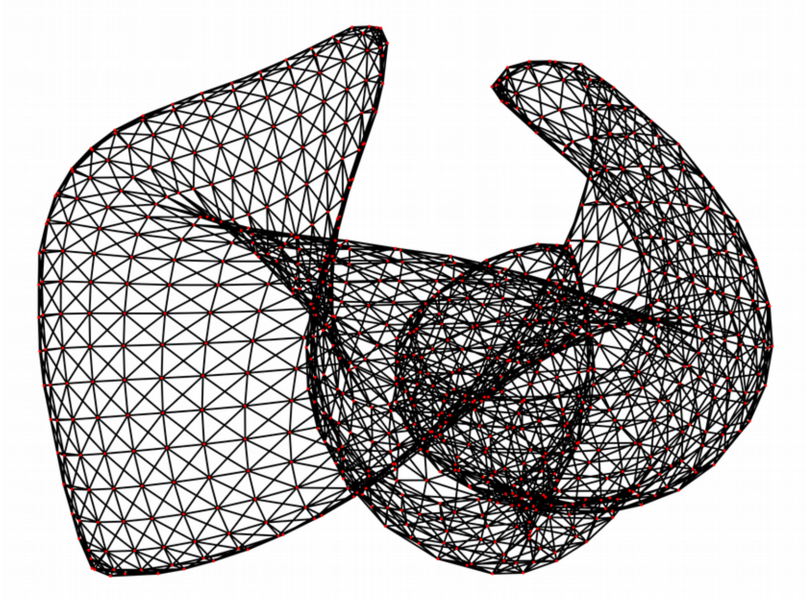
Graph Neural Networks

- Convolutions on non-euclidean domains
- Graph and spectral graph basics
- Graph Convolutional Neural Networks
  - Spatial domain
  - Spectral domain



Jonas Glombitza

Big Data Science in Astroparticle Research, Aachen, 17-19 February 2020

Time Schedule

Introduction: Graphs and Graph Convolutions

- Basics of graphs and graph theory
- Graph Convolution Networks (GCNs)
- *Practice 1: Semi-supervised node classification using GCNs (karate club)*

Convolutions in Spectral Domain

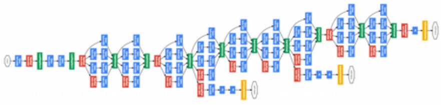
- Spectral graph theory
- Chebychev Convolutions (ChebNets)
- *Practice 2: MNIST on graphs unsing ChebNets*

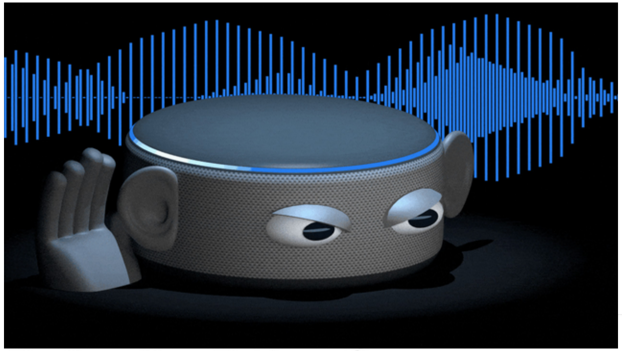
Convolutional in Spatial Domain

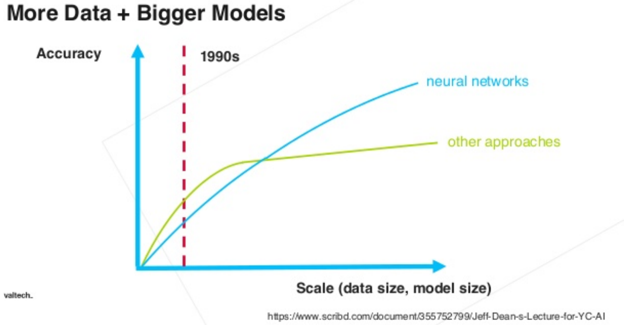
- Edge-Convolutions in Dynamic Graph Convolutional Neural Networks (DGCNNs)
- *Practice 3: Cosmic-ray classification using DGCNNs*

