Technion
Israel Institute of Technology

236862 – Introduction to Sparse and Redundant Representations

Linearized Kernel Dictionary Learning

Alona Golts, Prof. Miki Elad

4.1.18

What We Shall See Today

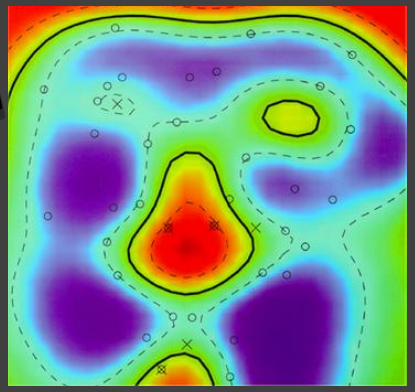
Sparse Representations as a model for signal processing

Wright *et al.* ('09)

This model is successful in machine learning tasks as well

http://cs.stanford.edu/people/karpathy

Kernels are also extremely popular in machine learning



http://alex.smola.org/books.html

Sparse representations and kernels "give birth" to an interesting combination

This new model has its share of growing pains in both:

runtime

space

Our pre-processing called LKDL, preserves the "good", while dealing with the "bad"

1. Sample signals from training set: $\mathbf{X} \rightarrow \mathbf{X}_R$

2. Compute $\mathbf{C} = \mathbf{K}(\mathbf{X}, \mathbf{X}_R)$

3. Compute $\mathbf{W} = \mathbf{K}(\mathbf{X}_R, \mathbf{X}_R)$

4. Approximate $\mathbf{W}$ $\mathbf{W} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$

5. Compute virtual train set $\mathbf{F} = (\boldsymbol{\Lambda}^+)^{1/2}\mathbf{V}^T\mathbf{C}^T$

6. Compute $\mathbf{c}_{test} = \mathbf{K}(\mathbf{x}_{test}, \mathbf{X}_R)$

7. Compute virtual test sample $\mathbf{f}_{test} = (\boldsymbol{\Lambda}^+)^{1/2}\mathbf{V}^T\mathbf{c}_{test}^T$

Classification using DL

Alona Golts

Outline

Background

1) Intro to sparse representations
2) Intro to kernels
3) Kernel dictionary learning

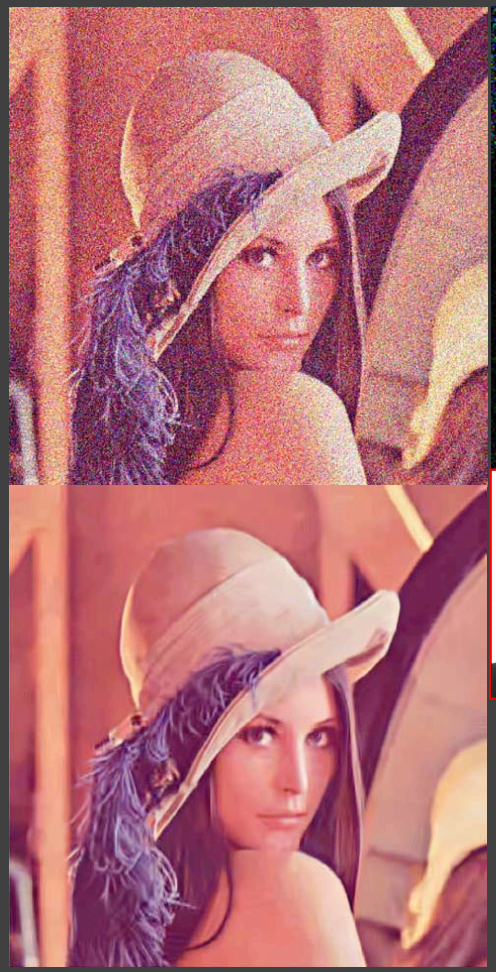
Our work

4) Linearized kernel dictionary learning (LKDL)
5) Results and summary

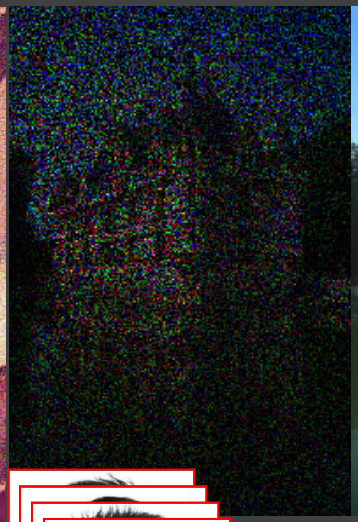
Intro to Sparse Representations

Why Use Sparse Representations?

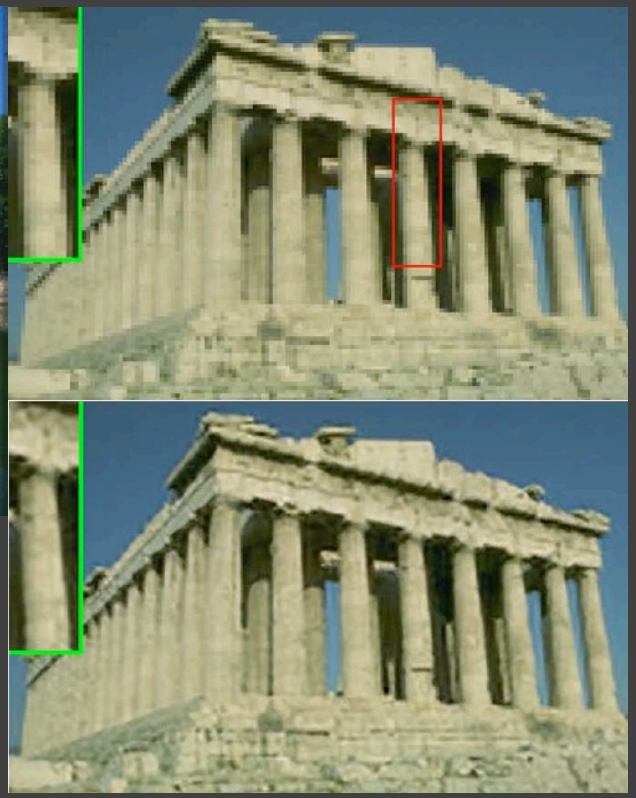
Denoising [1]



Inpainting [2]



Super-Resolution [3]



Compression [4]

[1] Dabov, Foi, Katkovnik and Egiazarian ('07)
[2] Mairal, Elad. and Sapiro ('08)
[3] Yang, Wright, Huang and Ma ('10)
[4] Bryt and Elad ('08)

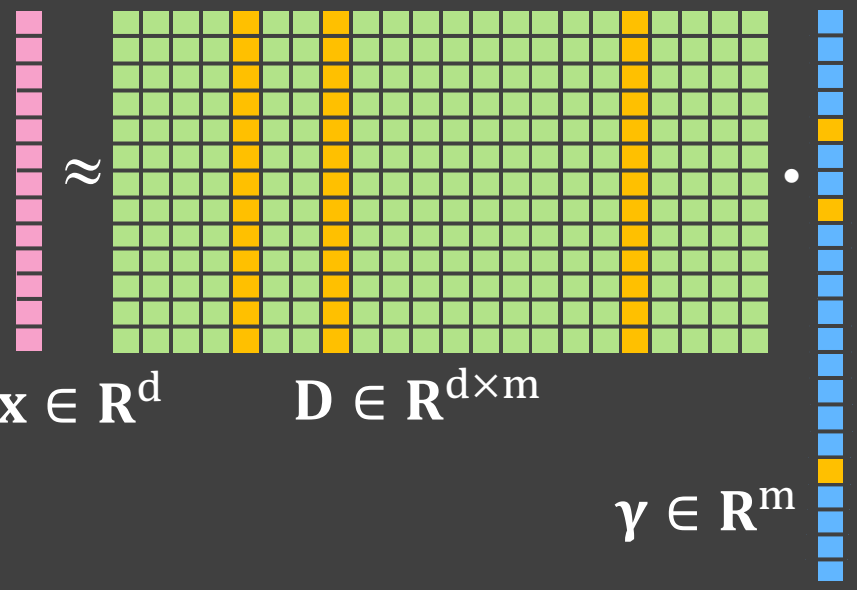
Sparse Coding

- "Sparse coding" – representing a signal with a sparse combination of "dictionary atoms"

$$(*) \quad \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{D}\boldsymbol{\gamma}\|_2^2 \quad \text{s.t. } \|\boldsymbol{\gamma}\|_0 \leq q$$

"cardinality"

signal dictionary sparse vector



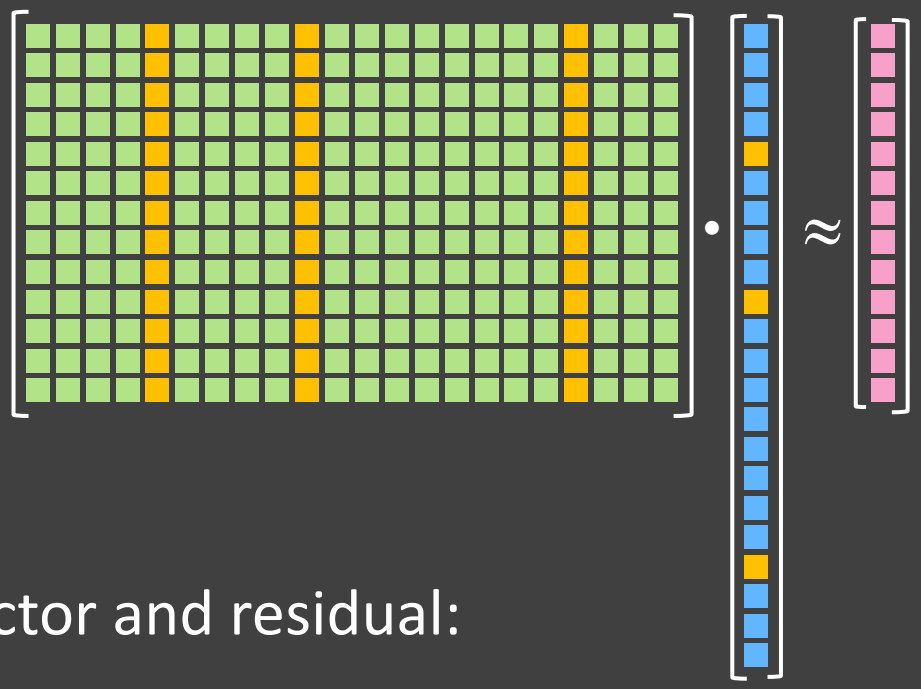
Naïve solution of solving $(*)$:

- scanning $\binom{m}{q}$ options of supports, $\boldsymbol{\gamma}$
- solving least squares
- choosing best reconstruction…

NOT a good idea!

