Apprentissage, réseaux de neurones et modèles graphiques (RCP209)
Neural Networks and Deep Learning

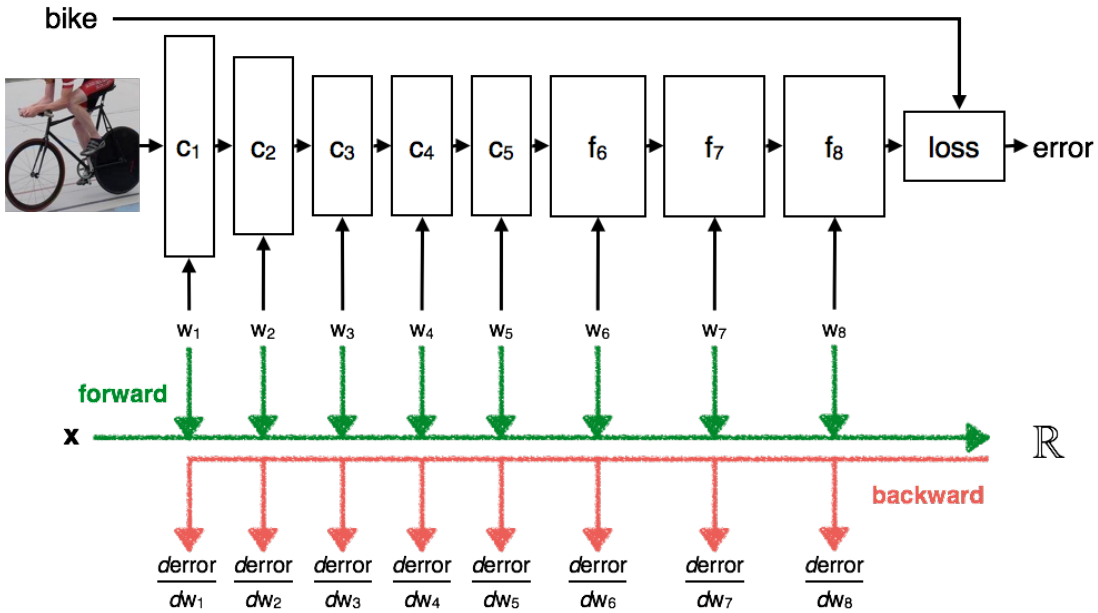
Nicolas Thome
Prenom.Nom@cnam.fr
`http://cedric.cnam.fr/vertigo/Cours/ml2/`

Département Informatique
Conservatoire Nationnal des Arts et Métiers (Cnam)

Outline

Deep Learning: resources for the community

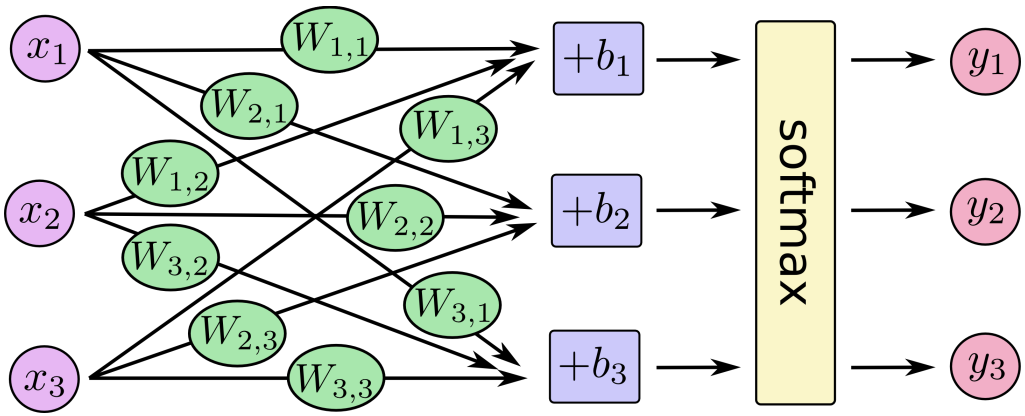
- Formal training of deep CNNs straightforward (backprop)
- Efficient CNN implementation far from trivial, especially convolution
- Libraries made available in the community:
  - Caffe / Decaf : script, non modular
  - MatConvNet (matlab): easy, Torch (Lua): efficiency, modularity
  - TensorFlow / Theano / PyTorch (python): auto-differentiation
  - Keras: wrapper on top of TensorFlow / Theano



Keras: simple example

Logistic Regression

$$p(\hat{y_{c,i}}|\mathbf{x}_i)) = \frac{e^{\langle \mathbf{x_i}; \mathbf{w_c} \rangle + b_c}}{\sum_{c'=1}^{K} e^{\langle \mathbf{x_i}; \mathbf{w_{c'}} \rangle + b_{c'}}}$$



- Example in MNSIT: $K = 10$ classes
- # parameters: $784 * 10 + 10 = 7850$

Keras: simple example

Logistic Regression

- Define an (empty) feedfoward network

```
from keras.models import Sequential
model = Sequential()
```

1

- Add fully connected layer (size 10) + softmax activation

```
from keras.layers import Dense, Activation
model.add(Dense(10, input_dim=784, name='fc1'))
model.add(Activation('softmax'))
```

2

Keras: simple example

Logistic Regression

- Compile model with cross-entropy loss

```
from keras.optimizers import SGD
learning_rate = 0.5
sgd = SGD(learning_rate)
model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics=['accuracy'])
```

- Optimize model parameters to fit training data (*e.g.* MNIST)

```
from keras.datasets import mnist
# MNIST data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# + some pre-processing ... and fit model to data
model.fit(X_train, y_train,batch_size=128, epochs=20,verbose=1)
```

Keras: more complex examples

- Design more complex model by adding layers : fully connected, convolution, non-linearity, pooling, *etc*
- Code for training remains unchanged (back-prop does the job)

```
s=(5,5)
ish=(28,28,1)
model = Sequential()
model.add(Conv2D(32,kernel_size=s,activation='sigmoid',input_shape=ish,padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (5, 5), activation='sigmoid', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='sigmoid'))
model.add(Dense(nb_classes, activation='softmax'))
```

Deep Learning Modules

Non-linearities

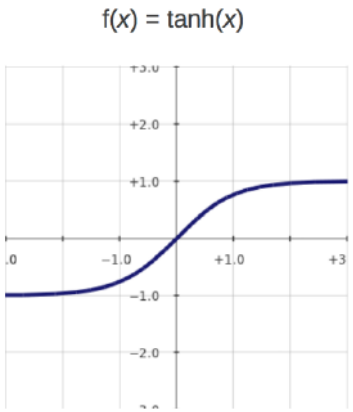
- Rectified Linear Unit (ReLU): $y = 0$ if $x < 0$, $y = x$ otherwise
- Solving vanishing gradients problems $\Rightarrow$ faster learning / convergence

RELU Nonlinearity

- Standard way to model a neuron

$$f(x) = tanh(x) \quad \text{or} \quad f(x) = (1 + e^{-x})^{-1}$$

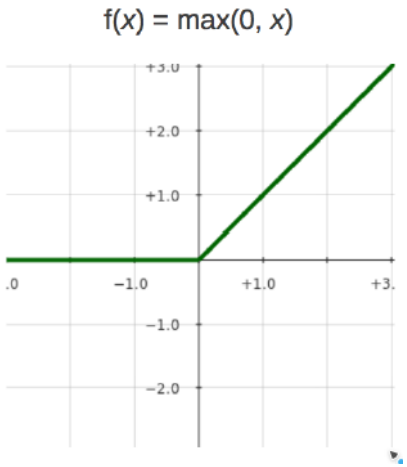
Very slow to train



- Non-saturating nonlinearity (RELU)

$$f(x) = max(0, x)$$

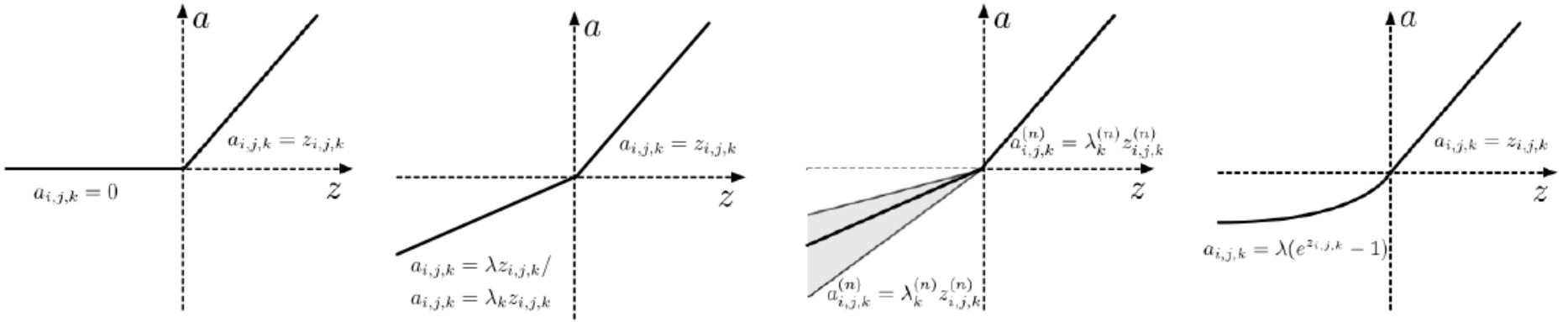
Quick to train



Deep Learning Modules

Non-linearities, ReLU variants

- Leaky ReLU (LReLU): λ is empirically predefined
- Parametric ReLU (PReLU) : λ_k is learned from training data
- Randomized ReLU (RReLU): λ_k^n is a random variable which is sampled from a given uniform distribution in training and keeps fixed in testing
- Exponential Linear Unit (ELU): λ fixed



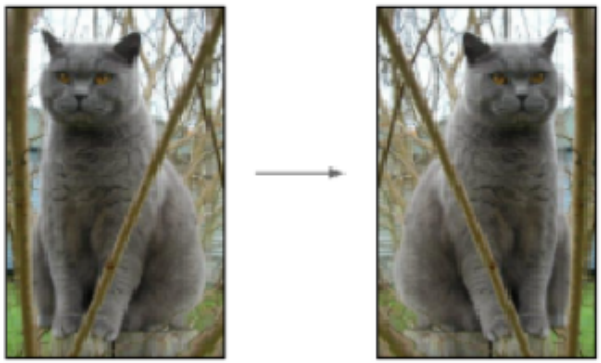
(a) ReLU (b) LReLU/PReLU (c) RReLU (d) ELU

From Gu *et. al.* [GWK+15]

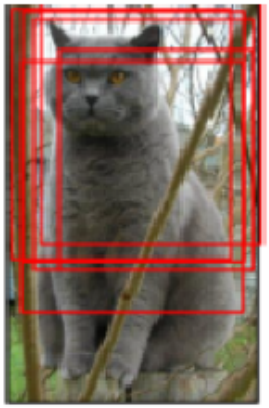
Deep Learning Modules

Training: data-augmentation

- Jittering, mirroring, color perturbation, rotation, stretching, shearing, lens distortions, *etc* of the original images
- Increases # training samples, adds robustness to irrelevant variations
- **Done in train AND in test**



Flip horizontally



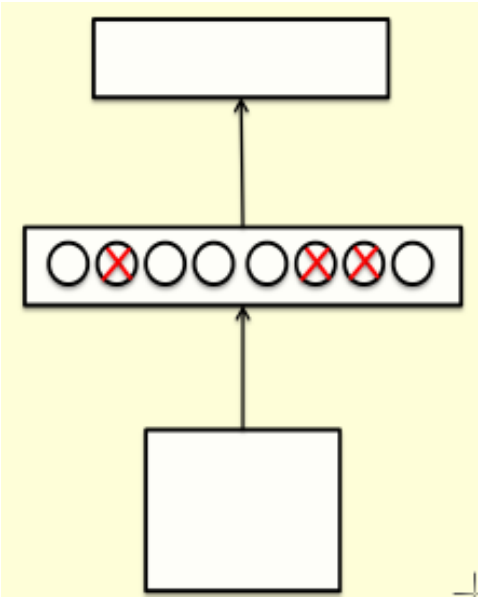
Random crops/scales

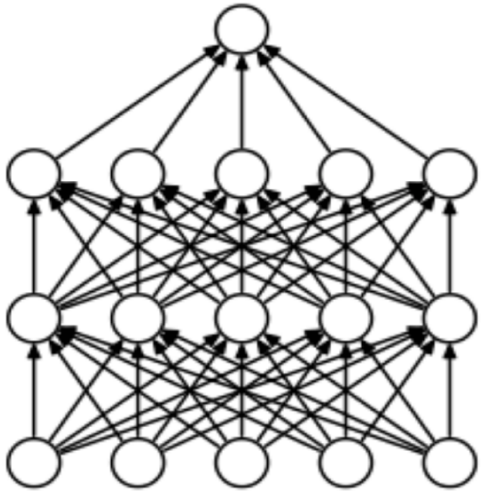
Color jittering

Deep Learning Modules

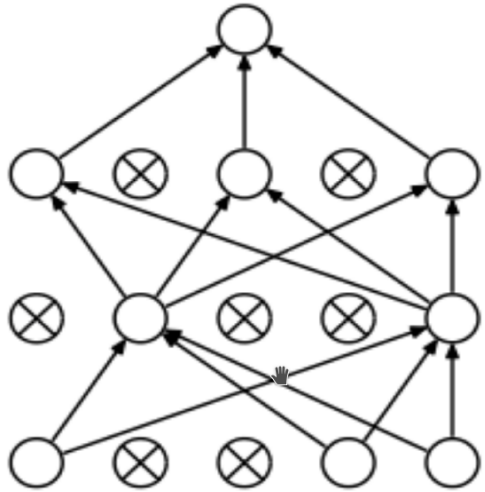
Training: droupout

- Randomly omit each hidden unit with probability 0.5
- **Regularization technique**, limits over-fitting (better generalization)
  - Pulls the weights towards what other models want, useful to prevent co-adaptation (feature only helpful when other specific features present)
  - May be viewed as averaging over many NN
  - Slower convergence





Standard Neural Net



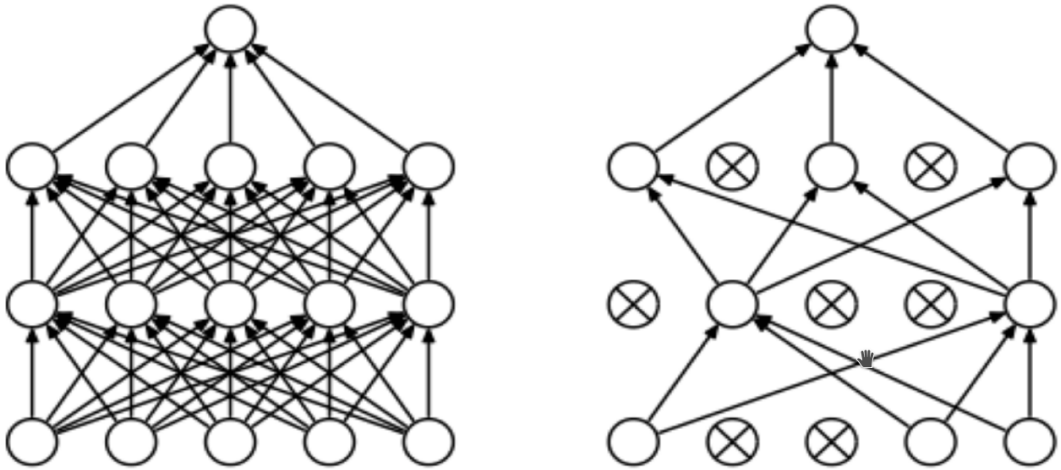
After applying dropout.

