Unsupervised Learning

Hwann-Tzong Chen

National Tsing Hua University

13th March 2017

Unsupervised Learning: What, Why, and When?

1. Only given inputs, $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{N}$; not told what the desired output is for each input.
2. The goal is to find interesting patterns or structures in the data.
3. Knowledge discovery: density estimation, clustering, learning representations, dimensionality reduction, finding latent factors.
4. Humans are good at unsupervised learning. (E.g., Iron Chicken?)
5. "The next frontier in AI: unsupervised learning" by Yann LeCun.

Example: Dimensionality Reduction for Data Visualization

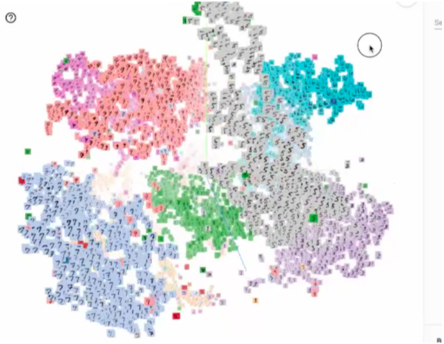


Figure 1: t-SNE visualization in TensorFlow.

Today's Topics

1. Model based clustering: mixture of Bernoullis, Gaussian mixture model (GMM)
2. Latent linear models: principal component analysis (PCA)
3. Sparse linear models: sparse coding
4. Nonlinear dimensionality reduction: locally linear embedding (LLE), t-distributed stochastic neighbor embedding (t-SNE)
5. Autoencoders: denoising autoencoder (DAE), variational autoencoder (VAE)

Mixture of Gaussians/Gaussian Mixture Model (GMM)

Each distribution in the mixture is a multivariate Gaussian with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) .$$

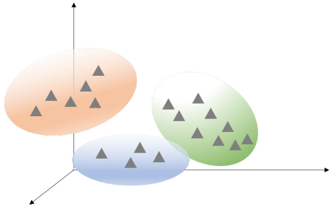


Figure 2: GMM.

$\boldsymbol{\theta} : \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$

GMM for Clustering

The responsibility and soft clustering

1. Fit the mixture model, and compute the posterior probability that a data point $\mathbf{x}_i$ belongs to cluster k.
2. The *responsibility* of cluster k for data point $\mathbf{x}_i$

$$r_{ik} = p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta}) = \frac{p(z_i = k|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})}{\sum_{k'=1}^{K} p(z_i = k'|\boldsymbol{\theta})p(\mathbf{x}_i|z_i = k', \boldsymbol{\theta})}.$$

$p(z_i = k|\boldsymbol{\theta})$: the importance of component k in the mixture
$p(\mathbf{x}_i|z_i = k, \boldsymbol{\theta})$: the likelihood of observing $\mathbf{x}_i$ in component k

EM Algorithm for GMMs

Expectation maximization (EM), an iterative algorithm, with closed-form updates at each step.

E step:

$$r_{ik} = \frac{\pi_k p(\mathbf{x}_i | \boldsymbol{\theta}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i | \boldsymbol{\theta}_{k'}^{(t-1)})} .$$

M step:

$$\pi_k = \frac{1}{N} \sum_i r_{ik} = \frac{r_k}{N} ,$$

$$\boldsymbol{\mu}_k = \frac{\sum_i r_{ik} \mathbf{x}_i}{\sum_i r_{ik}} = \frac{\sum_i r_{ik} \mathbf{x}_i}{r_k} ,$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{r_k} .$$

Example of Using GMM: Video Object Cosegmentation

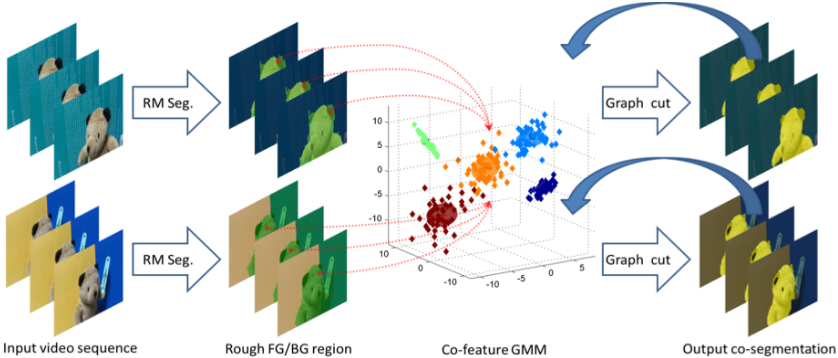


Figure 3: Video Object Cosegmentation.

