Machine Learning: Unsupervised Learning

Forest Agostinelli
University of South Carolina

Topics Covered in This Class

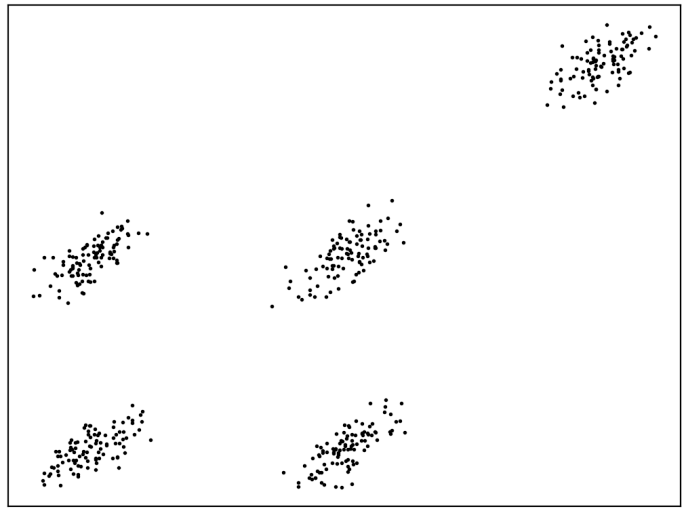
- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

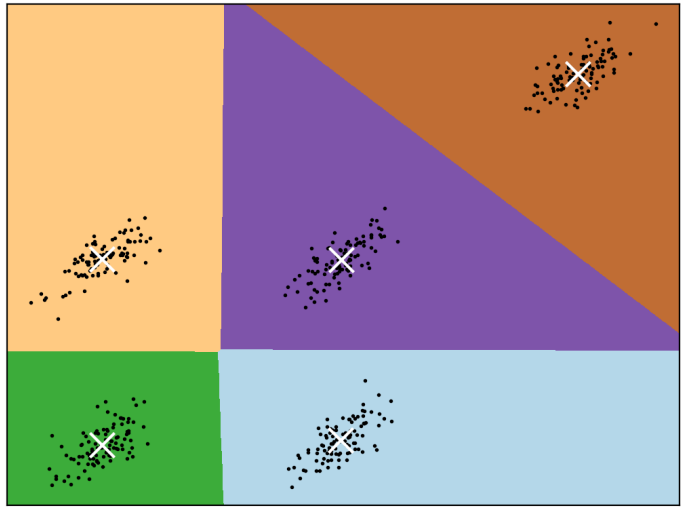
Outline

- Clustering
- Dimensionality reduction
  - PCA
  - Autoencoders
- Generative models

Clustering

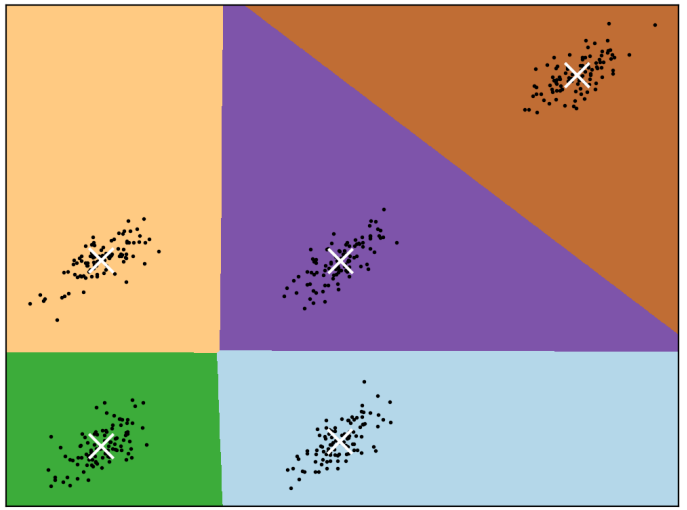
- Grouping objects into clusters where objects in a cluster are more similar than compared to those in other clusters
- Natural sciences
  - High-energy physics
  - Biology
  - Chemistry
- Medicine
  - Patients
  - Diseases
- Reinforcement learning
  - Cluster similar states for hierarchy
  - Cluster similar actions to create meta-actions