Credits: Geoffrey E. Hinton, NIPS 2012

Deep Learning Modules

Training: droupout

- What to do at test time ?
  - Sample many different architectures and take the geometric mean of their output distributions
  - Faster alternative: use all hidden units (but after halving their outgoing weights)
  - Equivalent to the geometric mean in case of single hidden layer
  - Pretty good approximation for multiple layers



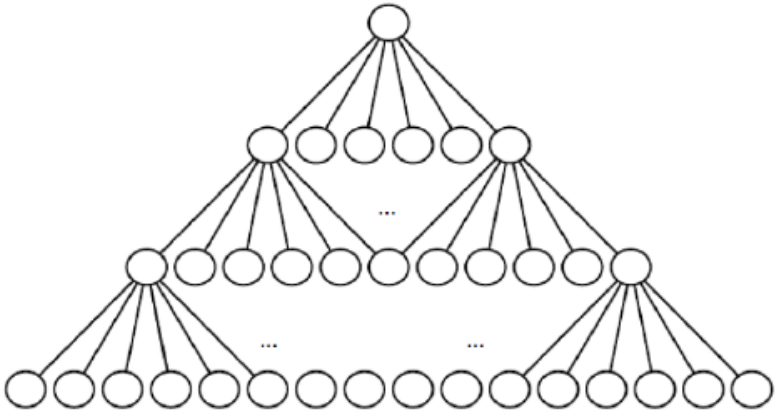
Standard Neural Net | After applying dropout.

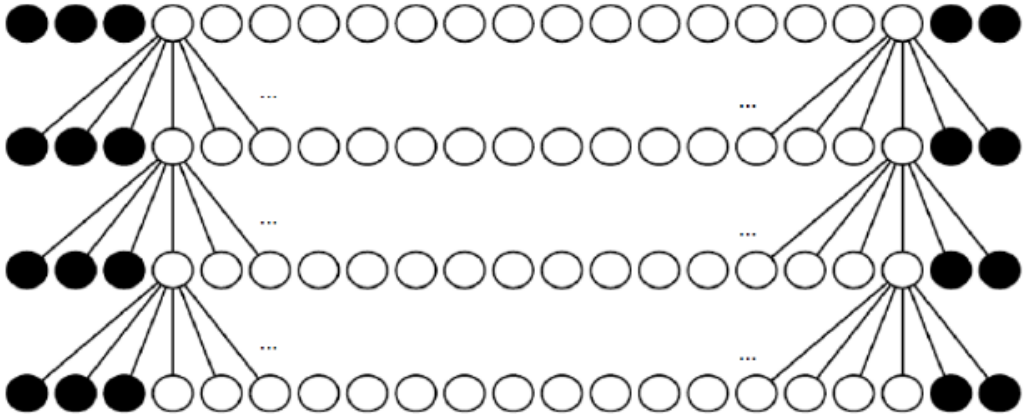
Credits: Geoffrey E. Hinton, NIPS 2012

Deep Learning Modules

Padding, *e.g.* zero-padding

- To avoid shrinking the spatial extent of the network rapidly

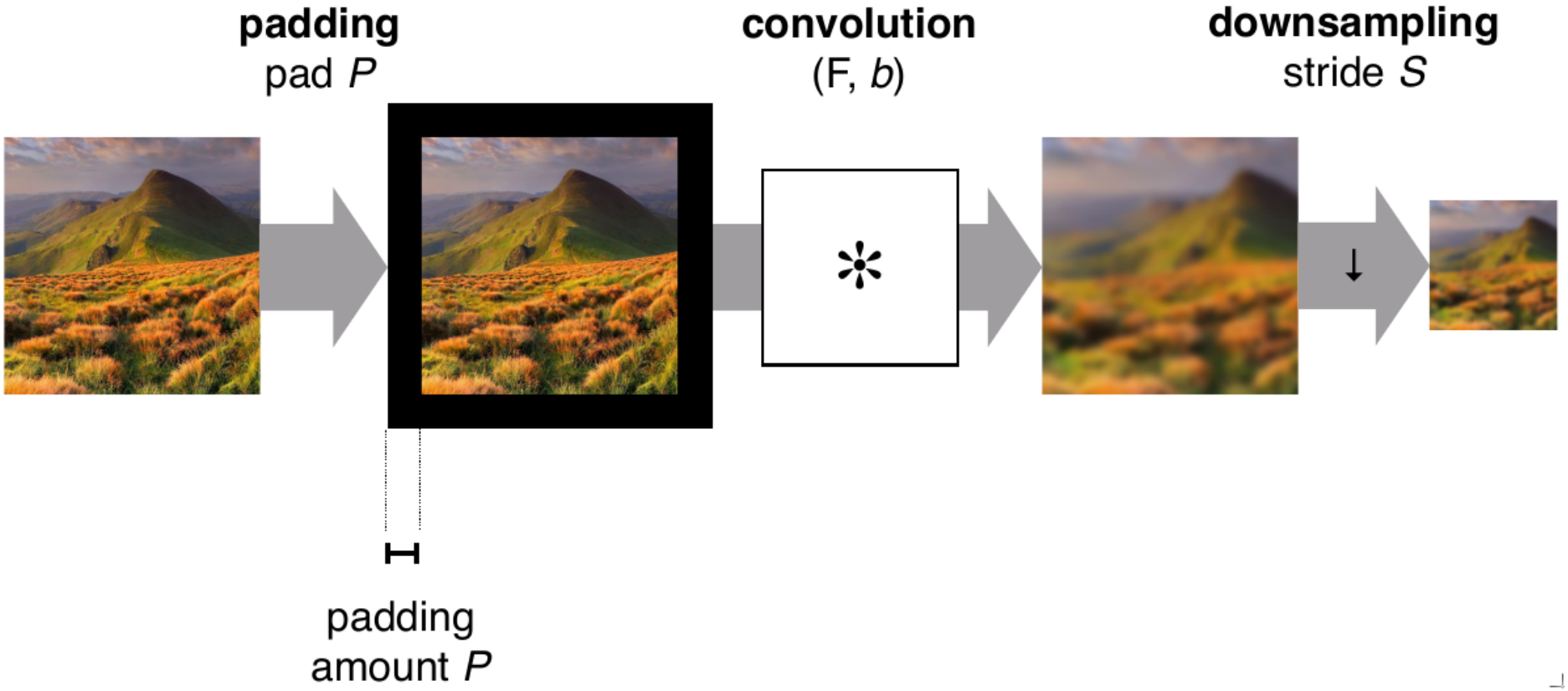




Deep Learning Modules

Padding, *e.g.* zero-padding

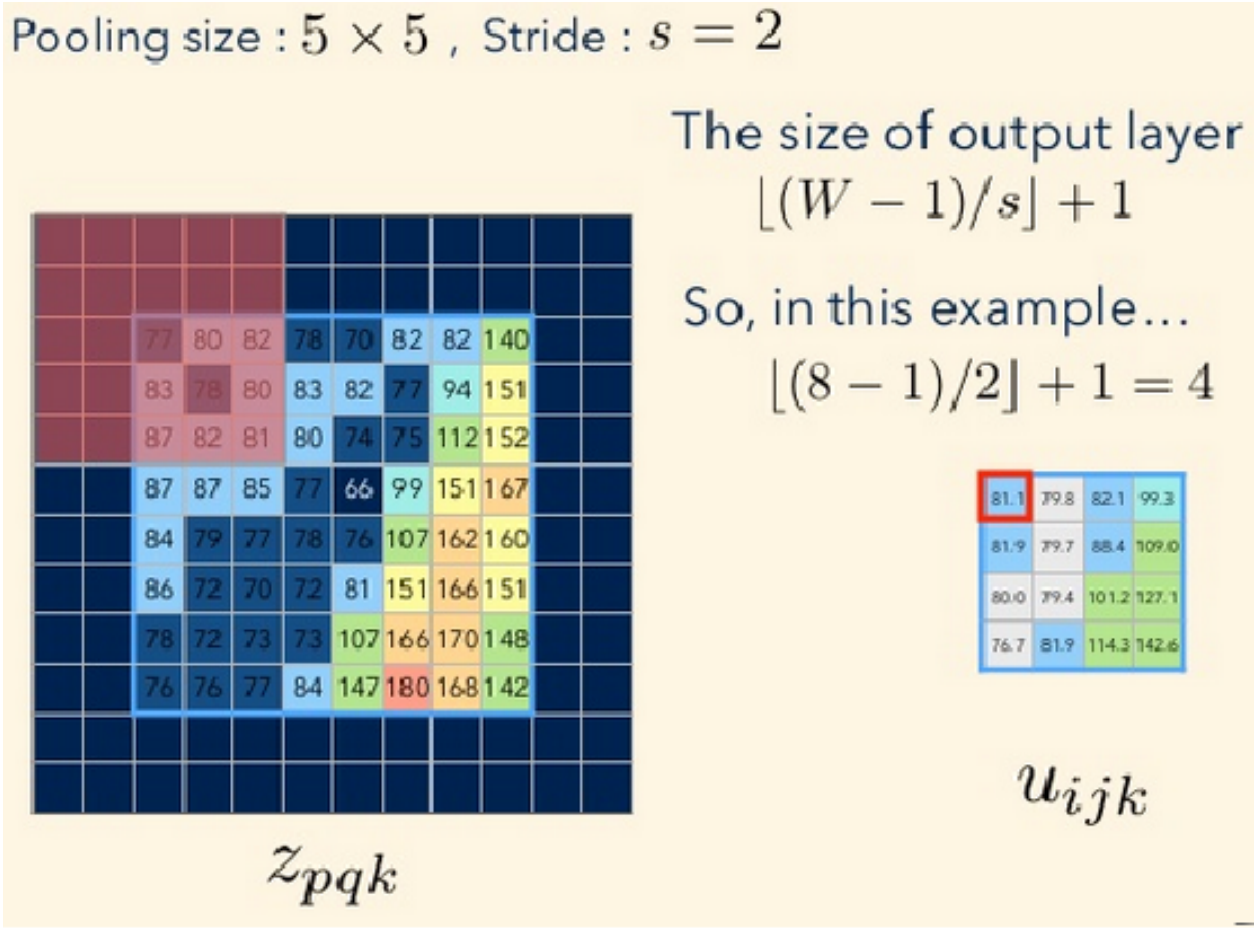
- Ex for images:



Credit: A. Vedaldi

Deep Learning Modules

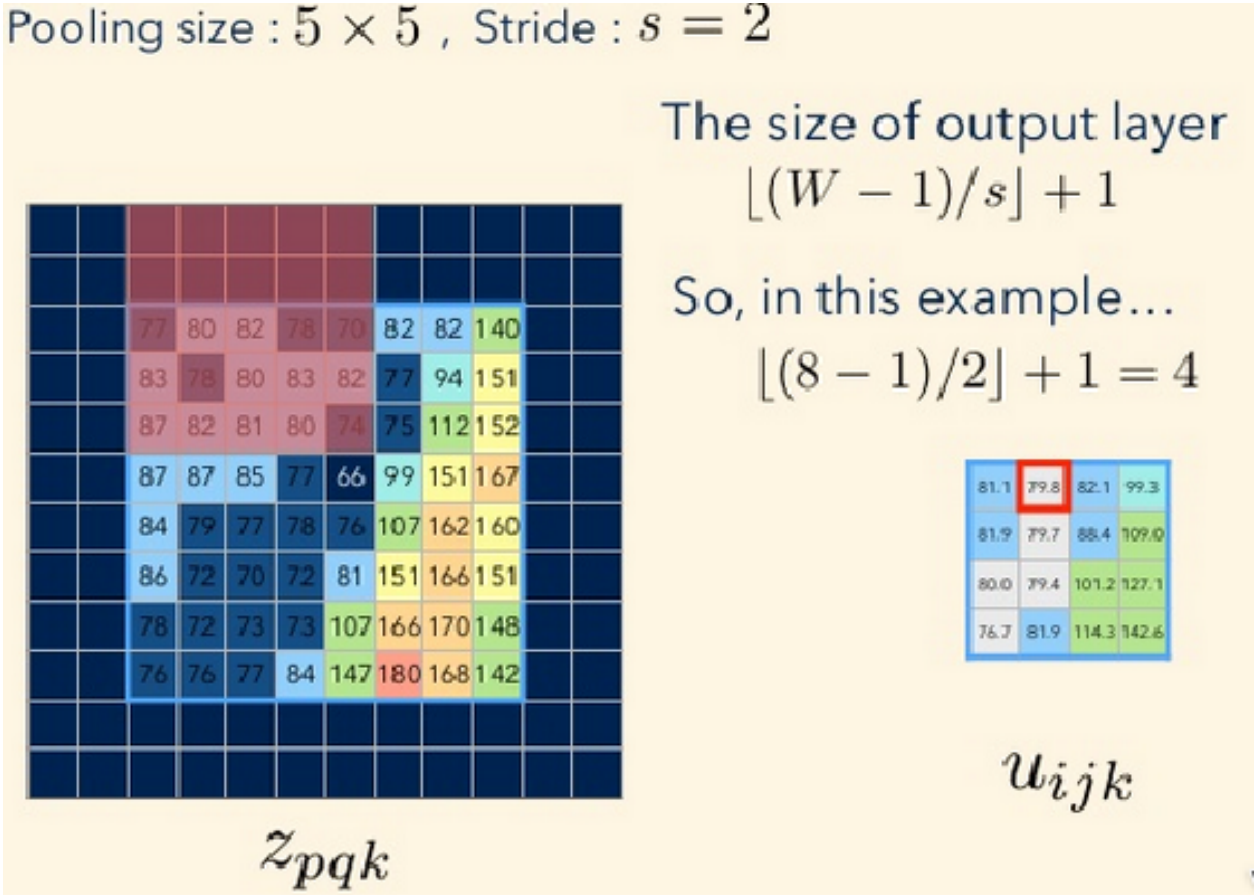
Overlapping Pooling



@Ken'ichi

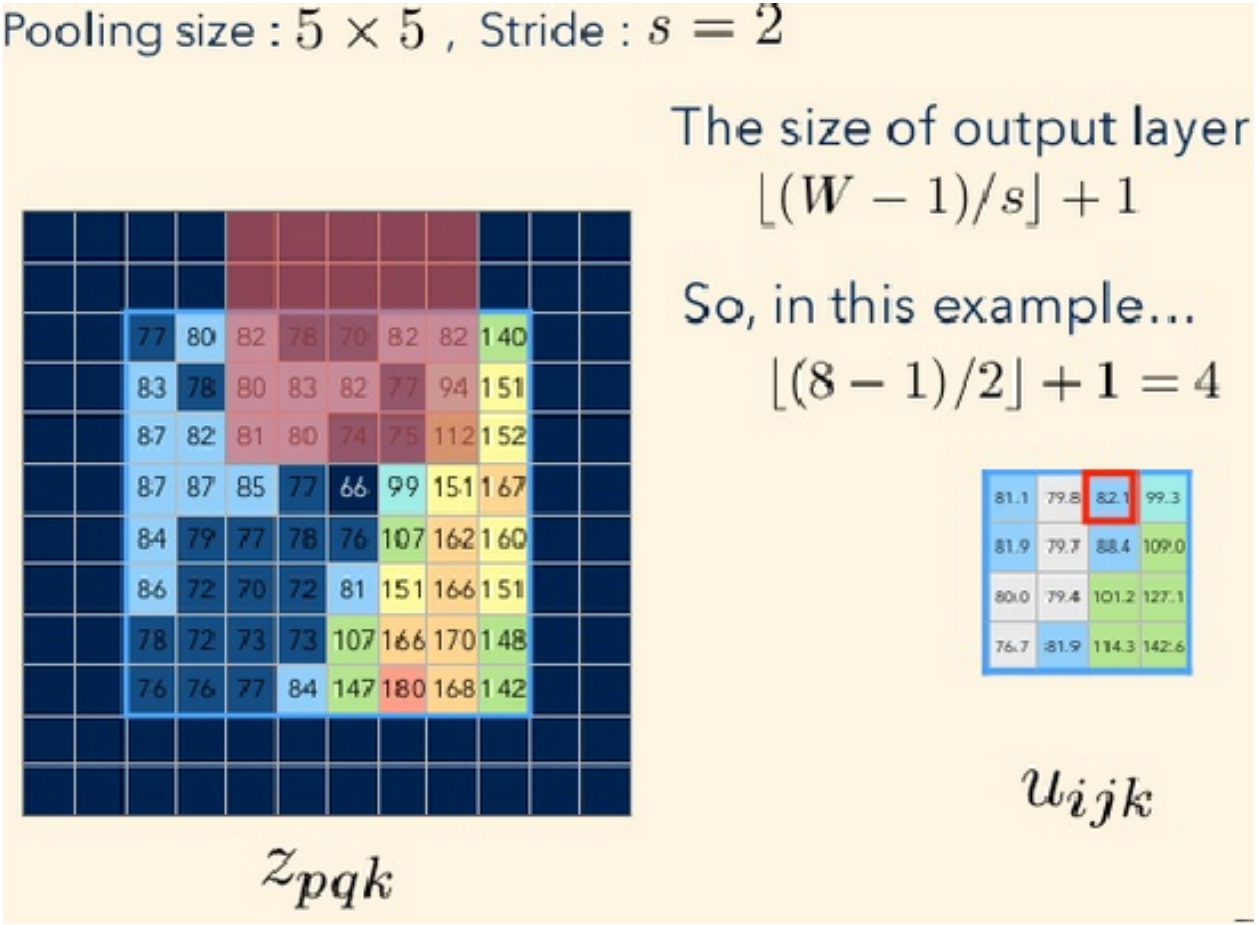
Deep Learning Modules

Overlapping Pooling



Deep Learning Modules

Overlapping Pooling

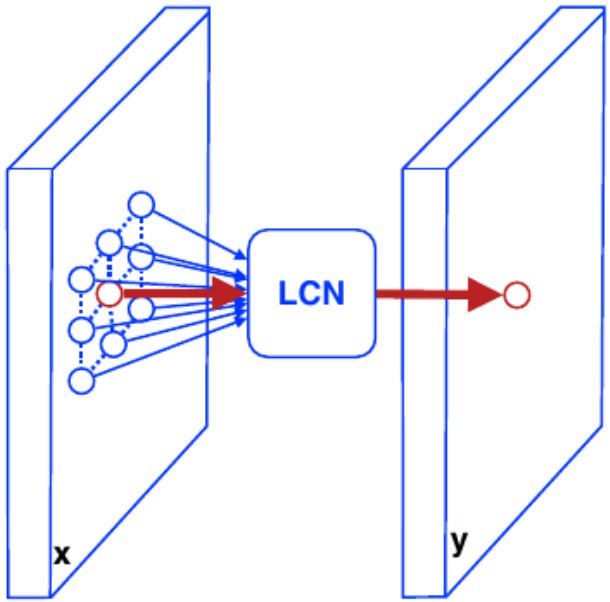


@Ken'ichi

Deep Learning Modules

Local Response Normalization

Normalise image/feature patches



$$y_{ijq} = \frac{x_{ijq} - \mu_{ijq}}{\sigma_{ijq}}$$

$$\mu_{ijq} = \frac{1}{|\mathcal{N}(i,j)|} \sum_{(u,v)\in\mathcal{N}(i,j)} y_{uvq}$$

$$\sigma^2_{ijq} = \frac{1}{|\mathcal{N}(i,j)|} \sum_{(u,v)\in\mathcal{N}(i,j)} (y_{uvq} - \mu_{ijq})^2$$

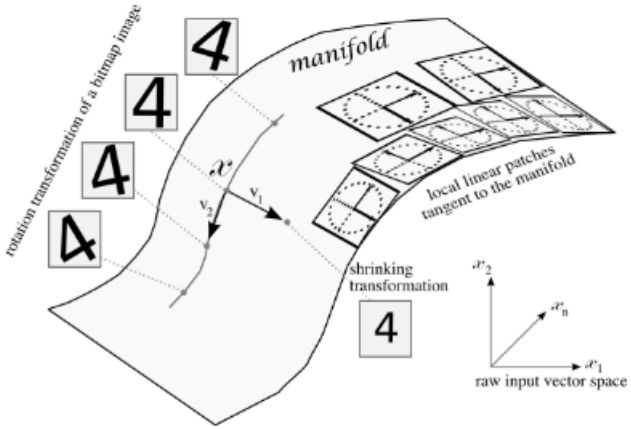
Credit: A. Vedaldi

Outline

Manifold Learning

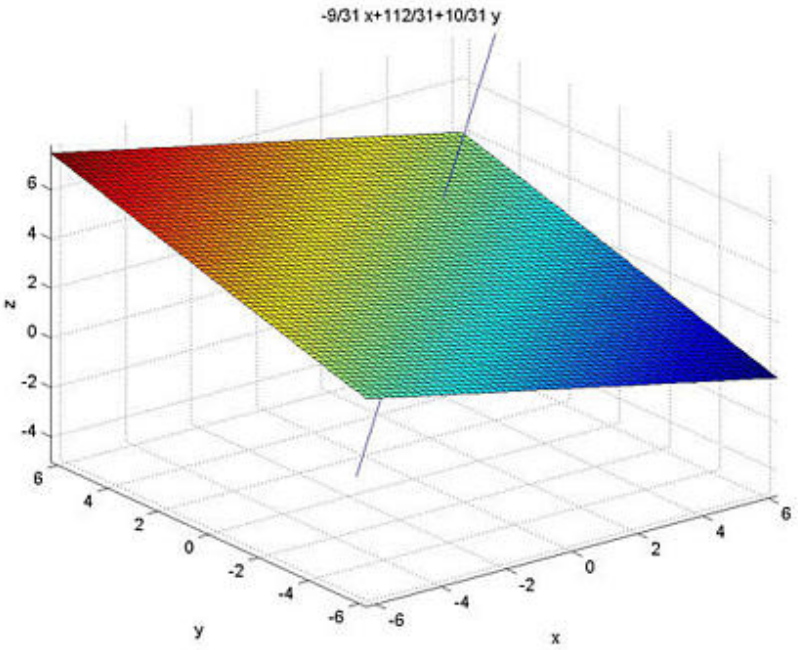
Manifold

- A curve is a manifold of dimension 1
- A surface is a manifold of dimension 2
- A manifold of dimension N is a locally euclidean space
- **Idea in manifold learning:** data lye in a (low-dimensional) manifold