Greedy Approach - OMP

- Step 1: choose atom **that best matches $\mathbf{x}$**
- Next steps: given the previously found atoms, choose next <u>one</u> **that best fits** residual. $j_0 = \operatorname{argmax}\left|\langle \mathbf{r}_{t-1}, \mathbf{d}_j \rangle\right|$



- update coefficients of sparse vector and residual:

$$\boldsymbol{\gamma}_t = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{D}_t \boldsymbol{\gamma}_t\|_2^2, \qquad \mathbf{r}_t = \mathbf{x} - \mathbf{D}_t \boldsymbol{\gamma}_t$$

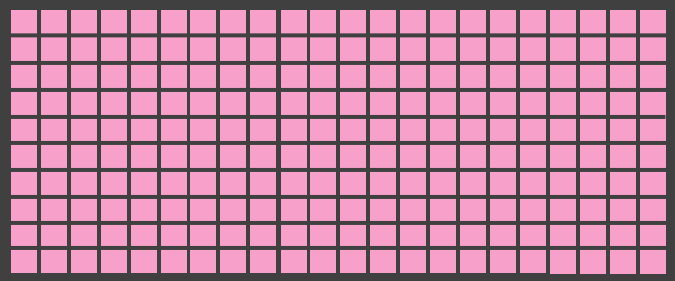
Repeat q times or until target threshold is reached.

Dictionary Learning

- "Dictionary learning" – finding a set of atoms and representations that "best sparsify" a collection of inputs $\mathbf{X}$

$$\underset{\mathbf{D},\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{D}\boldsymbol{\Gamma}\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \qquad \forall i = 1 \ldots N$$

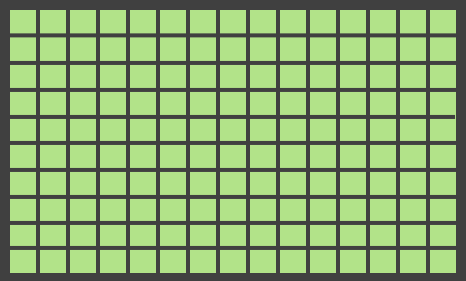
input signal matrix



$\mathbf{X} \in \mathbf{R}^{d \times N}$

$\approx$

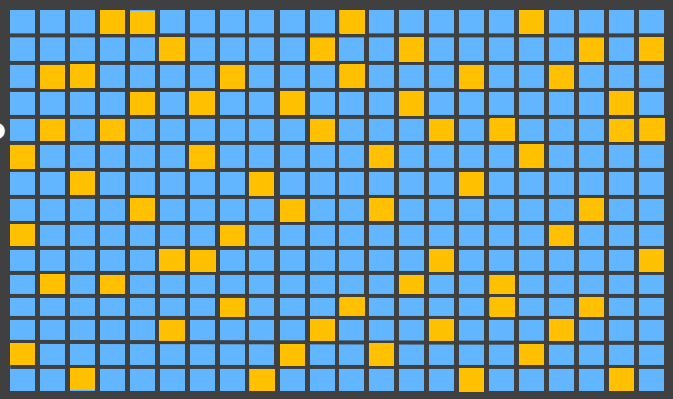
dictionary



$\mathbf{D} \in \mathbf{R}^{d \times m}$

$\bullet$

sparse representation matrix



$\boldsymbol{\Gamma} \in \mathbf{R}^{m \times N}$

Dictionary Learning

$$\underset{\mathbf{D},\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{D}\boldsymbol{\Gamma}\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \qquad \forall i = 1 \ldots N$$

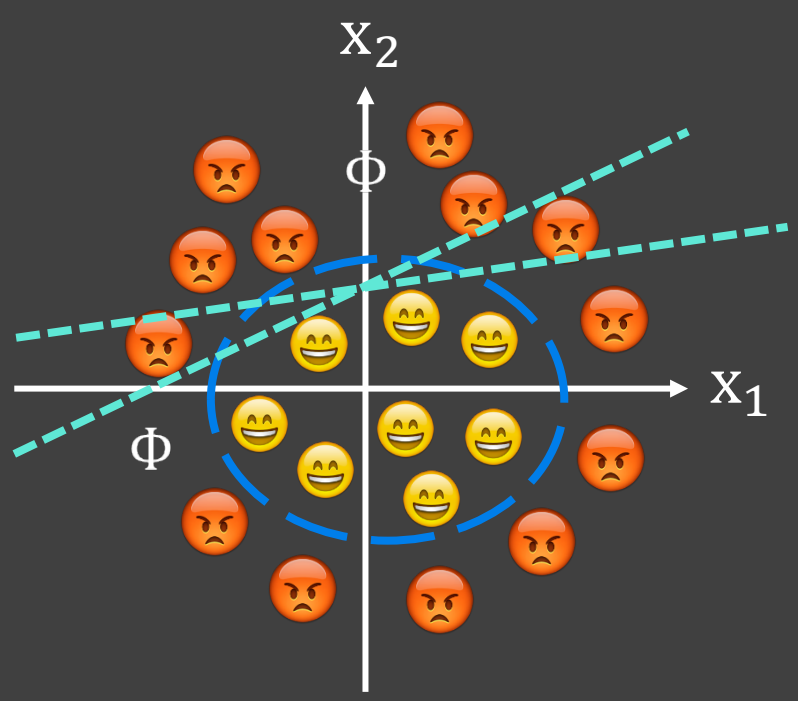
- Basic strategy: block coordinate descent

- Iterate over the following for T iterations:
  - Given $\mathbf{D}$, find sparse representations, $\boldsymbol{\Gamma}$
  - Given $\boldsymbol{\Gamma}$, update dictionary, $\mathbf{D}$
    - MOD [1] – update entire dictionary at once.
    - KSVD [2] – update one atom at a time, along with the coefficients, solving a rank-1 SVD problem.

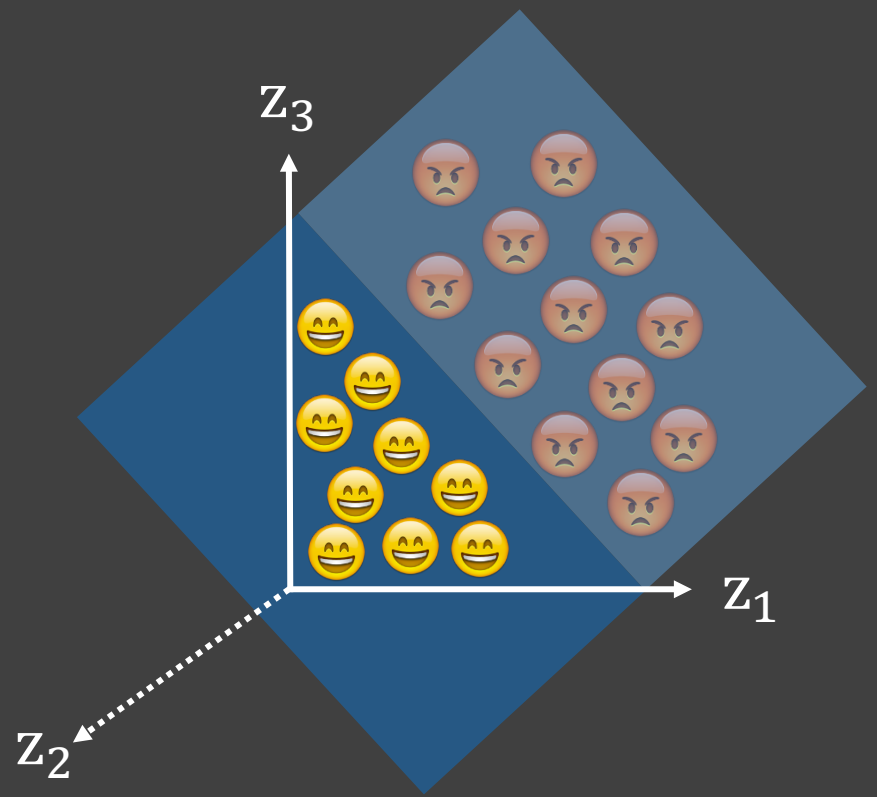
[1] Engan, Aake, Hakon and Husoy, ('99)
[2] Elad and Aharon. ('06)

Intro to Kernels

Classification Problem



$\mathcal{X}$ "input space"



$\mathcal{F}$ "feature space"

$$(x_1, x_2) \xrightarrow{\Phi} (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

Kernel Trick

- For the previous mapping, let us calculate the inner product between two signals in the feature space:

$$\langle \Phi(x), \Phi(y) \rangle = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) \begin{pmatrix} y_1^2 \\ \sqrt{2}y_1y_2 \\ y_2^2 \end{pmatrix} =$$

$$= x_1^2y_1^2 + 2x_1x_2y_1y_2 + x_2^2y_2^2 = (x_1y_1 + x_2y_2)^2 = \langle \mathbf{x}, \mathbf{y} \rangle^2$$

"kernel" $\longrightarrow$ $= \kappa(x, y)$

Positive Definite Kernels

The following two are equivalent:

- $\mathbf{K}$ is positive definite (PD), i.e., for any training points $(\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathcal{X}$ and for arbitrary scalars $(a_1, \dots, a_N) \in \mathbf{R}$, the following holds:

$$\sum_{i,j} a_i a_j \mathbf{K}_{i,j} \geq 0, \qquad \mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

- There exits a map Φ into a dot-product space $\mathcal{H}$ s.t.:

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$$

Types of kernels

Commonly used kernels:

