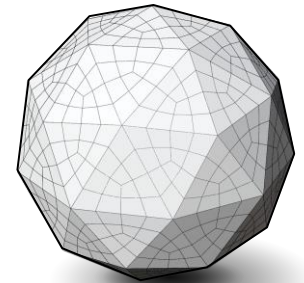
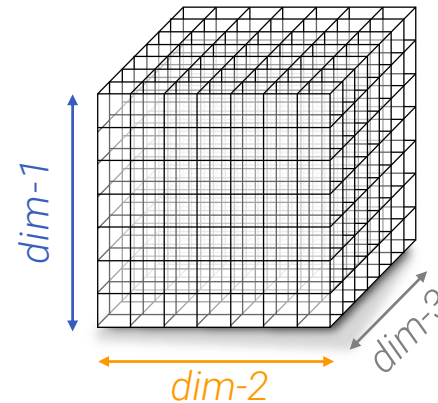
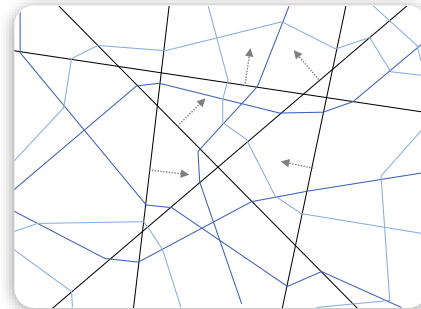
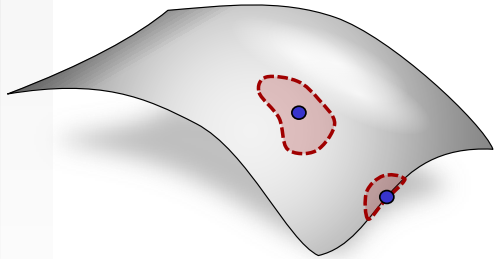


Modelling 2

STATISTICAL DATA MODELLING



Chapter 10 Space

Video #10

Space

- **High-dimensional space**
& the curse of dimensionality
- **Kernels:** flat space extended
- **Manifolds:** curved space

The Curse of Dimensionality

Issues with high-dimensional data

- **Structural anomalies – too much space**
 - Distance concentration
 - Naïve dimensionality reduction
 - The Johnson-Lindenstrauss Lemma
- **Generalization problems – too little data**
 - Sampling requirements
 - Curved space
- **Computational problems – too much work**
 - Searching
 - Integration

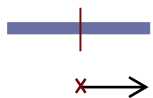
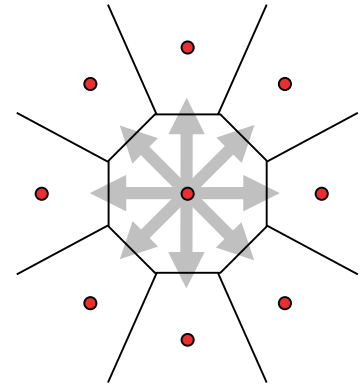
– **too much space** in high dimensions –

Structural / Logical Anomalies

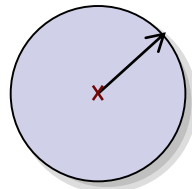
Higher Dimensions are Weird

Issues with High-Dimensional Spaces :

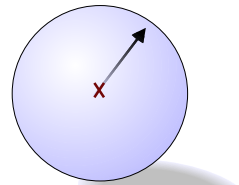
- d -dimensional space:
 d independent neighboring directions to each point
- Volume-distance ratio explodes



$d = 1$



$d = 2$

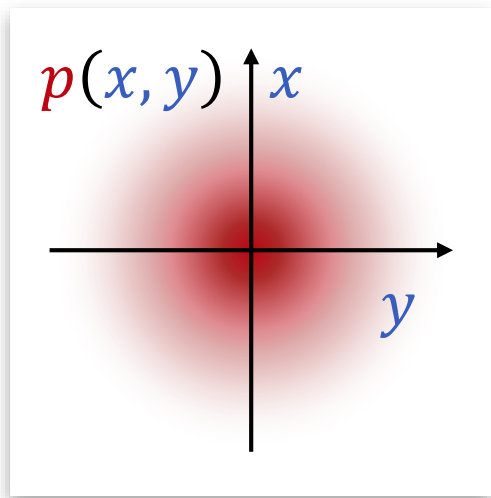
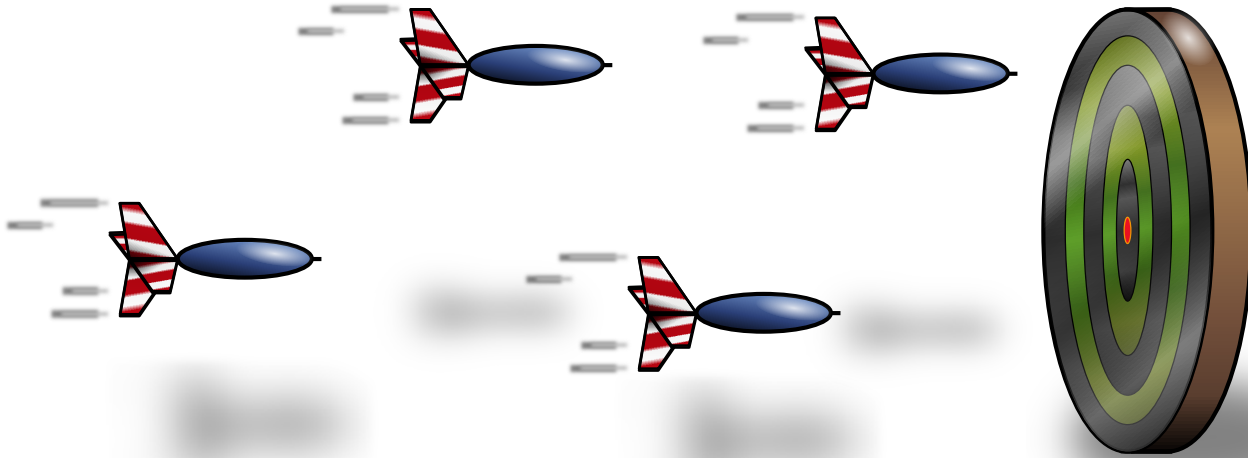


$d = 3$

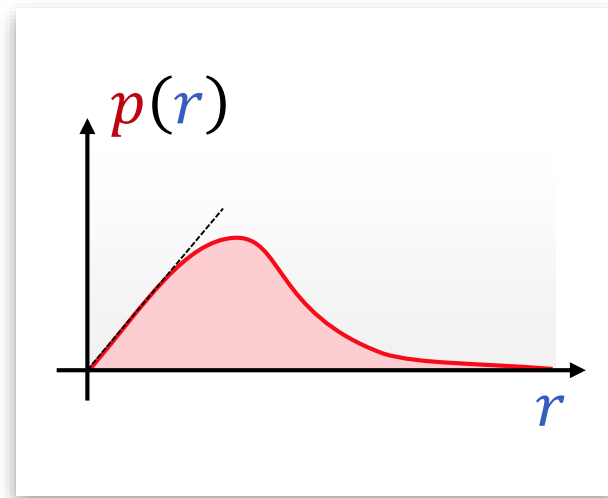
$$\text{vol}(r) \in \Theta(r^d)$$

$d \rightarrow \infty$

Dart Throwing

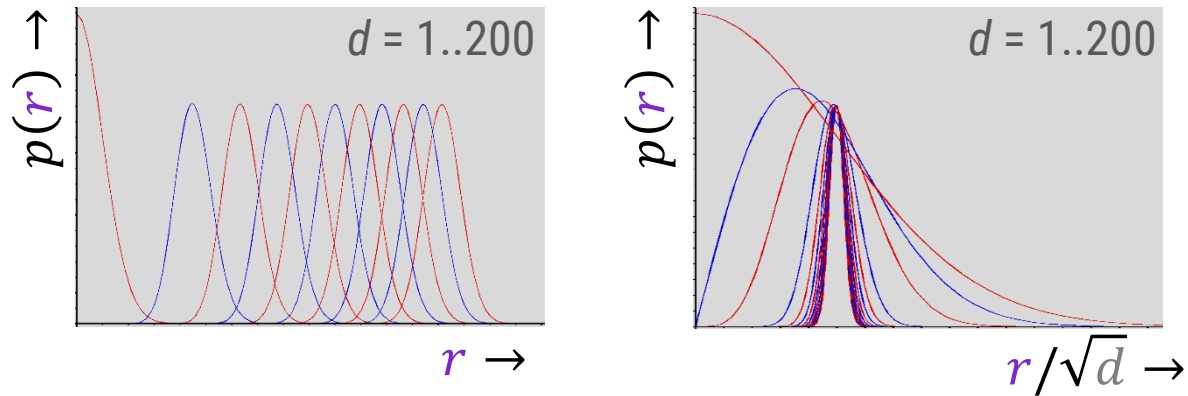


normal distribution



by radius: bulls-eye unlikely

Higher Dimensions are Weird



Concentration of distances

- “Dart-throwing anomaly”

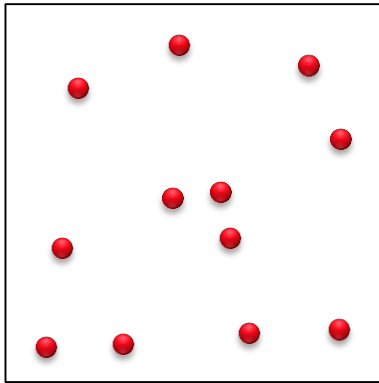
- Normal distributions
- Gather probability-mass in thin shells

$$p(r) \sim r^{d-1} e^{-r^2} \quad (\text{maximum in the limit: } \sqrt{d})$$

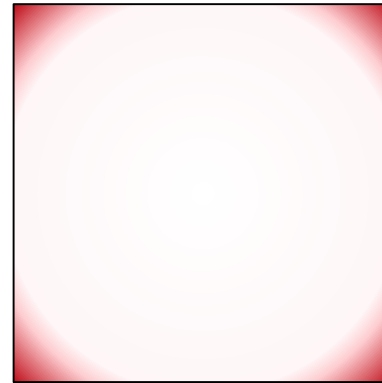
- Nearest neighbor \approx farthest neighbor

- For unstructured points (e.g. iid-random)
- Not true for if data is structured specifically

Heavy Corners



looks benign in 2D



all samples near corners
in high-dim.

Why do we always sample Gaussians?

- Uniform random variables on a cube
- Corners have most of the volume (growth r^d)
- Need symmetric shapes
 - Gaussian is rotationally symmetric (and is separable)
 - Sphere would also work

Dimensionality Reduction

Can we reduce dimensions?

- Assume point set

$$P = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subseteq \mathbb{R}^d$$

- Let's say, we only care about pairwise distances

$$|\mathbf{x}_i - \mathbf{x}_j|, \quad i, j \in \{1, \dots, n\}$$

- Example application: classifier
 - (general discriminative tasks)

Dimensionality Reduction

“Trivial” result

- Embedding n points in $d = n - 1$ dimensions
- Only interesting if $d > n$
 - Just use differences $\mathbf{x}_i - \mathbf{x}_1$ as coordinate vectors
 - Then run Gram-Schmidt-orthogonalization to get orthogonal coordinate frame

Johnsen-Lindenstrauss Lemma

- Good approximate embedding in $d \in O(\log n)$
 - Guaranteed quality for any point set
- A bit more surprising

Johnson-Lindenstrauss Lemma

JL-Lemma: [Dasgupta & Gupta 2003]

- Point set $P = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ in \mathbb{R}^d
- There exists $f: \mathbb{R}^d \rightarrow \mathbb{R}^k$ with $k \in \mathcal{O}(\epsilon^{-2} \ln n)$
($k \geq 4(\epsilon^2/2 - \epsilon^3/3)^{-1} \ln n$)
- ...that preserves all inter-point distances up to a factor of $(1 + \epsilon)$

Random orthogonal linear projection

- Works with probability $\geq (1 - 1/n)$

This means...

What Does the JL-Lemma Imply?

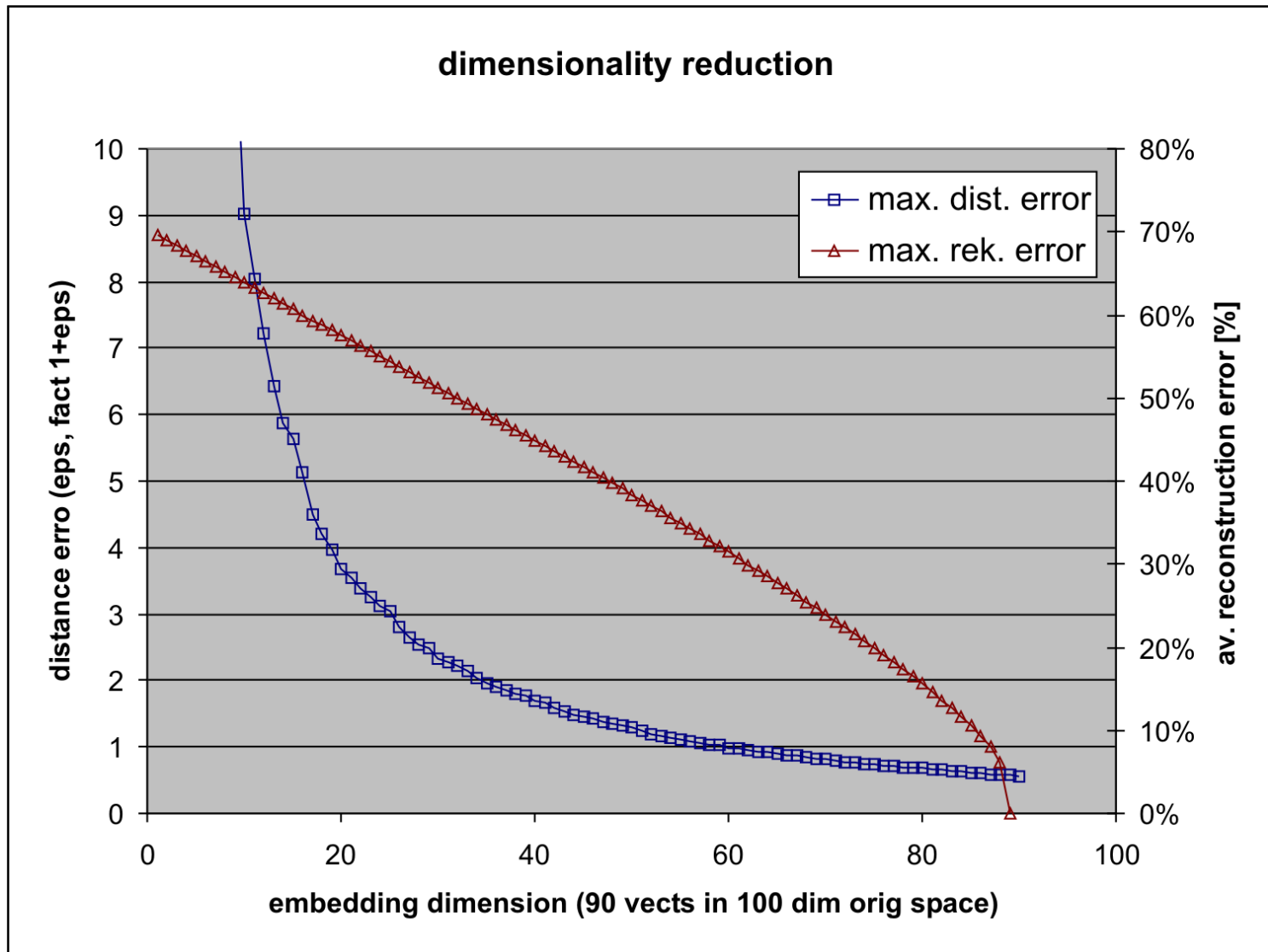
Pairwise distances in small point set P
(sub-exponential in d)

can be well-preserved in low-dimensional embedding

What does it not say?

Does not imply that the points *themselves* are well-represented (just the pairwise distances)

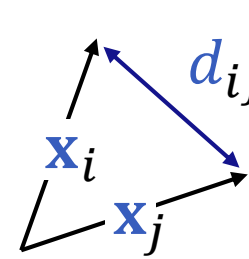
Experiment

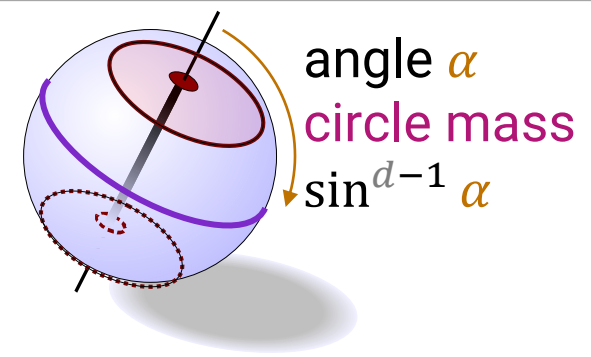
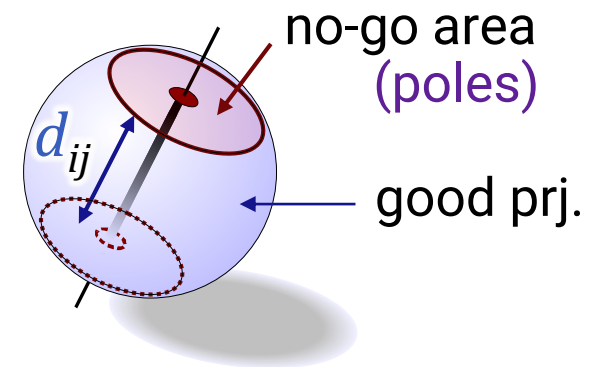
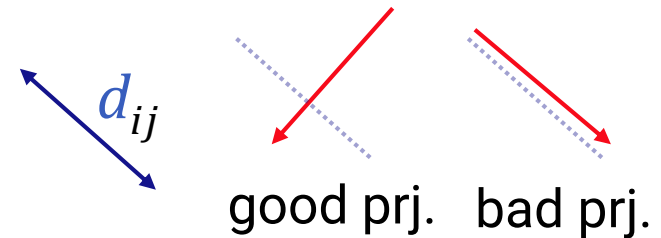


Proof Sketch

Difference Vectors

- Normalize (relative error)
- All n^2 pairs yield *poles*
- Pole yields bad approximation
 - n^2 poles d_{ij}
- Non-pole area much larger
 - High dimension
 - Volume grows with:
 $\sin^{d-1} \alpha$
- Covering sphere with poles
 - Need large number of poles
 - Exponential in d


$$d_{ij} = \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|} - \frac{\mathbf{x}_j}{\|\mathbf{x}_j\|}$$



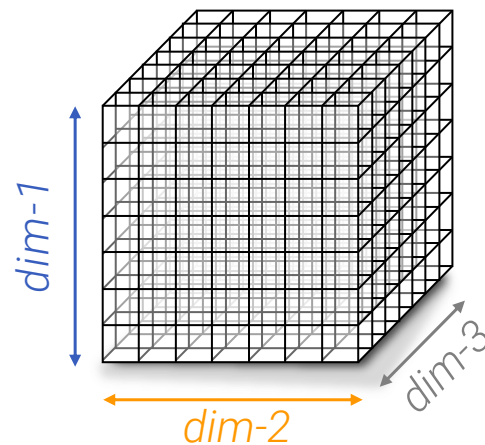
– **too little data** for high dimensions –

Generalization Problems

Sampling Requirements

Sampling costs grow exponentially with d

- Sampling a unit cube in \mathbb{R}^d
- Spacing $\epsilon \rightarrow n = 1/\epsilon$ samples
- Costs $\mathcal{O}(n^d)$



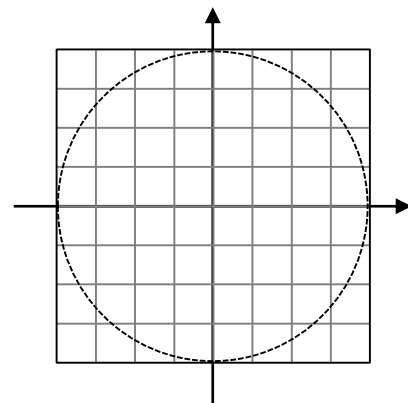
Sampling theory

- Resolve frequencies $\omega = 1/\epsilon$
- Tensor-product Fourier basis

Rect.: $\{e^{i(\omega_1 x_1 + \dots + \omega_d x_d)} \mid \omega_1, \dots, \omega_d = -n \dots n\}$

Isotropic: $\omega_1^2 + \dots + \omega_d^2 \leq n^2$

- Exponential costs



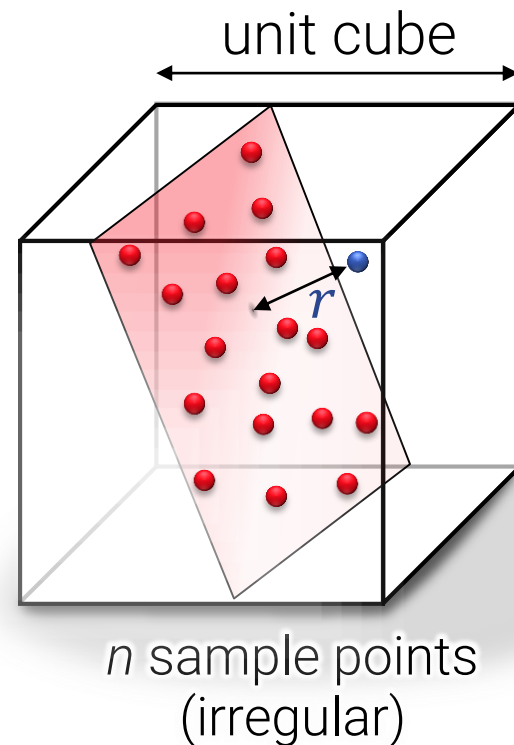
Subspace Sampling

Sampled Hyperplane

- Space has dimension d
- Hyperplane has dimension $k < d$

Discriminative Task

- Neighborhood based classifier
- Blue sample r away from plane
- Nearest neighbor on plane should be closer than blue sample
- Need $\Omega(n^k)$, $n = r^{-1}$ samples – exponential
- Need $\Omega(n^k \log n^k) = \Omega(kn^k \log n)$ random samples



Random Samples?

Coupon-Collectors Theorem

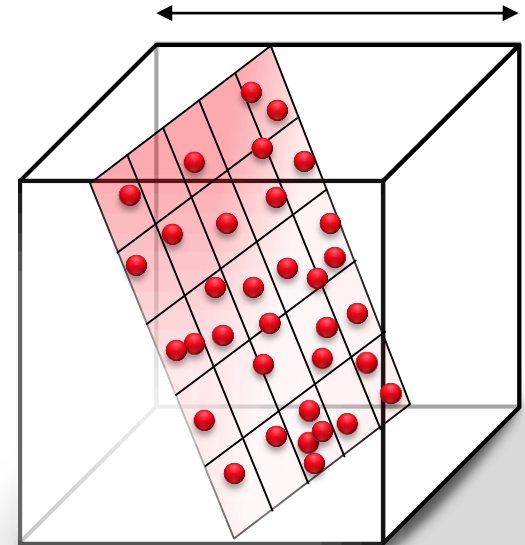
- On expectation, we need

$$n H_n \\ = (\ln n) \pm 1$$

random draws to hit n bins /
coupons

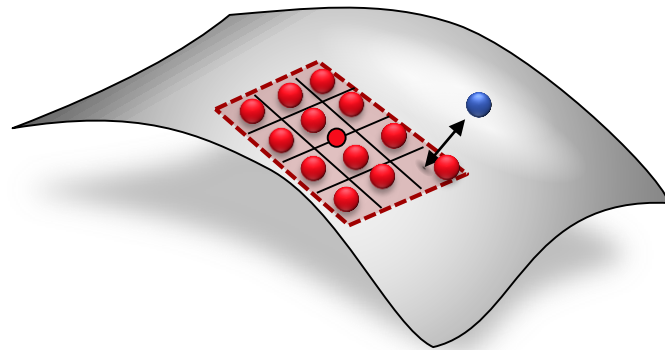
- Thus, random i.i.d. uniform
samples increase effort

$$O(n) \rightarrow O(n \ln n)$$



n bins $\rightarrow n \log n$
sample points
(irregular)

Learning Manifolds



(Common) Assumption

- Data of a class forms a smooth k -dimensional surface (“ k -manifold”) in d -dimensional space
- Model: Local flat approximation
- Again, costs are exponential in k

Consequence

Hard to learn

- Data manifolds with intrinsically high dimension
- Common – think of all the poses of a dog

Distance-based classifiers...

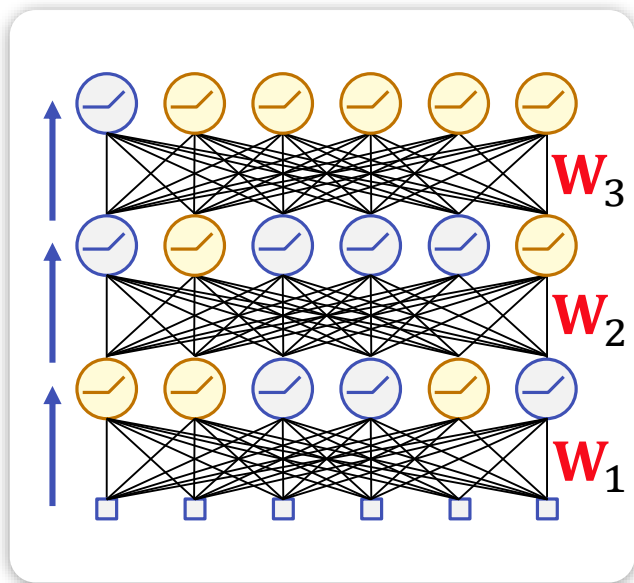
- ...will have exponential sampling cost
- I.e., need exponential amount of training data!

Smoothness is distance based [Bengio]

- Nearest-Neighbors, Histograms, Parzen Windows
- Gaussian-Kernel-SVM, Gaussian processes

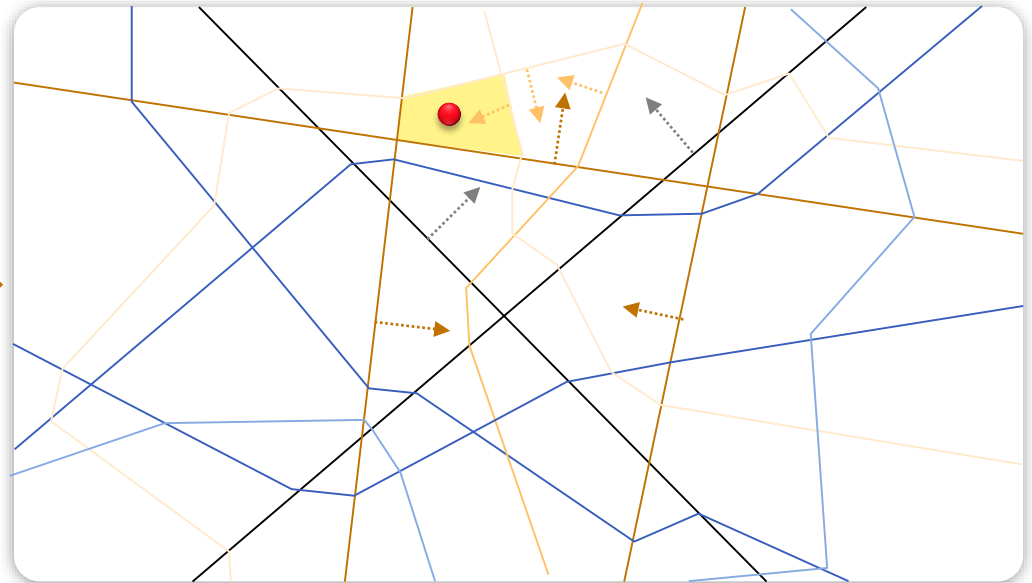
DNNs Can Learn combinatorially

three network layers



Interpretation

Nested ReLU-layer = nested convex cells



Activation Patterns

Encode combinatorial decisions
(which linear map to use)

Example: ReLU

Activations of a ReLU Neuron

- Binary weights - work as “or”-operator
- Negative weights act as “not”-operator
- Can build “NOR”-gates

NOR-gates are universal

- Can encode arbitrary logically functions with a network of NOR-gates
- Depth make it efficient
 - Shallow circuits might have exponential disadvantage

Impossible with distance-based methods

– **too much work** in high dimensions –

Computational Issues

– too much work in high dimensions –

Computational Issues

SEARCH

Search

Given

- Point cloud $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$
- Query point $\mathbf{x} \in \mathbb{R}^d$

How to efficiently find

- (k -)nearest-neighbors of \mathbf{x}
- Neighbors in fixed radius r from \mathbf{x}

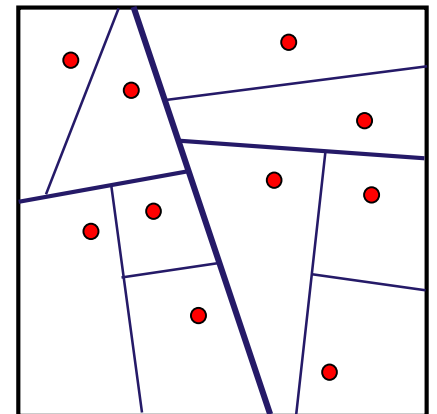
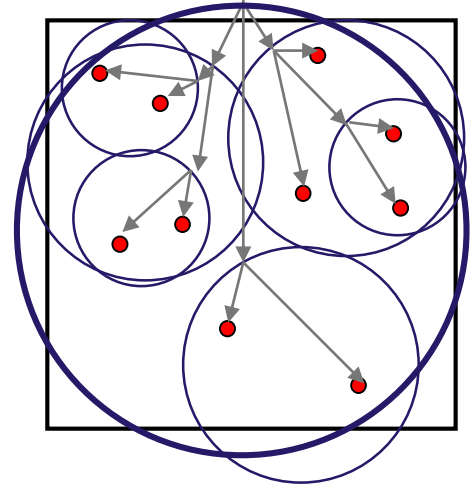
Example applications

- k-NN Classifier (old-school)
 - Using Siamese Networks (*new-school*)
- } Dimension d is large

Data Structures

Search Data structures

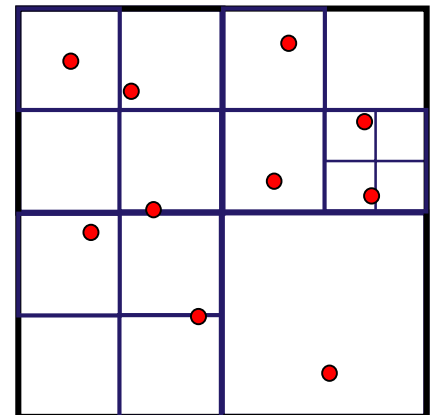
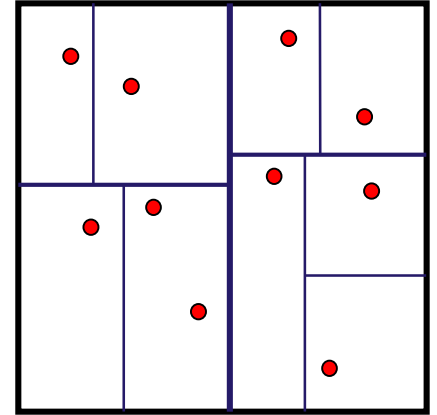
- **Bounding volume hierarchy**
 - Hierarchical grouping of points
 - Bounding volumes (e.g. spheres)
 - Generic idea – many variants
- **BSP-tree** (“binary space partition tree”)
 - Split by planes
 - (Usually) binary tree
 - Complex, convex cells as bounding volumes
 - Half-space test per node