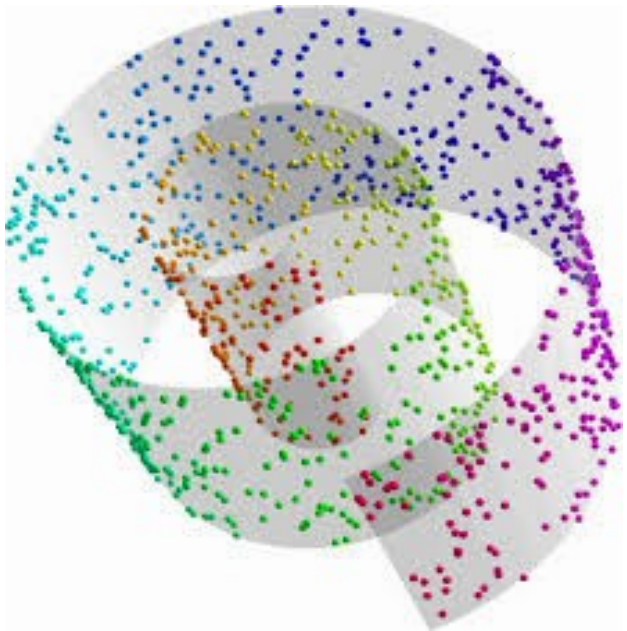


Manifold Learning

Linear and non-linear Manifolds



Linear manifold

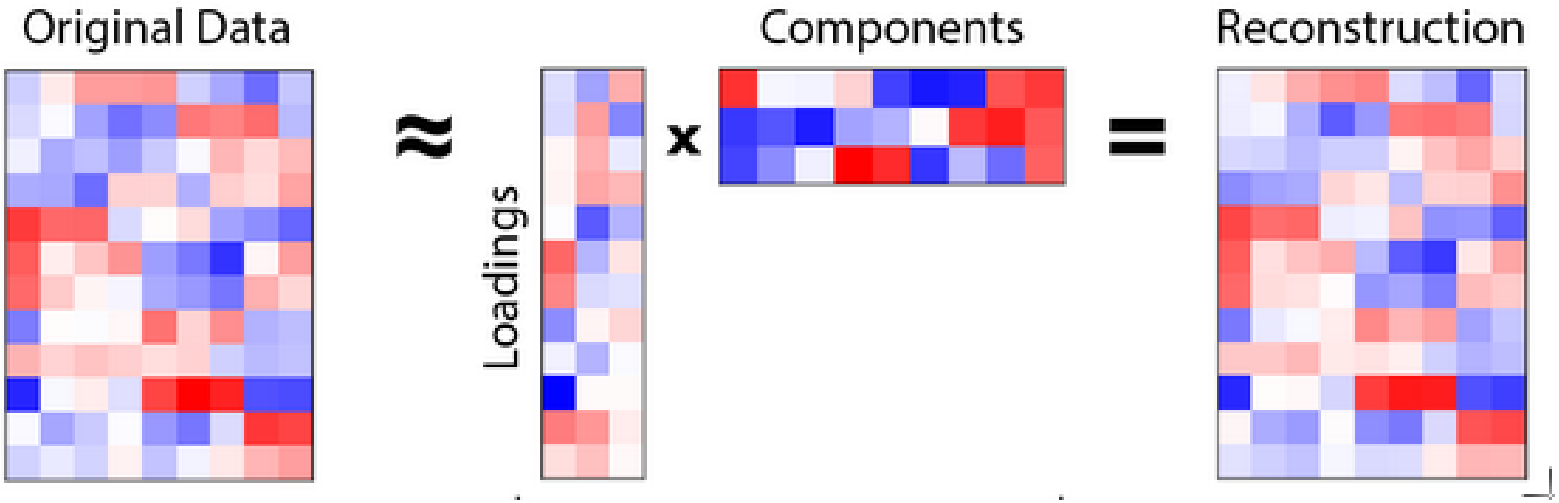


Non Linear Manifold

Linear Manifold Learning

Basis

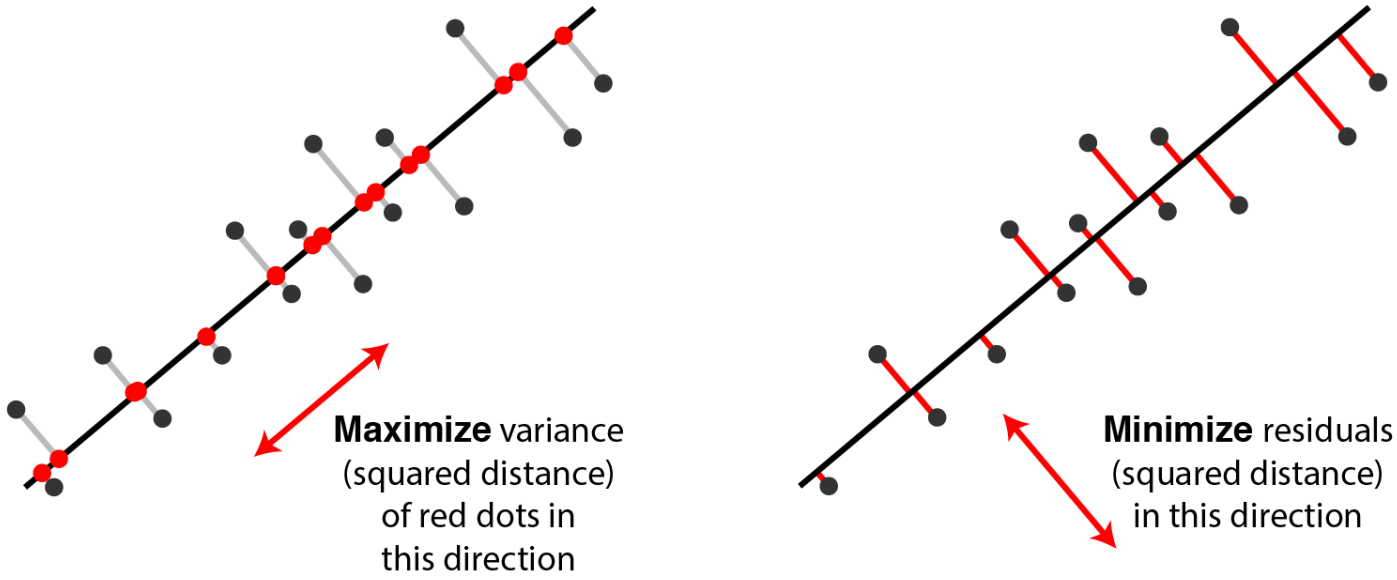
- Points in $\mathbb{R}^p$ vectorial space $\Rightarrow$: n points represented by $\mathbf{X}_{p\times n}$ matrix
- Looking for a new basis to represent data: $\mathbf{X}_{p\times n} = \mathbf{A}_{p\times p}\mathbf{S}_{p\times n}$
  - $\mathbf{A}$ columns: new basis vectors in $\mathbb{R}^p$
  - $\mathbf{S}$ columns: Projection of each example $\mathbf{A}$: p coefficients
- Often related to dimension reduction: only keep a subset of vector basis, $k < p$, $\mathbf{X}_{p\times n} \approx \mathbf{A}_{p\times k}\mathbf{S}_{k\times n}$
- Goal: Learning $\mathbf{A}$ and $\mathbf{S}$ from data
  - Changing representation space: $\mathbf{S}$ better than $\mathbf{X}$ to describe data



Linear Manifold Learning

Principal Component Analysis (PCA) Formulation

- Two equivalent views of PCA
  1. Maximizing projected variance
  2. Minimizing projected reconstruction error



Linear Manifold Learning

Principal Component Analysis (PCA) Formulation

- Goal: seeking vector leading to maximal projected variance
- Π projection of X on a vectorial line $\mathcal{V}$ defined by unit vector $\overrightarrow{v}$ ($\|\overrightarrow{v}\| = 1$): $\Pi = vv^T$
- Projected variance :

$$\sigma_v^2(X) = \frac{1}{n}\sum_{i=1}^{n} \|\Pi(X_i - g)\|^2 = \frac{1}{n}\sum_{i=1}^{n} \Pi(X_i - g)^T \, \Pi(X_i - g)$$

- It can be shown that $\sigma_v^2(X) = v^T \, \Sigma \, v$
  - $\Sigma = \frac{1}{n}\sum_{i=1}^{n}(X_i - g)(X_i - g)^T$ variance-covariance X matrix .

Principal Component Analysis (PCA)

Solution

- Maximize :

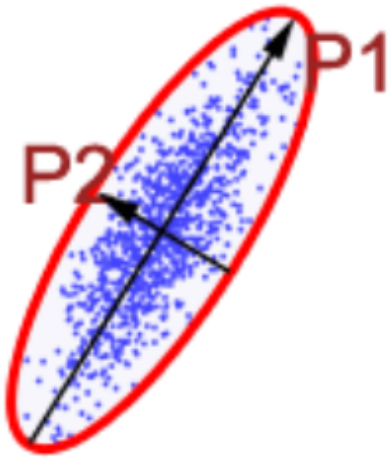
$$\max_{v}(v^T \Sigma v), \text{ st } ||\vec{v}|| = 1 \quad (1)$$

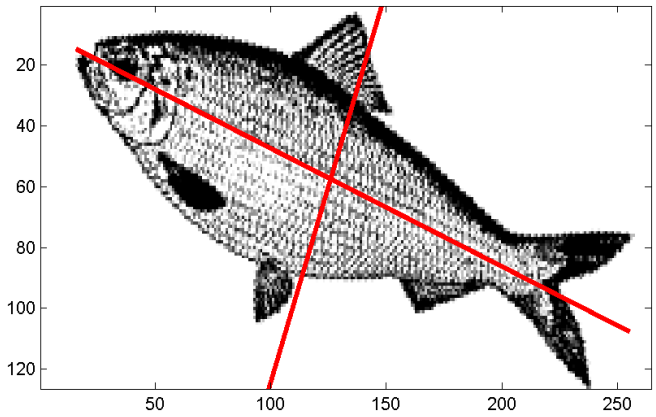
- Necessary condition: $\Sigma v = \lambda v$, $\lambda \in \mathbb{R}$
  - Solution: Σ diagonalization
  - λ_i : eigenvalue of i^{st} eigenvector $\lambda_i = \sigma^2_{v_i}(X)$
  - Sufficient condition: choosing eigenvector with largest eigenvalue
  - Extension to several vectors : look for i^{st} vector maximizing Eq. (1) and orthogonal to previous $i-1$
  - Orthonormal basis of eigenvectors, sorted by decreasing eigenvalues, is the solution to the problem

Principal Component Analysis (PCA)

Visualization

- Points in $\mathbb{R}^2$ plane
- 1^{ex} direction of maximal variation
- PCA solution gives best fitting ellipse





Principal Component Analysis (PCA)

Application: compression

- Various contexts:
  1. Compress a set of n images, each image with p pixels : $\mathbf{X}$
  2. Compress an image: application for a set of n patches with p pixels
  3. Compress an image: $\mathbf{X}$ matrix $\Leftrightarrow$ *image*, thus compressing the n columns, each column with p (# rows) dimensions
- Σ Diagonalization and keeping $k << p$ eigenvectors
- Data reconstruction in the sub eigenspace
- Compression : we have $2k + 1$ vectors ($\mathbf{A}$, $\mathbf{S}$ et mean) : $tx = \frac{2k+1}{n}$

Principal Component Analysis (PCA) : compression

17.7:1 compression
14 principal components

12.5:1 compression
20 principal components

8.4:1 compression
30 principal components

6.3:1 compression
40 principal components

4.2:1 compression
60 principal components

2.8:1 compression
90 principal components

2.1:1 compression
120 principal components

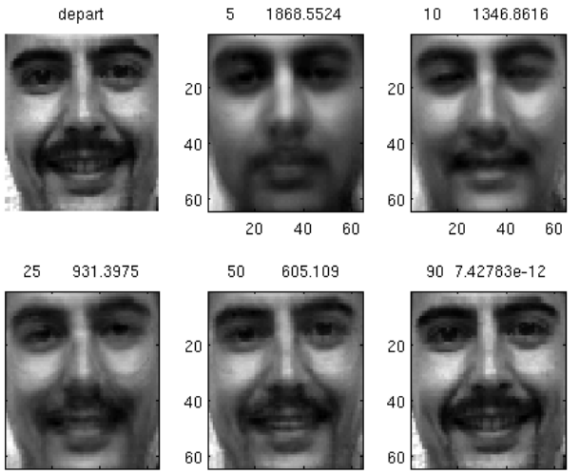
1.7:1 compression
150 principal components

1.4:1 compression
180 principal components

Principal Component Analysis (PCA)

Application : face recognition

- Previous case 1) : we have a set of n images (of faces), each image with p pixels : $\mathbf{X}$
- Σ Diagonalization and keeping $k << p$ eigenvectors, "eigenfaces"
- Data reconstruction in the sub-space corresponding to Eigenfaces



- Identification: project a given image on the sub-space corresponding to Eigenfaces
- Find k-nn in the eigen sub-space $\Rightarrow$ classification

Linear Manifold Learning

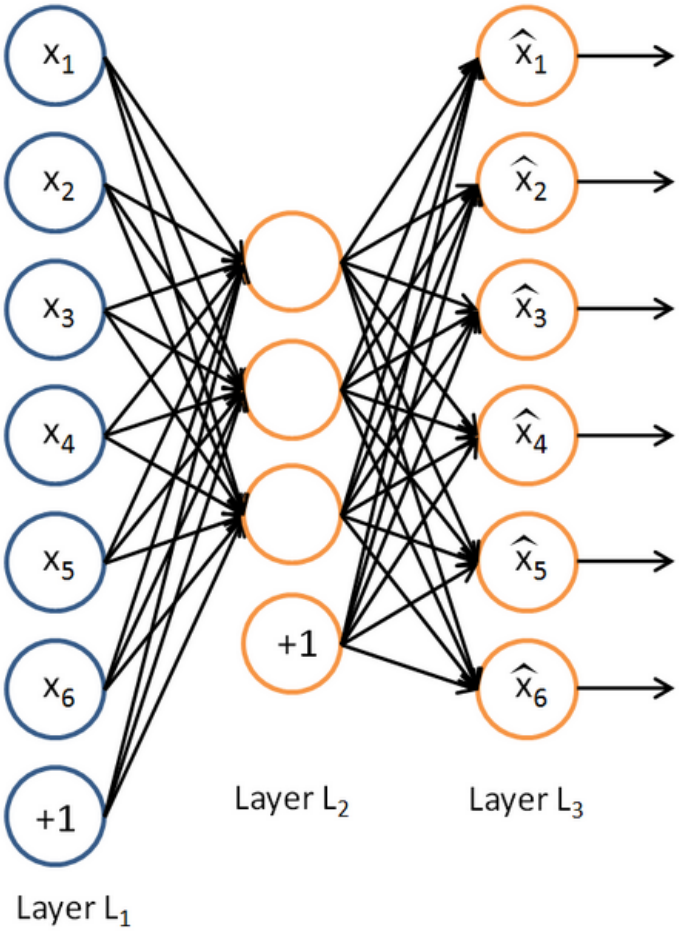
Auto-encoders

- $\mathbf{z} = f(\mathbf{W}\mathbf{x})$
- $\tilde{\mathbf{x}} = g(\mathbf{W}^t\mathbf{x})$
- Auto-encoder objective function:

$$C = \sum_{i=1}^{N} ||\mathbf{x}_i - \tilde{\mathbf{x}}||^2$$

- If $f = g = Id$ (linear auto-encoder): $\sim$ PCA (reconstruction formulation):

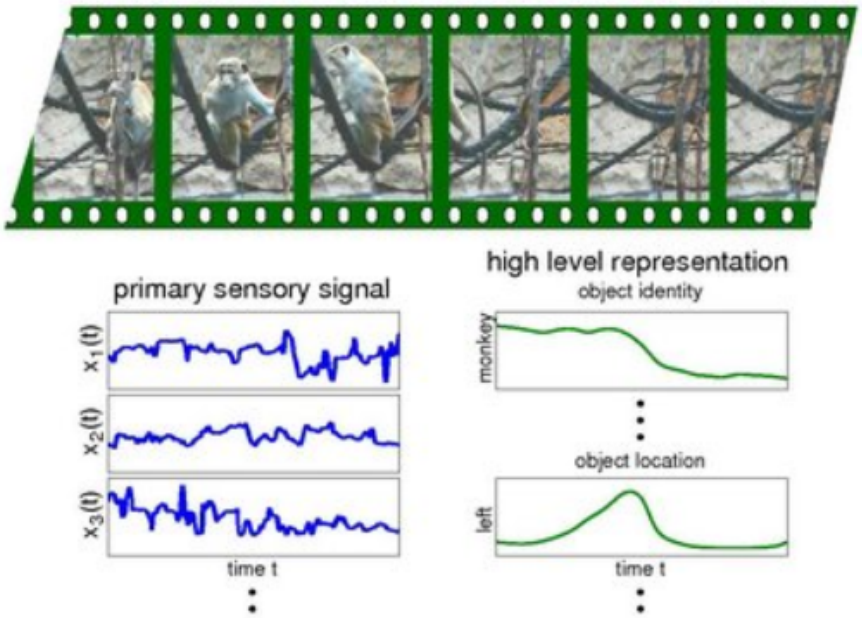
$$C = \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{W}^t\mathbf{W}\mathbf{x}||^2$$



