STA 437/2005:
Methods for Multivariate Data

Weeks 9-10: Non-linear Dimension Reduction Techniques

Piotr Zwiernik

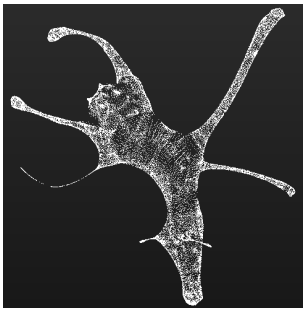
University of Toronto

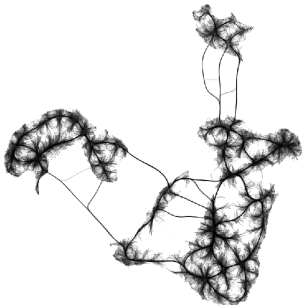
Why Principal Component Analysis may not be enough?

PCA struggles with non-linear relationships.

High-dimensional datasets often lie on low-dimensional manifolds.

Linear projections may destroy these geometric information.





We will now discuss four popular non-linear dimensionality reduction techniques: multi-dimensional scaling, spectral embedding, UMAP, and Autoencoders.

Multi-dimensional Scaling (MDS)

- In its classical version this is essentially PCA.
- MDS allows us to introduce some fundamental concepts.

Problem Setup

Consider n objects and a measure $\delta_{ij} \geq 0$ of their dissimilarity (small if similar); $\delta_{ii} = 0$.

Define $\Delta = (\delta_{ij}) \in \mathbb{R}^{n \times n}$: $\delta_{ii} = 0$ for all i, $\delta_{ij} \geq 0$ for all $i \neq j$.

In classical MDS: there exist $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^m$ such that $\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$.

In general, there need not be a Euclidean distance defining this metric.

Multidimensional Scaling

Find a configuration of points $\mathbf{y}_1, \ldots, \mathbf{y}_n$ in $\mathbb{R}^d$ ($d \ll n$) such that:

$$\|\mathbf{y}_i - \mathbf{y}_j\| \approx \delta_{ij}.$$

The solution for classical MDS is particularly simple.

Classical MDS: $\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$

If $\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$, we have:

$$\delta_{ij}^2 \;=\; (\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j) \;=\; (\mathbf{X}\mathbf{X}^\top)_{i,i} + (\mathbf{X}\mathbf{X}^\top)_{j,j} - 2(\mathbf{X}\mathbf{X}^\top)_{i,j}.$$

The Hadamard product $\Delta \odot \Delta = [\delta_{ij}^2]$ can be written as:

$$\Delta \odot \Delta = \mathrm{diag}(\mathbf{X}\mathbf{X}^\top)\mathbf{1}\mathbf{1}^\top + \mathbf{1}\mathbf{1}^\top \mathrm{diag}(\mathbf{X}\mathbf{X}^\top) - 2\mathbf{X}\mathbf{X}^\top$$

Reintroducing the centering matrix $H = I_n - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$, we obtain

$$B \;:=\; -\frac{1}{2} H(\Delta \odot \Delta)H \;=\; H\mathbf{X}(H\mathbf{X})^\top \;=\; \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top.$$

This matrix contains all inner products $\tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j$ for $1 \le i, j \le n$.

Classical MDS (2)

Let $\mathbf{Y} \in \mathbb{R}^{n \times d}$ be the matrix with projected data $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$.

We want to make sure $B = \tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top \approx \mathbf{Y}\mathbf{Y}^\top =: M$

- In this way $\|\mathbf{y}_i - \mathbf{y}_j\| \approx \|\mathbf{x}_i - \mathbf{x}_j\|$ as desired.
- One way to assure this is to minimize $\sum_{i,j} (\tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j - \mathbf{y}_i^\top \mathbf{y}_j)^2 = \|B - M\|_F^2$.
- The Frobenius norm $\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$.

Note that $\operatorname{rank}(M) \leq d$ but otherwise $M \in \mathbb{R}^{n \times n}$ is arbitrary.

Optimization problem: Minimize $\|B - M\|_F^2$ subject to $\operatorname{rank}(M) \leq d$.

Classical MDS (3)

Optimization problem: Minimize $\|B - M\|_F^2$ subject to $\mathrm{rank}(M) \leq d$.

Let $B = V\Lambda V^\top$ be the spectral decomposition with $\mathrm{diag}(\Lambda)$ non-increasing.