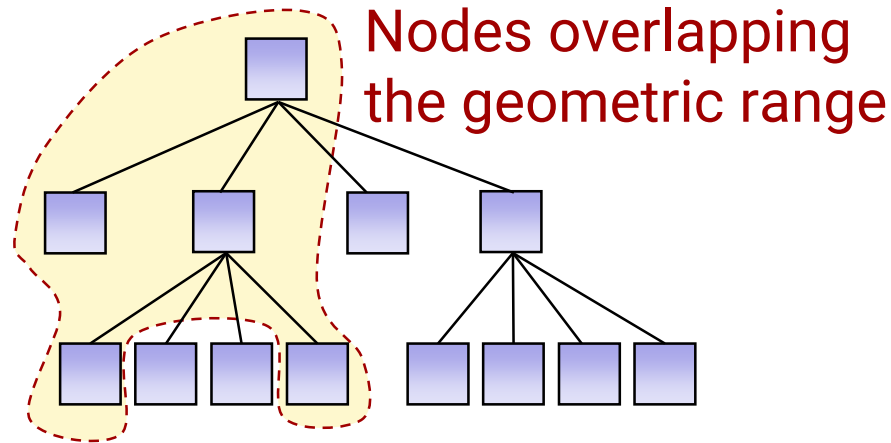
Variants

Variants

- **k-D-tree** (axis aligned BSP-tree)
 - Use axis parallel splitting planes
 - Cyclically alternate splitting dimension
 - Median cut
- **Quadtrees / Octrees**
 - Divide into 4 (8) congruent cubes
 - Costs exponential with dimension
 - Practically used only in \mathbb{R}^2 , \mathbb{R}^3



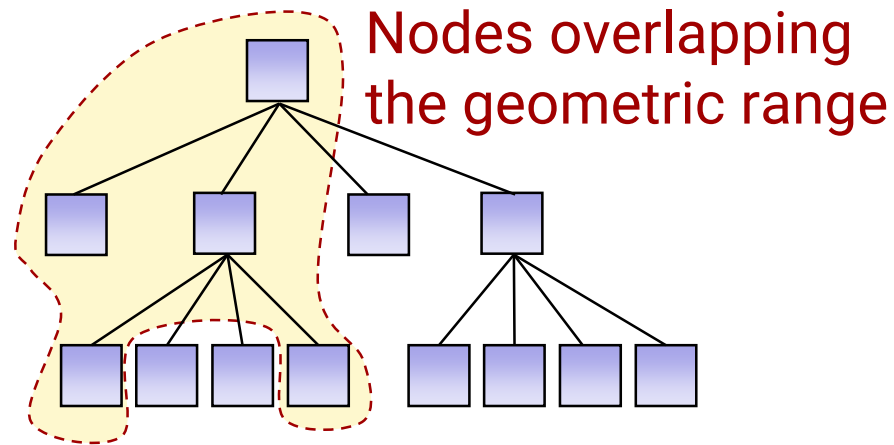
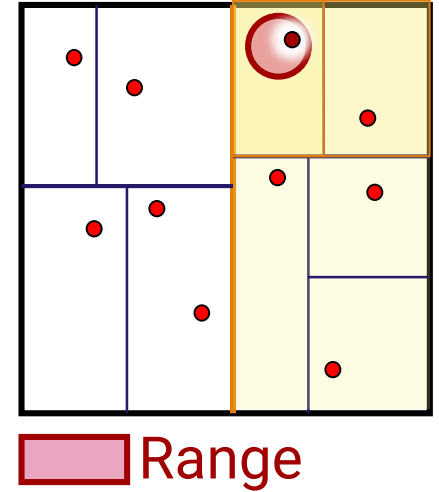
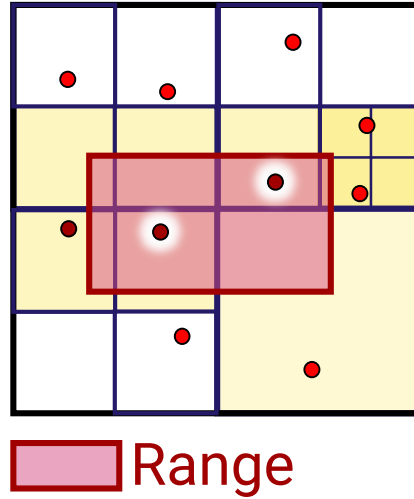
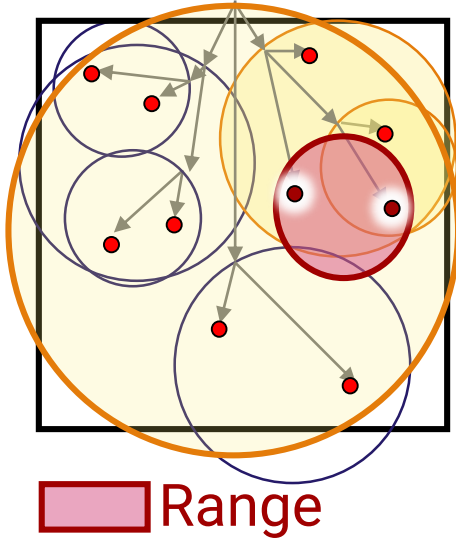
Range Query Algorithm



Recursively from root node

- If range overlaps bounding box
 - Collect points in node (if any)
Keep those in range
 - Recursion for child nodes
- If range does not overlap bounding box
 - Return empty

Examples



Nearest-Neighbor Query Algorithm

Algorithm: k nearest neighbors

Data structure: queue sorted by distance

Initialization: Put root node in queue

While not yet k points found **and** queue non-empty:

 Take closest object from queue

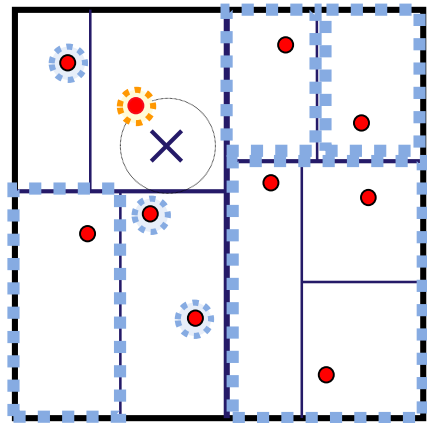
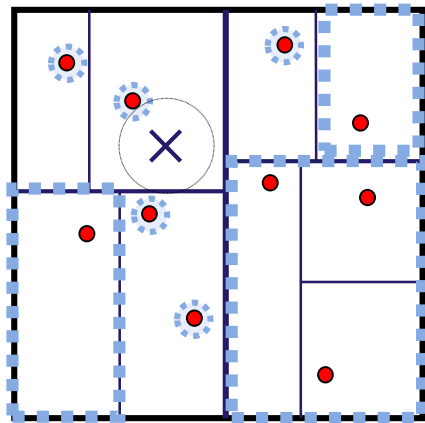
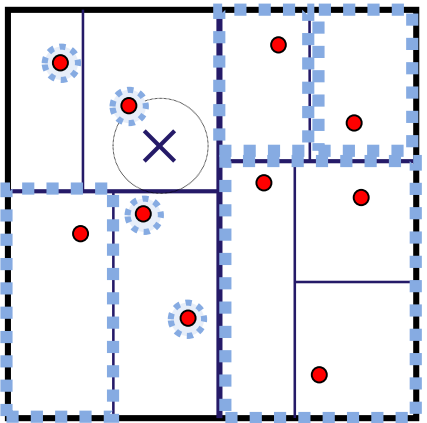
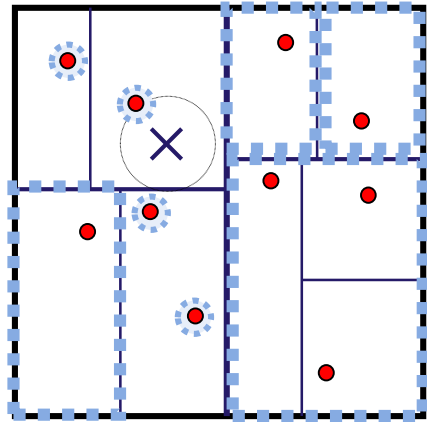
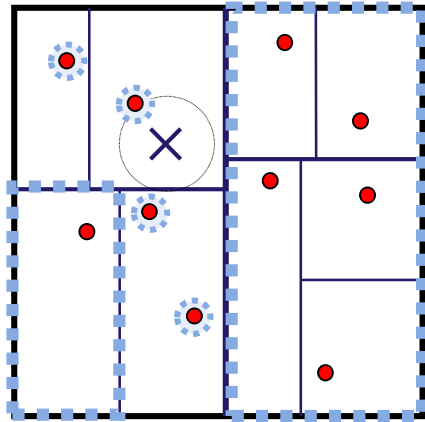
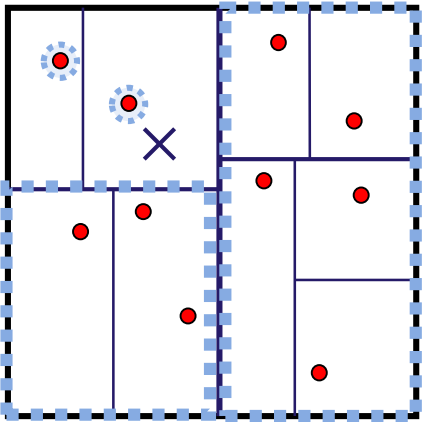
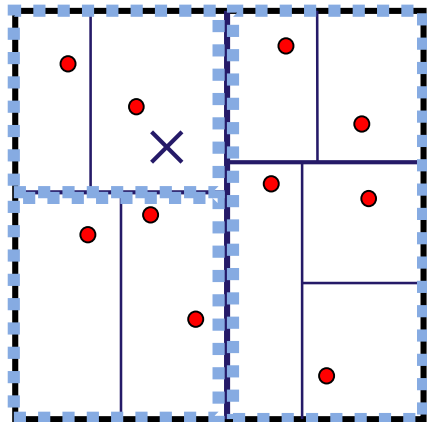
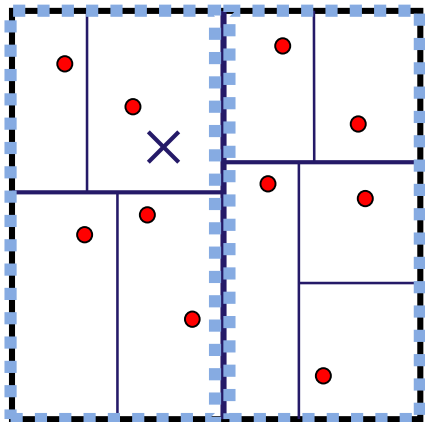
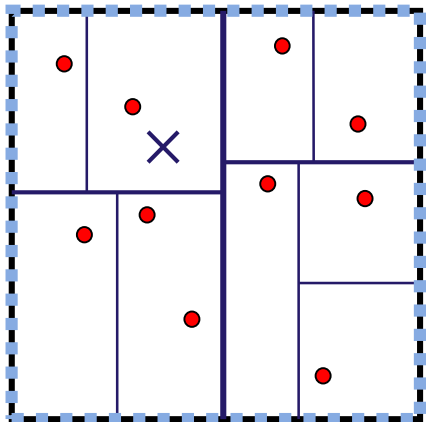
If this is a point:

 output the point

Otherwise, if this is a node:

If leaf node: Insert all points into queue

If inner node: Insert all child-boxes into queue



How to Search in High Dimensions

Nearest-Neighbor(s) / ϵ -Neighbors

- Linear-time brute-force search always work
- Tree-based algorithms
 - Reasonable space/ precomputation
 - Worst-case search time bounds exponential in dimension
- **In practice**
 - kD-Trees work up to dimension 10-20
 - Approximate search to speed it up
 - Libraries: ANN, FLANN
- **J-S-Lemma**
 - Reduce dimensionality to 10-20, then use ANN/FLANN
 - Direct application: Locality-sensitive-hashing (LSH)

– too much work in high dimensions –

Computational Issues

INTEGRATION

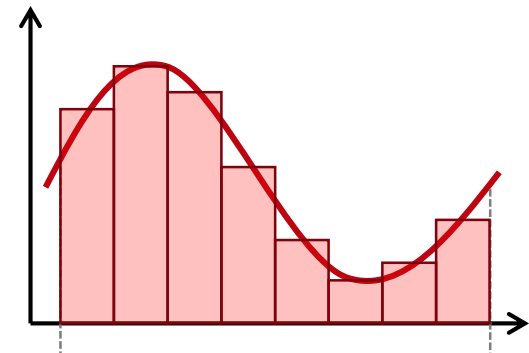
High Dimensional Integrals

Classic application domain

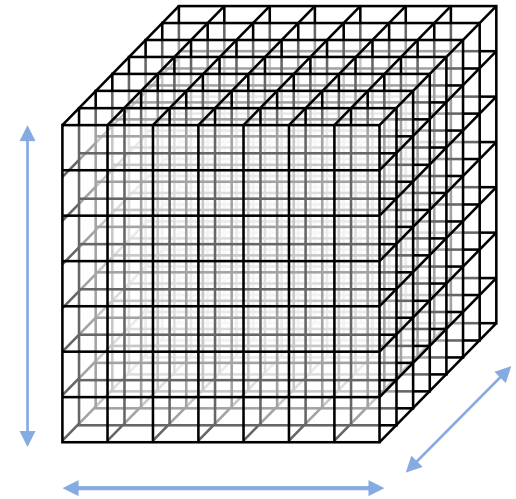
- High-dimensional integration domains
- Let's say, $\Omega = [0,1]^{20}$

Standard Integration

- Regular grid, k^{20} samples
- No need to try this...



Rieman-sum



k subdivisions
per axis

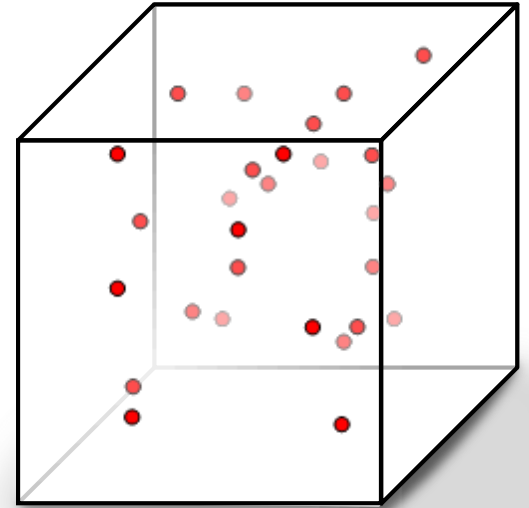
Higher Dimensions

Monte-Carlo Approach:

- Sample n points
- Compute average
- Multiply with domain volume

Property

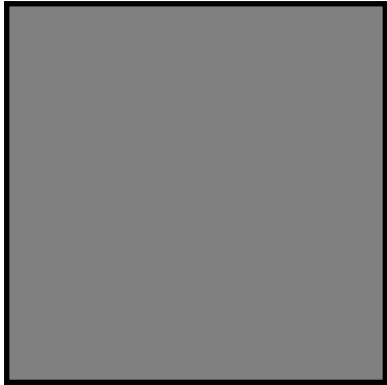
- Works if variance is not too large
- Dimension irrelevant



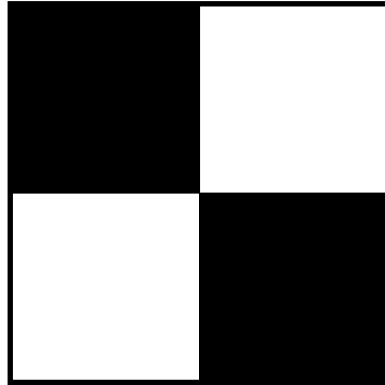
n sample points
(irregular)

Example

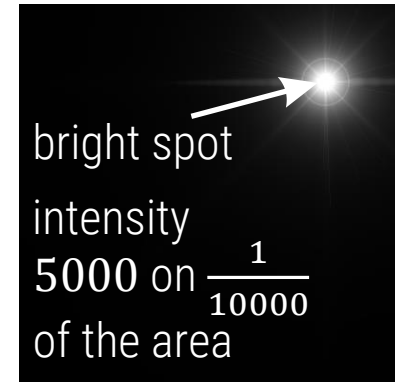
When is Monte-Carlo integration possible?



optimal –
no variance



moderate variance –
MC-int. possible



large variance –
not efficient

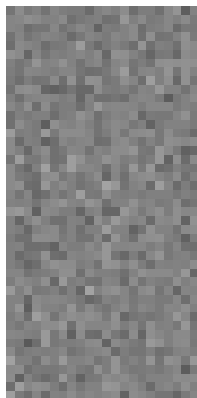
General observation

- Randomized algorithms are efficient if the *crucial information* is *easy to find* by random trials

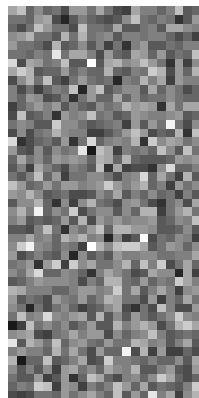
Numerical Example



$q = 1$



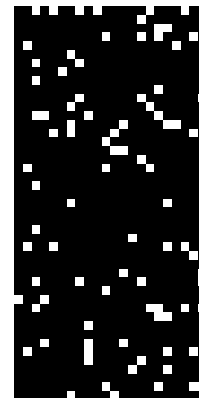
$q = 0.5$



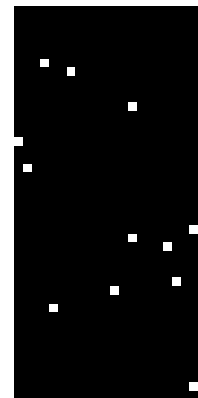
$q = 0.1$



$q = 0.01$



$q = 0.001$



$q = 0.0001$

Averaging Samples:

- $n = 100$ samples
- Fraction q of the domain with value $0.5/q$
- Showing multiple pixels

Example

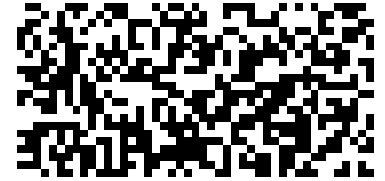
Speed of convergence:

- Now growing n
- Pixel: 50% black / 50% white
- Growing sample size

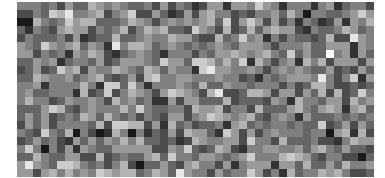
Observation

- Large sample size required before noise vanishes

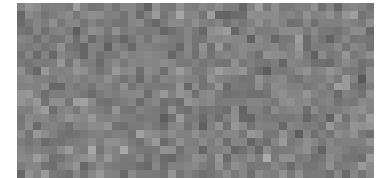
$n = 1$



$n = 10$



$n = 100$



$n = 1000$



$n = 10000$



Variance Reduction

Two reasons for long compute times

$$\sigma \in \mathcal{O} \left(\frac{\sigma(f)}{\sqrt{n}} \right)$$

Biggest Problem
primary estimator
variance

Possible solution: *Importance Sampling*

Importance Sampling

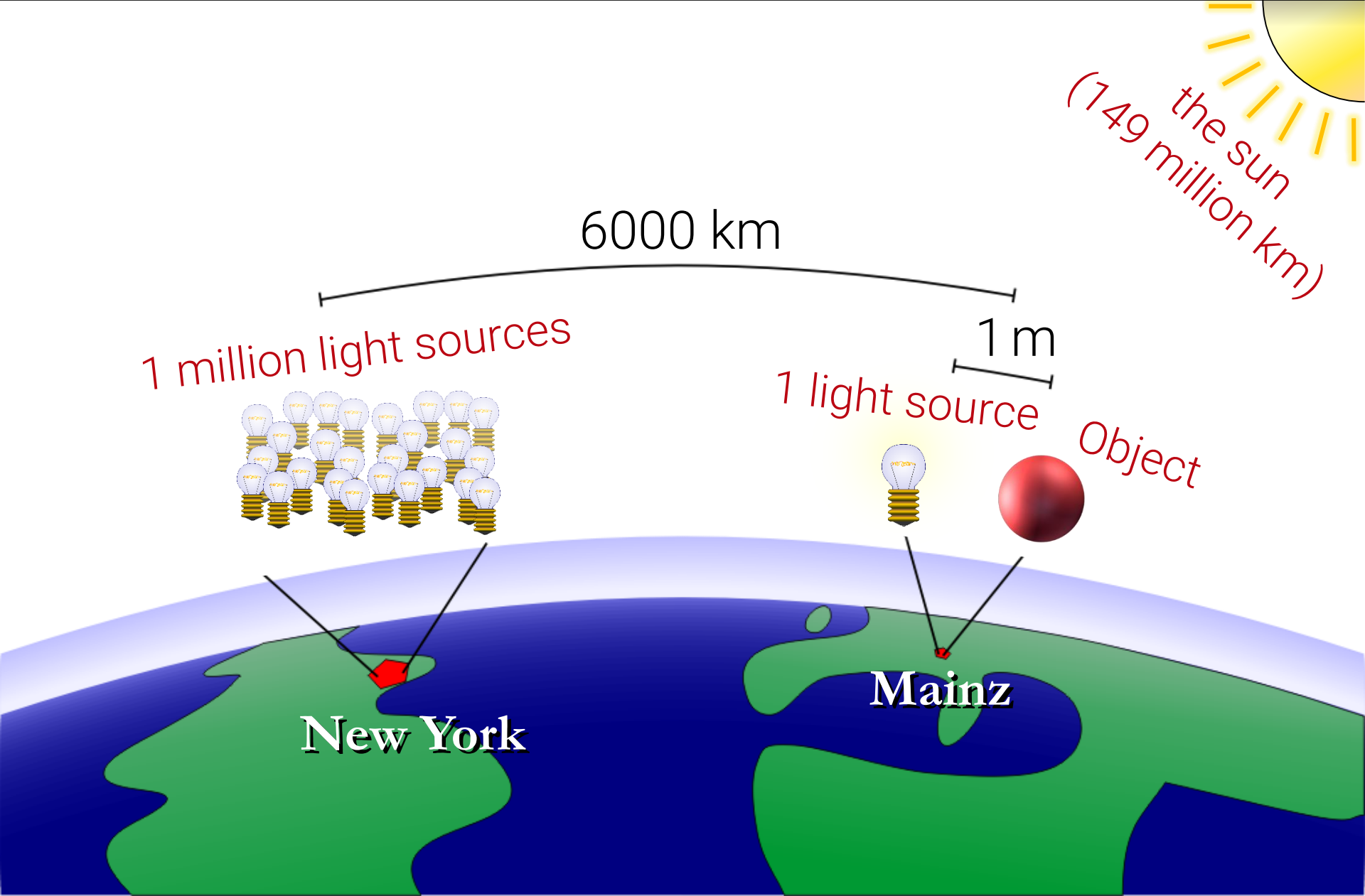
Importance Sampling

- Idea: More samples in important regions
 - Need to weight differently to avoid bias
- New estimator
 - Choose sampling density p on Ω

$$\int_{\Omega} f(x) dx \approx \frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{p(x_i)} \quad (p(x) > 0 \forall x \in \Omega)$$

- (Note: No $|\Omega|$ factor required here.)
- Sampling density p controls importance

Illustrative Example from Graphics



More Complex Sampling Problems

What if sampling itself is costly?

- For example, from a MRF

$$p(x_1, \dots, x_n) = \frac{1}{Z} \prod_i p(x_i) \prod_{i,j} p(x_i, x_j)$$

Markov-Chain Monte-Carlo

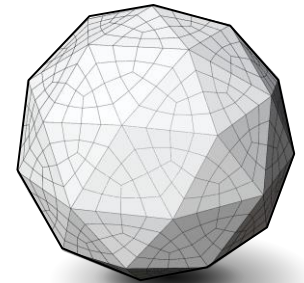
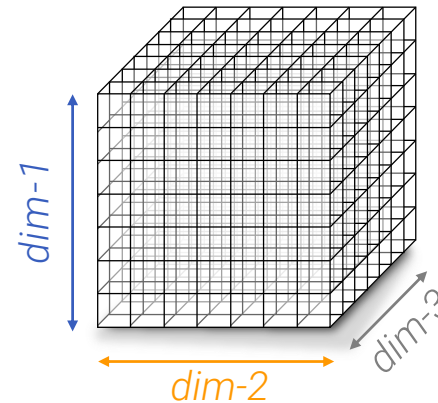
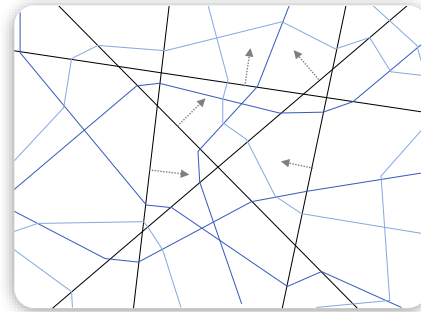
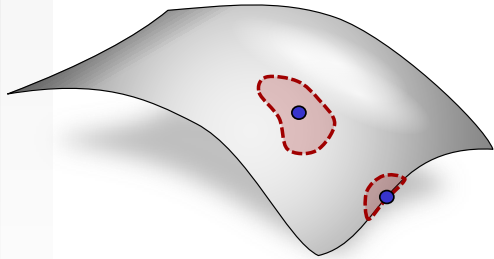
- Gibbs sampler (for graphical models / MRFs)
- Metropolis sampler (for unnormalized densities)

Nonetheless...

- Many (e.g. Bayesian) integration problems remain intractable

Modelling 2

STATISTICAL DATA MODELLING



Chapter 10 Space

Video #10

Space

- **High-dimensional space**
& the curse of dimensionality
- **Kernels:** flat space extended
- **Manifolds:** curved space

Data Modeling with Kernels

Topics

- Inner products & kernel
 - Definitions
 - Networks as kernels
 - Understanding kernels via dual PCA
- Gaussian Processes
 - Data analysis with GP
- Application to DNN analysis
 - Networks as GPs
 - Neural tangent kernel
 - Towards explaining “Double Descent”

Inner Products & Kernels

More on Kernel Methods

John Shawe-Taylor, Nello Cristianini:

Kernel Methods for Pattern Analysis.

Cambridge University Press, 2004

Inner Products

Vector space V

- Inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ of vectors $\mathbf{x}, \mathbf{y} \in V$

- Symmetric (commutative)

$$\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$$

- Bilinear

$$\langle \lambda \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y} \rangle = \lambda \langle \mathbf{x}_1, \mathbf{y} \rangle + \langle \mathbf{x}_2, \mathbf{y} \rangle$$

- Positive definite

$$\langle \mathbf{x}, \mathbf{x} \rangle \geq 0 \text{ and } \langle \mathbf{x}, \mathbf{x} \rangle = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$$

- In finite dimensions, this are exactly functions

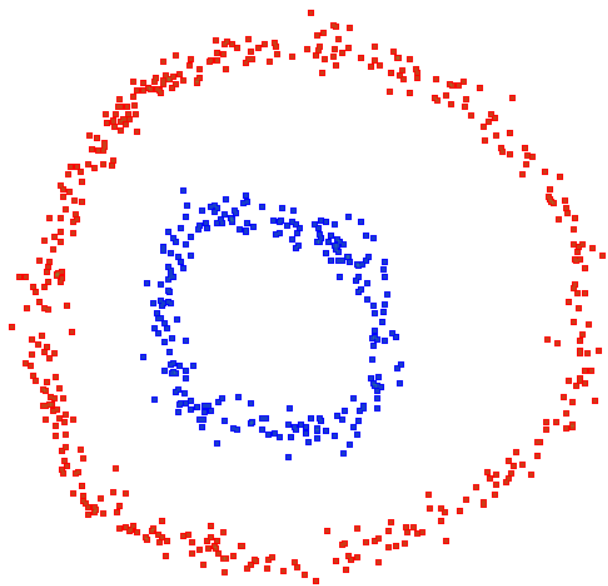
$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{M} \mathbf{y} \text{ for SPD matrices } \mathbf{M}$$

or equivalently

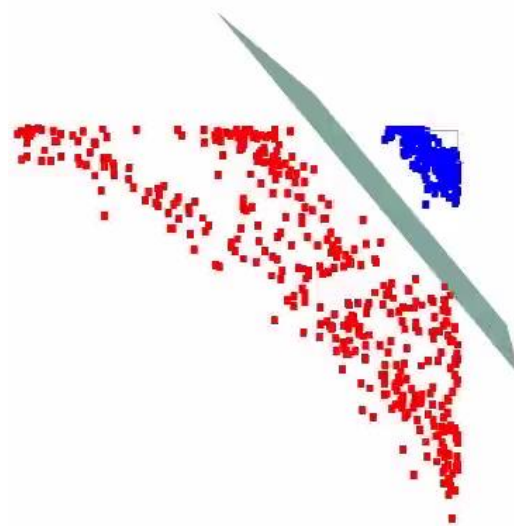
$$\langle \mathbf{x}, \mathbf{y} \rangle = (\mathbf{T} \mathbf{x})^T (\mathbf{T} \mathbf{y}) \text{ for invertible matrices } \mathbf{T}$$

Cartoon Example

Example Goal: Linear classification



original space

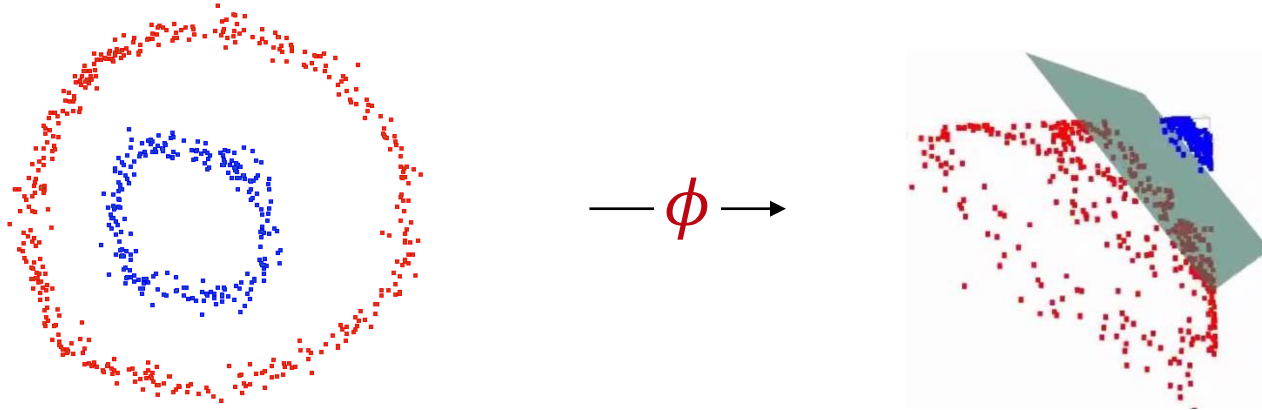


“feature space”

$$\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$(x, y) \mapsto (x^2, xy, y^2)$$

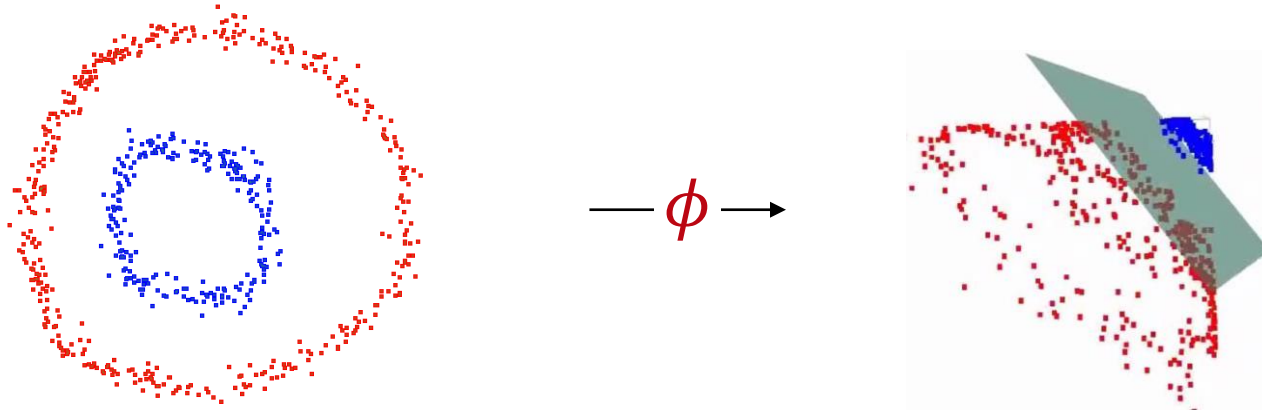
Kernels



Feature spaces

- Data $\mathbf{x} \in V$ from vector space V
- Transform data by function $\phi: V \rightarrow W$
- Vector space W is called “feature space”

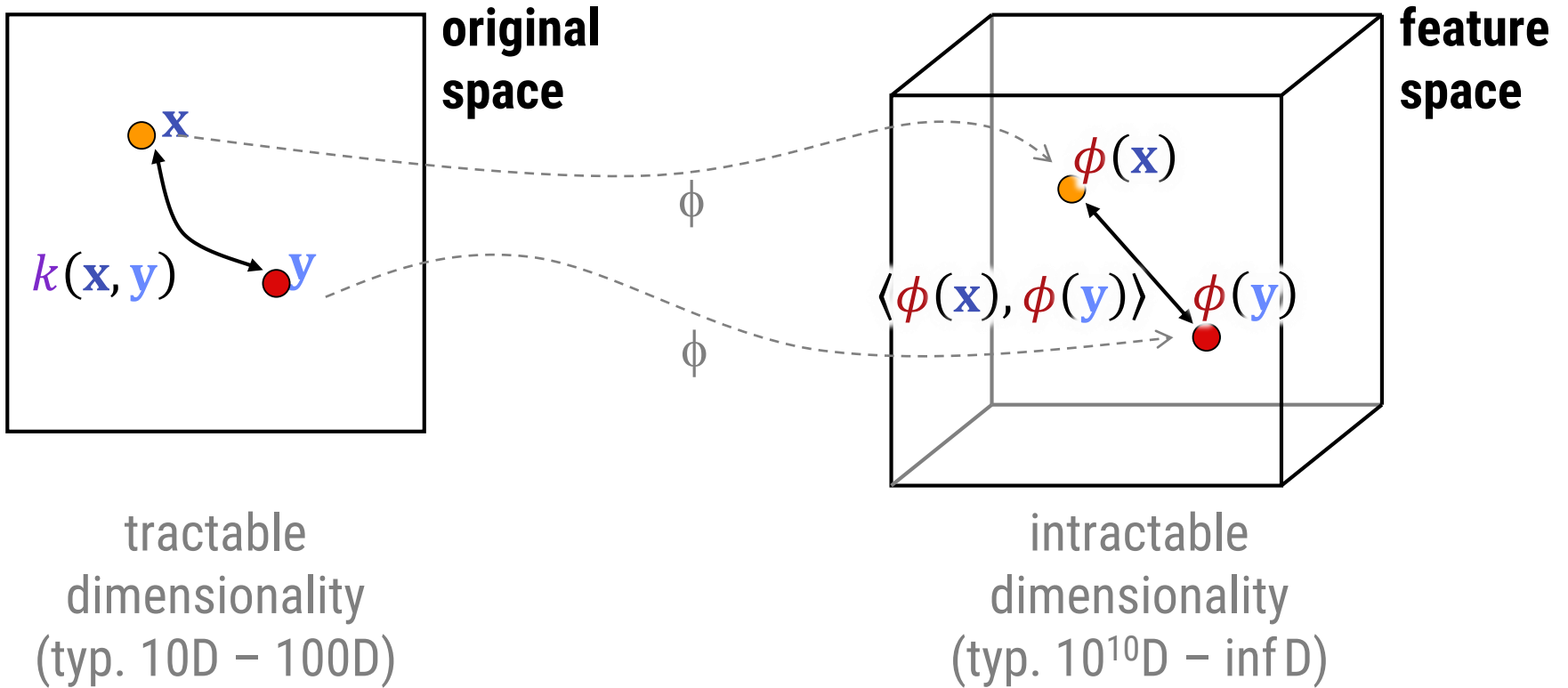
Kernels



Kernels: Inner products in feature space

- Kernel $\kappa(\mathbf{x}, \mathbf{y}) := \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$
 - Data $\mathbf{x}, \mathbf{y} \in V$
- Why ϕ ?
 - ϕ can emphasize / unveil structure
- Why kernels?
 - Sometimes easier to compute (as in “tractable”)

“The Kernel Trick”



Why Kernels?

