Geometric Learning

Generalized Convolutions

Credits:
Bronstein et al, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges", 2021

Rodolà, "Geometric Deep Learning", 2020

1

DEEP LEARNING RECAP

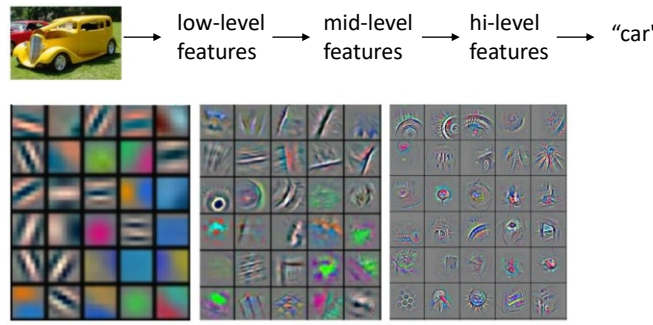
2

Deep learning

Deep learning is a task-driven paradigm to extract patterns and latent features from given observations.

However, features are not always the focus of deep learning; rather, they are instrumental for the given task and drive the decision.

Example: Visual classification

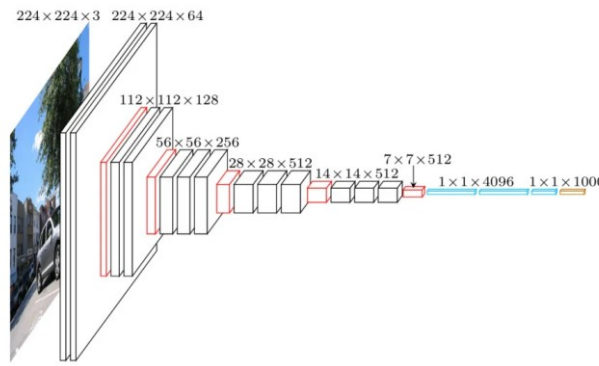


3

3

A glimpse into neural networks

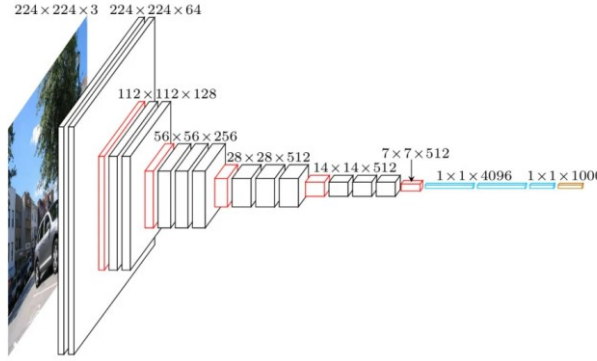
In deep learning, we deal with highly parametrized models called deep neural networks:



4

4

A glimpse into neural networks



- Each block has a predefined structure (e.g., a linear map)
- Each block is defined in terms of unknown parameters
- Finding the parameter values is called training...
- ...which is done by minimizing a function called loss
- Minimization requires computing gradients, called backpropagation

5

5

Deep composition

The simplest example of a nonlinear parametric model:

$$\sigma \circ f(\mathbf{x})$$

If σ is nonlinear, we have a nonlinear regression model.

Consider multiple layers of nonlinear regression models:

$$\text{output} \leftarrow \underbrace{(\sigma \circ f)}_{output\ layer} \circ (\sigma \circ f) \circ \cdots \circ \underbrace{(\sigma \circ f)}_{input\ layer} (\mathbf{x}) \leftarrow \text{input}$$

Popular choices for activation functions:

$$\sigma = \frac{1}{1+e^{-x}} \qquad \sigma = max\{0, x\}$$

continuous discontinuous gradient

6

6

Multi-layer perceptron

We call the composition with linear f and nonlinear σ:

$$(\sigma \circ f) \circ (\sigma \circ f) \circ \cdots \circ (\sigma \circ f)(\mathbf{x})$$

a multi-layer perceptron (MLP) or deep feed-forward neural network.

The parameters or weights of the MLP are scattered across the layers.

Each layer outputs an intermediate hidden representation:

$$x_{\ell+1} = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_\ell + \mathbf{b}_\ell)$$

where we encode the weights at layer ℓ in the matrix $\mathbf{W}_\ell$ and bias $\mathbf{b}_\ell$.

Remark: The bias can be integrated inside the weight matrix by writing:

$$\mathbf{W} \mapsto (\mathbf{W} \quad \mathbf{b}), \quad \mathbf{x} \mapsto \begin{pmatrix} \mathbf{x} \\ \mathbf{1} \end{pmatrix}.$$

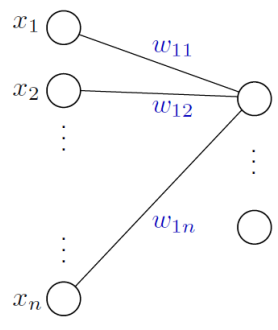
because each f is linear in the parameters just like in linear regression.

7

7

Single layer illustration

$$\sigma(\mathbf{W}\mathbf{x}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

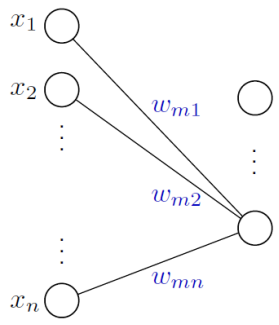


8

8

Single layer illustration

$$\sigma(\mathbf{Wx}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

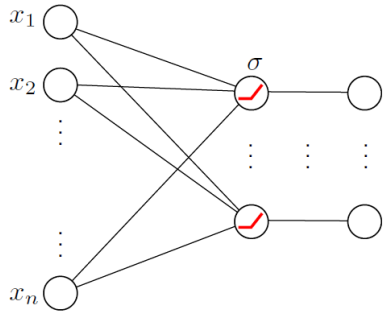


9

9

Single layer illustration

$$\sigma(\mathbf{Wx}) = \sigma \circ \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \sigma \circ \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$



10

10

Universality

What class of functions can we represent with an MLP?

If σ is sigmoidal, we have the following:

Universal Approximation Theorem - For any compact set $\Omega \subset \mathbb{R}^p$, the space spanned by the functions $\phi(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$ is dense in $\mathcal{C}(\Omega)$ for the uniform convergence. Thus, for any continuous function f and any $\epsilon > 0$, there exists $q \in \mathbb{N}$ and weights s.t.:

$$\left| f(\mathbf{x}) - \sum_{k=1}^{q} u_k \phi(\mathbf{x}) \right| \leq \epsilon \quad \text{for all } \mathbf{x} \in \Omega$$

The network in the theorem has just one hidden layer.

For large enough q, the training error can be made arbitrarily small.

11

11

Training

Given a MLP with training pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

$$g_\Theta(\mathbf{x}_i) = \left(\sigma \circ f_{\Theta_n}\right) \circ \left(\sigma \circ f_{\Theta_{n-1}}\right) \circ \cdots \circ \left(\sigma \circ f_{\Theta_1}\right)(\mathbf{x}_i) = \mathbf{y}_i$$

Consider the MSE loss:

$$\ell_\Theta(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - g_\Theta(\mathbf{x}_i)\|_2^2$$

Solving for the weights Θ is referred to as training.

In general, the loss is not convex w.r.t. Θ.

Some special cases are convex:

- One layer, no activation, MSE loss ($\Rightarrow$ linear regression).
- One-layer, sigmoid activation, logistic loss ($\Rightarrow$ logistic regression).

12

12

Training

We train using gradient descent-like algorithms.

Each parameter gets updated so as to decrease the loss:

$$\Theta_i \leftarrow \Theta_i - \alpha \frac{\partial \ell}{\partial \Theta_i}$$

13

13

Training

Bottleneck: Computation of gradients $\nabla \ell_\Theta$.

For the basic MSE, this means:

$$\nabla \ell_\Theta(\{\mathbf{x}_i, \mathbf{y}_i\}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\Theta \|\mathbf{y}_i - g_\Theta(\mathbf{x}_i)\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} \nabla_\Theta \left\| \mathbf{y}_i - (\sigma(f_{\Theta_n}((\sigma\left(f_{\Theta_{n-1}}\left(\cdots\left(\sigma\left(f_{\Theta_1}(\mathbf{x}_i)\right)\cdots\right)\right)\right)\right\|_2^2$$

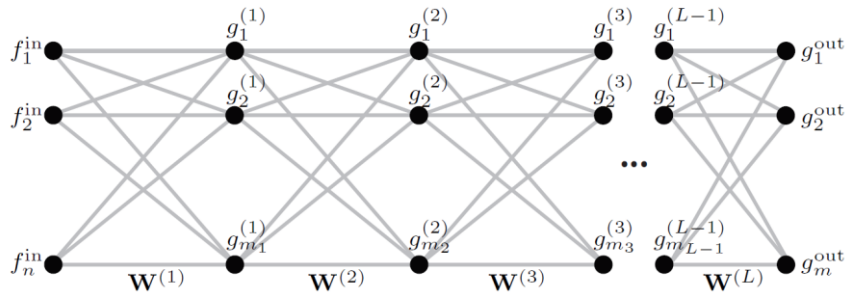
- Computing the gradients by hand is infeasible.
- Finite differences require $\mathcal{O}(\#weights)$ evaluations of ℓ_Θ.
- Using the chain rule is sub-optimal.

A computational technique called back-propagation is used.

14

14

Neural network (NN)



Deep neural network consisting of L layers

- Net output $g^{out} = \sigma(\dots \mathbf{W}^{(2)} \sigma \mathbf{W}^{(1)} \mathbf{f}^{in})$
- Activation, e.g., $\sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)
- Parameters weights of all layers $\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}$ (including biases)

15

15

The need for priors

Deep feed-forward networks are provably universal.

However:

- We can make them arbitrarily complex.
- The number of parameters can be huge.
- Very difficult to optimize.
- Very difficult to achieve generalization.

We need additional priors as a (partial) remedy to the above.

Look for “universal" priors that are task-independent to some extent.

Task-independent priors must come with the data.

16

16

Structure as a strong prior

Key insight: Data often carries structural priors in terms of repeating patterns, compositionality, locality, …

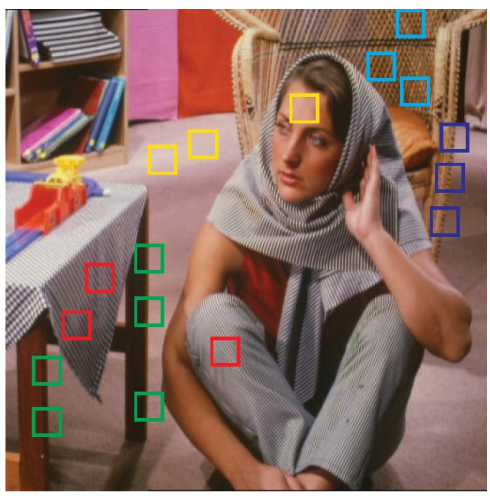
Take advantage of the structure of the data.

17

17

Self-similarity

Data tends to be self-similar across the domain:



18

18

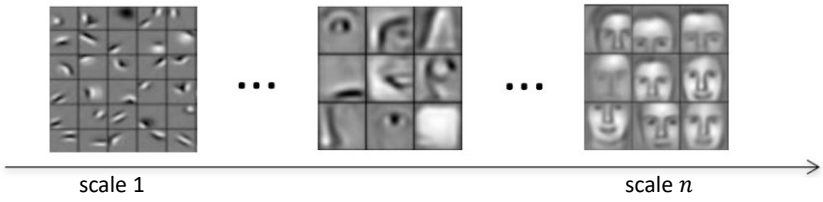
Hierarchy and compositionality

Translation invariance is desirable across multiple scales:

We expect local features to be invariant to their location in the image:

$$z(T_v p) = z(p) \quad \forall p, T_v$$

where p are image patches of variable size.



19

19

Convolutional neural networks (CNN)

Data is often composed of hierarchical, local, shift-invariant patterns.

CNNs directly exploit this fact as a prior.