Linear: $\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle + c$

Polynomial: $\kappa(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^b$

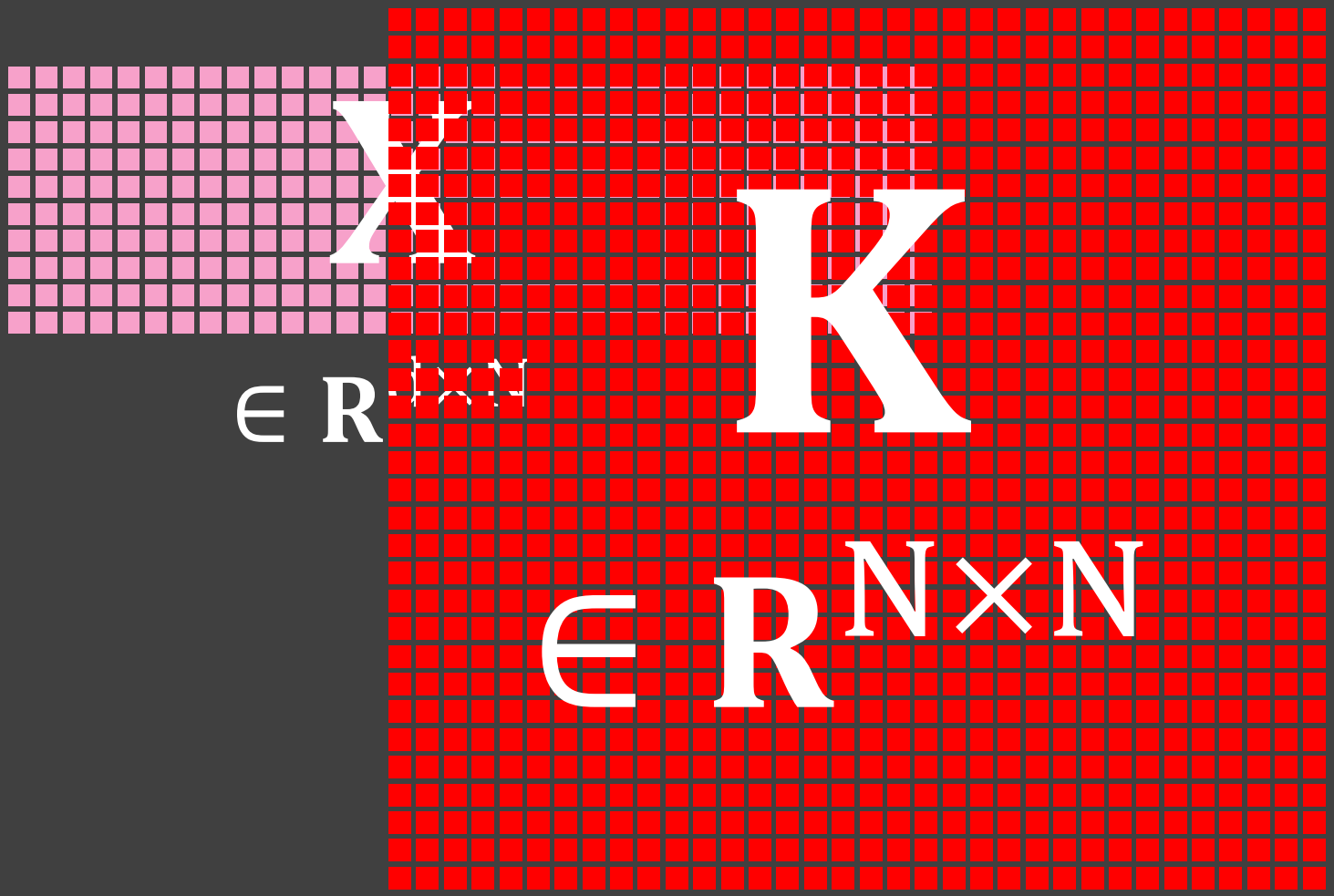
Gaussian/RBF: $\kappa(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$

The kernel matrix consists of inner products of the feature vectors in the high dimensional space.

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N], \qquad \mathbf{K} = \Phi(\mathbf{X})^T \Phi(\mathbf{X}),$$

$$\mathbf{K}_{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

Kernel Matrix



Kernels in Machine Learning

- Kernels provide powerful representational power to linear machine learning algorithms, thus have been used extensively over the past 20 years:
  - SVM
  - Kernel PCA
  - Kernel Regression
  - Kernel K-means
  - Kernel NN
  - ...

Classification using Sparsity

- The sparsity model is also effective in discriminative tasks, as well as generative ones:
  - "Sparse Representation for Signal Classification", Huang *et al.*, ('06)
  - "Robust Face Recognition using Sparse Representations", Wright *et al.*, ('09)
  - "Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification", Yang *et al.*, ('09)
  - "Sparse representation for computer vision and pattern recognition", Wright *et al.*, ('10)
  - "Robust Visual Tracking and Vehicle Classification via Sparse Representation", Mei *et al.*, ('11)
  - "Learning Sparse Representations for Human Action Recognition", Guha *et al.*, ('12)
  - "Learning Structured Low-rank Representations for Image Classification", Zhang *et al.*, ('13)
  - "Multiview Hessian Discriminative Sparse Coding for Image Annotation", Liu *et al.*, ('14)
  - "Learning Discriminative Sparse Representations for Hyperspectral Image Classification", Du *et al.*, ('15)
- Why then not "kernelize" classic sparse representation algorithms?

Kernel Sparse Representations

- In the past 5 years there has been a multitude of work concentrated on kernel sparse representations.
- Some examples:
  - Vincent & Bengio, ('02)
  - Gao, Tsang & Chia, ('10)
  - Zhang, Zho, Chang, Liu, Wang & Li, ('12)
  - Nguyen, Patel, Nasarabadi & Chellappa, ('12)
- We choose to concentrate on kernel dictionary learning to highlight the benefit of our approach.

Kernel Dictionary Learning

Nguyen, Patel, Nasrabadi and Chellappa ('12)

Kernel Dictionary Learning

- Perform linear dictionary learning in feature space:

$$\mathbf{X} \rightarrow \Phi(\mathbf{X}), \mathbf{D} \rightarrow \Phi(\mathbf{D})$$

$$\underset{\Phi(\mathbf{D}),\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\boldsymbol{\Phi}(\mathbf{X}) - \boldsymbol{\Phi}(\mathbf{D})\boldsymbol{\Gamma}\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \qquad \forall i = 1 \ldots N$$

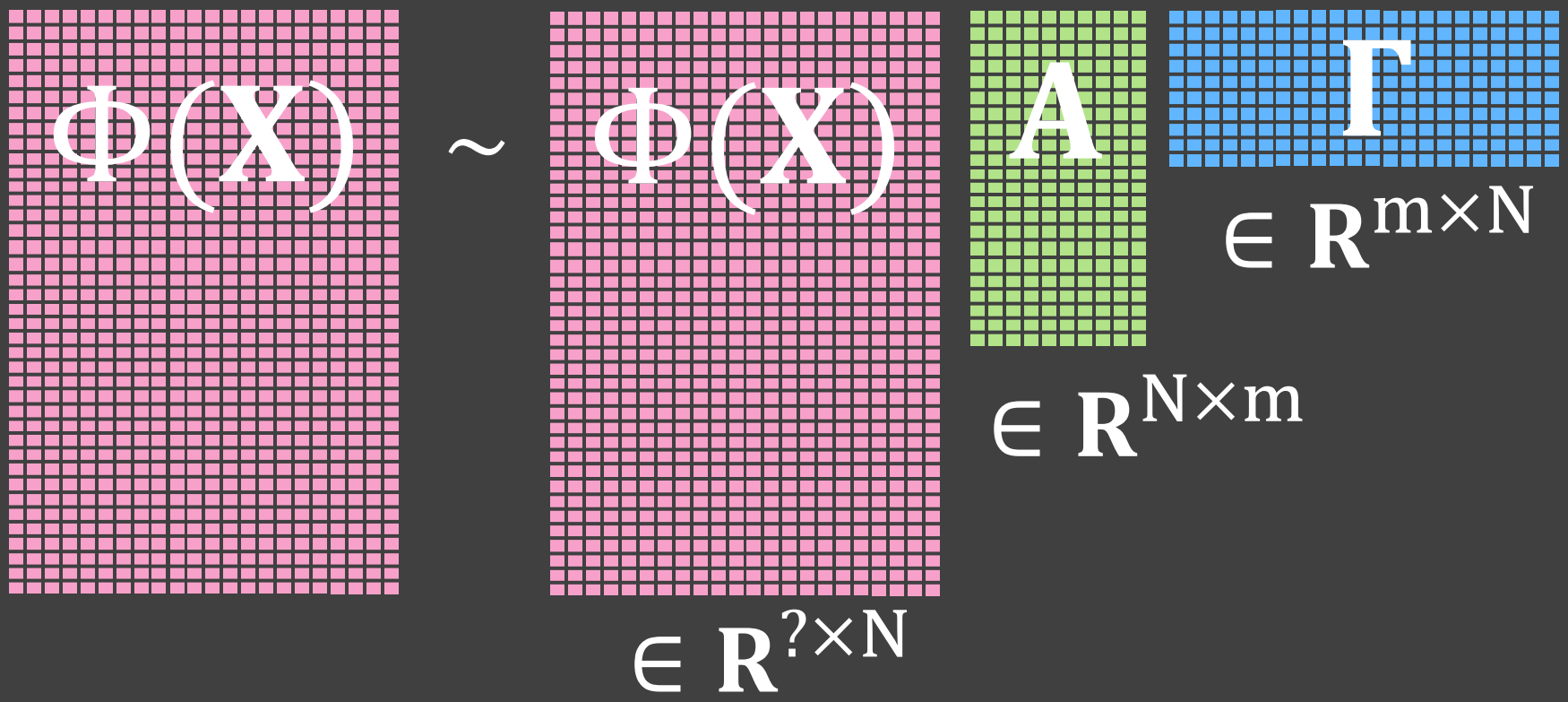
$$(*) \;\; \Phi(\mathbf{D}) = \Phi(\mathbf{X})\mathbf{A}, \qquad \mathbf{A} \in \mathbf{R}^{N \times m}$$

$$\underset{\mathbf{A},\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\boldsymbol{\Phi}(\mathbf{X}) - \boldsymbol{\Phi}(\mathbf{X})\mathbf{A}\boldsymbol{\Gamma}\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \qquad \forall i = 1 \ldots N$$

(*) “Representer theorem” - Kimeldorf and Wahba (‘71)
(*) “Double Sparsity” - Rubinstein, Zibulevsky and Elad (‘10)

Kernel Dictionary Learning

$$\underset{\mathbf{A},\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\boldsymbol{\Phi}(\mathbf{X}) - \boldsymbol{\Phi}(\mathbf{X})\mathbf{A}\boldsymbol{\Gamma}\|_F^2 \quad \text{s.t.} \quad \|\boldsymbol{\gamma}_i\|_0 \leq q, \qquad \forall i = 1 \ldots N$$



“Kernelization” of OMP

AS: Choose atom that best matches residual:

Classic:

$$j_0 = \operatorname{argmax}\left|\left\langle \overbrace{\mathbf{x} - \mathbf{D}_{t-1}\boldsymbol{\gamma}_{t-1}}^{\mathbf{r}_{t-1}}, \mathbf{d}_j \right\rangle\right|$$

Kernel:

$$j_0 = \operatorname{argmax}\left|\left\langle \Phi(\mathbf{x}) - \Phi(\mathbf{X})\mathbf{A}_{t-1}\boldsymbol{\gamma}_{t-1}, \Phi(\mathbf{X})\mathbf{a}_j \right\rangle\right|$$

$$= \left|\mathbf{K}(\mathbf{x}, \mathbf{X})\mathbf{a}_j - \boldsymbol{\gamma}_{t-1}^T \mathbf{A}_{t-1}^T \mathbf{K}(\mathbf{X}, \mathbf{X})\mathbf{a}_j\right|$$

