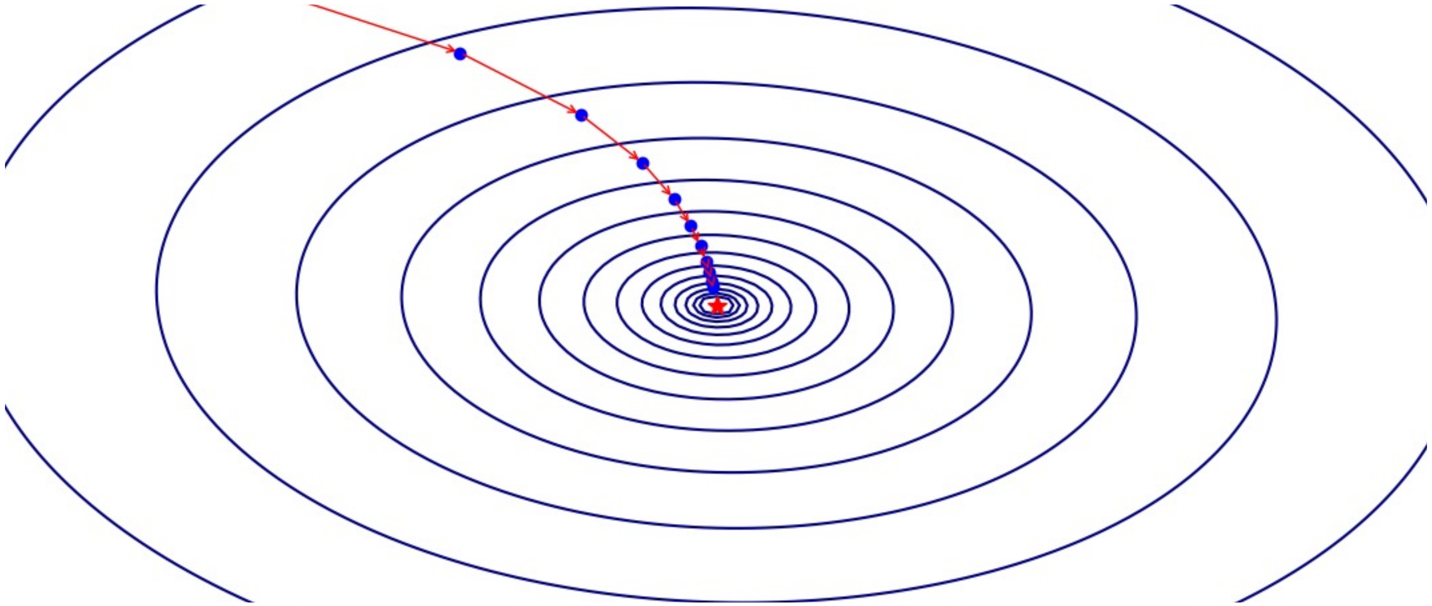


Machine Learning in Signal Processing

Winter Semester 2023/24

11. Dimensionality reduction

23.01.2024

Prof. Dr. Vasileios Belagiannis

Chair of Multimedia Communications and Signal Processing

Friedrich-Alexander-Universität
Technische Fakultät

Course Topics

1. Introduction.
2. Basics and terminology.
3. Linear regression.
4. Linear classification.
5. Performance evaluation.
6. Neural networks.
7. Deep neural networks.
8. Decision trees.
9. Ensemble Learning.
10. Clustering.
11. **Dimensionality reduction.**
12. Support vector machines.
13. Recap and Q&A.

- The exam will be written.
- Test (ungraded) around the middle of the lectures.
- We will have an exam preparation test before the end of the year.

Acknowledgements
Ideas and inspiration from:
- CSC311 Introduction to Machine Learning, University of Toronto.
- Introduction to Machine Learning: LMU Munich.
- Introduction to Machine Learning, CSAIL, MIT.
- CSE 574 Introduction to Machine Learning, University of Buffalo.
- Special thanks Arij Bouazizi, Julia Hornauer, Julian Wiederer, Adrian Holzbock and Youssef Dawoud for contributing to the lecture preparation.

Last Lecture Recap

- Unsupervised learning
- Clustering
- K-Means algorithm
- Mixture of Gaussians
- Expectation Maximization
- Agglomerative Clustering

Today’s Agenda and Objectives

- Dimensionality reduction.
- PCA.
- Autoencoder.
- Stacked Autoencoder.
- t-SNE.