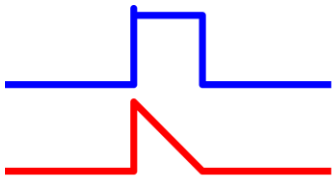
20

20

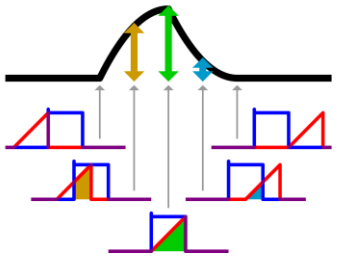
Convolution

Given two functions $f, g : [-\pi, +\pi] \to \mathbb{R}$ their convolution is a function:

$$\underbrace{(f * g)(x)}_{feature\ map} = \int_{-\pi}^{+\pi} f(t)\, \underbrace{g(x-t)}_{kernel}\, dt$$



$f * g$



d'Alembert 1754

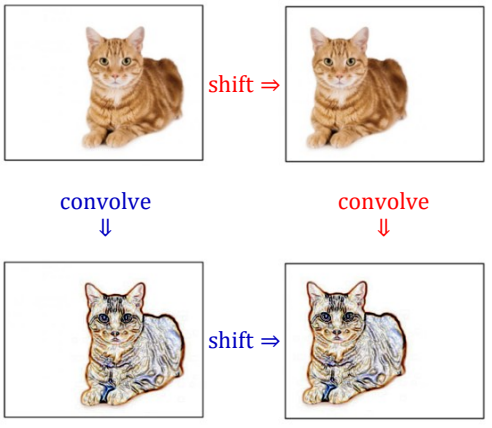
21

21

Convolution: Shift-equivariance

Convolution is shift-equivariant:

$$f(x - x_0) * g(x) = (f * g)(x - x_0)$$



22

22

Convolution: Linearity

We can see convolution as the application of a linear operator $\mathcal{G}$:

$$\mathcal{G}f(x) = (f * g)(x) = \int_{-\pi}^{+\pi} f(t)\, \underbrace{g(x-t)}_{kernel}\, dt$$

It is easy to show that $\mathcal{G}$ is linear:

$$\mathcal{G}\big(\alpha f(x)\big) = \alpha \mathcal{G}\big(f(x)\big)$$
$$\mathcal{G}(f+h)(x) = \mathcal{G}f(x) + \mathcal{G}h(x)$$

Translation equivariance can then be phrased as:

$$\mathcal{G}(\mathcal{T}f) = \mathcal{T}(\mathcal{G}f)$$

i.e., the convolution and translation operators commute.

23

23

Discrete convolution

In the discrete setting, we deal with vectors $\mathbf{f}$, $\mathbf{g}$.
We define the convolution sum:

$$(\mathbf{f} * \mathbf{g})[n] = \sum_{k=-\infty}^{+\infty} \mathbf{f}[k]\mathbf{g}[n-k]$$

Assuming cyclic boundary conditions, the convolution operator can be encoded as a Toeplitz matrix:

$$\mathbf{f} * \mathbf{g} = \begin{pmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{pmatrix} \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}$$

24

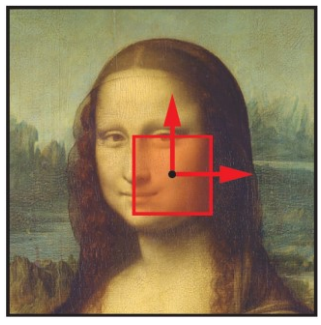
24

Discrete convolution

On 2D domains (e.g., RGB images) $f: \mathbb{R}^2 \to \mathbb{R}^3$, for each channel:

$$(\mathbf{f} * \mathbf{g})[m,n] = \sum_k \sum_\ell \mathbf{f}[k,\ell]\mathbf{g}[m-k, n-\ell]$$

We get the classical interpretation in terms of a moving window:

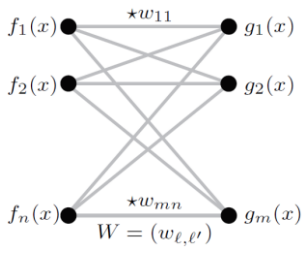


25

25

Convolutional neural network (CNN)

Main idea: Compose equivariant layers implemented via convolution.



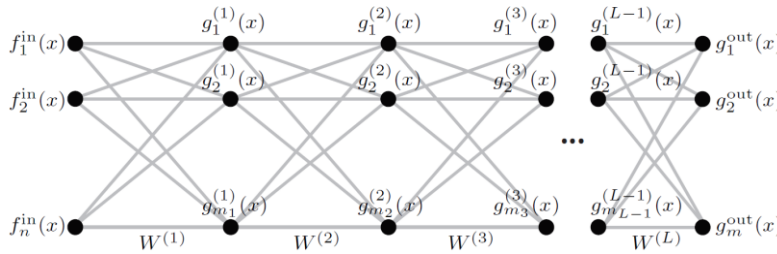
Single convolutional layer

- Conv. layer $\quad g_\ell(x) = \sigma\left(\sum_{\ell'=1}^{n} (f_{\ell'} * w_{\ell,\ell'})(\mathbf{x})\right) \quad \ell = 1, \dots, m \quad \ell' = 1, \dots, n$
- Activation, e.g., $\quad \sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)
- Parameters $\quad$ filters W

26

26

Convolutional neural network (CNN)



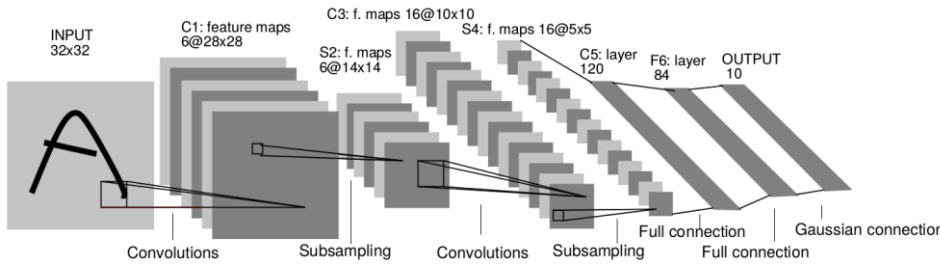
Multiple convolutional layer

- Conv. layer $\quad g_\ell^{(k)}(x) = \sigma\left(\sum_{\ell'=1}^{m_{k-1}} (g_{\ell'}^{(k-1)} * w_{\ell,\ell'}^{(k-1)})\right)(\mathbf{x}) \quad \ell = 1, \dots, m_k \quad \ell' = 1, \dots, m_{k-1}$
- Activation, e.g., $\quad \sigma(x) = \max\{x, 0\}$ rectified linear unit (ReLU)
- Parameters $\quad$ filters of all layers $W^{(1)}, \dots, W^{(L)}$

27

27

Key properties of CNNs



- Convolutional filters (**Translation equivariance**)
- Multiple layers (**Compositionality**)
- Filters localized in space (**Locality**)
- Weight sharing (**Self-similarity**)
- $\mathcal{O}(1)$ parameters per filter (independent of input image size n)

28

28

BACK TO GEOMETRIC LEARNING

29

Non-Euclidean convolution?

	Euclidean	**Non-Euclidean**
Spatial domain	$(f * g)(x) = \int_{-\pi}^{+\pi} f(x')g(x - x')\, dx'$	?
Spectral domain	$(\widehat{f * g})(\omega) = \hat{f}(\omega) \cdot \hat{g}(\omega)$	?

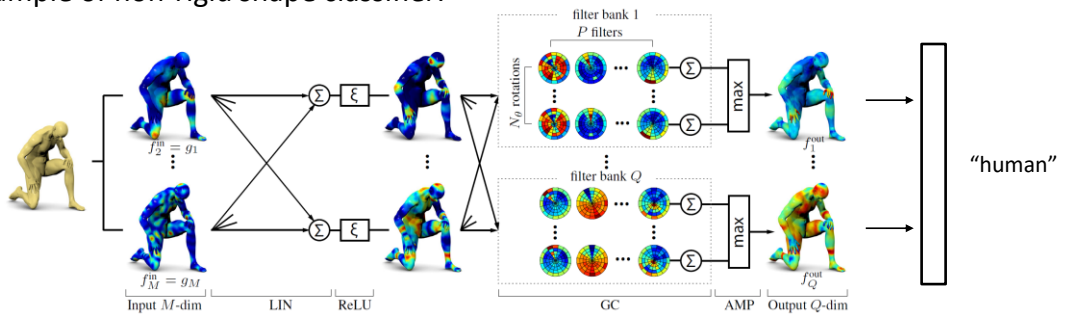
"Convolution Theorem"

30

30

Non-Euclidean convolution?

Example of non-rigid shape classifier:



Input: Vertex-wise quantity, e.g., curvature, texture, SHOT descriptors

Output: Shape category

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

31

31

Geodesic convolution on meshes

Local system of geodesic polar coordinates constructed at point x to extract patches on the manifold.

- The radial coordinate is constructed as ρ-level sets $\{x' : d_X(x, x') = \rho\}$ of the geodesic (shortest path) distance function for $\rho \in [0, \rho_0]$, where ρ_0 is the radius of the geodesic disc
- The angular coordinate is constructed as a set of geodesics $\Gamma(x, \theta)$ emanating from x in direction θ; such rays are perpendicular to the geodesic distance level sets

Construction of local geodesic polar coordinates on a manifold: example of local geodesic patches

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

32

32

Geodesic convolution on meshes

Let $\Omega(x): B_{\rho_0}(x) \to [0, \rho_0] \times [0,2\pi]$ denote the bijective map from the manifold into the local geodesic polar coordinates (ρ, Θ) around x, and let $(D(x)f)(\rho, \Theta) = (f \circ \Omega^{-1}(x))(\rho, \Theta)$ be the patch operator interpolating f in the local coordinates.

$(D(x)f)$ can be regarded as a 'patch' on the manifold and used to define the geodesic convolution (GC)

$$(f * a)(x) = \sum_{\Theta, r} a(\Theta + \Delta\Theta, r)(D(x)f)(r, \Theta)$$

where $a(\Theta, r)$ is a filter applied on the patch.

Due to angular coordinate ambiguity, the filter can be rotated by arbitrary angle $\Delta\Theta$.

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

33

33

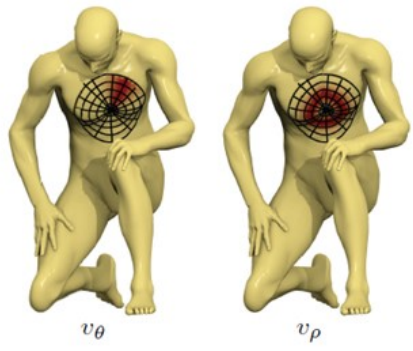
Patch operator

In[1], the patch operator was constructed as

$$(D(x)f)(\rho, \theta) = \int_X v_{\rho,\theta}(x, x') f(x') dx',$$

$$v_{\rho,\theta}(x, x') = \frac{v_\rho(x, x') v_\theta(x, x')}{\int_X v_\rho(x, x') v_\theta(x, x') dx'}.$$

- The radial interpolation weight is a Gaussian $v_\rho(x, x')$ of the geodesic distance from x, centered around ρ
- The angular weight is a Gaussian $v_\theta(x, x')$ of the point-to-set distance $d_X(\Gamma(x, \theta), x') = \min_{x'' \in \Gamma(x,\theta)} d_X(x'', x')$ to the geodesic $\Gamma(x, \theta)$



example of angular and radial weights v_θ, v_ρ, respectively (red denotes larger weights)

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

[1]I. Kokkinos et al. "Intrinsic shape context descriptors for deformable shapes", CVPR 2012

34

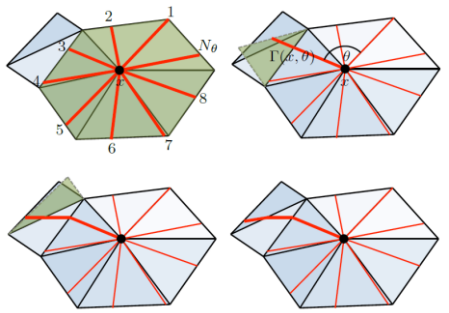
34

Discrete patch operator

On triangular meshes, a discrete local system of geodesic polar coordinates has N_θ angular and N_ρ radial bins.

- The 1-ring of a vertex i is first partitioned by N_θ rays into equi-angular bins, aligning the first ray with one of the edges
- Next, the rays are propagated into adjacent triangles using an unfolding procedure, producing poly-lines that form the angular bins
- Radial bins are created as level sets of the geodesic distance function computed using fast marching

The discrete patch operator is an $N_\theta N_\rho N \times N$ matrix applied to a function defined on the mesh vertices.



Division of 1-ring of vertex x_i into N_θ equi-angular bins; propagation of a ray (bold line) by unfolding the respective triangles (marked in green)

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

35

35

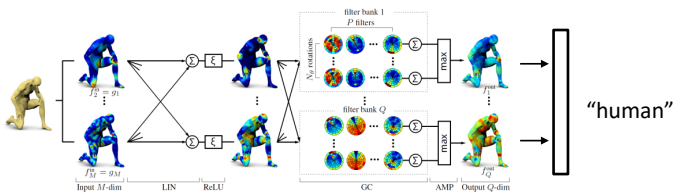
Geodesic convolutional neural network (GCNN)

GCNN consists of several layers that are applied subsequently, i.e., the output of the previous layer is used as the input into the subsequent one.

- Linear (LIN) layer typically follows the input layer and precedes the output layer to adjust the input and output dimensions by means of a linear combination:

$$f_q^{\text{out}}(x) = \xi\left(\sum_{p=1}^{P} w_{qp} f_p^{\text{in}}(x)\right); \quad q = 1, \ldots, Q,$$

optionally followed by a non-linear function such as the ReLU.



Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

36

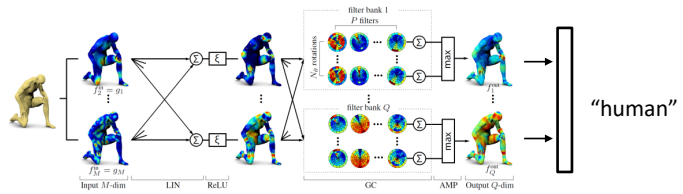
36

Geodesic convolutional neural network (GCNN)

- Geodesic convolution (GC) layer replaces the convolutional layer used in classical Euclidean CNNs. Due to the angular coordinate ambiguity, the geodesic convolution result is computed for all N_θ rotations of the filters,

$$f^{\text{out}}_{\Delta\theta,q}(x) = \sum_{p=1}^{P} (f_p \star a_{\Delta\theta,qp})(x), \quad q = 1, \ldots, Q,$$

where $a_{\Delta\theta,qp}(\theta, r) = a_{qp}(\theta + \Delta\theta, r)$ are the coefficients of the p-th filter in the q-th filter bank rotated by $\Delta\theta = 0, \frac{2\pi}{N_\theta}, \ldots, \frac{2\pi(N_\theta - 1)}{N_\theta}$ and the convolution is the GC introduced above.



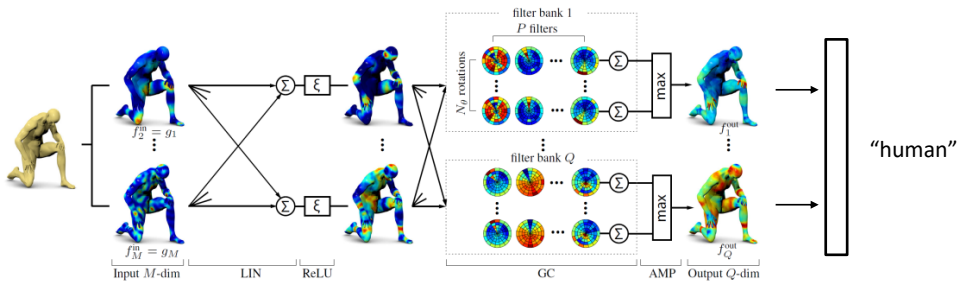
Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

Geodesic convolutional neural network (GCNN)

- Angular max-pooling (AMP) is a fixed layer used in conjunction with the GC layer that computes the maximum over the filter rotations,

$$f^{\text{out}}_p(x) = \max_{\Delta\theta} f^{\text{in}}_{\Delta\theta,p}(x), \quad p = 1, \ldots, P = Q,$$

where $f^{in}_{\Delta\vartheta_p}$ is the output of the GC layer.



Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

Geodesic convolutional neural network (GCNN)

Other layers are

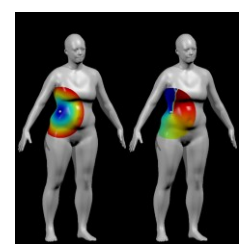
- Fourier transform magnitude (FTM) layer is a fixed layer that applies the patch operator to each input dimension, followed by Fourier transform w.r.t. the angular coordinate and absolute value. The Fourier transform translates rotational ambiguity into complex phase ambiguity, which is removed by taking the absolute value
- Covariance (COV) layer is used in applications such as retrieval, where one needs to aggregate the point-wise descriptors and produce a global shape descriptor

Masci et al, "Geodesic convolutional neural networks on Riemannian manifolds", CVPR 2015

39

39

Spatial convolution on meshes

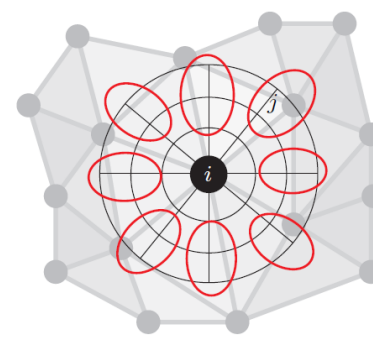


Intrinsic local polar coordinates (ρ, Θ) on manifold around a point marked in white

Local system of coordinates u_{ij} around i (e.g., geodesic polar).

Local weights $w(\mathbf{u}_{ij})$, e.g., Gaussians with learnable μ, Σ

$$w = \exp(-(\mathbf{u}_{ij} - \mu)^{\mathrm{T}} \Sigma^{-1} (\mathbf{u}_{ij} - \mu))$$



Spatial convolution of feature f with filter g:

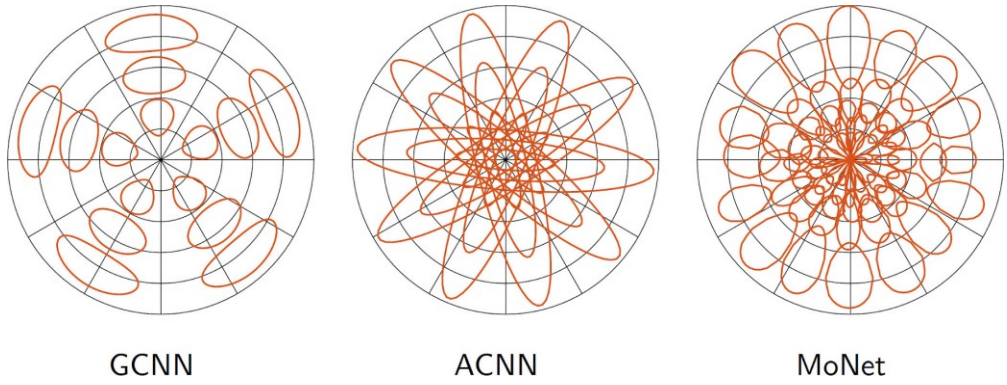
- Represent the input f as above $\Rightarrow \mathbf{f}$
- Represent the learnable filter g as above $\Rightarrow \mathbf{g}$
- Sum up the element-wise products $\Rightarrow \mathbf{f}^{\mathrm{T}}\mathbf{g}$

Monti et al, "Geometric deep learning on graphs and manifolds using mixture model CNNs", CVPR 2016

40

40

Local weighting kernels



Patch operator weighting functions $w_i(\rho, \Theta)$ used in different generalizations of convolution on the manifold (hand-crafted in GCNN and ACNN and learned in MoNet).
All kernels are L1-normalized; red curves represent the 0.5 level set.

Monti et al, “Geometric deep learning on graphs and manifolds using mixture model CNNs", CVPR 2016

41

41

Spiral convolution

These approaches aggregate neighboring node features based on trainable weight functions.

With the spiral convolution method, node features are encoded under an explicitly defined spiral sequence, and a fully connected layer follows to encode input features combined with ordering information.

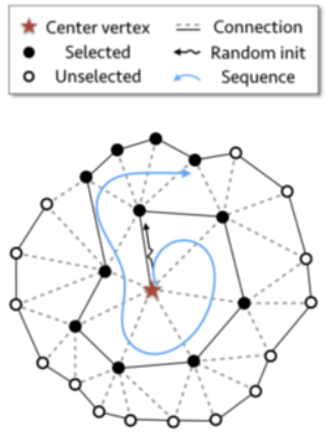
The definition of the spiral sequences, is the core step of the proposed operator.

42

42

Spiral convolution

Given a center vertex, the sequence can be enumerated by intuitively following a spiral.



The degrees of freedom are the orientation within each ring (clockwise or counter-clockwise) and the choice of the starting direction.

The orientation is fixed to counter-clockwise and an arbitrary starting direction is chosen.

The spirals are pre-computed only once.

Lim et al, "A Simple Approach to Intrinsic Correspondence Learning on Unstructured 3D Meshes", ECCV 2018
Gong et al, "SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator", GMDL 2019

43

43

Spiral convolution

A k-ring and a k-disk are defined around a center vertex v as follows:

$$\begin{aligned} 0\text{−ring}(v) &= \{v\}, \\ k\text{−disk}(v) &= \cup_{i=0,\ldots,k} i\text{−ring}(v), \\ (k+1)\text{−ring}(v) &= \mathcal{N}(k\text{−ring}(v)) \backslash k\text{−disk}(v), \end{aligned}$$

where $\mathcal{N}(V)$ is the set of all vertices adjacent to any vertex in set V.

The spiral length is denoted as l.

A spiral sequence $S(v, l)$ is an ordered set consisting of l vertices from a concatenation of k-rings

$$\mathrm{S}(v, l) \subset (0\text{−ring}(v), 1\text{−ring}(v), \ldots, k\text{−ring}(v)).$$

Gong et al, "SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator", GMDL 2019

44

44

Spiral convolution

A common extension of CNNs into irregular domains, such as graphs, is typically expressed as a neighborhood aggregation or message passing scheme.

With $x_i^{(k-1)} \in \mathbb{R}^F$ denoting node features of node i and $e_{i,j}^{(k-1)} \in \mathbb{R}^D$ denoting (optional) edge features from node i to node j in layer $(k-1)$, message passing GNNs can be described as:

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathbf{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \, \phi^{(k)}(\mathbf{x}_i^{(k-1)}, \mathbf{x}_j^{(k-1)}, \mathbf{e}_{i,j}^{(k-1)}) \right)$$

where $x_i^k \in \mathbb{R}^{F'}$, $\square$ denotes a differentiable permutation-invariant function, e.g., sum, mean or max, and ϕ denotes a differentiable kernel function. γ represents MLPs.

Gong et al, "SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator", GMDL 2019

45

45

Spiral convolution

The main challenge in the case of irregular domains is to define the correspondence between neighbors and weight matrices which relies on the kernel function ϕ.

Thanks to the nature of the spiral serialization of neighboring nodes, the spiral convolution can be defined in an equivalent manner to the Euclidean CNNs, easing the pain of calculating the assignment value of x_j to the weight matrix.

The spiral convolution operator for a node i is defined as

$$\mathbf{x}_i^{(k)} = \gamma^{(k)} \left(\mathop{\|}_{j \in S(i,l)} \mathbf{x}_j^{(k-1)} \right)$$

where γ denotes MLPs and $||$ is the concatenation operation.

Note that node features are concatenated in the spiral sequence following the order defined in $S(i,l)$.

Gong et al, "SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator", GMDL 2019

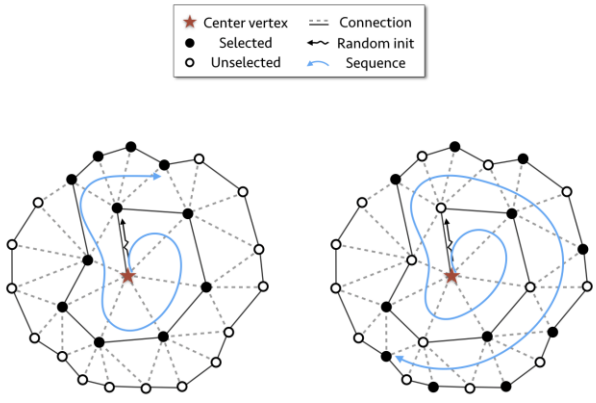
46

46

Dilated spiral convolution

With the motivation of exponentially expanding the receptive field without losing resolution or coverage, dilated spiral convolution operators are also defined.

Spiral convolution operators could immediately gain the power of capturing multi-scale contexts without increasing complexity from uniformly sampling the spiral sequence, while keeping the same spiral length.

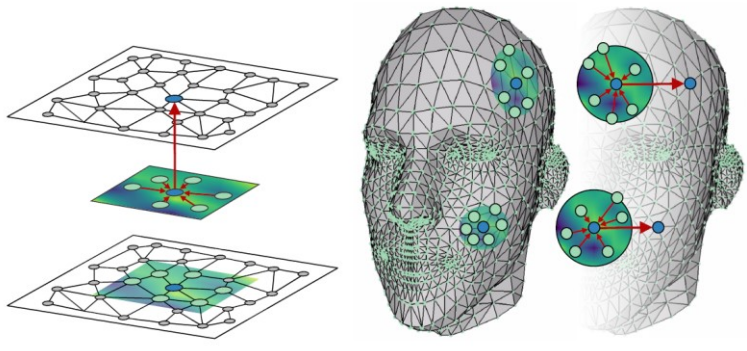


(a) Spiral++ (b) DilatedSpiral++

47

47

Other spatial convolutions



Fey et al, “SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels", CVPR 2018

48

48

PyTorch geometric

latest

Search docs

NOTES

Installation

Introduction by Example

Creating Message Passing Networks

Creating Your Own Datasets

Advanced Mini-Batching

Memory-Efficient Aggregations

TorchScript Support

Colab Notebooks

External Resources

PACKAGE REFERENCE

torch_geometric

torch_geometric.nn

Convolutional Layers

Dense Convolutional Layers

Normalization Layers

`SignedConv`	The signed graph convolutional operator from the "Signed Graph Convolutional Network" paper
`DNAConv`	The dynamic neighborhood aggregation operator from the "Just Jump: Towards Dynamic Neighborhood Aggregation in Graph Neural Networks" paper
`PointConv`	The PointNet set layer from the "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation" and "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space" papers
`GMMConv`	The gaussian mixture model convolutional operator from the "Geometric Deep Learning on Graphs and Manifolds using Mixture Model CNNs" paper
`SplineConv`	The spline-based convolutional operator from the "SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels" paper
`NNConv`	The continuous kernel-based convolutional operator from the "Neural Message Passing for Quantum Chemistry" paper.
`ECConv`	alias of `torch_geometric.nn.conv.nn_conv.NNConv`
`CGConv`	The crystal graph convolutional operator from the "Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties" paper
`EdgeConv`	The edge convolutional operator from the "Dynamic Graph CNN for Learning on Point Clouds" paper

49

49

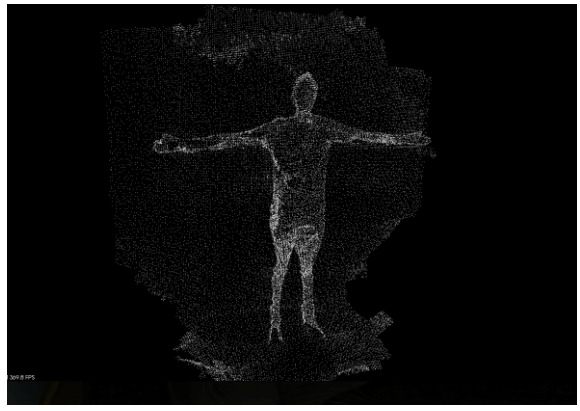
POINT CLOUDS

50

Point cloud

Point cloud data ara typically collected from either a lidar or radar sensor.