Input signal

Train set

$\in \mathbf{R}^{1\times N}$

$\in \mathbf{R}^{N\times N}$

“Kernelization” of OMP

LS: Update sparse vector using least squares:

Classic:

$$\boldsymbol{\gamma}_t = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{D}_t\boldsymbol{\gamma}\|_2 = \left(\mathbf{D}_t^T\mathbf{D}_t\right)^{-1}\mathbf{D}_t^T\mathbf{x}$$

Kernel:

$$\boldsymbol{\gamma}_t = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}} \|\boldsymbol{\Phi}(\mathbf{x}) - \Phi(\mathbf{X})\mathbf{A}_t\boldsymbol{\gamma}\|_2 = [\Phi(\mathbf{X})\mathbf{A}_t]^+\boldsymbol{\Phi}(\mathbf{x})$$

$$= \left[\mathbf{A}_t^T\mathbf{K}(\mathbf{X},\mathbf{X})\mathbf{A}_t\right]^{-1}\mathbf{A}_t^T\mathbf{K}(\mathbf{X},\mathbf{x})$$

“Kernelization” of MOD

- Once the sparse representation $\mathbf{\Gamma}$ is known, update $\mathbf{A}$:

$$\underset{\mathbf{A}}{\operatorname{argmin}} \|\Phi(\mathbf{X}) - \Phi(\mathbf{X})\mathbf{A}\mathbf{\Gamma}\|_{\mathrm{F}}^2$$

- Update for Kernel MOD: $\mathbf{A} = \mathbf{\Gamma}^{+} = \mathbf{\Gamma}^{\mathrm{T}}\left(\mathbf{\Gamma}\mathbf{\Gamma}^{\mathrm{T}}\right)^{-1}$

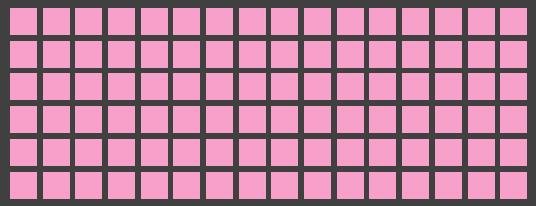
- KSVD can be updated too using kernels only

More

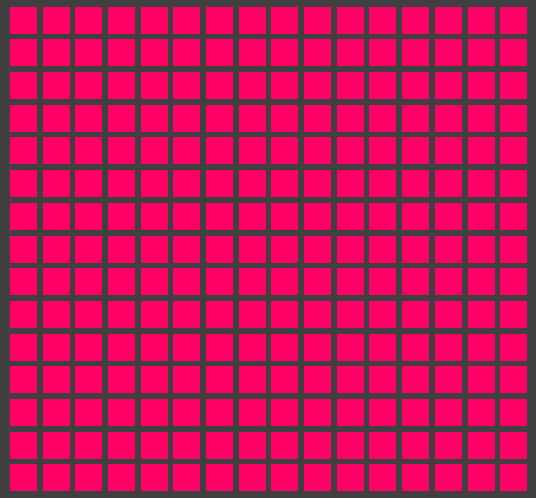
Problems with KDL

Memory:

$\mathbf{X} \in \mathbf{R}^{d \times N}$



$\mathbf{K} \in \mathbf{R}^{N \times N}$



Runtime:

Step:	Complexity:
OMP- Atom Selection	$O(dq + d)$
KOMP- Atom Selection	$O(N^2 + Nq + N)$
OMP- Least Squares	$O(dq^2 + dq + q^3)$
KOMP – Least Squares	$O(N^2 q + Nq + q^3)$

N – number of signals
q – target cardinality
d – signal dimension

$N \gg d \gg q$

KDL: Pros and Cons

The Good

- Introduces nonlinearity to sparse representation algorithms.
- Fairly easy to substitute dot products with kernels.
- Flexibility with choice of kernel.

The Bad

- High dependence on a possibly huge kernel matrix.
- Complexity of algorithms depends on number of signals instead of their dimension.
- A specific “tailoring” of the kernel is needed in each individual algorithm.
- Algorithm cannot always be written using dot products.

Our Work:

Linearized Kernel Dictionary Learning

Our Objective

Incorporate nonlinearity into dictionary learning by kernelizing

Faster runtime, less memory

Turning any DL into kernel DL in an easy way

Kernel Matrix Decomposition

Any PD kernel matrix can be decomposed into:

$$\mathbf{K} = \Phi(\mathbf{X})^{\mathrm{T}}\Phi(\mathbf{X}) = \mathbf{F}^{\mathrm{T}}\mathbf{F}$$

Original Samples

$\mathbf{X} \in \mathbf{R}^{\mathrm{d}\times\mathrm{N}}$

"Virtual Samples"

$\mathbf{F} \in \mathbf{R}^{\mathrm{N}\times\mathrm{N}}$

Zhang, Lan, Wang and Moerchen, ('12)

Linearized Kernel DL (LKDL)

Decompose kernel matrix to inner product of "virtual samples"

$$\mathbf{K} = \mathbf{F}^{\mathrm{T}}\mathbf{F}$$

Perform classical (linear) DL on virtual samples

$$\underset{\mathbf{D},\boldsymbol{\Gamma}}{\operatorname{argmin}} \|\mathbf{F} - \mathbf{D}\boldsymbol{\Gamma}\|_{\mathrm{F}}^2$$

Produce classification result

How to Decompose K?

- Eigen decomposition:

$$\mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^{\mathrm{T}} = \mathbf{K} = \mathbf{F}^{\mathrm{T}}\mathbf{F}$$

$$\rightarrow \mathbf{F} = \boldsymbol{\Sigma}^{1/2}\mathbf{U}^{\mathrm{T}}$$

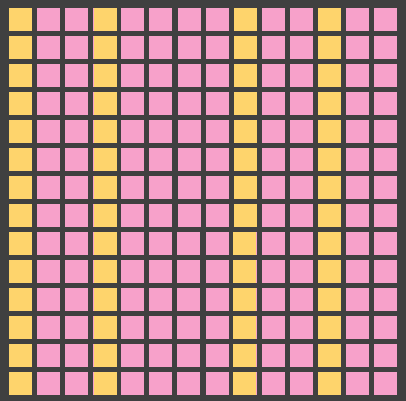
- Not practical for large kernel matrices:

$$\mathbf{K} \in \mathbf{R}^{\mathrm{N}\times\mathrm{N}}$$

computational cost: $O(\mathrm{N}^3)/O(\mathrm{N}^2\mathrm{k})$

Nyström Method

- Find an approximation of the PD matrix: $\tilde{\mathbf{K}} \approx \mathbf{K}$

$\mathbf{K} =$  $\mathbf{C} =$

$\in \mathbf{R}^{N \times N}$ $\in \mathbf{R}^{N \times c}$

Sampling

c columns from $\mathbf{K} \rightarrow \mathbf{C}$

$$\mathbf{K} = \begin{bmatrix} \mathbf{W} & \mathbf{S}^T \\ \mathbf{S} & \mathbf{B} \end{bmatrix}$$

$$= \mathbf{C} \quad c \ll N$$

Nyström Method

$\mathbf{C} =$ $\in \mathbf{R}^{N \times c}$

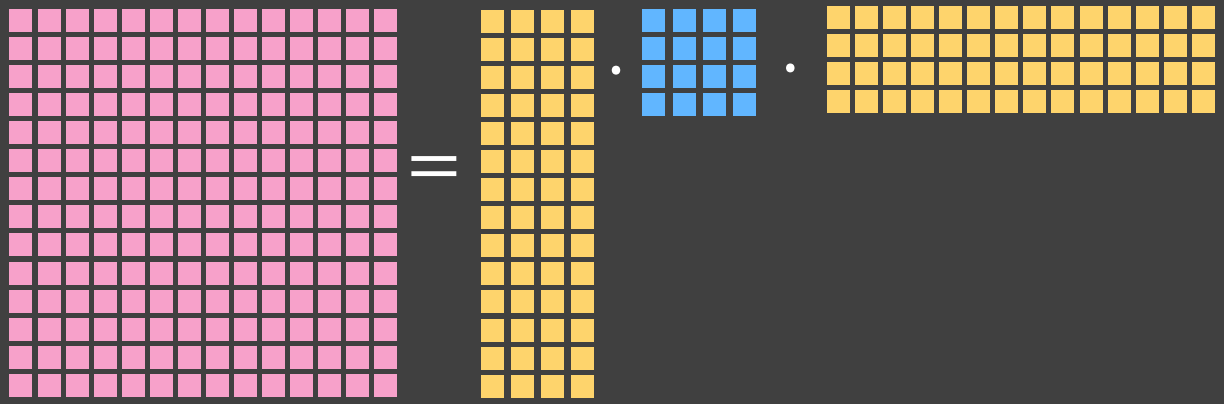
$\mathbf{W} =$ $\in \mathbf{R}^{c \times c}$

Approximation

for $k \leq c$

$$\widetilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^{+}\mathbf{C}^{T}$$

$\widetilde{\mathbf{K}}$ $\mathbf{C}$ $\mathbf{W}^{+}$ $\mathbf{C}^{T}$



Virtual Sample Computation

Nyström:

$$\widetilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^{+}\mathbf{C}^{\mathrm{T}}$$

eigen-decomposition:

$$\mathbf{W} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$$

$$\mathbf{W}^{+} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$$

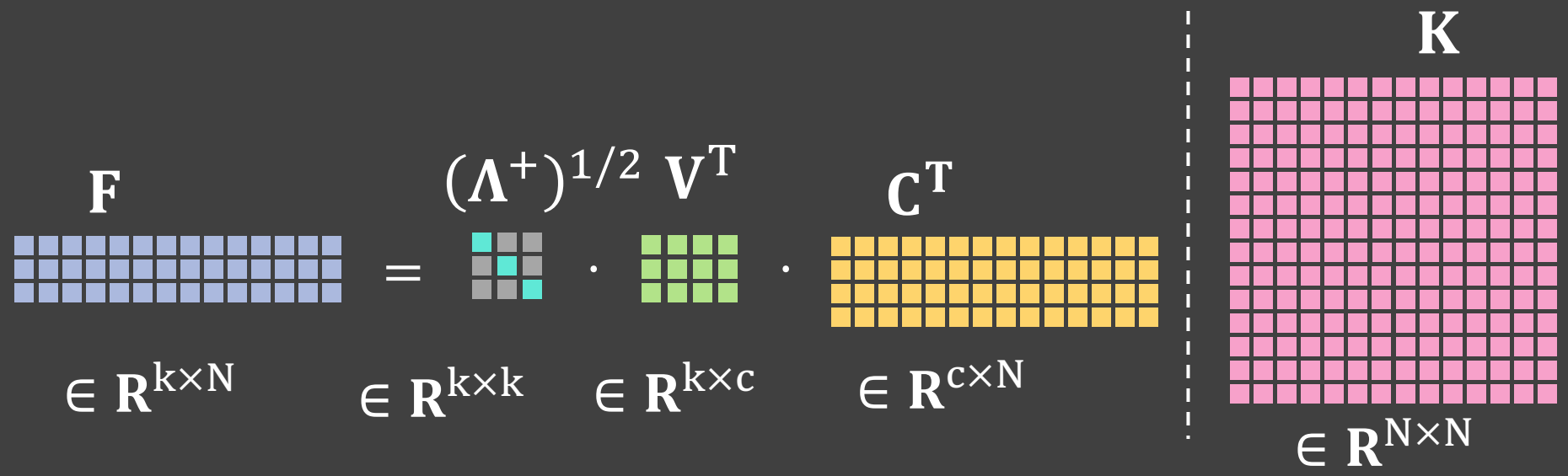
$$\mathbf{W}^{+} = \mathbf{V} \cdot \boldsymbol{\Lambda}^{+} \cdot \mathbf{V}^{\mathrm{T}}$$

