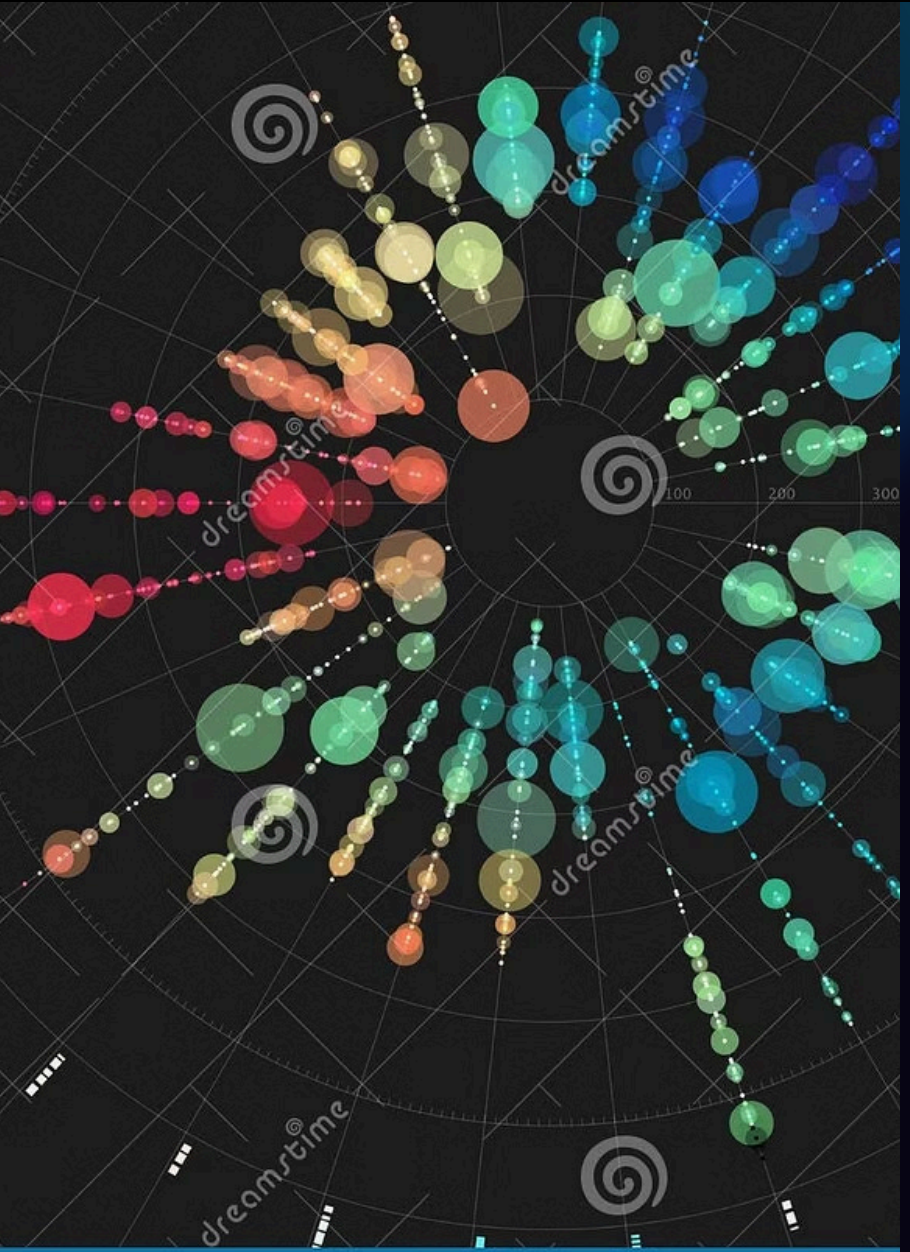
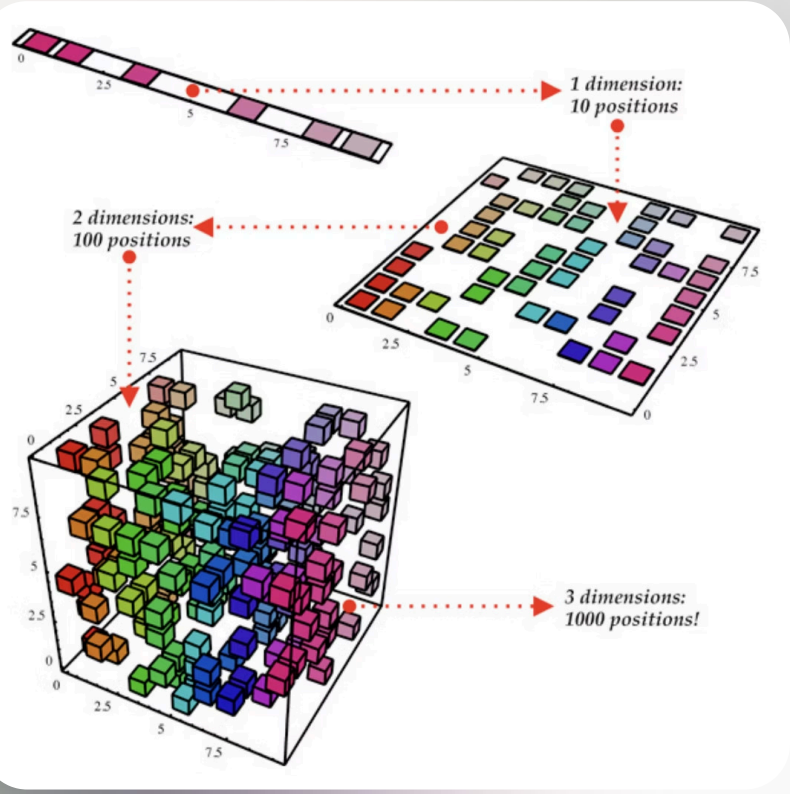


