CSE 6230: Dimensionality Reduction

Srinivas Eswar
Argonne National Laboratory
30-Mar-2023

Outline

1. Motivation
2. **Linear** Methods
    a. Unconstrained
    b. Constrained
3. **Nonlinear** Methods
    a. Kernel methods
    b. Autoencoders
4. Demonstration

Outline

1. Motivation
2. **Linear** Methods
   a. Unconstrained
   b. Constrained
3. **Nonlinear** Methods
   a. Kernel methods
   b. Autoencoders
4. Demonstration

Outline

1. Motivation
2. **Linear** Methods
    a. Unconstrained
    b. Constrained
3. **Nonlinear** Methods
    a. Kernel methods
    b. Autoencoders
4. Demonstration

Outline

1. Motivation
2. **Linear** Methods
    a. Unconstrained
    b. Constrained
3. **Nonlinear** Methods
    a. Kernel methods
    b. Autoencoders
4. Demonstration

Outline

1. Motivation
2. **Linear** Methods
    a. Unconstrained
    b. Constrained
3. **Nonlinear** Methods
    a. Kernel methods
    b. Autoencoders
4. Demonstration

Notation

Standard input

1. **Samples** of dimension m are arranged as **columns** of a matrix.
    a. MNIST: 784 x 70000.
    b. SC22: 7.06m documents, 405m proteins, 10m geospatial locations.
2. Mainly consider the **distributed-memory** model.
3. ***Assume* that data lies in or near a low-dimensional structure!**

$$\mathbf{X} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & \cdots & | \end{bmatrix} \in \mathbb{R}^{m \times n}$$

*https://www.hpcwire.com/2022/08/12/sc22-unveils-acm-gordon-bell-prize-finalists/

Notation

Standard input

1. **Samples** of dimension m are arranged as **columns** of a matrix.
    a. MNIST: 784 x 70000.
    b. SC22: 7.06m documents, 405m proteins, 10m geospatial locations.
2. Mainly consider the **distributed-memory** model.
3. *Assume* **that data lies in or near a low-dimensional structure!**

$$\mathbf{X} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & \cdots & | \end{bmatrix} \in \mathbb{R}^{m \times n}$$

*https://www.hpcwire.com/2022/08/12/sc22-unveils-acm-gordon-bell-prize-finalists/

Notation

Standard input

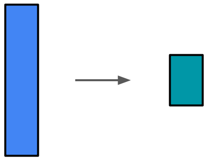
1. **Samples** of dimension m are arranged as **columns** of a matrix.
   a. MNIST: 784 x 70000.
   b. SC22: 7.06m documents, 405m proteins, 10m geospatial locations.
2. Mainly consider the **distributed-memory** model.
3. ***Assume* that data lies in or near a low-dimensional structure!**

$$\mathbf{X} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & \cdots & | \end{bmatrix} \in \mathbb{R}^{m \times n}$$

*https://www.hpcwire.com/2022/08/12/sc22-unveils-acm-gordon-bell-prize-finalists/

Motivation

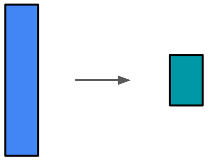
1. Data **compression**.
   a. Compress vectors to a smaller dimension (say **m** to **k**).
   b. Savings in space.
   c. Savings in computation.

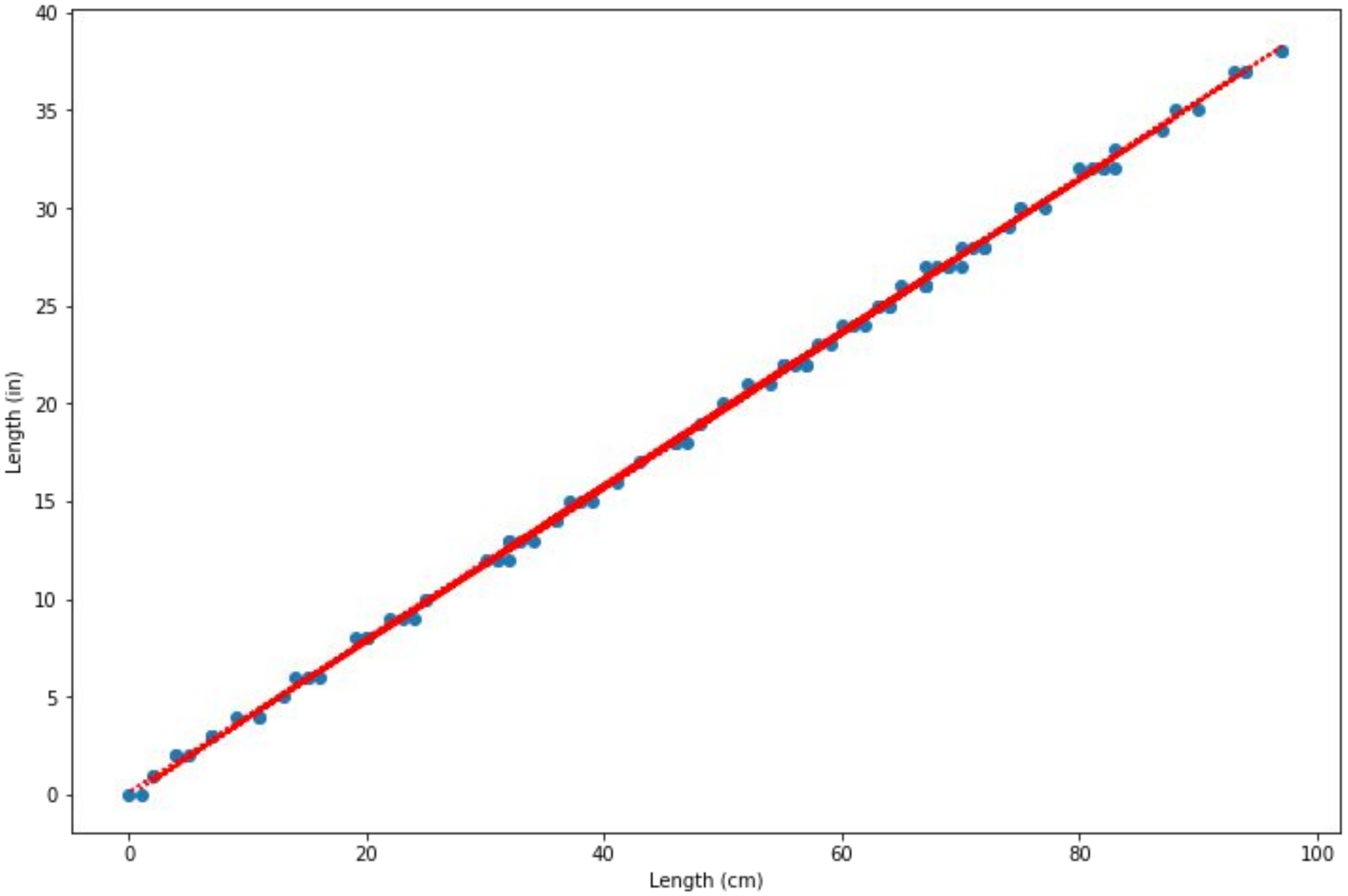


2. Feature **finagling**.
   a. Removes **redundant** or highly **correlated** features.
   b. Discover **hidden** correlations.
   c. **Noisy** features.
3. **Visualise**.
4. No other choice!