Mixture of Bernoullis for Binary Data

Consider binary variables $\mathbf{x}_i \in \{0, 1\}$.

Multivariate Bernoulli

Like a binary image of D pixels or a bag of D coins:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{j=1}^{D} \mu_j^{x_j}(1-\mu_j)^{1-x_j} .$$

Mean and covariance:

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}, \ \mathrm{cov}[\mathbf{x}] = \mathrm{diag}\{\mu_j(1-\mu_j)\} .$$

Mixture of K Bernoullis (K bags of D coins)

$$p(\mathbf{x}|\{\boldsymbol{\mu}_k, \pi_k\}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k) .$$

Mixture of Bernoullis for Binary Data

Mixture of K Bernoullis

Like K bags of D coins

$$p(\mathbf{x}|\{\boldsymbol{\mu}_k, \pi_k\}) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}|\boldsymbol{\mu}_k) .$$

Mean:

$$\mathbb{E}[\mathbf{x}] = \sum_{k}^{K} \pi_k \boldsymbol{\mu}_k ,$$

Covariance (not diagonal anymore):

$$\mathrm{cov}[\mathbf{x}] = \sum_{k}^{K} \pi_k [\mathrm{diag}\{\mu_{kj}(1-\mu_{kj})\} + \boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] - \mathbb{E}[\mathbf{x}]\mathbb{E}[\mathbf{x}]^T .$$

EM for Mixtures of Bernoullis

E step:

$$r_{ik} = \frac{\pi_k p(\mathbf{x}_i | \boldsymbol{\mu}_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(\mathbf{x}_i | \boldsymbol{\mu}_{k'}^{(t-1)})} .$$

M step (kth component, jth dimension):

$$\mu_{kj} = \frac{\sum_i r_{ik} x_{ij}}{\sum_i r_{ik}} .$$

Example: MNIST ‘3’, ‘5’, and ‘8’

Load MNIST data:

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Principal Component Analysis (PCA)

The synthesis view of PCA

Goal: to find an *orthogonal* set of L linear basis vectors $\mathbf{w}_j \in \mathbb{R}^D$, and the corresponding coefficients $\mathbf{z}_i \in \mathbb{R}^L$ for each data point $\mathbf{x}_i$, such that the average reconstruction error is minimized:

$$J(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|^2 ,$$

or equivalently

$$J(\mathbf{W}, \mathbf{Z}) = \|\mathbf{X}^T - \mathbf{W}\mathbf{Z}^T\|_F^2 , \text{ with } \mathbf{X} \in \mathbb{R}^{N \times D}, \mathbf{Z} \in \mathbb{R}^{N \times L}, \mathbf{W}^T\mathbf{W} = \mathbf{I}_L .$$

The Frobenius norm of matrix $\mathbf{A}$

$$\|A\|_F = \sqrt{\sum_i \sum_j a_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^T\mathbf{A})} = \sqrt{\mathrm{tr}(\mathbf{A}\mathbf{A}^T)} = \|\mathbf{A}(:)\|_2 .$$

Principal Component Analysis (PCA)

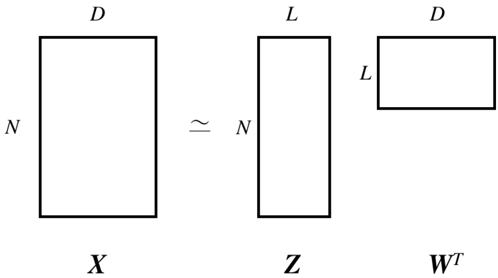


Figure 4: Matrix representation.

Principal Component Analysis (PCA)

The synthesis view of PCA

Solution: obtained by setting $\mathbf{W} = \mathbf{V}_L$, where $\mathbf{V}_L$ consists of the L eigenvectors corresponding to the L largest eigenvalues of the empirical covariance matrix

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T \, .$$

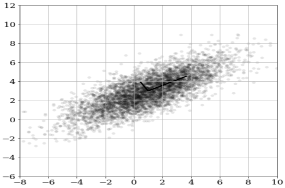


Figure 5: Gaussian PCA.