K-means Clustering

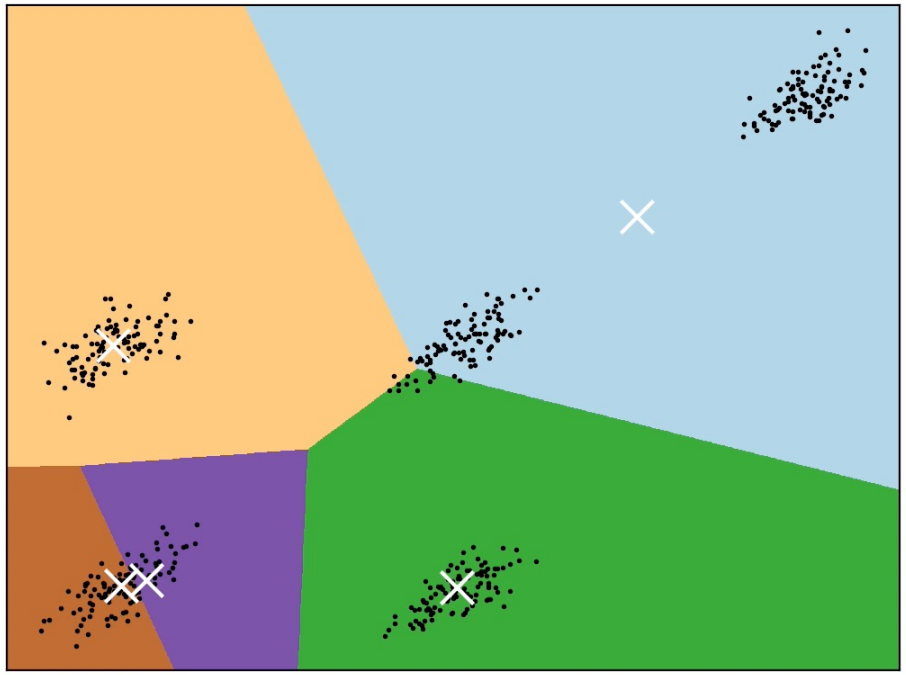
- Suppose we have n data points
- We would like to partition them into K clusters
- Each cluster C_k has a mean (centroid) $\boldsymbol{\mu}_k$
- Each data point can only be associated with one cluster
- Our objective is to minimize the sum of squares within each cluster
  - $\text{argmin}_{\boldsymbol{C}} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in C_k} \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2$



K-means Clustering: Algorithm

- Randomly initialize K centroids
- While True
  - Assign each data point to the nearest centroid (Euclidean distance)
  - Update the K centroids by averaging all points assigned to them
  - If the location of the centroids do not change
    - Break

K-means Clustering: Example

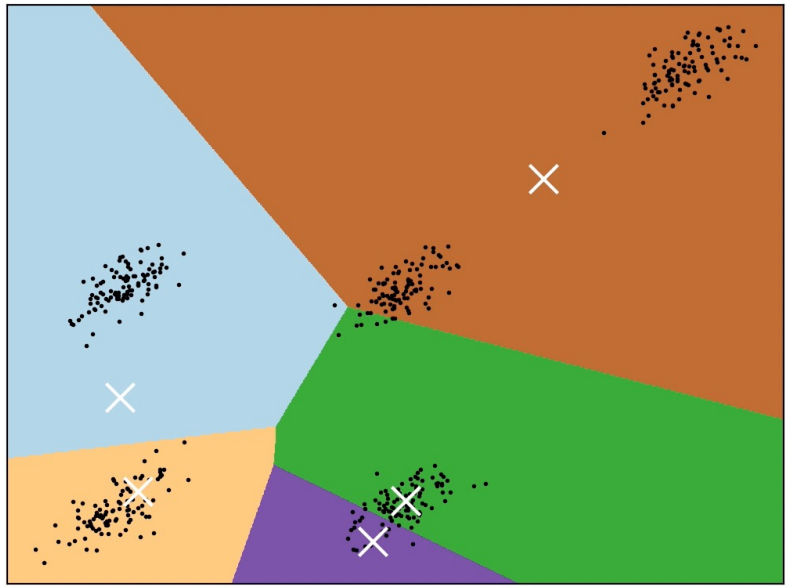


- Is this clustering good?
- What can we do to improve it?

K-means: Convergence

- K-means is not guaranteed to converge to the optimal clusters
  - $\text{argmin}_{\boldsymbol{C}} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in C_k} \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2$
- Initialization
  - We can get better results with better initialization
  - Take into account the range of the data
  - We can partition the data into sections
  - Many other initialization methods
- Multiple restarts
  - Many K-means initialization methods have randomization
  - Therefore, we can run K-means multiple times and select the best clusters according to our optimization criteria

K-means: Random Restarts



- After multiple tries, we may end up with an acceptable solution

Selecting the Best K

- We may not know K beforehand
- We can measure the best sum of squares we obtain for each K
  - $\text{argmin}_{\boldsymbol{C}} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in C_k} \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2$
- Larger K allows for the possibility of finding a lower sum of squares
- However, there is a point in which the sum of squares does not decrease as fast
- We can use this heuristic to find a good K