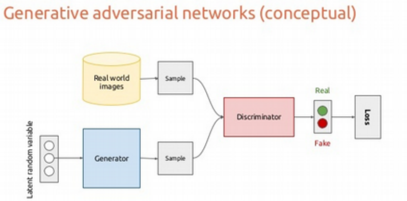
Deep Learning

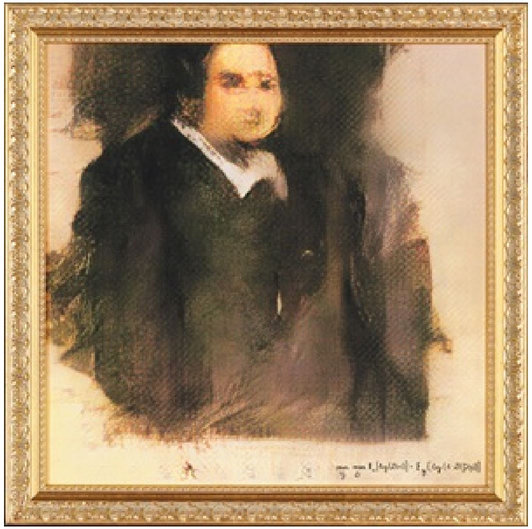
- Outstanding results
  - Speech recognition
  - Image recognition → Convolutions