Motivation

1. Data **compression**.
    a. Compress vectors to a smaller dimension (say **m** to **k**).
    b. Savings in space.
    c. Savings in computation.
2. Feature **finagling**.
    a. Removes **redundant** or highly **correlated** features.
    b. Discover **hidden** correlations.
    c. **Noisy** features.
3. **Visualise**.
4. No other choice!

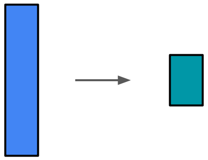




40
35
30
25
20
15
10
5
0
Length (in)
0
20
40
60
80
100
Length (cm)

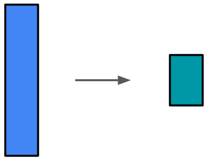
Motivation

1. Data **compression**.
    a. Compress vectors to a smaller dimension (say **m** to **k**).
    b. Savings in space.
    c. Savings in computation.
2. Feature **finagling**.
    a. Removes **redundant** or highly **correlated** features.
    b. Discover **hidden** correlations.
    c. **Noisy** features.
3. **Visualise**.
4. No other choice!



Motivation

1. Data **compression**.
    a. Compress vectors to a smaller dimension (say **m** to **k**).
    b. Savings in space.
    c. Savings in computation.
2. Feature **finagling**.
    a. Removes **redundant** or highly **correlated** features.
    b. Discover **hidden** correlations.
    c. **Noisy** features.
3. **Visualise**.
4. No other choice!



Outline

1. Motivation
2. **Linear** Methods
    a. Unconstrained
    b. Constrained
3. **Nonlinear** Methods
    a. Kernel methods
    b. Autoencoders
4. Demonstration

Linear Methods

Approximate the input in a new **basis**.

$$\underset{m\times n}{\mathbf{X}} \approx \underset{m\times k}{\mathbf{W}} \cdot \underset{k\times n}{\mathbf{H}}$$

1. **Simplicity**.
   a. Basecase to most nonlinear methods.
   b. Analysable.
2. **Interpretability** and extensibility.
3. **Fast** and **scalable** methods.
   a. Standard libraries (BLAS, LAPACK, …) and constant improvement (communication-avoiding, randomisation, …).

Linear Methods

Approximate the input in a new **basis**.

$$\underset{m \times n}{\mathbf{X}} \approx \underset{m \times k}{\mathbf{W}} \cdot \underset{k \times n}{\mathbf{H}}$$

1. **Simplicity**.
   a. Basecase to most nonlinear methods.
   b. Analysable.
2. **Interpretability** and extensibility.
3. **Fast** and **scalable** methods.
   a. Standard libraries (BLAS, LAPACK, …) and constant improvement (communication-avoiding, randomisation, …).

Linear Methods

Approximate the input in a new **basis**.

$$\underset{m \times n}{\mathbf{X}} \approx \underset{m \times k}{\mathbf{W}} \cdot \underset{k \times n}{\mathbf{H}}$$

1. **Simplicity**.
   a. Basecase to most nonlinear methods.
   b. Analysable.
2. **Interpretability** and extensibility.
3. **Fast** and **scalable** methods.
   a. Standard libraries (BLAS, LAPACK, …) and constant improvement (communication-avoiding, randomisation, …).

Linear Methods

Approximate the input in a new **basis**.

$$\underset{m\times n}{\mathbf{X}} \approx \underset{m\times k}{\mathbf{W}} \cdot \underset{k\times n}{\mathbf{H}}$$

1. **Simplicity**.
   a. Basecase to most nonlinear methods.
   b. Analysable.
2. **Interpretability** and extensibility.
3. **Fast** and **scalable** methods.
   a. Standard libraries (BLAS, LAPACK, …) and constant improvement (communication-avoiding, randomisation, …).

Linear Methods - SVD

Singular Value Decomposition

$$\mathbf{X} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

1. **Gold standard** for teasing a matrix apart.
2. **Minimises** both the 2-norm and Frobenius-norm solutions.

3. Two broad categories of solvers: **dense** and **sparse**.

Linear Methods - SVD

Singular Value Decomposition

$$\mathbf{X} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

1. **Gold standard** for teasing a matrix apart.
2. **Minimises** both the 2-norm and Frobenius-norm solutions.
3. Two broad categories of solvers: **dense** and **sparse**.

Linear Methods - SVD

Singular Value Decomposition

$$\mathbf{X} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

1. **Gold standard** for teasing a matrix apart.
2. **Minimises** both the 2-norm and Frobenius-norm solutions.

$$\min_{\text{rank}(\mathbf{A}) \leq k} \|\mathbf{X} - \mathbf{A}\|_\xi$$

3. Two broad categories of solvers: **dense** and **sparse**.

Linear Methods - SVD

Singular Value Decomposition

$$\mathbf{X} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$$

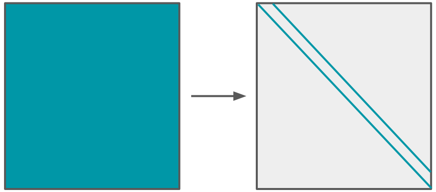
1. **Gold standard** for teasing a matrix apart.
2. **Minimises** both the 2-norm and Frobenius-norm solutions.

$$\min_{\text{rank}(\mathbf{A}) \leq k} \|\mathbf{X} - \mathbf{A}\|_\xi$$

3. Two broad categories of solvers: **dense** and **sparse**.

Linear Methods - SVD

Dense Case



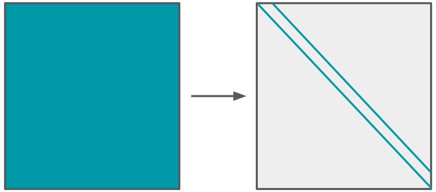
$$\mathbf{X} \rightarrow \mathbf{U}_1\mathbf{B}\mathbf{V}_1^\top \rightarrow \mathbf{U}_1\mathbf{U}_2\mathbf{\Sigma}\mathbf{V}_2^\top\mathbf{V}_1^\top$$

1. Reduction to **bidiagonal** form via two-sided **orthogonal** transformations.
2. Solve the bidiagonal matrix **iteratively**.
3. BLAS calls **xGESVD** and **xGEBRD**.

*LAPACK Users' Guide

Linear Methods - SVD

Dense Case



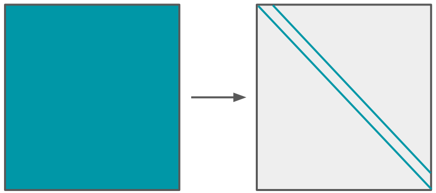
$$\mathbf{X} \to \mathbf{U}_1 \mathbf{B} \mathbf{V}_1^\top \to \mathbf{U}_1 \mathbf{U}_2 \mathbf{\Sigma} \mathbf{V}_2^\top \mathbf{V}_1^\top$$

1. Reduction to **bidiagonal** form via two-sided **orthogonal** transformations.
2. Solve the bidiagonal matrix **iteratively**.
3. BLAS calls **xGESVD** and **xGEBRD**.