Dimensionality Reduction: PCA, SVD, t-SNE, UMAP, and Autoencoders

Data Science & Machine Learning for Physics

Hernan Andres Morales-Navarrete



The challenge of high-dimensional data

Physics data reality

Scientific data is inherently high-dimensional : images, time series, and simulations create massive datasets that challenge traditional analysis methods.

Curse of dimensionality

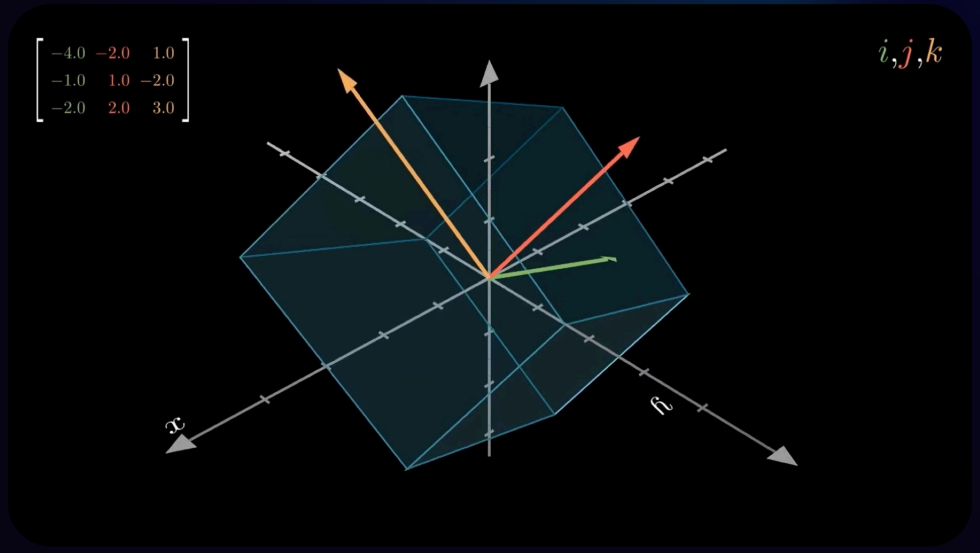
As dimensions increase, distances lose meaning and computations explode exponentially, making analysis nearly impossible.

The solution (?)

Dimensionality reduction compresses data while preserving essential structure, enabling visualization, noise reduction, and feature extraction.