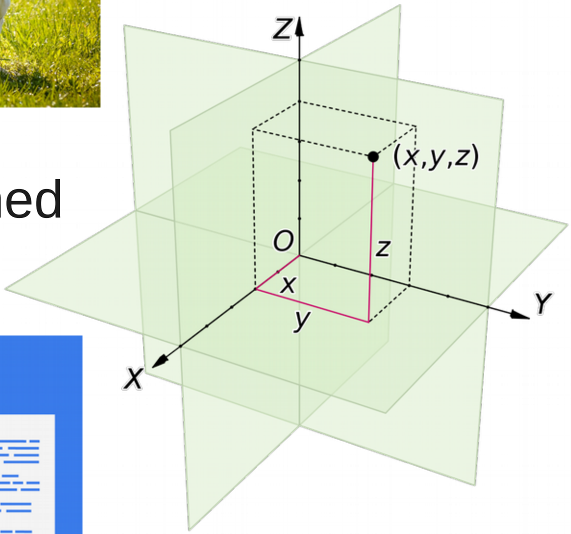


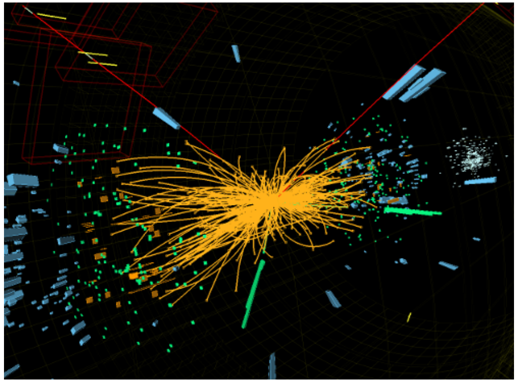


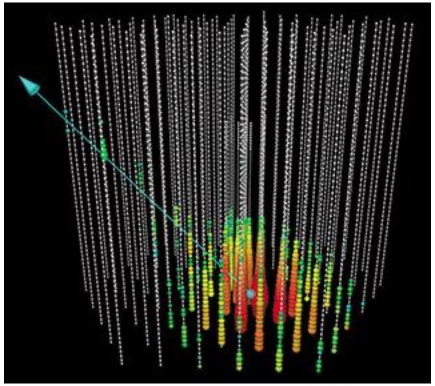


Convolutions and Datasets

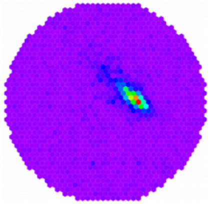
- Works in well defined euclidean space

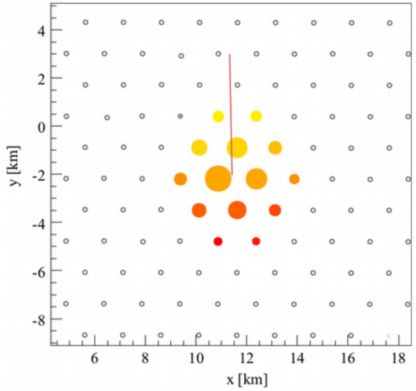


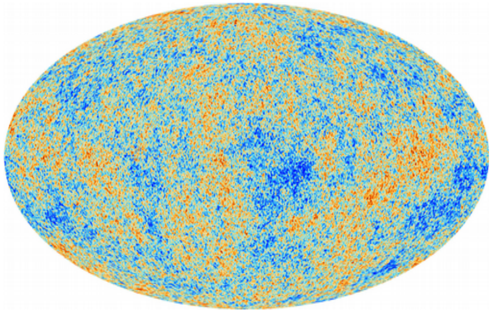




- physics data often feature different geometries

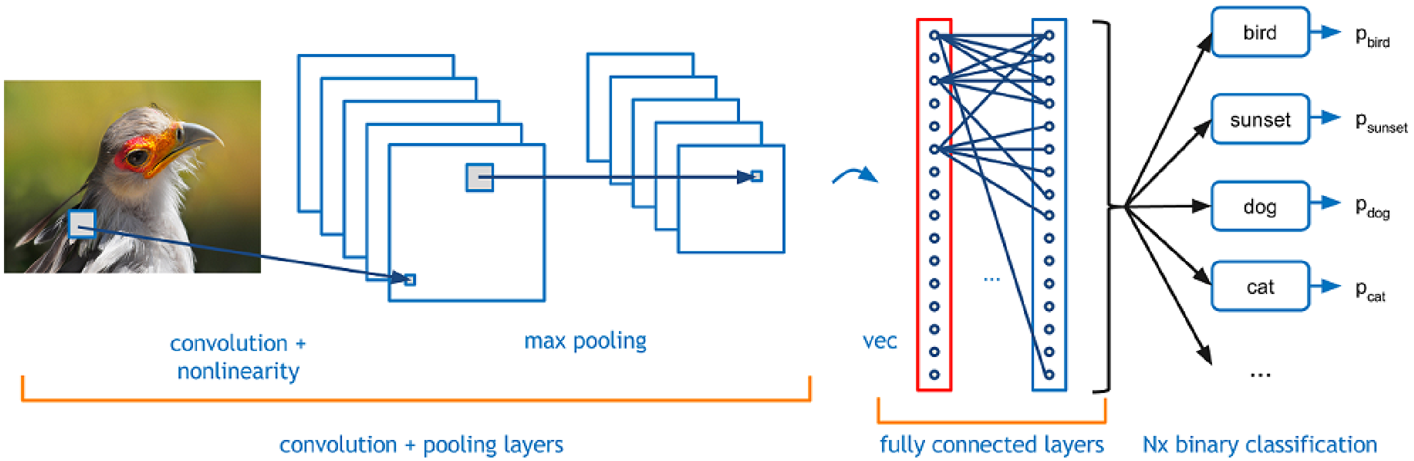




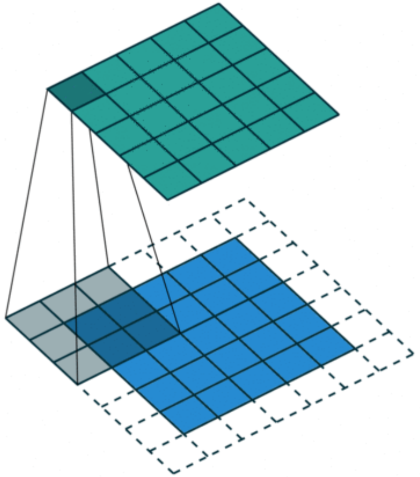


Convolutions

- Translational invariance
- Scale separation (hierarchy learning)
- Deformation stability (filters are localized in space)
- Parameters are independent from input size



