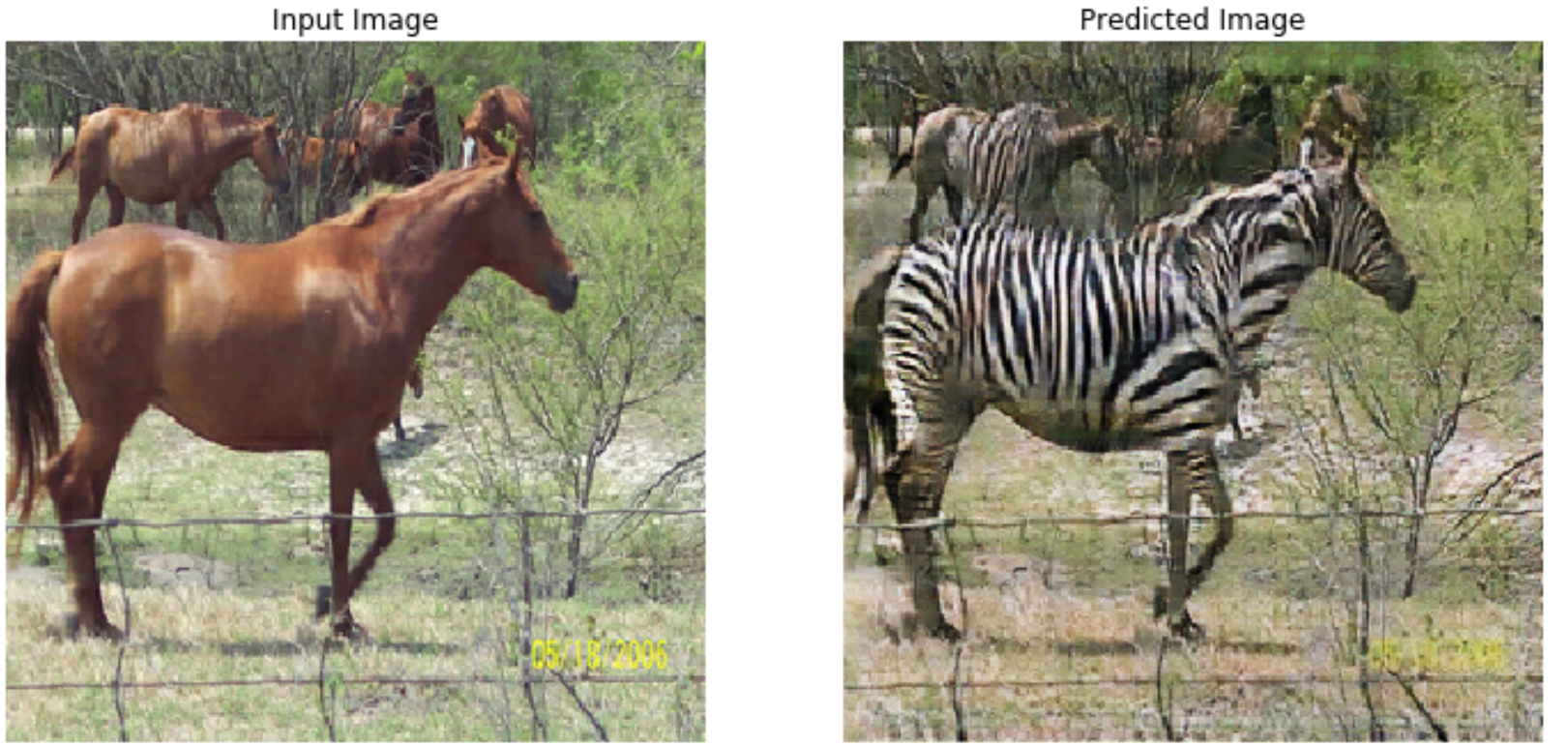
- **Autoencoder**
  - Train both encoder and decoder to reconstruct inputs.



A schematic illustration of an architecture of a Denoising autoencoder.
By training to extract feature of digits, it can reduce unrelated noises.

Deep Learning

- **Converter**
  - Train encoder and decoder to construct new images.



An example of image translation using CycleGAN.

[https://arxiv.org/abs/1703.10593]

Deep Learning

- **Generator**
  - Construct a decoder from latent space.