Example of PCA: Eigenfaces

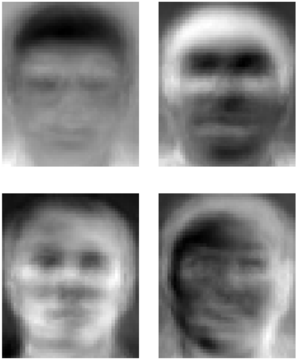


Figure 6: Eigenfaces.

Image credits: eigenface images from Wikipedia by MH & Ylebru, original dataset by AT&T Laboratories Cambridge

Principal Component Analysis (PCA)

The analysis view of PCA

Minimizing the reconstruction error is equivalent to maximizing the variance of the projected data.

The variance of the projected data can be written as

$$\frac{1}{N}\sum_{i=1}^{N} z_{1i}^2 = \frac{1}{N}\sum_{i=1}^{N} \mathbf{w}_1^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{w}_1 = \mathbf{w}_1^T \hat{\mathbf{\Sigma}} \mathbf{w}_1 \,,$$

where

$$\hat{\mathbf{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^T \,.$$

We need to impose the constraint $\|\mathbf{w}_1\| = 1$, and it can be shown that $\mathbf{w}_1^T \hat{\mathbf{\Sigma}} \mathbf{w}_1 = \lambda_1$ is an eigenvalue of $\hat{\mathbf{\Sigma}}$.

Singular Value Decomposition (SVD) and PCA

Decompose an $N \times D$ data matrix $\mathbf{X}$:

$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, where $\mathbf{U}^T\mathbf{U} = \mathbf{I}_N, \mathbf{V}^T\mathbf{V} = \mathbf{I}_D$, and $\mathbf{S}^2$ is a diagonal matrix.

To get $\mathbf{U}$ and $\mathbf{V}$, compute the eigen-decomposition for

$$\mathbf{X}\mathbf{X}^T = \mathbf{U}\mathbf{S}\mathbf{V}^T\mathbf{V}\mathbf{S}^T\mathbf{U}^T = \mathbf{U}(\mathbf{S}\mathbf{S}^T)\mathbf{U}^T ,$$
$$\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{S}^T\mathbf{U}^T\mathbf{U}\mathbf{S}\mathbf{V}^T = \mathbf{V}(\mathbf{S}^T\mathbf{S})\mathbf{V}^T .$$

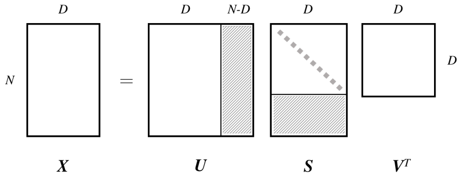


Figure 7: Singular Value Decomposition.

Singular Value Decomposition (SVD) and PCA

Express the empirical covariance matrix in PCA by matrix multiplication

$$\hat{\mathbf{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T = \frac{1}{N}\mathbf{X}^T\mathbf{X}\,.$$

The eigenvectors of $\hat{\mathbf{\Sigma}}$ are equal to the right singular vectors of $\mathbf{X}$.

We can compute PCA using just a few lines of code based on (thin) SVD.

Singular Value Decomposition (SVD) and PCA

The low-rank approximation view

$$\|\mathbf{X} - \mathbf{X}_L\|_F \approx \sigma_{L+1}$$

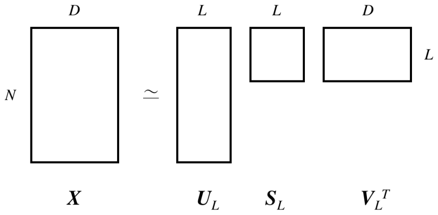


Figure 8: Truncated Singular Value Decomposition.

Sparse Coding

Negative log-likelihood:

$$NLL(\mathbf{W}, \mathbf{Z}) = \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 + \lambda \|\mathbf{z}_i\|_1 .$$

$$\mathcal{C} = \{\mathbf{W} \in \mathbb{R}^{D \times L} \text{ s.t. } \mathbf{w}_j^T \mathbf{w}_j \leq 1\} .$$

$\mathbf{W}$ is called a dictionary.
The columns of $\mathbf{W}$ are not required to be orthogonal.
Usually $L > D$: overcomplete representation.
$\mathbf{z}_i$ is sparse: only a few columns of $\mathbf{W}$ are needed for reconstructing $\mathbf{x}_i$.