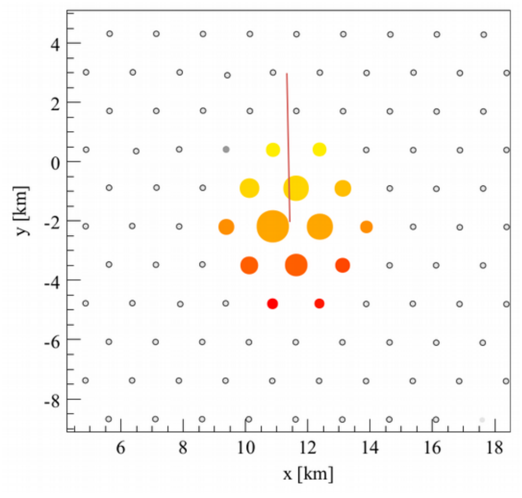
Adit Deshpande - https://adeshpande3.github.io/adeshpande3.github.io/

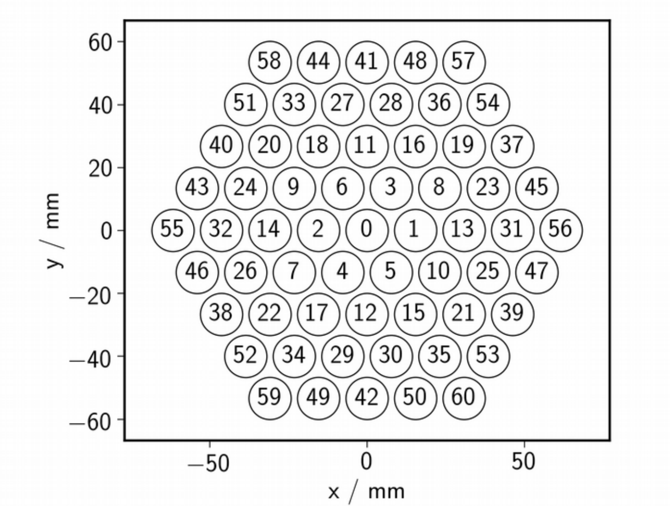


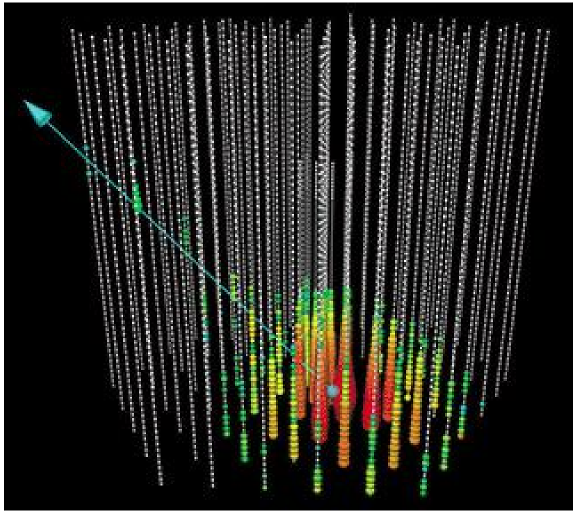
Paul-Louis Pröve,
Towards Data Science

Hexagonal Grids

What do these experiments have in common?