$\in \mathbf{R}^{\mathbf{c}\times\mathbf{c}}$ $\quad \in \mathbf{R}^{\mathbf{c}\times\mathrm{k}}$ $\quad \in \mathbf{R}^{\mathrm{k}\times\mathrm{k}}$ $\quad \in \mathbf{R}^{\mathrm{k}\times\mathrm{c}}$

c – number of sampled columns in Nyström

k – degree of eigen-decomposition

Virtual Sample Computation



"virtual sample" computation:

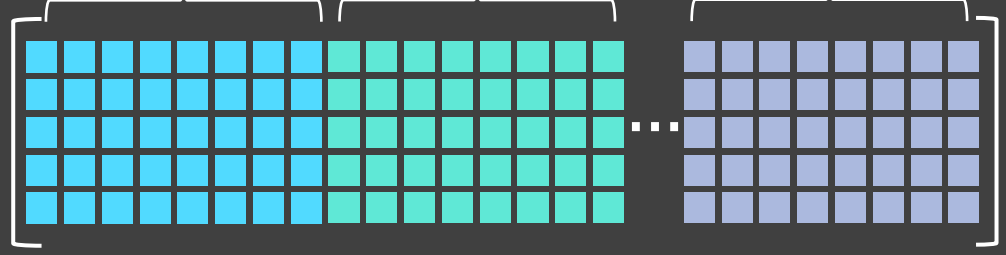
$$\widetilde{\mathbf{K}} = \mathbf{F}^{\mathrm{T}}\mathbf{F} \rightarrow \mathbf{F} = (\mathbf{\Lambda}^{+})^{1/2}\mathbf{V}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}$$

Classification using DL

Train L dictionaries, one for each class:

$$\underset{\mathbf{D}_i,\mathbf{\Gamma}_i}{\operatorname{argmin}} \|\mathbf{X}_i - \mathbf{D}_i\mathbf{\Gamma}_i\|_F^2$$

$$\mathbf{X} = \left[\overbrace{\quad}^{\mathbf{X}_1}\ \overbrace{\quad}^{\mathbf{X}_2}\ \cdots\ \overbrace{\quad}^{\mathbf{X}_L}\right]$$



KSVD

$\mathbf{D}_1$ $\mathbf{D}_2$ … $\mathbf{D}_L$

Classification using DL

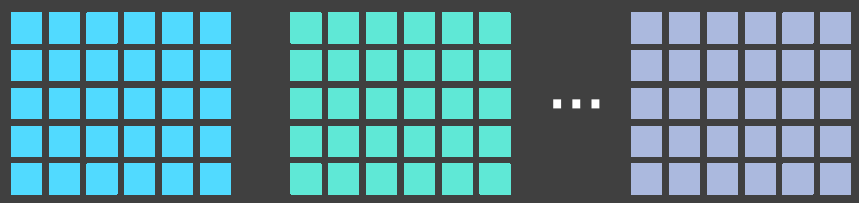
Sparse code each test sample over L dictionaries:

$$\underset{\boldsymbol{\gamma}_i}{\operatorname{argmin}} \|\mathbf{x}_{test} - \mathbf{D}_i \boldsymbol{\gamma}_i\|_2^2$$

$$\text{s.t. } \|\boldsymbol{\gamma}_i\|_0 \leq q\,,$$

$$\forall i = 1 \ldots L$$

$\mathbf{x}_{test}$

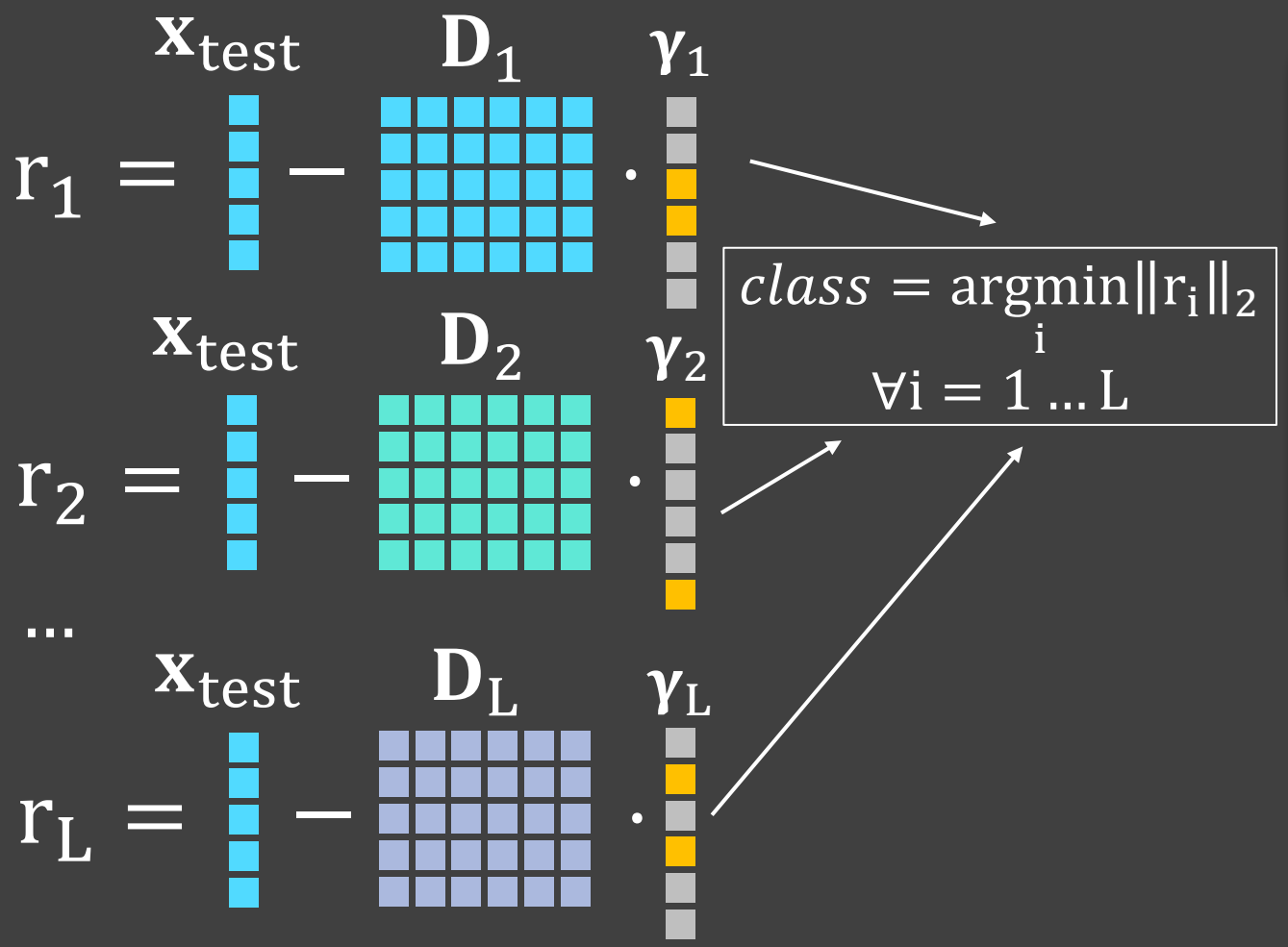


$\mathbf{D}_1$ $\mathbf{D}_2$ … $\mathbf{D}_L$

OMP

$\boldsymbol{\gamma}_1$ $\boldsymbol{\gamma}_2$ … $\boldsymbol{\gamma}_L$

Classification using DL



Chosen class is the one with minimal representation error:

$$\text{class} = \underset{i}{\operatorname{argmin}} \|r_i\|_2$$

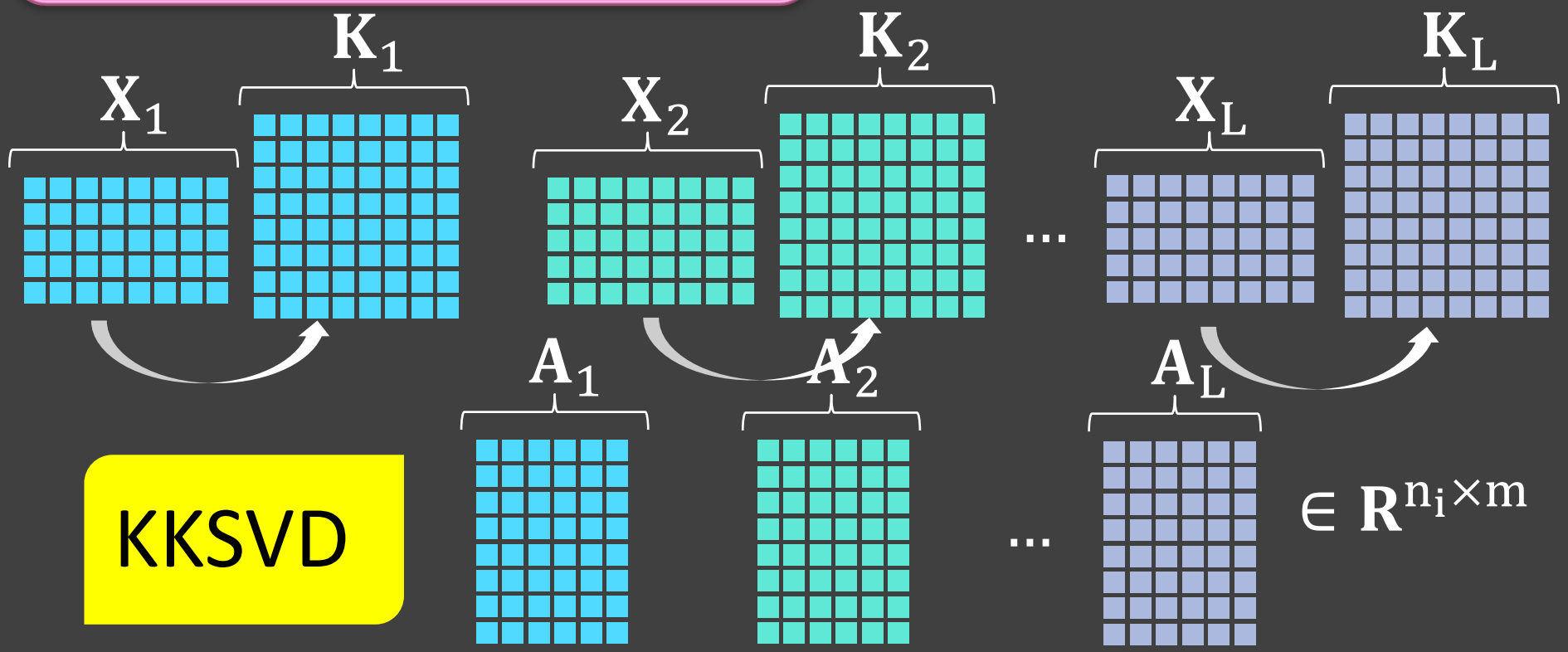
$$\|\mathbf{x}_{\text{test}} - \mathbf{D}_i \boldsymbol{\gamma}_i\|_2^2,$$