Learning a sparse coding dictionary

$$\min_{\mathbf{W} \in \mathcal{C}, \mathbf{Z} \in \mathbb{R}^{L \times N}} \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 + \lambda \|\mathbf{z}_i\|_1 .$$

Why ℓ_1 Regularization

$\|\mathbf{z}\|_p = (|z_1|^p + |z_2|^p + \cdots + |z_L|^p)^{1/p}$
$\|\mathbf{z}\|_\infty = \max\{|z_1|, |z_2|, \ldots, |z_L|\}$

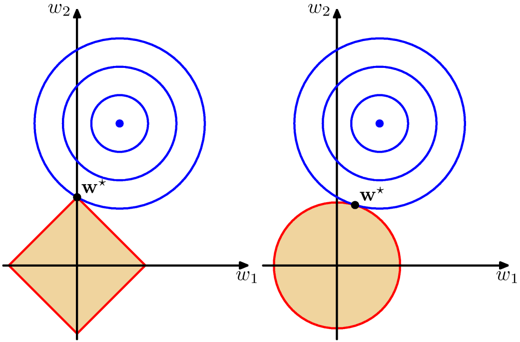


Figure 9: ℓ_1 vs. ℓ_2 regularization. (Lasso vs. ridge regression.) [Image from Bishop, PRML]

Dictionary

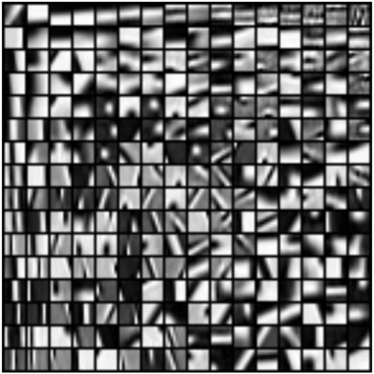


Figure 10: Dictionary. [Elad & Aharon]

Dictionary Learning

Sparse coding: For a fixed dictionary $\mathbf{W}$, the optimization problem over $\mathbf{Z}$ is identical to the lasso problem (least absolute shrinkage and selection operator [Tibshirani]), which can be solved by LARS algorithm (least angle regression)

SPAMS: J. Mairal, F. Bach and J. Ponce. Sparse Modeling for Image and Vision Processing. http://spams-devel.gforge.inria.fr
Optimization with Sparsity-Inducing Penalties
https://hal.archives-ouvertes.fr/hal-00613125v1/document

Dictionary update: With $\mathbf{Z}$ fixed, solve for $\mathbf{W}$ using projected gradient descent.

Other Formulations

ℓ_0 regularization

Learn the dictionary using ℓ_1-penalty. For the final reconstruction step, ℓ_0-penalty is better:

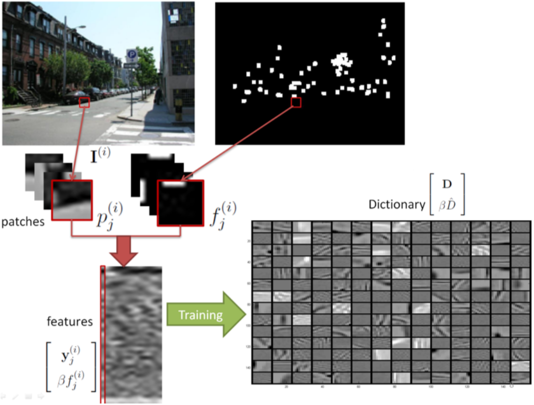
$$\min_{\mathbf{z}_i \in \mathbb{R}^L} \|\mathbf{z}_i\|_0 \quad \text{s.t.} \quad \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 \leq \epsilon \, ,$$

which can be solved by orthogonal matching pursuit (OMP).