*LAPACK Users' Guide

Linear Methods - SVD

Dense Case



$$\mathbf{X} \rightarrow \mathbf{U}_1 \mathbf{B} \mathbf{V}_1^\top \rightarrow \mathbf{U}_1 \mathbf{U}_2 \mathbf{\Sigma} \mathbf{V}_2^\top \mathbf{V}_1^\top$$

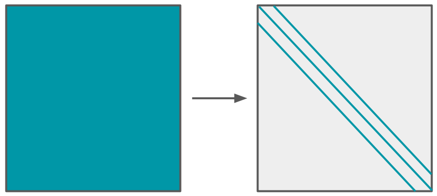
1. Reduction to **bidiagonal** form via two-sided **orthogonal** transformations.
2. Solve the bidiagonal matrix **iteratively**.
3. BLAS calls **xGESVD** and **xGEBRD**.

*LAPACK Users' Guide

Linear Methods - SVD

Other condensed forms.

1. **Tridiagonal** form.
   a. Symmetric Eigenvalue Problems.
   b. **xSYTRD** routine.
2. Upper **Hessenberg** form.
   a. Nonsymmetric Eigenvalue Problems.
   b. **xGEHRD** routine.
3. Employ **Householder** reflectors.

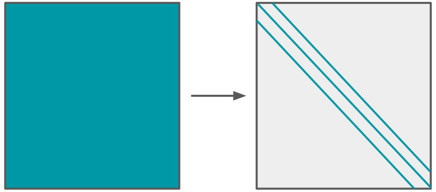


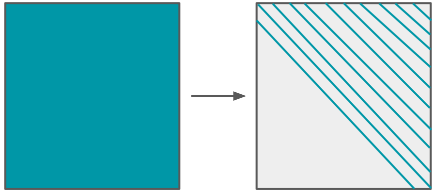
Ballard, Demmel, Knight. “Avoiding communication in successive band reduction” (2015)

Linear Methods - SVD

Other condensed forms.

1. **Tridiagonal** form.
    a. Symmetric Eigenvalue Problems.
    b. **xSYTRD** routine.
2. Upper **Hessenberg** form.
    a. Nonsymmetric Eigenvalue Problems.
    b. **xGEHRD** routine.
3. Employ **Householder** reflectors.



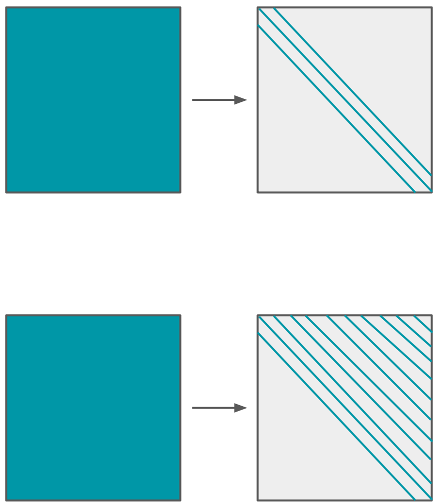


Ballard, Demmel, Knight. “Avoiding communication in successive band reduction” (2015)

Linear Methods - SVD

Other condensed forms.

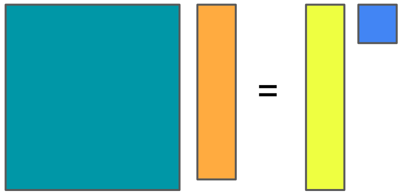
1. **Tridiagonal** form.
   a. Symmetric Eigenvalue Problems.
   b. **xSYTRD** routine.
2. Upper **Hessenberg** form.
   a. Nonsymmetric Eigenvalue Problems.
   b. **xGEHRD** routine.
3. Employ **Householder** reflectors.



Ballard, Demmel, Knight. “Avoiding communication in successive band reduction” (2015)

Linear Methods - SVD

Sparse Case



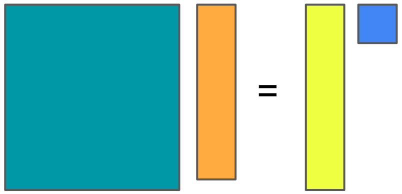
$$\mathbf{X}\mathbf{V}_k = \mathbf{U}_k\mathbf{B}_k$$

1. Lanczos bidiagonalisation to generate a **k-by-k** system.
2. Need only **xGEMV** calls.

Larsen. “Lanczos bidiagonalisation with partial reorthogonalisation” (1998)

Linear Methods - SVD

Sparse Case



$$\mathbf{X}\mathbf{V}_k = \mathbf{U}_k\mathbf{B}_k$$

1. Lanczos bidiagonalisation to generate a **k-by-k** system.
2. Need only **xGEMV** calls.

Larsen. “Lanczos bidiagonalisation with partial reorthogonalisation” (1998)

Linear Methods - SVD

1. Choose a starting vector $p_0 \in \mathbb{R}^m$, and let $\beta_1 = \|p_0\|_2,\ \ u_1 = p_0/\beta_1$ and $v_0 \equiv 0$
2. **for** $j = 1, 2, \ldots, k$ **do**

$$r_j = A^T u_j - \beta_j v_{j-1}$$
$$\alpha_j = \|r_j\|_2$$
$$v_j = r_j/\alpha_j$$
$$p_j = A v_j - \alpha_j u_j$$
$$\beta_{j+1} = \|p_j\|_2$$
$$u_{j+1} = p_j/\beta_{j+1}$$

end

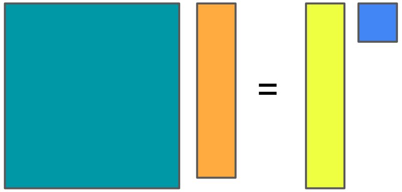
$$\mathbf{B}_k = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{bmatrix}$$

Larsen. “Lanczos bidiagonalisation with partial reorthogonalisation” (1998)

Linear Methods - SVD

Randomisation.

1. Multiply the input by a ***random*** matrix Ω (n-by-(k+p)) to find its range.
2. The SVD is now **approximated** in this range (upto k).



$$\mathbf{X\Omega} = \mathbf{A} = \mathbf{QR}$$

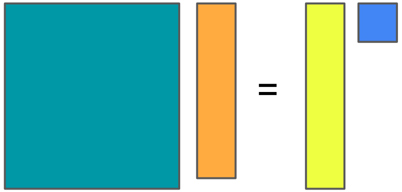
$$\text{svd}(\mathbf{Q}^\top \mathbf{X})$$

Halko, Martinsson, Tropp. “Finding Structure with Randomness” (2011)
Gu. “Subspace Iteration Randomisation and Singular Value Problems” (2015)

Linear Methods - SVD

Randomisation.

1. Multiply the input by a ***random*** matrix Ω (n-by-(k+p)) to find its range.
2. The SVD is now **approximated** in this range (upto k).



$$\mathbf{X\Omega} = \mathbf{A} = \mathbf{QR}$$

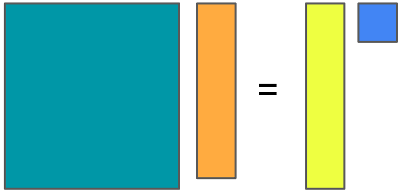
$$\text{svd}(\mathbf{Q}^\top \mathbf{X})$$

Halko, Martinsson, Tropp. “Finding Structure with Randomness” (2011)
Gu. “Subspace Iteration Randomisation and Singular Value Problems” (2015)

Linear Methods - SVD

Randomisation.