Expensive Kernels

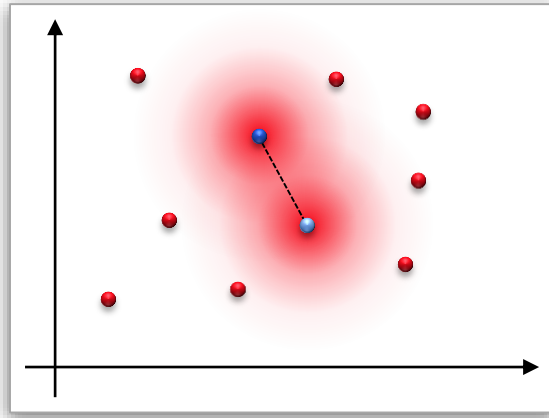
- „Polynomial kernel“

$$\kappa(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^D$$

computes implicitly all monomials up to degree D

- Attention: with non-uniform coefficients
- Direct mapping ϕ would be exponential in D
- Can analyze higher moments at moderate costs

Another Popular Kernel



Gaussian / RBF / squared-exponential kernel

$$k(\mathbf{x}, \mathbf{y}) = \exp(-a\|\mathbf{x} - \mathbf{y}\|^2)$$

- Popular choice
- Ignores detail in data below length-scale $\approx \sigma$
- Go-to solution for kernel SVMs, GP regression

Fourier Analysis

Analysis

$$k(r) = \exp(-ar^2) \quad \text{with } r := \|\mathbf{x} - \mathbf{y}\|$$

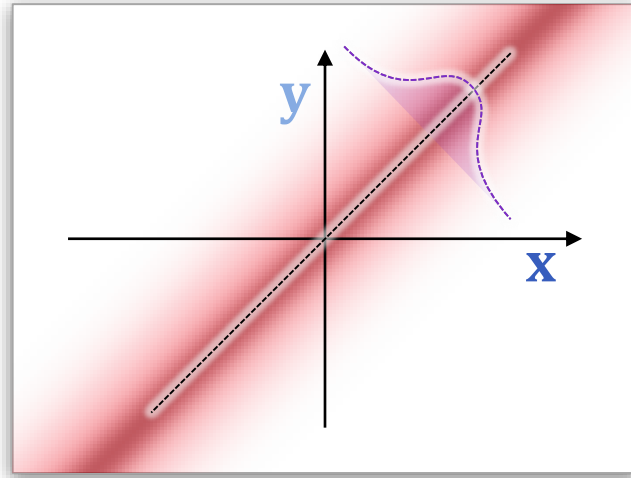
Fourier-Domain

- Fourier-transformation

$$K(\omega) = \sqrt{\frac{\pi}{a}} \cdot e^{-\frac{(\pi\omega)^2}{a}}$$

- Low-pass filter on the distance function
 - Exponential frequency drop-off

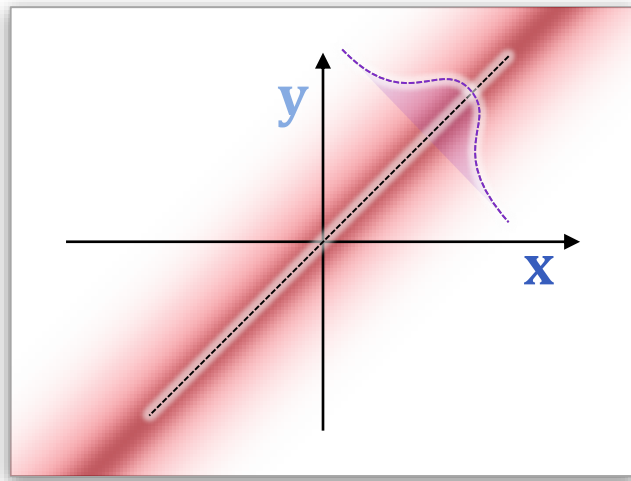
Fourier Analysis



Multi-Dimensional FT

- Gaussian cross section along $\mathbf{x} \perp \mathbf{y}$
- Constant along $\mathbf{x} \parallel \mathbf{y}$

Fourier Analysis



Numerical approximation

- Assume Data in $\mathbf{x} \in [0, 2\pi]^d$

$$k(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{w} \in \mathbb{Z}^d} z_{\mathbf{w}} e^{-i\langle \mathbf{w}, \mathbf{x} - \mathbf{y} \rangle} = \sum_{\mathbf{w} \in \mathbb{Z}^d} z_{\mathbf{w}} e^{-i\langle \mathbf{w}, \mathbf{x} \rangle} e^{i\langle \mathbf{w}, \mathbf{y} \rangle}$$

with non-zero Fourier coefficients $z_{\mathbf{w}} \in \mathbb{R}$

Note Networks as Kernels

How does a deep network classifier work?

- Apply deep network $f^L \circ \dots \circ f^1$ on inputs \mathbf{x}
- Final layer f^L usually a (fully-connected) linear layer
- Followed by soft-max & x-entropy loss or hinge-loss

Kernel-machine

- Consider $f^{L-1} \circ \dots \circ f^1$ a learned kernel
- Last layer: logistic/softmax regression or SVM
- Kernel LR / Kernel-SVM
 - But the kernel is rather “fancy”
 - Most traditional practice: “handcrafted” kernels

Simple Kernelization

How to Kernelize “any” Algorithm

Run **Kernel-PCA** (a.k.a. dual-PCA / MDS)

- Data

$$\mathbf{X} = \begin{pmatrix} | & & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ | & & | \end{pmatrix}$$

- Feature matrix (do not compute!)

$$\phi(\mathbf{X}) = \begin{pmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ | & & | \end{pmatrix}$$

- Its Gram matrix (kernel evaluations only)

$$\mathbf{G} = \begin{pmatrix} \ddots & & \ddots \\ & \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle & \\ \ddots & & \ddots \end{pmatrix} = \phi(\mathbf{X})^T \phi(\mathbf{X}),$$

How to Kernelize “any” Algorithm

Run Kernel-PCA (dual-PCA / MDS)

- Take a “square root” of \mathbf{G} :

$$\text{„}\sqrt{\mathbf{G}}\text{“} = \phi(\mathbf{X}) = \begin{pmatrix} | & & | \\ \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n) \\ | & & | \end{pmatrix}$$

- Obtained from eigenvalue decomposition

$$\mathbf{G} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = (\mathbf{V}\sqrt{\mathbf{\Lambda}})(\sqrt{\mathbf{\Lambda}}^T\mathbf{V}^T)$$

- Recovering rotated feature space

$$\phi(\mathbf{X}) = \mathbf{R}(\sqrt{\mathbf{\Lambda}}\mathbf{V}^T) \text{ for some orthogonal } \mathbf{R}$$

- Because $\sqrt{\mathbf{G}}$ is not unique

$$\mathbf{G} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T = (\mathbf{V}\sqrt{\mathbf{\Lambda}})\underbrace{\mathbf{R}^T\mathbf{R}}_{\mathbf{I}}(\sqrt{\mathbf{\Lambda}}\mathbf{V}^T)$$

Why is this good?

Kernel-algorithm recipe

- Compute kernel-matrix \mathbf{G}
- Eigendecomposition $\sqrt{\mathbf{G}} = \sqrt{\mathbf{\Lambda}}\mathbf{V}^T$
- Embedding = columns of $\sqrt{\mathbf{\Lambda}}\mathbf{V}^T$
- Use embedded points in ML-algorithm

Complexity reduction

- n data points
- Embedding: at most n dimensions
- Potentially high-dim. feature space (before \mathbf{R})

Does this do the trick?

Invariance

- Kernel (Gram) matrix is rotation invariant

$$[\phi(\mathbf{X})]^T [\phi(\mathbf{X})] = [\phi(\mathbf{X})]^T \mathbf{R}^T \mathbf{R} [\phi(\mathbf{X})] = [\mathbf{R}\phi(\mathbf{X})]^T [\mathbf{R}\phi(\mathbf{X})]$$

- Embedding does not alter information

Costs

- Kernel matrix is always needed
 - $\mathcal{O}(n^2)$ costs
- Spectral decomposition is typically $\mathcal{O}(n^3)$
 - Might be suboptimal
 - But very easy to employ

Nyström Projection

Nyström Projection

- Embed new feature $\phi(\mathbf{x})$:

$$\text{emb}(\phi(\mathbf{x})) = \mathbf{V}^T \mathbf{\Lambda}^{-1} \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}) \\ \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}) \end{pmatrix}$$

Nyström Projection

Nyström Projection

- Reminder: $\phi(\mathbf{X}) = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$ $\mathbf{G} = \phi(\mathbf{X})^T \phi(\mathbf{X}) = \mathbf{V}\mathbf{\Lambda}^2\mathbf{V}^T$
 $emb_{PCA}(\phi(\mathbf{X})) = \mathbf{U}^T \phi(\mathbf{X})$ $emb_{dual}(\phi(\mathbf{X})) = \mathbf{\Lambda}\mathbf{V}^T$
- Project new feature $\phi(\mathbf{x})$ on principal axes $\mathbf{u}_1, \dots, \mathbf{u}_d$:

$$\begin{aligned} emb(\phi(\mathbf{x})) &= \mathbf{U}^T \phi(\mathbf{x}) \\ &= (\mathbf{V}^T \mathbf{\Lambda}^{-1} \phi(\mathbf{X})^T) \phi(\mathbf{x}) \\ &= \begin{pmatrix} \sum_{i=1}^n \frac{1}{\lambda_1} v_{i,1} \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle \\ \vdots \\ \sum_{i=1}^n \frac{1}{\lambda_n} v_{i,n} \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle \end{pmatrix} \end{aligned}$$

$$\begin{aligned} \phi(\mathbf{X}) &= \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \\ \Rightarrow \mathbf{U} &= \phi(\mathbf{X})\mathbf{V}\mathbf{\Lambda}^{-1} \\ \Rightarrow \mathbf{U}^T &= \mathbf{V}^T \mathbf{\Lambda}^{-1} \phi(\mathbf{X})^T \end{aligned}$$

Nyström Projection

Nyström Projection

- Reminder: $\phi(\mathbf{X}) = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$ $\mathbf{G} = \phi(\mathbf{X})^T \phi(\mathbf{X}) = \mathbf{V}\mathbf{\Lambda}^2\mathbf{V}^T$
 $emb_{PCA}(\phi(\mathbf{X})) = \mathbf{U}^T \phi(\mathbf{X})$ $emb_{dual}(\phi(\mathbf{X})) = \mathbf{\Lambda}\mathbf{V}^T$
- Project new feature $\phi(\mathbf{x})$ on principal axes $\mathbf{u}_1, \dots, \mathbf{u}_d$:

$$\begin{aligned} emb(\phi(\mathbf{x})) &= \mathbf{U}^T \phi(\mathbf{x}) \\ &= (\mathbf{V}^T \mathbf{\Lambda}^{-1} \phi(\mathbf{X})^T) \phi(\mathbf{x}) \\ &= \begin{pmatrix} \sum_{i=1}^n \frac{1}{\lambda_1} v_{i,1} \kappa(\mathbf{x}_i, \mathbf{x}) \\ \vdots \\ \sum_{i=1}^n \frac{1}{\lambda_n} v_{i,n} \kappa(\mathbf{x}_i, \mathbf{x}) \end{pmatrix} \end{aligned}$$

Nyström Projection

Nyström Projection

- Embed new feature $\phi(\mathbf{x})$:

$$\text{emb}(\phi(\mathbf{x})) = \mathbf{V}^T \mathbf{\Lambda}^{-1} \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}) \\ \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}) \end{pmatrix}$$

Training & Inference

- Determine factorization for training
- Use Nyström-Projection for inference
 - Input new data points into non-kernelized ML-algorithm
- Embedding varies with $\kappa(\mathbf{x}_i, \cdot)$
 - Gaussian: Smoothed proximity to $\mathbf{x}_1, \dots, \mathbf{x}_n$
 - Distance-based learning scheme

Note: MDS

We can convert

- Gram matrix → all pairwise distances
 - Losing global translation
- All pairwise distances → Gram matrix
 - Up to a global translation

Consequences

- After feature-map, algorithms are distance-based
 - Johnson-Lindenstrauss-Lemma
 - Can be approximated in rel. low dimensions
 - Much less information than full vectors
- Kernel design by distances-design
 - Often more intuitive

Gaussian Process Regression

Background Literature

Carl Edward Rasmussen, Christopher K. I. Williams

Gaussian Processes for Machine Learning

The MIT Press, 2006.

<http://www.gaussianprocess.org/gpml/>

Linear Regression (w/linear basis)

Regression

- Data points

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$$

- Looking for approx. function

$$f(\mathbf{x}_i) \approx y_i$$

Linear Regression

- Ansatz

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$$

- Objective

$$\sum_{i=1}^n \|\mathbf{w}^T \mathbf{x}_i - y_i\|^2 = \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w}^T \mathbf{w} \mathbf{x}_i - 2\mathbf{w}^T \mathbf{x}_i y_i + y_i^2) \rightarrow \min.$$

Linear Regression (w/linear basis)

Kernelized Linear Regression

- Ansatz

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

- Objective

$$\begin{aligned} & \sum_{i=1}^n \|\mathbf{w}^T \phi(\mathbf{x}_i) - y_i\|^2 \\ &= \sum_{i=1}^n (\mathbf{w}^T [\phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T] \mathbf{w} - 2\mathbf{w}^T \phi(\mathbf{x}_i) y_i + y_i^2) \rightarrow \min. \end{aligned}$$

- Still: Quadratic optimization problem in \mathbf{w}
 - Gaussian probabilistic model
 - Simply solve a linear system

How Interesting is This?

Moderately Interesting?

- Ansatz

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$$

with

$$\phi(\mathbf{x}) = \begin{pmatrix} b_1(\mathbf{x}) \\ \vdots \\ b_k(\mathbf{x}) \end{pmatrix}$$

is just approximation with a linear basis (Mod-1)

- For example

$$\phi(x) = (1, x, x^2, \dots, x^D)^T$$

yields polynomial fitting (Video 05d)

Non-Parametric GPs

Gaussian Processes

Main idea

- Consider function space
- Define Gaussian distribution in function space
 - Gaussian priors
 - Gaussian data terms
- Use this to solve various ML-problems

Technical challenge

- Gaussians in infinite-dim. space

Gaussian Processes

Gaussian processes

- We would like to infer functions

$$f: \mathbb{R}^d \rightarrow \mathbb{R}$$

- We assume a “Gaussian distribution”

- For any finite sample $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$

$f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)$ are normal distributed

- ...with mean $\mu = 0$,

i.e. $\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_n) = 0$

- ...and covariance

$$\begin{aligned} \text{cov} \left(f(\mathbf{x}_i), f(\mathbf{x}_j) \right) &= \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ &= \mathbb{E} \left[\underbrace{(f(\mathbf{x}_i) - \mathbb{E}[f(\mathbf{x}_i)]) \cdot (f(\mathbf{x}_j) - \mathbb{E}[f(\mathbf{x}_j)])}_{\text{covariance}} \right] \end{aligned}$$

- The “kernel” κ is called the *covariance function*.

Nothing to see here, move on...

Comparison: vector case

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_n \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \ddots & & \\ & \text{COV}(\mathbf{x}_1, \mathbf{x}_j) & \\ & & \ddots \end{pmatrix}$$

Functions / GPs

$$f \rightarrow \begin{pmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}, \quad \boldsymbol{\mu} \rightarrow \begin{pmatrix} \mu(x_1) \\ \vdots \\ \mu(x_n) \end{pmatrix}, \quad \mathbf{K} = \begin{pmatrix} \ddots & & \\ & \kappa(\mathbf{x}_1, \mathbf{x}_j) & \\ & & \ddots \end{pmatrix}$$

Gaussian Process Regression

Regression

- Data points

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$$

- Looking for approx. function

$$f(\mathbf{x}_i) \approx y_i$$

Prior on function space

- Normal distribution on functions $f \sim \mathcal{N}_{0, \kappa}$
 - In the sense of the previous slide

Data term

- Noisy observations $f(\mathbf{x}_i) \sim \mathcal{N}_{y_i, \sigma}(y)$

Bayes Rule

Combining Data + Prior

$$P(f|D) = \frac{P(D|f)P(f)}{P(D)} \sim P(D|f)P(f)$$

Notation (next slide)

- Training data points $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$
with values $y_1, \dots, y_n \in \mathbb{R}$
- Query data points $\mathbf{x}_1^*, \dots, \mathbf{x}_m^* \in \mathbb{R}^d$
- Unknown function f :
 - Unknown function values $\mathbf{Y} = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_n))^T$
 - Not necessarily equal to (y_1, \dots, y_n)
 - Unknown query function values $\mathbf{Y}^* = (f(\mathbf{x}_1^*), \dots, f(\mathbf{x}_m^*))^T$

Bayes Rule

Data Term

$$P(f|D) \sim P(D|f)P(f)$$

$$\blacksquare P(\mathbf{Y}|D) = \prod_{i=1}^n \mathcal{N}_{\mathbf{y}_i, \sigma}(\mathbf{Y})$$

Prior

$$\blacksquare P\left(\begin{bmatrix} \mathbf{Y} \\ \mathbf{Y}^* \end{bmatrix}\right) = \prod_{i=1}^n \mathcal{N}_{\mathbf{0}, \Sigma}\left(\begin{bmatrix} \mathbf{Y} \\ \mathbf{Y}^* \end{bmatrix}\right) \text{ with } \Sigma = \begin{pmatrix} K(\mathbf{Y}, \mathbf{Y}) & K(\mathbf{Y}, \mathbf{Y}^*) \\ K(\mathbf{Y}^*, \mathbf{Y}) & K(\mathbf{Y}^*, \mathbf{Y}^*) \end{pmatrix}$$

$$\text{where } K(\mathbf{X}, \mathbf{Y}) = \begin{pmatrix} \ddots & & \ddots \\ & \kappa(\mathbf{x}_i, \mathbf{y}_j) & \\ \ddots & & \ddots \end{pmatrix}$$

Inference

Determine new function values

- Multiply Gaussians
 - Result is again a Gaussian
 - Mean and covariance matrix change (combination)
- Means for new variables yield interpolation
 - Variances are also available!

Corresponding linear system

$$\left[\begin{pmatrix} K(\mathbf{Y}, \mathbf{Y}) + \sigma^{-2} \mathbf{I} & K(\mathbf{Y}, \mathbf{Y}^*) \\ K(\mathbf{Y}^*, \mathbf{Y}) & K(\mathbf{Y}^*, \mathbf{Y}^*) \end{pmatrix} \right] \begin{pmatrix} \mathbf{Y} \\ \mathbf{Y}^* \end{pmatrix} = \begin{pmatrix} \sigma^{-2} \mathbf{y} \\ \mathbf{0} \end{pmatrix}$$

Example 1: Image Reconstruction



Minimization Problem

Continuous

$$\sigma_D^{-2} \int_{\Omega} (f(\mathbf{x}) - d(\mathbf{x}))^2 d\mathbf{x} + \sigma_X^{-2} \int_{\Omega} \|\nabla f(\mathbf{x})\|^2 d\mathbf{x} \rightarrow \min.$$

Minimize

$$\begin{aligned} & E(D|X) + E(X) \\ &= \sum_{i=1}^w \sum_{j=1}^h \frac{(x_i - d_i)^2}{2\sigma_D^2} + \sum_{i=1}^{w-1} \sum_{j=1}^{h-1} \frac{(x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2}{2\sigma_X^2} \end{aligned}$$

Equivalent minimization objective

$$\sum_{i=1}^w \sum_{j=1}^h (x_i - d_i)^2 + \frac{\sigma_X^2}{\sigma_D^2} \sum_{i=1}^{w-1} \sum_{j=1}^{h-1} (x_{i+1,j} - x_{i,j})^2 + (x_{i,j+1} - x_{i,j})^2$$

Euler-Lagrange Equation

Variational problem

$$E(f) \rightarrow \min. \quad \text{with}$$

$$E(f) = \int_{\Omega} F \left(x_1, \dots, x_d, f(\mathbf{x}), \partial_{x_1} f(\mathbf{x}), \dots, \partial_{x_d} f(\mathbf{x}) \right) d\mathbf{x}$$

Necessary condition

$$\frac{\partial F}{\partial \arg\{f(\mathbf{x})\}} = \sum_{i=1}^d \frac{\partial}{\partial x_i} \frac{\partial F}{\partial \arg\{\partial_{x_i} f(\mathbf{x})\}}$$

ordinary partial derivative by x_i

derivative of F by corr. argument

Diffusion / Poisson Equation

Harmonic energy

$$E(f) = \int_{\Omega} \|\nabla f(\mathbf{x})\|^2 d\mathbf{x} \rightarrow \min.$$

Necessary condition

$$\Delta f(\mathbf{x}) = 0$$

Proof

- Euler-Lagrange-Equation (Mod-1)

Diffusion / Poisson Equation

Harmonic energy

$$E(f) = \int_{\Omega} \|\nabla f(\mathbf{x})\|^2 d\mathbf{x} = \int_{\Omega} \left(\partial_{x_1} f(\mathbf{x}) \right)^2 + \dots + \left(\partial_{x_d} f(\mathbf{x}) \right)^2 d\mathbf{x}$$

Necessary condition

$$\begin{aligned} 0 &= \sum_{i=1}^d \frac{\partial}{\partial x_i} \frac{\partial \left(\mathbf{x} \rightarrow \left(\partial_{x_1} f(\mathbf{x}) \right)^2 + \dots + \left(\partial_{x_d} f(\mathbf{x}) \right)^2 \right)}{\partial \arg\{\partial_{x_i} f(\mathbf{x})\}} \\ &= \sum_{i=1}^d \frac{\partial}{\partial x_i} 2 \partial_{x_i} f(\mathbf{x}) = 2 \sum_{i=1}^d \frac{\partial^2 f(\mathbf{x})}{\partial x_i^2} \Rightarrow \Delta f(\mathbf{x}) = 0 \end{aligned}$$

Eigen Analysis

Eigenfunctions of the Laplacian

$$\Delta f = \partial_{x_1}^2 f + \dots + \partial_{x_d}^2 f$$

Eigenfunctions

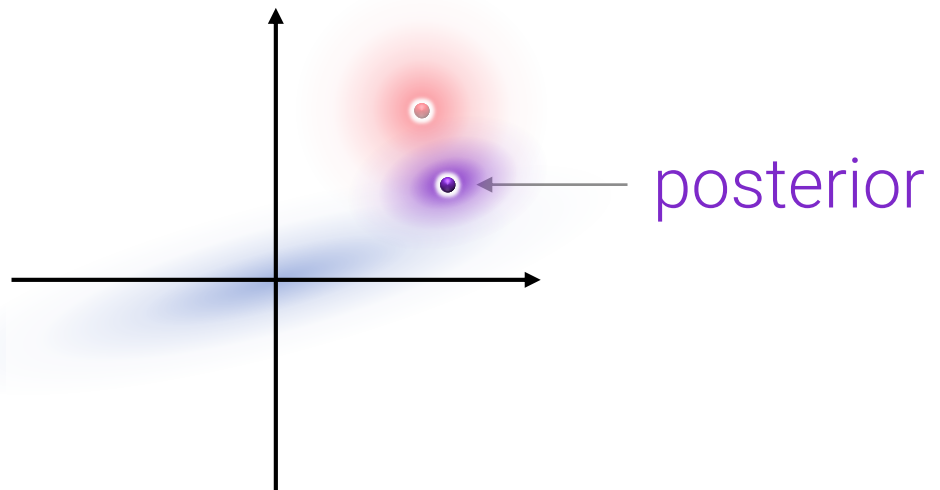
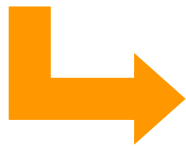
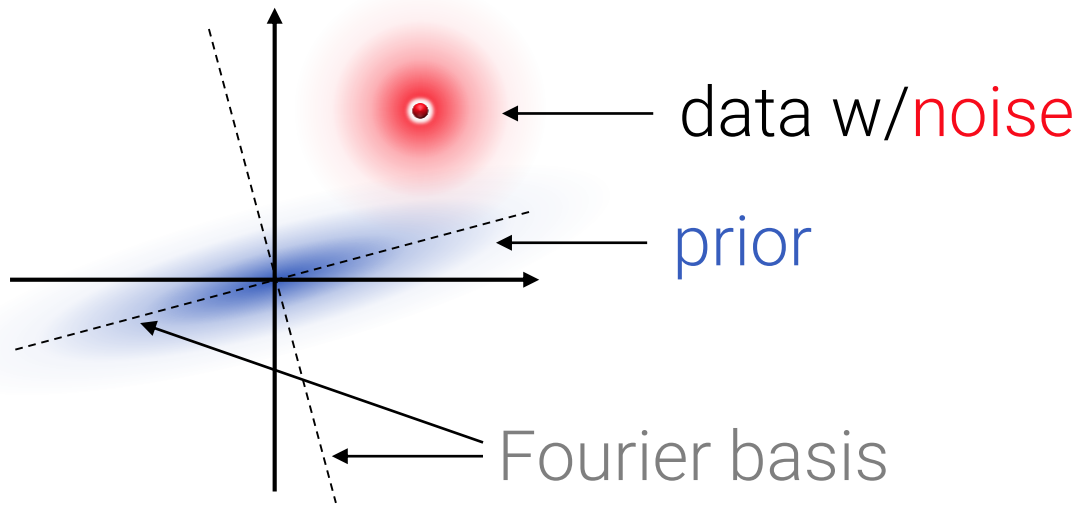
- Square domain $\Omega = [0, 2\pi]^2$
- Eigenbasis: Fourier-Basis

$$b_{\omega_1, \omega_2}(x, y) = e^{-i(\omega_1 x + \omega_2 y)}, \quad \omega_1, \omega_2 \in \mathbb{Z}$$

- Eigenvalues

$$\lambda_{\omega_1, \omega_2} = \omega_1^2 + \omega_2^2$$

All Just Gaussians...

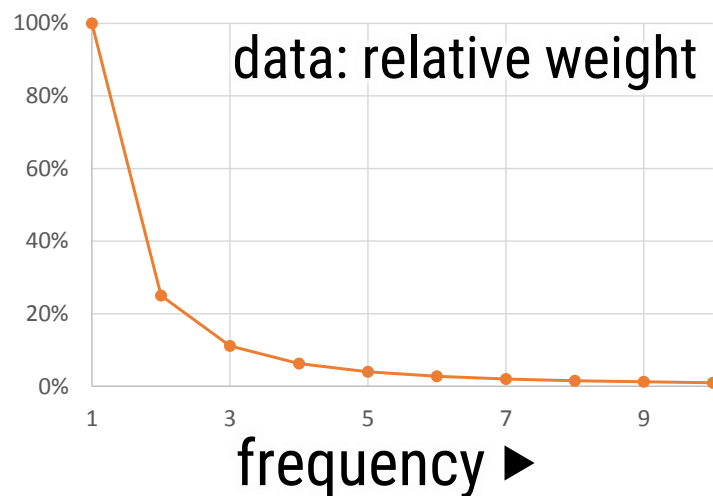
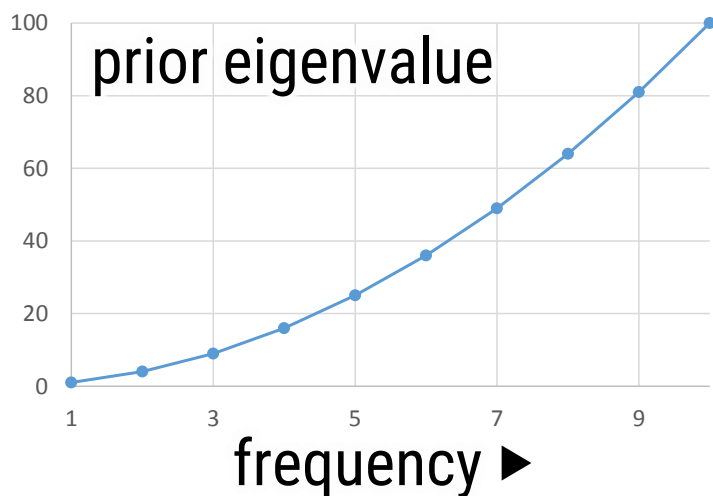


So what does it do?