$$\forall i = 1 \dots L$$

Classification using KDL

Train L dictionaries:

$$\underset{\mathbf{A}_i,\mathbf{\Gamma}_i}{\operatorname{argmin}} \|\Phi(\mathbf{X}_i) - \Phi(\mathbf{X}_i)\mathbf{A}_i\mathbf{\Gamma}_i\|_F^2$$

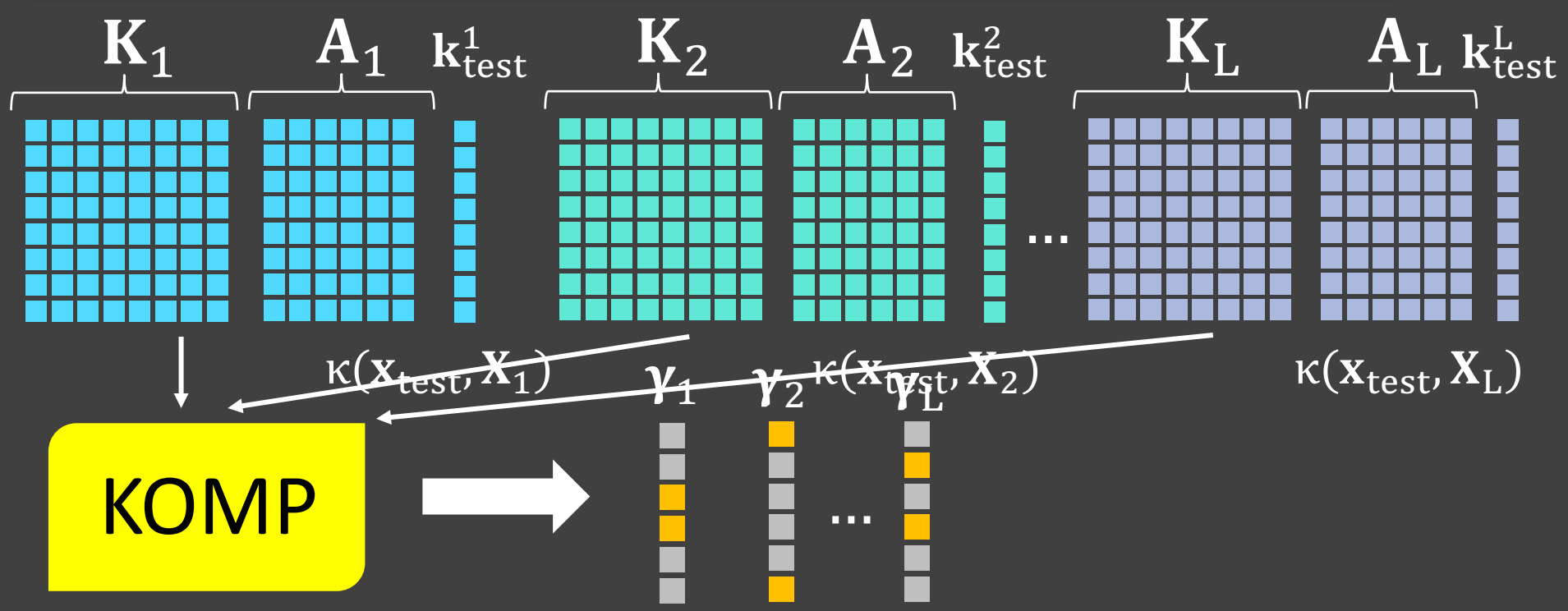


KKSVD

Classification using KDL

Sparse code each test sample over L dictionaries:

solve: $\underset{\boldsymbol{\gamma}_i}{\operatorname{argmin}} \|\Phi(\mathbf{x}_{test}) - \Phi(\mathbf{X}_i)\mathbf{A}_i\boldsymbol{\gamma}_i\|_2^2 \;\; \text{s.t.} \; \|\boldsymbol{\gamma}_i\|_0 \leq q\,, \;\; \forall i = 1 \ldots L$

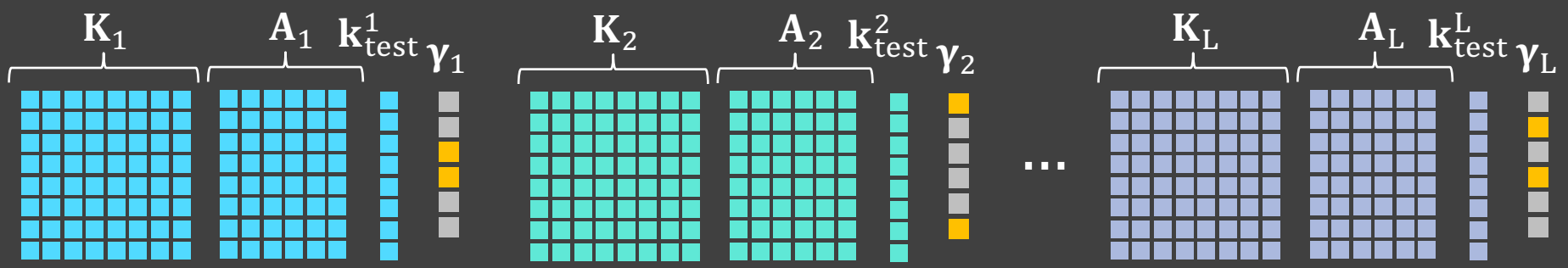


Alona Golts

Classification using KDL

Chosen class is the one with minimal representation error:

$$class = \underset{i}{\operatorname{argmin}}[r_i] = \operatorname{argmin}\|\Phi(\mathbf{x}_{test}) - \Phi(\mathbf{X}_i)\mathbf{A}_i\boldsymbol{\gamma}_i\|_2^2, \qquad \forall i = 1 \dots L$$



$$class = \underset{i}{\operatorname{argmin}}(r_i)$$
$$\forall i = 1 \dots L$$

Classification using LKDL

1.

Sample signals from training set: $\mathbf{X} \rightarrow \mathbf{X}_{\mathrm{R}}$

2.

Compute

$$\mathbf{C} = \mathbf{K}(\mathbf{X}, \mathbf{X}_{\mathrm{R}})$$

3.

Compute

$$\mathbf{W} = \mathbf{K}(\mathbf{X}_{\mathrm{R}}, \mathbf{X}_{\mathrm{R}})$$

4.

Approximate $\mathbf{W}$

$$\mathbf{W} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathrm{T}}$$

5.

Compute virtual train set

$$\mathbf{F} = (\boldsymbol{\Lambda}^{+})^{1/2}\mathbf{V}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}$$

6.

Compute

$$\mathbf{c}_{\mathrm{test}} = \mathbf{K}(\mathbf{x}_{\mathrm{test}}, \mathbf{X}_{\mathrm{R}})$$

7.

Compute virtual test sample

$$\mathbf{f}_{\mathrm{test}} = (\boldsymbol{\Lambda}^{+})^{1/2}\mathbf{V}^{\mathrm{T}}\mathbf{c}_{\mathrm{test}}^{\mathrm{T}}$$

Classification using DL

Results

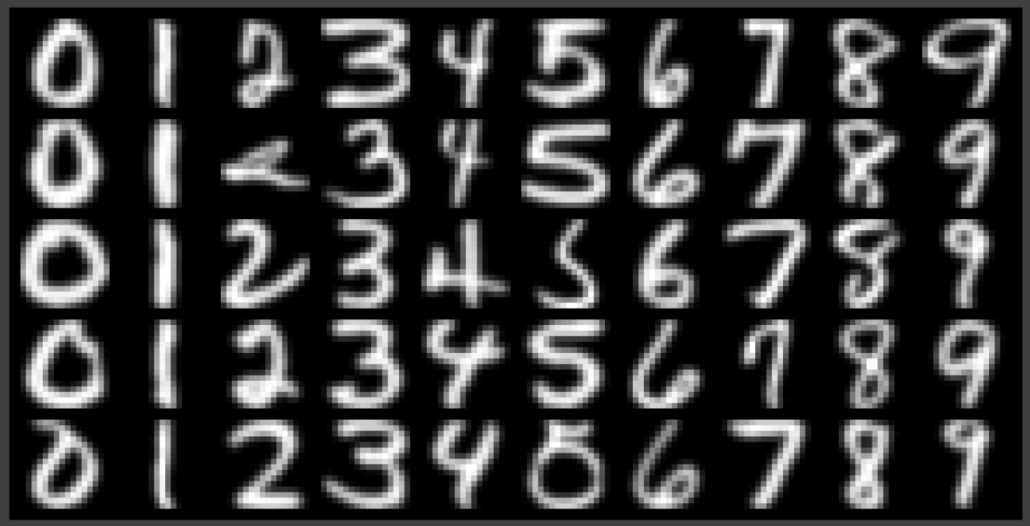
LKDL

Results - Objective

1. LKDL improves discriminability over linear DL.
2. LKDL works as good or better than KDL.
3. LKDL is more efficient with respect to KDL.
4. LKDL can be incorporated seamlessly in virtually any DL algorithm.