Eckart-Young Theorem

The optimal M satisfies $\widehat{M} = V_d \Lambda_d V_d^\top$, where

- $\Lambda_d = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$ has d largest eigenvalues of B.
- $V_d \in \mathbb{R}^{n \times d}$ contains the first d columns of V.

We then take $\mathbf{Y} = V_d \Lambda_d^{1/2}$, which gives us our low-dimensional embedding.

We next show that this is the same answer we would get using PCA!

Duality Between MDS and PCA

Both methods rely on the singular value decomposition (SVD) of $\tilde{\mathbf{X}} = H\mathbf{X} = VDU^\top$.

Here is the key insight:

- **PCA**: Finds principal components from the eigenvectors of $\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}} = U(D^\top D)U^\top$.
- **MDS**: Finds embeddings from the eigenvectors of $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top = V(DD^\top)V^\top$.

The columns of U are the principal directions and the scores $\mathbf{y}_1, \ldots, \mathbf{y}_n$ are taken as the first d columns of $\tilde{\mathbf{X}}U = VD$.

As a result, $\mathbf{y}_1, \ldots, \mathbf{y}_n$ are precisely the points obtained by classical MDS.

Spectral Embedding (aka Laplacian Eigenmaps)

Main ideas

Data: $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^m$. Find low dimensional representation $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$.

Links to manifold learning

We look for a truly nonlinear method that is able to learn the underlying manifold.

The main idea is to keep track of local geometry:

- The embedding of $\mathbf{x}_i$ should depend mostly on points close to $\mathbf{x}_i$.

How to keep track of the local geometry in the data?

Construct a weighted graph $G = (V, E, W)$:

- Vertices $V = \{1, 2, \ldots, n\}$ (data points).
- Edges E based on proximity (e.g., k-nearest neighbors or ϵ-neighborhood).
- Weights W_{ij} measure similarity, e.g. $W_{ij} = 1$ or $W_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$.

If $ij \notin E$ we always set $W_{ij} = 0$, also $W_{ii} = 0$ for all $i = 1, \ldots, n$.

Graph Laplacian

Graph Laplacian is the main object encoding the "geometry of the data".

The Laplacian matrix $L \in \mathbb{S}^n$ encodes the structure of the graph:

- Degree matrix D (diagonal): $D_{ii} = \sum_j W_{ij}$, $i = 1, \ldots, n$.
- Graph Laplacian: $L = D - W$, where W is the weight matrix $W = (W_{ij})$.
- Normalized Laplacian: $L_{\mathrm{N}} = D^{-1/2} L D^{-1/2}$.

Important exercise: Show $\mathbf{x}^\top L \mathbf{x} = \frac{1}{2} \sum_{i,j} W_{ij}(x_i - x_j)^2$ for all $\mathbf{x} \in \mathbb{R}^n$.

Properties of L:

- L is positive semi-definite.
- $L\mathbf{1} = \mathbf{0}$, that is, smallest eigenvalue is zero with eigenvector $\mathbf{1}$.
- If G is connected $\mathrm{rank}(L) = n - 1$.

The key idea behind spectral embedding

Fix d. The embedding $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$ is obtained by minimizing:

$$\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2$$

Key insight

High W_{ij} enforces small $\|\mathbf{y}_i - \mathbf{y}_j\|$.

Note: This is still not well defined because $\mathbf{y}_1 = \ldots = \mathbf{y}_n = \mathbf{0}$ is a solution so we need to refine this idea a bit.

Problem reformulation

Let $\mathbf{Y} \in \mathbb{R}^{n \times d}$ be the embedded data matrix. Recall $L = D - W$ and $L\mathbf{1} = \mathbf{0}$.

Proposition

We have: $\frac{1}{2}\sum_{i,j} W_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 = \text{tr}(\mathbf{Y}^\top L\mathbf{Y}) = \text{tr}(\mathbf{Y}^\top D\mathbf{Y}) - \text{tr}(\mathbf{Y}^\top W\mathbf{Y})$

Proof: As for MDS we can show that the matrix $E = \|\mathbf{y}_i - \mathbf{y}_j\|^2$ takes the form

$$E = \text{diag}(\mathbf{Y}\mathbf{Y}^\top)\mathbf{1}\mathbf{1}^\top + \mathbf{1}\mathbf{1}^\top\text{diag}(\mathbf{Y}\mathbf{Y}^\top) - 2\mathbf{Y}\mathbf{Y}^\top$$

and so $\frac{1}{2}\sum_{i,j} W_{ij}\|\mathbf{y}_i - \mathbf{y}_j\|^2 = \frac{1}{2}\text{trace}(WE)$.