Motivation

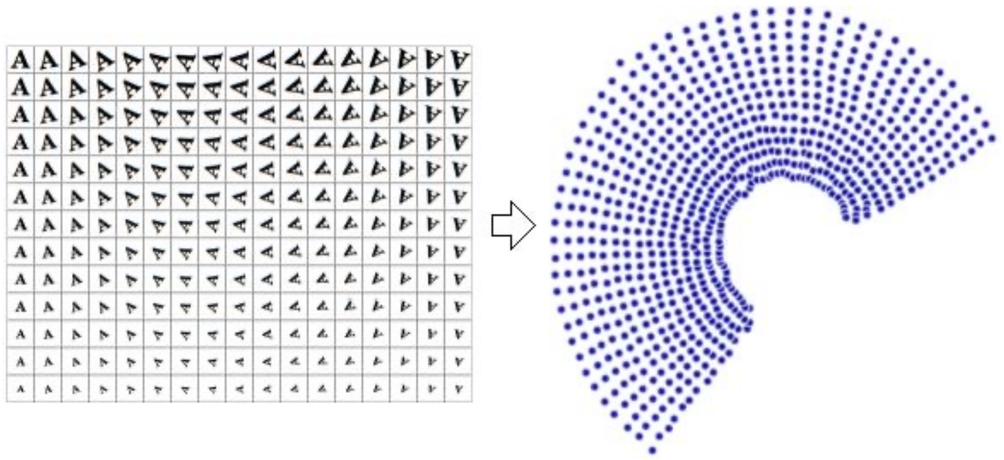
- Visual data, such as images, audio or video signals, are easy for us to perceive, but complex to be analyzed my machine learning algorithm.
- *What is the limitation of this type of data?*
- In general, high dimensional data can be:
  - Difficult for analysis and interpretation.
  - Expensive for storage and processing.
- Dimensionality reduction explores the data structure and correlations to allows us create a compact data representation of the data.
  - The ideal goal of dimensionality reduction is to reduce the data dimensions without losing information.
  - It is like data compression, e.g., compressing a PNG image to JPEG.

Non-linear Dimensionality Reduction

- We assume that the data lied dimensional non-linear manifold, within the higher dimensional space.
  - A dimensionality reduction algorithms aims to map the high-dimensional data to the low-dimensional manifold.
- We seek for an algorithm that performs non-linear dimensionality reduction (NLDR).
  - An autoencoder or the t-distributed stochastic neighbor embedding (t-SNE) are non-linear approaches for NLDR.
- Nevertheless, we can rely on a linear approach to reduce the data dimensions.
  - Principal component analysis (PCA) is a linear approach for dimensionality reduction.

Non-linear Dimensionality Reduction (Cont.)

- We can reduce grayscale images to a 2D-space.
  - Data reduction to the two-dimensional space (rotation and scale).
- By reducing the dimensions of the data to 1D, 2D or 3D, we can easily visualize them.



Public Domain, https://commons.wikimedia.org/w/index.php?curid=106259071

Principal Component Analysis

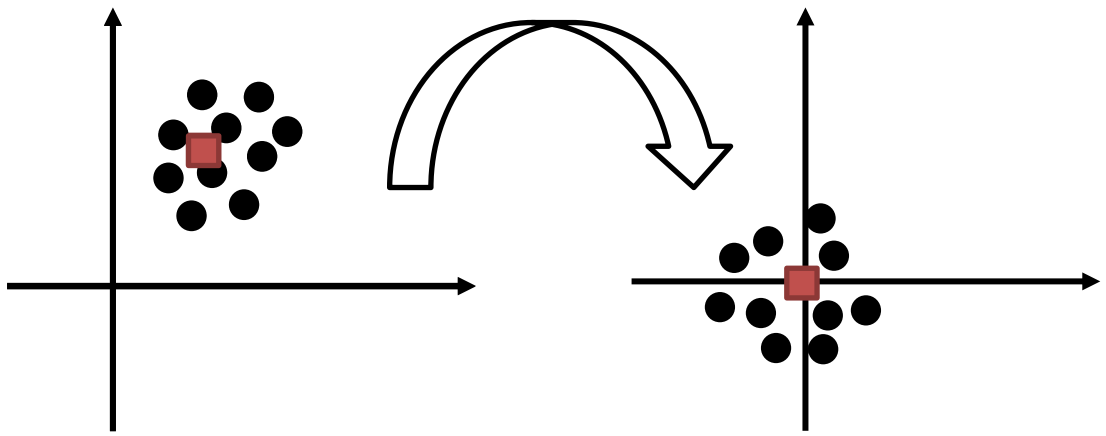
- Principal component analysis (PCA) is a linear transformation approach that we can use to reduce the dimensions of our input features.
- We assume the dataset $\{x_n\}_{n=1}^{N}, x_n \in \mathbb{R}^{D}$ with mean 0 and the covariance matrix $C = \frac{1}{N}\sum_{n=1}^{N} x_n x_n{}^{T}$.
- Furthermore, we assume that each sample x_n has the low-dimensional compressed representation $z_n = B^T \in \mathbb{R}^{M}$.
  - The projection matrix is $B = [b_1, \dots, b_M] \in \mathbb{R}^{D \times M}$ with orthonormal columns.
  - M are the projected data dimensions and thus $M < D$.
- We seek to find the lower dimensional projection for each sample x_n.
  - We look for a low-dimensional representation that retains the maximum available information and minimizes the loss from the projection.
- By transforming the dataset from the high-dimensional space to a lower dimensional space, the covariance between the new dimensions is 0.