Optimization problem

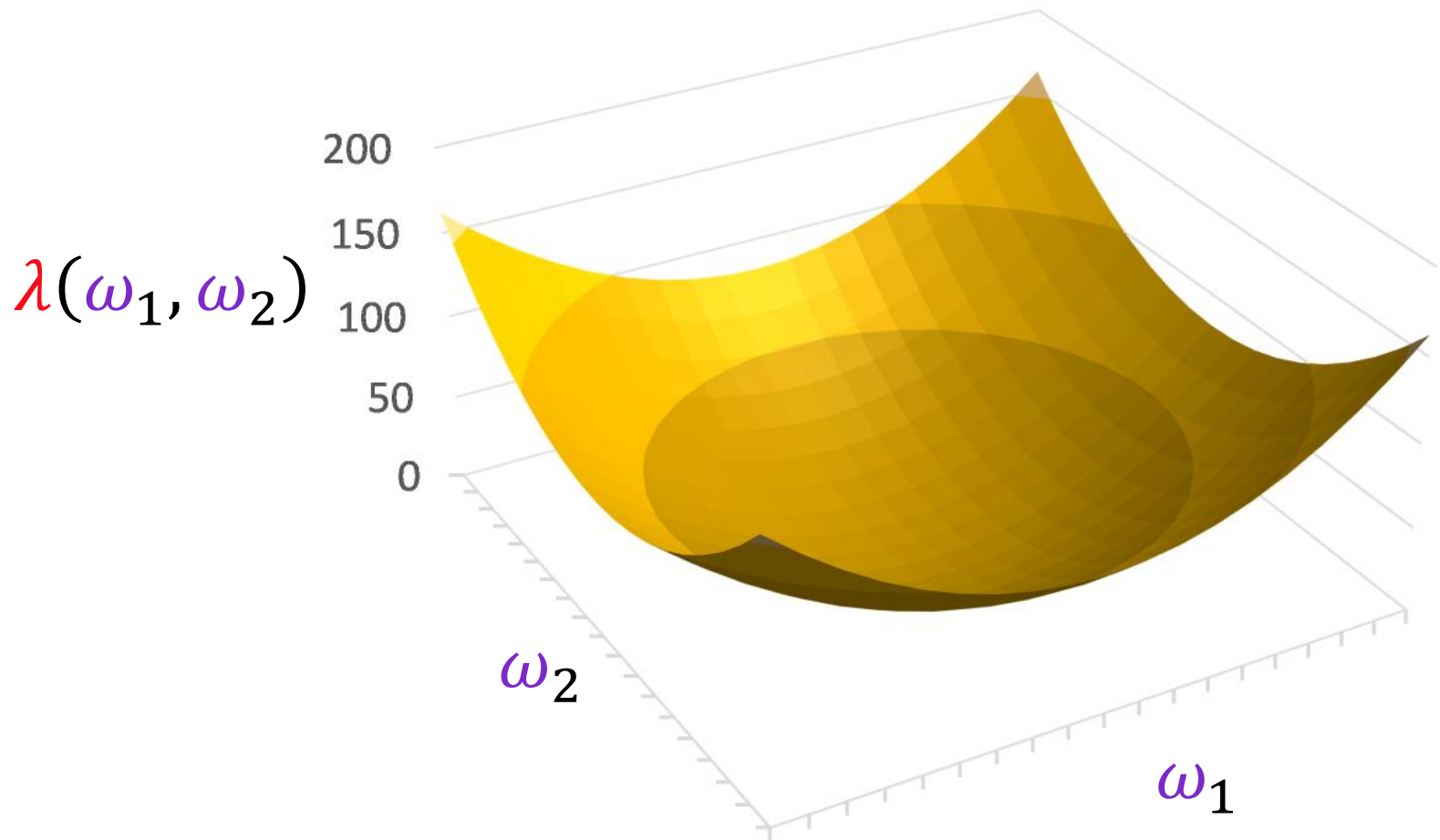
$$E(f) = \underbrace{\int_0^{2\pi} (f(x) - d(x))^2 dx}_{\text{likelihood (data)}} + c \cdot \underbrace{\int_0^{2\pi} (\nabla f(x))^2 dx}_{\text{prior}}$$

Frequency response



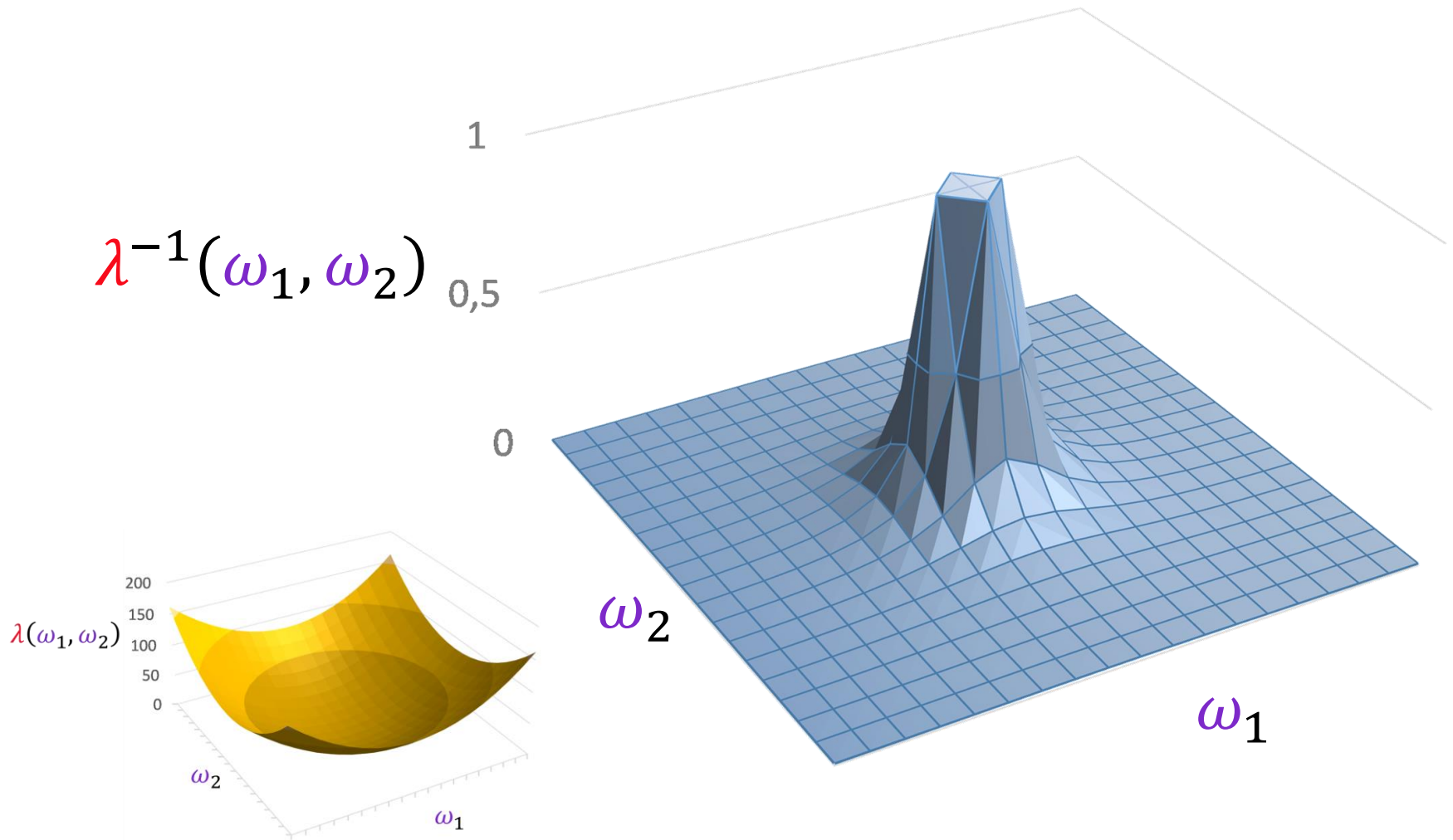
Visualization (2D)

Eigenvalues grow with 2D frequencies



Visualization (2D)

Dampening of high frequencies (2D)



Insights

The following three things are identical

- Maximum a priori reconstruction with $\|\nabla f\|^2$ -prior
- Dampening of frequency spectrum with

$$\frac{1}{1 + \|\omega\|^2}$$

frequency response (ω = frequency)

Differential Regularization as GP

Kernel interpretation

- Assume f is given as Fourier series

$$f(\mathbf{x}) = \sum_{\boldsymbol{\omega} \in \mathbb{Z}^2} z_{\boldsymbol{\omega}} e^{-i\boldsymbol{\omega}\mathbf{x}}$$

- Derivative ∇f has Fourier series

$$\nabla f(\mathbf{x}) = \sum_{\boldsymbol{\omega} \in \mathbb{Z}^2} -i\boldsymbol{\omega} \cdot z_{\boldsymbol{\omega}} e^{-i\boldsymbol{\omega}\mathbf{x}}$$

- Thus

$$\kappa(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = \boldsymbol{\omega}_1 \cdot \boldsymbol{\omega}_2$$

- Spatial kernel via inverse FT
 - See [Rasmussen & Williams 2004] for details

Better Inference

Go full Bayesian

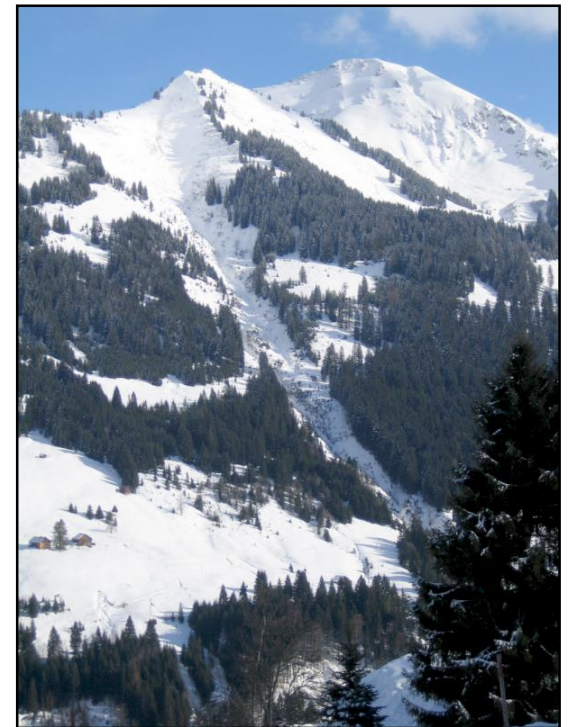
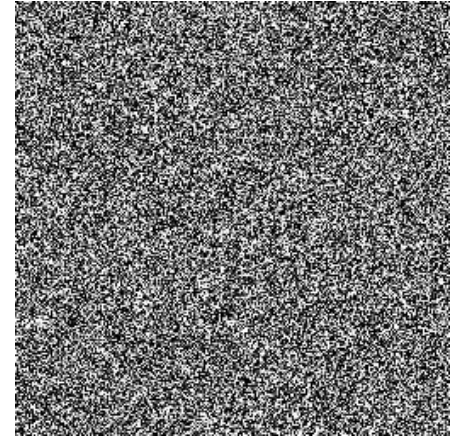
- We usually have hyperparameters
 - Strength of the regularizer
 - Properties of the kernel
 - E.g. parameter a in RBF-kernel
 - Determines spectral properties / smoothing
- We can just marginalize over everything
 - Regularizer weight
 - Kernel parameters
- As everything is Gaussian, the marginal likelihood can be computed
 - Integration over “all models” can be done in closed-form
 - Averaging still exponential in number of parameters

Example 2: Fractal Brownian Motion

Brown Noise

Fractal noise

- Uniform i.i.d. noise rare in nature
- „Fractal Brownian Motion“ (FBM):
Noise with decaying Power-Spectrum



FBM Noise (1D)

Formal Definition

- Function

$$f: [0, 2\pi]^d \rightarrow \mathbb{R}$$

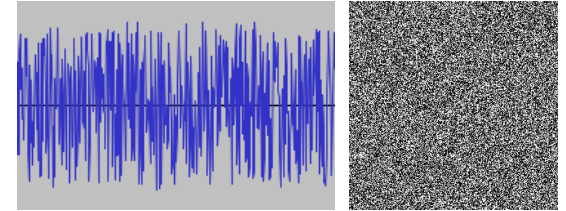
- Gaussian distribution on such functions

- Gaussian process

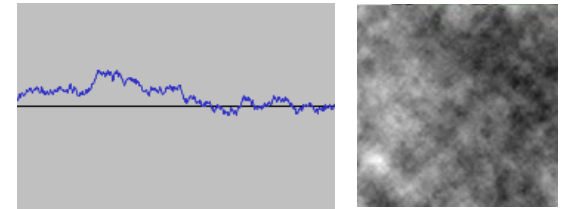
- Fourier spectrum

- Each Fourier-coefficient is i.i.d. Gaussian
- Mean $\mu_\omega = 0$
- Variance is $\sigma_\omega^2 = \frac{1}{\omega^{2h}}$ for $h > 0$

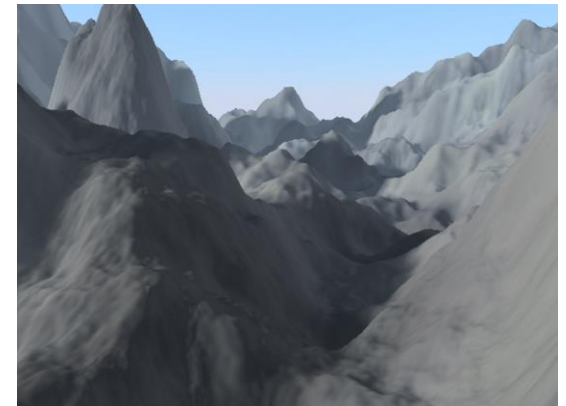
white noise



FBM noise



“fractal landscape”



FBM Noise (1D)

Fourier synthesis

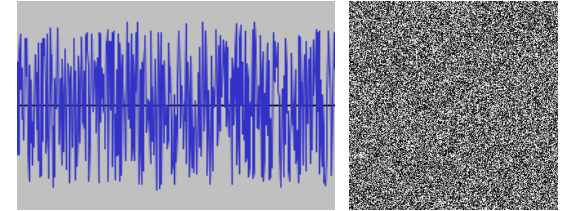
$$f(x) = \sum_{\omega=1}^{\infty} a_{\omega} \sin(\omega x + \varphi_{\omega}),$$

$$a_{\omega} \sim \mathcal{N}_{\mu=0, \sigma=\frac{1}{k^h}},$$

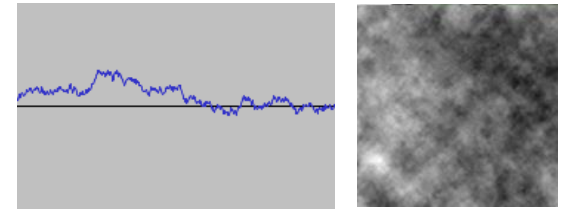
$$\varphi_{\omega} \sim \text{rnd}[0, 2\pi]$$

$h > 0$ („fractal exponent“)

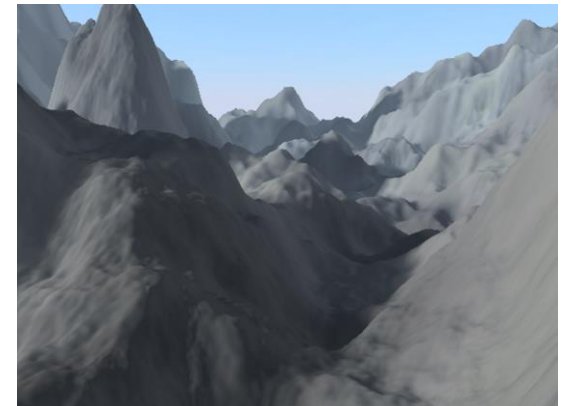
white noise

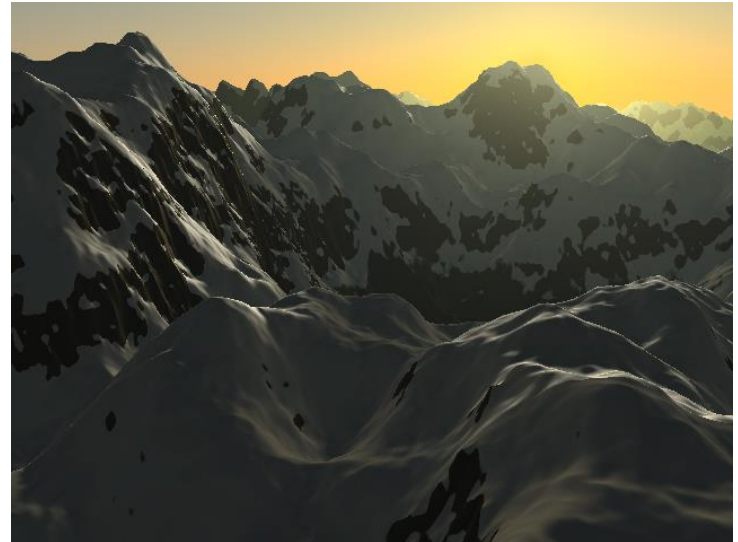
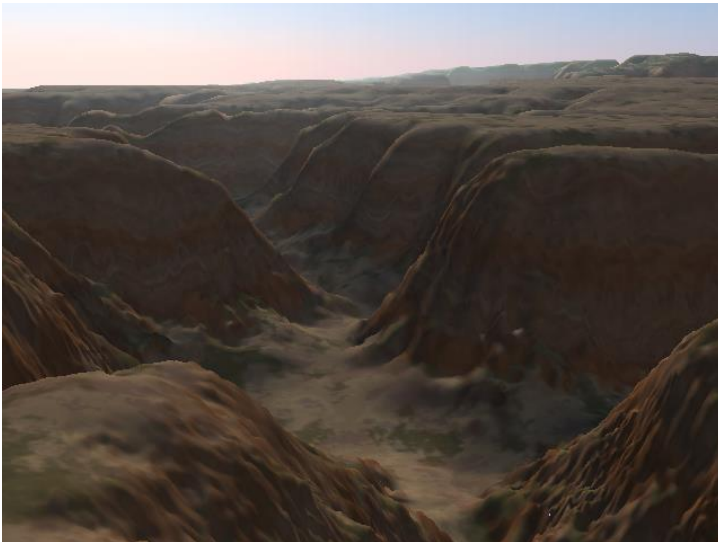
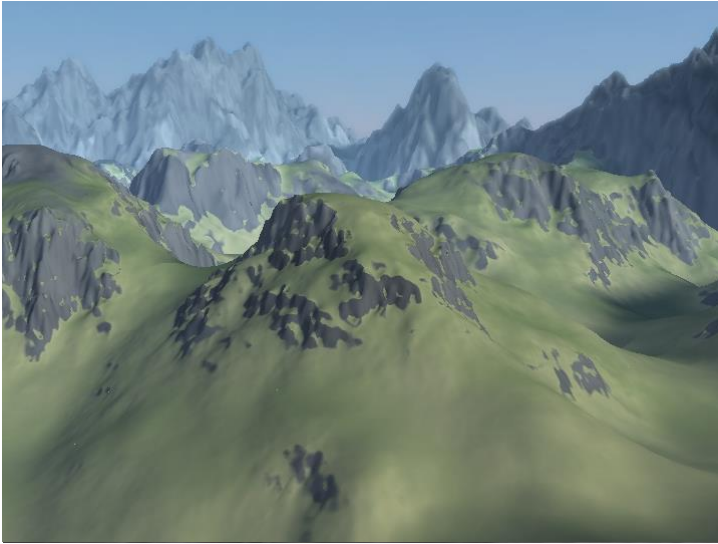


FBM noise

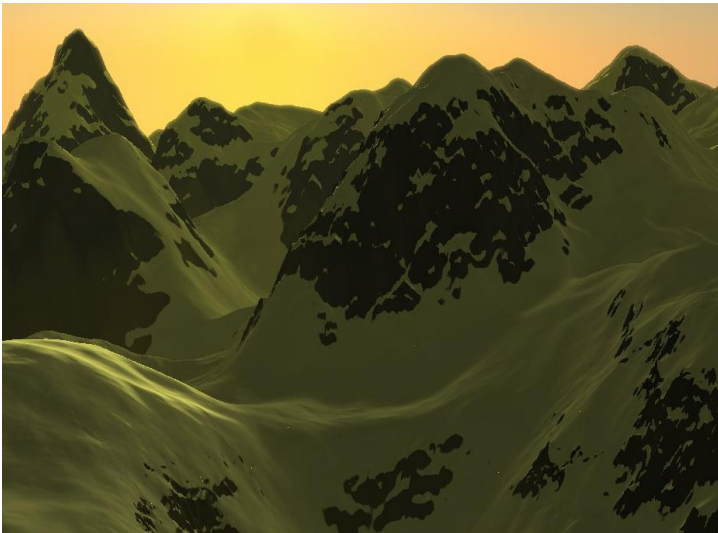


“fractal landscape”





[joint work with Martin Bokeloh, 2006]



[joint work with Martin Bokeloh, 2006]

Deep Networks

**Sounds all nice, but how
is this relevant to demystifying deep learning?**

GPs & DNNs

Three Examples

- Wide network layers approach GPs
- The neural tangent kernel (NTK)
- A model for the “double-descent” phenomenon

Two Layers Network [Neal 1996]

Considering Preactivations (and next. Layer)

$$y_i^{(l)} = \sum_{j=1}^{d_l} w_{i,j}^{(l)} \cdot f_i^{(l-1)}(\mathbf{x})$$

$$f_i^{(l)} = \varphi(y_i^{(l)})$$

- Assuming that weights are initialized i.i.d.
 - Mean zero, typ. normal distributed (not important)
- Each $y_i^{(l)}$ is the sum of i.i.d. random variables
 - Converges to normal distribution (CLT)
 - Assuming distributions with mean and variance
 - Mean zero
 - Variance grows linearly with d_l

Layer Output

In the infinite-width-limit

- Networks yield Gaussian processes **at initialization**

Speaking of initialization...

$$y_i^{(l)} = \sum_{j=1}^{d_l} \frac{1}{\sqrt{d_l}} \mathbf{w}_{i,j}^{(l)} \cdot f_i^{(l-1)}(\mathbf{x})$$

- **LeCun-Initialization**
 - Normal distributed initialization
 - Normalize output to unit variance
- **Our equation**
 - We assume $\mathbf{w}_{i,j}^{(l)} \sim \mathcal{N}_{0,1}$ and normalize by $\frac{1}{\sqrt{d_l}}$ explicitly

The Neural Tangent Kernel

Something very simple

- Let $f(\mathbf{x}; \mathbf{W})$ the full multi-layer network
 - f is a function of inputs \mathbf{x} and weights \mathbf{W}
 - f is highly non-linear
 - Using LeCun-Initialization \mathbf{W}_0

- Taylor-approximation

$$f(\mathbf{x}; \mathbf{W}) \approx f(\mathbf{x}; \mathbf{W}_0) + \nabla_{\mathbf{W}} f(\mathbf{x}_0; \mathbf{W}_0) \cdot (\mathbf{W} - \mathbf{W}_0)$$

- Linearized version is a Gaussian process
 - Non-linear feature map in \mathbf{x}
 - Linear in weights \mathbf{W}
 - Training amounts to solving a linear system
 - Think of fitting non-linear basis functions w/linear weights

The Neural Tangent Kernel

First-order Taylor

- NTK-Approximation

$$f(\mathbf{x}; \mathbf{W}) \approx f(\mathbf{x}; \mathbf{W}_0) + \nabla_{\mathbf{W}} f(\mathbf{x}_0; \mathbf{W}_0) \cdot (\mathbf{W} - \mathbf{W}_0)$$

This can't be good, can it?

- Linear approximation only valid close to \mathbf{W}_0

Now: Infinite width limit

- Very wide networks:
weights \mathbf{W} change very little during training
 - Empirical finding (for now)
- Seems to converge

It Does Converge...

Proof sketch:

- Measuring non-linearity as

$$\frac{\|H_f\|^2}{\|\nabla f\|^2}$$

- Ratio of Hessian H_f to Gradient (Jacobian) ∇f

- Chain rule: replace $f(x) \rightarrow f(\alpha x)$ leads to

$$f'(x) \rightarrow \alpha f'(\alpha x) \quad f'(x) \rightarrow \alpha^2 f''(\alpha x)$$

- Multi-variate

$$f(\mathbf{W}) \rightarrow f(\alpha \mathbf{W}) \text{ leads to } \frac{\alpha^2 \|H_f(\alpha \mathbf{W})\|^2}{\alpha \|\nabla f(\alpha \mathbf{W})\|^2}$$

Layer scaling

Deep Network Layer

$$y_i^{(l)} = \sum_{j=1}^{d_l} \frac{1}{\sqrt{d_l}} \mathbf{w}_{i,j}^{(l)} \cdot f_i^{(l-1)}(\mathbf{x})$$

Going wide...

- We take $d_l \rightarrow \infty$
- Thus, we take $\alpha = \frac{1}{\sqrt{d_l}} \rightarrow 0$
- Network becomes approximately linear
 - Converges to linear for infinite width
 - One can compute the limit kernel analytically!

So, All Linear Regression Then?

Practical findings

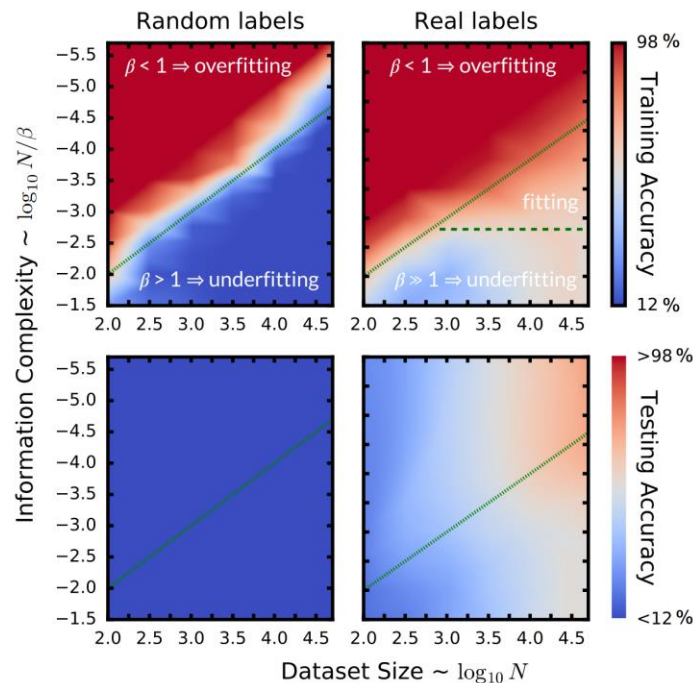
- Reasonably good performance
 - Better than standard kernels such as RBF
- But: still below finite-width DNNs
 - Fully-connected NTK networks quoted 7% below standard
 - Best convolutional NTK network I've seen performs at "AlexNet"-Level on CIFAR-10
- Finite width seems to be important

Strong theoretical tool

- GPs are much easier to understand than DNNs

(Deep?) Double Descent

The Generalization Conundrum

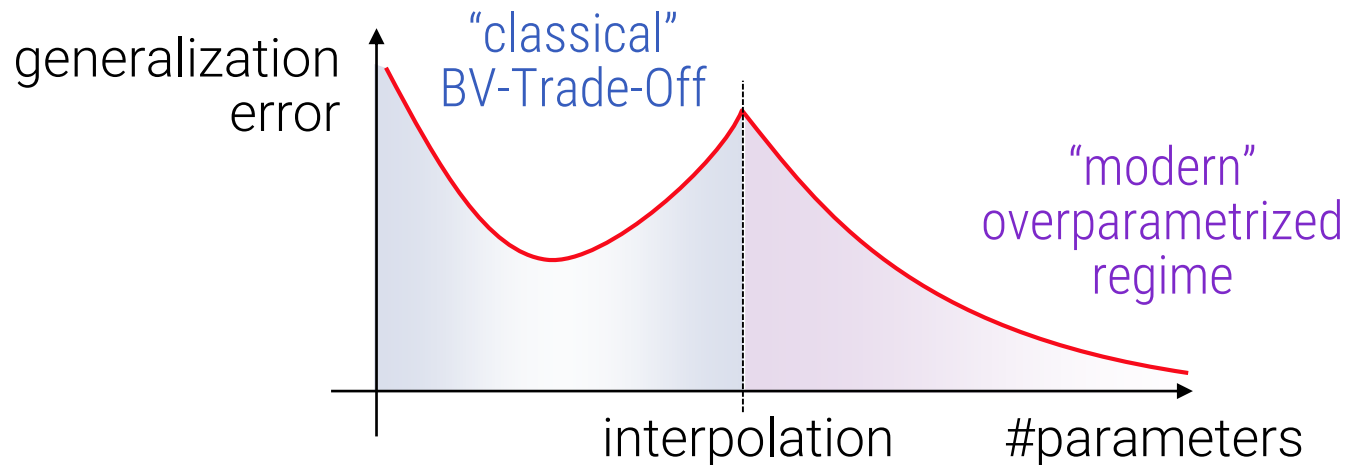


Generalization behavior is weird

- Networks are able to fit random data
- Still generalize on “reasonable” data

[A. Achille, S. Soatto: Emergence of Invariance and Disentanglement in Deep Representations
Journal of Machine Learning Research 18 (2018) 1-34. (Figure 1, CC-BY 4.0)]

The “Double-Descent”



Closer inspection: “Double-Descent” [Belkin 2019]

- Underparametrized regime:
“Classical” Bias-Variance-Trade-Off
- Overparametrized regime:
Error reduced again (maybe even lower)

Overparametrized Double-Descent

Let's assume, we just do function fitting

- Searching function

$$f: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

- Basis functions

$$b_1, \dots, b_k: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

- Ansatz

$$f(\mathbf{x}) = \sum_{i=1}^k \lambda_i b_i(\mathbf{x})$$

- With Gaussian prior $p(\lambda_i) = \mathcal{N}_{0, \sigma_i}$

Overparametrized Double-Descent

Example: Image reconstruction

- Fourier basis functions

$$b_{\omega_1, \omega_2} = \exp(i(\omega_1 x_1 + \omega_2 x_2))$$

- Prior

$$p(\lambda_{\omega_1, \omega_2}) = \mathcal{N}_{0, \sigma_i = \omega_1^2 + \omega_2^2}$$

Overparametrized Double-Descent

Let's assume, we just do function fitting

- Searching function

$$f: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

- Basis functions

$$b_1, \dots, b_k: \mathbb{R}^d \rightarrow \mathbb{R}^m$$

- Ansatz

$$f(\mathbf{x}) = \sum_{i=1}^k \lambda_i b_i(\mathbf{x})$$

- With Gaussian prior $p(\lambda_i) = \mathcal{N}_{0, \sigma_i}$

We Now Do Function Fitting

“Realistic” Numerics

- We replace the prior by uniform prior

$$p(\lambda_i) = \mathcal{N}_{0,1}$$

- Rescale basis functions accordingly

$$b'_i = \frac{1}{\sigma_i} b_i$$

- Yields same solution
 - That is also what a kernel feature map would do

We Now Do Function Fitting

Least-Squares-Fitting

- Given training data $\mathbf{x}_j, \mathbf{y}_j, j = 1, \dots, n$
- We solve

$$\arg \min_{\lambda_1, \dots, \lambda_k} \sum_{j=1}^n \left\| \sum_{i=1}^k \lambda_i b_i(\mathbf{x}_j) - \mathbf{y}_j \right\|^2$$

- And pick $\lambda_1, \dots, \lambda_k$ with minimal $\|\lambda\|^2$ in case of ambiguity
 - An SVD-solver (pseudo inverse) would do this
 - Most (mildly-regularized) numerical descent solvers would do this

What Do We Get?