D is diagonal and E has zeros on the diagonal and so $\frac{1}{2}\text{trace}(WE) = -\frac{1}{2}\text{trace}(LE)$.

Since $L\mathbf{1} = \mathbf{0}$ we get also that $-\frac{1}{2}\text{trace}(LE) = \text{trace}(L\mathbf{Y}\mathbf{Y}^\top)$.

Introducing constraints to the optimization problem

Constraint 1

To avoid trivial solutions it is convenient to assume $\mathbf{Y}^\top D\mathbf{Y} = I_d$.

Defining $\tilde{\mathbf{Y}} = D^{1/2}\mathbf{Y}$ we get $\tilde{\mathbf{Y}}^\top\tilde{\mathbf{Y}} = I_d$ (orthonormal columns $\tilde{\mathbf{y}}_1, \ldots, \tilde{\mathbf{y}}_d$).

Now $\operatorname{trace}(\mathbf{Y}^\top L\mathbf{Y}) = \operatorname{trace}(\tilde{\mathbf{Y}}^\top L_{\mathrm{N}}\tilde{\mathbf{Y}}) = \sum_{i=1}^{d} \tilde{\mathbf{y}}_i^\top L_{\mathrm{N}}\tilde{\mathbf{y}}_i$.

From PCA: the optimum given by eigenvectors of L_{N} for smallest eigenvalues.

Note that $L_{\mathrm{N}}D^{1/2}\mathbf{1} = D^{-1/2}L\mathbf{1} = \mathbf{0}$ so $\tilde{\mathbf{y}}_0 := D^{1/2}\mathbf{1}$ is a zero-eigenvector.

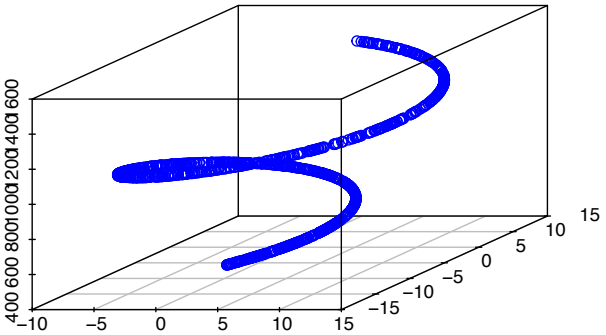
Constraint 2: $\tilde{\mathbf{y}}_0 \perp \tilde{\mathbf{y}}_i$ for $i = 1, \ldots, n$

In addition we assume $\tilde{\mathbf{Y}}^\top D^{1/2}\mathbf{1} = \mathbf{Y}^\top D\mathbf{1} = \mathbf{0}$.

Spectral embedding: minimize $\operatorname{trace}(\mathbf{Y}^\top L\mathbf{Y})$ subject to $\mathbf{Y}^\top D\mathbf{Y} = I_d$ and $\mathbf{Y}^\top D\mathbf{1} = \mathbf{0}$

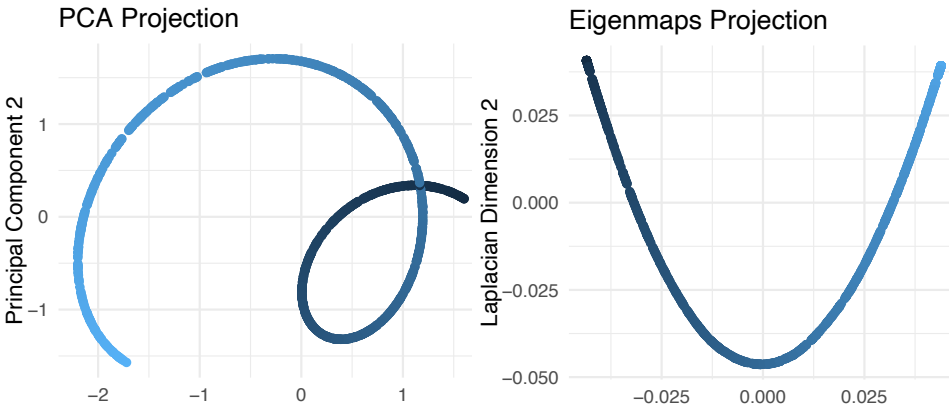
Example: Twisted curve

Consider datapoints lying on the twisted curve as on the picture below:



We now represent these data in 2D comparing PCA and Laplacian Eigenmaps.