Non-negative matrix factorization

$$\min_{\mathbf{W} \in \mathcal{C}, \mathbf{Z} \in \mathbb{R}^{L \times N}} \sum_{i=1}^{N} \frac{1}{2} \|\mathbf{x}_i - \mathbf{W}\mathbf{z}_i\|_2^2 \text{ s.t. } \mathbf{W} \geq 0, \mathbf{z}_i \geq 0 \, .$$

Example: Learning Sparse Dictionaries for Saliency Detection



$$\underset{\mathbf{D},\boldsymbol{\alpha},\hat{D}}{\text{minimize}} \sum_i \sum_{p_j^{(i)} \in \mathbf{I}^{(i)}} \frac{1}{2} \left\| \begin{bmatrix} \mathbf{D} \\ \beta \hat{D} \end{bmatrix} \boldsymbol{\alpha}_j^{(i)} - \begin{bmatrix} \mathbf{y}_j^{(i)} \\ \beta f_j^{(i)} \end{bmatrix} \right\|_2^2 + \lambda \left\| \boldsymbol{\alpha}_j^{(i)} \right\|_1 .$$

Figure 11: Overview of dictionary training for saliency detection.

Locally Linear Embedding (LLE)

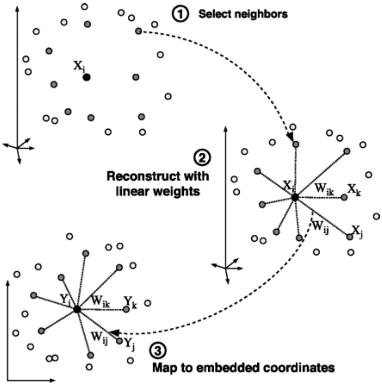


Figure 12: LLE.

LLE

Each data point $\mathbf{x}_i$ is reconstructed only from its neighbors. The rows of the weight matrix sum to one: $\sum_j \mathbf{w}_{ij} = 1$.

$$\mathcal{E}(\mathbf{W}) = \sum_i \|\mathbf{x}_i - \sum_j \mathbf{w}_{ij}\mathbf{x}_j\|^2 .$$

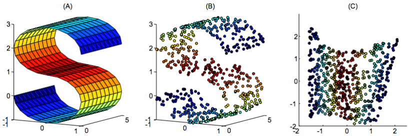


Figure 13: Discovering the 2D manifold in 3D space.

Solve for $\mathbf{Y}$ by minimizing

$$\Phi(\mathbf{Y}) = \sum_i \|\mathbf{y}_i - \sum_j \mathbf{w}_{ij}\mathbf{y}_j\|^2 .$$

LLE

LLE Algorithm

1. Compute the neighbors of each data point $\mathbf{x}_i$;
2. Compute the weights $\mathbf{w}_{ij}$ that best reconstruct each data point $\mathbf{x}_i$ from its neighbors, minimizing the cost $\mathcal{E}$ by constrained linear fits;
3. Compute the vectors $\mathbf{y}_i$ best reconstructed by the weights $\mathbf{w}_{ij}$, minimizing the quadratic form Φ by its bottom nonzero eigenvectors.

Solve $\mathcal{E}(\mathbf{W})$ in LLE

For a data point $\mathbf{x}$ and its K neighbors $\boldsymbol{\eta}_j$, the local reconstruction error

$$\epsilon = |\mathbf{x} - \sum_j w_j \boldsymbol{\eta}_j|^2 = |\sum_j w_j(\mathbf{x} - \boldsymbol{\eta}_j)|^2 = \sum_{j,k} w_j w_k C_{jk} \, ,$$

where $C_{jk} = (\mathbf{x} - \boldsymbol{\eta}_j) \cdot (\mathbf{x} - \boldsymbol{\eta}_k)$ is the local convariance matrix.

The error can be minimized in closed from (with constraint $\sum_j w_j = 1$):

$$w_j = \frac{\sum_k C_{jk}^{-1}}{\sum_{l,m} C_{lm}^{-1}} \, .$$

In practice, solve $\sum_k C_{jk} w_k = 1$ and rescale the weights to make them sum to one.

Solve $\Phi(\mathbf{Y})$ in LLE

Eigenvalue problem

$$\min_{\mathbf{Y}} \ \Phi(\mathbf{Y}) = \sum_i \|\mathbf{y}_i - \sum_j \mathbf{w}_{ij}\mathbf{y}_j\|^2 .$$

$$\Phi(\mathbf{Y}) = \sum_{i,j} \mathbf{m}_{ij}(\mathbf{y}_i\mathbf{y}_j) .$$

Constraints $\sum_i \mathbf{y}_i = 0$ and $\frac{1}{N}\sum_i \mathbf{y}_i\mathbf{y}_i^T = \mathbf{I}$.