Principal Component Analysis Algorithm

- Input: $\{x_i\}_{i=1}^{n}, x_i \in \mathbb{R}^D$
- Output: $\{z_i\}_{i=1}^{n}, z_i \in \mathbb{R}^M$ with $M \leq D$ and the projection matrix B.
- Algorithm:
  - Data centering (or standardization).
  - Covariance matrix computation.
  - Eigenvectors and eigenvalues computation.
  - Projection Matrix.

PCA: Center the Data

- Shift the points by their mean $\mu \in \mathbb{R}^D$ such that $\tilde{x}_i = x_i - \mu$
  with $\mu = \begin{bmatrix} \mu_1 \\ \vdots \\ \mu_D \end{bmatrix} = \frac{1}{D}\sum_{i=1}^{D} x_i$.
- This ensures that the dataset has mean 0.
- In addition, we could divide with the standard deviation.
  - We perform both operations, then we perform <u>standardization</u> ($\frac{x_i - \mu}{\sigma}$).



PCA: Covariance Matrix Computation

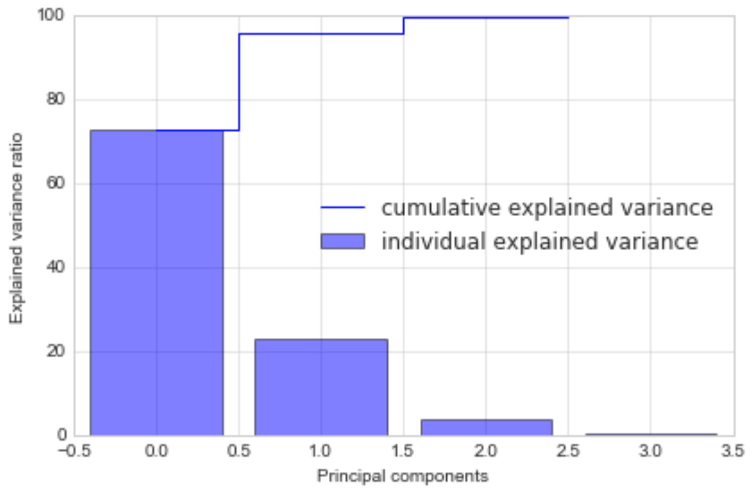
- Next, we compute the covariance matrix to find correlation between the different dimensions of our input dataset.
  - The covariance matrix C is a $D \times D$ symmetric matrix.
  - The main diagonal elements correspond to the variance of each feature dimension.
  - The rest elements represent the covariance between pairs of features.
  - For two features $f1, f2$, the covariance is computed as: $\sigma_{f1,f2} = \frac{1}{N}\sum_{i=1}^{N}(x_{i,f1} - \mu_{f1})(x_{i,f2} - \mu_{f2})$ with $\mu_{f1} = \frac{1}{N}\sum_{i=1}^{N} x_{i,f1}$ and $\mu_{f2} = \frac{1}{N}\sum_{i=1}^{N} x_{i,f2}$ the mean of the feature dimensions $f1$ and $f2$.
  - Positive covariance values show positive correlation, i.e., two variables increase / decrease together.
  - Negative covariance values show negative correlation, i.e., when one variable increases, the other decreases.

PCA: Eigenvectors and Eigenvalues Computation

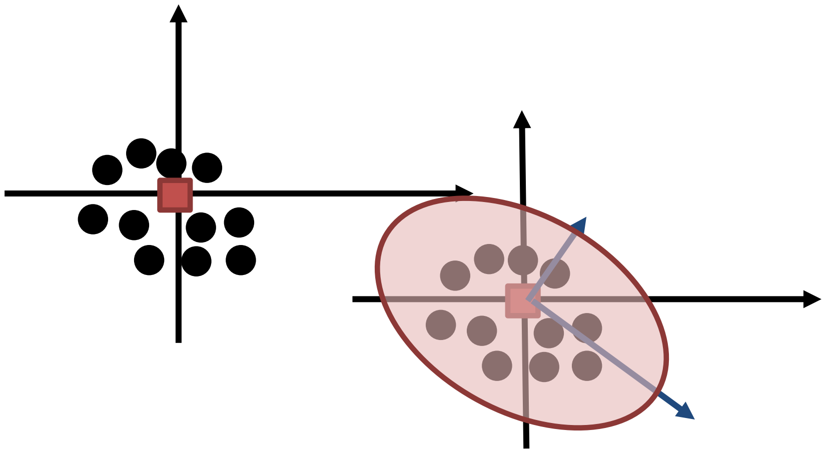
- We are going to compute the eigenvectors and eigenvalues from the covariance matrix to determine the <u>principal components</u> of our data.
  - The principal components are new features / variables that results from a linear combination of our original D dimensional features.
  - The principal components are uncorrelated.
  - Most of the information of our original D dimensional features is compressed to the first principal components.
  - For example, $D = 10$ will give a 10 principal components. With PCA, we aim to have the most information at the first principal component, the maximum remaining information in the second one and so on.
- By collecting the most information in the first components, we can drop the rest ones and thus compress our data.