Unlike 2D pixel arrays (images) or 3D voxel arrays, point clouds have an unstructured representation in that the data is simply a collection (a set) of the points captured during a lidar or radar sensor scan.



51

51

Input data

To leverage existing techniques built around (2D and 3D) convolutions, many researchers and practitioners often discretize a point cloud by taking multi-view projections onto 2D space or quantizing it to 3D voxels.

Given that the original data is manipulated, either approach can have negative impacts.

For simplicity, we will assume that a point in a point cloud is fully described by its (x, y, z) coordinates.

In practice, other features may be included, such as surface normal and intensity.

52

52

PointNet

PointNet is a seminal paper in 3D perception, applying deep learning to point clouds for object classification and part/scene semantic segmentation.

PointNet consumes raw point cloud data, so it is based on an architecture that conforms to the unique properties of point sets.

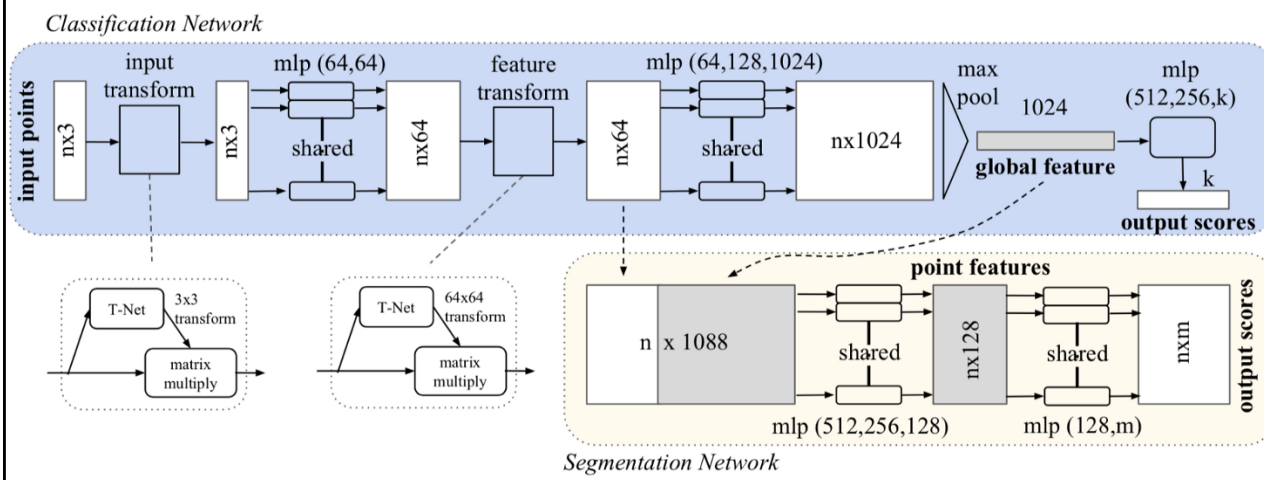
- Permutation (order) invariance**:** given the unstructured nature of point cloud data, a scan made up of N points has $N!$ permutations. The subsequent data processing must be invariant to the different representations
- Transformation invariance: classification and segmentation outputs should be unchanged if the object undergoes certain transformations, including rotation and translation
- Point interactions: the interaction between neighboring points often carries useful information (i.e., a single point should not be treated in isolation). Whereas classification need only to make use of global features, segmentation must leverage local point features along with global point features

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

53

53

Architecture



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

54

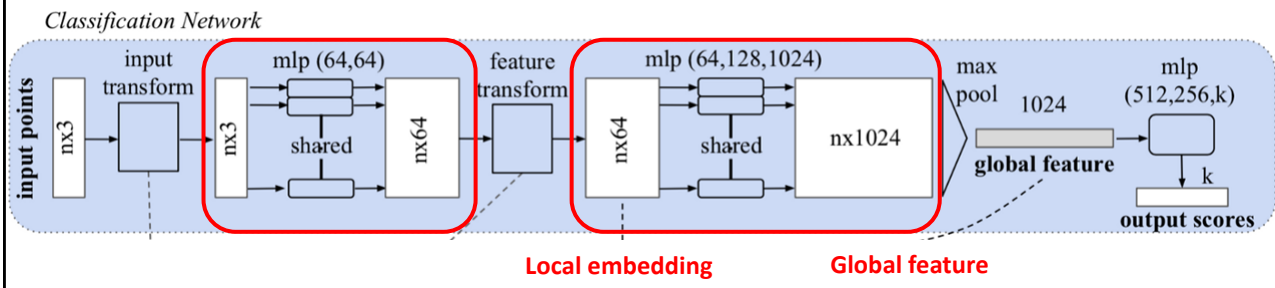
54

Architecture: classification network

A shared multi-layer perceptron (MLP) is used to map each of the n points from three dimensions (x, y, z) to 64 dimensions.

- It is important to note that a *single* multi-layer perceptron is shared for each of the n points (i.e., mapping is identical and independent on the n points)

This procedure is repeated to map the n points from 64 to 1024 dimensions.



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

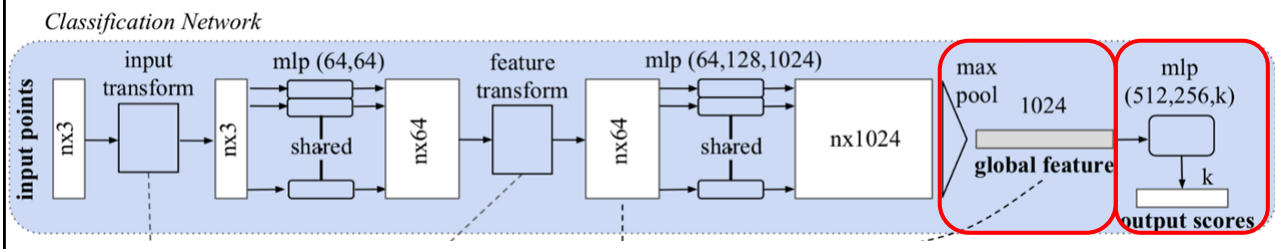
55

55

Architecture: classification network

With the points in a higher-dimensional embedding space, max pooling is used to create a global feature vector in $\mathbb{R}^{1024}$.

Finally, a three-layer fully-connected network is used to map the global feature vector to k output classification scores.



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

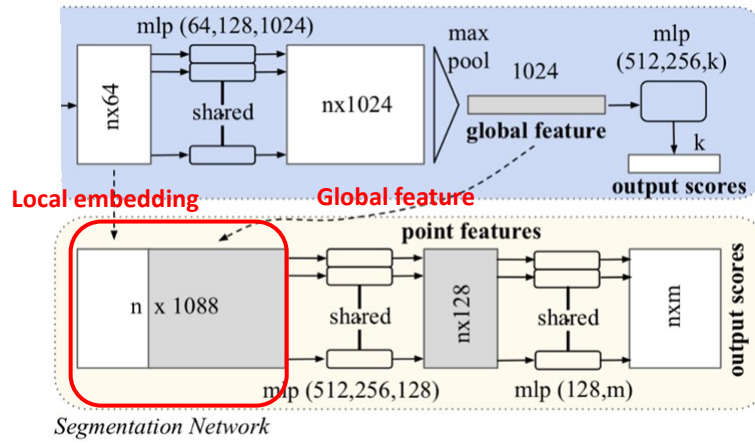
56

56

Architecture: segmentation network

Each of the n input points needs to be assigned to one of m segmentation classes.

Because segmentation relies on local and global features, the points in the 64-dimensional embedding space (local point features) are concatenated with the global feature vector (global point features), resulting in a per-point vector in $\mathbb{R}^{1088}$.



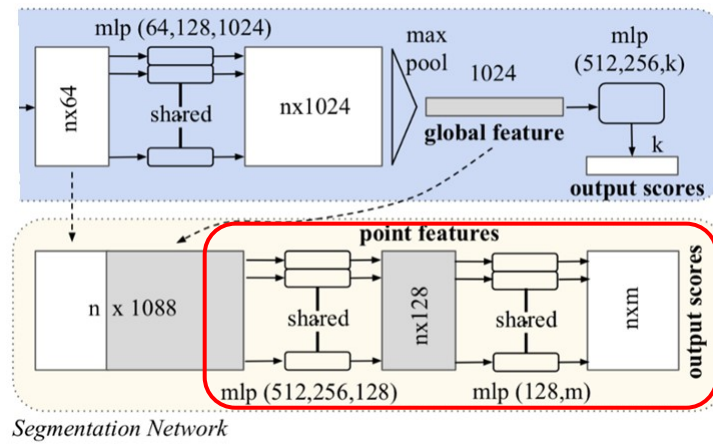
R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

57

57

Architecture: segmentation network

Similar to the multi-layer perceptrons used in the classification network, MLPs are used (identically and independently) on the n points to lower the dimensionality from 1088 to 128 and again to m, resulting in an array of $n \; x \; m$.



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

58

58

Permutation invariance

Point clouds are inherently unstructured data and are represented as numerical sets.

Specifically, given N data points, there are $N!$ permutations

In order to make PointNet invariant to input permutations, symmetric functions are used (i.e., functions whose value given n arguments is the same regardless of the order of the arguments).

For binary operators, this is also known as the commutative property.

Common examples include

$$sum(a,b) = sum(b,a)$$
$$average(a,b) = average(b,a)$$
$$\max(a,b) = \max(b,a)$$

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

59

59

Permutation invariance

The symmetric function is used once the n input points are mapped to higher-dimensional space.

The result is a global feature vector that aims to capture an aggregate signature of the n input points.

Naturally, the expressiveness of the global feature vector is tied to the dimensionality of it (and thus the dimensionality of the points that are input to the symmetric function).

The global feature vector is used directly for classification and is used alongside local point features for segmentation.

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

60

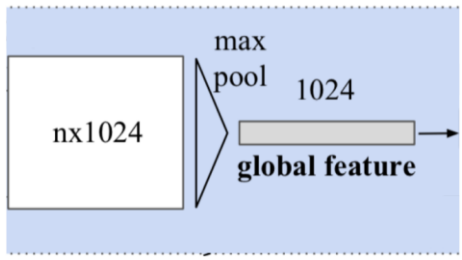
60

Permutation invariance

PointNet implements the symmetric function with max pooling.

Alternatives, including summing and averaging, produced inferior results.

	accuracy
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	**87.1**



Usage of max pool as symmetric function

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

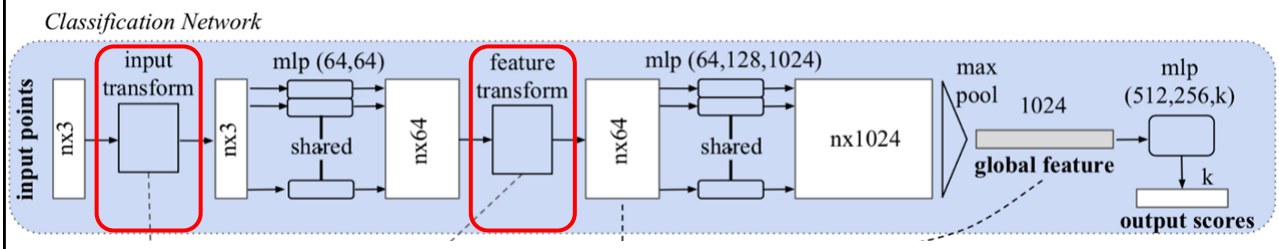
61

61

Transformation invariance

The classification (and segmentation) of an object should be invariant to certain geometric transformations (e.g., rotation).

Motivated by Spatial Transformer Networks (STNs)[1], the "input transform" and "feature transform" are modular sub-networks that seek to provide pose normalization for a given input.