1. Multiply the input by a ***random*** matrix Ω (n-by-(k+p)) to find its range.
2. The SVD is now **approximated** in this range (upto k).



$$\mathbf{X}\boldsymbol{\Omega} = \mathbf{A} = \mathbf{Q}\mathbf{R}$$

$$\text{svd}(\mathbf{Q}^\top \mathbf{X})$$

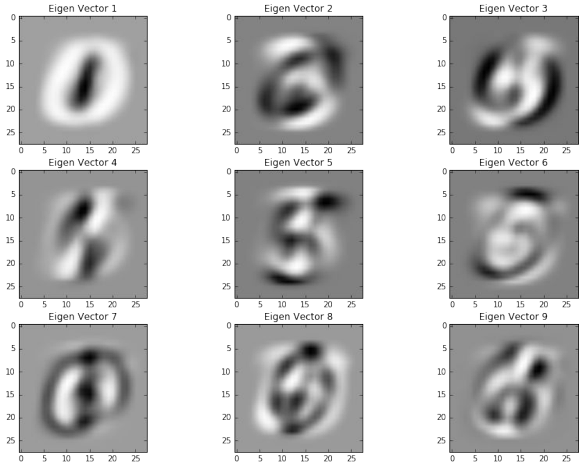
$$\mathbb{E}\|\mathbf{X} - \hat{\mathbf{X}}_k\|_F \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}$$

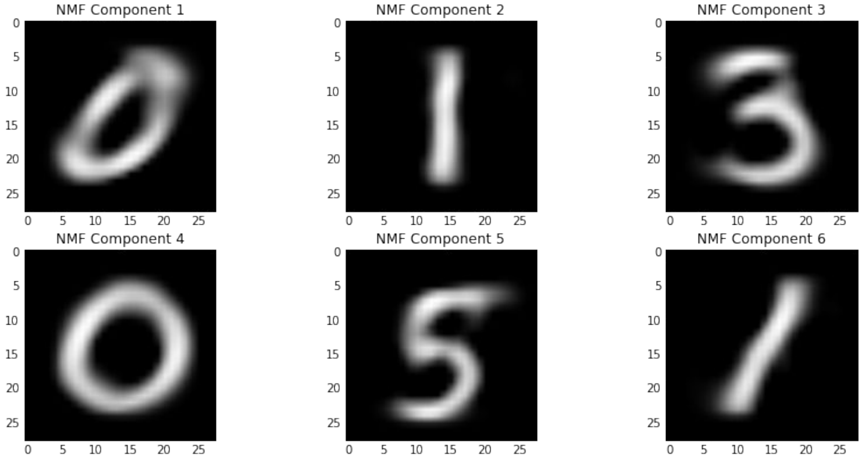
$$\mathbb{E}\|\mathbf{X} - \hat{\mathbf{X}}_k\|_2 \leq \left(1 + \sqrt{\frac{k}{p-1}}\right) \sigma_{k+1} + \frac{e\sqrt{k+p}}{p} \left(\sum_{j>k} \sigma_j^2\right)^{1/2}$$

Halko, Martinsson, Tropp. “Finding Structure with Randomness” (2011)
Gu. “Subspace Iteration Randomisation and Singular Value Problems” (2015)

Linear Methods - NMF

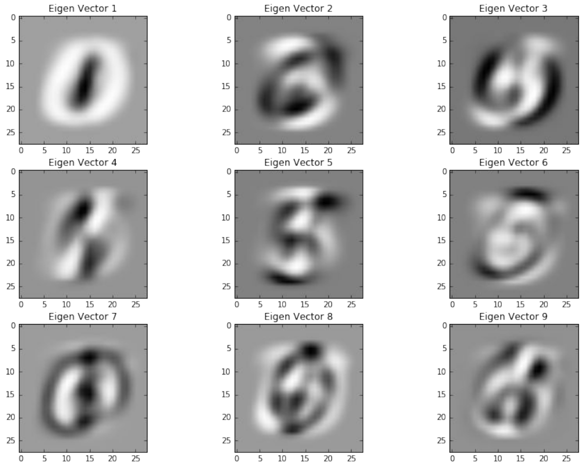
1. **SVD** not good for **interpretability**.
2. Can impose **constraints** on factors to improve interpretability (at what cost?).
    a. Column Subset Methods, Sparse Dictionary Learning, Nonnegative Matrix Factorisation.

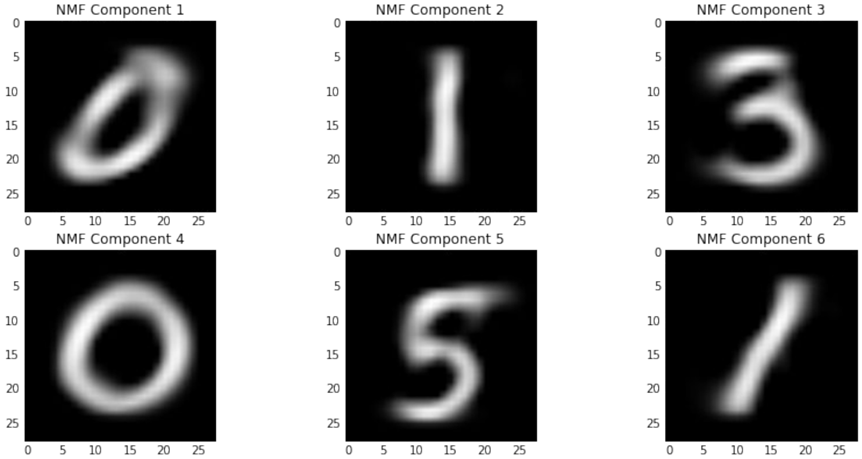




Linear Methods - NMF

1. **SVD** not good for **interpretability**.
2. Can impose **constraints** on factors to improve interpretability (at what cost?).
    a. Column Subset Methods, Sparse Dictionary Learning, Nonnegative Matrix Factorisation.





Linear Methods - NMF

Nonnegative Matrix Factorisation

$$\min_{\mathbf{W}\geq 0, \mathbf{H}\geq 0} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$$

1. **Block Coordinate Descent**.
   a. Splits the **variables** into **subsets** which are easier to compute.
2. Bottleneck computation becomes a **xGEMM** call.
   a. All the terms in the **gradient**.
3. **Three variants** of xGEMM in the distributed case.
   a. What variant is NMF in?