PCA: Eigenvectors and Eigenvalues Computation (Cont.)

- Our original features can be interpretable, e.g., height and width features. However, the principal components are not interpretable since they are formed as a linear combination of the original features.
- We can think the principal components as the directions where the variance of the projected data is maximized.
  - Higher variance indicates more information towards the component direction.

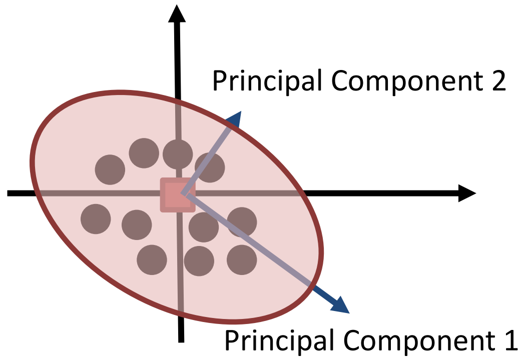


Source Code: https://github.com/rasbt/pattern_classification



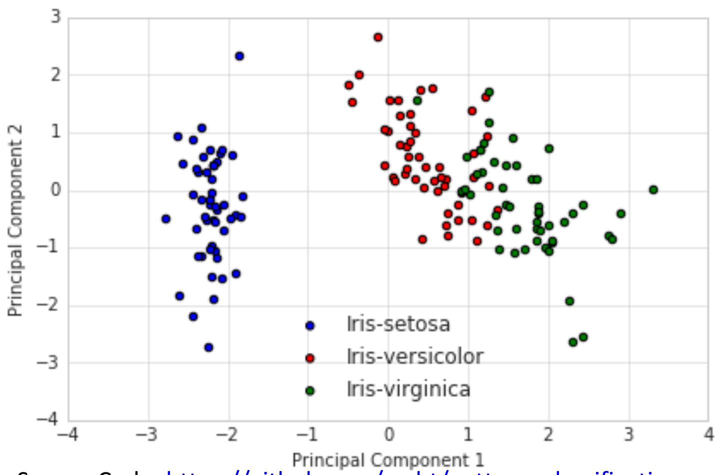
PCA: Eigenvectors and Eigenvalues Computation (Cont.)

- The first principal component shows direction with the highest variance in the data.
- The second principal component shows the direction with the second highest variance in the data.
  - It must be <u>uncorrelated</u> with the first principal component. This means that it must be perpendicular to the first one.
- The third principal component and every other is computed in the same way.
- In total, we compute D principal components.
- The <u>eigenvectors</u> of the covariance matrix C represent where the D highest variances and thus correspond to the principal components. The <u>eigenvalues</u> (paired with the eigenvectors) show the amount of variance. We can use them to sort the eigenvectors from the most to the least significant.
  - Having the ordered eigenvectors, we could then keep only the principal components with the most information and discard the rest ones.



PCA: Projection Matrix

- We have now the eigenvectors ordered from the most to the least significant. This is our feature vector that we will use to project our data to the principal subspace.
  - Note that we could also keep all the D eigenvectors if we do not aim to reduce the data dimensions.
- Finally, we project our data from the original dimensions (axes) to the principal components (axes). This operation is defined as $z_i = B^T x_i$ with the projection matrix B^T to contain the eigenvectors (column-wise).



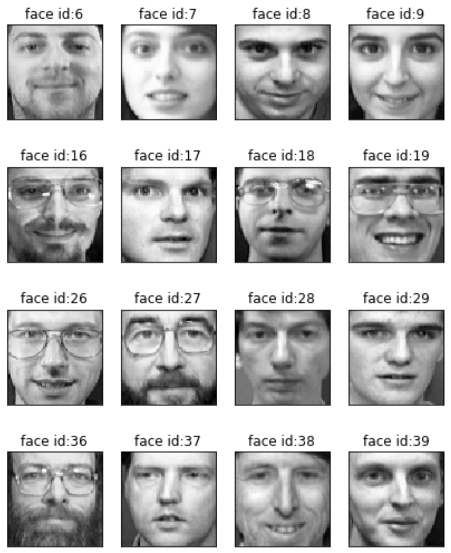
Source Code: https://github.com/rasbt/pattern_classification

PCA Observations

- It removed correlated features from our data.
- It is easy to understand and implement.
- It can reduce overfitting.
- It is often used for data visualisation.
- It can back-project data from the subspace to the input dimensions.
- *What are the two clear limitations of PCA?*