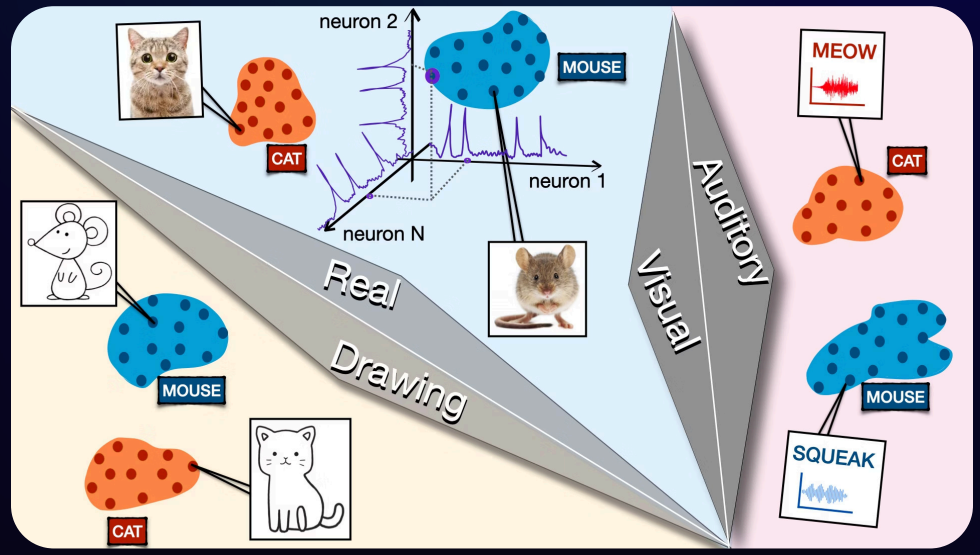
Two Fundamental Approaches

Linear Methods



- Principal Component Analysis (PCA)
- Singular Value Decomposition (SVD)
- Fast and interpretable
- Captures linear relationships

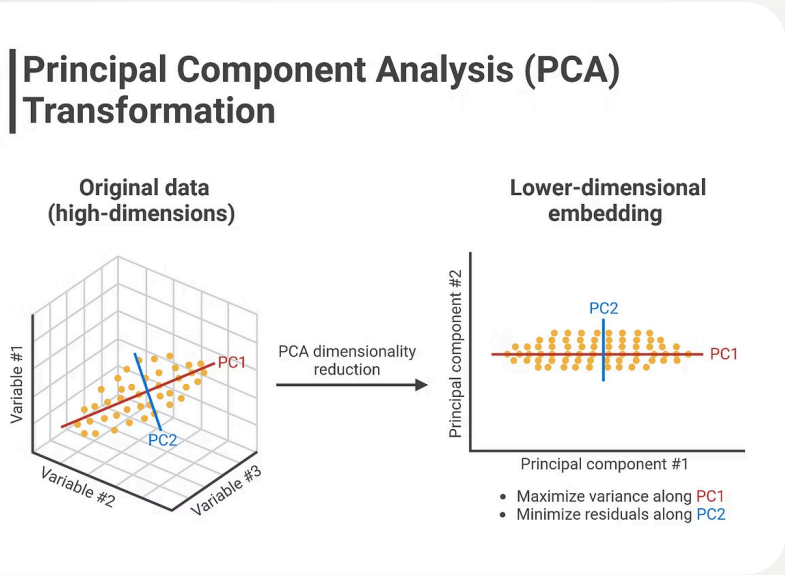
Non-linear Methods



- t-distributed Stochastic Neighbor Embedding (t-SNE)
- Uniform Manifold Approximation (UMAP)
- Autoencoders
- Captures complex patterns

PCA: Finding the best viewpoint

Principal Component Analysis finds new orthogonal axes (principal components) that capture maximum variance in your data. Think of it as finding the best angle to photograph a 3D object on a 2D surface.



01

Identify direction of maximum variance

Find the axis along which data points are most spread out

02

Find orthogonal components

Identify perpendicular directions with decreasing variance

03

Project and reduce

Transform data onto first few principal components

Physics connection: PCA is analogous to diagonalizing covariance matrices, similar to finding eigenmodes in physical systems.

The mathematics behind PCA

PCA transforms the mathematical problem of dimensionality reduction into an eigenvalue decomposition:

1

Center the Data

$$X_c = X - \mu$$

Remove the mean to focus on variance patterns

2

Compute Covariance Matrix

$$C = \frac{1}{n} X_c^T X_c$$

Measure how variables change together

3

Eigen-decomposition

$$Cv = \lambda v$$

Principal components = eigenvectors, variances = eigenvalues

Retain the top-k components for dimensionality reduction while preserving maximum information.