- **PCA**: Projects data linearly, collapsing structure.
- **Laplacian Eigenmaps**: Preserves local geometry, unfolding the manifold.



Note that PCA joins points that are far from each other in the original dataset.

Uniform Manifold Approximation and Projection (UMAP)

Introduction to UMAP

UMAP is a nonlinear dimensionality reduction technique that improves on eigenmaps.

Advantages over PCA, MDS, and Eigenmaps:

- Has manifold learning abilities.
- Balances local and global structure.
- Scales efficiently to large datasets.
- More robust to parameter choices.

UMAP is a state-of-the-art data visualization and pattern discovery tool.

UMAP Algorithm Overview

The key idea is similar to the spectral embedding.

1. **Construct k-Nearest Neighbor (kNN) Graph**.
2. **Initialize Embedding** using Laplacian Eigenmaps.
3. **Optimize** embedding via stochastic gradient descent (SGD).

UMAP uses a different loss function than Laplacian Eigenmaps, which makes it, in principle, more robust to parameter choices.

Step 1: Data Graph in the Input Space

Construct k-Nearest Neighbors (kNN) graph; e.g. with $k = 15$.

Define "probabilities" of i, j being connected based on neighbor distances:

$$p_{j|i} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\| - \rho_i}{\sigma_i}\right),$$

where $\rho_i = \min_{k \neq i} \|\mathbf{x}_i - \mathbf{x}_k\|$ and σ_i is a scaling factor.

Symmetrize probabilities:

$$p_{ij} = p_{j|i} + p_{i|j} - p_{j|i} p_{i|j}.$$

Note that the closest neighbor gets always connected with pr. 1.

Step 2 and 3: Data Graph in the Embedding Space and matching

Compute pairwise similarities in low-dimensional space:

$$q_{ij} = \frac{1}{1 + a\|\mathbf{y}_i - \mathbf{y}_j\|^{2b}}, \tag{1}$$

where, by default, $a \approx 1.929$, $b \approx 0.7915$.

The matching between the original and the embedded space is probability-inspired.

Cost Function (Fuzzy Cross-Entropy)

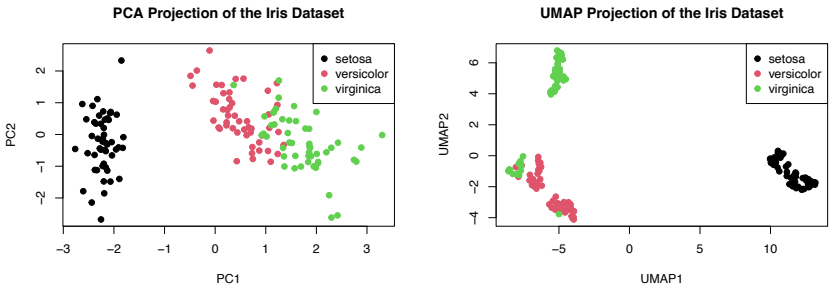
$$c(\mathbf{y}_1, \ldots, \mathbf{y}_n) = \sum_{i \neq j} \left(p_{ij} \log \frac{p_{ij}}{q_{ij}} + (1 - p_{ij}) \log \frac{1 - p_{ij}}{1 - q_{ij}} \right).$$

Here c depends on $\mathbf{y}_1, \ldots, \mathbf{y}_n$ through q_{ij}'s defined in (1).

- Attractive and repulsive forces to balance local and global structure.
- Uses block-coordinate descent to minimize cost.

Example: Iris Dataset

- Comparison of PCA and UMAP on Iris dataset.
- PCA struggles to separate classes clearly.
- UMAP better preserves local and global structures.



Note: Given a new data point UMAP has to be recalculated from scratch!