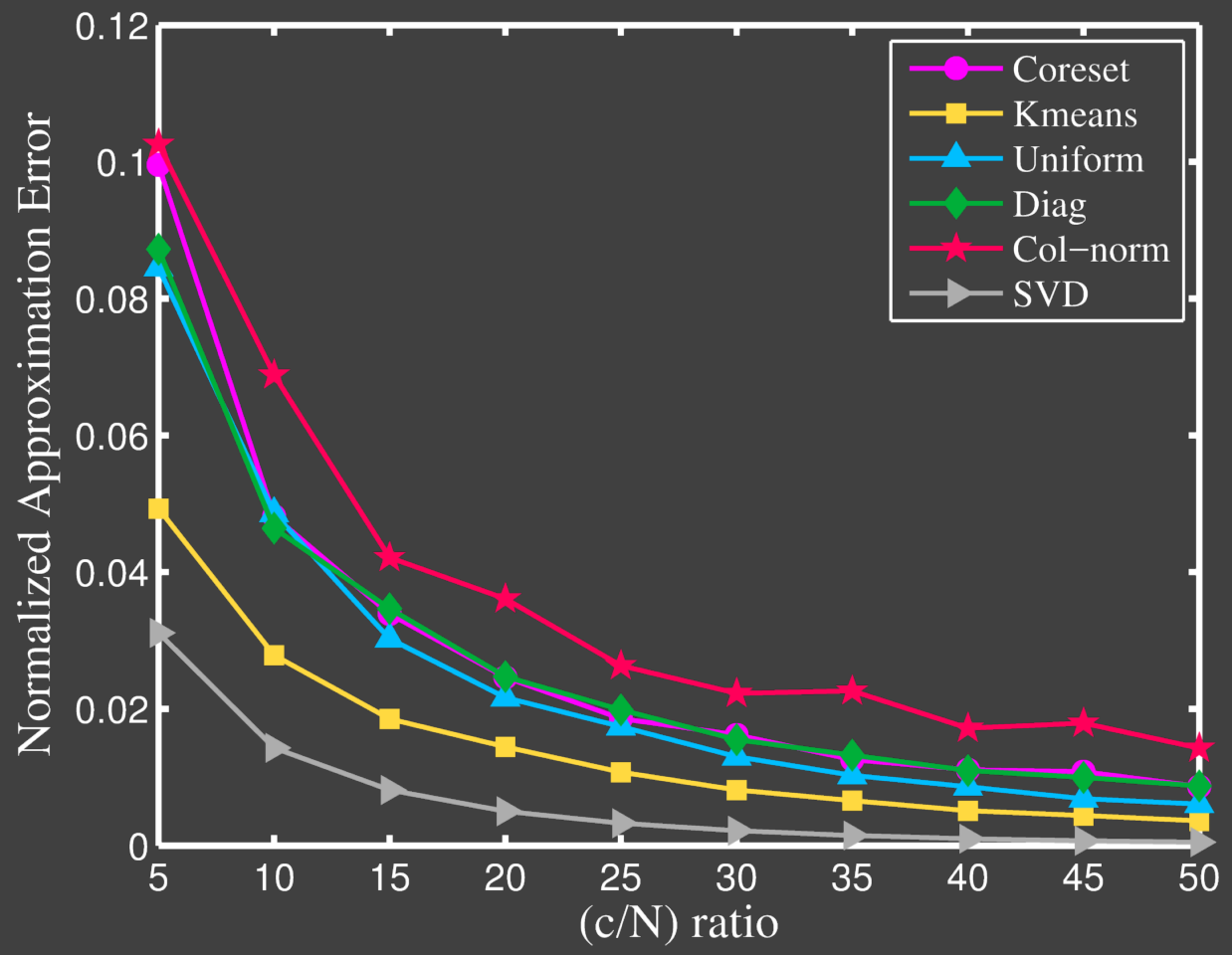
USPS Dataset

signal dim.	256
size of train set	7291
size of test set	2007
# classes	10 (digits)
# atoms per class	300
cardinality	5
# iterations	5
kernel	Polynomial
kernel parameter	2
c – number of samples in Nyström	20% of train samples
k – approx. dim.	256

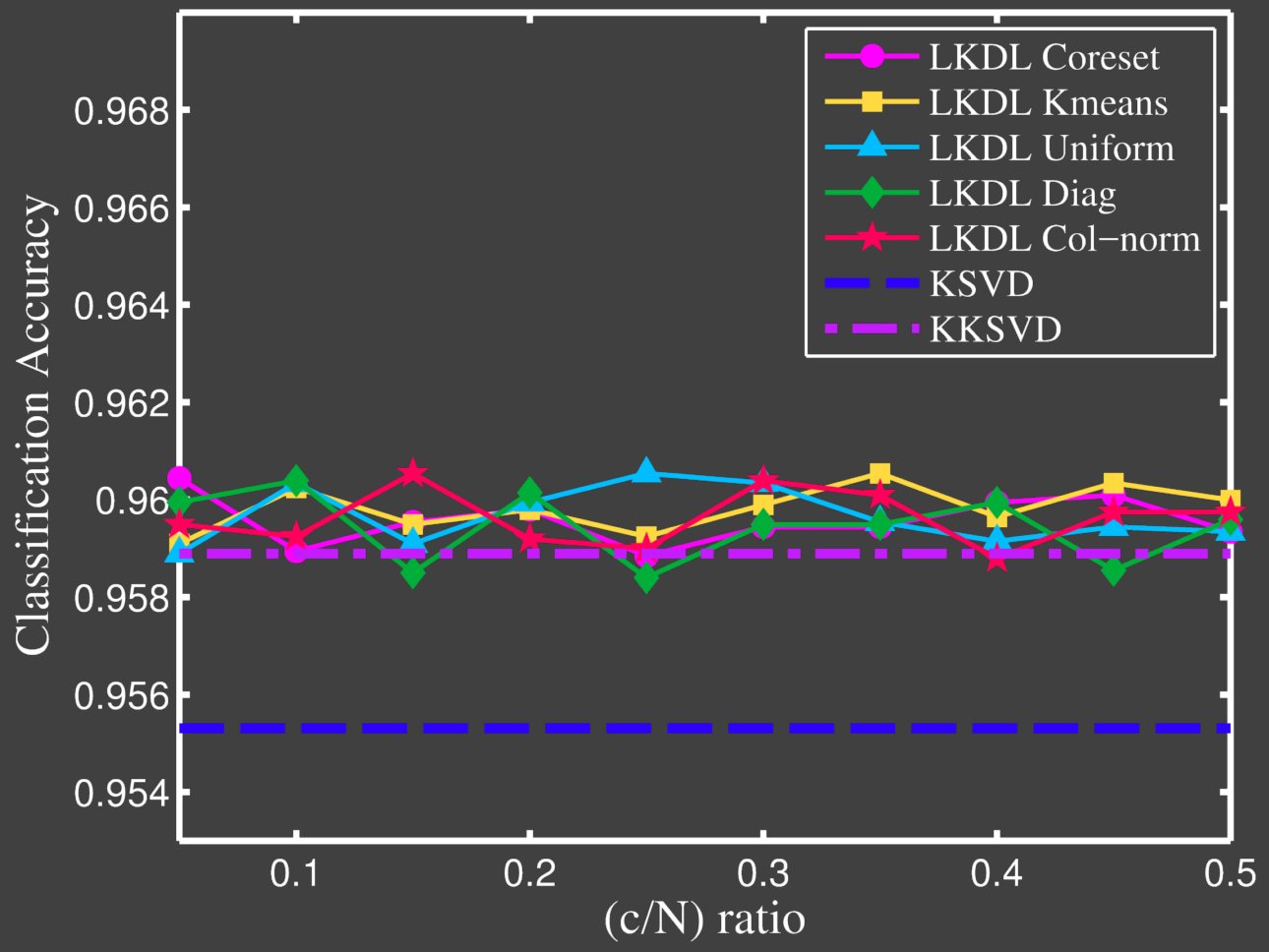


Approximation Quality

$$\mathrm{err} = \frac{\|\mathbf{K} - \widetilde{\mathbf{K}}\|_F}{\|\mathbf{K}\|_F}$$

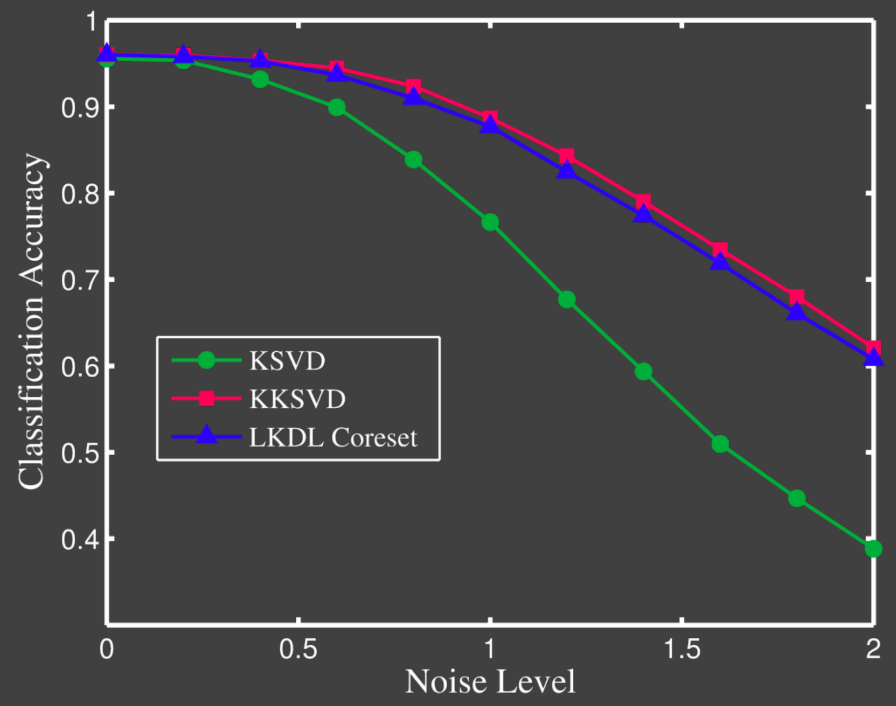


Dependence on c/N

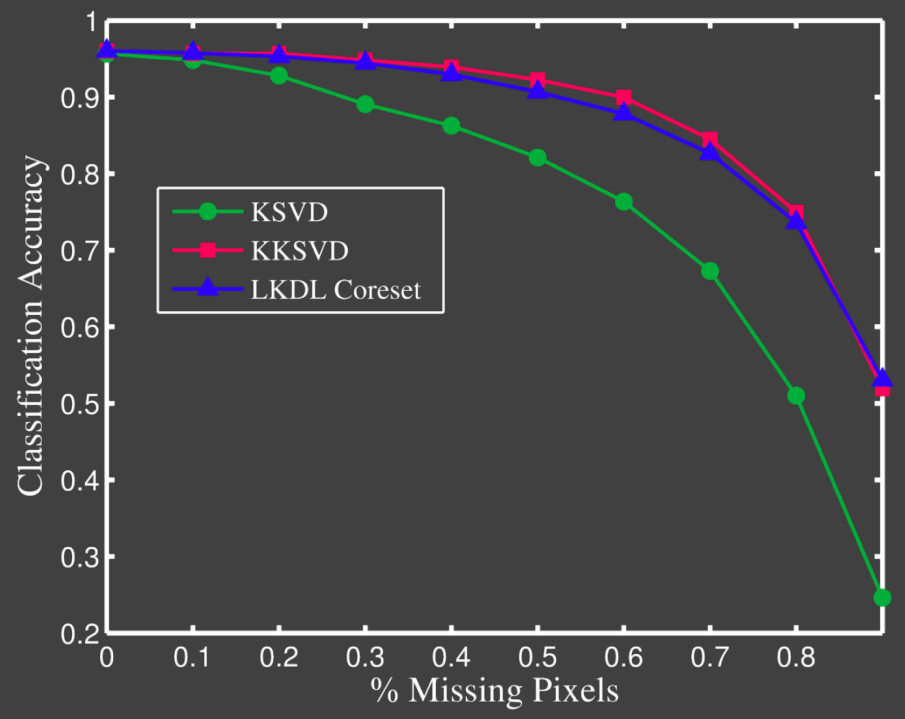


Robustness to Corruptions

Effect of Noise:



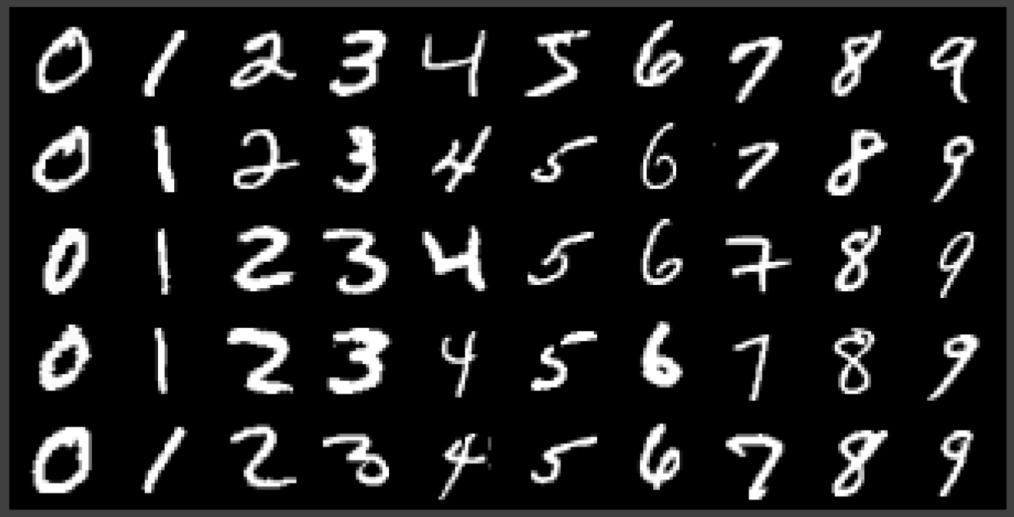
Effect of Missing Pixels:



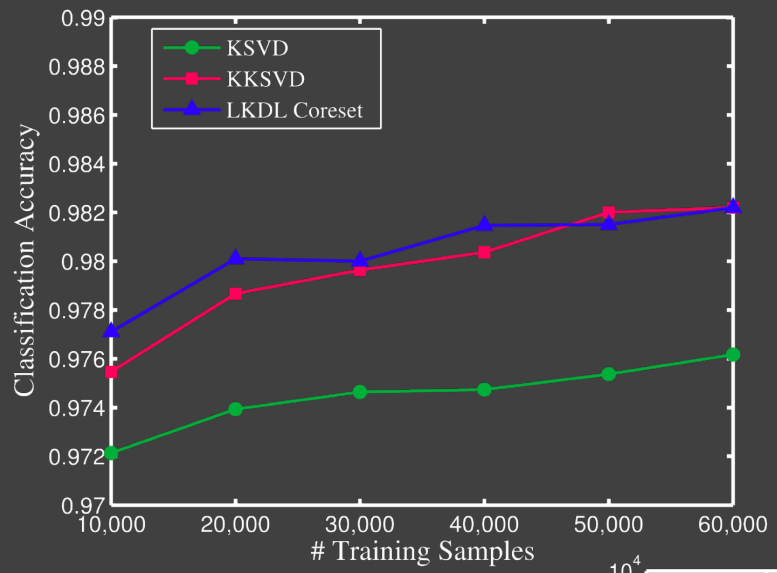
MNIST Dataset

signal dim.	784
size of train set	60,000
size of test set	10,000
# classes	10 (digits)
# atoms per class	700
cardinality	11
# iterations	2
kernel	Polynomial
kernel parameter	2
c – number of samples in Nyström	15% of train samples
k – approx. dim.	784

LeCun *et al.* ('98)

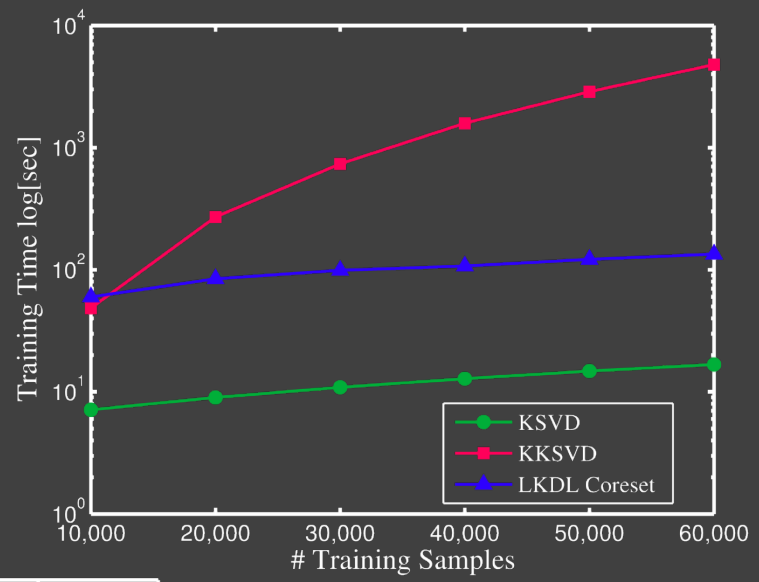


Runtime Improvement

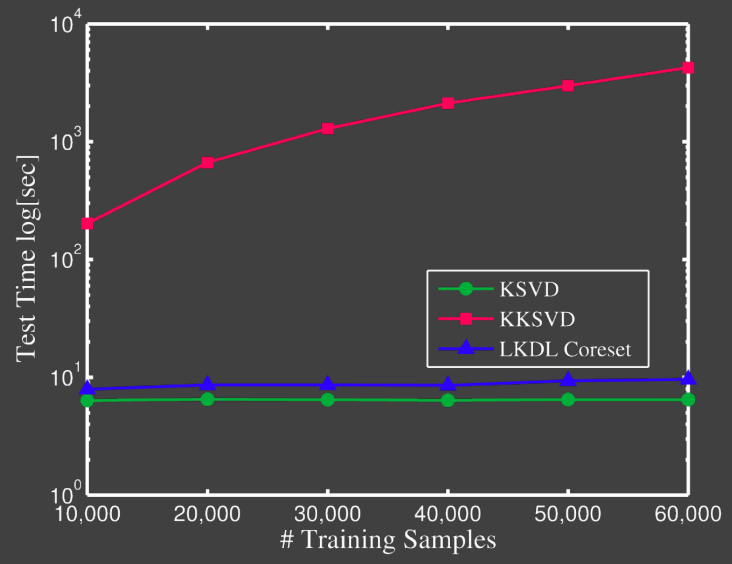


Accuracy

Test Time



Train Time



LKDL – Pros and Cons

The Good

- Introduces nonlinearity to sparse representation algorithms.
- Can scale-up and deal with relatively high number of input samples
- Can be easily added to any dictionary learning algorithm.
- Flexibility with choice of kernel.

The Bad

- Nyström method requires calculating and storing the matrix $\mathbf{C}$, which is large in itself
- Eigen-decomposition of $\mathbf{W}$ is computationally demanding for very large datasets
- Virtual samples don't usually relate to the original data, thus image processing tasks off limits

Summary

- There are benefits in using kernels in DL-based classification tasks.
- Kernel DL improves accuracy over DL but suffers from dimensionality problems.
- LKDL – a method of combining kernels as features and using linear DL on top of them, was presented.
- LKDL provides comparable accuracy to KDL, with faster training and testing.
- LKDL can combined on top of any DL algorithm.

Thank You!

Kernel KSVD

Update stage:

$$\|\Phi(\mathbf{X}) - \Phi(\mathbf{X})\mathbf{A}\boldsymbol{\Gamma}\|_F^2 = \left\|\Phi(\mathbf{X}) - \Phi(\mathbf{X})\sum_{j=1}^{m}\mathbf{a_j}\boldsymbol{\gamma}^j\right\|_F^2 =$$

$$= \left\|\Phi(\mathbf{X})\left(\mathbf{I} - \sum_{j\neq k}^{m}\mathbf{a}_j\boldsymbol{\gamma}^j\right) - \Phi(\mathbf{X})\left(\mathbf{a}_k\boldsymbol{\gamma}^j\right)\right\|_F^2 = \|\Phi(\mathbf{X})\mathbf{E}_k - \Phi(\mathbf{X})\mathbf{M}_k\|_F^2$$

$$\mathbf{E}_k^R = \mathbf{E}_k\boldsymbol{\Omega}_k \rightarrow \left\|\Phi(\mathbf{X})\mathbf{E}_k^R - \Phi(\mathbf{X})\left(\mathbf{a}_k\boldsymbol{\gamma}_R^k\right)\right\|_F^2 \longleftarrow \text{Rank-1}$$

$$\Phi(\mathbf{X})\mathbf{E}_k^R = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \rightarrow \quad \Phi(\mathbf{X})\mathbf{a}_k\boldsymbol{\gamma}_R^k = \sigma_1\mathbf{u}_1\mathbf{v}_1^T,$$

$$\boldsymbol{\gamma}_R^k = \sigma_1\mathbf{v}_1^T, \qquad \Phi(\mathbf{X})\mathbf{a}_k = \mathbf{u}_1, \qquad \mathbf{a_k} = \sigma_1^{-1}\mathbf{E}_k^R\mathbf{v}_1$$

Return