PCA in action: Digits dataset

PCA excels at compressing particle trajectories and images by finding collective behavior patterns, similar to eigenmodes in physical systems.

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

digits = load_digits()
X = digits.data
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=digits.target, cmap="tab10")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA on Digits Dataset")
plt.show()
```

This example reduces 64-dimensional digit images to just 2 dimensions while preserving class structure.

SVD: The universal factorization

Singular Value Decomposition provides a general matrix factorization that works beyond the constraints of PCA:

$$X = U\Sigma V^T$$

SVD is closely related to PCA when applied to centered data, but offers greater flexibility for any rectangular matrix. In physics, it's invaluable for image compression and solving ill-posed inverse problems.

```
import numpy as np
from sklearn.datasets import load_sample_image

image = load_sample_image("china.jpg")
gray = np.mean(image, axis=2) / 255.0
U, S, Vt = np.linalg.svd(gray, full_matrices=False)

k = 50  # keep top-k components
compressed = U[:, :k] @ np.diag(S[:k]) @ Vt[:k, :]
```

PCA vs SVD: Understanding the connection

PCA approach

Uses eigen-decomposition of the covariance matrix.

Specifically designed for variance-based dimensionality reduction of centered data.

SVD approach

More general factorization that works for any rectangular matrix.

Can handle non-centered data and various matrix operations.

Practical reality

Scikit-learn's PCA implementation actually uses SVD under the hood for numerical stability and computational efficiency.

Why linear methods aren't enough

Beyond Linearity

PCA and SVD excel at capturing linear relationships, but many physics datasets exist on complex nonlinear manifolds that require more sophisticated approaches.

Protein Folding

Protein conformations lie on curved manifolds in high-dimensional space, requiring nonlinear methods to capture folding pathways.

Chaotic Attractors

Strange attractors in dynamical systems exhibit complex, nonlinear geometric structures that linear methods cannot capture.

Curved latent spaces

Many physical phenomena naturally exist in curved spaces that require nonlinear dimensionality reduction techniques.

t-SNE: Preserving Local Neighborhoods

t-distributed Stochastic Neighbor Embedding focuses on preserving local neighborhood relationships through probability-based optimization.

How t-SNE Works

- Converts distances to probabilities
- Similar points get high probability
- Minimizes Kullback–Leibler (KL) divergence between high and low dimensions
- Excellent for 2D/3D visualization

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=42)
X_tsne = tsne.fit_transform(X)

plt.scatter(X_tsne[:, 0], X_tsne[:, 1],
            c=digits.target, cmap="tab10")
plt.title("t-SNE on Digits Dataset")
plt.show()
```

Physics Applications

Particularly useful for classifying particle detector signals and visualizing complex experimental data clusters.

UMAP: The best of both worlds

Uniform Manifold Approximation and Projection offers similar goals to t-SNE but with significant improvements in both local and global structure preservation.

Balanced Approach

Preserves both local neighborhoods and global structure, providing more interpretable embeddings than t-SNE.

Computational Efficiency

Significantly faster and more scalable than t-SNE, making it suitable for larger datasets.

Mathematical Foundation

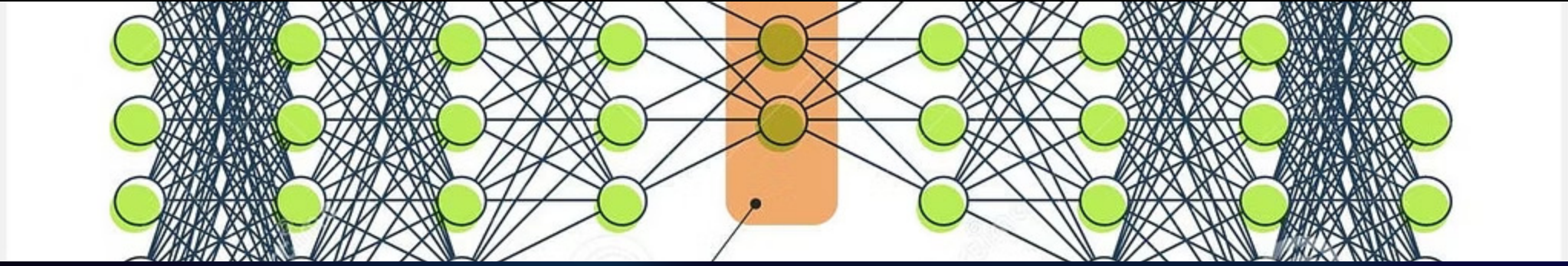
Based on Riemannian geometry and topological graph theory, providing solid theoretical grounding.

```
import umap

umap_model = umap.UMAP(n_components=2, random_state=42)
X_umap = umap_model.fit_transform(X)

plt.scatter(X_umap[:, 0], X_umap[:, 1], c=digits.target, cmap="tab10")
plt.title("UMAP on Digits Dataset")
plt.show()
```