Curse of Dimensionality

- The phrase was coined by Richard Bellman in reference to solving problems with dynamic programming
- However, this is relevant to many other cases
- In high-dimensions, data has many possibly surprising properties
- In particular, data points tend to be sparse when the dimensionality is increased
  - Euclidean distance becomes less meaningful
  - This makes partitioning data into meaningful clusters difficult or impossible

Curse of Dimensionality: Examples

- Suppose we have a, relatively small, 28 x 28 images
  - There are $28 \times 28 = 784$ −dimensional data points
  - Running K-means on this data will most likely result in meaningless clusters

Outline

- Clustering
- Dimensionality reduction
  - PCA
  - Autoencoders
- Generative models

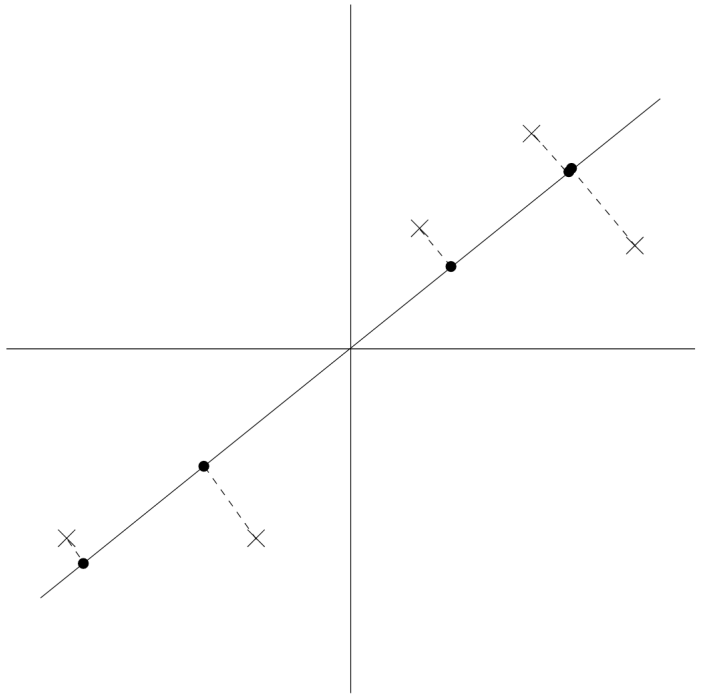
Principal Component Analysis (PCA)

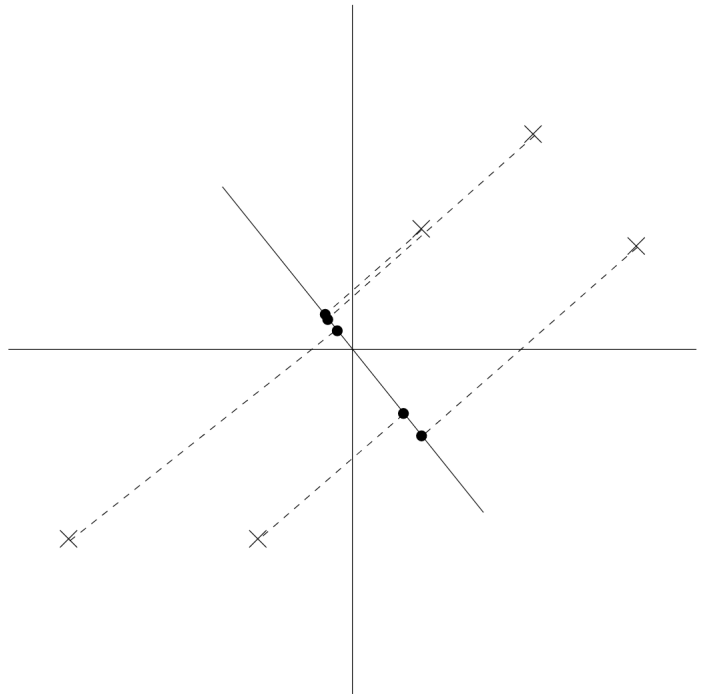
- Suppose we have n data points, each with dimensionality d
  - Each data point $x^{(i)} \in \mathcal{R}^d$
- We would like to find principal components with which to linearly project this data into a new orthogonal coordinate system of lower dimensionality that preserves as much variance as possible