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

[1]M. Jaderberg, et al, "Spatial Transformer Network", 2015

62

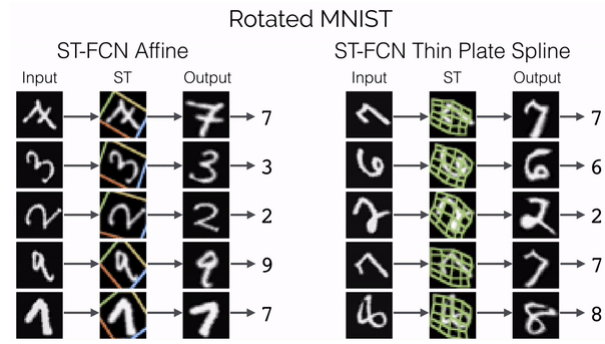
62

Transformation invariance

How STNs operate[1].

- The ST provides pose normalization to an otherwise rotated input
- Using this type of pose normalization in a digit classifier would relax the constraints of a downstream algorithm and reduce the extent to which data augmentation is needed

Pose normalization is beneficial in the case of point clouds as well, as objects can similarly take on an unlimited number of poses.



Various inputs and corresponding outputs of a Spatial Transformer

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

[1] https://towardsdatascience.com/review-stn-spatial-transformer-network-image-classification-d3cbd98a70aa

63

63

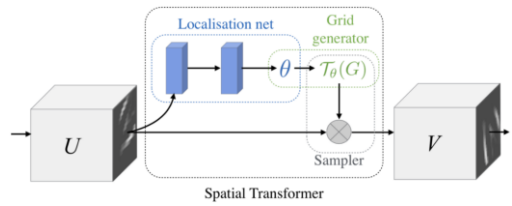
Transformation invariance

Based on input U, a small regression network, the *localization net*, outputs transformation parameter θ.

To construct output V given U and θ, a grid generator and sampler are used.

- Imagine that the output of a localization net corresponds to rotating a handwritten "7" by an angle θ; in order to create a new image with the proper rotation, the original image needs to undergo appropriate sampling
- Note that the ST is not confined to the input space and can operate on any downstream feature/embedding space

Components of the Spatial Transformer



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

64

64

T-Net

Going back to PointNet, a similar approach can be taken: for a given input point cloud, apply an appropriate rigid or affine transformation to achieve pose normalization.

Because each of the n input points are represented as a vector and are mapped to the embedding spaces independently, applying a geometric transformation simply amounts to matrix multiplying each point with a transformation matrix.

Unlike the image-based application of Spatial Transformers, no sampling is needed.

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

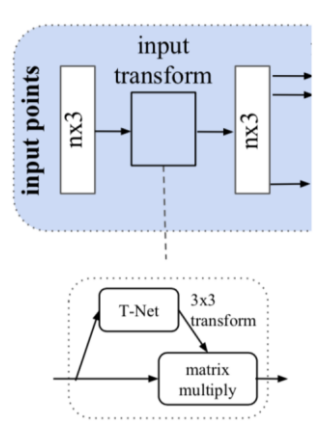
65

65

T-Net

Similar to the *localization net* in STs, the T-Net is a *regression network* that is tasked with predicting an input-dependent 3-by-3 transformation matrix that is then matrix multiplied with the n-by-3 input.

Snapshot of the input transform



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

66

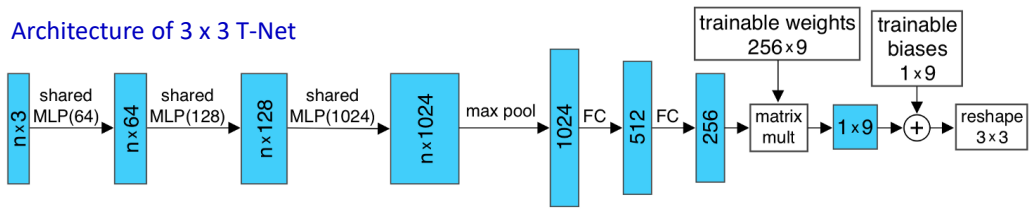
66

T-Net

The operations comprising the T-Net are motivated by the higher-level architecture of PointNet.

MLPs (or fully-connected layers) are used to map the input points independently and identically to a higher-dimensional space; max pooling is used to encode a global feature vector whose dimensionality is then reduced to $\mathbb{R}^{256}$ with FC layers.

The input-dependent features at the final FC layer are then combined with globally trainable weights and biases, resulting in a 3-by-3 transformation matrix.



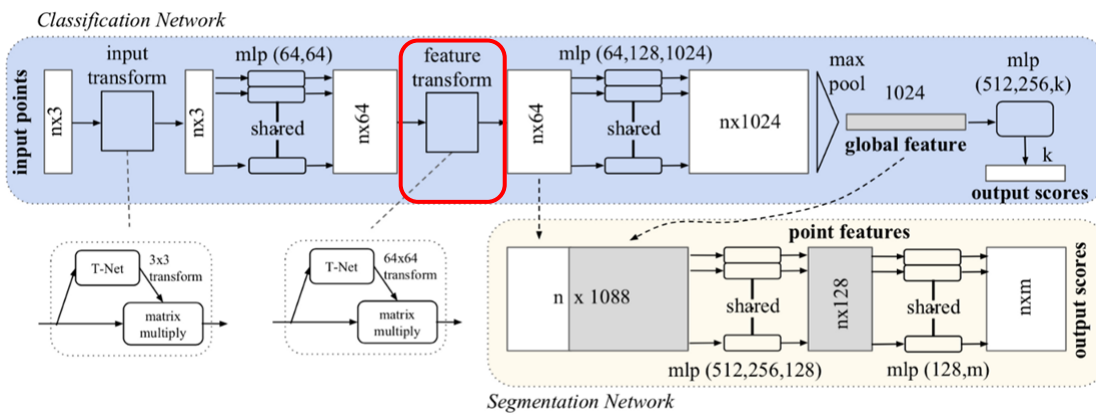
R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

67

67

T-Net

The concept of pose normalization is extended to the 64-dimensional embedding space ("*feature transform*" block in the overall architecture).



R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

68

68

T-Net

The corresponding T-Net is nearly identical to the 3 x 3 T-Net except for the dimensionality of the trainable weights and biases, which become 256-by-4096 and 4096, respectively, resulting in a 64-by-64 transformation matrix.

The increased number of trainable parameters leads to the potential for overfitting and instability during training, so a regularization term is added to the loss function.

The regularization term encourages the resulting 64-by-64 transformation matrix (represented as A below) to approximate an orthogonal transformation

$$\mathcal{L}_{reg} = \|I - AA^T\|^2$$

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

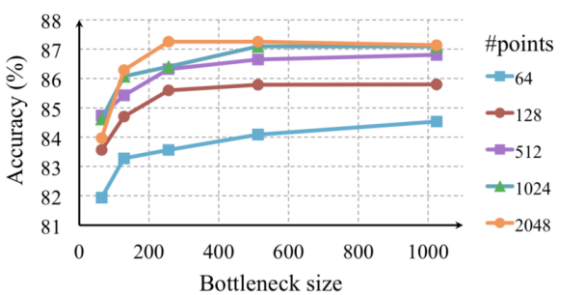
69

69

Analysis and visualization

There is a considerable amount of intuition that can be drawn from the global feature vector.

- The dimensionality of the vector, referred to by the authors as the bottleneck dimension and symbolized by K, relates directly to the expressiveness of the model.
- A larger value of K leads to a more complex — and, likely, accurate — model, and vice versa. For reference, PointNet is designed with $K = 1024$.



PointNet accuracy across K and number of points comprising an input point cloud

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

70

70

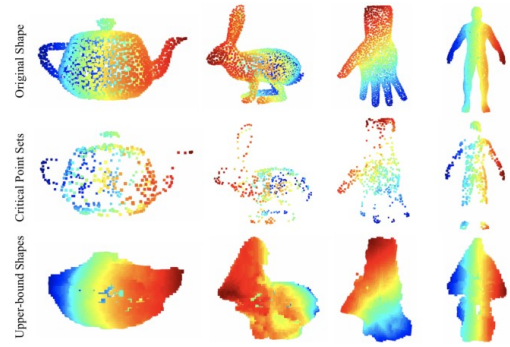
Analysis and visualization

The feature vector was the result of applying a symmetric function (for permutation invariance)

- PointNet makes use of max pooling

Similar to using the max operator to compress multiple real-valued inputs to a single value, the output of max pooling compresses the n points of the input point cloud to a subset of points.

- At most K points can contribute to the global feature vector
- The points that do contribute to and define the global feature vector are referred to as the *critical point set* and encode the input with a sparse set of key points



Visualization of critical point sets and upper-bound shapes

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

71

71

Analysis and visualization

Similar to how the output of the max operator is unchanged by inputs that are lesser than the true maximum, there exists a bound on input points that won't impact the global feature vector.

In the previous figure, this bound is represented by the *upper-bound shape*.

Note that noise beyond the upper-bound shape alters the global feature vector but may not necessarily result in misclassification.

In summary, the global feature vector is unchanged for points between the critical point set and the upper-bound shape, resulting in considerable robustness.

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

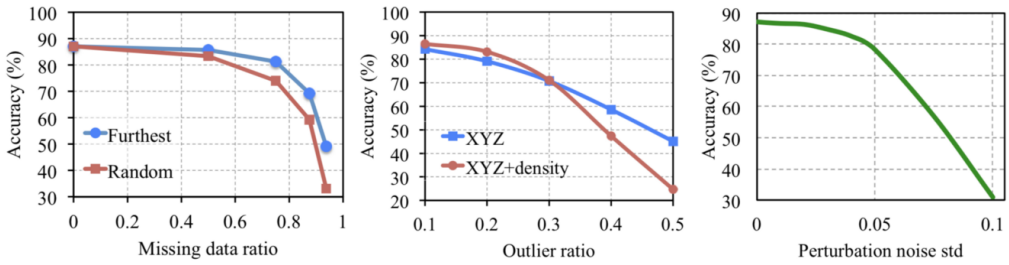
72

72

Analysis and visualization

The robustness described above can be visualized in a more quantitative manner, as shown below.

Missing data refers to deleting points from the input point cloud, whereas outlier refers to insertion of random/noisy points.



PointNet robustness test

R. Charles, et al., "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," CVPR, 2017

73

73

PointNet++

PointNet++ is an improved version of PointNet.

PointNet does not capture local structures induced by the metric space points live in, limiting its ability to recognize fine-grained patterns and generalizability to complex scenes.

PointNet++ applies PointNet recursively on a nested partitioning of the input point set.

By exploiting metric space distances, PointNet++ is able to learn local features with increasing contextual scales.

C. R. Qi et al, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space", 2017

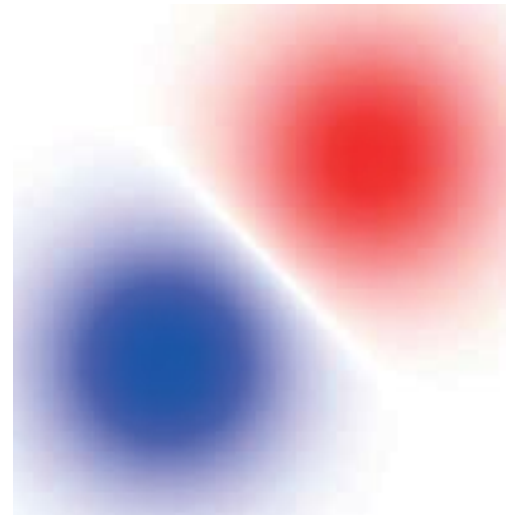
74

74

SPECTRAL CONVOLUTION

75

Laplace operator: Geometric intuition



Smooth scalar field f

76

76

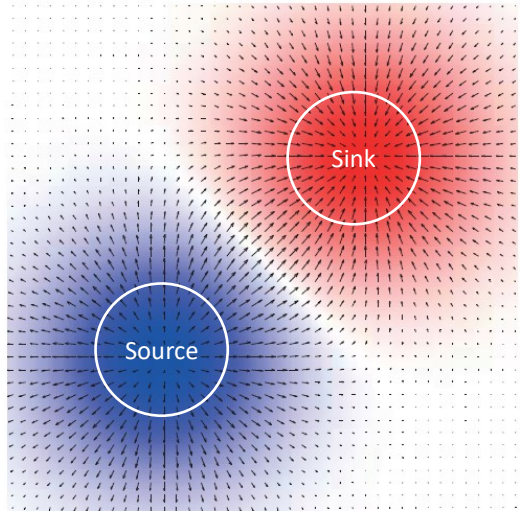
Laplace operator: Geometric intuition

- Gradient $\nabla f(x)$

`direction of the steepest increase of f at x'

- Divergence $\mathrm{div}(F(x))$

`scalar density of an outward flux of F from an infinitesimal volume around x'