Generated images using StyleGAN

[NVIDIA research paper at ICLR 2018]

Model Selection

Model Selection

- **Measure of distribution**
  - Evaluate distance between true sample distribution ($p(x)$) and predicted probability distribution ($q(x)$).
  - Distance (L2):

$$D(p,q) = \int |p(x) - q(x)|^2 dx$$

  - Kullback–Leibler divergence:

$$KL(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx$$

both D and KL become minimum and equal to 0 iff $p(x) \equiv q(x)$

Model Selection

- **Kullback–Leibler divergence:**

$$KL(p||q) = \int p(x)\log\frac{p(x)}{q(x)}dx$$

$$= \int p(x)\log p(x)dx - \int p(x)\log q(x)dx$$

(negative) Entropy of $p(x)$ Mean log likelihood

When $p(x)$ is the distribution of sample X,

$$= -S(X) - E_X[L(q)] \qquad x \in X$$

where $S(X)$ is the entropy, $L(q)$ denote log likelihood,

$E_X[]$ is the expected value sampled from X.

Model Selection

- **Mean Log Likelihood**
  - Consider a model with parameters $q(x;\alpha)$

    When the model is the best, $KL(p||q(\alpha))$ is minimum.

    Though generally, we don't know $p(x)$ to estimate KL.

$$KL(p||q(\alpha)) = -S(X) - E_X[L(q(\alpha))]$$

S does not depend on α

Maximize MLL

And according to the "law of large numbers",

$$E_X[L(q(\alpha))] = \lim_{N\to\infty} \frac{1}{N} \sum_i \log q(x_i;\alpha)$$

We no longer need to know $p(x)$, if the sample size is large enough.

Model Selection

- **Information Criteria**
  - We can approximately evaluate MLL and find the parameter that maximize it.

    Note: When we estimate α from the observed samples, there remains a **bias** that reduce L compared to the true distribution.
  - Akaike Information Criteria

$$AIC = -2\log(\hat{L}) + 2k \sim -2MLL$$

  - Bayesian Information Criteria

$$BIC = -2\log(\hat{L}) + k\log n \sim -2MLL$$

The constant "-2" derived from a historical background.

Model Selection

- **Leave One Out Cross Validation**
  - LOOCV can be regarded as an approximation of MLL.
    > Smaller LOOCV is equivalent to smaller MLL.
- **Model comparison**
  - Though we do not know $S(X)$, it is constant. We can compare relative difference of MLL, i.e., log likelihood ratio (LLR) **to select better model**.
  - Most of **statistical tests** can be regarded as comparison of LLR to that of "null model".

-log(p-value) generally correlates with LLR in statistical tests.

Afterwords

- **Modeling and Learning**
  - Why we need models?

    To predict future observation from past observed sample data.
  - What is "learning" in data science?

    Construct a model with adjustable parameters, then find their optimum to improve prediction accuracy.
  - How can we know which modeling is better?

    By measuring difference between predicted and observed probability distribution.

Report

Report

- **Task**
  - You have thousands samples of multivariate (less than 100 dimensions) observations. Discuss about a method how to separate 'noise' component from the data and how to evaluate appropriateness of the method.

    Hint: Propose arbitrary models (either linear or non-linear).
  - **Words:** less than 500 words (about A4 1page)
  - **Language:** English or Japanese (日本語でも可)

    Note: You may refer textbooks and web pages, and must cite all references properly.

The word count does not include the references.

Report

- **課題**
  - 数千件、数十次元の多次元のサンプルデータの配列があるものとする。このデータから「ノイズ成分」を分離する方法について、またその方法を評価する手法について議論せよ。

    ヒント: なんらかの具体的なモデルを（線形でも非線形でも）提案し、そのモデルに基づいて考察すること。
  - **分量:** 1500 字 (A4 1ページ) 以下
  - **言語:** 日本語でも可

    注: 教科書やwebサイトを参考にしても構わないが、参照した文献をすべて適切に引用すること。

    （引用及び図表は分量に含まない）

Report

- **Submission**
  - Open the NAIST online syllabus.

    https://syllabus.naist.jp/subjects/preview_detail/621
  - Find the link to the DropBox and open it.

    Password: naist2020ds
  - Drop you file via your web browser.
  - **Name your report file as follows,**

    3_{StudentID}_{LastName}_{FirstName}.pdf

    (example: 3_0105072_Ono_Naoaki.pdf)
  - **Keep the deadline.**

    2020/6/26, 23:59