https://cs229.stanford.edu/notes2022fall/main_notes.pdf

Linear Projection and Variance

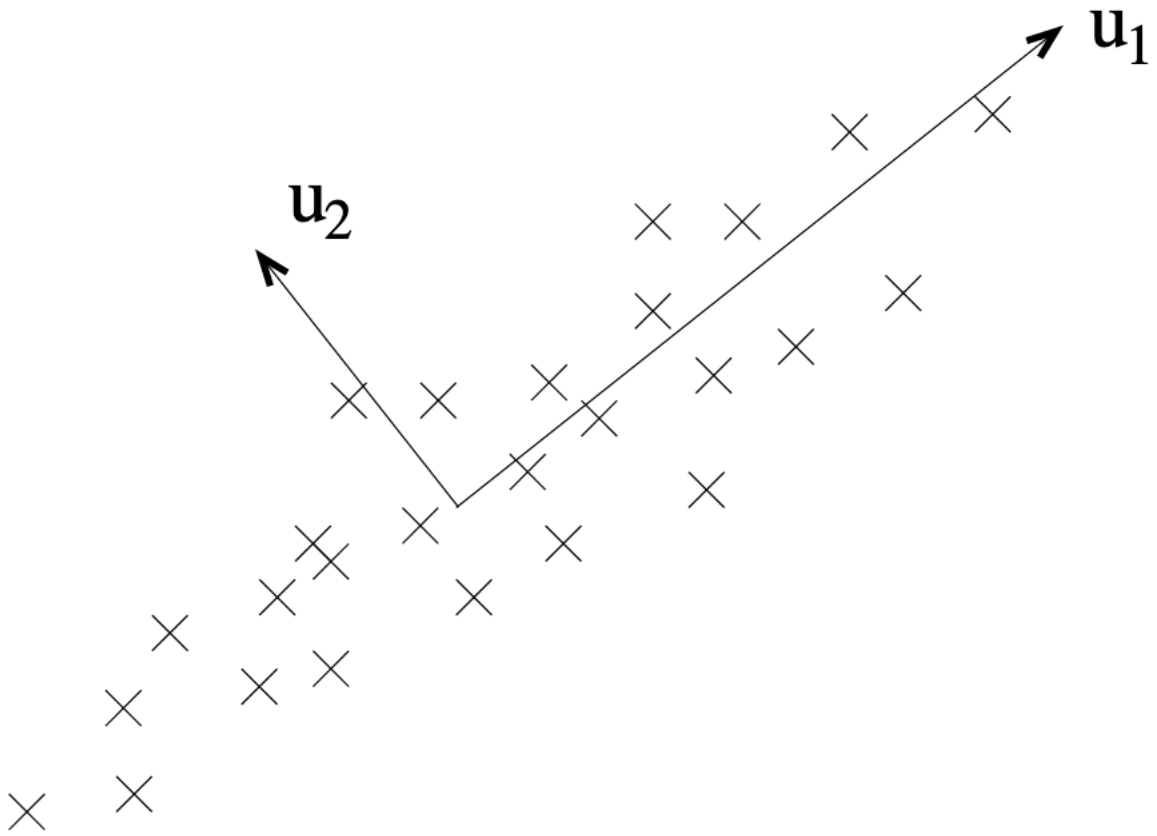
- Given the following data points (x), projecting them on to the first line results in higher variance than the second line
- PCA seeks to find the line with maximum variance





PCA Visualization

- u1 and u2 are the two principal components
- They are orthogonal to one another



Linear Projection and Variance

- Assuming the principal components are unit vectors, the length of a projection of $x^{(i)}$ on to a principal component u is $x^{(i)^T} u$
- The first principal component, $u_1 \in \mathcal{R}^d$, should maximize the variance of projected data
- Empirical variance: $\frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)} - m\right)^2$, where m is the mean.
- We normalize our data beforehand so that it has a mean of zero and a standard deviation of 1
- Therefore u_1 should maximize $\frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)^T} u_1\right)^2$
  - subject to $||u_1||_2 = 1$

Linear Projection and Variance

- $\frac{1}{n}\sum_{i=1}^{n}\left(x^{(i)^T}u_1\right)^2 = \frac{1}{n}\sum_{i=1}^{n} u_1^T x^{(i)} x^{(i)^T} u_1$
- $= u_1^T(\frac{1}{n}\sum_{i=1}^{n} x^{(i)} x^{(i)^T})u_1$
- We see that $\boldsymbol{\Sigma} = (\frac{1}{n}\sum_{i=1}^{n} x^{(i)} x^{(i)^T})$ is the empirical covariance matrix given that our data has mean zero
- How can we choose a u_1 that maximizes the expression $u_1^T \boldsymbol{\Sigma} u_1$ subject to $||u_1||_2 = 1$?