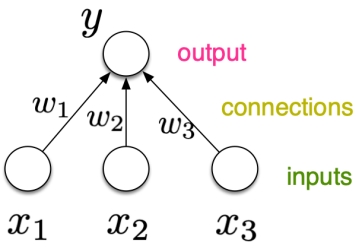
Introducing neural networks

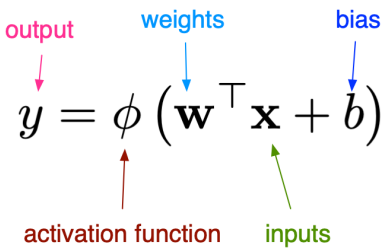
We finish by briefly discussing Autoencoders:

- This is a dimension reduction technique based on neural networks.
- We ignore computational aspects.

A Simple Neuron

For neural nets, we use a simple model for neuron, or **unit**:

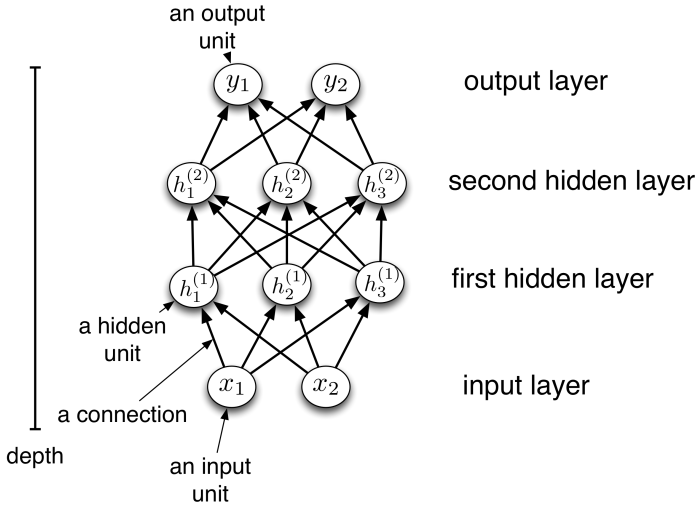




- By throwing together lots of these simple neuron-like processing units, we can do some powerful computations!

A Feed-Forward Neural Network

- A directed acyclic graph
- Units are grouped into layers

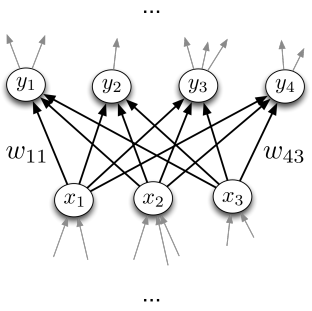


Multilayer Perceptrons

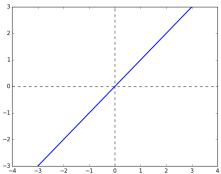
- A multi-layer network consists of fully connected layers.
- In a fully connected layer, all input units are connected to all output units.
- The outputs are a function of the input units:

$$\mathbf{y} = f(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$\phi : \mathbb{R} \to \mathbb{R}$ is applied component-wise.

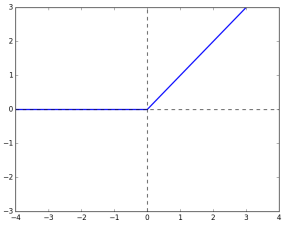


Some Activation Functions



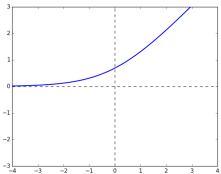
Identity

$\phi(z) = z$



Rectified Linear Unit (ReLU)

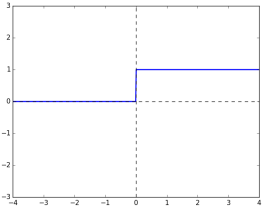
$\phi(z) = \max(0, z)$



Soft ReLU

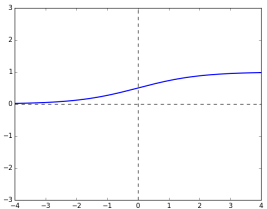
$\phi(z) = \log 1 + e^z$

More Activation Functions



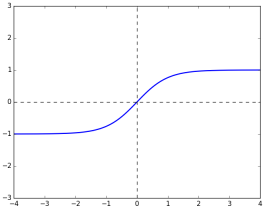
Hard Threshold

$$\phi(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



Logistic

$$\phi(z) = \frac{1}{1 + e^{-z}}$$



Hyperbolic Tangent (tanh)

$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Computation in Each Layer

Each layer computes a function.