The optimal embedding is found by computing the bottom $d+1$ eigenvectors of the matrix $\mathbf{M}$

$$\mathbf{M} = (\mathbf{I}-\mathbf{W})^T(\mathbf{I}-\mathbf{W}) .$$

The bottom eigenvector is a unit vector enforcing the zero mean constraint.

t-Distributed Stochastic Neighbor Embedding (t-SNE)

High-dimensional map p_{ij}

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)},\ p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}.$$

Low-dimensional map q_{ij} (student t-distribution, heavy tailed, infinite mixture of Gaussians with different variances)

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}}.$$

Symmetric SNE:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}},$$

$p_{ij} = p_{ji}$, $q_{ij} = q_{ji}$, $p_{ii} = 0$, and $q_{ii} = 0$.

t-SNE

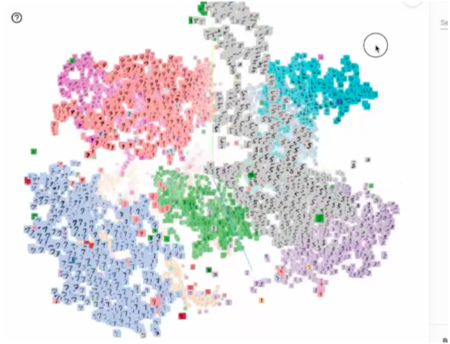


Figure 14: t-SNE visualization in TensorFlow.

Derivation of the t-SNE gradient

Perplexity (a smooth measure of the effective number of neighbors) for σ_i:

$$Perp(P_i) = 2^{H(P_i)}\,,\ H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}\,.$$

Using binary search to find σ_i to make the entropy of the distribution over neighbors equal to $\log Perp(P_i)$.

$$\frac{\delta C}{\delta \mathbf{y}_i} = 4\sum_j (p_{ij} - q_{ij})(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}(\mathbf{y}_i - \mathbf{y}_j)\,.$$

Algorithm

1. Compute $p_{j|i}$ with perplexity, and then compute p_{ij}.
2. Loop:
   Compute q_{ij} and $\frac{\delta C}{\delta \mathbf{Y}}$
   Set $\mathbf{Y}^{(t)} = \mathbf{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathbf{Y}} + \alpha(t)(\mathbf{Y}^{t-1} - \mathbf{Y}^{t-2})$

Denosing Autoencoders (DAE)

An autoencoder is a neural network that is trained to attempt to copy its input to its output.
Autoencoders with linear neurons + squared loss = PCA

The *denoising autoencoder* (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.

DAE

1. From the original input $\mathbf{x}$, generate a corrupted input $\tilde{\mathbf{x}} \sim q(\tilde{\mathbf{x}}|\mathbf{x})$. (Simulating missing data, dropout)
2. Hidden representation $\mathbf{z} = f_\theta(\tilde{\mathbf{x}})$.
3. From $\mathbf{z}$, reconstruct $\mathbf{y} = g_{\theta'}(\mathbf{z})$.
4. Minimize the cross entropy $\mathbb{E}_{\mathcal{B}(\mathbf{z})}[-\log \mathcal{B}(\mathbf{y})]$ or $\|\mathbf{x} - \mathbf{y}\|^2$ as the reconstruction error between $\mathbf{x}$ and $\mathbf{y}$ to train the parameters θ and θ'.

DAE Example

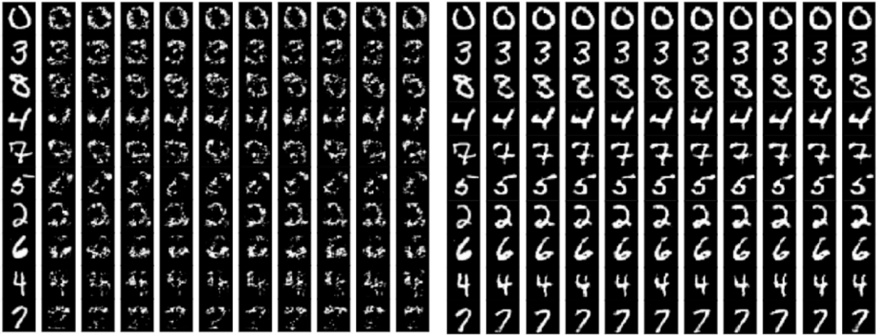


Figure 15: An example of denoising autoencoder.