Linear Manifold Learning

Slow Feature Analysis (SFA)

- Intuition: noisy measurements, stable perceptions

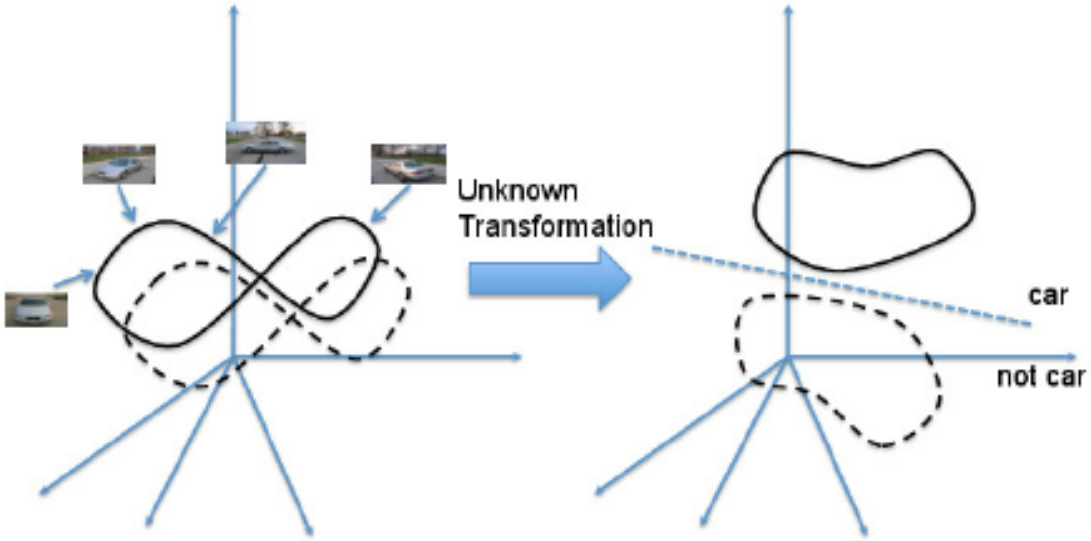


Source : http://www.scholarpedia.org/article/Slow_feature_analysis

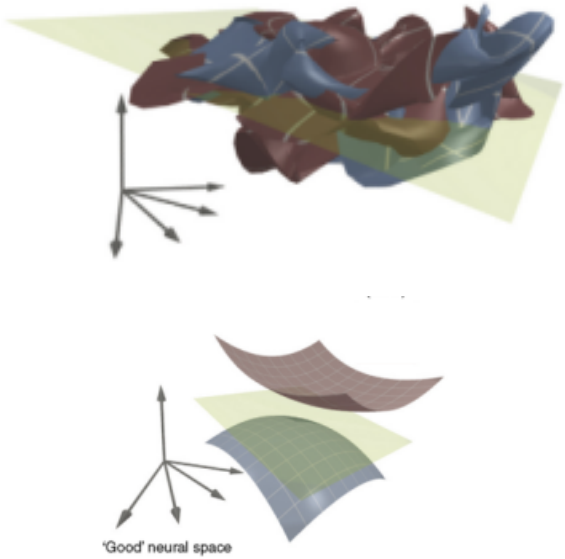
- Goal: learning "stable" representations from input signal
  - Related to the manifold untangling problem
- Mean: Extract low signal components, *i.e* "slow down" signal
- Ex application for images / videos

Slow Feature Analysis

Manifold Untangling



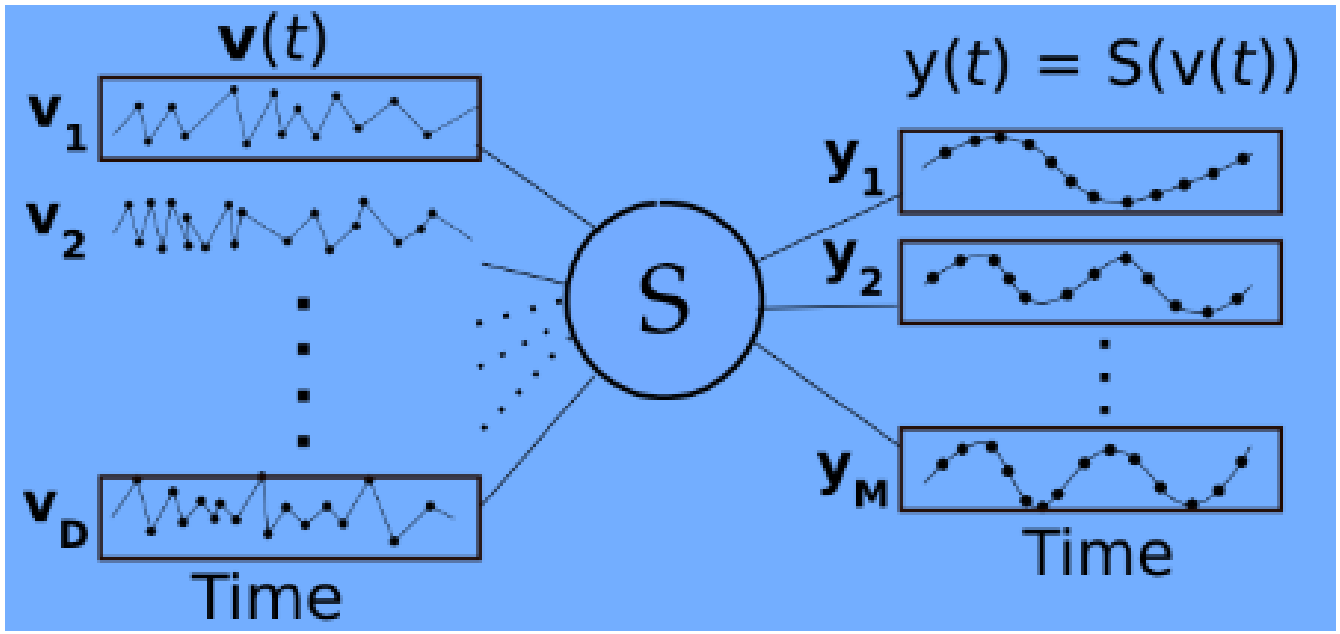
Credit: DiCarlo



Slow Feature Analysis

Formulation

- Still in the basis context (*e.g.* PCA), but different criterion
- Input : D-dimensional temporal signal $\mathbf{v}(t) = [v_1(t)v_2(t)...v_D(t)]^T$
- Output : M-dimensional temporal signal $\mathbf{y}(t) = [y_1(t)y_2(t)..y_M(t)]^T$
- $y_j(t) = S_j v(t)$, $\forall t$ et $\mathbf{S} \in \mathbb{R}^{D \times M}$



Slow Feature Analysis

Formulation

- $y_j(t) = S_j v(t)$, $\forall t$ and $\mathbf{S} \in \mathbb{R}^{D \times M}$. Let us define:
  - $\langle y \rangle_t$ temporal average of y
  - $\dot{y}$ temporal derivative of y
- Objective function :

$$\min_{S_j} \langle \dot{y_j}^2 \rangle_t \tag{2}$$

Under constraints:

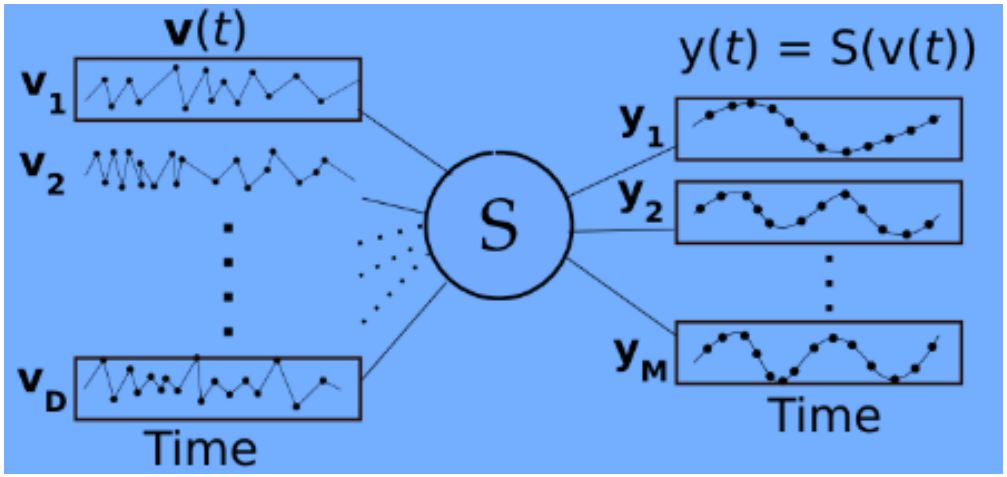
1. $\langle y_j \rangle_t = 0$ (zero mean)
2. $\langle y_j^2 \rangle_t = 1$ (unit variance)
3. $\forall j < j' : \langle y_j, y_{j'} \rangle_t = 0$ (decorrelation)

- Problem leading to solve:

$$\langle \dot{\mathbf{v}} \dot{\mathbf{v}}^T \rangle_t S_j = \lambda_j S_j \tag{3}$$

Slow Feature Analysis

Formulation

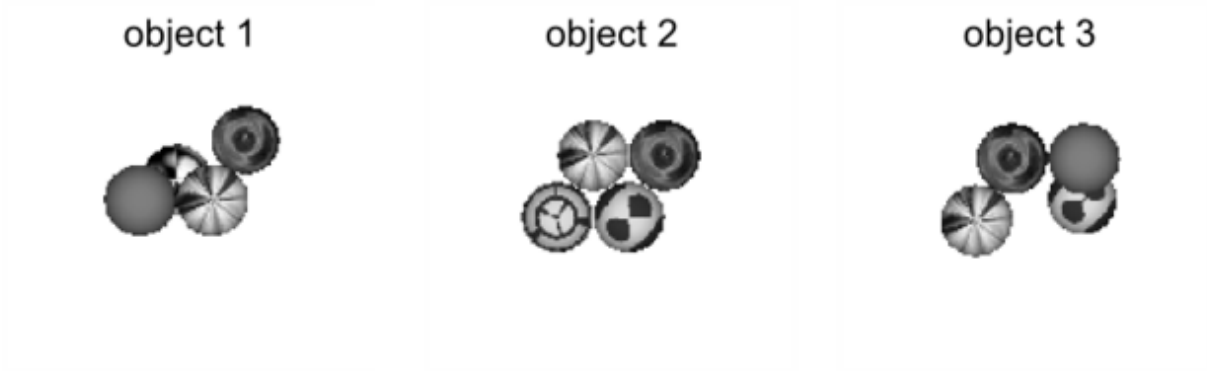


- Problem leading to solve: $\langle \dot{\mathbf{v}}\dot{\mathbf{v}}^T \rangle_t S_j = \lambda_j S_j$
- Diagonalizing $\dot{\mathbf{v}}\dot{\mathbf{v}}^T$
- Keep the M eigenvectors associated to the **smallest eigenvalues**