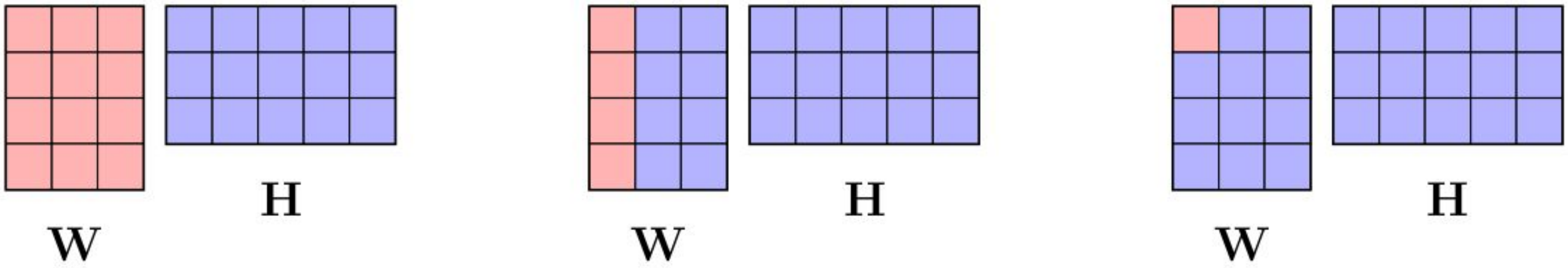
Linear Methods - NMF

Nonnegative Matrix Factorisation

$$\min_{\mathbf{W}\geq 0, \mathbf{H}\geq 0} \|\mathbf{X} - \mathbf{WH}\|_F^2$$

1. **Block Coordinate Descent**.
   a. Splits the **variables** into **subsets** which are easier to compute.
2. Bottleneck computation becomes a **xGEMM** call.
   a. All the terms in the **gradient**.
3. **Three variants** of xGEMM in the distributed case.
   a. What variant is NMF in?

Linear Methods - NMF



(a) Two blocks. (b) $2k$ blocks. (c) $(m+n)\,k$ blocks.

Linear Methods - NMF

Nonnegative Matrix Factorisation

$$\min_{\mathbf{W}\geq 0, \mathbf{H}\geq 0} ||\mathbf{X} - \mathbf{WH}||_F^2$$

1. **Block Coordinate Descent**.
   a. Splits the **variables** into **subsets** which are easier to compute.
2. Bottleneck computation becomes a **xGEMM** call.
   a. All the terms in the **gradient**.
3. **Three variants** of xGEMM in the distributed case.
   a. What variant is NMF in?

Linear Methods - NMF

Gradient computation.

1. Matrix multiplications involving the **input matrix**.
   a. Major **bottleneck**.
2. **Gram matrix** computations.
   a. Also causes **communication** in distributed case.

$$\nabla_{\mathbf{W}} = 2(\mathbf{W}\mathbf{H}\mathbf{H}^\top - \mathbf{X}\mathbf{H}^\top)$$

$$\nabla_{\mathbf{H}} = 2(\mathbf{W}^\top\mathbf{W}\mathbf{H} - \mathbf{W}^\top\mathbf{X})$$

Kannan, Ballard, Park. “A high-performance parallel algorithm for nonnegative matrix factorisation” (2016)

Linear Methods - NMF

Gradient computation.

1. Matrix multiplications involving the **input matrix**.
   a. Major **bottleneck**.
2. **Gram matrix** computations.
   a. Also causes **communication** in distributed case.

$$\nabla_{\mathbf{W}} = 2(\mathbf{W}\mathbf{H}\mathbf{H}^\top - \mathbf{X}\mathbf{H}^\top)$$

$$\nabla_{\mathbf{H}} = 2(\mathbf{W}^\top\mathbf{W}\mathbf{H} - \mathbf{W}^\top\mathbf{X})$$

Kannan, Ballard, Park. “A high-performance parallel algorithm for nonnegative matrix factorisation” (2016)

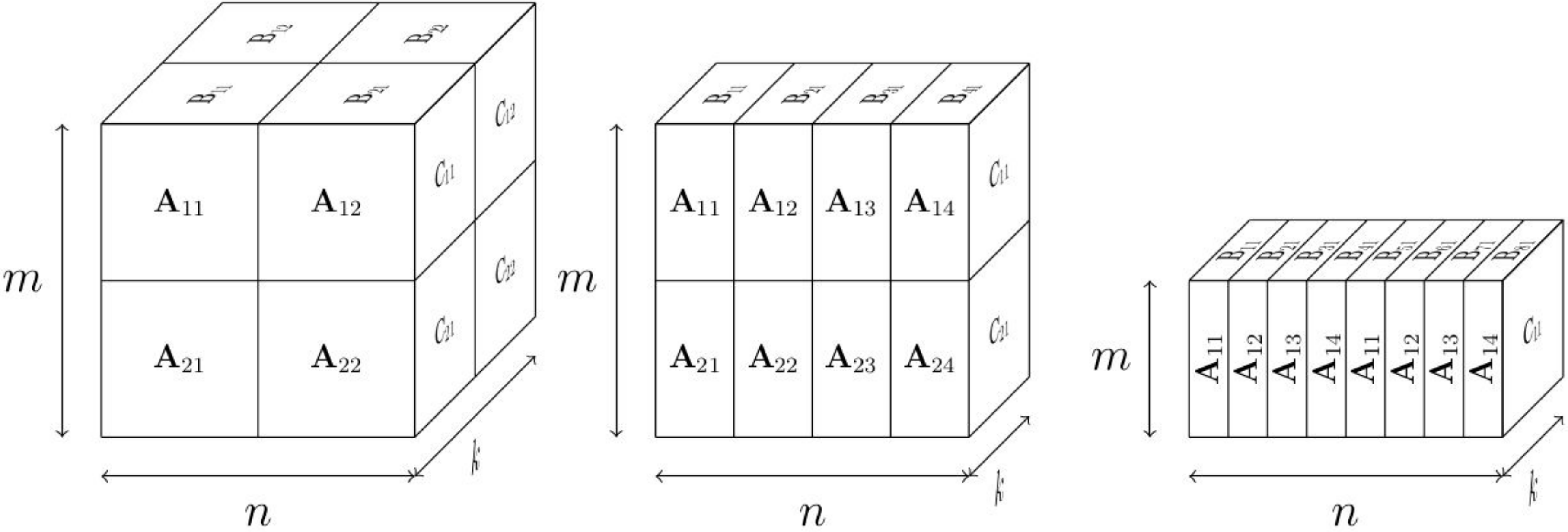
Linear Methods - NMF

Nonnegative Matrix Factorisation

$$\min_{\mathbf{W}\geq 0, \mathbf{H}\geq 0} \|\mathbf{X} - \mathbf{W}\mathbf{H}\|_F^2$$

1. **Block Coordinate Descent**.
    a. Splits the **variables** into **subsets** which are easier to compute.
2. Bottleneck computation becomes a **xGEMM** call.
    a. All the terms in the **gradient**.
3. **Three variants** of xGEMM in the distributed case.
    a. What variant is NMF in?

Linear Methods - NMF



(a) Three large dimensions. (b) Two large dimensions. (c) One large dimension.

Demmel et al.. “Communication-optimal parallel recursive rectangular matrix multiplication” (2013)
Daas et al. “Brief Announcement: Tight Memory-Independent Parallel Matrix Multiplication Communication Lower Bounds” (2022)

Outline

1. Motivation
2. **Linear** Methods
   a. Unconstrained
   b. Constrained
3. **Nonlinear** Methods
   a. Kernel methods
   b. Autoencoders
4. Demonstration

Nonlinear Methods

1. Sometimes linear methods don’t cut it!
    a. Distance no longer **Euclidean**.
2. Lower dimensional **manifold**.
    a. **Embedded** in higher dimensional ambient space.
    b. **Geodesic** distance is the measure.
    c. Usually measured as a **graph walk** or via a **kernel**.
3. Some **caveats**.
    a. Multiple **hyperparameter** choices.
    b. Hard to **interpret** discovered manifold.