t-SNE vs UMAP: Choosing your tool

Aspect	t-SNE	UMAP
Local Structure	Excellent clustering	Good clustering
Global Structure	Poor distance preservation	Better global relationships
Speed	Slower, less scalable	Faster, more scalable
Interpretability	Visualization focused	More interpretable embeddings
Determinism	Non-deterministic	Non-deterministic
Hyperparameters	Sensitive to perplexity	Sensitive to n_neighbors

Both methods are sensitive to hyperparameter choices and produce different results on each run, requiring careful validation and multiple runs for robust analysis.



Autoencoders: Neural Network approach

Autoencoders represent a fundamentally different approach to dimensionality reduction using neural networks trained to reconstruct their input.

1

Encoder

Compresses input data into a lower-dimensional latent representation

2

Latent Space

Learned nonlinear reduced representation capturing essential features

3

Decoder

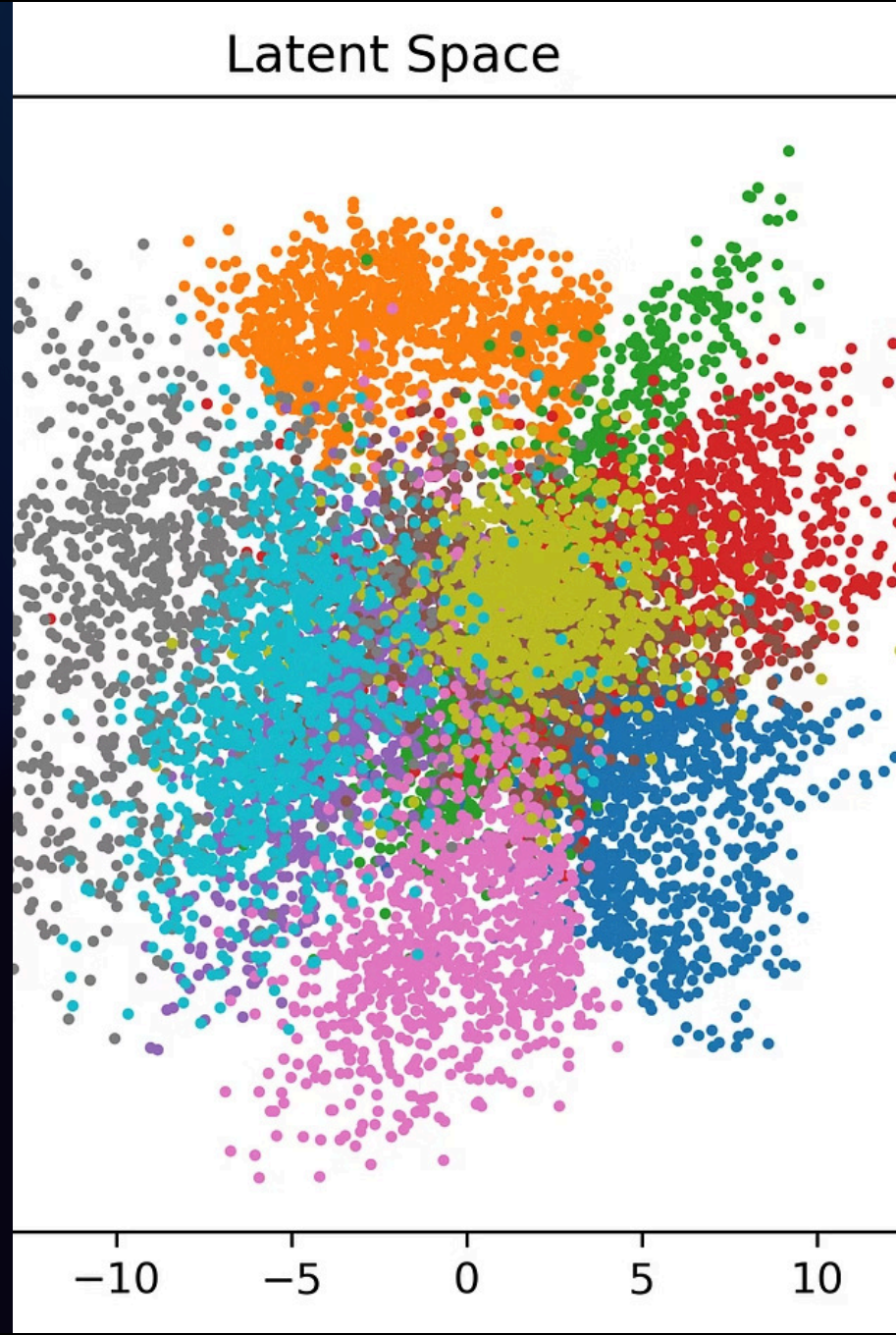
Reconstructs original data from the compressed latent representation