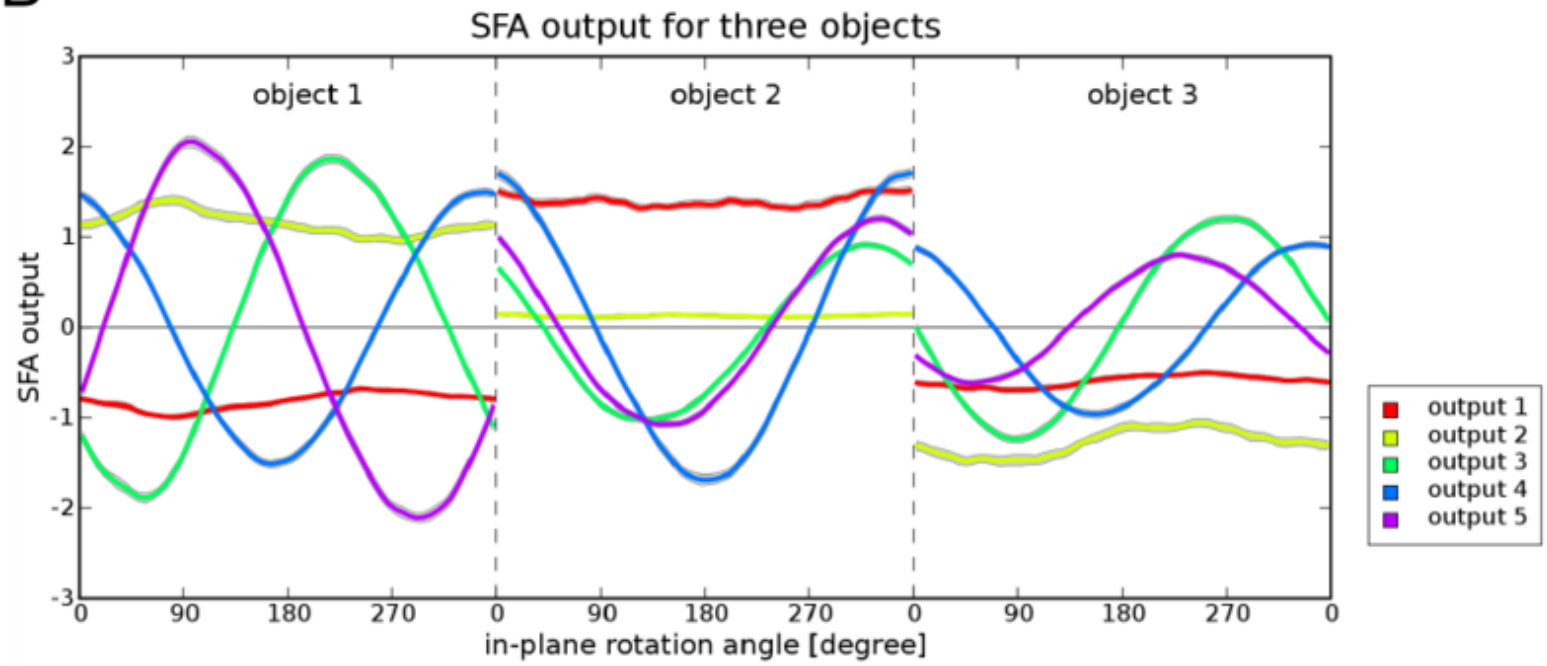
Slow Feature Analysis

Application: learning invariances

A

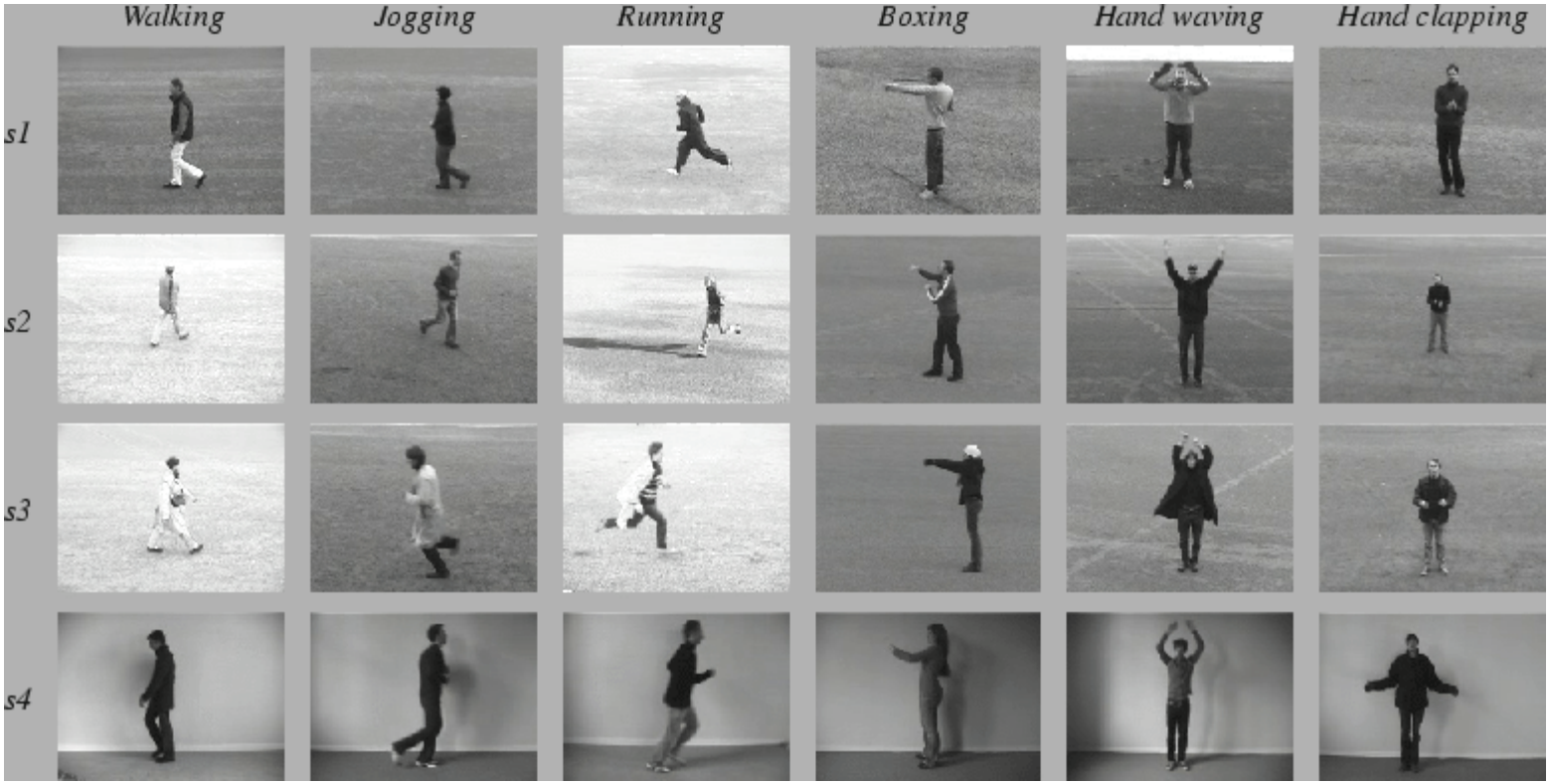


B



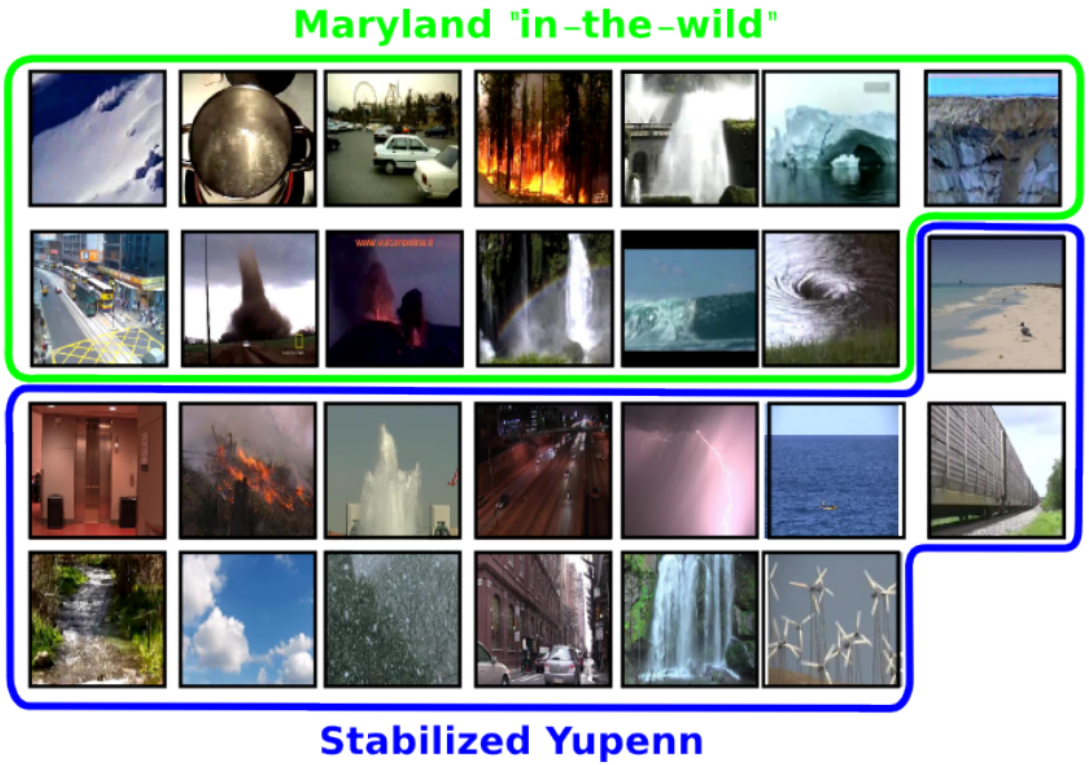
Slow Feature Analysis

Application: human action recognition



Slow Feature Analysis

Application: dynamic scene classification



- Input $v(t)$: image patch which
- SFA learns motion descriptors

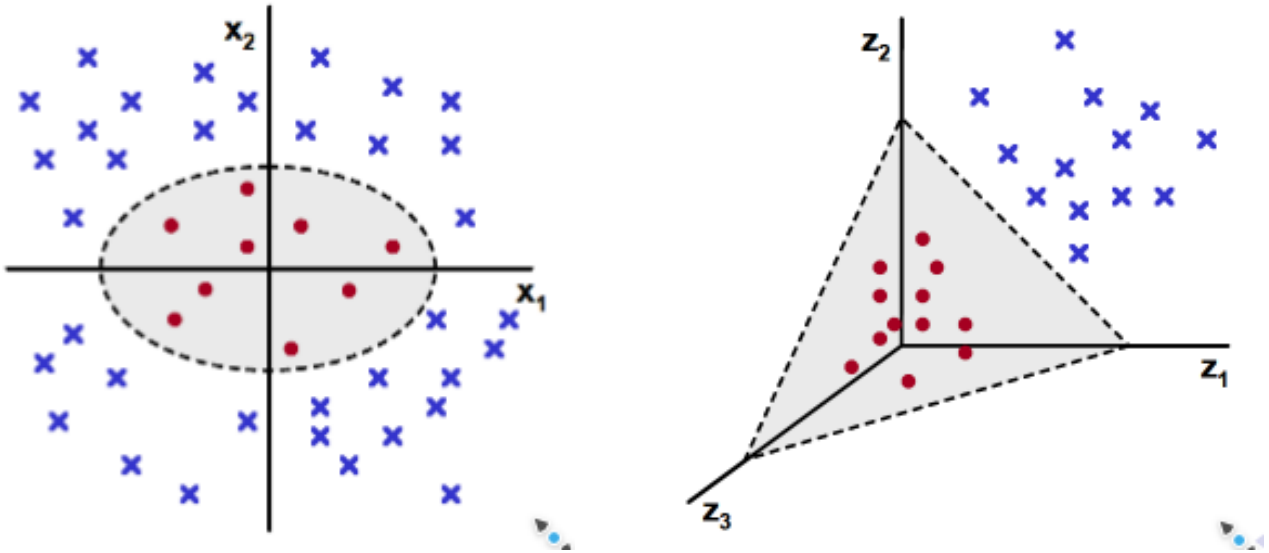
Non-Linear Manifold Learning

- Limit of linear methods: "good" data representations often require non-linear mapping
  1. Kernelization
  2. Visualization methods

Non-Linear Manifold Learning

Kernelization

- One option: define a non-linear injection function ϕ. Ex : $\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$



- Apply linear methods in the induced space

Kernelization

Kernel Trick

- Injection function $\phi : \mathbb{R}^p \rightarrow \mathbb{R}^m$, often $m > p$
- Apply linear methods in the induced space $\mathbb{R}^m$
- Projection in the induced space: dot product $\langle \phi(x); \phi(y) \rangle$
- Mercer Kernel: if $k(x, y) = \langle \phi(x); \phi(y) \rangle$, then:
  - $\forall (x_1, ..., x_n)$ $k(x_i, x_j)$ is a PSD matrix
  - Reverse: if $k(x, y)$ is PSD, $\exists \phi$ s.t. $k(x, y) = \langle \phi(x); \phi(y) \rangle$
- Mercer Kernel examples:
  - Linear : $k(x, y) = \langle x; y \rangle$
  - Polynomial : $k(x, y) = (1 + \langle x; y \rangle)^d$
  - RBF : Gaussian $k(x, y) = e^{\frac{-||x-y||^2}{\sigma^2}}$, Laplace $k(x, y) = 1/2e^{-\gamma|x-y|}$

Kernelization

Dual Formulation

- For many linear methods: learned vector $v_k = \sum_{i=1}^{N} \alpha_i \phi(x_i)$
- For $x \in \mathbb{R}^m$: $\langle \phi(x); v_k \rangle = \sum_{i=1}^{N} \alpha_i \langle \phi(x); \phi(x_i) \rangle = \sum_{i=1}^{N} \alpha_i k(x, x_i)$
- **Convenience:** Computing projection in induced space only using vectors in the input space $\mathbb{R}^n$
  - Especially useful for infinite induced spaces (*e.g.* Gaussian kernels)
- Optimization problem: rewritten wrt α parameters (see Kernel PCA)

Kernel PCA (KPCA)