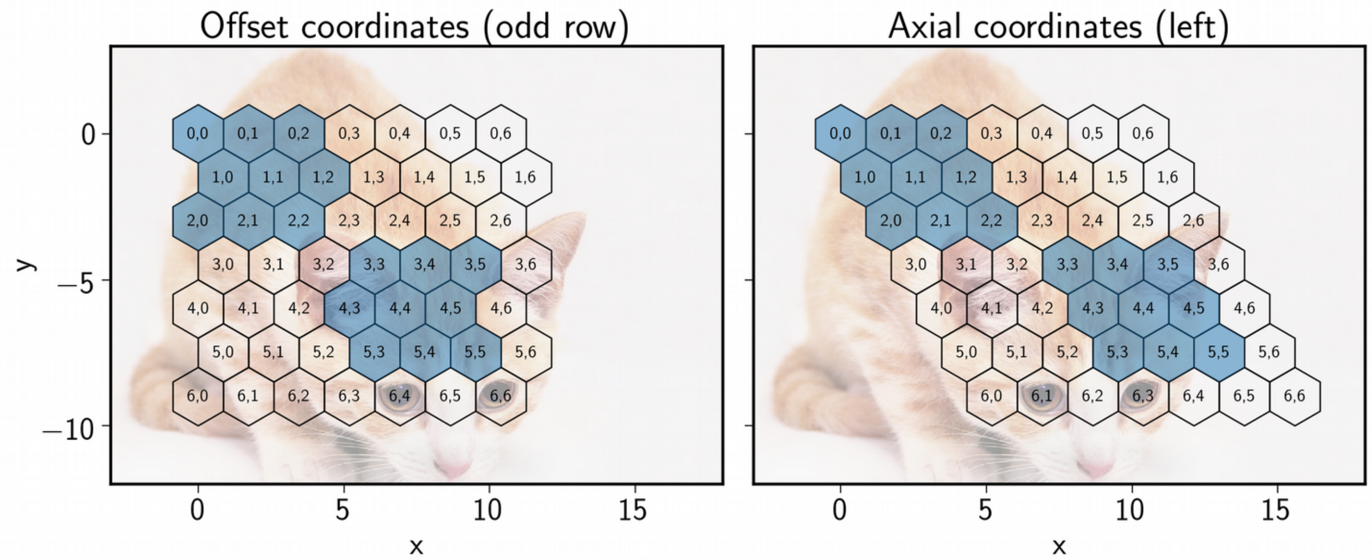






- Most astroparticle detectors feature hexagonal grids
- Need change of coordinate system for matrix representation of data

Hexagonal Grids - Convolutions



Offset coordinates: neighbor relations in convolution filter changes between rows

✗ No translational invariance unless using a stride of 2

Axial coordinates: need to pad edges → need more storage

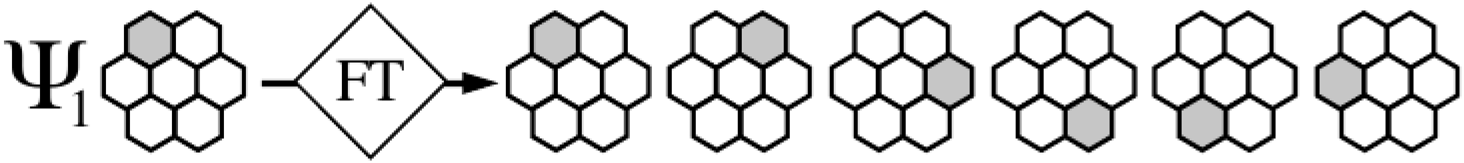
✔ translational invariance

Hexagonal Convolution

- Expand the concept of invariance to rotation symmetries → **group convolutions**
  - Use complete symmetry of hexagonal data (translation + rotation of filters)
    - Hexagonal convolutions

- Weights of the filter shared over translation and over rotations
- Each filter creates 6 orientation channels