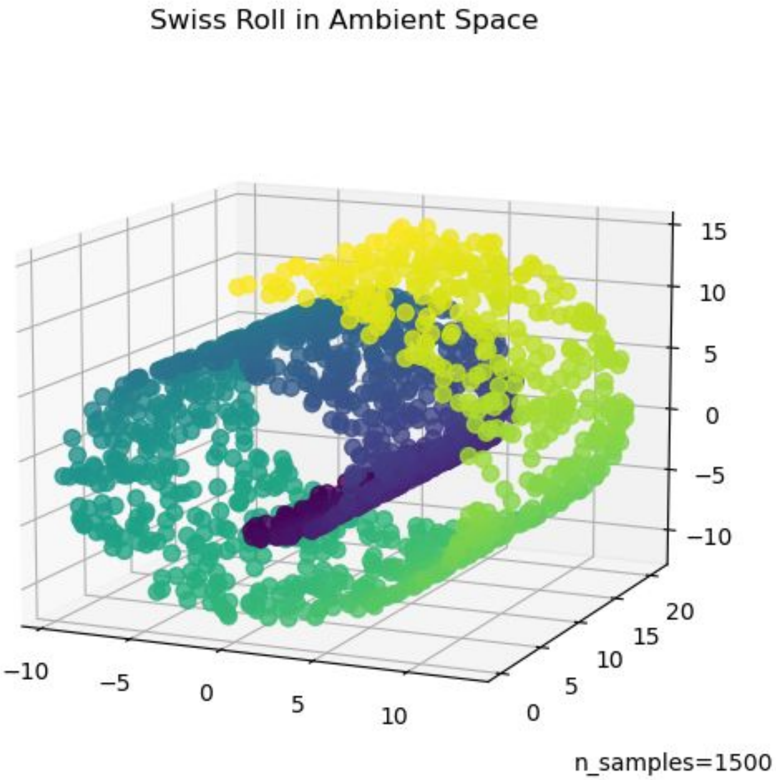


Image source: scikit-learn, “Swiss Roll and Swiss-Hole Reduction”

Nonlinear Methods

1. Sometimes linear methods don’t cut it!
   a. Distance no longer **Euclidean**.
2. Lower dimensional **manifold**.
   a. **Embedded** in higher dimensional ambient space.
   b. **Geodesic** distance is the measure.
   c. Usually measured as a **graph walk** or via a **kernel**.
3. Some **caveats**.
   a. Multiple **hyperparameter** choices.
   b. Hard to **interpret** discovered manifold.

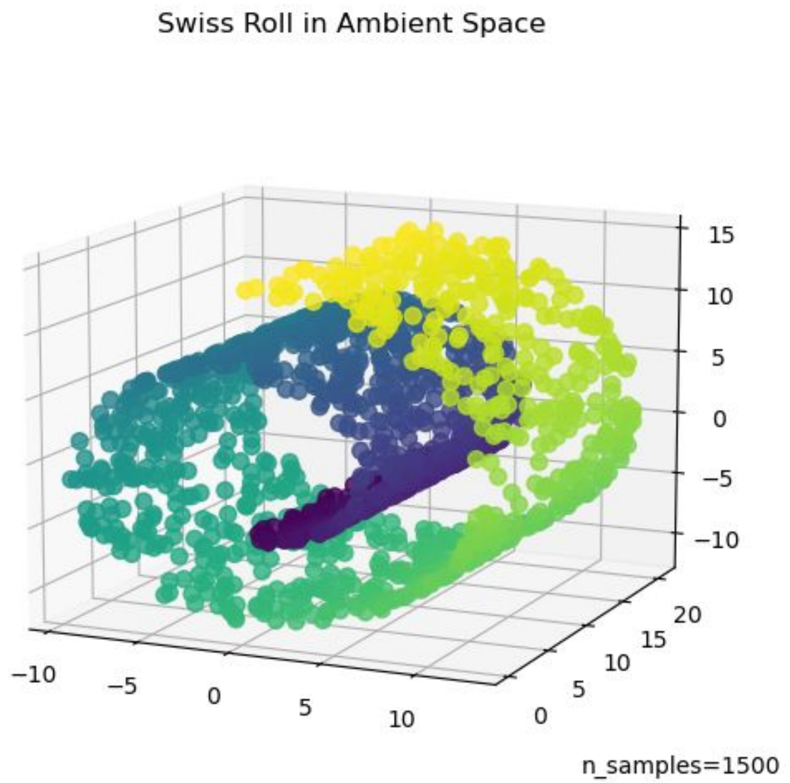


Image source: scikit-learn, “Swiss Roll and Swiss-Hole Reduction”

Nonlinear Methods

1. Sometimes linear methods don't cut it!
   a. Distance no longer **Euclidean**.
2. Lower dimensional **manifold**.
   a. **Embedded** in higher dimensional ambient space.
   b. **Geodesic** distance is the measure.
   c. Usually measured as a **graph walk** or via a **kernel**.
3. Some **caveats**.
   a. Multiple **hyperparameter** choices.
   b. Hard to **interpret** discovered manifold.

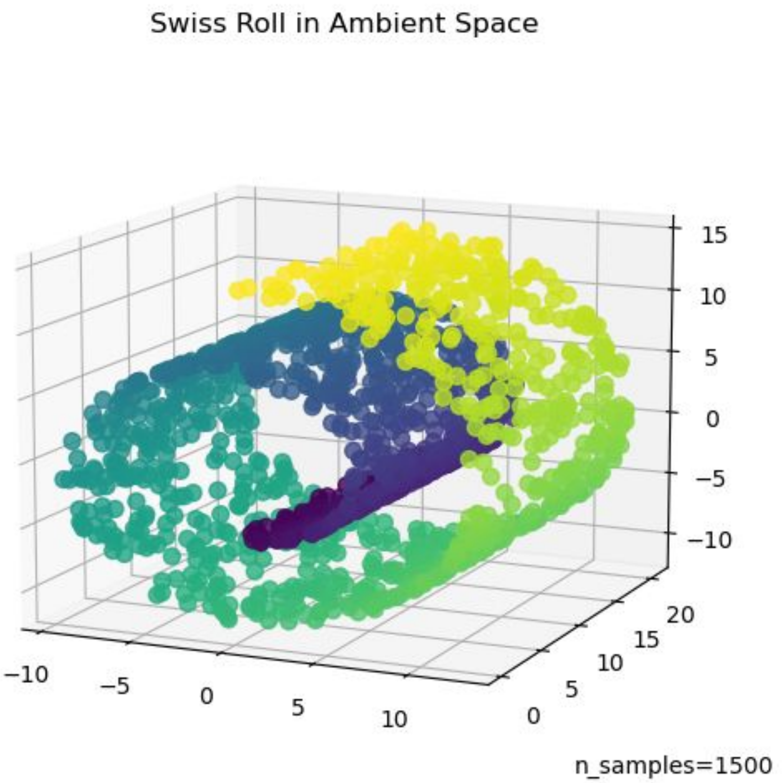


Image source: scikit-learn, "Swiss Roll and Swiss-Hole Reduction"

Nonlinear Methods - Kernel PCA

1. Use the "**kernel trick**" to make things linear.
   a. Assume a **function** f, or kernel , is provided to compute distances.
   b. This corresponds to a Euclidean distance between in a "**lifted feature**" space.

$$\mathbf{K}_{ij} = f(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

2. Now apply **SVD** on this similarity matrix.
   a. Corresponds to **best least-squares approximation** in the lifted space.
   b. Typically, only require a **few** singular vectors.
3. **Tree codes** to compute the Kernel matrix quickly.