Formulation

- PCA, recall: $\Sigma v = \lambda v$, $\lambda \in \mathbb{R}$, avec $\Sigma = \frac{1}{n} \sum_{i=1}^{n} (X_i - g)(X_i - g)^T$
- Non-linear mapping with ϕ: X_i and $v \in$ induced space
- Assuming centered data, $v = \sum_i \alpha_i \phi(X_i)$ since
  $v = \frac{1}{n\lambda} \sum_i \phi(X_i)\phi(X_i)^T v = \frac{1}{n\lambda} \sum_i \langle \phi(X_i); v \rangle \phi(X_i)$
- Substituting, KPCA formulation becomes:

$$K\alpha_k = n\lambda_k \alpha_k \tag{4}$$

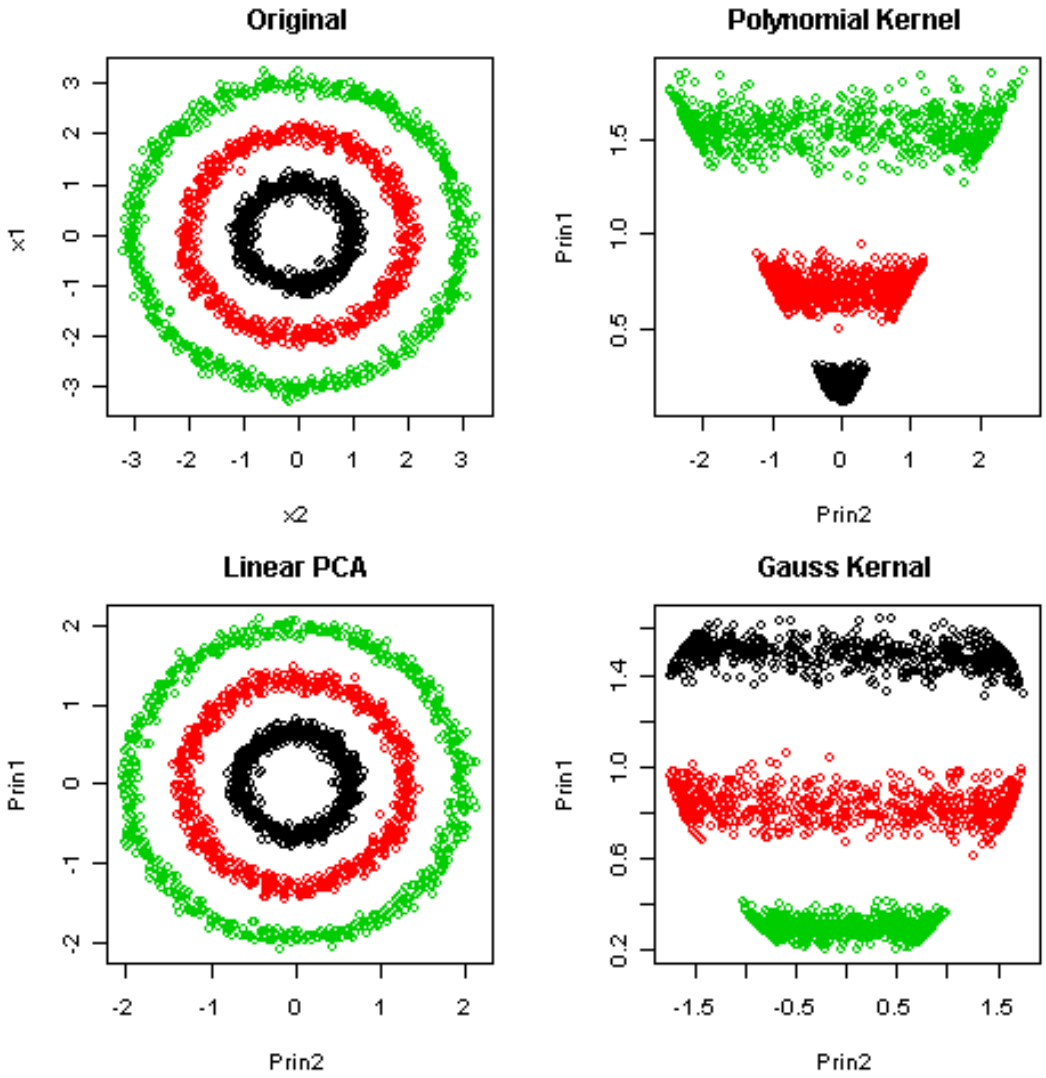
KPCA

Algorithm

- Diagonalizing Gram matrix $K = \{k(x_i, x_j)\}$, $(i, j) \in n \times n$
  - Each α_k vector defines v_k
- Normalization condition:
  $\langle v_k; v_k \rangle = 1 \Rightarrow \alpha_k^T K \alpha_k = 1 \Rightarrow \lambda_k n \alpha_k^T \alpha_k = 1$
  - Each α_k vector must be of norms $\frac{1}{\sqrt{\lambda_k n}}$
- Projecting new point $\phi(x) \in \mathbb{R}^m$ on v_k : $\sum_{i=1}^{n} \alpha_{k,i} k(x, x_i)$

KPCA

Illustration



Non-Linear Manifold Learning

MDS (Multi-Dimensional Scaling)

- Input: matrix **D** of euclidean distances between N points
- Distance **D** $\sim$ similarity $\mathbf{B} = \mathbf{X}^t\mathbf{X}$
  - **X** size dxN, **D** and **B** size NxN
  - $d_{ij}^2 = d_{ii}^2 + d_{jj}^2 - 2bij$
- Goal: find points **y** in low dimensional space, which reflects these pairwise distances:

$$\phi(\mathbf{Y}) = \sum_{i \neq j=1,..,N} (d_{ij}^2 - \|\mathbf{y}_i - \mathbf{y}_j\|)^2 \quad (5)$$

- Result: Minimizing Eq. 5 obtained by finding eigen decomposition of gram matrix $\mathbf{B} = X^tX$

MDS (Multi-Dimensional Scaling)

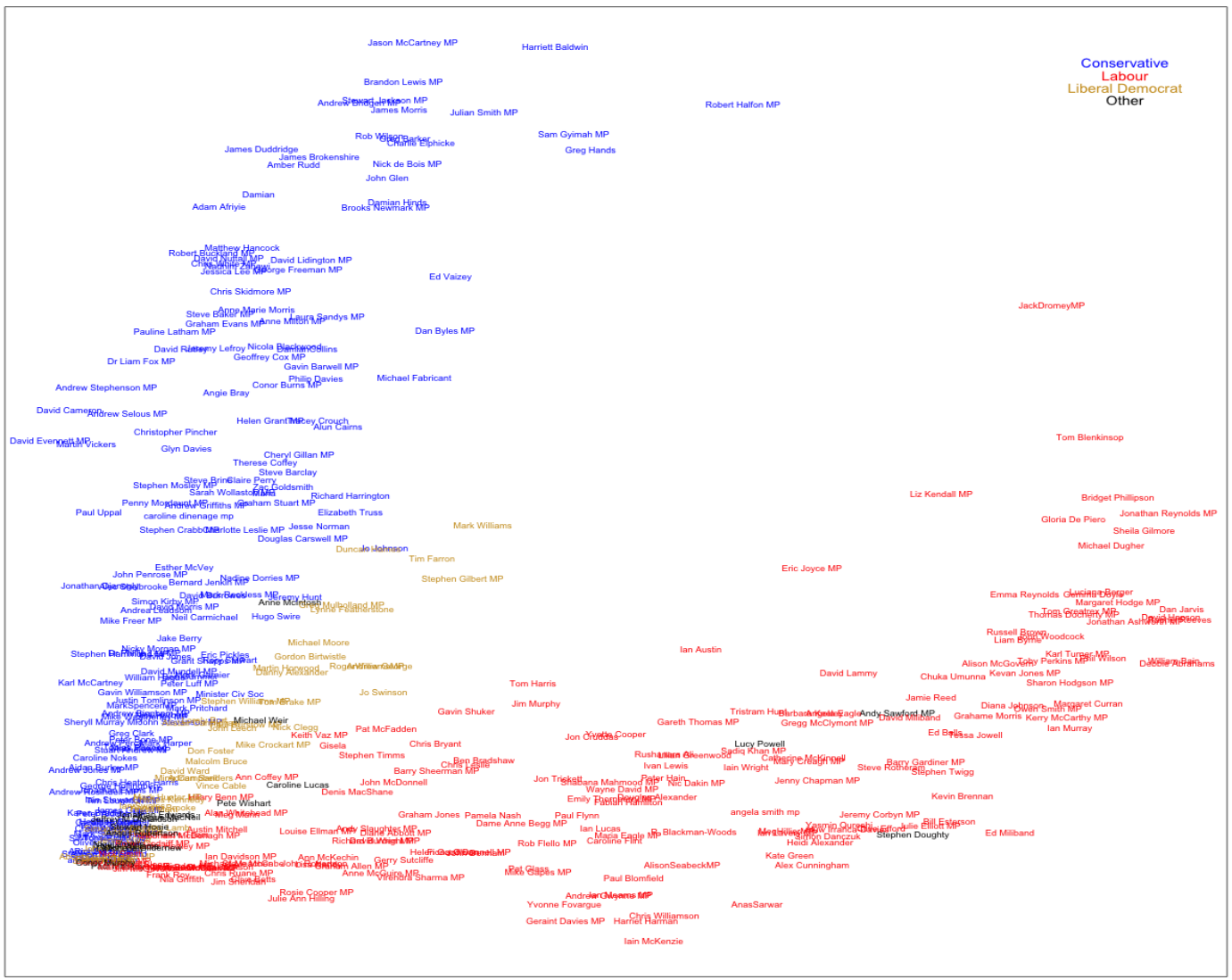
classical Multidimensional Scaling

input: $D \in \mathbb{R}^{n \times n} (D_{ii} = 0, \ D_{ij} \geq 0), \ d \in \{1, \ldots, n\}$

1. Set $B := -\frac{1}{2} HDH$, where $H = I - \frac{1}{n} \mathbf{1}\mathbf{1}^\top$ is the centering matrix.
2. Compute the spectral decomposition of B: $B = U \Lambda U^\top$.
3. Form Λ_+ by setting $[\Lambda_+]_{ij} := \max\{\Lambda_{ij}, 0\}$.
4. Set $X := U \Lambda_+^{1/2}$.
5. Return $[X]_{n \times d}$.

- MDS is actually closely connected to PCA (KPCA and Graam matrix)
- Actually linear dimension reduction from input space

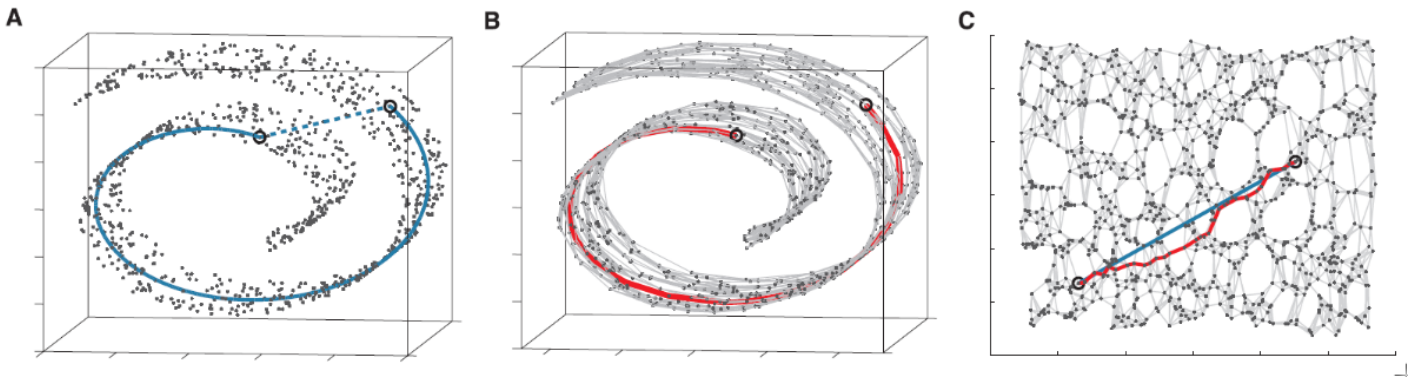
MDS (Multi-Dimensional Scaling)



Non-Linear Manifold Learning

ISOMAP

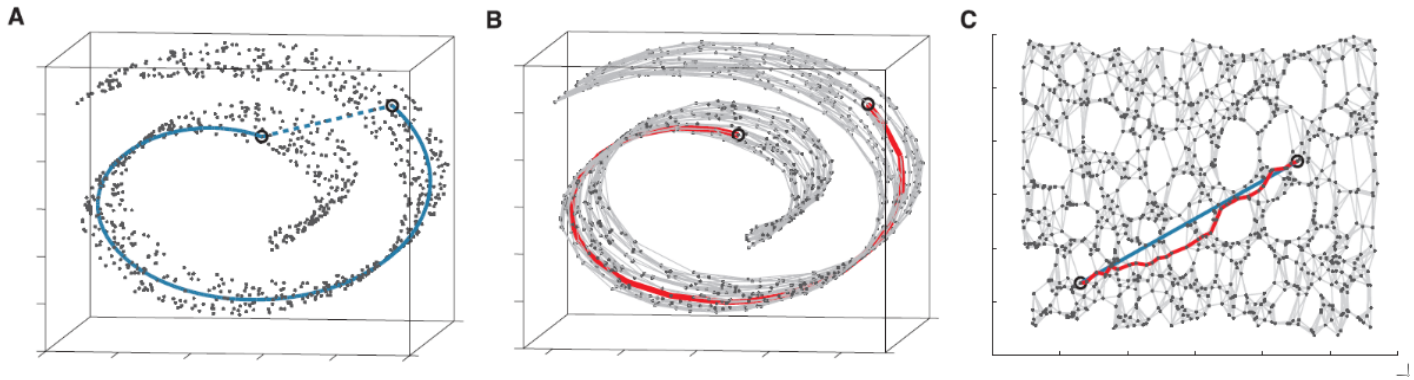
- Problem with MDS: euclidean distance when data lie in non-linear manifold
- ISOMAP solution: apply MDS using geodesic distance in the manifold !