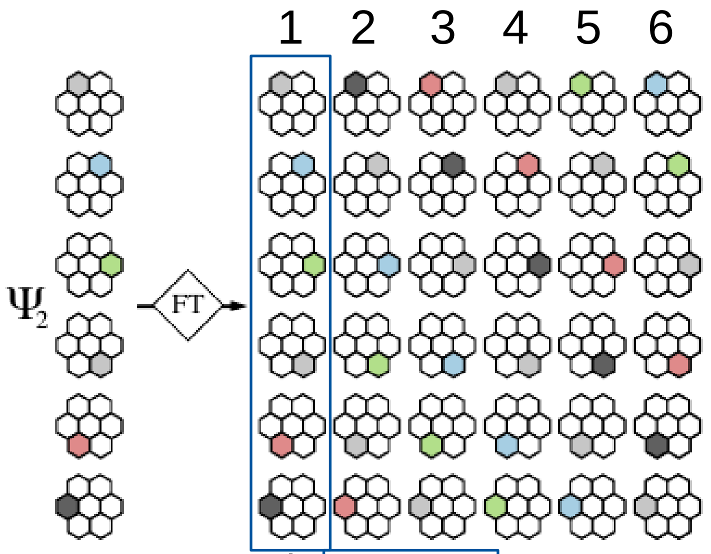
- Decrease number of parameters
- Increase training time

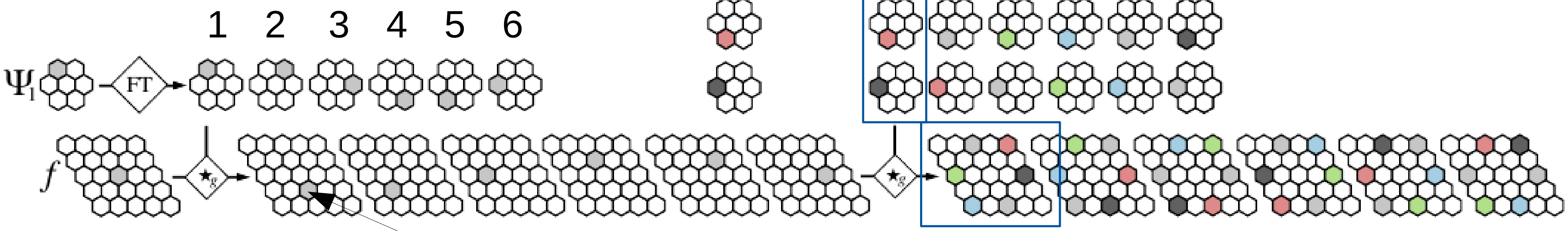


HexaConv

- Extend convolutions to invariance on hexagonal grids (p6 group)
- Transformation makes filter invariant to rotations
- Orientation channel cycling

- Example:
- Initial Convolution: $\mathbb{Z}^2 \rightarrow p6$
- Convolution from: $p6 \rightarrow p6$





Initial convolution

Activated pixel when using filter 1

E. Hoogeboom, J. Peters, T. Cohen, M. Welling: ArXiv/1803.02108

Code Example

- Filter-size 3: → 7 adaptive parameter
- Filter-size 5: → 19 adaptive parameters
- Need data in axial coordinates
- Beta implementation of keras / tf layers by Lukas Geiger