PCA in Practice

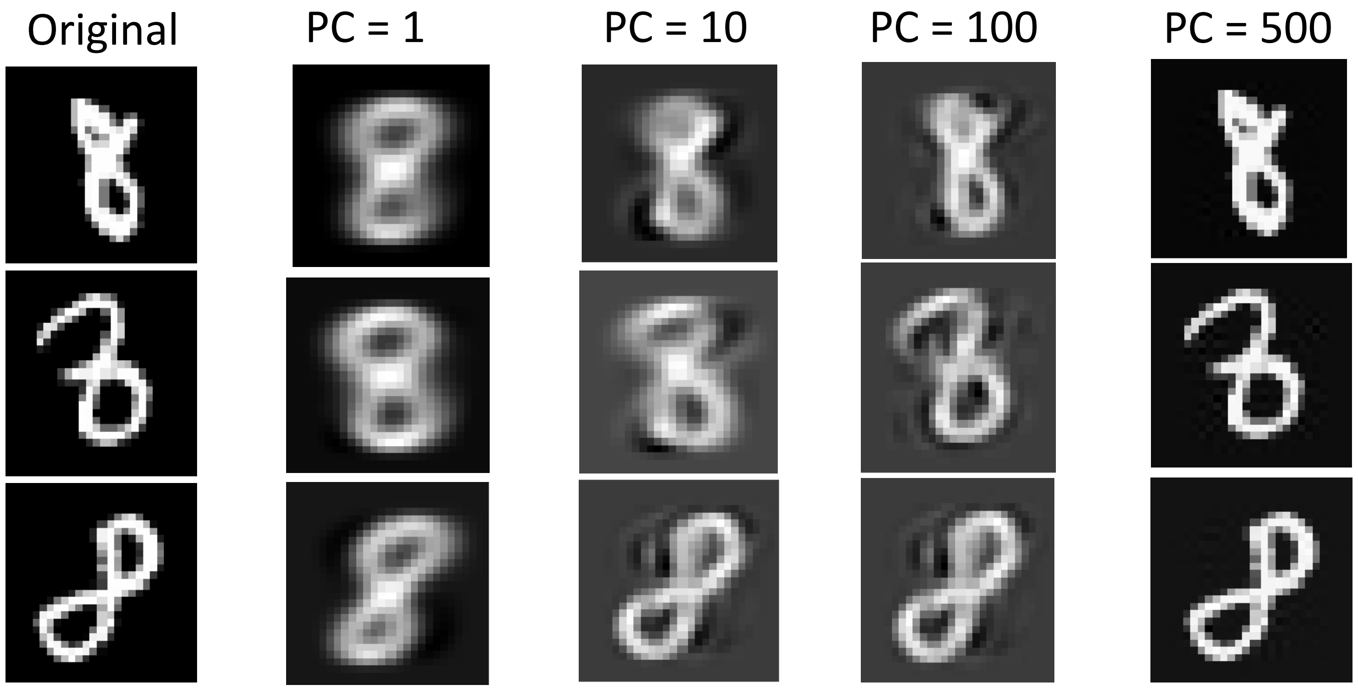
- Consider a database of faces, where our task is to match a face image to the database.
  - We have grayscale images of 100x100 resolution.
- We will use PCA to reduce the image dimensions.
- Then we can rely on the compact image representation (feature vector) to find the closest match in our database using nearest neighbour search as $min_k \left\| y_k^T - x \right\|$.



Source: https://www.kaggle.com/code/serkanpeldek/face-recognition-on-olivetti-dataset

PCA for Reconstruction

- We can use the projection matrix to reconstruct a set of images. The first column shows original digits from the test set. Following columns show reconstructions of these digits when using a principal subspace of dimensions 1, 10, 100, and 500, respectively.
- The image come from the MNIST database http://yann.lecun.com/exdb/mnist/