Now, change parameters

- We pick only a subset S of basis functions

$$S \subset B = \{b_1, \dots, b_k\}$$

- Underparametrized

- $\#S \ll n$: underfitting possible

- Interpolation

- $\#S \approx n$: exact fit to the data with random S
- Results might be rather bad

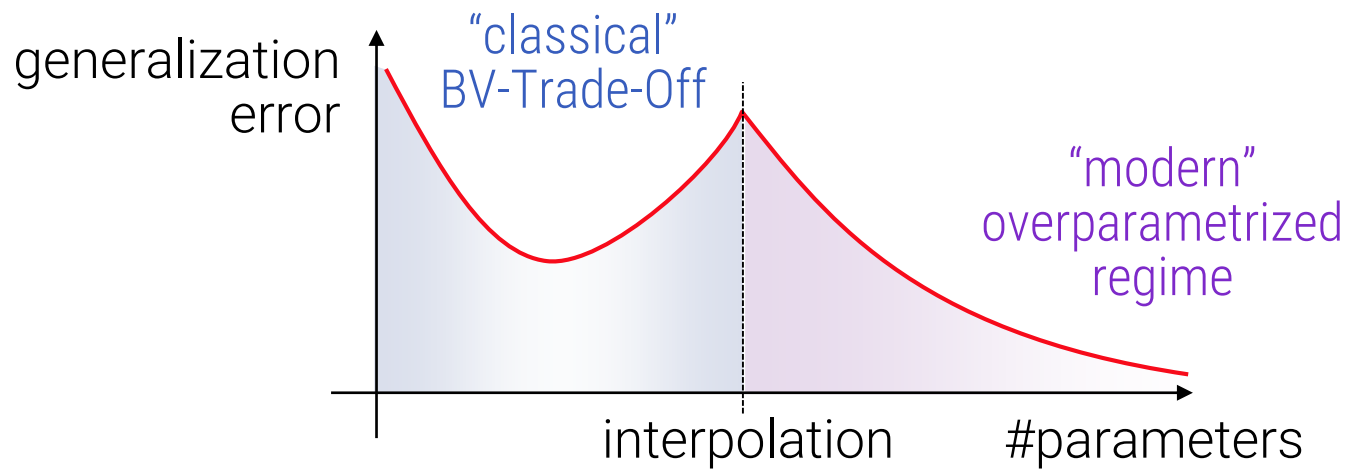
- Overparametrized

- Convergence to regularized solution
- Many solutions, picking with minimal $\|\lambda\|^2$
- This leads to regularization!

“Classical”
BV-Trade-Off

Convergence
to regularized
(better) solution

Et Voilà!



Double Descent

Double-Descent in GPs with RFF

Conclusions

- **Simple least-squares fitting can double-descent**
 - Increase parameters
 - By adding more basis functions
 - Approximation first overfits, then gets better again
- **Deep networks can be approximated by GPs**
 - Belkin et al. discuss “Random Fourier Features” for approximating an RBF-kernel
 - NTK-view: Better approximation by increasing width
- **Not a complete explanation**
 - Only plausible hypothesis for effect structure

Sources on Double-Descent

C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals:

Understanding deep learning requires rethinking generalization. ICLR 2017.

<https://arxiv.org/pdf/1611.03530.pdf>

A. Achille, S. Soatto:

Emergence of Invariance and Disentanglement in Deep Representations.

Journal of Machine Learning Research 18 (2018) 1-34.

<https://arxiv.org/pdf/1706.01350.pdf>

M. Belkin, D. Hsu, S. Ma, S. Mandal:

Reconciling modern machine-learning practice and the classical bias–variance trade-off. Proc. of the National Academy of Sciences 116 (32), 15849-15854, 2019.

<https://arxiv.org/pdf/1812.11118.pdf>

P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak I. Sutskever:

Deep Double Descent: Where Bigger Models and More Data Hurt. ICLR 2020.

<https://openreview.net/forum?id=B1g5sA4twr>

Summary

Kernels – Virtual Euclidean Space

Feature maps w/kernels

- Map input into “deformed” feature space
- Kernels are the scalar product of the feature space
- Efficient handling of complex feature spaces

Gaussian processes

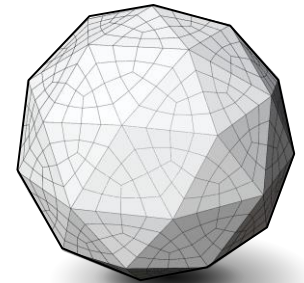
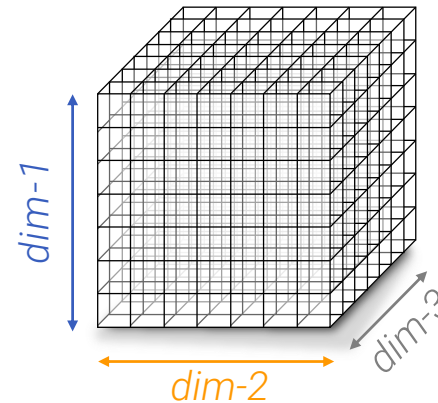
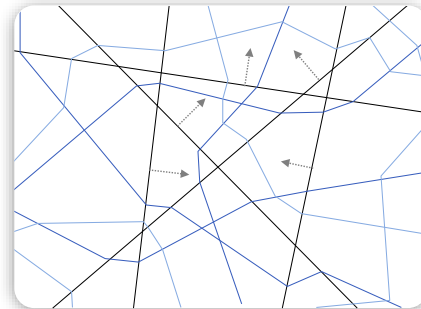
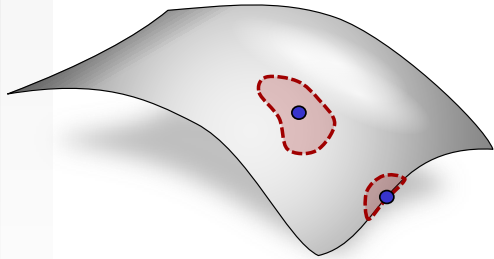
- Gaussian model on functions
- Covariance function could be interpreted as kernel

Analysis of DNNs

- Kernel / GP approximation provide models

Modelling 2

STATISTICAL DATA MODELLING



Chapter 10 Space

Video #10

Space

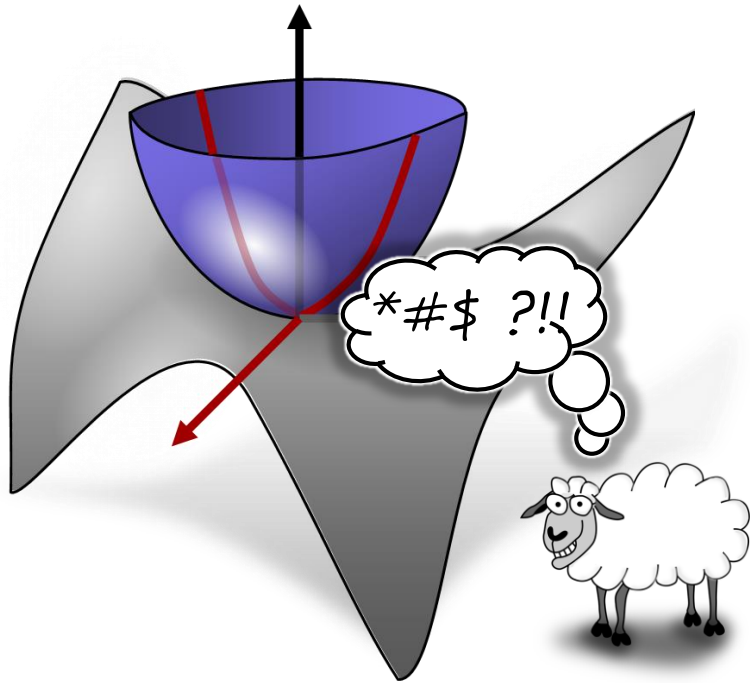
- **High-dimensional space**
& the curse of dimensionality
- **Kernels:** flat space extended
- **Manifolds:** curved space

Overview

Curved Space

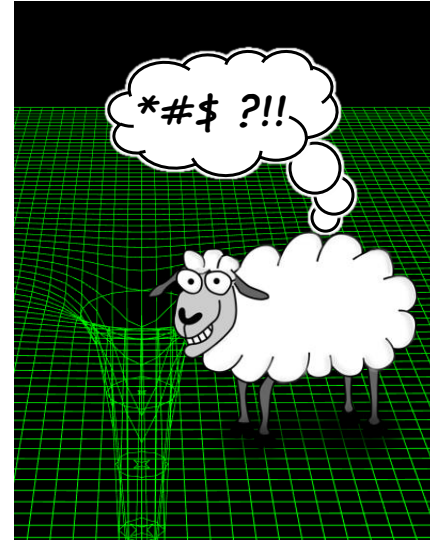
- Brief intro to concepts from differential geometry
 - Fundamental forms: Metric & Curvature
 - 2D and 3D Curves & Surfaces
 - Intrinsic geometry
- Applications to deep learning

Differential Geometry Intro



Embedded Geometry

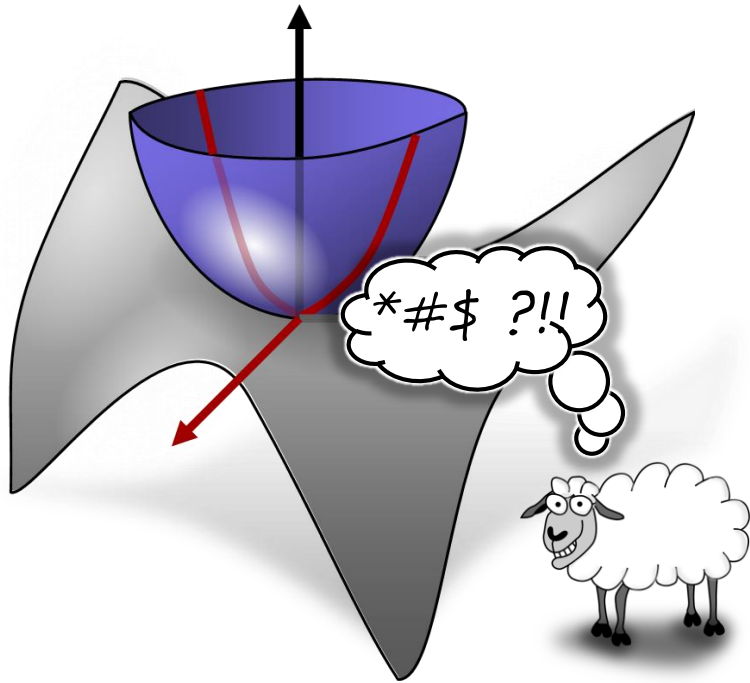
d -dim. Manifold embedded in \mathbb{R}^n
($d \leq n$)



Intrinsic Geometry

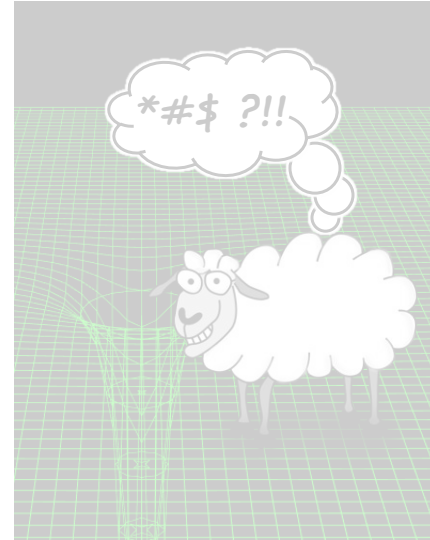
no ambient space
("general relativity")

Differential Geometry Intro



Embedded Geometry

d -dim. Manifold embedded in \mathbb{R}^n
($d \leq n$)



Intrinsic Geometry

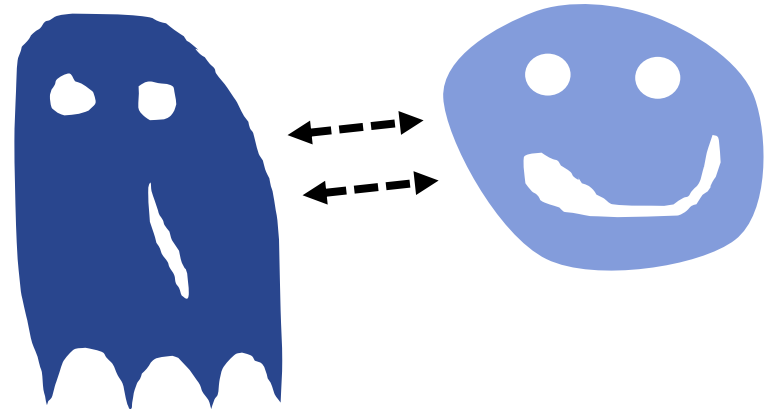
no ambient space
("general relativity")

Manifolds

Elementary Topology

Homeomorphism

- $h: X \rightarrow Y$
- h is bijective
- h is continuous
- h^{-1} exists and is continuous
- Basically, a continuous deformation



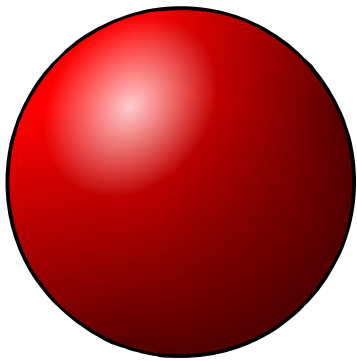
Topological equivalence

- Objects are topologically equivalent if there exists a homeomorphism that maps between them
- “Can be deformed into each other”

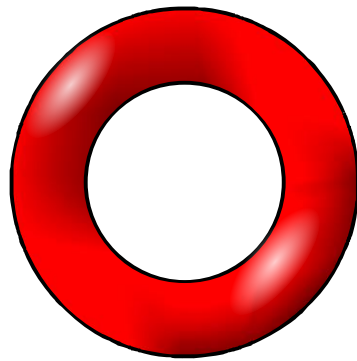
Surfaces of Volumes

Boundaries of volumes in 3D

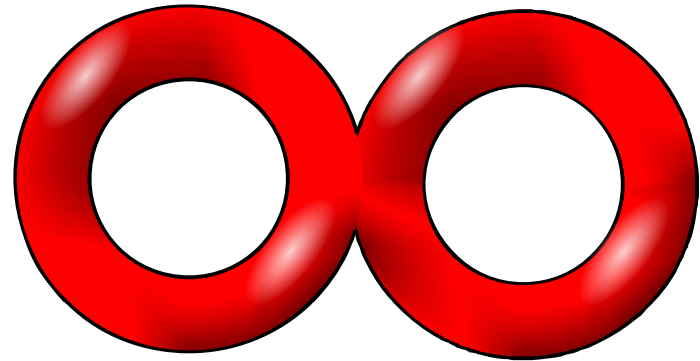
- Topological equivalence classes
 - Sphere
 - Torus
 - n-fold Torus
- Genus = number of tunnels



$g = 0$



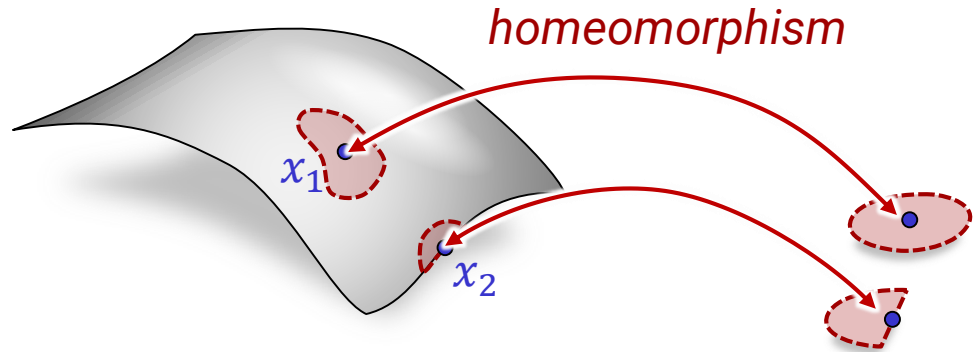
$g = 1$



$g = 2$

...

Manifold

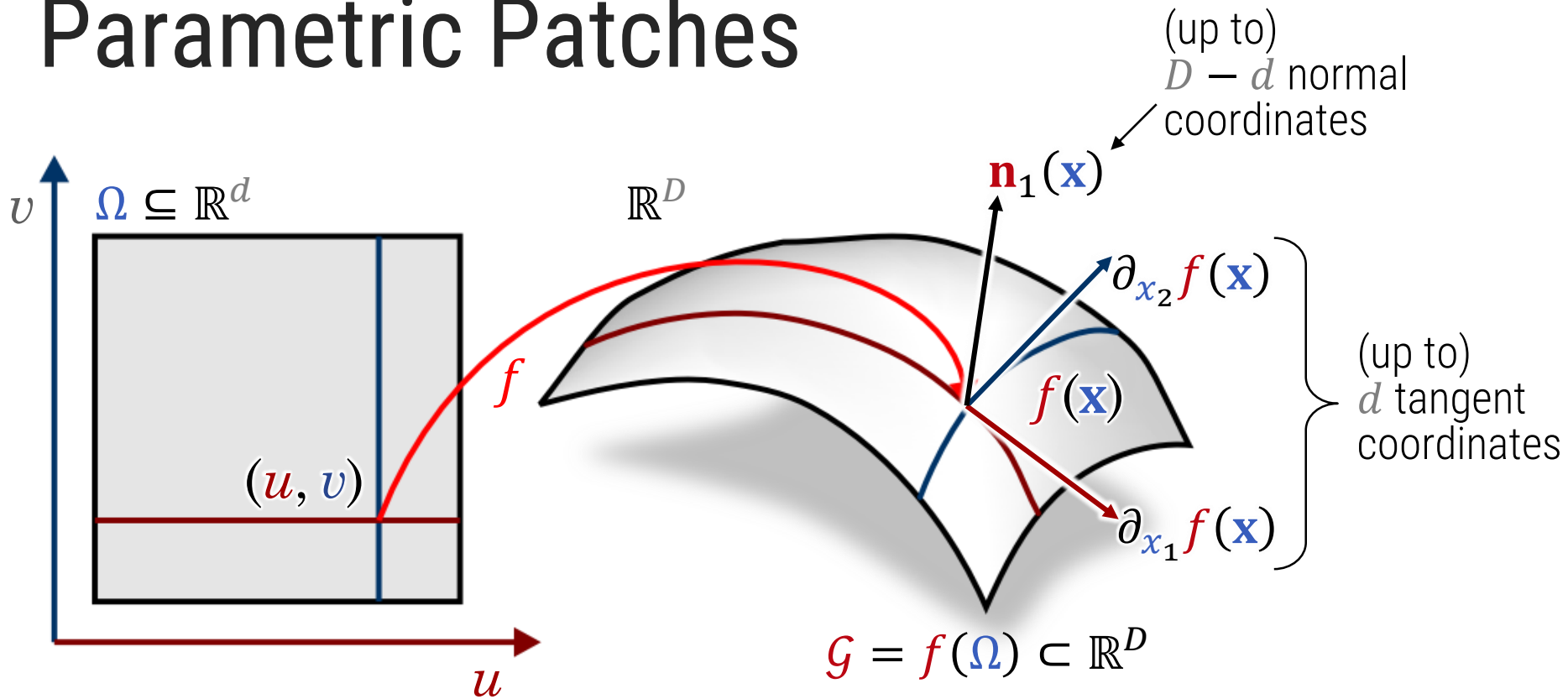


Definition: Manifold

- A d -manifold \mathcal{M} :
At every $\mathbf{x} \in \mathcal{M}$ there exists an ϵ -environment homeomorphic to a d -dimensional *disc*
 - “With boundary”: *disc* or *half-disc*

Parametric Functions

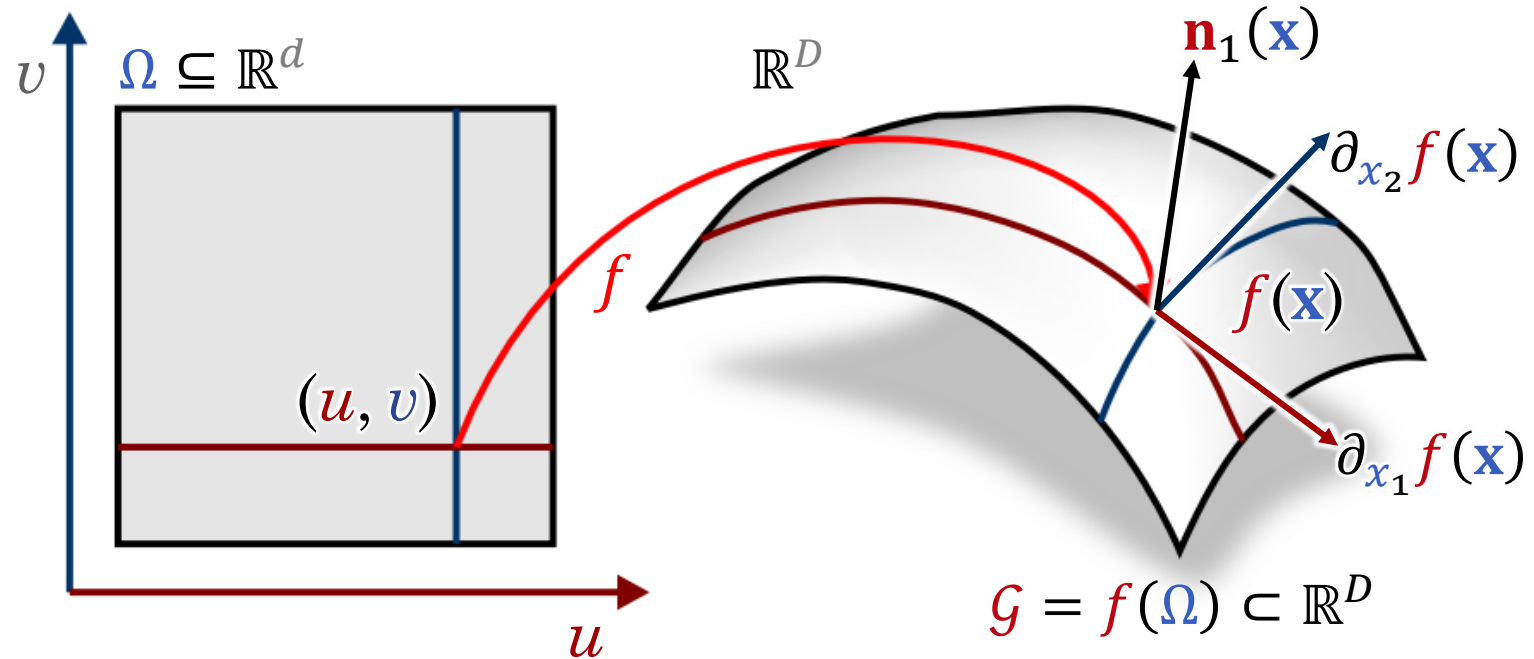
Parametric Patches



Parametric Patch

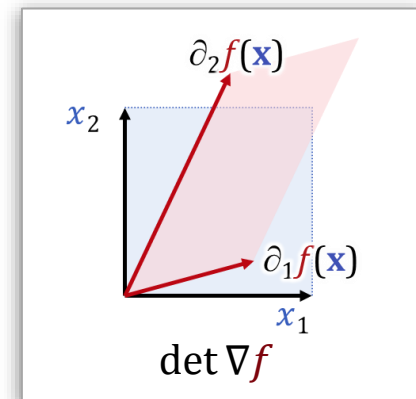
- Mapping $f: \mathbb{R}^d \supseteq \Omega \rightarrow \mathbb{R}^D$
 - Assumption: $f \in C^\infty$, Ω open
- Geometry $\mathcal{G} = f(\Omega)$

Regular Parametrizations

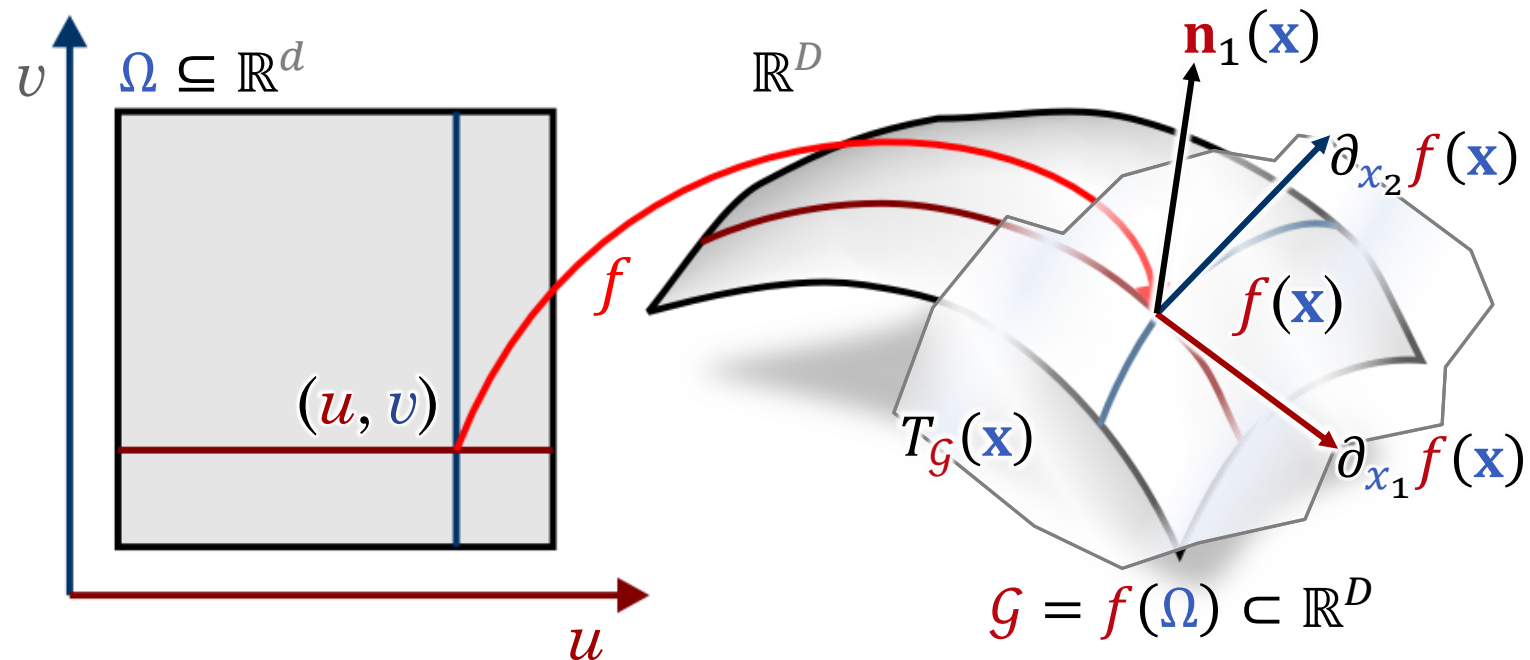


Regular Parametrization

- “Does not stop anywhere”
- Formally: $\forall \mathbf{x} \in \Omega: \det \nabla f (\nabla f)^T \neq 0$



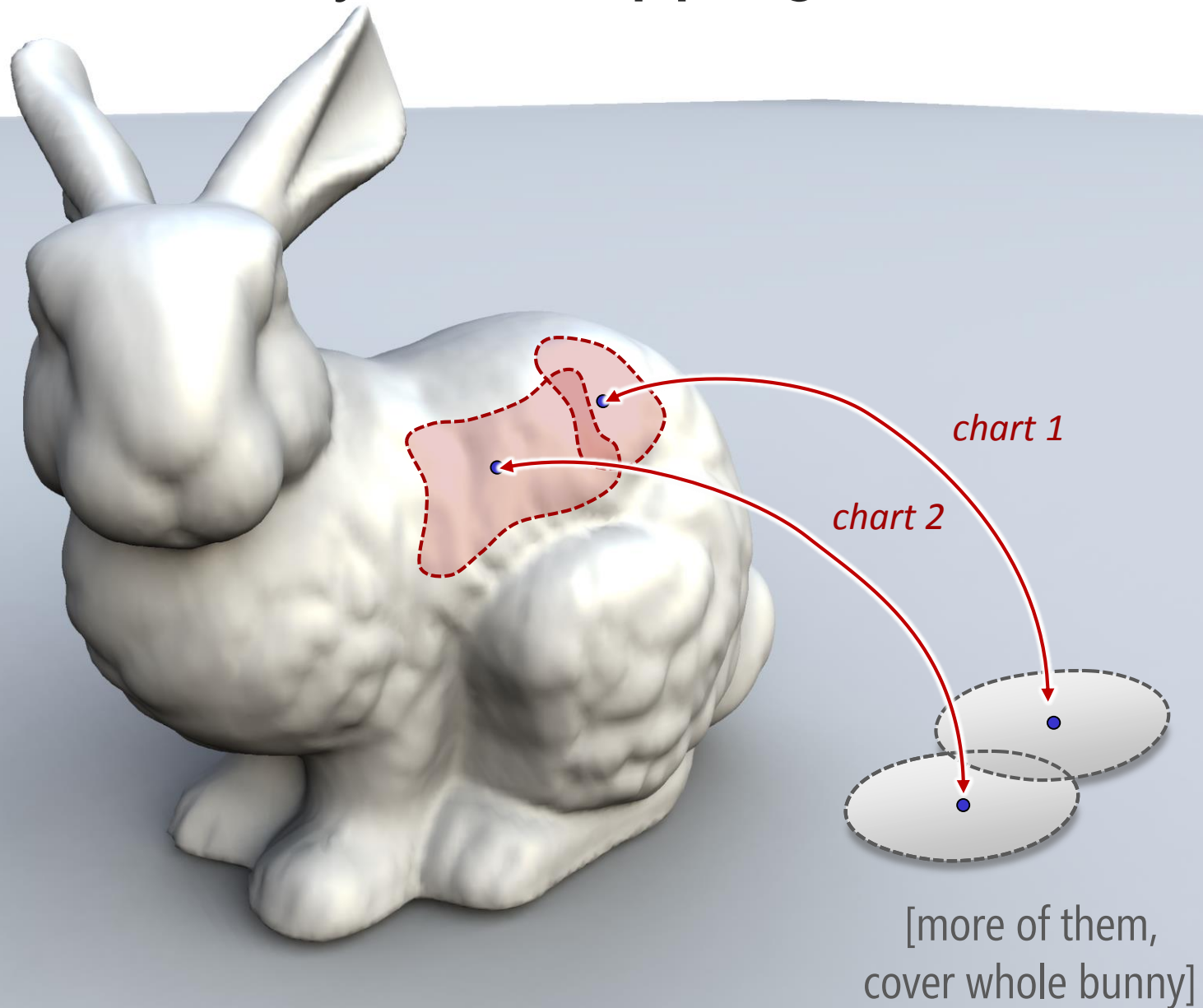
Tangent Space



Tangent Space

- Assume regular parametrization
- Tangent space $T_G(\mathbf{x}) = \text{span}(\partial_1 f(\mathbf{x}), \dots, \partial_d f(\mathbf{x}))$
- Vector space – as affine space: origin $f(\mathbf{x})$

Complex Geometry: Overlapping “Charts”



Examples

(Curves & Surfaces)

Parametric Curves

Parametric Curves:

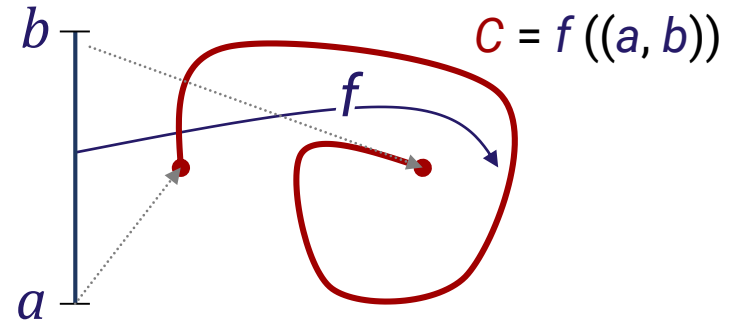
- A differentiable function

$$f: (a, b) \rightarrow \mathbb{R}^n$$

describes a *parametric curve*

$$C = f((a, b)), C \subseteq \mathbb{R}^n.$$

- Parametrization *regular*: $f'(t) \neq 0$ for all t
- *Unit-speed* parametrization: $\|f'(t)\| \equiv 1$



Tangents

Tangents / normals

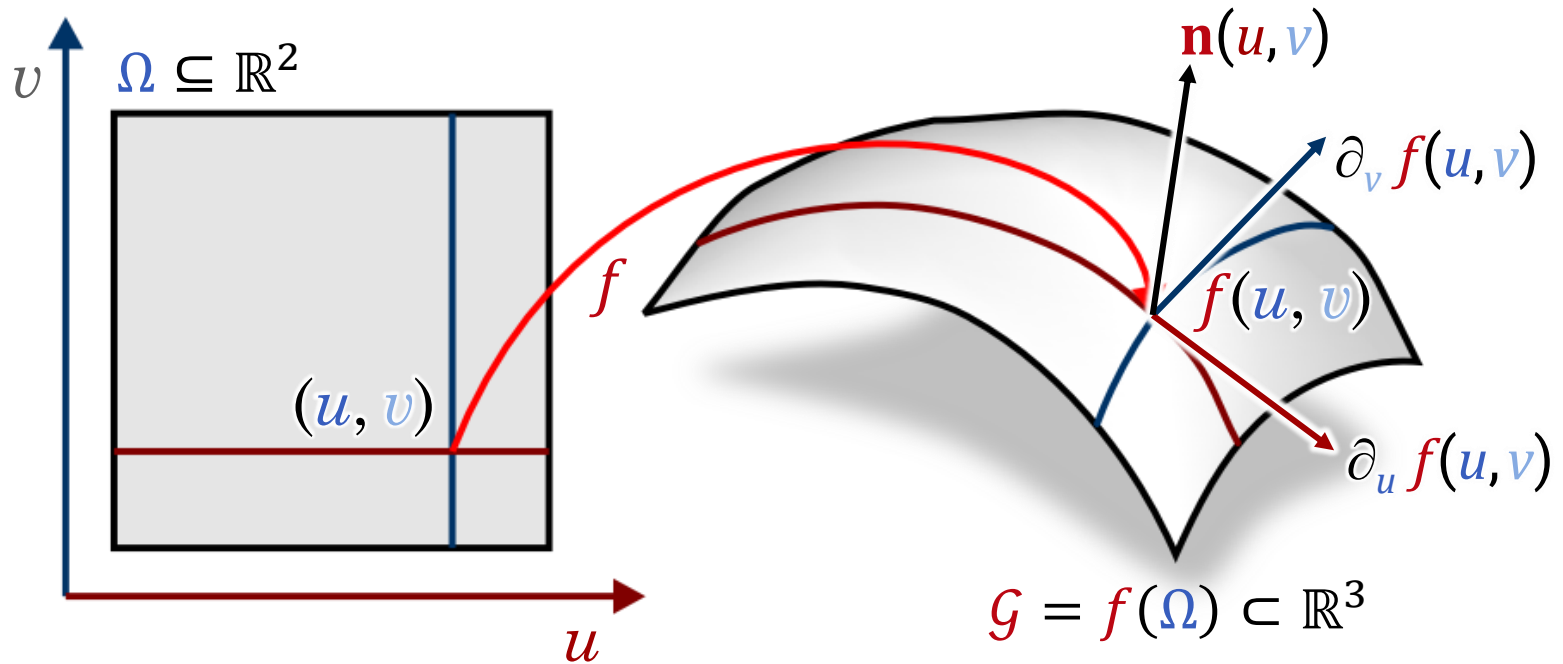
- Any curve $C \subseteq \mathbb{R}^n$: *unit tangent vector*

$$\text{tangent}(t) = \frac{f'(t)}{\|f'(t)\|}$$

- For curves $C \subseteq \mathbb{R}^2$: *unit normal vector*

$$\text{normal}(t) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \frac{f'(t)}{\|f'(t)\|}$$

Parametric Surfaces



Function $f(\mathbf{x}) = f(u, v) \rightarrow \mathbb{R}^3$

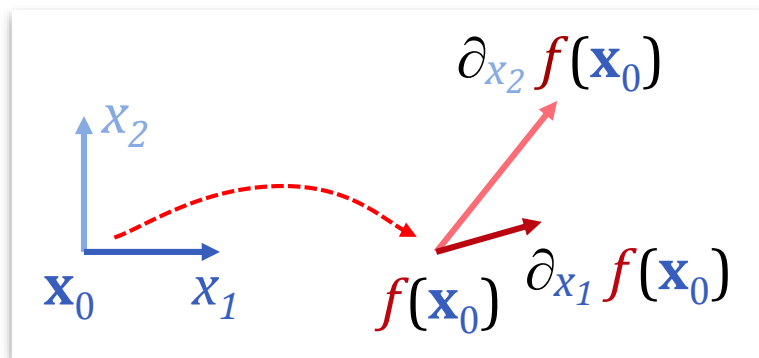
▪ Tangents: $\mathbf{u}(u, v) = \partial_u f(u, v),$

$\mathbf{v}(u, v) = \partial_v f(u, v)$

▪ Normal: $\mathbf{n}(u, v) = \frac{\partial_u f(u, v) \times \partial_v f(u, v)}{\|\partial_u f(u, v) \times \partial_v f(u, v)\|}$

The Metric Tensor

First Fundamental Form



First fundamental form a.k.a. metric tensor

- Regular parametric patch

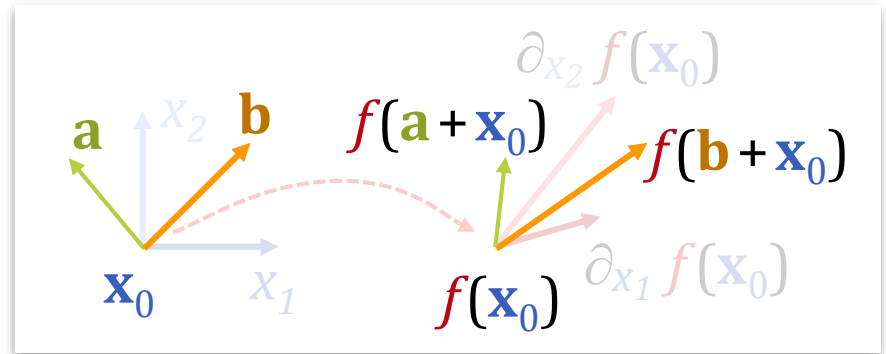
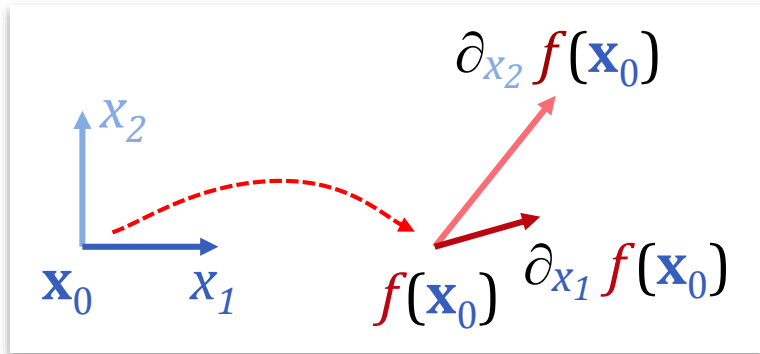
$$f: \mathbb{R}^d \supseteq \Omega \rightarrow \mathbb{R}^D$$

- f will distort angles and distances
 - Visible in the scalar product.

- First order Taylor approximation measures effect

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

First Fundamental Form



First Fundamental Form a.k.a. metric tensor

- First order Taylor approximation:

$$f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

- Scalar product of vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$:

$$\langle f(\mathbf{x}_0 + \mathbf{a}) - f(\mathbf{x}_0), f(\mathbf{x}_0 + \mathbf{b}) - f(\mathbf{x}_0) \rangle$$

$$\approx \langle \nabla f(\mathbf{x}_0) \cdot \mathbf{a}, \nabla f(\mathbf{x}_0) \cdot \mathbf{b} \rangle = \mathbf{a}^T \underbrace{\left(\nabla f(\mathbf{x}_0)^T \cdot \nabla f(\mathbf{x}_0) \right)}_{\text{first fundamental form } \mathbf{I}_f(\mathbf{x}_0)} \mathbf{b}$$

Surfaces (2-Manifolds)

First Fundamental Form

- Metric tensor is a $d \times d$ matrix
 - Symmetric, positive definite (regular parametrization)
 - Generalized scalar product

- Bilinear Form

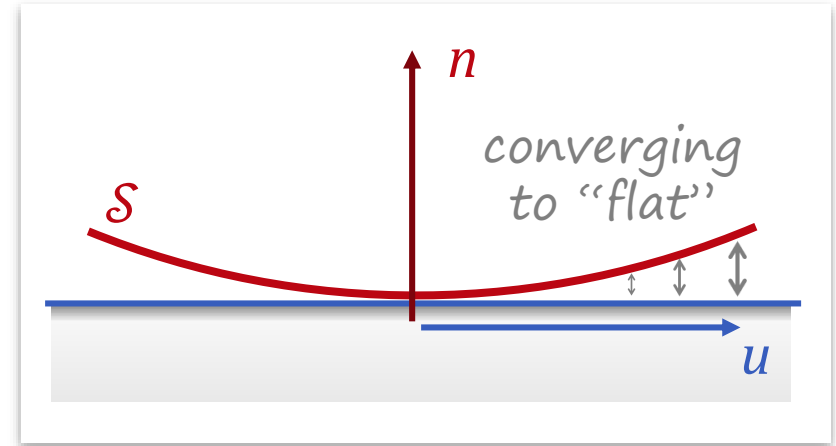
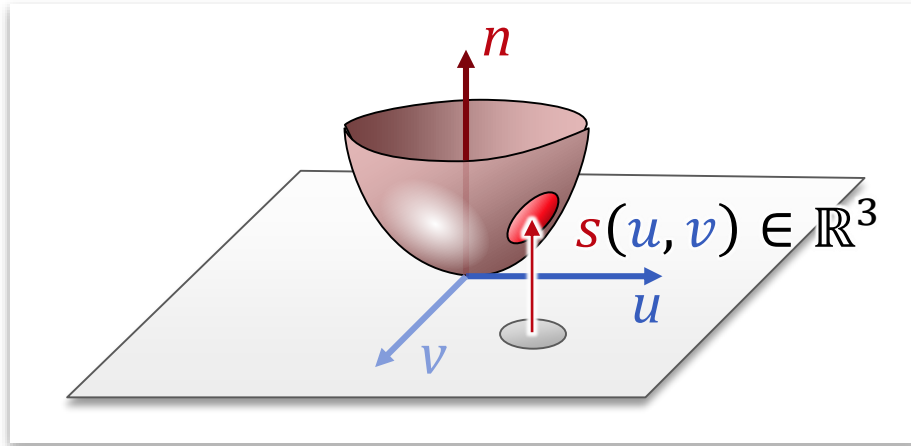
$$\mathbf{I}_f(\mathbf{a}, \mathbf{b}) := \mathbf{a}^T \cdot (\nabla f^T \cdot \nabla f) \cdot \mathbf{b}$$

- For surfaces ($d = 2$)

$$(\nabla f^T \cdot \nabla f) = \begin{pmatrix} \partial_u f \partial_u f & \partial_u f \partial_v f \\ \partial_u f \partial_v f & \partial_v f \partial_v f \end{pmatrix} =: \begin{pmatrix} E & F \\ F & G \end{pmatrix}$$

$$\mathbf{I}_f(\mathbf{a}, \mathbf{b}) = E a_1 b_1 + F(a_1 b_2 + a_2 b_1) + G a_2 b_2$$

Remark



First fundamental form

- Property of the parametrization
 - Does not characterize surface itself
 - Can always find parametrization with $\mathbf{I}_f =$ identity matrix
 - Local orthogonal tangent-frame
- Higher-order derivatives capture geometry
 - Derivative of first fundamental form

Examples

(Curves & Surfaces)

Length of a Curve

The length of a curve

- The length of a regular curve C is defined as:

$$\begin{aligned}\text{length}(C) &= \int_a^b \|f'(t)\| dt \\ &= \int_a^b \sqrt{\det \mathbf{I}_f(t)} dt\end{aligned}$$

- Independent of the parametrization
 - Proof: integral transformation theorem
- $\text{length}(C) = |b - a|$ for a unit-speed parametrization

Surface Area

Surface Area

- Patch \mathcal{S}

$$s: \mathbb{R}^2 \supseteq \Omega \rightarrow \mathbb{R}^3$$

- Integrate over constant function

$$\mathcal{S} \ni \mathbf{y} \mapsto 1$$

over surface

- Then apply integral transformation theorem:

$$\begin{aligned} \text{area}(\mathcal{S}) &= \int_{\Omega} \sqrt{\det \mathbf{I}_s(t)} d\mathbf{x} \\ &= \int_{\Omega} \|\partial_u s(\mathbf{x}) \times \partial_v s(\mathbf{x})\| d\mathbf{x} \end{aligned}$$

Curvature

(of curves)

Curvature

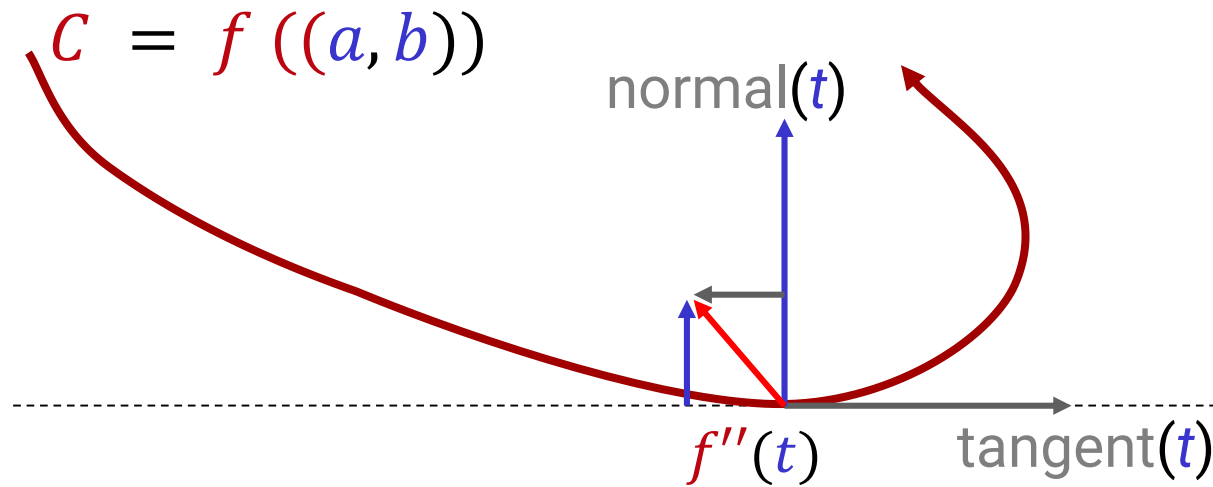
Curvature:

- First derivatives:
 - Curve direction / speed of movement
- Curvature:
 - Encoded in 2nd order information

Why not just use f'' ?

- Problem: Depends on parametrization
 - Different velocity yields different results
- Need to distinguish acceleration...
 - ...in tangential and
 - ...non-tangential directions.

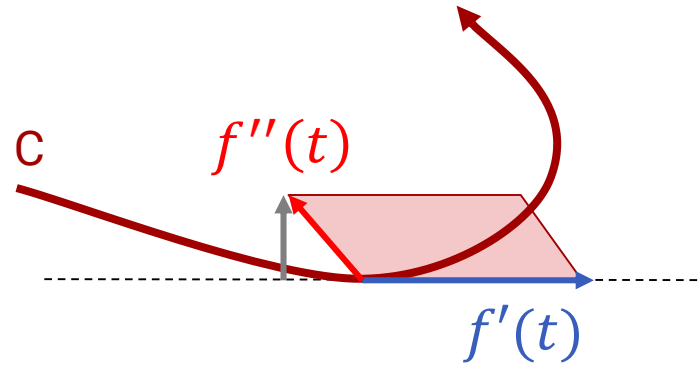
Curvature & 2nd Derivatives



Definition of curvature

- Need non-tangential component of f''
 - Project on normal
- Ignore accelerating/slowing down
 - Normalize speed

Space Curves



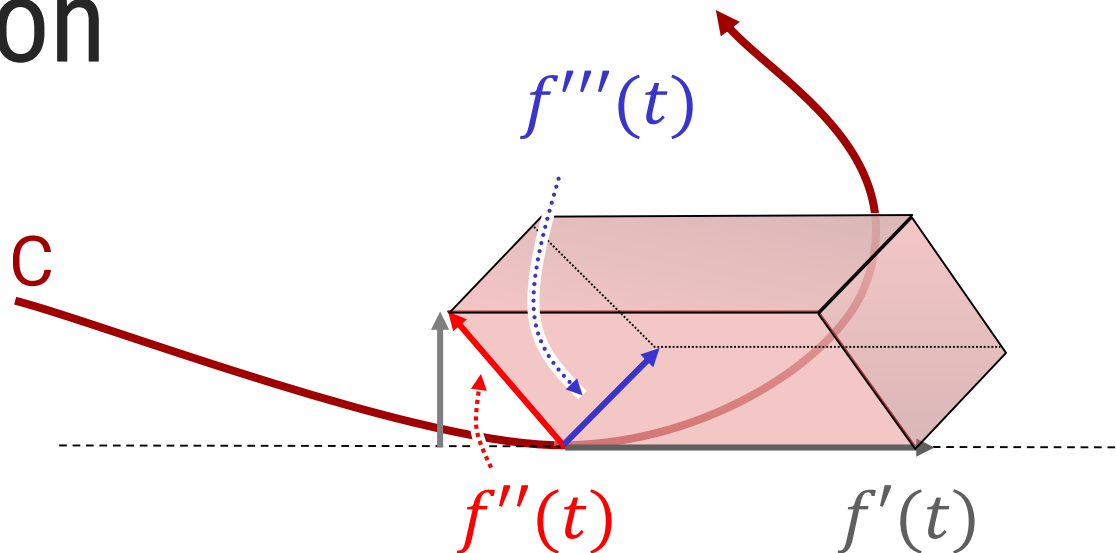
Curvature of a curve $C \subseteq \mathbb{R}^3$

- Curvature defined as

$$\kappa(t) = \frac{\|f'(t) \times f''(t)\|}{\|f'(t)\|^3}$$

- Assuming *regular parametrization*
 - f' does not vanish

Torsion



$$\tau(t) = \frac{f'(t) \times f''(t) \cdot f'''(t)}{\|f'(t) \times f''(t)\|^2} = \frac{\det(f'(t), f''(t), f'''(t))}{\|f'(t) \times f''(t)\|^2}$$

Definition *torsion* of f at t

- Curve $C \subseteq \mathbb{R}^3$
- Regular parametrization
- Non-zero curvature

Theorem

Fundamental Theorem of Space Curves

- Two curves $C \subseteq \mathbb{R}^3$
 - *unit speed parameterized*
 - *identical, positive curvature* ($\kappa > 0$)
 - and *identical torsion*

are identical up to a rigid motion.

- In the 2D case, torsion is not required
 - Would be zero everywhere

Curvature (of surfaces)

Second Fundamental Form

Again: Missing Information

- First fundamental form measures only length changes.
- Cylinder looks like a flat sheet

Complete (extrinsic) geometry

- Measure curvature of a surface as well.
- Requires second order information
 - Anything first order is inherently “flat”

Second Fundamental Form

Basic Idea

- Compute second derivative vectors
- Project in normal direction
 - Remove tangential acceleration

Second Fundamental Form

Definition

- Regular parametrization $s: \mathbb{R}^2 \supseteq \Omega \rightarrow \mathbb{R}^3$
- *Second fundamental* form of s :

$$\mathbb{I\!I}_s(\mathbf{x}_0) = \begin{pmatrix} \partial_{uu}s(\mathbf{x}_0) \cdot \mathbf{n}(\mathbf{x}_0) & \partial_{uv}s(\mathbf{x}_0) \cdot \mathbf{n}(\mathbf{x}_0) \\ \partial_{uv}s(\mathbf{x}_0) \cdot \mathbf{n}(\mathbf{x}_0) & \partial_{vv}s(\mathbf{x}_0) \cdot \mathbf{n}(\mathbf{x}_0) \end{pmatrix} = \begin{pmatrix} e & f \\ f & g \end{pmatrix}$$

Notation as bilinear form

$$\mathbb{I\!I}_s(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T \begin{pmatrix} \partial_{uu}s \cdot \mathbf{n} & \partial_{uv}s \cdot \mathbf{n} \\ \partial_{uv}s \cdot \mathbf{n} & \partial_{vv}s \cdot \mathbf{n} \end{pmatrix} \mathbf{b}$$

Remark: Christoffel Symbols

Second fundamental form

$$\mathbf{II} = \begin{pmatrix} \partial_{uu}\mathbf{S} \cdot \mathbf{n} & \partial_{uv}\mathbf{S} \cdot \mathbf{n} \\ \partial_{uv}\mathbf{S} \cdot \mathbf{n} & \partial_{vv}\mathbf{S} \cdot \mathbf{n} \end{pmatrix}$$

- Extrinsic curvature
- Projection on normal – measure only curvature away from tangent space

Full picture

- We can measure tangential curvature, too
- Useful for intrinsic view (non-embedded manifolds)
- “Christoffel Symbols”

Full Second-Order Expansion

Parametric surface

$$s: \mathbb{R}^2 \supseteq \Omega \rightarrow \mathbb{R}^3$$

Second order representation

$$\begin{aligned}\partial_{uu} s &= \Gamma_{11}^1 \mathbf{u} + \Gamma_{11}^2 \mathbf{v} + e \mathbf{n} \\ \partial_{uv} s &= \Gamma_{12}^1 \mathbf{u} + \Gamma_{12}^2 \mathbf{v} + f \mathbf{n} \\ \partial_{vv} s &= \Gamma_{22}^1 \mathbf{u} + \Gamma_{22}^2 \mathbf{v} + g \mathbf{n}\end{aligned}$$

Christoffel Symbols Γ_{ij}^k

- Projections of second derivatives into tangent plane
- Intrinsic curvature properties

Shape Operator

Second fundamental form

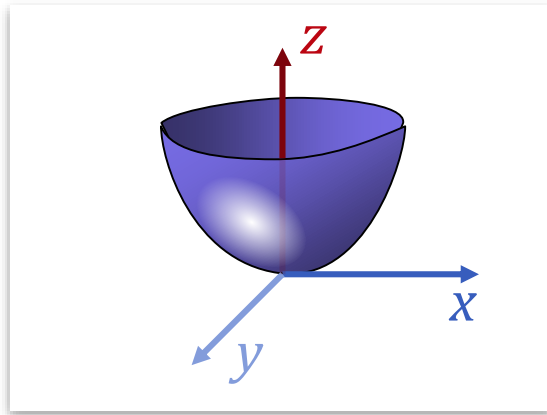
$$\mathbf{II}(\mathbf{x}_0) = \begin{pmatrix} \partial_{uu} \mathbf{S} \cdot \mathbf{n} & \partial_{uv} \mathbf{S} \cdot \mathbf{n} \\ \partial_{uv} \mathbf{S} \cdot \mathbf{n} & \partial_{vv} \mathbf{S} \cdot \mathbf{n} \end{pmatrix} \Big|_{(\text{at } \mathbf{x}_0)}$$

- 2nd fundamental form is parametrization dependent!

Definition: The shape operator

- Orthogonal tangent vectors \mathbf{u}, \mathbf{v} yield the *shape operator* $\mathbf{S}(\mathbf{x}_0)$ (a.k.a. *curvature tensor*)
 - Directional derivative of normal vector
 - Still depends on choice of coordinates (e.g., rotation of \mathbf{u}, \mathbf{v}).

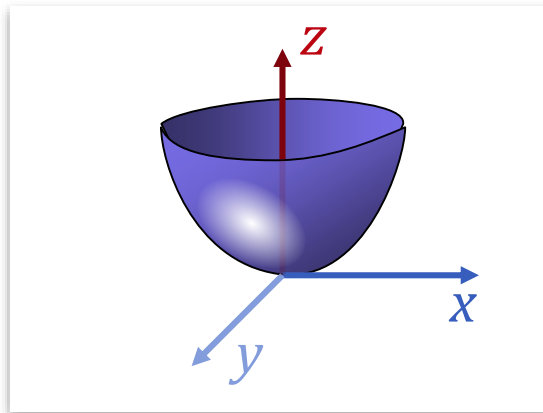
Alternative Formulation (Gauss)



Orthogonal tangent frame

- Local height field parameterization $s(\mathbf{x}) = z(x, y)$
 - Orthonormal x, y coordinates *tangential* to surface
 - Function values z in normal direction
 - Origin at zero
- Then: *shape operator* = *second fundamental form*
= *matrix of second derivatives*

Alternative Formulation (Gauss)



Tangential height fields,
orthogonal frame:

$$\mathbf{H}(\mathbf{x}) = \mathbf{S}(\mathbf{x}) = \mathbf{H}_z(\mathbf{x})$$

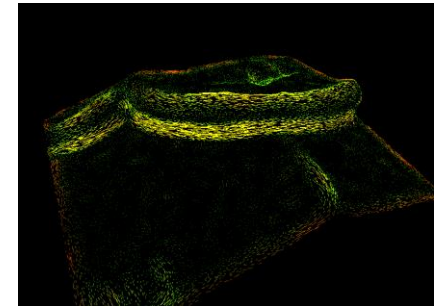
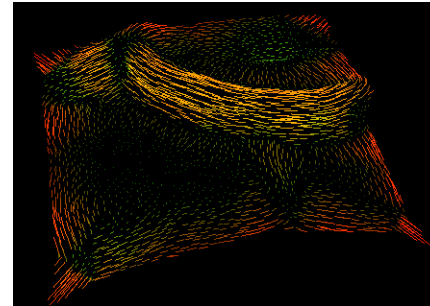
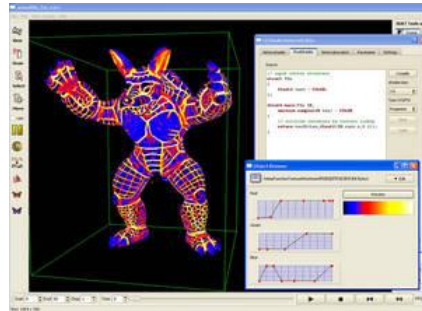
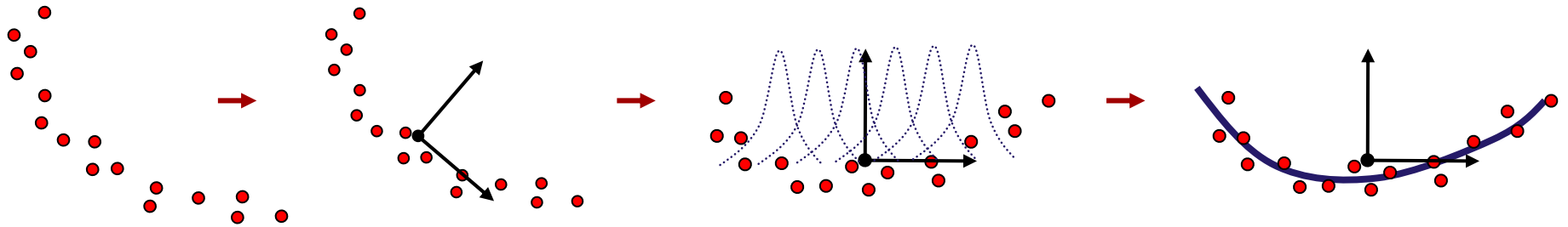
Local height field parameterization

- 2nd order Taylor approximation

$$z(\mathbf{x}) \approx \underbrace{\frac{1}{2} \mathbf{x}^T \cdot \mathbf{H}_z(\mathbf{x}) \cdot \mathbf{x}}_{=ex^2+2fxy+gy^2} + \underbrace{\mathbf{J}_z(\mathbf{x}) \cdot \mathbf{x}}_0 + \underbrace{z(0)}_0$$

$$\begin{pmatrix} e & f \\ f & g \end{pmatrix} = \begin{pmatrix} \partial_{uu}z & \partial_{uv}z \\ \partial_{uv}z & \partial_{vv}z \end{pmatrix}$$

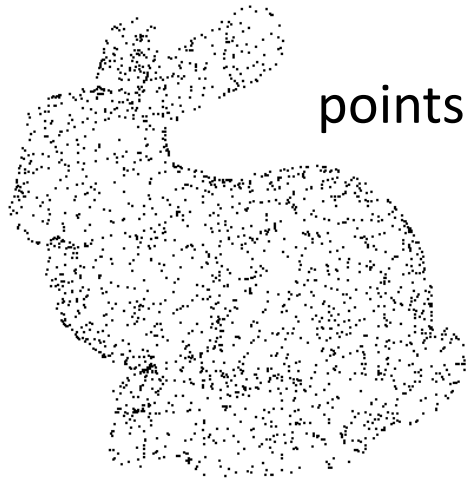
In Practice



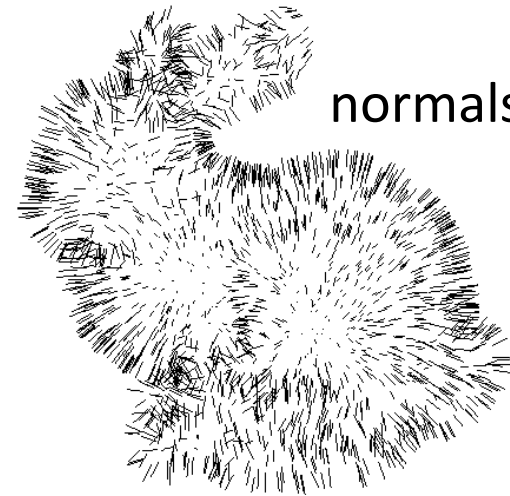
Cloud of data points

- k-nearest neighbors
- PCA for approx. tangent plane
- Least-squares fitting of height field

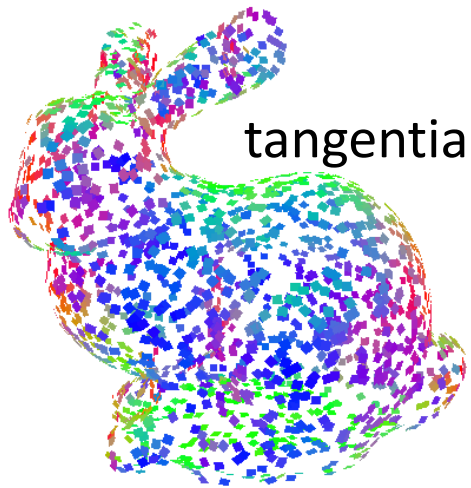
Example



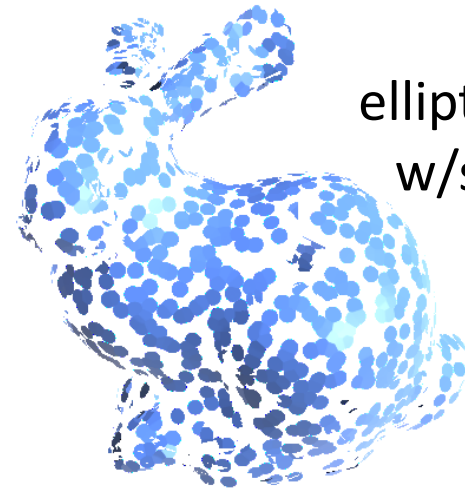
points



normals



tangential frames

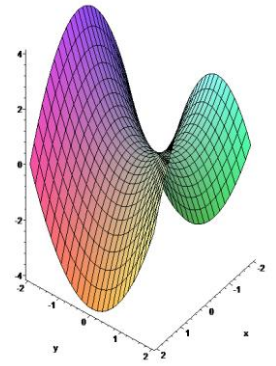
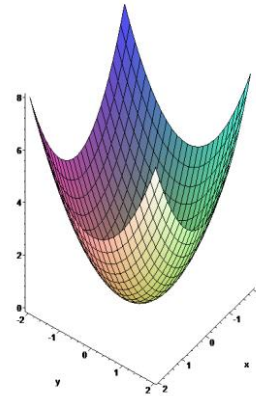


elliptic splats
w/shading

Basic Idea

In other words:

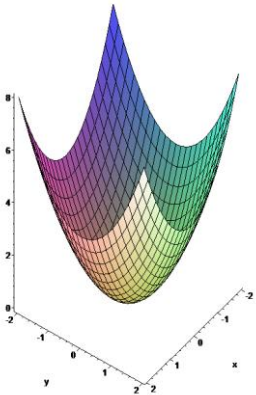
- *First fundamental form: I*
Linear part (squared) of local Taylor approximation.
- *Second fundamental form: II*
Quadratic part of heightfield approximation
- Both matrices are symmetric.
 - Next: eigenanalysis, of course...