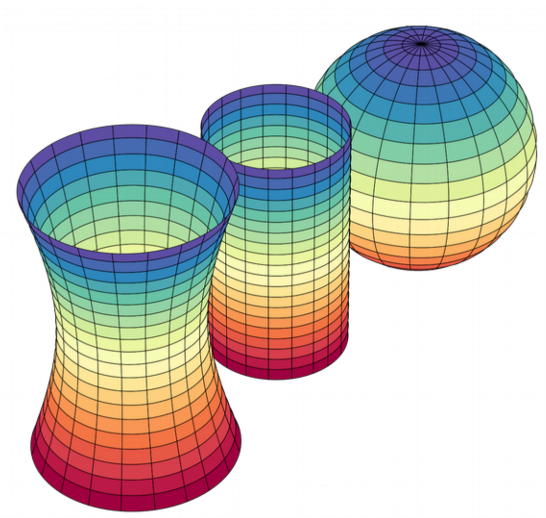
```python
import tensorflow as tf
from tensorflow import keras
from groupy.gconv.gconv_tensorflow.keras.layers import P6ConvZ2Axial, P6ConvP6Axial
layers = keras.layers

input1 = layers.Input(shape=(9, 9, 2))
kwargs = dict(activation='relu', kernel_initializer='he_normal')
# initial convolution
z = P6ConvZ2Axial(3, 3, padding='same', activation='relu')(input1)
z = P6ConvP6Axial(6, 3, padding='same', **kwargs)(z)
z = layers.Flatten()(z)
```

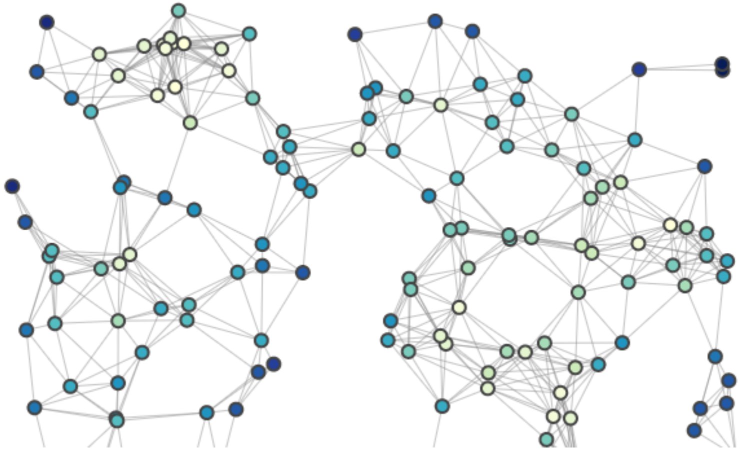
Check GitHub: https://github.com/ehoogeboom/hexaconv

Generalization to Non-Euclidean Domains

- Defining convolutions, challenging on non-euclidean domains
  - Deformation of filters, changing neighbor relations
  - Non-isometric connections on graphs