Smooth vector field F

77

77

Laplace operator: Geometric intuition

- Gradient $\nabla f(x)$

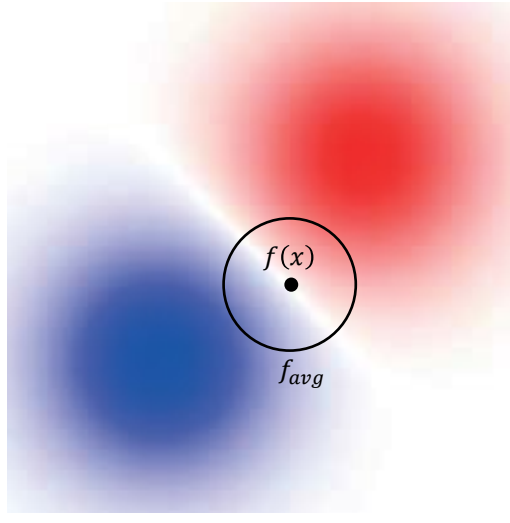
`direction of the steepest increase of f at x'

- Divergence $\mathrm{div}(F(x))$

`scalar density of an outward flux of F from an infinitesimal volume around x'

- Laplacian $\Delta f(x) = -\mathrm{div}(\nabla f(x))$

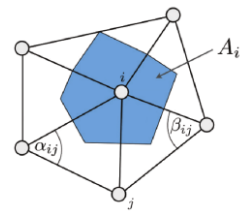
`scalar difference between $f(x)$ and the average of f on an infinitesimal sphere around x'



78

78

Discrete Laplacian



The discrete Laplace operator on a mesh is the $n \times n$ matrix:

$$\mathbf{L} = \mathbf{A}^{-1}\mathbf{S}$$

where

$$s_{ij} = \begin{cases} -\frac{1}{2}(\cot\alpha_{ij} + \cot\beta_{ij}) & \text{if } e_{ij} \in E \\ -\sum_{k \neq i} s_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Cotangent matrix or stiffness matrix

$$a_{ij} = \begin{cases} \frac{1}{12}(A(T_{jii'}) + A(T_{ji''i})) & \text{if } e_{ij} \in E \\ \frac{1}{6}\sum_{k \in \mathcal{N}(i)} A(T_k) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Mass matrix

A similar formula defines the graph Laplacian.

...and other Laplacian operators on irregular domains.

79

79

Self-adjointness of Δ and Laplacian eigenvalues

For the Laplacian Δ it can be easily shown:

$$\langle f, \Delta g \rangle = \langle \Delta f, g \rangle$$

i.e., Δ is self-adjoint.

Self-adjoint operators have real eigenvales:

$$\Delta\phi = \lambda\phi$$

These are countable and are canonically ordered non-decreasingly:

$$0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \cdots \to \infty$$

The sequence of eigenvalues is called the spectrum of ϕ.

80

80

Laplacian eigenfunctions

$$\Delta\phi = \lambda\phi$$

Consider distinct eigenfunctions ϕ_i, ϕ_j with $\lambda_i \neq \lambda_j$. We have:

$$\langle \phi_i, \Delta\phi_j \rangle = \langle \Delta\phi_i, \phi_j \rangle$$

It follows that

$$\lambda_j \langle \phi_i, \phi_j \rangle = \lambda_i \langle \phi_i, \phi_j \rangle$$

can only be true if $\langle \phi_i, \phi_j \rangle = 0$

Therefore, the eigenfunctions of ϕ are orthogonal, and can be rescaled to be orthonormal:

$$\langle \phi_i, \phi_j \rangle = \delta_{ij}$$

where $\delta_{ij} = 1$ if $i = j$, and 0 otherwise.

81

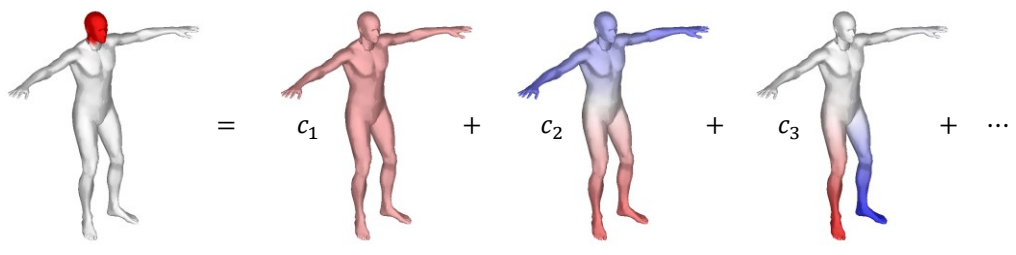
81

Spectral theorem

$$\Delta\phi = \lambda\phi$$

Now recall that the Laplacian $\Delta: \mathcal{F}(\mathcal{X}) \to \mathcal{F}(\mathcal{X})$ operates on a vector space of functions $\mathcal{F}(\mathcal{X})$.

Theorem: The eigenfunctions $\{\phi_i\}$ of Δ form an orthonormal basis of $\mathcal{F}(\mathcal{X})$.

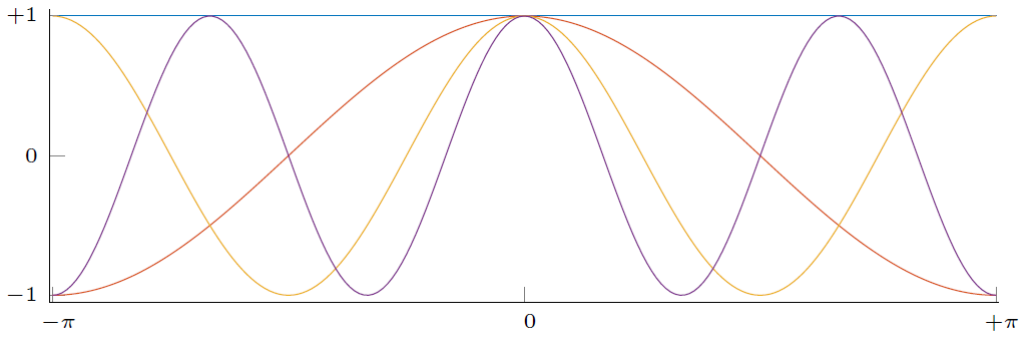


f ϕ_1 ϕ_2 ϕ_3

82

82

Laplacian eigenfunctions: Euclidean

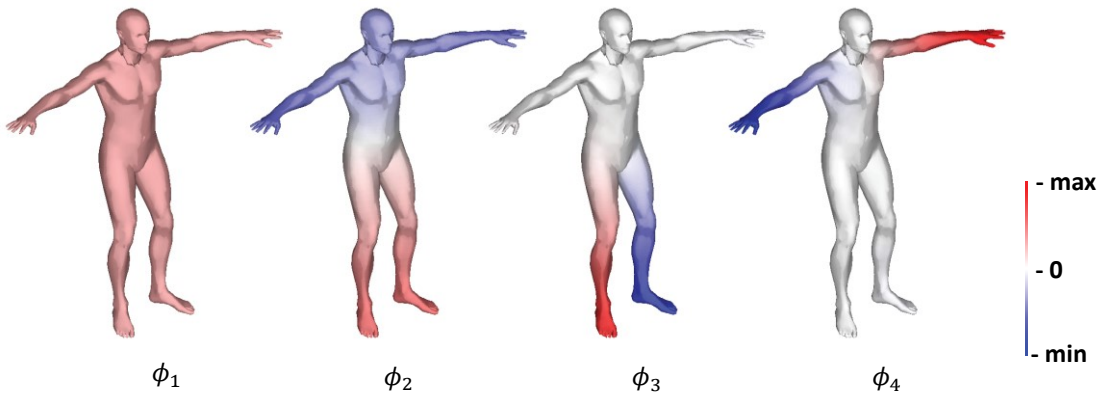


First eigenfunctions of 1D Euclidean Laplacian = standard Fourier basis

83

83

Laplacian eigenfunctions: manifold

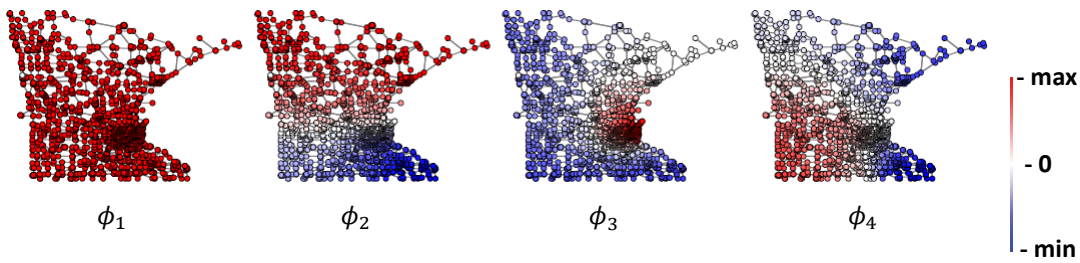


First eigenfunctions of a manifold Laplacian

84

84

Laplacian eigenfunctions: graph



First eigenfunctions of a graph Laplacian

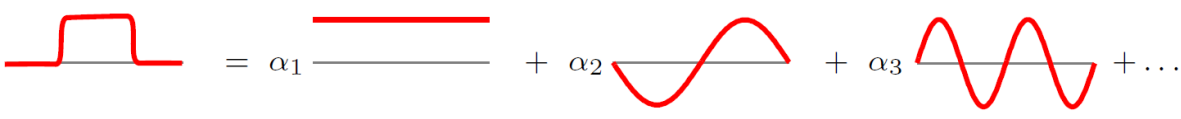
85

85

Fourier analysis: Euclidean space

A function $f: [-\pi, +\pi] \to \mathbb{R}$ can be written as Fourier series

$$f(x) = \sum_{k \geq 0} \underbrace{\frac{1}{2\pi} \int_{-\pi}^{+\pi} f(x') e^{-ikx'} dx'}_{\hat{f}_k = \langle f, e^{ikx} \rangle_{L^2([-\pi,+\pi])}} e^{ikx}$$



Fourier basis = Laplacian eigenfunctions: $\frac{d^2}{dx^2} e^{ikx} = k^2 e^{ikx}$

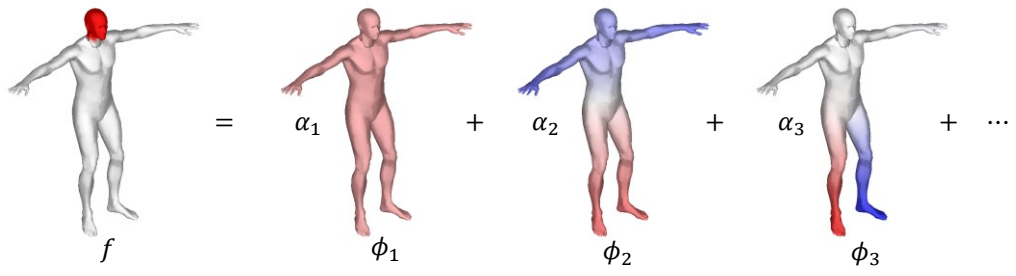
86

86

Fourier analysis: non-Euclidean space

A function $f: \mathcal{X} \to \mathbb{R}$ can be written as Fourier series

$$f(x) = \sum_{k \geq 1} \underbrace{\int_{\mathcal{X}} f(x')\phi_k(x')dx'}_{\hat{f}_k = \langle f, \phi_k \rangle_{L^2(\mathcal{X})}} \phi_k(x)$$



Fourier basis = Laplacian eigenfunctions: $\Delta\phi_k(x) = \lambda_k \phi_k(x)$

87

87

Convolution theorem

Given two functions $f, g: [-\pi, +\pi] \to \mathbb{R}$ their convolution is a function:

$$(f * g)(x) = \int_{-\pi}^{+\pi} f(x')g(x - x')dx'$$

Convolution theorem: Fourier transform diagonalizes the convolution operator ⇒ convolution can be computed in the Fourier domain as:

$$\widehat{(f * g)} = \hat{f} \cdot \hat{g}$$

88

88

Convolution theorem

Convolution of two vectors $\mathbf{f} = (f_1, f_2, \dots, f_n)^{\mathrm{T}}$ and $\mathbf{g} = (g_1, g_2, \dots, g_n)^{\mathrm{T}}$

$$\mathbf{f} * \mathbf{g} = \underbrace{\begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix}}_{circulant\ matrix\ \mathbf{G}} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} =$$

$$= \mathbf{\Phi} \underbrace{\begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix}}_{diagonalized\ by\ Fourier\ basis} \qquad \mathbf{\Phi}^{\mathrm{T}}\mathbf{f} = \mathbf{\Phi} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \begin{bmatrix} \hat{f}_1 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \mathbf{\Phi} \begin{bmatrix} \hat{f}_1 \cdot \hat{g}_1 \\ \vdots \\ \hat{f}_n \cdot \hat{g}_n \end{bmatrix}$$

89

89

Convolution theorem

It turns out that all circulant matrices are diagonalized by the same basis $\mathbf{\Phi}_{\boldsymbol{\varepsilon}} = \{\Phi_{\mathbf{1}}, \dots, \Phi_{\boldsymbol{n}}\}$

$$\mathbf{G} = \begin{bmatrix} g_1 & g_2 & \cdots & \cdots & g_n \\ g_n & g_1 & g_2 & \cdots & g_{n-1} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ g_3 & g_4 & \cdots & g_1 & g_2 \\ g_2 & g_3 & \cdots & \cdots & g_1 \end{bmatrix} = \mathbf{\Phi}_{\boldsymbol{\varepsilon}} \begin{bmatrix} \hat{g}_1 & & \\ & \ddots & \\ & & \hat{g}_n \end{bmatrix} \mathbf{\Phi}_{\boldsymbol{\varepsilon}}^{\boldsymbol{T}}$$

90

90

Convolution theorem

This basis $\mathbf{\Phi}_{\boldsymbol{\varepsilon}}$ is very special, it is the discretized Fourier basis in the Euclidean domain:

$$\Phi_{\mathcal{E}} = \{\phi_1, \cdots, \phi_n\} \quad \text{with} \quad \phi_k = \begin{pmatrix} w_n^{0k} \\ w_n^{1k} \\ w_n^{2k} \\ \vdots \\ w_n^{(n-1)k} \end{pmatrix} \quad \text{and} \quad w_n^{jk} = e^{\frac{2\pi i}{n} jk}$$

91

91

Convolution theorem

The expression of $\mathbf{G}$ as $\mathbf{\Phi}_{\boldsymbol{\varepsilon}}\widehat{\mathbf{G}}\mathbf{\Phi}_{\boldsymbol{\varepsilon}}^{\boldsymbol{T}}$ will be our bridge towards non-Euclidean domains.