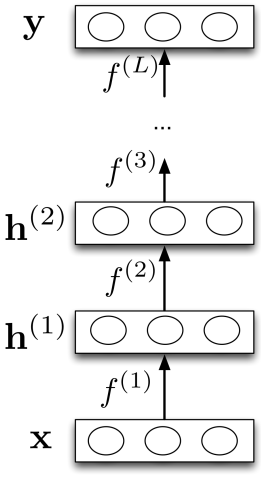
$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x}) = \phi(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)}) = \phi(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

The network computes a composition of functions.

$$\mathbf{y} = f^{(L)} \circ \cdots \circ f^{(1)}(\mathbf{x}).$$

The last layer depends on the task.

- Regression: $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = (\boldsymbol{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)}$
- Classification: $\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)}) = \sigma\big((\boldsymbol{w}^{(L)})^\top \mathbf{h}^{(L-1)} + b^{(L)}\big)$



Expressive Power of Linear Networks

- Consider a linear layer: the activation function was the identity. The layer just computes an affine transformation of the input.
- Any sequence of linear layers is equivalent to a single linear layer.

$$\mathbf{y} = \underbrace{\boldsymbol{W}^{(3)}\boldsymbol{W}^{(2)}\boldsymbol{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

- Deep linear networks can only represent linear functions — no more expressive than linear regression.

Expressive Power of Non-linear Networks

- Multi-layer feed-forward neural networks with non-linear activation functions
- **Universal Function Approximators**: They can approximate any function arbitrarily well.
- True for various activation functions (e.g. thresholds, logistic, ReLU, etc.)

Learning Weights in a Neural Network

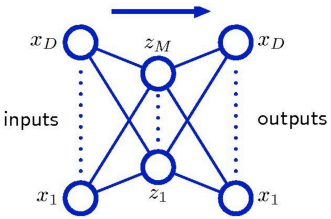
- Goal is to learn weights in a multi-layer neural network using gradient descent.
- Weight space for a multi-layer neural net: one set of weights for each unit in every layer of the network
- Define a loss $\mathcal{L}(\boldsymbol{t}, \mathbf{y}) = \mathcal{L}(\boldsymbol{t}, \mathbf{y}(\mathbf{x}, \boldsymbol{w}))$ and compute the gradient of the cost

$$E(\boldsymbol{w}) \;=\; \tfrac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\boldsymbol{t}_n, \mathbf{y}(\mathbf{x}_n, \boldsymbol{w})),$$

  which is the average loss over all the training examples.
- To calculate $\nabla E(\boldsymbol{w})$ efficiently we use backpropagation. (we omit details here)

Autoencoders

Non-linear Dimension Reduction



- Neural networks can be used for nonlinear dimensionality reduction.
- This is achieved by having the same number of outputs as inputs. These models are called autoencoders.
- Consider a feed-forward neural network that has D inputs, D outputs, and M hidden units, with $M < D$.
- We can squeeze the information through a bottleneck.
- If we use a linear network (linear activation) this is very similar to Principal Components Analysis.

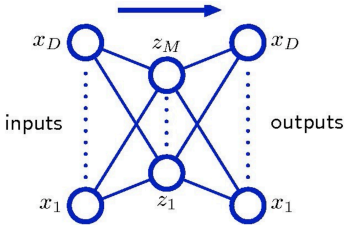
Autoencoders and PCA

- Given an input $\boldsymbol{x}$, its corresponding reconstruction is given by:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{M} w_{kj}^{(2)} \sigma\Big(\sum_{i=1}^{D} w_{ji}^{(1)} x_i \Big), \quad k = 1, ..., D.$$

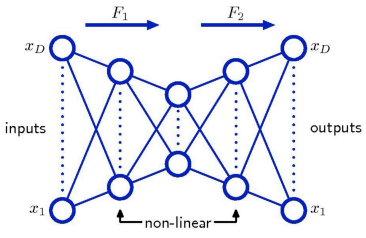
- We learn the parameters $\boldsymbol{w}$ by minimizing the reconstruction error:

$$E(\boldsymbol{w}) = \frac{1}{2N} \sum_{n=1}^{N} \|y(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{x}_n\|^2$$



- In the case when layers are linear:
  - it will learn hidden units that are linear functions of the data and minimize squared error.
  - M hidden units will span the same space as the first M principal components (PCA).