Linear Projection and Variance

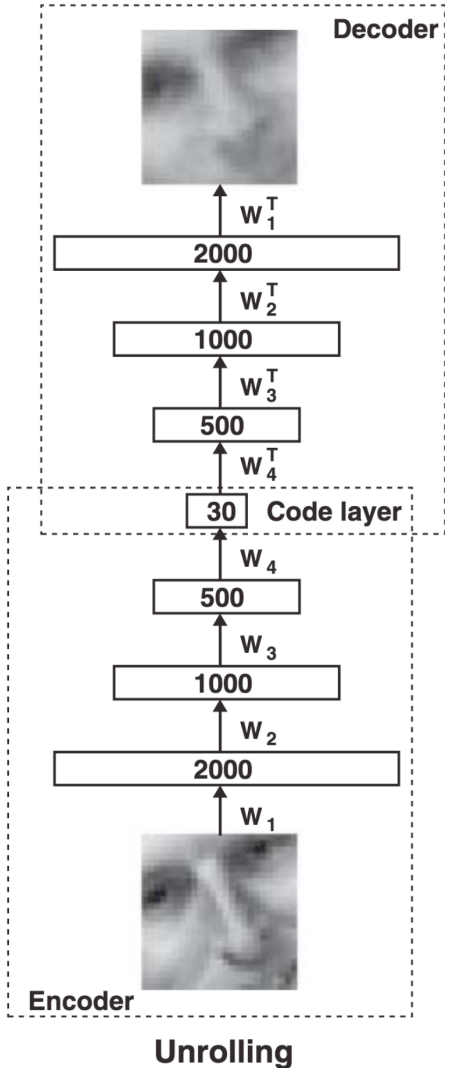
- Eigenvectors of a linear transformation are non-zero vectors that do not change direction when the linear transformation is applied and change, at most, by some scalar factor known as its corresponding eigenvalue
- Therefore, if u_1 is an eigenvector of $\boldsymbol{\Sigma}$ with corresponding eigenvalue
  - $\lambda \boldsymbol{\Sigma} u_1 = \lambda u_1$
- Therefore, because the angle between a vector with itself is 0
  - $u_1^T \lambda u_1 = \lambda ||u_1||_2 ||u_1||_2$
- Therefore, maximizing this expression amounts to setting u_1 to the eigenvector of the covariance matrix with the largest corresponding eigenvalue
- In general, to obtain k principal components where k < d, we can find the top k eigenvectors of the covariance matrix

Outline

- Clustering
- Dimensionality reduction
  - PCA
  - Autoencoders
- Generative models

Autoencoders

- Neural networks that are trained without labels
- The input is passed through an encoder
  - The dimensionality of the output of the encoder is usually much less than the dimensionality of the input
  - Called code layer or bottleneck layer
- The output of the encoder is then passed to the decoder which is trained to mimic the input
- This is known as minimizing the reconstruction error



Unrolling

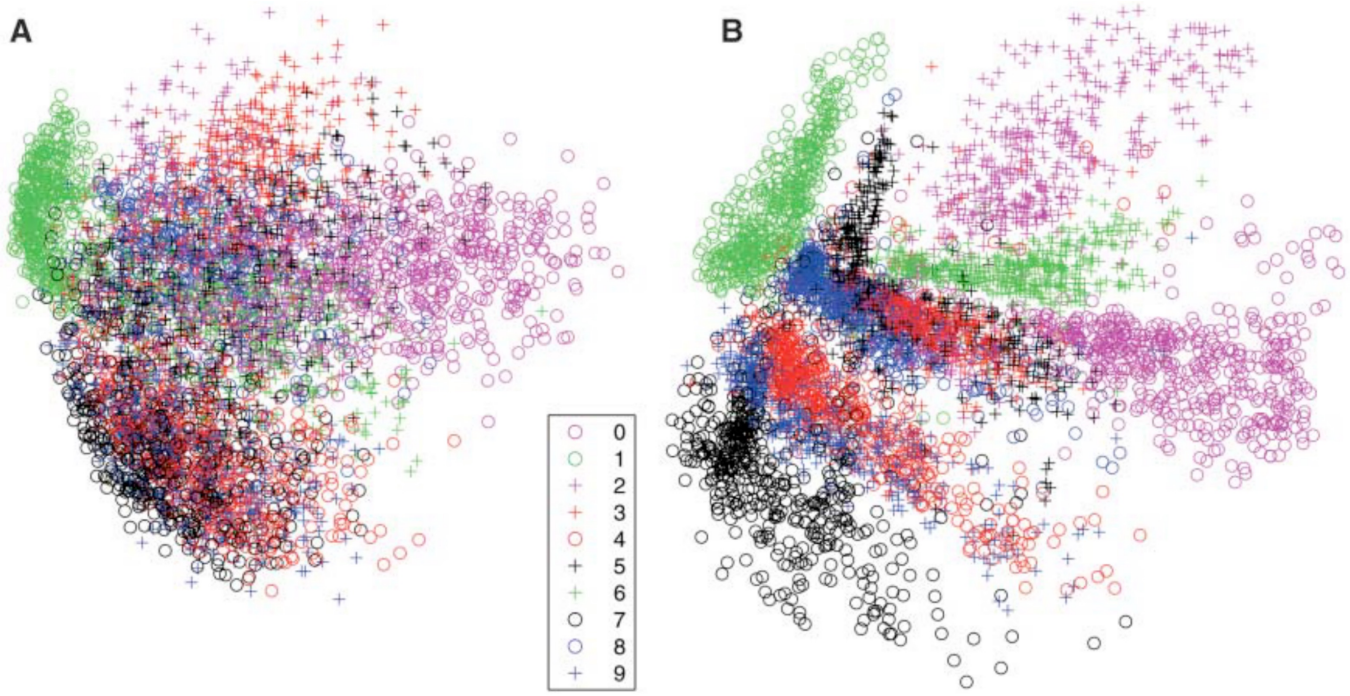
Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507.