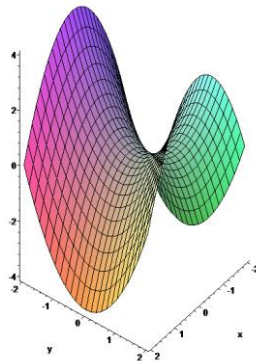
Principal Curvature

Eigenanalysis

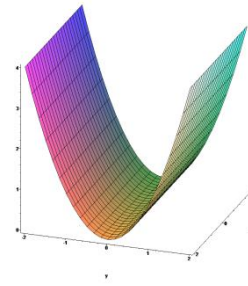
- Eigenvalues of **shape operator** are called *principal curvatures* κ_1, κ_2 .
- Corresponding eigenvectors are called *directions of principle curvature*.



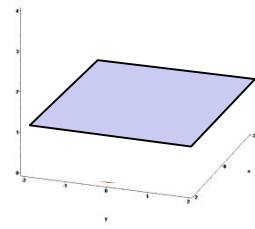
$$\kappa_j > 0$$



$$\kappa_0 > 0, \kappa_1 < 0$$



$$\kappa_0 = 0, \kappa_1 > 0$$



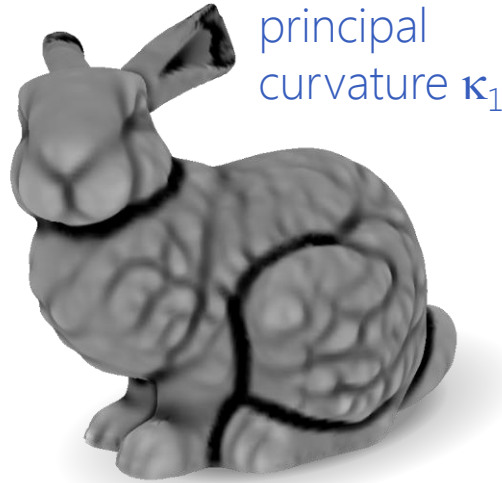
$$\kappa_0 = 0, \kappa_1 = 0$$

...

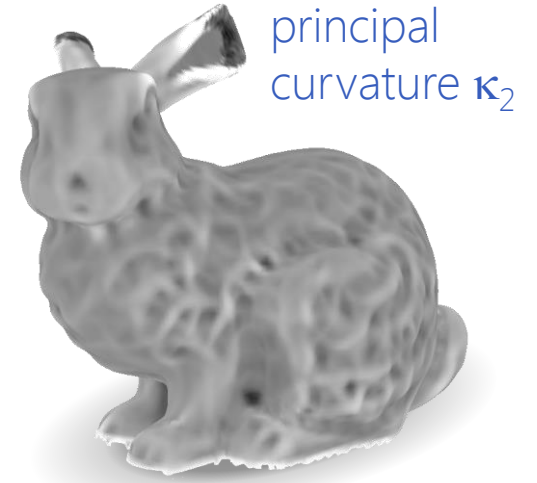
Examples



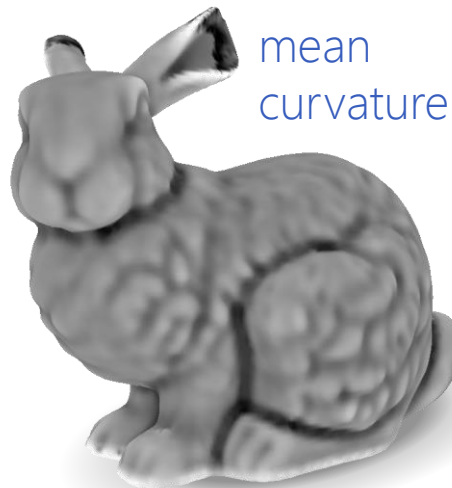
Stanford Bunny
(dense point cloud)



principal
curvature κ_1



principal
curvature κ_2



mean
curvature



Gaussian
curvature

[courtesy of Martin Bokeloh]

Normal Curvature

Definition

- *Normal curvature* $k(\mathbf{r})$ in direction \mathbf{r} at \mathbf{x}_0

$$k_{\mathbf{x}_0}(\mathbf{r}) := \mathbf{r}^T \cdot \mathbf{S}(\mathbf{x}_0) \cdot \mathbf{r}$$

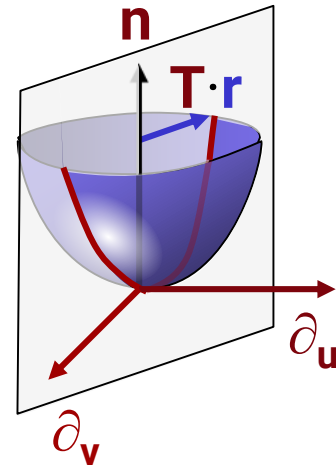
(for $\|\mathbf{r}\| = 1, \mathbf{r} \in \mathbb{R}^2$)

Relation to curvature of plane curves

- Intersect the surface with plane spanned by

$\mathbf{n}(\mathbf{x}_0)$ and $\begin{pmatrix} | & | \\ \mathbf{u}(\mathbf{x}_0) & \mathbf{v}(\mathbf{x}_0) \\ | & | \end{pmatrix} \cdot \mathbf{r}$ through $s(\mathbf{x}_0)$.

- Identical curvatures (up to sign)



Normal & Principal Curvatures

Relation to principal curvature

- Maximum principal curvature κ_1
= maximum of normal curvature
- Minimum principal curvature κ_2
= minimum of normal curvature

Gaussian & Mean Curvature

More Definitions

- *Gaussian curvature* $K := \kappa_1 \kappa_2$
 - Product of principal curvatures
- *Mean curvature* $H := \frac{1}{2}(\kappa_1 + \kappa_2)$
 - Average of principle curvatures

Theorems

- $K(\mathbf{x}) = \det(\mathbf{S}(\mathbf{x}))$
- $H(\mathbf{x}) = \frac{1}{2} \text{tr}(\mathbf{S}(\mathbf{x}))$

Gaussian & Mean Curvature

More Definitions

- *Gaussian curvature* $K := \kappa_1 \kappa_2$
 - Product of principal curvatures
- *Mean curvature* $H := \frac{1}{2}(\kappa_1 + \kappa_2)$
 - Average of principle curvatures

Theorems

- $K(\mathbf{x}) = \det(\mathbf{S}(\mathbf{x})) = \frac{\det \mathbf{II}}{\det \mathbf{I}} = \frac{eg - f^2}{EG - F^2}$

- $H(\mathbf{x}) = \frac{1}{2} \operatorname{tr}(\mathbf{S}(\mathbf{x})) = \frac{eG - 2fF + gE}{2(EG - F^2)}$

last part:
holds for general
fundamental forms!

shape operator only!

Global Properties

Definitions

- An *isometry* is a mapping between surfaces that preserves distances on the surface (“*geodesic distances*”)
- *Developable surface*: Gauss curvature zero everywhere
 - I.e. no curvature in at least one direction.
 - Examples: Cylinder, Cone, Plane

Developable surfaces

- Developable surfaces can be (locally) mapped to a plane isometrically (flattening out, unroll).

Theorema Egregium

Theorema egregium (Gauss, 1828)

- Surfaces (2-manifolds) in 3D
- Any isometric mapping preserves Gaussian curvature
 - Gaussian curvature is invariant under isometric maps
 - “Intrinsic surface property”

Consequence

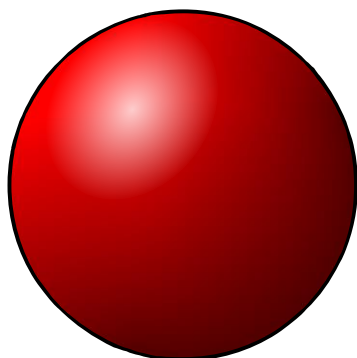
- The earth (\approx sphere) cannot be mapped to a plane in a length preserving way.
- Maps / atlases distort distances

Gauss Bonnet Theorem

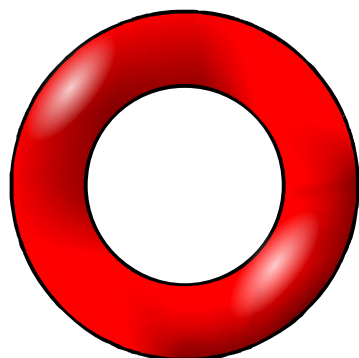
Gauss Bonnet Theorem

- Let $\mathcal{S} \subset \mathbb{R}^3$ be smooth, compact, orientable surface without boundary
- Then, the area integral of the Gauss curvature is related to the genus g of the surface:

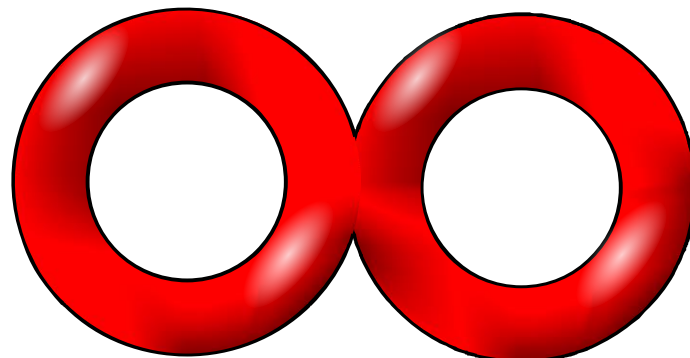
$$\int_{\mathcal{S}} K(\mathbf{x}) d\mathbf{x} = 4\pi(1 - g)$$



$g = 0$



$g = 1$



$g = 2$

...

Fundamental Theorem of Surfaces

Theorem

- Given two parametric patches in $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathbb{R}^3$,
- defined on the same domain Ω :

$$\mathcal{S}_i = s_i(\Omega).$$

- Assume that first and second fundamental form are identical

$$\mathbf{I}_1 \equiv \mathbf{I}_2, \quad \mathbf{II}_1 \equiv \mathbf{II}_2.$$

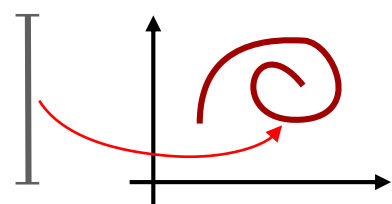
- Then there exists a rigid motion that maps on surface to the other

$$\mathcal{S}_2 = \mathbf{T}(\mathcal{S}_1), \text{ for some } \mathbf{T} \in E(3).$$

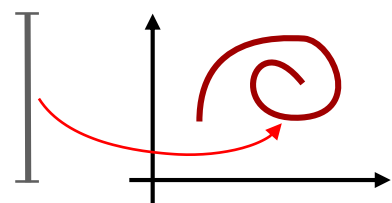
Summary

Objects are the same up to a rigid motion, if...:

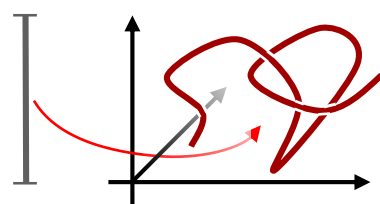
- Curves $\mathbb{R} \rightarrow \mathbb{R}^2$: Same *speed*, same *curvature*
- Curves $\mathbb{R} \rightarrow \mathbb{R}^3$: Same *speed*, same *curvature*, *torsion*
- Surfaces $\mathbb{R}^2 \rightarrow \mathbb{R}^3$: Same *first* & *second* fundamental form
- Volumetric objects $\mathbb{R}^3 \rightarrow \mathbb{R}^3$: Same *first* fundamental form



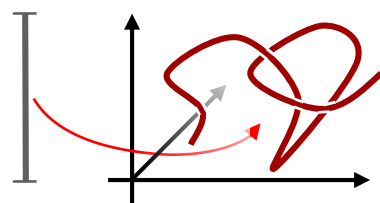
=



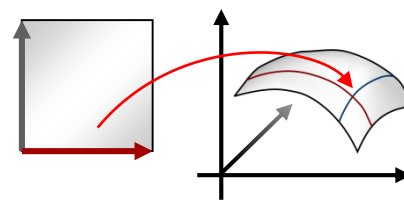
plane curve



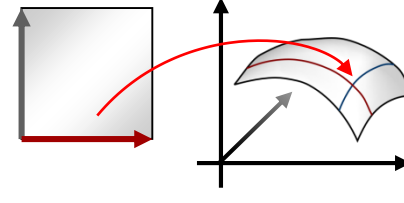
=



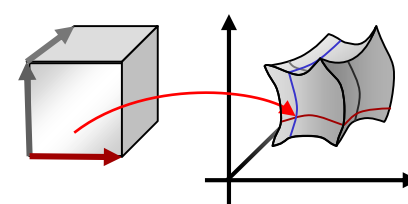
space curve



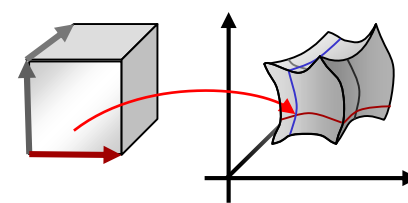
=



surface



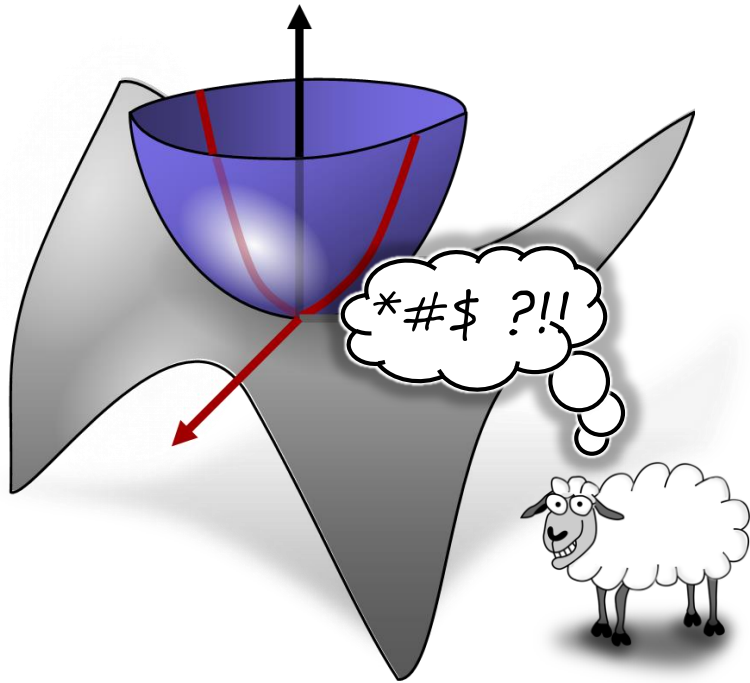
=



space warp

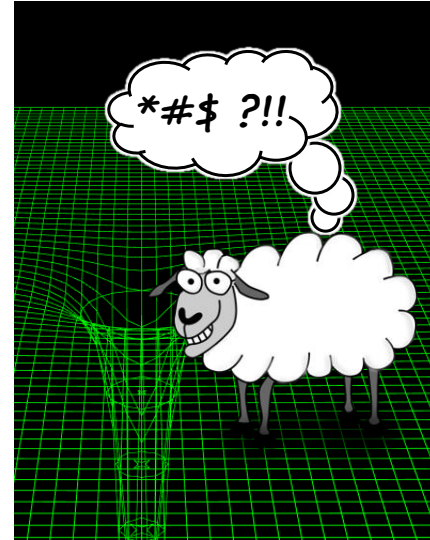
Intrinsic Differential Geometry

Differential Geometry Intro



Embedded Geometry

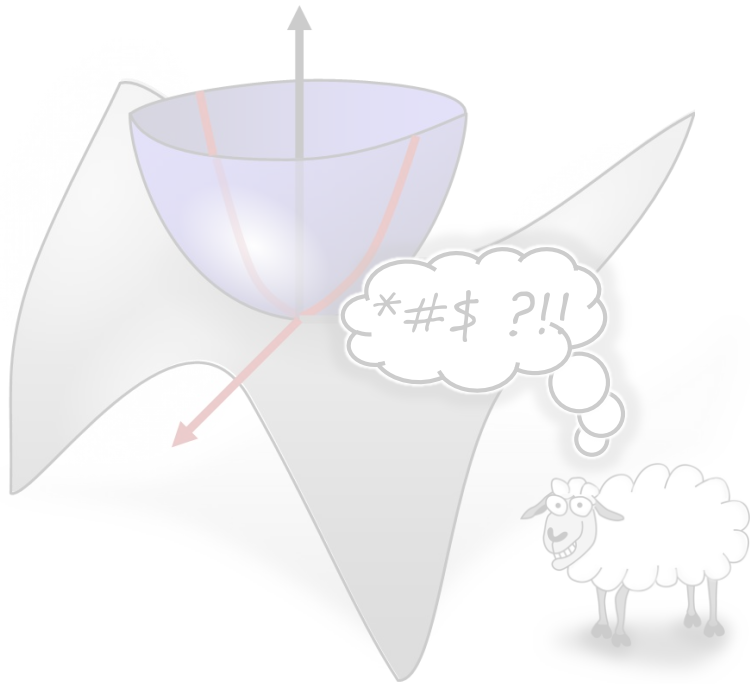
d -dim. Manifold embedded in \mathbb{R}^n
($d \leq n$)



Intrinsic Geometry

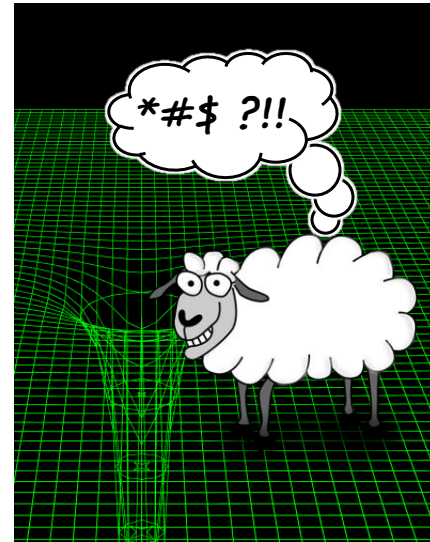
no ambient space
("general relativity")

Differential Geometry Intro



Embedded Geometry

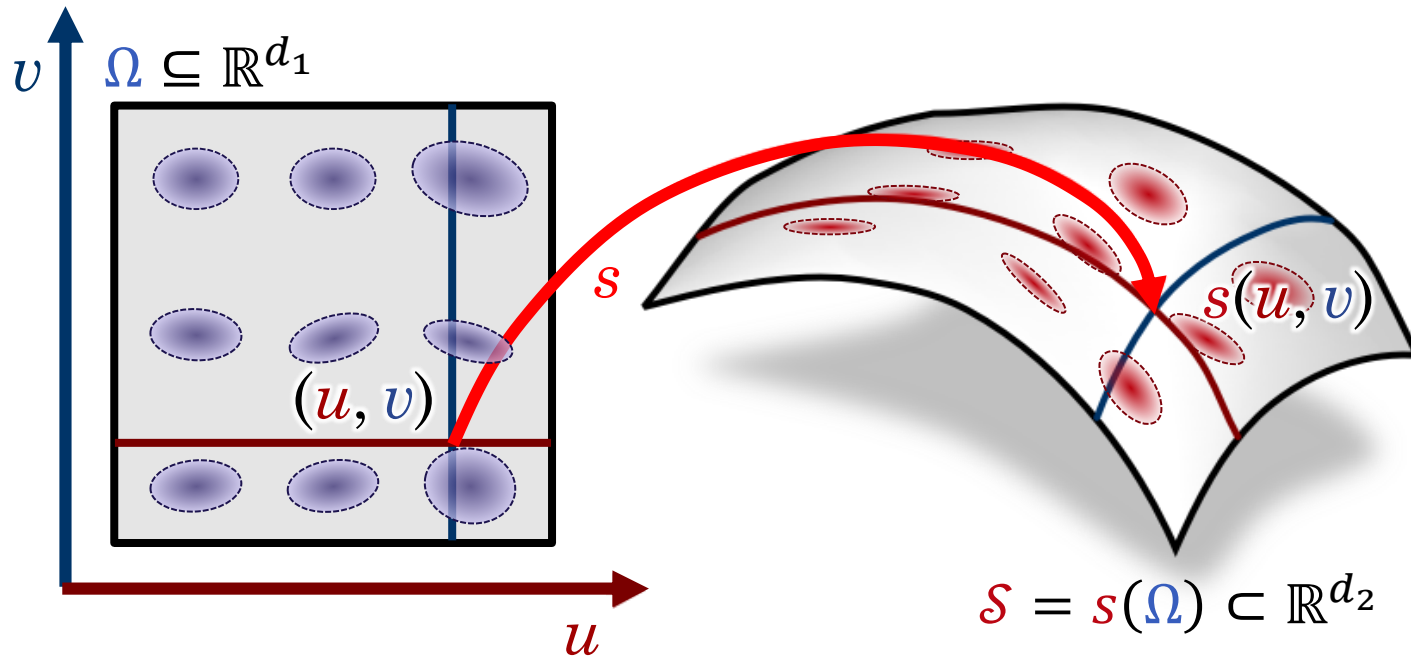
d -dim. Manifold embedded in \mathbb{R}^n
($d \leq n$)



Intrinsic Geometry

no ambient space
("general relativity")

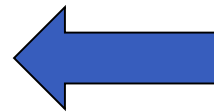
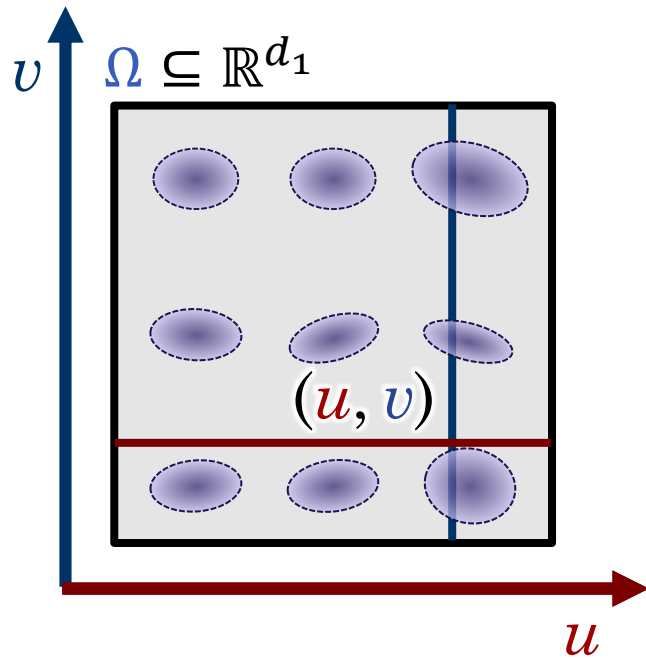
Illustration



Metric Distortion
 $(g_{ij})_{ij} = [\nabla s^T \nabla s]$

(Extrinsic Counterpart: $[\nabla s \nabla s^T]$)

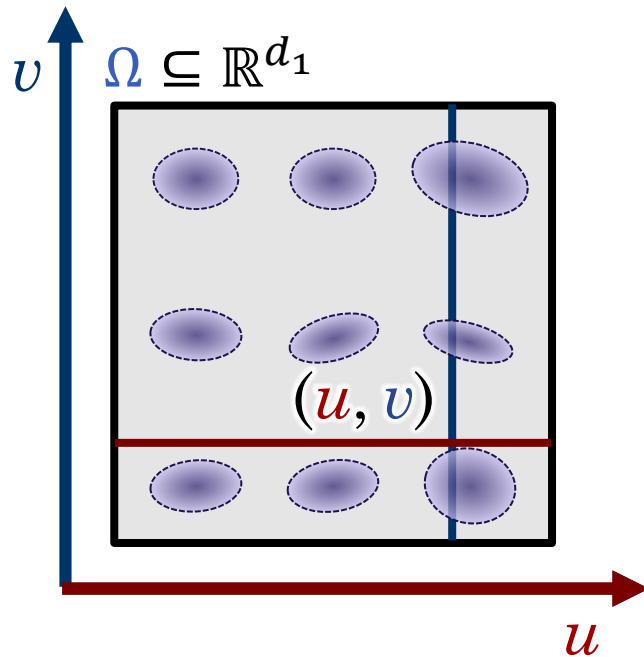
Illustration



Riemannian Metric
(on a Riemannian Manifold)

Metric Tensor
 $(g_{ij})_{ij} = [\nabla s^T \nabla s]$

Illustration

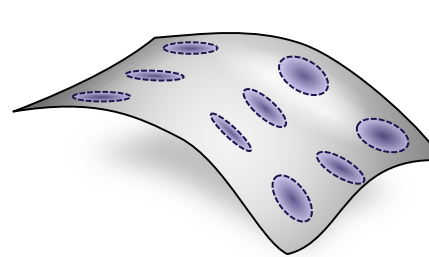


Metric Tensor

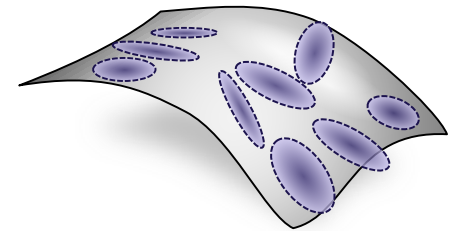
$$(g_{ij})_{ij} = [\nabla s^T \nabla s]$$

Example:

We alter the standard metric



standard
metric



non-standard
(SPD quadric
at each point)

Curvature

Given

- Abstract parameter domain $\Omega \subset \mathbb{R}^d$
- Metric $g: \Omega \rightarrow \mathbb{R}^{d \times d}$

Higher-order properties

(1) Define derivative of g

- “Covariant derivative” or “connection”
- Canonical choice: Levi-Cevita-connection
 - Behaves like projection into tangent plane
 - No torsion

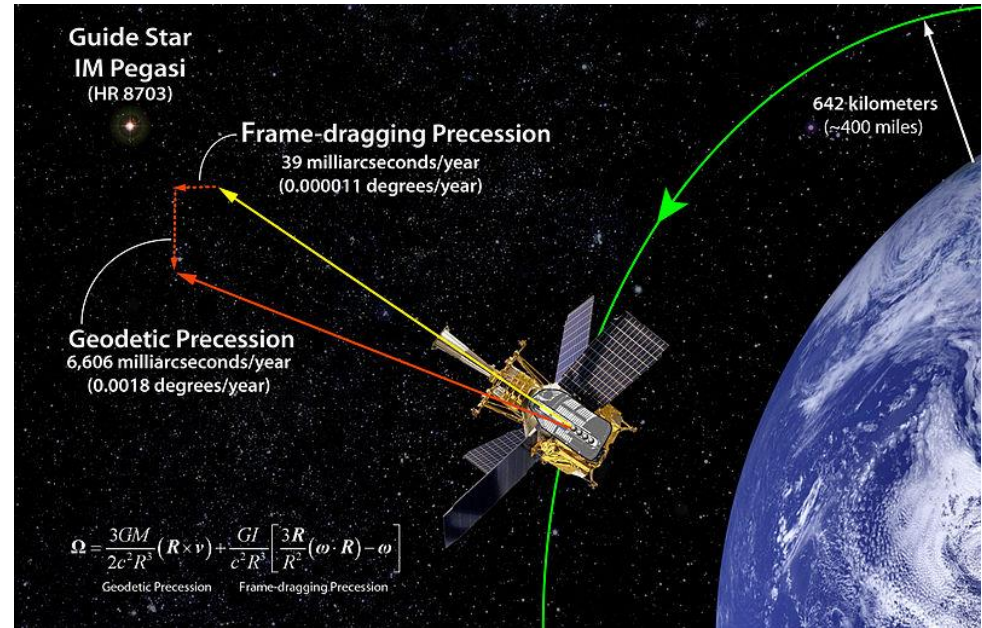
(2) Riemann Curvature Tensor

- Invariants are the analog to “Gaussian curvature”

Space(-Time) is not Euclidean



[NASA]



[NASA]

Geodesics

Geodesics

Definition

- A geodesic is a curve with no intrinsic curvature

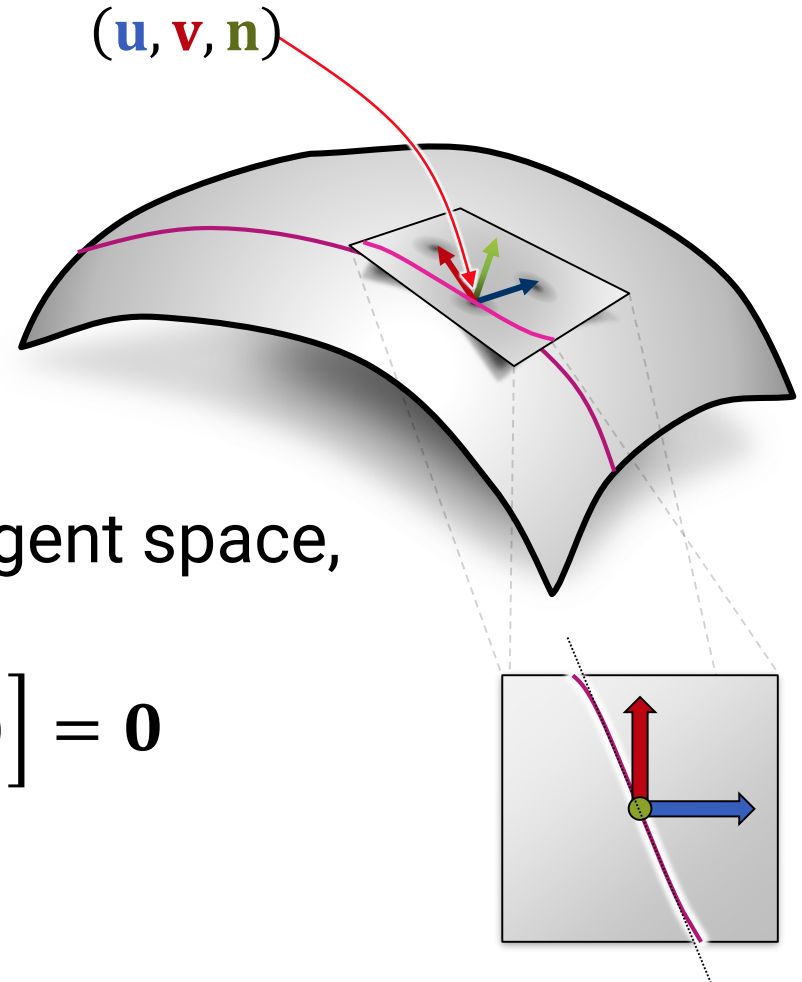
Embedded case

- After projection into the tangent space, we have no curvature

$$\kappa \left[\begin{pmatrix} -\mathbf{u} & - \\ -\mathbf{v} & - \end{pmatrix} f(t) \right] = \mathbf{0}$$