Physics Analogy: The latent space represents effective degrees of freedom in your system, similar to how physical systems can be described by a smaller number of key variables.

Building an Autoencoder

Here's how to implement an autoencoder for the digits dataset, creating a 2-dimensional latent space representation:

```
from tensorflow.keras import layers, models

input_dim = X.shape[1]

encoder = models.Sequential([
 layers.Input(shape=(input_dim,)),
 layers.Dense(64, activation="relu"),
 layers.Dense(2) # latent space
])

decoder = models.Sequential([
 layers.Input(shape=(2,)),
 layers.Dense(64, activation="relu"),
 layers.Dense(input_dim, activation="sigmoid")
])

autoencoder = models.Sequential([encoder, decoder])
autoencoder.compile(optimizer="adam", loss="mse")
autoencoder.fit(X, X, epochs=20, batch_size=256, verbose=0)

X_latent = encoder.predict(X)
plt.scatter(X_latent[:, 0], X_latent[:, 1], c=digits.target, cmap="tab10")
plt.title("Autoencoder Latent Space")
plt.show()
```



Method Comparison Matrix

Method	Type	Pros	Cons
PCA	Linear	Simple, fast, interpretable	Only captures linear variance
SVD	Linear	General, robust factorization	Requires interpretation
t-SNE	Nonlinear	Excellent visualization	Only 2D/3D, not scalable
UMAP	Nonlinear	Preserves local + global, scalable	Sensitive to parameters
Autoencoders	Nonlinear	Powerful, flexible architecture	Needs training, black-box

Choose your method based on your specific needs: interpretability (PCA/SVD), visualization (t-SNE/UMAP), or flexibility (Autoencoders).

Implementation Toolkit

Scikit-learn

Comprehensive library providing PCA, SVD, t-SNE implementations with consistent APIs and excellent documentation.

UMAP-learn

Specialized library for advanced UMAP implementations with extensive customization options and optimization features.

PyTorch / TensorFlow

Deep learning frameworks for building sophisticated autoencoders with custom architectures and training procedures.

Recommended datasets for learning

- **MNIST digits:** Intuitive visualization and easy interpretation
- **Physics datasets:** Microscopy images or synthetic PDE solutions

Physics case studies

Lattice Vibrations

Use PCA to analyze normal modes in crystal lattice vibrations, revealing fundamental vibrational patterns and phonon structures.

Particle Detection

Apply t-SNE and UMAP for clustering particle detector signals, identifying different particle types and collision signatures.

Biological Imaging

Use autoencoders to reduce dimensionality of zebrafish embryo microscopy images, extracting developmental features and patterns.

Critical thinking questions

Physical Intuition

What physical analogy helps you understand PCA better? Consider eigenmodes, resonance, or wave interference patterns.

Method Selection

When would you choose t-SNE over UMAP? Consider your data size, visualization needs, and computational constraints.

Fundamental Differences

How does an autoencoder differ from PCA? Think about linearity, interpretability, and computational requirements.

Hidden Variables

Can latent dimensions reveal "hidden" physical variables? Consider what the reduced dimensions might represent in your system.

Key Takeaways

Mastering High-Dimensional Physics Data

Essential Tool

Dimensionality reduction is crucial for analyzing complex physical systems and extracting meaningful patterns from high-dimensional data.