Nonlinear Methods - Kernel PCA

1. Use the “**kernel trick**” to make things linear.
   a. Assume a **function** f, or kernel , is provided to compute distances.
   b. This corresponds to a Euclidean distance between in a “**lifted feature**” space.

$$\mathbf{K}_{ij} = f(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

2. Now apply **SVD** on this similarity matrix.
   a. Corresponds to **best least-squares approximation** in the lifted space.
   b. Typically, only require a **few** singular vectors.
3. **Tree codes** to compute the Kernel matrix quickly.

Nonlinear Methods - Kernel PCA

1. Use the "**kernel trick**" to make things linear.
   a. Assume a **function** f, or kernel , is provided to compute distances.
   b. This corresponds to a Euclidean distance between in a "**lifted feature**" space.

$$\mathbf{K}_{ij} = f(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

2. Now apply **SVD** on this similarity matrix.
   a. Corresponds to **best least-squares approximation** in the lifted space.
   b. Typically, only require a **few** singular vectors.
3. **Tree codes** to compute the Kernel matrix quickly.

Nonlinear Methods - Kernel PCA

N-body problem

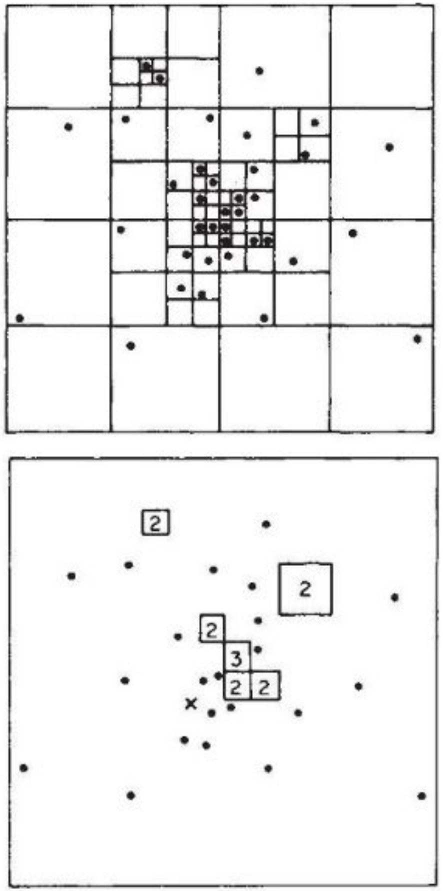
1. Compute the **gravitational interactions** between N bodies.
    a. Naively computes between all **pairs**: $O(N^2)$.
    b. **Approximately** compute as $O(N \log N)$.
2. **Divide** the space for approximation.
    a. **Recursive subdivision** of the space.
    b. **Near** and **far** particles.
    c. Store **total mass** at the **centre-of-mass**.

Barnes and Hut. “A hierarchical O(N log N) force-calculation algorithm” (1986)

Nonlinear Methods - Kernel PCA

N-body problem

1. Compute the **gravitational interactions** between N bodies.
   a. Naively computes between all **pairs**: $O(N^2)$.
   b. **Approximately** compute as $O(N \log N)$.
2. **Divide** the space for approximation.
   a. **Recursive subdivision** of the space.
   b. **Near** and **far** particles.
   c. Store **total mass** at the **centre-of-mass**.



Barnes and Hut. “A hierarchical O(N log N) force-calculation algorithm” (1986)

Nonlinear Methods - Kernel PCA

1. View **kernel matrix** as a similarity graph.
    a. In KPCA all-pairs distances are captured.
2. Need to **sparsify** the kernel matrix.
    a. Prune neighbours in tree construction.
3. Other versions of the similarity graph results in different **embeddings**.
4. **UMAP** similarity graph.
    a. Only store the **"nearest"** d neighbours distances.
    b. Perform a second optimisation for **graph layout**.
    c. Results in **manifolds** where data is uniformly distributed.

Lee, Vuduc, Gray. "A Distributed Kernel Summation Framework for General-Dimensional Machine Learning" (2012)
Curtin. "Improving dual-tree algorithms." (2015)
McInnes, Healy, Melville. "Umap: Uniform manifold approximation and projection for dimension reduction" (2018)

Nonlinear Methods - Kernel PCA

1. View **kernel matrix** as a similarity graph.
    a. In KPCA all-pairs distances are captured.
2. Need to **sparsify** the kernel matrix.
    a. Prune neighbours in tree construction.
3. Other versions of the similarity graph results in different **embeddings**.
4. **UMAP** similarity graph.
    a. Only store the **“nearest”** d neighbours distances.
    b. Perform a second optimisation for **graph layout**.
    c. Results in **manifolds** where data is uniformly distributed.

Lee, Vuduc, Gray. “A Distributed Kernel Summation Framework for General-Dimensional Machine Learning” (2012)
Curtin. “Improving dual-tree algorithms.” (2015)
McInnes, Healy, Melville. “Umap: Uniform manifold approximation and projection for dimension reduction” (2018)

Nonlinear Methods - Kernel PCA

1. View **kernel matrix** as a similarity graph.
   a. In KPCA all-pairs distances are captured.
2. Need to **sparsify** the kernel matrix.
   a. Prune neighbours in tree construction.
3. Other versions of the similarity graph results in different **embeddings**.
4. **UMAP** similarity graph.
   a. Only store the **"nearest"** d neighbours distances.
   b. Perform a second optimisation for **graph layout**.
   c. Results in **manifolds** where data is uniformly distributed.

Lee, Vuduc, Gray. "A Distributed Kernel Summation Framework for General-Dimensional Machine Learning" (2012)
Curtin. "Improving dual-tree algorithms." (2015)
McInnes, Healy, Melville. "Umap: Uniform manifold approximation and projection for dimension reduction" (2018)

Nonlinear Methods - Kernel PCA

1. View **kernel matrix** as a similarity graph.
   a. In KPCA all-pairs distances are captured.
2. Need to **sparsify** the kernel matrix.
   a. Prune neighbours in tree construction.
3. Other versions of the similarity graph results in different **embeddings**.
4. **UMAP** similarity graph.
   a. Only store the **“nearest”** d neighbours distances.
   b. Perform a second optimisation for **graph layout**.
   c. Results in **manifolds** where data is uniformly distributed.

Lee, Vuduc, Gray. “A Distributed Kernel Summation Framework for General-Dimensional Machine Learning” (2012)
Curtin. “Improving dual-tree algorithms.” (2015)
McInnes, Healy, Melville. “Umap: Uniform manifold approximation and projection for dimension reduction” (2018)

Nonlinear Methods - Autoencoders

1. Remove the assumptions of **specifying** out a kernel.
    a. **Learn** it from the data!
2. Enter the **autoencoder**.
    a. Many different **flavours**: fully connected, convolutional, variational, …
3. Convert **convolutions** to **xGEMM**.
    a. Need **multiple** of these small matrix multiplies.
    b. **Batched** xGEMM.
4. Kernel **fusion**.
    a. Fuse all **elementwise** operations (e.g. ReLU, sigmoid, …).