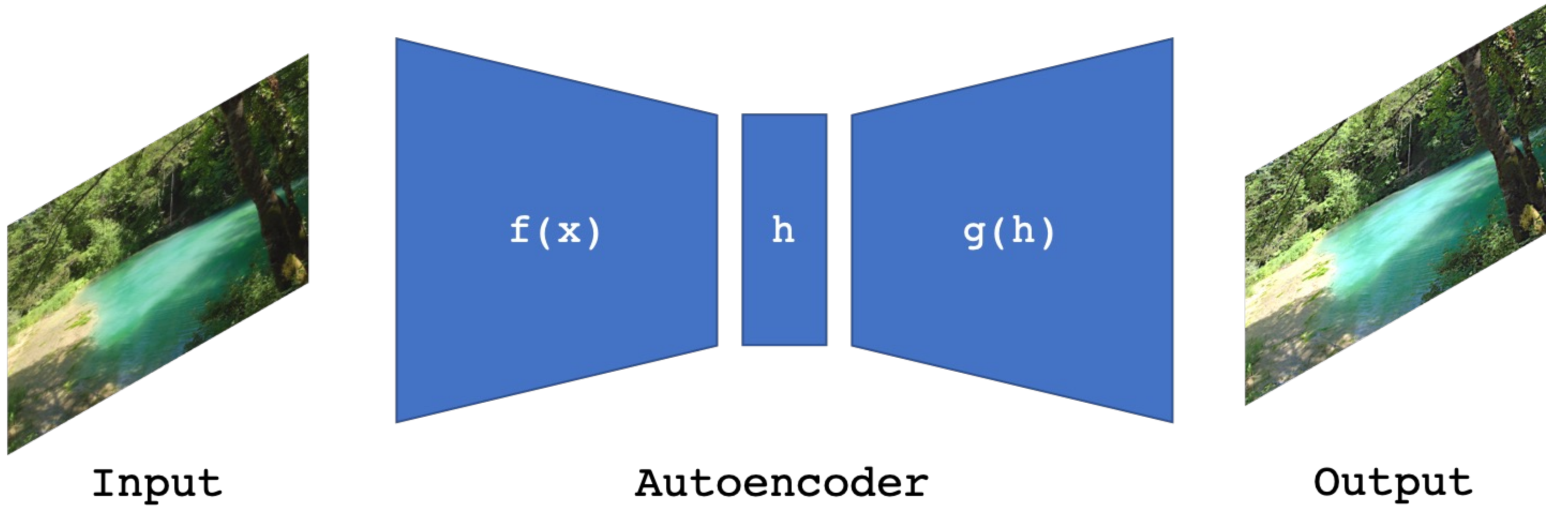


Autoencoder

- An autoencoder is a neural network that is trained to reconstruct its input. It is composed of the encoder, latent code and the decoder. The encoder $f(\cdot)$ transforms the input x to the latent code h, given by $\mathrm{h} = f(x)$. The decoder $\mathrm{g}(\cdot)$ reconstructs the input x from the latent code h, given by $\hat{x} = g(h)$.
- In general, the autoencoder tries to copy the input. Since the latent code has usually smaller dimensions than the input, it can capture useful information in the latent code to reconstruct the input data.

Autoencoder Illustration

- An autoencoder has three major components.
- It can be trained as following:
  - Input: x
  - Output: $\hat{x}$
  - Goal:$x \approx \hat{x}$
  - Loss: $\mathcal{L} = \|x - \hat{x}\|^2$ where $\hat{x} = g(f(x))$.



Autoencoder Motivation

- The idea of building a neural network that learns a compressed representation of the input signal is relatively old.
  - Learning to copy the input[*], known as <u>recirculation</u>, was a step towards autoencoders.
  - The motivation for the development of the autoencoder was to <u>reduce</u> the data dimensions[**].
- Autoencoder learns in an <u>unsupervised</u> manner.
- Unlike PCA, autoencoder is a <u>non-linear</u> approach towards dimensionality reduction.

*Geoffrey E Hinton and James L McClelland. Learning representations by recirculation. In Neural information processing systems, pages 358–366, 1988.
**Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313(5786):504–507, 2006.

Under-Complete Autoencoder

- When the latent code has <u>smaller</u> dimensions than the input data of an autoencoder, then the autoencoder is called under-complete.
  - There is the possibility to build the opposite constellation, i.e. an over-complete autoencoder.
- Learning a linear decoder is similar to spanning the subspace, similar to PCA. However, a non-linear decoder and encoder with the right capacity can be much more powerful then PCA.
- The problem with the autoencoder is <u>overfitting</u>. Increasing the capacity can result in memorizing the data and not actually learning a useful latent code.
- To avoid over-fitting and constrain the latent code to learn useful information, different autoencoder models, training protocols and in general <u>regularization</u> approaches have been proposed.

Regularized Autoencoders

- The idea of a regularized encoder is to impose constraints during learning for the extraction of useful information from the input to the latent code.
- The constraints can be defined as:
  - Weight decay regularization.
  - Noise on the input to become more robust $\rightarrow$ Denoising autoencoder.
  - Small gradients during parameter update $\rightarrow$ Contractive autoencoder.
  - Sparsity of the latent code $\rightarrow$ Sparse autoencoder.
- The autoencoder can be formed with a MLP or Convolutional neural network.

Regularized Autoencoders: Weight Decay

- This is the standard L2 regularization which we already discssed in the past. It is given by:
  - $\mathcal{L}_{RAE} = \mathcal{L} + \beta||\boldsymbol{w}||^2$.
  - β controls the influence of the regularizer. This is the easiest way to regularize an autoencoder.
  - $\mathcal{L}$ is the standard mean-squared-error.