Autoencoders: Reconstruction Results

- The reconstructions of autoencoders are generally not exactly the same as the input
- However, they tend to capture the salient features

Autoencoders: Dimensionality Reduction on Digits

- A - PCA
- B - Autoencoder
- By visual inspection we see that the data is grouped into clusters based on the type of digit



Outline

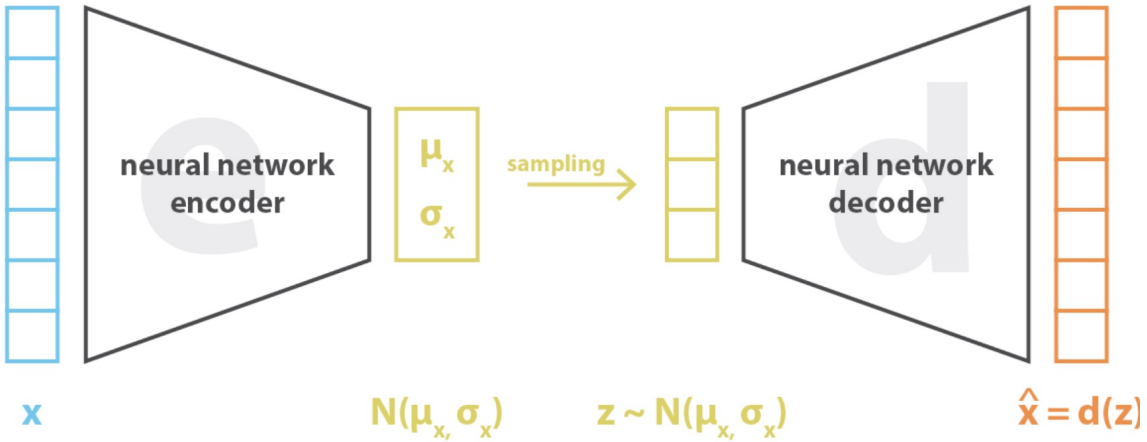
- Clustering
- Dimensionality reduction
  - PCA
  - Autoencoders
- Generative models

Generative Models

- By observing real-world data, we can learn to generate (imagine) new data that we have never seen before
- Simple case: if we assume the data has a Gaussian distribution, we can fit a Gaussian distribution to the data
  - There are known ways to sample from a Gaussian distribution