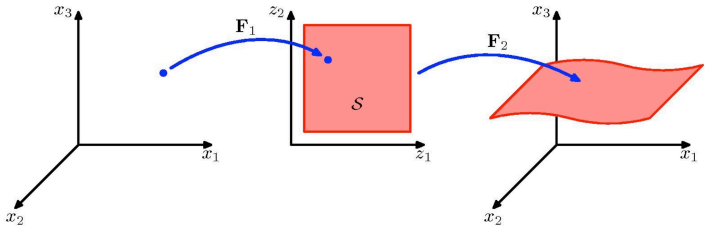
Deep Autoencoders



- We can put extra nonlinear hidden layers between the input and the bottleneck and between the bottleneck and the output.
- This gives nonlinear generalization of PCA, providing non-linear dimensionality reduction.
- The network can be trained by the minimization of the reconstruction error function.
- Much harder to train.

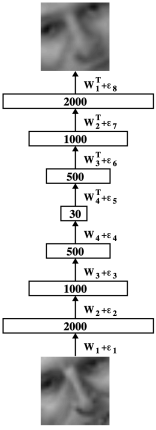
Geometrical Interpretation

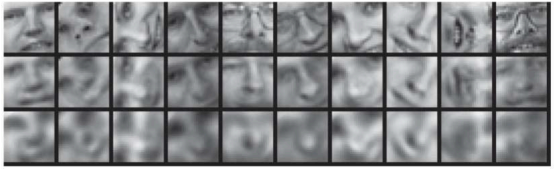
- Geometrical interpretation of the mappings performed by the network with 2 hidden layers for the case of $D = 3$ and $M = 2$ units in the middle layer.



- The mapping F_1 defines a nonlinear projection of points in the original D-space into the M-dimensional subspace.
- The mapping F_2 maps from an M-dimensional space into D-dimensional space.

Deep Autoencoders

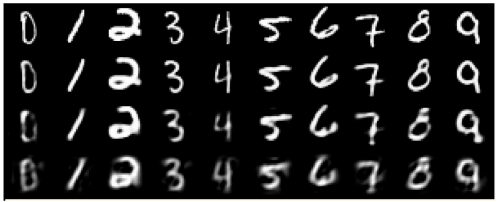




- We can consider very deep autoencoders.
- By row: Real data, Deep autoencoder with a bottleneck of 30 units, and 30-d PCA.

Deep Autoencoders

- Similar model for MNIST handwritten digits:



- Deep autoencoders tend to produce better reconstructions.
- Of course, they are much more involved computationally.

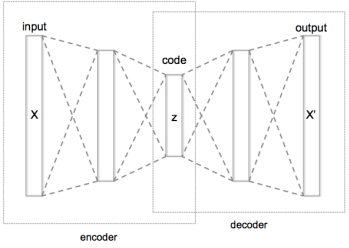
Autoencoders: Summary

Autoencoders reconstruct their input via an encoder and a decoder.

- **Encoder**: $e(x) = z \in F, \quad x \in X$
- **Decoder**: $d(z) = \tilde{x} \in X$
- where X is the data space, and F is the feature (latent) space.
- z is the code, compressed representation of the input, x. It is important that this code is a bottleneck, i.e. that

$$\dim F \ll \dim X$$

- Goal: $\tilde{x} = d(e(x)) \approx x$.



Issues with (deterministic) Autoencoders

- **Issue 1**: Proximity in data space does not mean proximity in feature space
  - The codes learned by the model are deterministic, i.e.

$$g(x_1) = z_1 \implies f(z_1) = \tilde{x}_1$$
$$g(x_2) = z_2 \implies f(z_2) = \tilde{x}_2$$

  - but proximity in feature space is not "directly" enforced for inputs in close proximity in data space, i.e.

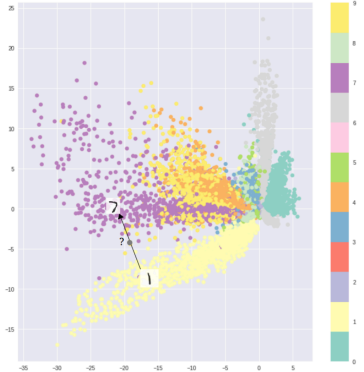
$$x_1 \approx x_2 \not\implies z_1 \approx z_2$$

  - The latent space may not be continuous, or allow easy interpolation.

One issue

Proximity in data space does not mean proximity in feature space.

- If the space has discontinuities (eg. gaps between clusters), the decoder may generate unrealistic outputs.



Two dimensional latent space for the MNIST digit data.(Image credit: I. Shafkat)