Denoising Autoencoders

- An autoencoder learns eventually the identity transformation, which can easily lead to overfitting. Another approach to avoid over-fitting are denoising autoencoders.
- Motivation: Humans can recognize an image even if it’s corrupted by noise. The same functionality should be possible for an autoencoder as well.
- Input: It is partially corrupted by noise. The noise comes from some prior distribution. This is a stochastic mapping of the clean signal x to a noisy signal $\hat{x}$, represented by:
  - $\hat{x} \sim q_D(\hat{x}|x)$.
  - The stochastic mapping $q_D(\cdot)$ can combine multiple types of prior distributions. In the simple case, it can set a number of elements to zero.
- Output: Given the noisy signal $\hat{x}$ as input, the output will be the clean signal x.

Denoising Autoencoders (Cont.)

- The loss function for n training samples is defined as:
  - $\mathcal{L}_{DAE} = \frac{1}{n}\sum_{i=1}^{n}(x^i - g(f(\hat{x}^i)^2.$
  - By minimizing the loss, we force the encoder to extract features that contain useful information to reconstruct the clean signal.
- The original model* was a shallow deep neural network. It used just a single hidden layer.
- The denoising autoencoder can be interpreted as manifold learning where the low-dimensional manifold is the latent code.
- *How can we evaluate for the quality of the latent code?*

*Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM, 2008.

Stacked Denoising Autoencoders

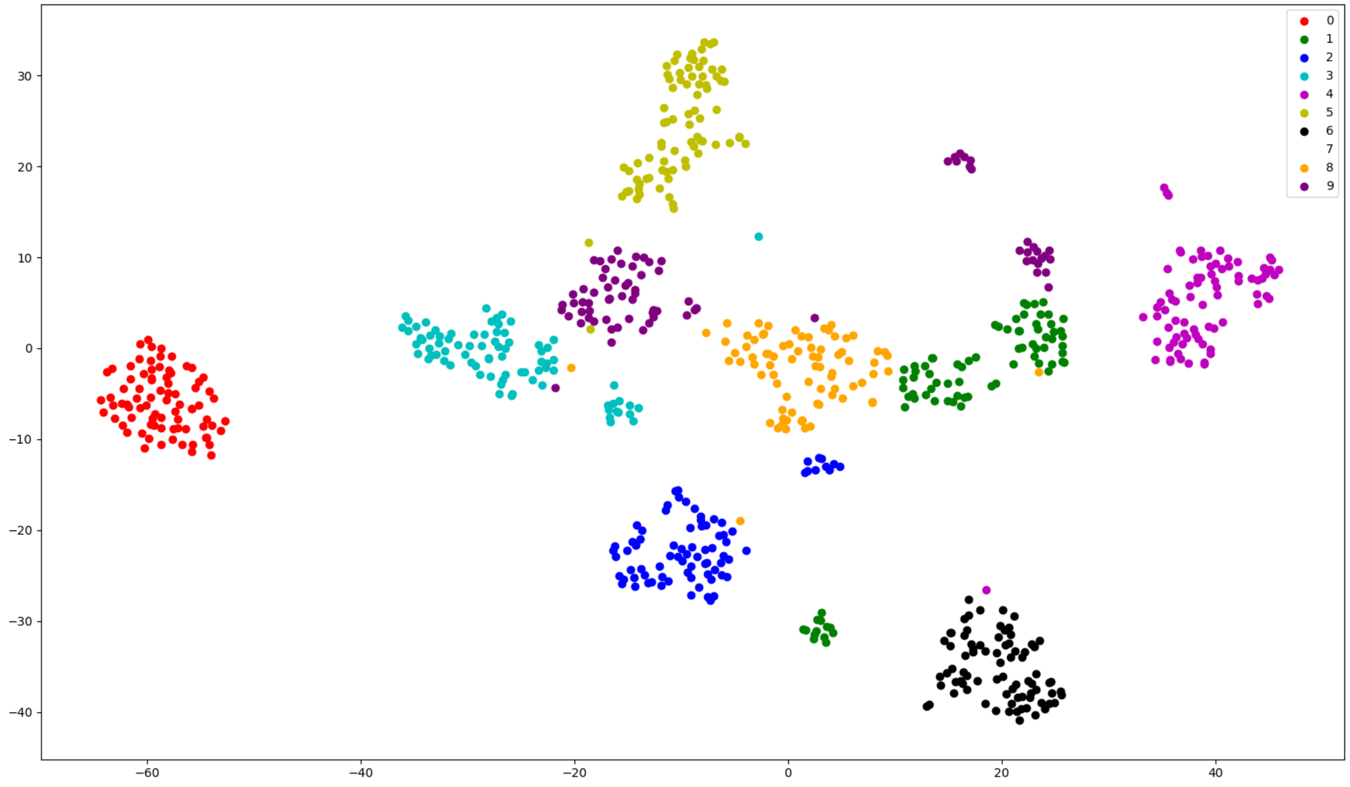
- Stacked denoising autoencoders extend the original model to multiple hidden layers. This model is closer to deep learning.
- Learning is performed in an incremental way:
  - Train the autoencoder with the single hidden layer. Corrupt the input with noise.
  - Add a second hidden layer.
  - Train the second hidden layer (only) by corrupting the first hidden layer's output (only). This means that we pass a clean signal from the input and corrupt it after passing it through the first hidden layer.
  - Add a third hidden layer.
  - Train the third hidden layer (only) by corrupting the second hidden layer's output (only).
  - …