Variational Autoencoders (VAEs)

Maximize the probability of generating each data point $\mathbf{x}$ in the dataset

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z};\theta)p(\mathbf{z})d\mathbf{z}\,,$$

where

$$p(\mathbf{x}|\mathbf{z};\theta) = \mathcal{N}(\mathbf{x}|f(\mathbf{z};\theta),\sigma^2\mathbf{I}) \text{ and } p(\mathbf{z}) = \mathcal{N}(\mathbf{0},\mathbf{I})\,.$$

How to define the latent variable $\mathbf{z}$?

How to deal with the integral over $\mathbf{z}$?

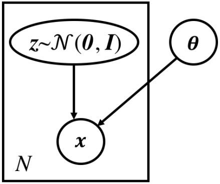


Figure 16: Probability model.

Variational Autoencoders (VAEs)

Introduce a function $q(\mathbf{z}|\mathbf{x})$ for sampling values of $\mathbf{z}$ that are likely to have produced $\mathbf{x}$.

Kullback-Leibler divergence

$$KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{z}|\mathbf{x})]\,.$$

Bayes rule

$$KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q}[\log q(\mathbf{z}|\mathbf{x}) - \log p(\mathbf{x}|\mathbf{z}) - \log p(\mathbf{z})] + \log p(\mathbf{x})\,.$$

$$\log p(\mathbf{x}) - KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_{\mathbf{z}\sim q}[\log p(\mathbf{x}|\mathbf{z})] - KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})]\,.$$

Perform stochastic gradient descent on the right hand side of the above equation.

Solving VAE

Optimize

$$\mathbb{E}_{\mathbf{x}\sim D}[\log p(\mathbf{x}) - KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})]] = \mathbb{E}_{\mathbf{x}\sim D}[\mathbb{E}_{\mathbf{z}\sim q}[\log p(\mathbf{x}|\mathbf{z})] - KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})$$

1. Let $q(\mathbf{z}|\mathbf{x})$ be a multivariate Gaussian $\mathcal{N}(\mu(\mathbf{x};\theta), \Sigma(\mathbf{x};\theta))$ depending on $\mathbf{x}$.
2. $\mu(\mathbf{x};\theta)$ and $\Sigma(\mathbf{x};\theta)$ are implemented via neural networks.
3. $\Sigma(\mathbf{x};\theta)$ is constrained to be a diagonal matrix.
4. $KL[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})]$ is the KL divergence between two Gaussians, which can be computed in closed form.

Reparameterization trick: We can sample from $\mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}))$ by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and then computing $\mathbf{z} = \mu(\mathbf{x}) + \Sigma 1/2(\mathbf{\Sigma})e$.

VAEs

VAEs are generative. q is the encoder and p (f) is the decoder.

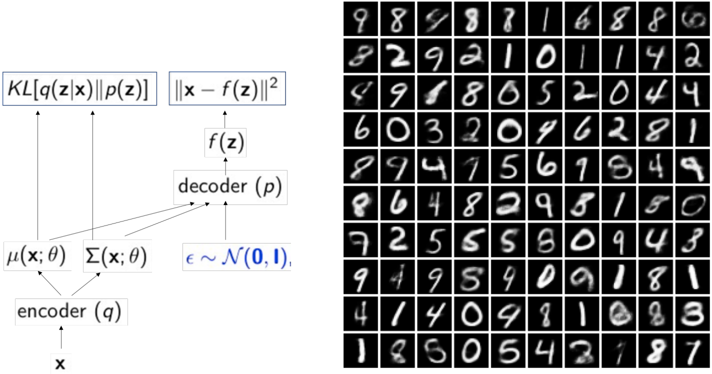


Figure 17: Left: Pipeline of VAE. Right: Samples generated by a trained VAE.

References

1. Kevin P. Murphy, "Machine Learning: A Probabilistic Perspective"
2. Christopher Bishop, "Patter Recognition and Machine Learning"

Thank You