Linear Foundation

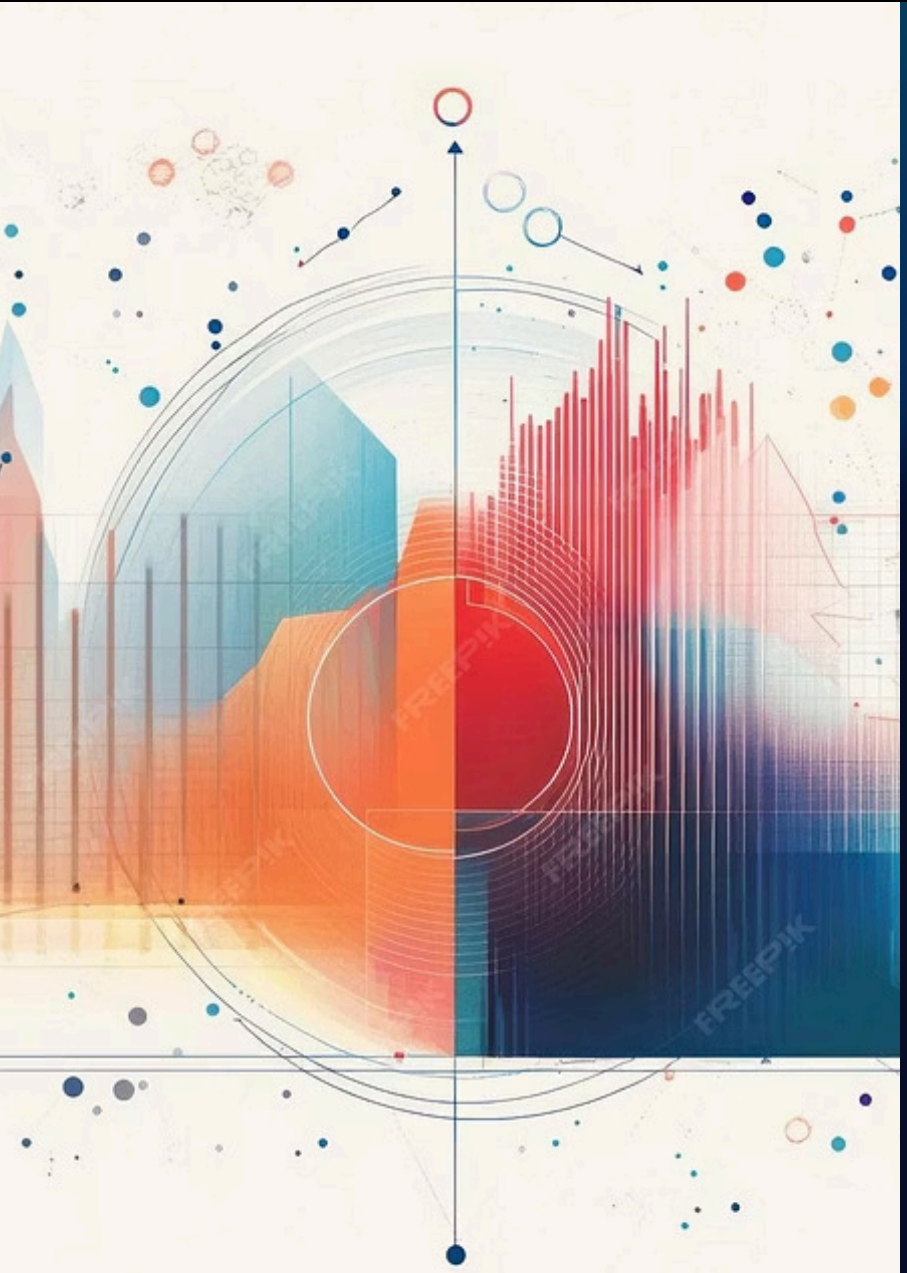
PCA and SVD provide interpretable, fast solutions for linear relationships and serve as the foundation for understanding more complex methods.

Nonlinear Power

t-SNE and UMAP excel at nonlinear visualization, while autoencoders offer flexible, deep learning-based approaches for complex patterns.

Structure Discovery

All methods help us find hidden structure in high-dimensional data, revealing the underlying physics and enabling new insights.



Thank You

Questions & Discussion

Data Science & Machine Learning for Physics

Dr. Hernán A. Morales-Navarrete