In fact, we know a generalization of the Fourier basis to graphs and manifolds, the eigenvectors $\mathbf{\Phi}$ of the Laplacian operator:

$$\mathbf{\Delta} = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^{\mathbf{T}}$$

where $\mathbf{\Lambda}$ is the diagonal matrix containing the eigenvalues of the Laplacian.

92

92

Convolution theorem

The idea on these non-Euclidean domains is to calculate the eigenvectors of the Laplacian in the first place, which constitutes the generalized Fourier basis $\mathbf{\Phi}$, and then define the convolution operator as:

$$\mathbf{W} = \Phi \begin{pmatrix} \hat{w}_1 & & \\ & \ddots & \\ & & \hat{w}_n \end{pmatrix} \Phi^\top$$

Where $\widehat{w}_i$ are learnable parameters.

Notice that in the Euclidean case this expression coincides with the standard convolution defined above, since the eigenvectors of the Laplacian in that case are the Euclidean Fourier basis. This is a desired property.

93

93

Spectral convolution

Generalized convolution of $f, g \in L^2(\mathcal{X}) \to \mathbb{R}$ can be defined by analogy

$$f * g = \underbrace{\sum_{k \geq 1} \underbrace{\langle f, \phi_k \rangle_{L^2(\mathcal{X})} \langle g, \phi_k \rangle_{L^2(\mathcal{X})}}_{\text{product in the Fourier domain}} \phi_k}_{\text{inverse Fourier transform}}$$

In matrix-vector notation

$$\mathbf{f} * \mathbf{g} = \mathbf{\Phi}\left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{g}\right) \circ \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{f}\right) = \underbrace{\mathbf{\Phi}\mathrm{diag}(\hat{g}_1, \dots, \hat{g}_n)\mathbf{\Phi}^{\mathrm{T}}}_{\mathbf{G}} \mathbf{f}$$

- Not shift-invariant! ($\mathbf{G}$ has no circulant structure)
- Filter coefficients depend on basis $\phi_1, \dots, \phi_n$

94

94

Spectral convolution

However, we have several drawbacks:

- The filters coefficients $\widehat{w}_i$ depend on the basis $\mathbf{\Phi}$. Learned filters do not generalize across domains; the addition of a single node in a graph or the small differences in a mesh after a change of pose fatally changes the basis. For instance, a convolutional filter with parameters $\widehat{w}_i$ tuned to spot edges changes completely behavior on a slightly different domain.

Domain	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{Y}$
Basis	$\mathbf{\Phi}$	$\mathbf{\Phi}$	$\mathbf{\Psi}$
Signal	$\mathbf{f}$	$\mathbf{\Phi}\widehat{\mathbf{W}}\mathbf{\Phi}^\top\mathbf{f}$	$\mathbf{\Psi}\widehat{\mathbf{W}}\mathbf{\Psi}^\top\mathbf{f}$

95

95

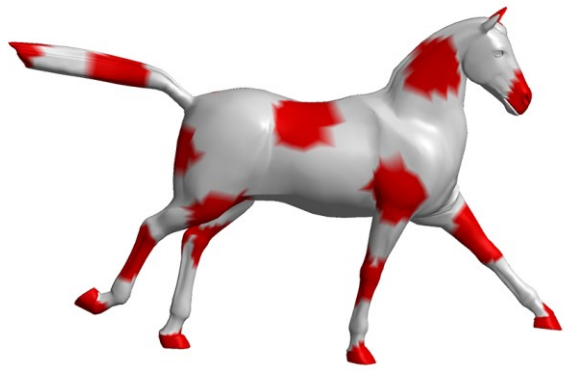
Spectral convolution

- The number of trainable parameters per filter depends on n, the size of the domain. We want a convolutional filter with a fixed number of parameters like in the Euclidean case.
- Since the trainable parameters are not properly constrained, there is a high chance that the learned filter is not localized in space.

96

96

Basis dependence



Function $\mathbf{x}$

97

97

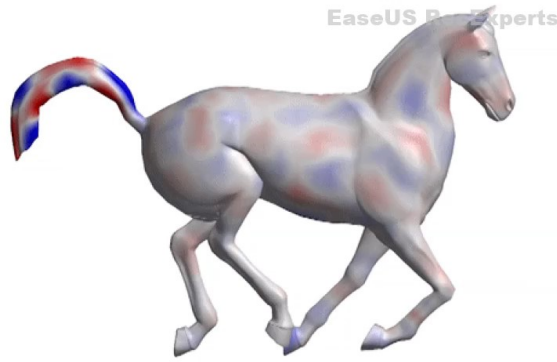
Basis dependence

`Edge detecting' spectral filter $\mathbf{\Phi}\hat{\mathbf{Y}}\mathbf{\Phi}^{\mathrm{T}}\mathbf{x}$

98

98

Basis dependence

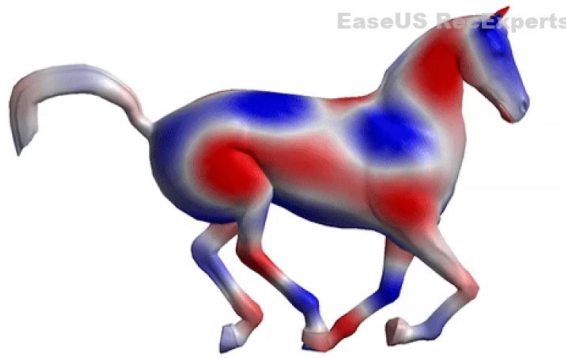


Same spectral filter, different basis $\mathbf{\Psi}\hat{\mathbf{Y}}\mathbf{\Psi}^{\mathrm{T}}\mathbf{x}$

99

99

Basis dependence



Basis function with index i across different shapes

100

100

Locality and smoothness

To address the problems listed previously, we put some constraints on the matrix $\widehat{\mathbf{W}}$, parametrizing it in a different way.

Instead of having a degree of liberty per element of the diagonal (n learnable parameters), we substitute $\widehat{\mathbf{W}}$ with the fixed eigenvalues of the Laplacian $\mathbf{\Lambda}$ altered by a single parametrized transformation $\tau_\alpha(\lambda)$, which depends on a fixed number of learnable parameters α.

101

101

Locality and smoothness

In the Euclidean setting (by Parseval's identity), the following holds:

$$\int_{-\infty}^{+\infty} |x|^{2k} |f(x)|^2 dx = \int_{-\infty}^{+\infty} \left| \frac{\partial^k \hat{f}(\omega)}{\partial \omega^k} \right|^2 dx$$

Localization in space = smoothness in frequency domain

Parametrize the filter using a smooth spectral transfer function $\tau(\lambda)$.
Application of the filter

$$\tau(\mathbf{\Delta})\mathbf{f} = \mathbf{\Phi}\tau(\mathbf{\Lambda})\mathbf{\Phi}^{\mathrm{T}}\mathbf{f} = \mathbf{\Phi}\begin{bmatrix} \tau(\lambda_1) & & \\ & \ddots & \\ & & \tau(\lambda_n) \end{bmatrix}\mathbf{\Phi}^{\mathrm{T}}\mathbf{f}$$

Hena et al, “Deep Convolutional Networks on Graph-Structured Data", 2015

102

102

Locality and smoothness

Parametrize the filter using a smooth spectral transfer function $\tau(\lambda)$.

Application of the parametric filter with learnable parameters α

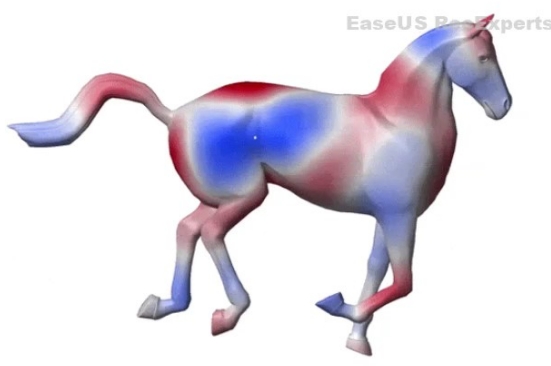
$$\tau_\alpha(\mathbf{\Delta})\mathbf{f} = \mathbf{\Phi}\begin{bmatrix}\tau_\alpha(\lambda_1) & & \\ & \ddots & \\ & & \tau_\alpha(\lambda_n)\end{bmatrix}\mathbf{\Phi}^{\mathrm{T}}\mathbf{f}$$

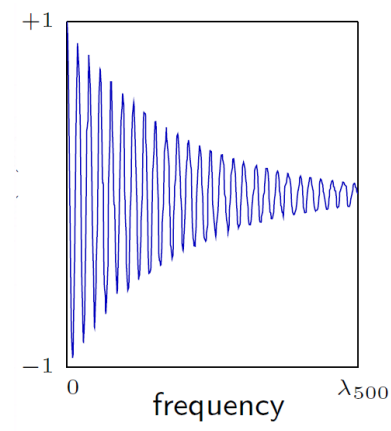
Hena et al, “Deep Convolutional Networks on Graph-Structured Data", 2015

103

103

Locality and smoothness



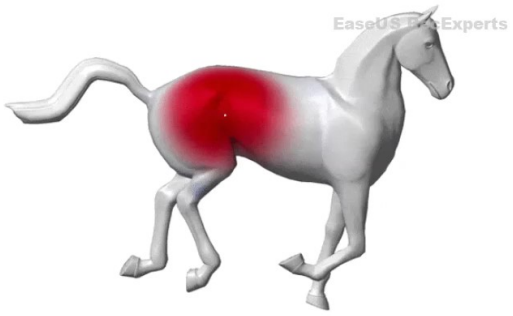


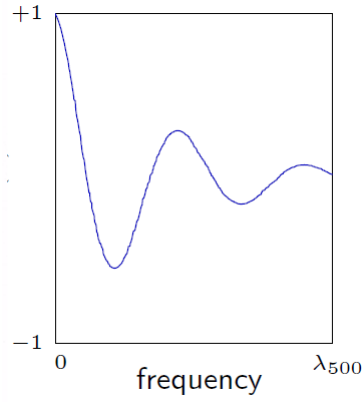
Non-smooth spectral filter (delocalized in space)

104

104

Locality and smoothness



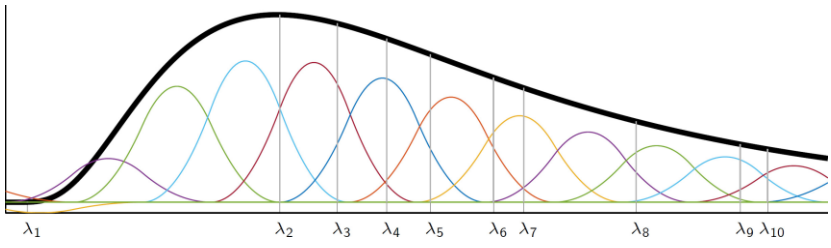


Smooth spectral filter (localized in space)

105

105

Spectral graph CNN with smooth spectral filters



Consider a linear combination of smooth kernel functions $\beta_1(\lambda), \dots, \beta_r(\lambda)$

$$\tau_\alpha(\lambda) = \sum_{j=1}^{r} \alpha_j \beta_j(\lambda)$$

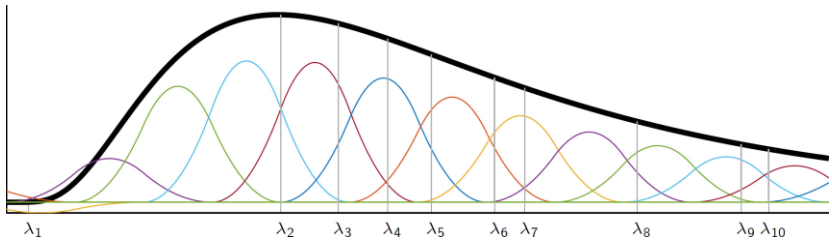
where $\alpha = (\alpha_1, \dots, \alpha_r)^{\mathrm{T}}$ is the vector of filter parameters.