Data Visualisation

- Our motivation of understanding the autoencoders was initially to reduce the data dimensions on a non-linear manner. Some advantages of dimensionality reduction are:
  - Deployment to devices with limited memory. We can use the latent code as input to neural networks instead of the original data.
  - Deployment to devices with limited computational power. Again, using the latent code allows to build smaller network architectures and thus perform faster inference.
  - Another advantage is the ability to easier visualize the data. The latent code is a low-dimensional representation, which we could directly visualize (if possible) or further reduce it's dimensions with non-learning approaches.

Data Visualisation (Cont.)

- Visualisation of the MNIST dataset projected into 2 dimensions.



t-distributed Stochastic Neighbor Embedding

- The t-distributed Stochastic Neighbor Embedding (t-SNE) is another non-linear dimensionality reduction approach* that is often used for the visualization of high-dimensional data.
  - It is common for the 2D or 3D visualization of high-dimensional data, e.g., bioinformatics data.
  - It is a stochastic approach and based on the idea of computing the similarity between neighbor samples.
- t-SNE aims to preserve local structure in terms of pairwise distance between the samples.
  - PCA aim to maximize the variance and thus the pairwise distance.

*Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.11 (2008).

t-distributed Stochastic Neighbor Embedding (Cont.)

- We have again the dataset $\{x_n\}_{n=1}^N$, $x_n \in \mathbb{R}^D$.
- Our goal is to find the low-dimensional representation $\{y_n\}_{n=1}^N$, $x_y \in \mathbb{R}^M$ with M usually to be 2 or 3 for allowing visualization.
- Our criterion is that nearby samples should stay close and distant samples should stay far away from each other.
- For each sample x_j we define the probability to belong to the class with the sample x_i using a Gaussian distribution as:
  - $p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i{}^2}\right)}{\sum_{k\neq i}\exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i{}^2}\right)}$.
  - σ_i is the variance of the Gaussian that is centered on the sample x_i. It is given as an input by specifying the expected number of neighbors. It is called perplexity.
- Also, we have:
  - $p_{ii} = 0$.
  - $p_{j|i} = p_{i|j}$ are symmetrized.
  - $\sum p_{ij} = 1$.
- We form the distribution $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$.

t-SNE Optimization

- For the low-dimensional representation y_j and y_i, we can also compute a similar conditional probability distribution as:
  - $q_{ij} = \frac{\left(1+ \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k}(1+ \|y_k - y_l\|^2)}$.
  - $q_{ii} = 0$.
  - We model the neighborhood sample relation with a t-distribution.
- We have the two distributions for p_{ij} (high-dimensional features) and q_{ij} (low-dimensional map), but y_i is unknown.
- To estimate each y_i, we would like to bring the two distributions as close as possible. For that reason, we can minimize the Kullback-Leibler divergence that is given by:
  - $KL(P|Q) = \sum_{i=1}^{N} \sum_{j=1, i \neq j}^{N} p_{ij} \log(\frac{p_{ij}}{q_{ij}})$.
  - In this way, we formulated the low-dimensional mapping problem as an optimization.
- To solve for each y_i we can rely on the gradient:
  - $\frac{\partial KL(P|Q)}{\partial y_i} = 4 \sum_{j=1}^{N} (p_{ij} - q_{ij}) \left(y_i - y_j\right) \left(1 + |y_i - y_j|^2\right)^{-1}$.

t-SNE Optimization (Cont.)

- We can now rely on gradient descent with momentum to solve for each y_i.
- The complete algorithm can be summarised as:
  - Set perplexity, set number of iterations, velocity, learning rate.
  - Compute pairwise affinities $p_{j|i}$ and set p_{ij}.
  - Sample initial solution for each y_i using a Gaussian distribution.
  - Iterate:
    - Compute low-dimensional affinities q_{ij}.
    - Compute gradient $\frac{\partial KL(P|Q)}{\partial y_i}$.
    - Compute velocity $u = \alpha u + \eta \frac{\partial KL(P|Q)}{\partial y_i}$.
    - Set $y_i = y_i + u$.

Study Material

- *Chapter 10, Deisenroth, Marc Peter, A. Aldo Faisal, and Cheng Soon Ong. Mathematics for machine learning. Cambridge University Press, 2020.*
- *Deep Learning(Ian Goodfellow and Yoshua Bengio and Aaron Courville), Chapter 14, Autoencoders.*

Next Lecture

Support Vector Machines