Variational Autoencoders

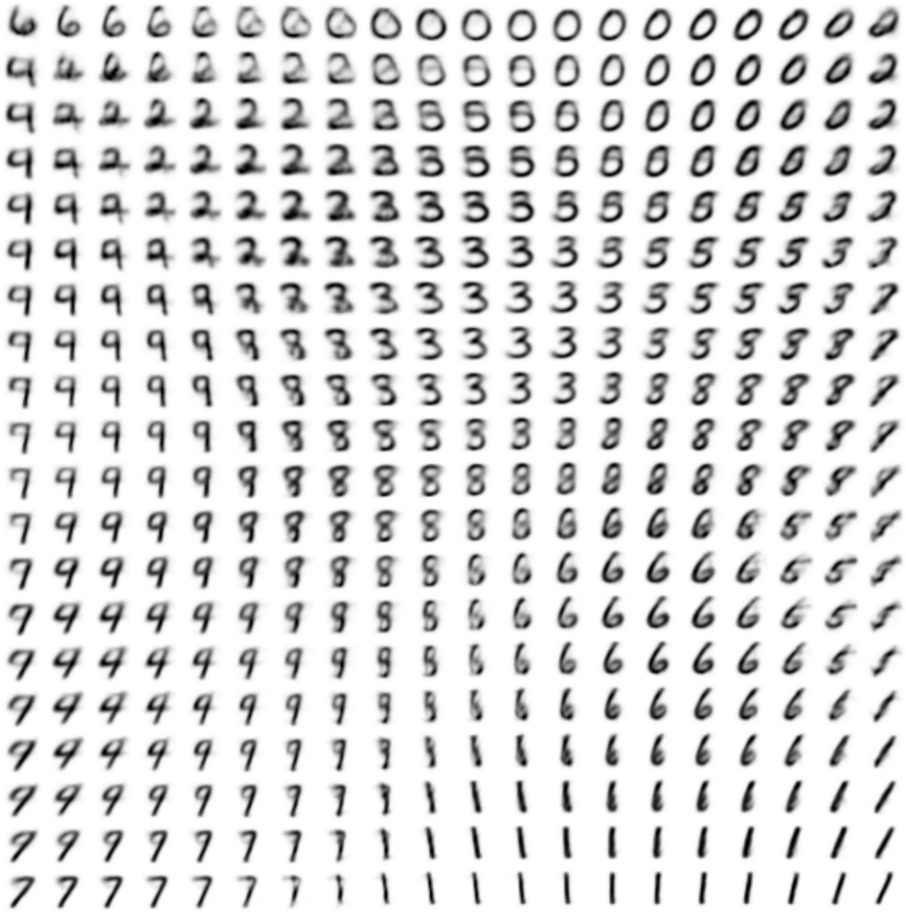
- The bottleneck layer of an autoencoder can be designed to follow a Gaussian distribution
- After training, we only have to sample from this Gaussian distribution and feed this sample to the decoder to generate data



https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Variational Autoencoders

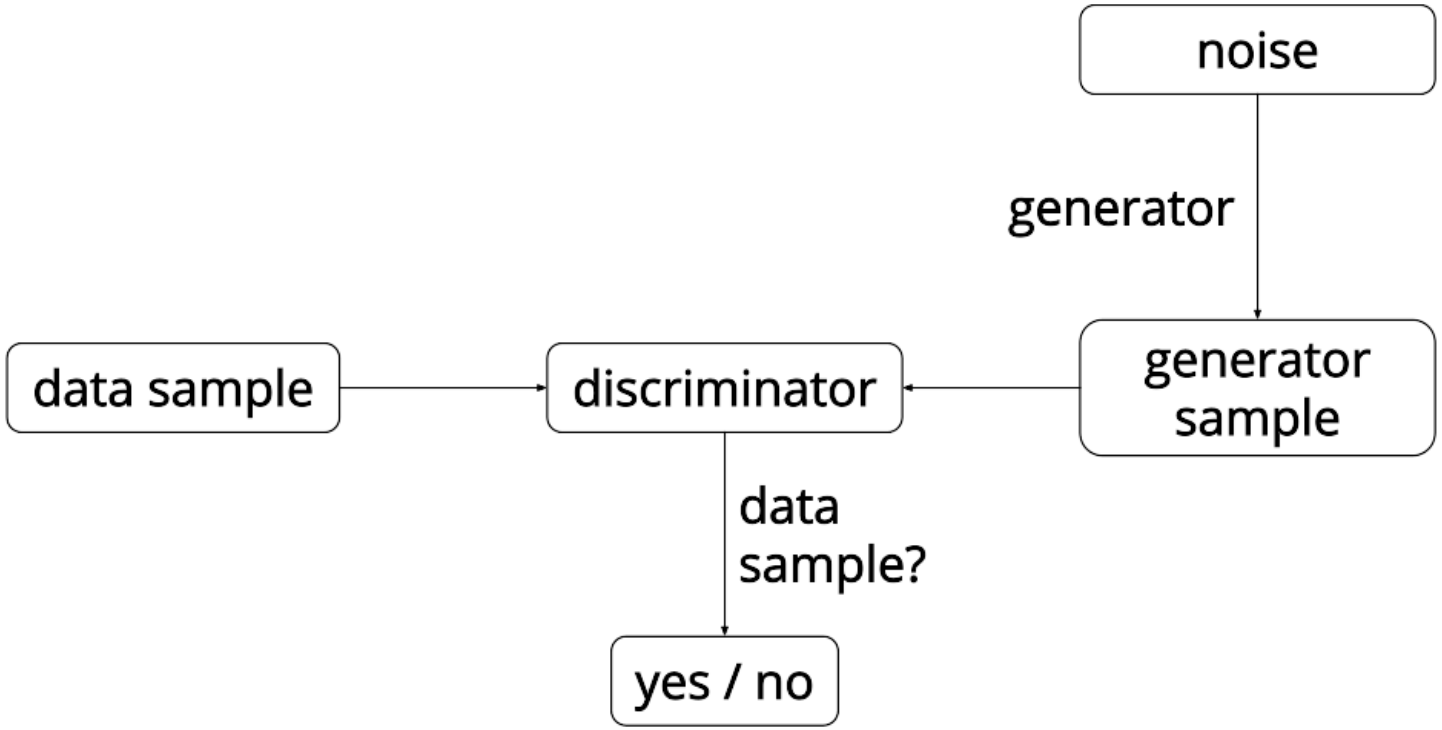
- We can modify a single datapoint by moving its latent code in a certain direction



Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

Generative Adversarial Networks

- We can generate data by trying to fool a discriminator



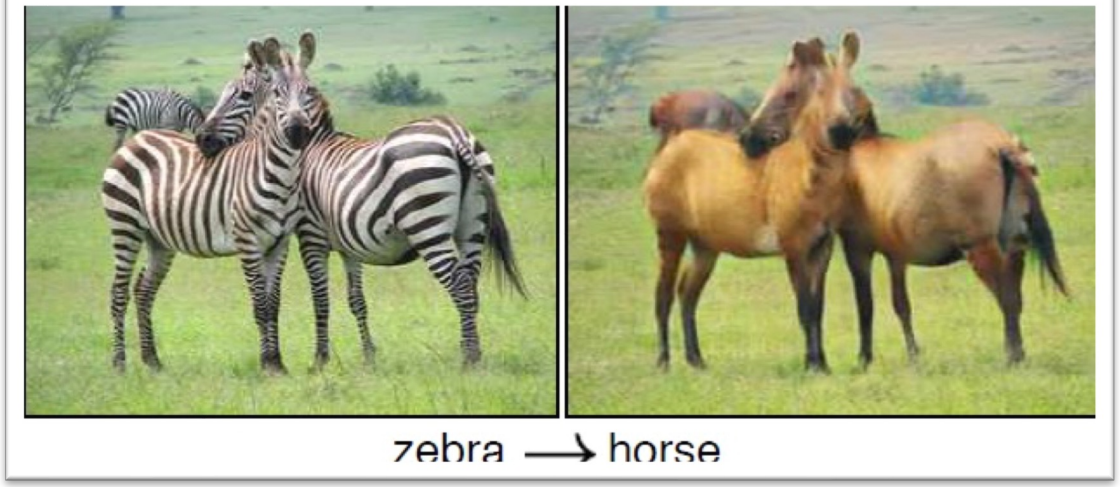
Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. In *NIPS* (pp. 2672-2680).
http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

Generative Adversarial Networks

- Unpaired domain transfer

Monet → photo

photo → Monet



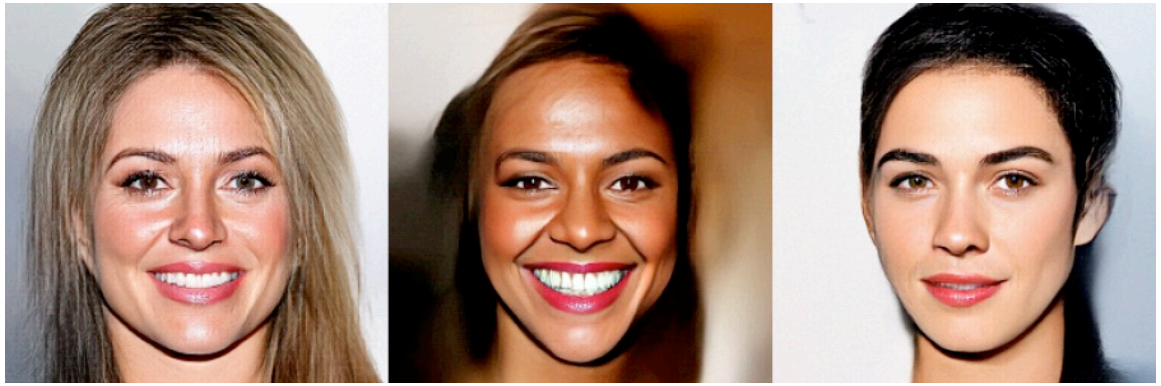
zebra → horse

horse → zebra

J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle consistent adversarial networks. arXiv preprint arXiv:1703.10593, 2017.

Normalizing Flows

- Using **invertible** neural networks one can directly do maximum likelihood during training
- Data can then be generative by sampling from a Gaussian and inverting passing it through in reverse



(a) Smiling

(c) Blond Hair

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*.