Hena et al, “Deep Convolutional Networks on Graph-Structured Data", 2015

106

106

Spectral graph CNN with smooth spectral filters



Consider a linear combination of smooth kernel functions $\beta_1(\lambda), \dots, \beta_r(\lambda)$

$$\tau_\alpha(\lambda_k) = \sum_{j=1}^{r} \alpha_j \beta_j(\lambda_k) = (\mathbf{B}\alpha)_k, \qquad \mathbf{W} = \mathrm{diag}(\mathbf{B}\alpha)$$

where $\alpha = (\alpha_1, \dots, \alpha_r)^{\mathrm{T}}$ is the vector of filter parameters.

$\mathcal{O}(1)$ parameters per layer.

Hena et al, "Deep Convolutional Networks on Graph-Structured Data", 2015

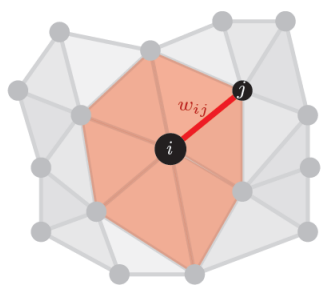
107

107

Spectral convolution on meshes

- Laplacian operator Δ acting locally on the neighborhood of i:

$$(\mathbf{\Delta x})_i = \sum_j w_{ij}(\mathbf{x}_j - \mathbf{x}_i)$$
$$= \textit{neighborhood avg} - \textit{value at } i$$



- Eigenvectors of the Laplacian $\mathbf{\Delta} = \mathbf{\Phi\Lambda\Phi}^{\mathrm{T}}$ are a generalization of the Fourier transform: $\hat{\mathbf{x}} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{x}$
- Spectral convolution

$$\mathbf{x} * \mathbf{y} = \mathbf{\Phi} \underbrace{\begin{pmatrix} \hat{y}_1 & & \\ & & \\ & & \hat{y}_n \end{pmatrix}}_{\hat{\mathbf{Y}}} \hat{\mathbf{x}}$$

Bruna et al, "Spectral Networks and Locally Connected Networks on Graphs", 2014

108

108

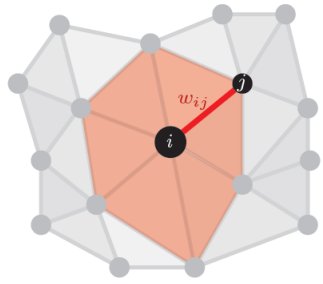
Spectral convolution on meshes

- Laplacian operator Δ acting locally on the neighborhood of i:

$$(\mathbf{\Delta x})_i = \sum_j w_{ij}(\mathbf{x}_j - \mathbf{x}_i)$$
$$= \textit{neighborhood avg} - \textit{value at } i$$

- Eigenvectors of the Laplacian $\mathbf{\Delta} = \mathbf{\Phi\Lambda\Phi}^{\mathrm{T}}$ are a generalization of the Fourier transform: $\hat{\mathbf{x}} = \mathbf{\Phi}^{\mathrm{T}}\mathbf{x}$
- Spectral convolution defined as a filter applied on the Laplacian:

$$\mathbf{X}' = \mathbf{\Phi}\tau(\mathbf{\Lambda})\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}$$



Bruna et al, “Spectral Networks and Locally Connected Networks on Graphs", 2014

109

109

SPECTRUM FREE

110

Adjacency matrices: Vertex-to-vertex

Graph connectivity can be encoded in adjacency matrices.

Let $|V| = n$, $|E| = e$, $|F| = m$ for a mesh $\mathcal{M} = (V, E, F)$

The vertex-to-vertex adjacency is defined as the $n \times n$ binary matrix:

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix}$$

where $a_{ij} = 1$ if vertex v_i is connected to v_j (that is, $e_{ij} \in E$)

- The diagonal is always 0
- $\mathbf{A}$ is symmetric
- Each row and column has at least one 1 (that is, $\sum_{ij} a_{ij} = e$)

111

111

Adjacency matrices: Powers

The k-th power of $\mathbf{A}$ corresponds to composing $\mathbf{A}$ with itself $k \geq 1$ times.

For example, for $k = 2$:

$$\mathbf{A}^2 = \mathbf{A}\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix}$$

The result is a $n \times n$ matrix encoding 2nd order adjacency.

For $k > 2$:

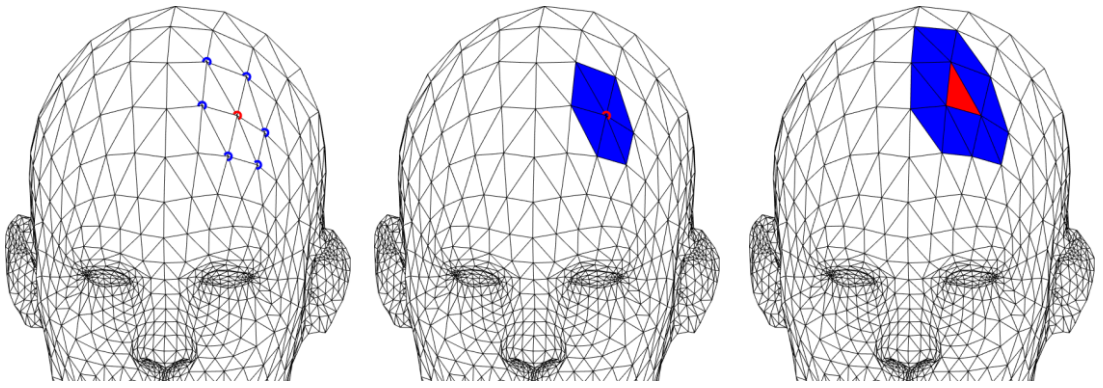
$$\mathbf{A}^k = \mathbf{A} \cdots \mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix} \cdots \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix}$$

The result is a $n \times n$ matrix encoding k-th order adjacency.

112

112

Examples: Powers

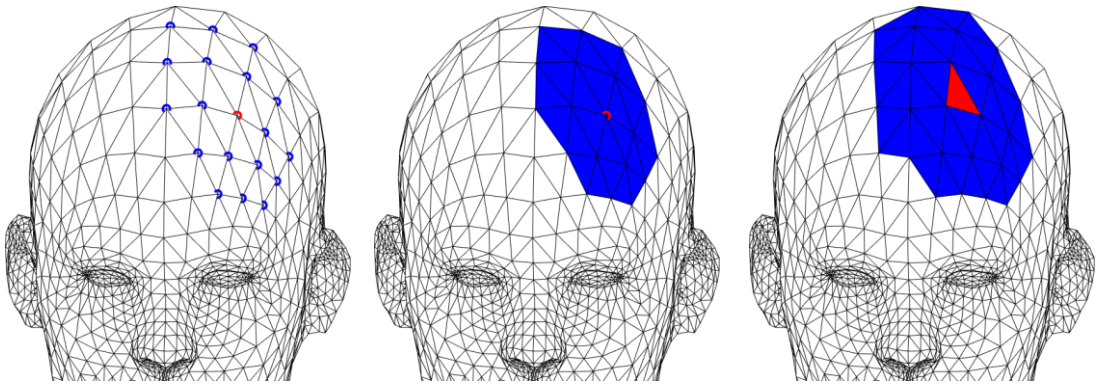


vertex-to-vertex $k = 1$ | vertex-to-triangle $k = 1$ | triangle-to-triangle $k = 1$

113

113

Examples: Powers

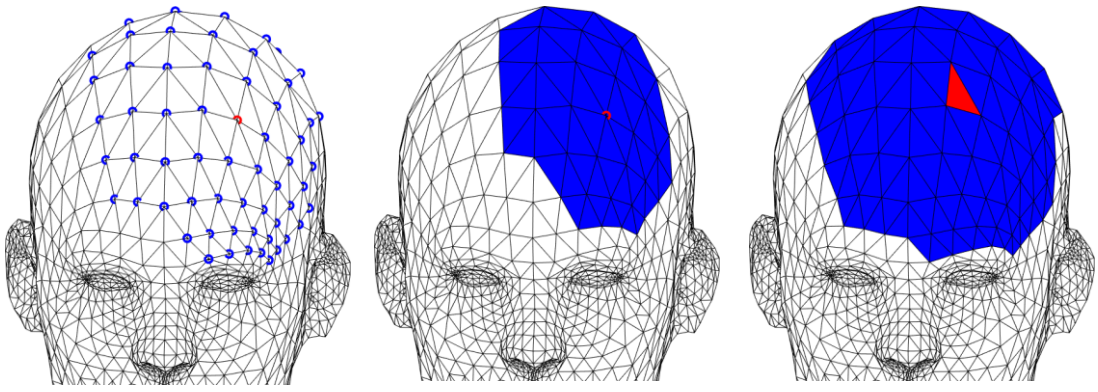


vertex-to-vertex $k = 2$ | vertex-to-triangle $k = 2$ | triangle-to-triangle $k = 2$

114

114

Examples: Powers



vertex-to-vertex $k = 4$ vertex-to-triangle $k = 3$ triangle-to-triangle $k = 3$

115

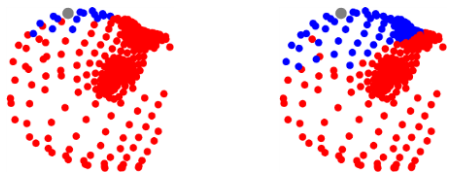
115

Adjacency matrices: Point clouds

Adjacency is a general notion that can be extended to point clouds.

For example, use Euclidean distance within a threshold τ:

$$a_{ij} = \begin{cases} 1 & if \ \|\mathbf{v}_i - \mathbf{v}_j\|_2 \leq \tau \\ 0 & otherwise \end{cases}$$



$k = 1$ $k = 2$

Similarly to before, $\mathbf{A}^k$ encodes k-th order adjacency.

116

116

Adjacency matrices as operators

We can see adjacency matrices as operators when applied to functions.
For example, $\mathbf{g} = \mathbf{Af}$ is written as:

$$\mathbf{g} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 1 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 1 & 0 & 1 & \cdots & 0 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

$\mathbf{g} = \mathbf{Af}$ yields a vertex-based function $\mathbf{g}$ defined as:

$$g(v_i) = \sum_{e_{ij} \in E} f(v_j)$$

117

117

Local operators

On this observation, one can construct new operators such as $\mathbf{I} - \mathbf{A}$:

$$g(v_i) = f(v_i) - \sum_{e_{ij} \in E} f(v_j)$$

Or such as:

$$g(v_i) = f(v_i) - \frac{1}{d_i} \sum_{j:(i,j) \in E} f(v_j)$$

118

118

Graph Laplacian

$$g(v_i) = f(v_i) - \frac{1}{d_i} \sum_{j:(i,j)\in E} f(v_j)$$

In matrix notation, we define the $n \times n$ matrix $\mathbf{L}$ as:

$$L_{ij} = \begin{cases} 1 & if\ i = j \\ -\frac{1}{d_i} & e_{ij} \in E \\ 0 & otherwise \end{cases}$$

also known as the graph Laplacian of G.

Variants with different properties exist (e.g., normalized Laplacian, random walk Laplacian, etc.).

119

119

Learnable polynomial filters

Consider again the spectral filter:

$$\tau_\theta(\Delta)\mathbf{f} = \mathbf{\Phi}\tau_\theta(\Lambda)\mathbf{\Phi}^{\mathrm{T}}\mathbf{f}$$

with the polynomial parametrization:

$$\tau_\theta(\Delta) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

This corresponds to just taking powers of the Laplacian:

$$\tau_\theta(\Delta)\mathbf{f} = \left(\sum_{k=0}^{K-1} \theta_k \Delta^k\right)\mathbf{f}$$

Therefore, it is a spectrum-free convolution.

120

120

Learnable polynomial filters

$$\tau_\theta(\Delta)\mathbf{f} = \left(\sum_{k=0}^{K-1} \theta_k \, \Delta^k\right)\mathbf{f}$$

Different convolutions are obtained with different polynomials (e.g., Chebyshev polynomials for the ChebNet model).

The notion of convolution is now replaced with the application of a local operator such as the Laplacian.

This idea will allow us to work with point clouds.

Defferrard et al, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering", NIPS 2016
Levie et al, "CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters", IEEE TSP 2018

121

121

Suggested reading

Bronstein et al, "Geometric deep learning: going beyond Euclidean data", 2016
https://arxiv.org/abs/1611.08097

Bronstein et al, "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges", 2021
https://arxiv.org/abs/2104.13478

122

122