Nonlinear Methods - Autoencoders

1. Remove the assumptions of **specifying** out a kernel.
   a. **Learn** it from the data!
2. Enter the **autoencoder**.
   a. Many different **flavours**: fully connected, convolutional, variational, …
3. Convert **convolutions** to **xGEMM**.
   a. Need **multiple** of these small matrix multiplies.
   b. **Batched** xGEMM.
4. Kernel **fusion**.
   a. Fuse all **elementwise** operations (e.g. ReLU, sigmoid, …).

Nonlinear Methods - Autoencoders

Convolutional Autoencoder

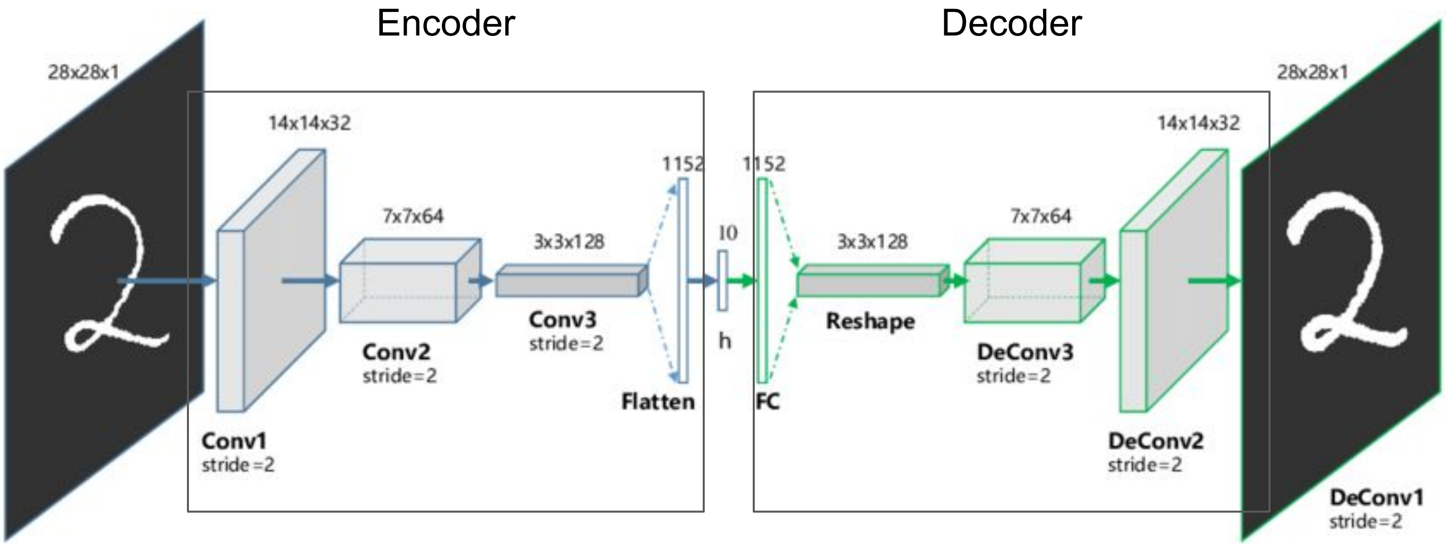


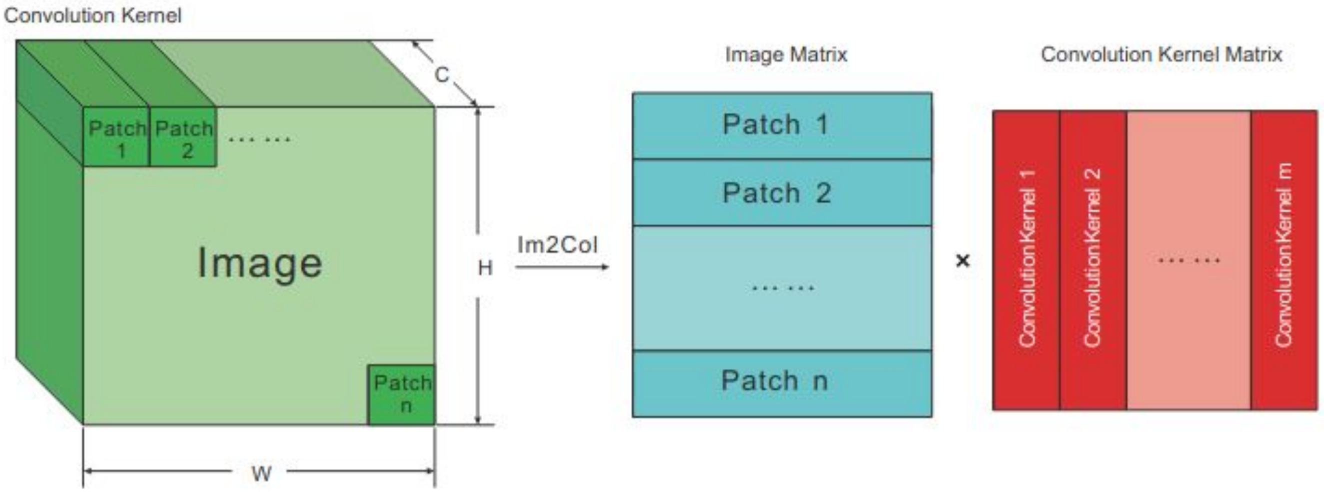
Image source: analyticsindiamag, “How to implement Convolutional Autoencoder in PyTorch with CUDA”

Nonlinear Methods - Autoencoders

1. Remove the assumptions of **specifying** out a kernel.
   a. **Learn** it from the data!
2. Enter the **autoencoder**.
   a. Many different **flavours**: fully connected, convolutional, variational, …
3. Convert **convolutions** to **xGEMM**.
   a. Need **multiple** of these small matrix multiplies.
   b. **Batched** xGEMM.
4. Kernel **fusion**.
   a. Fuse all **elementwise** operations (e.g. ReLU, sigmoid, …).

Nonlinear Methods - Autoencoders

Convolutions as xGEMM



$$\mathbf{C}^{(i)} \leftarrow \alpha^{(i)} \mathbf{A}^{(i)} \mathbf{B}^{(i)} + \beta^{(i)} \mathbf{C}^{(i)}$$

Dongarra et al.. “Optimised Batched Linear Algebra for Modern Architectures” (2017)
Yang, Lu, Wang. “A batched GEMM optimisation framework for deep learning” (2022)

Nonlinear Methods - Autoencoders

1. Remove the assumptions of **specifying** out a kernel.
   a. **Learn** it from the data!
2. Enter the **autoencoder**.
   a. Many different **flavours**: fully connected, convolutional, variational, …
3. Convert **convolutions** to **xGEMM**.
   a. Need **multiple** of these small matrix multiplies.
   b. **Batched** xGEMM.
4. Kernel **fusion**.
   a. Fuse all **elementwise** operations (e.g. ReLU, sigmoid, …).