Non-Linear Manifold Learning

ISOMAP

- ISOMAP: apply MDS using geodesic distance on the manifold
- Problem: manifold is unknown to us !
- Assumption: manifold locally smooth, euclidean distance good assumption of the geodesic distance for close points
  1. Generate an adjacency graph between points in input space
  2. Compute shortest path in graph between all pairs of points (Dijkstra's or Floyd's shortest-path) $\Rightarrow$ Approximation of the geodesic distance
  3. Apply MDS on the estimated geodesic distance matrix



Non-Linear Manifold Learning

Stochastic Neighbor Embedding (SNE)

- Observation: large distance in high-dimensional space unreliable (curse of dimensionality)
- Trying to reflect large distance as in MSD / ISOMAP not a good thing
- Idea: Focus on reflecting small distances for close points
  - Motivation shared by Local Linear Embedding (LLE)
- In input space (*e.g.* $\mathbb{R}^d$): $p_{i|j} = \frac{e^{-||x_i - x_j||^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-||x_i - x_k||^2/2\sigma_i^2}}$
- In output space (*e.g.* $\mathbb{R}^2$): $q_{i|j} = \frac{e^{-||y_i - y_j||^2/2\sigma_i^2}}{\sum_{k \neq i} e^{-||y_i - y_k||^2/2\sigma_i^2}}$

Non-Linear Manifold Learning

Stochastic Neighbor Embedding (SNE)

- $P_i = \sum_j p_{j|i}$ conditional probability distribution over all other datapoints given $\mathbf{x}_i$
- Objective function: Kullback-Leiber divergence between P_i and Q_i
- $C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \ log \frac{p_{j|i}}{q_{j|i}}$
- Solved by gradient descent: $\frac{\partial C}{\partial y_i} = 2 \sum_i (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(\mathbf{y}_i - \mathbf{y}_j)$

Non-Linear Manifold Learning

t-SNE

Two improvements from SNE:

1. Symmetric SNE
   - Defining P as the joint probabiliy over all pairs of points
   - Objective function becomes Kullback-Leiber divergence between P and Q:

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

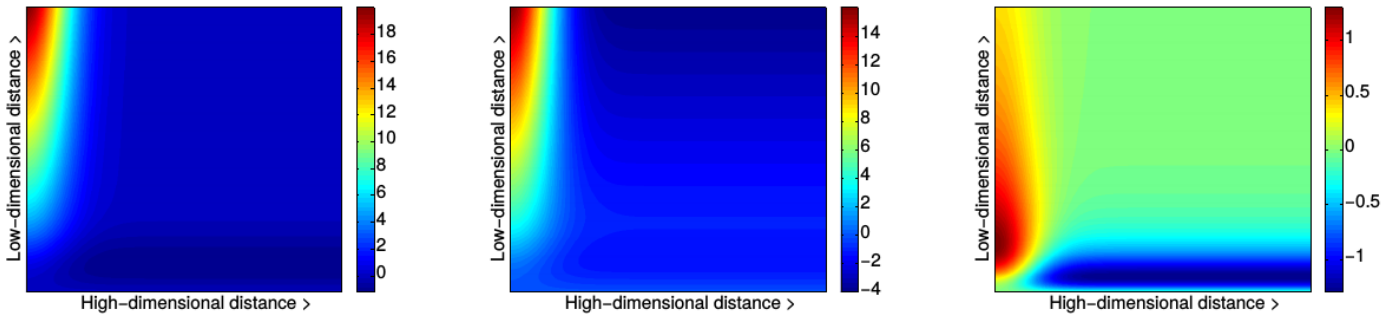
   - $p_{ij} = \frac{e^{-||x_i - x_j||^2/2\sigma_i^2}}{\sum_{k \neq l} e^{-||x_k - x_l||^2/2\sigma^2}}$
   - $q_{ij} = \frac{e^{-||y_i - y_j||^2/2\sigma_i^2}}{\sum_{k \neq l} e^{-||y_k - y_l||^2/2\sigma^2}}$
   - Gradient: $\frac{\partial C}{\partial y_i} = 4 \sum_i (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)$

Non-Linear Manifold Learning

t-SNE

Two improvements from SNE:

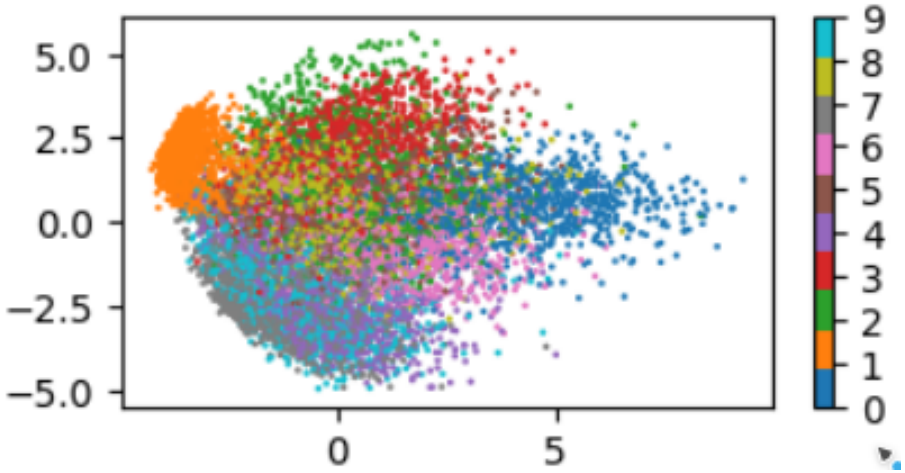
2. Using a Student t-distribution (with 1 dof, *i.e. Cauchy distribution*) for the output space:
   - $q_{ij} = \frac{(1+\|\mathbf{y_i}-\mathbf{y_j}\|^2)^{-1}}{\sum_{k \neq l}(1+\|\mathbf{y_k}-\mathbf{y_l}\|^2)^{-1}}$
   - Motivation:
   - Gradient: $\frac{\partial C}{\partial y_i} = 4 \sum_i (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}$



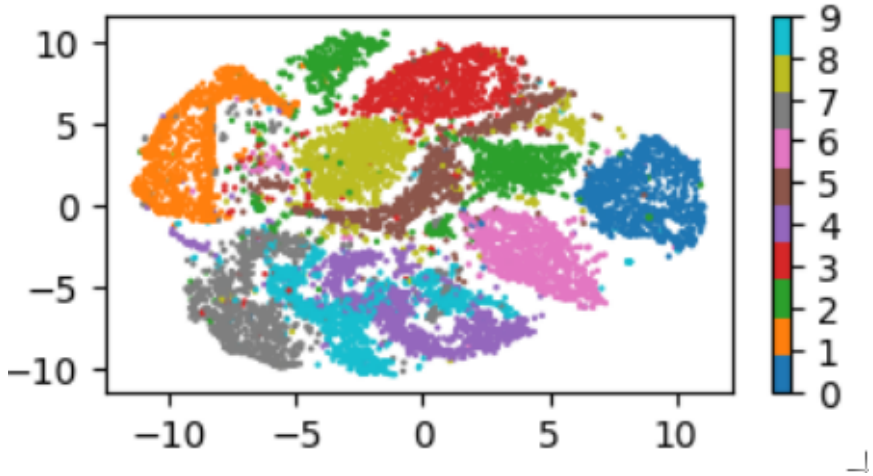
(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

Visualization of Deep Representations: Practical Session

1. Project points of the original dataset: MNIST 784d $\Rightarrow$ 2d
   - Compare PCA and t-SNE



PCA

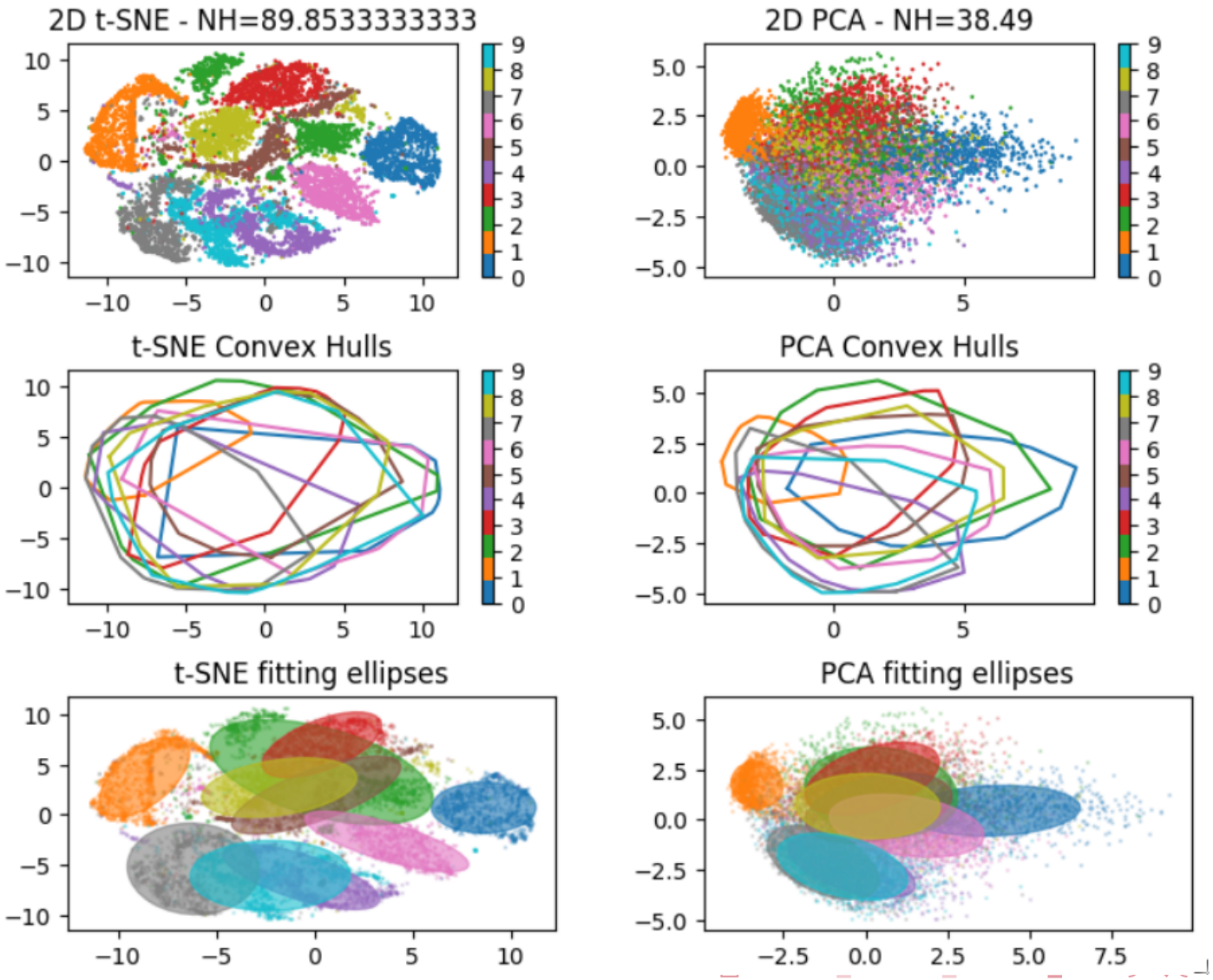


t-SNE

Visualization of Deep Representations: Practical Session

❷ Extract metric to measure class separability ⇔ untangling properties

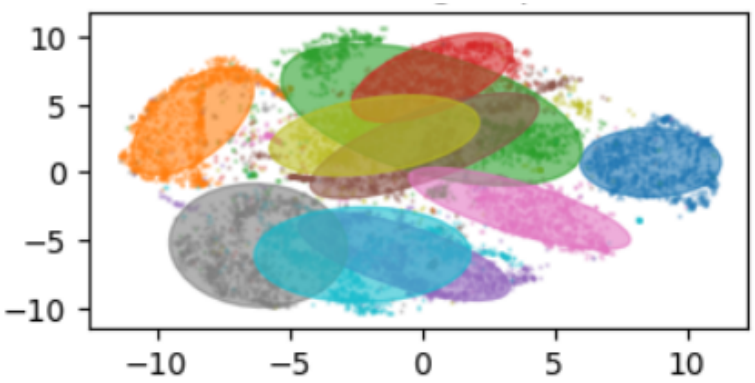
- Convex hull for each class
- Best fitting ellipse (*i.e.* Gaussian distribution) for each class
- "Neighborhood Hit" (NH): find k nearest neighbors (k-nn) for each point, compute k-nn rate of points which belong to the same class



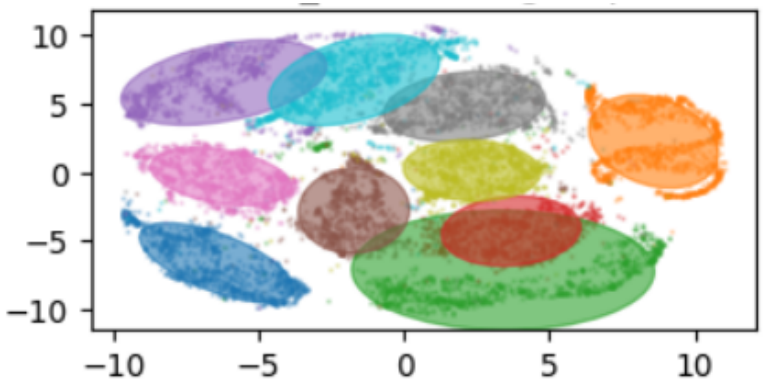
Visualization of Deep Representations: Practical Session

3. Visualize manifold untangling potential of deep neural nets

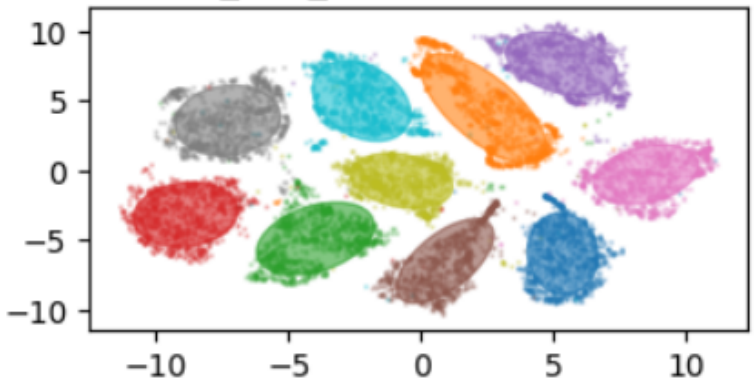
Input



MLP



ConvNet



References I

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, and Gang Wang, *Recent advances in convolutional neural networks*, CoRR abs/1512.07108 (2015).