Shortest path

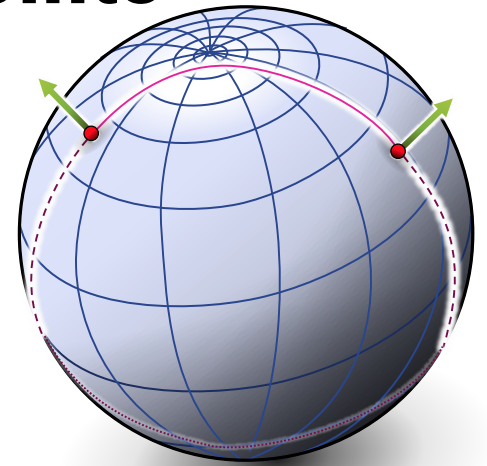
- Shortest paths on smooth manifolds are geodesics



Geodesic Distances

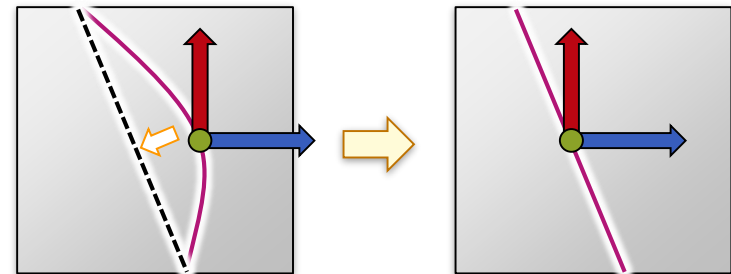
Shortest distance between two points

- “Geodesic distance”
- Path itself:
Often also called “Geodesic”



Intuition

- If there was still intrinsic curvature
 - Path could be straightened
 - Shortens path

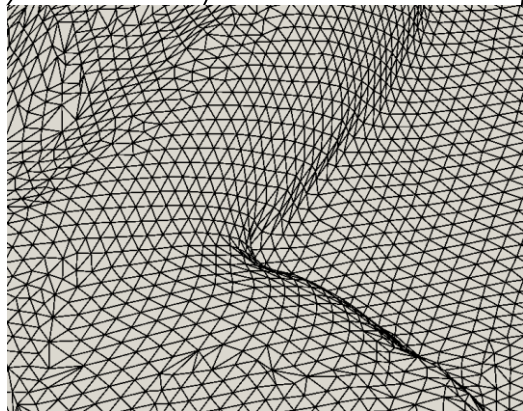
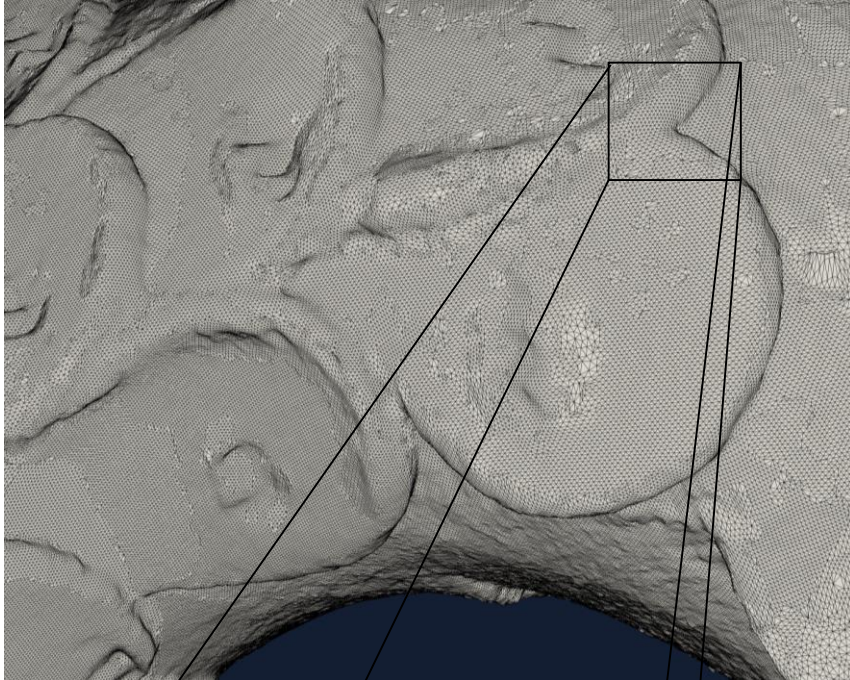


Computing Shortest Paths

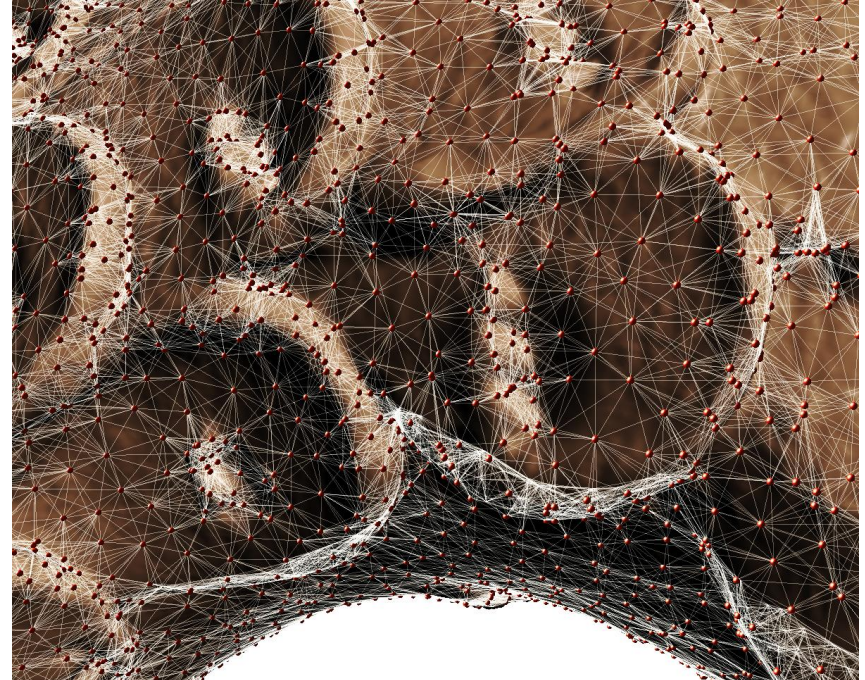
Approximate Global Optimum

- **Discretize**
 - Graph representation
 - Sample points on surface
 - Mesh or Point-Cloud with k-nearest-neighbor-Graph
- **Connect nearby points with edges**
 - Local Euclidean distance as weights
 - First-order approximation of intrinsic metric
 - First-order consistent error
- **Dijkstra graph shortest path**
 - Not consistent – metrification errors
 - Discrete directions lead to overestimation

Neighborhood Graphs



original
mesh



resampled point cloud with
20-nearest-neighbors graph

Discrete Geodesics

“Dijkstra” geodesics

- Advantages
 - Easy to implement
 - Global optimum
- Disadvantages
 - $O(n \log n)$ cost for n points, one-to-all paths (one-to-one not faster!)
 - Approximate – substantial errors (overestimation)



shortest path with
point-cloud NN-graph
[Image: Art Tevs]

Continuous Geodesics

Continuous geodesics

■ Smoothing

- Start with coarse path
- Minimize path length
- $\int_a^b \left\| \frac{d}{dt} c(t) \right\|^2 dt \rightarrow \min.$
- Constrained least-squares

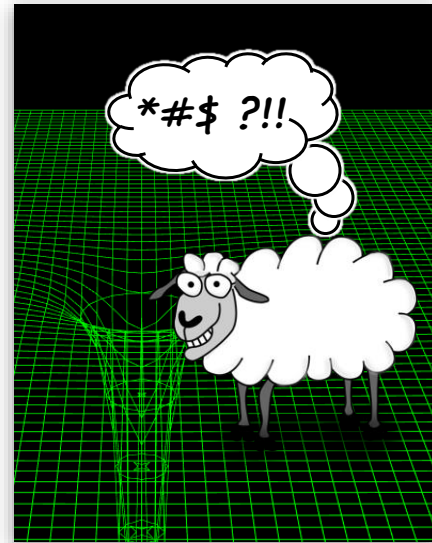
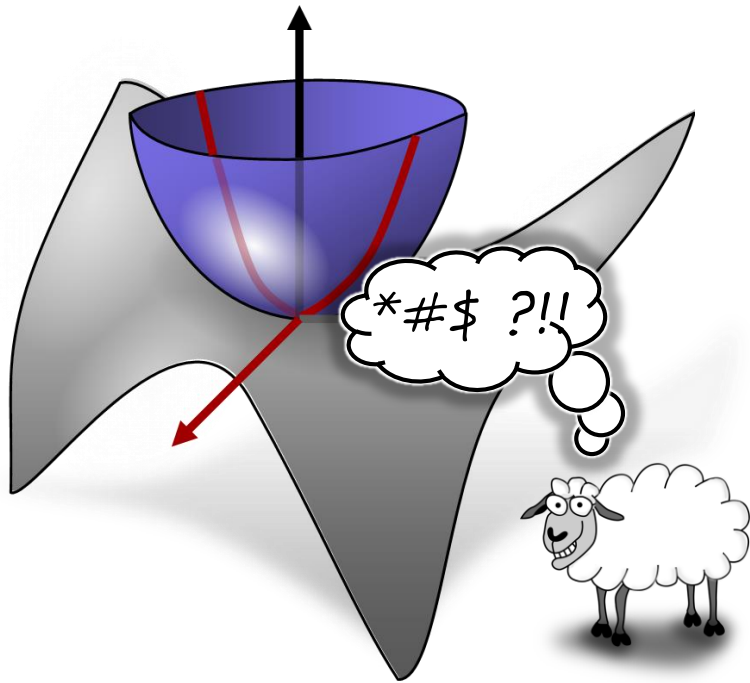
■ Disadvantages

- Expensive
- Global optimum not guaranteed (theoretical issue, works in practice)



[Image: Art Tevs]

Applications



Differential Geometry in ML

Example Applications

- Isomap
 - Approximate intrinsic geometry
- The Fisher information matrix
 - A natural metric for distributions
- Intrinsic views of deep networks
 - Networks in input space

ISOMAP

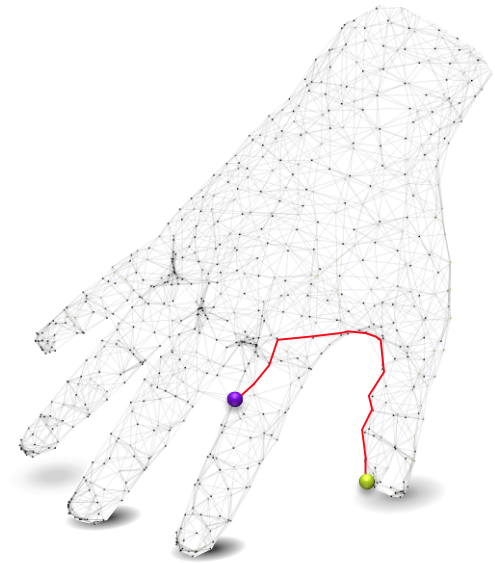
Isomap

Mapping Manifolds to Euclidean Space

- Approximation
- Assuming disc topology

Algorithm: “ISOMAP”

- Compute all pairwise intrinsic distances
 - Typically: k-NN graph, Dijkstra’s Algorithm
- Run MDS on pairwise distances
 - Another kernel-PCA variant
 - Intrinsic metric for embedding



The Fisher Information Matrix

References

James Martens: New Insights and Perspectives on the Natural Gradient Method
Journal of Machine Learning Research 21 (2020) 1-76
<https://jmlr.org/papers/volume21/17-678/17-678.pdf>

Agustinus Kristiadi: Fisher Information Matrix / Natural Gradient Descent
<https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/>

Fisher Information

Big picture

- We often use parametric distributions

$$p_{\theta}(\mathbf{x}), \quad \theta \in \mathbb{R}^d$$

- Natural metric on parameter space $\Omega(\theta)$

Information theory

- Use KL-divergence to measure distance

Differential geometry

- Derive metric tensor for changes in distribution
 - Not distance in parameter space

Note: Notation

Gradient operator

$$\nabla_{\mathbf{x}} = \begin{bmatrix} \partial_{x_1} \\ \vdots \\ \partial_{x_d} \end{bmatrix}, \quad \nabla_{\mathbf{x}}^T = [\partial_{x_1} \cdots \partial_{x_d}]$$

Hessian operator / matrix

$$\nabla_{\mathbf{x}}^T \nabla_{\mathbf{x}} = \begin{bmatrix} \partial_{x_1} \partial_{x_1} & \cdots & \partial_{x_d} \partial_{x_1} \\ \vdots & & \vdots \\ \partial_{x_1} \partial_{x_d} & \cdots & \partial_{x_d} \partial_{x_d} \end{bmatrix}$$

KL-Divergence

Consider

- $KL(p_{\theta} \| p_{\theta+\epsilon})$ for $\epsilon \rightarrow 0$
- $\theta, \epsilon \in \mathbb{R}^d$
- p_{θ} smooth in θ

Let's see

$$KL(p_{\theta} \| p_{\theta+\epsilon}) = \sum_{x \in \Omega(X)} p_{\theta}(x) (\log_2 p_{\theta}(x) - \log_2 p_{\theta+\epsilon}(x))$$

- Note: for small $\|\epsilon\|$, the KL -divergence is symmetric

KL-Divergence

Gradients

$$\begin{aligned}\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) &= \sum_{\mathbf{x} \in \Omega(X)} [\nabla_{\epsilon} p_{\theta}(\mathbf{x})] (\log_2 p_{\theta}(\mathbf{x}) - \log_2 p_{\theta+\epsilon}(\mathbf{x})) \\ &\quad - \sum_{\mathbf{x} \in \Omega(X)} p_{\theta}(\mathbf{x}) (\nabla_{\epsilon} \log_2 p_{\theta+\epsilon}(\mathbf{x}))\end{aligned}$$

KL-Divergence


Gradients

$$\begin{aligned}\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) &= \sum_{\mathbf{x} \in \Omega(X)} [\nabla_{\epsilon} p_{\theta}(\mathbf{x})] (\log_2 p_{\theta}(\mathbf{x}) - \log_2 p_{\theta+\epsilon}(\mathbf{x})) \\ &\quad - \sum_{\mathbf{x} \in \Omega(X)} p_{\theta}(\mathbf{x}) (\nabla_{\epsilon} \log_2 p_{\theta+\epsilon}(\mathbf{x})) \\ &= - \sum_{\mathbf{x} \in \Omega(X)} p_{\theta}(\mathbf{x}) (\nabla_{\epsilon} \log_2 p_{\theta+\epsilon}(\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] \\ &= 0 (!)\end{aligned}$$

Expected “Score Function”

Expected gradients of log-likelihoods are zero


$$\mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} (\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}))] = \sum_{\mathbf{x} \in \Omega(X)} p_{\boldsymbol{\theta}}(\mathbf{x}) (\nabla_{\boldsymbol{\theta}} (\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})))$$

$$\nabla \log f(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{f(\mathbf{x})}$$


Expected “Score Function”

Expected gradients of log-likelihoods are zero

$$\mathbb{E}_{\mathbf{x} \sim p_{\boldsymbol{\theta}}(\mathbf{x})} [\nabla_{\boldsymbol{\theta}} (\log_2 p_{\boldsymbol{\theta}}(\mathbf{x}))] = \sum_{\mathbf{x} \in \Omega(X)} p_{\boldsymbol{\theta}}(\mathbf{x}) (\nabla_{\boldsymbol{\theta}} (\log_2 p_{\boldsymbol{\theta}}(\mathbf{x})))$$

$$\nabla \log f(\mathbf{x}) = \frac{\nabla f(\mathbf{x})}{f(\mathbf{x})} \left. \vphantom{\nabla \log f(\mathbf{x})} \right\} = \sum_{\mathbf{x} \in \Omega(X)} p_{\boldsymbol{\theta}}(\mathbf{x}) \left(\frac{\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})} \right)$$


$$= \sum_{\mathbf{x} \in \Omega(X)} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{x})$$

$$= \nabla_{\boldsymbol{\theta}} \sum_{\mathbf{x} \in \Omega(X)} p_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} 1 = 0$$

KL-Divergence

Gradients

$$\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] = 0$$

Hessian

$$[\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}^T [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))]]$$

KL-Divergence

Gradients

$$\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] = 0$$

Hessian

$$[\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}^T [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))]]$$

Note (probably not stressed enough in the video):

- The gradient only vanishes, because θ are the true parameters
- We take $\mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\dots]$ of $\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))$
 - ← true distribution
 - ← comparison
- In the proof sketch (Slide 214/215), θ is at the “true” value
- In general, gradients of log-likelihoods do not vanish!
 - Optimization in DL is all about gradient descent neg-log-likelihoods!

KL-Divergence

Gradients

$$\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] = 0$$

Hessian

$$[\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}^T [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))]]$$

KL-Divergence

Gradients

$$\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] = 0$$

Hessian

$$[\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}^T [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))]]$$

$$= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[-\nabla_{\epsilon}^T \frac{\nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})} \right]$$

$$= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[-\frac{([\nabla_{\epsilon}^T \nabla_{\epsilon}] p_{\theta+\epsilon}(\mathbf{x})) p_{\theta+\epsilon}(\mathbf{x}) - \nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x}) \cdot \nabla_{\epsilon}^T p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})^2} \right]$$

$$\nabla \frac{f(\mathbf{x})}{g(\mathbf{x})} = \frac{\nabla f(\mathbf{x}) g(\mathbf{x}) - f(\mathbf{x}) \nabla g(\mathbf{x})}{g(\mathbf{x})^2}$$

KL-Divergence

Hessian

$$\begin{aligned} & [\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[- \frac{([\nabla_{\epsilon}^T \nabla_{\epsilon}] p_{\theta+\epsilon}(\mathbf{x})) p_{\theta+\epsilon}(\mathbf{x}) - \nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x}) \cdot \nabla_{\epsilon}^T p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})^2} \right] \end{aligned}$$

KL-Divergence

Hessian

$$\begin{aligned} & [\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[- \frac{([\nabla_{\epsilon}^T \nabla_{\epsilon}] p_{\theta+\epsilon}(\mathbf{x})) p_{\theta+\epsilon}(\mathbf{x}) - \nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x}) \cdot \nabla_{\epsilon}^T p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})^2} \right] \\ &= - \sum_{\mathbf{x} \in \Omega(X)} [\nabla_{\epsilon}^T \nabla_{\epsilon}] p_{\theta+\epsilon}(\mathbf{x}) + \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[\left(\frac{\nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})} \right) \left(\frac{\nabla_{\epsilon}^T p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})} \right) \right] \\ &= [\nabla_{\epsilon}^T \nabla_{\epsilon}] \sum_{\mathbf{x} \in \Omega(X)} p_{\theta+\epsilon}(\mathbf{x}) + \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} \left[\left(\frac{\nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})} \right) \left(\frac{\nabla_{\epsilon} p_{\theta+\epsilon}(\mathbf{x})}{p_{\theta+\epsilon}(\mathbf{x})} \right)^T \right] \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x})) \cdot \nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))^T] \end{aligned}$$

Summary

“Score Function”: Derivative of neg-log-likelihood

$$\nabla_{\epsilon}(-\log_2 p_{\theta+\epsilon}(\mathbf{x}))$$

Gradient: Vanishes

$$\nabla_{\epsilon} KL(p_{\theta} \| p_{\theta+\epsilon}) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}(-\log_2 p_{\theta+\epsilon}(\mathbf{x}))] = 0$$

Hessian: Covariance Matrix

$$\begin{aligned} & [\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon}(-\log_2 p_{\theta+\epsilon}(\mathbf{x})) \cdot \nabla_{\epsilon}(-\log_2 p_{\theta+\epsilon}(\mathbf{x}))^T] \end{aligned}$$

Summary

Hessian is the Fischer information matrix

$$\begin{aligned}\mathbf{F} &:= [\nabla_{\epsilon}^T \nabla_{\epsilon}] KL(p_{\theta} \| p_{\theta+\epsilon}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_{\theta}(\mathbf{x})} [\nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x})) \cdot \nabla_{\epsilon} (-\log_2 p_{\theta+\epsilon}(\mathbf{x}))^T]\end{aligned}$$

Usage as metric tensor

$$\langle d\theta_a, d\theta_b \rangle_F = (d\theta_a)^T \cdot \mathbf{F} \cdot d\theta_b$$

Applications

“Natural Gradient Descent”

- Standard Gradient Descent
 - Deep network $f: \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_L}$
 - Loss function: Neg-log-likelihood $L(f_{\theta})$
 - Parameters θ (weights)
 - Learning rate λ
 - Gradient descent

$$\theta_{i+1} \leftarrow \theta_i - \lambda \nabla_{\theta} L(f_{\theta})$$

Applications

“Natural Gradient Descent”

- Standard Gradient Descent

$$\theta_{i+1} \leftarrow \theta_i - \lambda \nabla_{\theta} L(f_{\theta})$$

- “Natural” Gradient Descent

$$\theta_{i+1} \leftarrow \theta_i - \lambda \mathbf{F}^{-1} \nabla_{\theta} L(f_{\theta})$$

Discussion

- Problem: Inverting the F-Matrix
 - Too expensive for deep networks
- Approximations possible
 - ADAM uses diagonal \mathbf{F}

Application

Jeffreys Prior

- Inferring parameters via

$$p(\theta|D) \sim p(D|\theta)P(\theta)$$

- We have a likelihood $p(D|\theta)$
- What prior $P(\theta)$ should we use?

Approach

- We want “uninformative” prior
- Independent of parametrization

reparameterization
scales quadratically

$$P_{\text{Jeffreys}}(\theta) := \sqrt{\det \mathbf{F}_{p(D|\theta)}}$$

volume element
in \mathbf{F} -metric

Jeffreys Prior

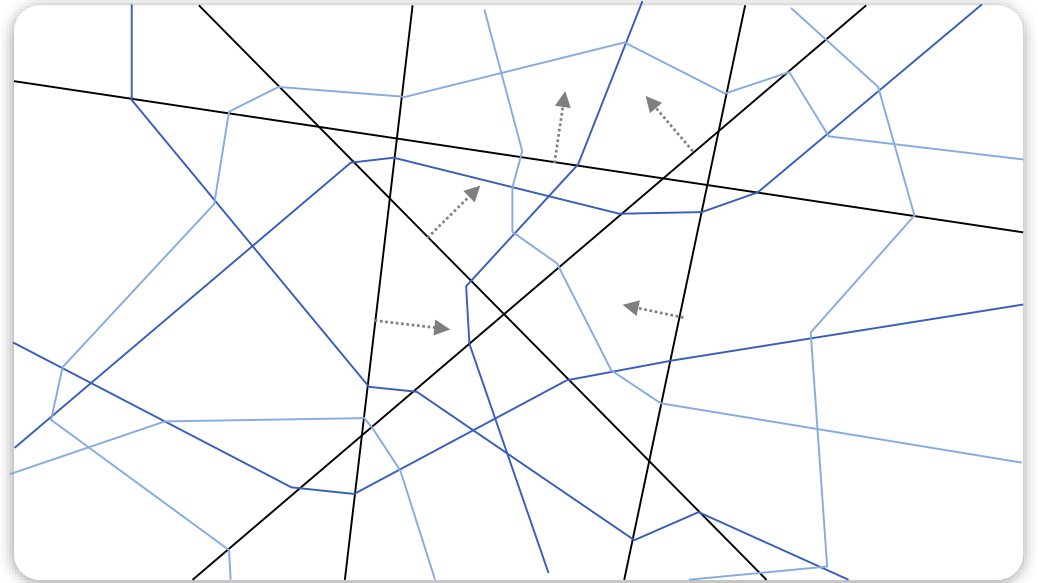
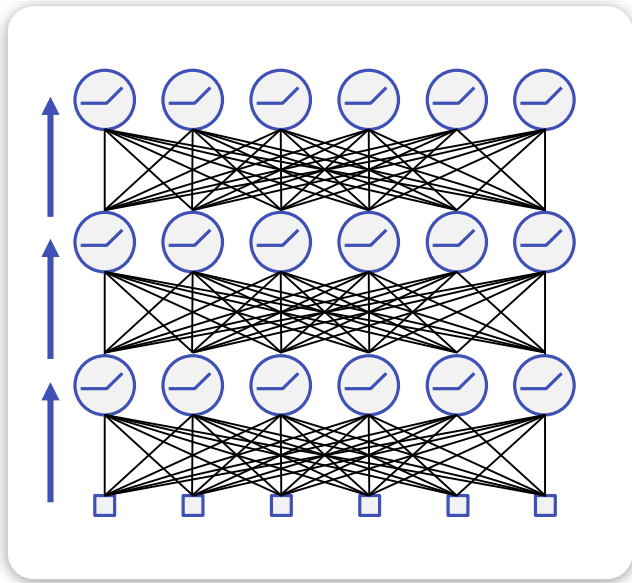
Discussion

- Often used as “objective” Bayesian prior
- It does not solve the problem of infinite domains
 - E.g., improper prior for mean of a Gaussian
- Results invariant under change of domain parametrization
 - However, not invariant under transformations of the output
- Computation might be costly

Intrinsic View of Deep Networks

Credits: David Hartmann

ReLU Networks Subdivide Input Space



Take this one step further

- Feedforward network with ReLU nonlinearity
- Map outputs into input space
- Input $\mathbf{x} \in \mathbb{R}^{d_0}$ \rightarrow Outputs $f(\mathbf{x}) \in \mathbb{R}^{d_L}$ in d_0 -manifold
- Embed outputs in \mathbb{R}^{d_0}

ReLU Networks

Fully-connected ReLU network

$$f^{(L)}(\mathbf{x}, \mathbf{W}) \\ = \varphi \left(\mathbf{W}^{(L)} \varphi \left(\mathbf{W}^{(L-1)} \varphi \left(\dots \mathbf{W}^{(0)} \mathbf{x} \right) \dots \right) \right)$$

In matrix notation

$$f^{(L)}(\mathbf{x}, \mathbf{W}) \\ = \mathbf{R}^{(L)} \mathbf{W}^{(L)} \mathbf{R}^{(L-1)} \mathbf{W}^{(L-1)} \dots \mathbf{R}^{(1)} \mathbf{W}^{(0)} \mathbf{x}$$

- Diagonal 0/1 ReLU matrices $\mathbf{R}^{(l)}$
- Attention! $\mathbf{R}^{(l)}$ depends on preactivation
 - Non-linear, non-constant function of \mathbf{x}

ReLU Networks

Embedding into input space

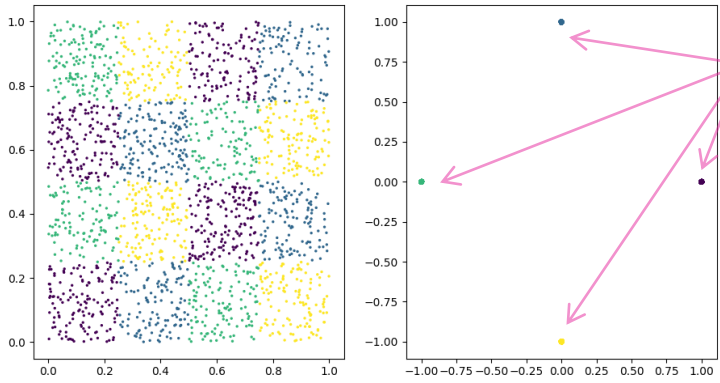
$$\mathbf{x}_f := \left(\mathbf{R}^{(1)} \mathbf{W}^{(1)} \right)^\dagger \dots \left(\mathbf{R}^{(L)} \mathbf{W}^{(L)} \right)^\dagger f^{(L)}(\mathbf{x}, \mathbf{W})$$

where \mathbf{M}^\dagger is the Moore-Penrose pseudo-inverse of \mathbf{M}
(data dependent, different P.-I. for each \mathbf{x})

What does it show?

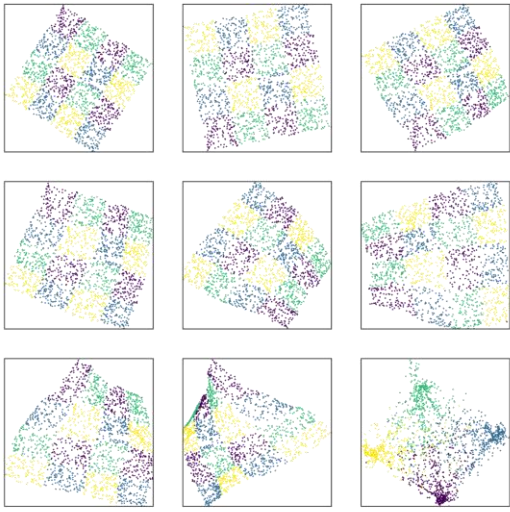
- Visualizes network f as deformation of the input
- Visualization uses additional PCA-dimensionality reduction

Results after PCA [David Hartmann]

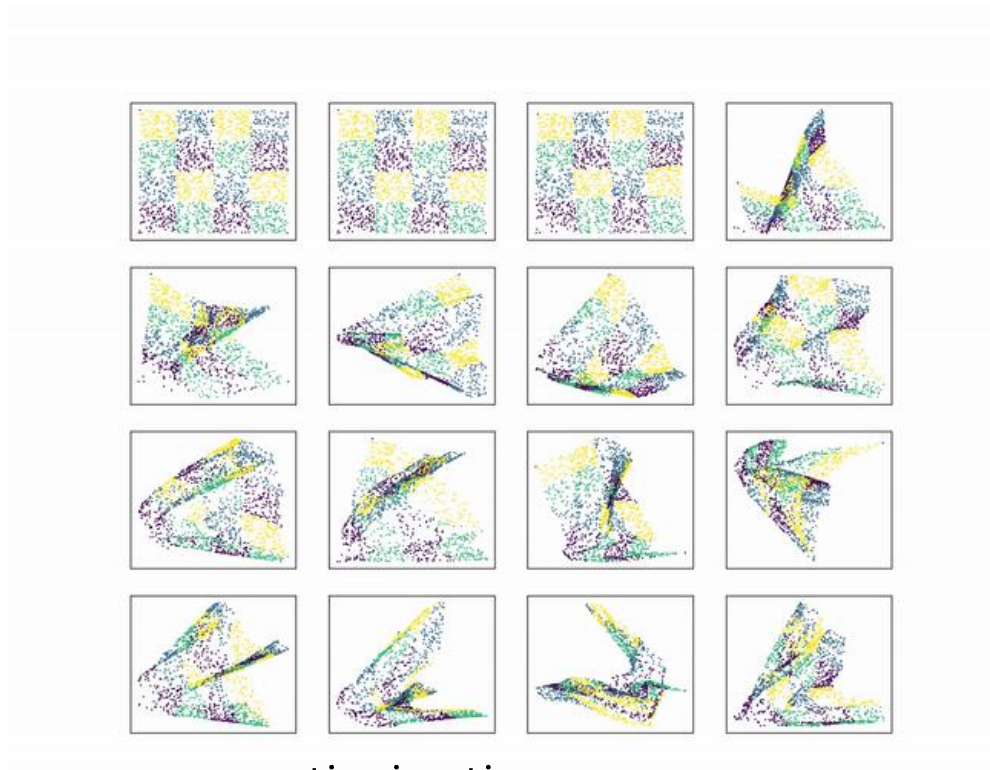


targets

classification task



layer-wise result



optimization process

Summary

Space might not be flat...

Differential geometry

- Studying geometry independent of parametrization
 - Useful to abstract from “implementation details”
- Length/volumes, curvature, higher-order moments

Intrinsic (differential) geometry

- View from inside the manifold
 - Ignore outer space
 - Useful if this does not matter for the application
- Starts with the metric
 - Specify metric tensor
 - Intrinsic curvature can be derived (under assumptions)

Intrinsic View of Curved Space