- **Manifolds**



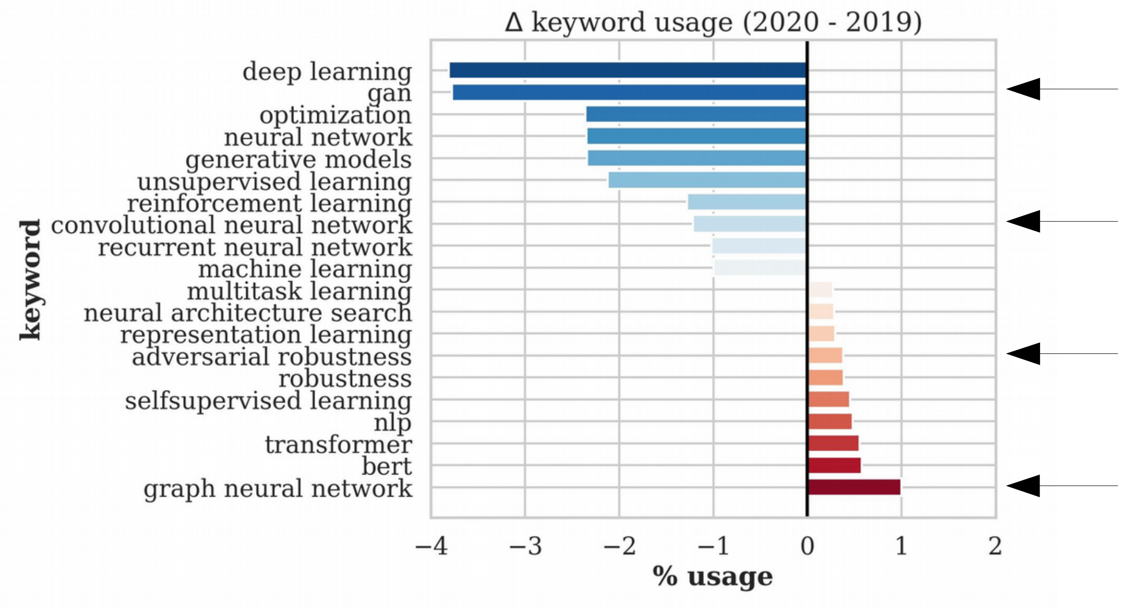
- **Graphs**

How can we generalize Convolutions?

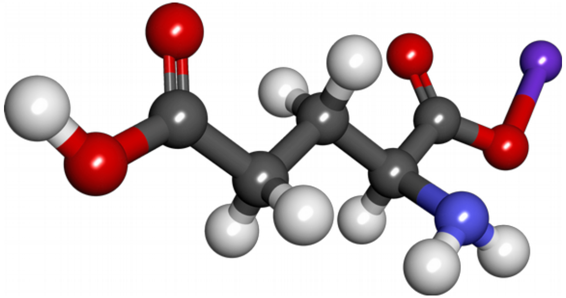
Deep Learning on Graphs

- Introduction to graphs
- Graph basics
- Spectral graph theory

ICLR2020 submissions - growth

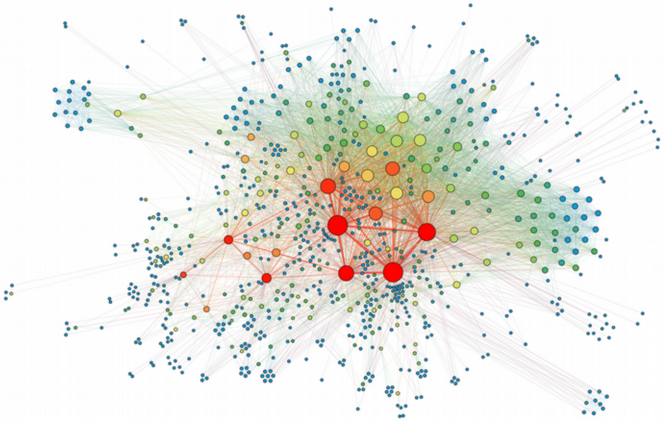


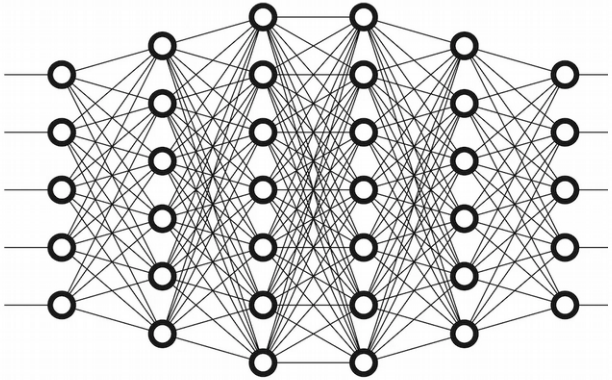
Types of Graphs



heterogeneous graph

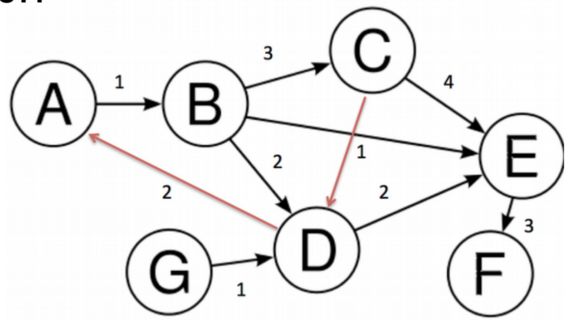
graphs with edge information





bipartite graph

undirected graph



directed graph

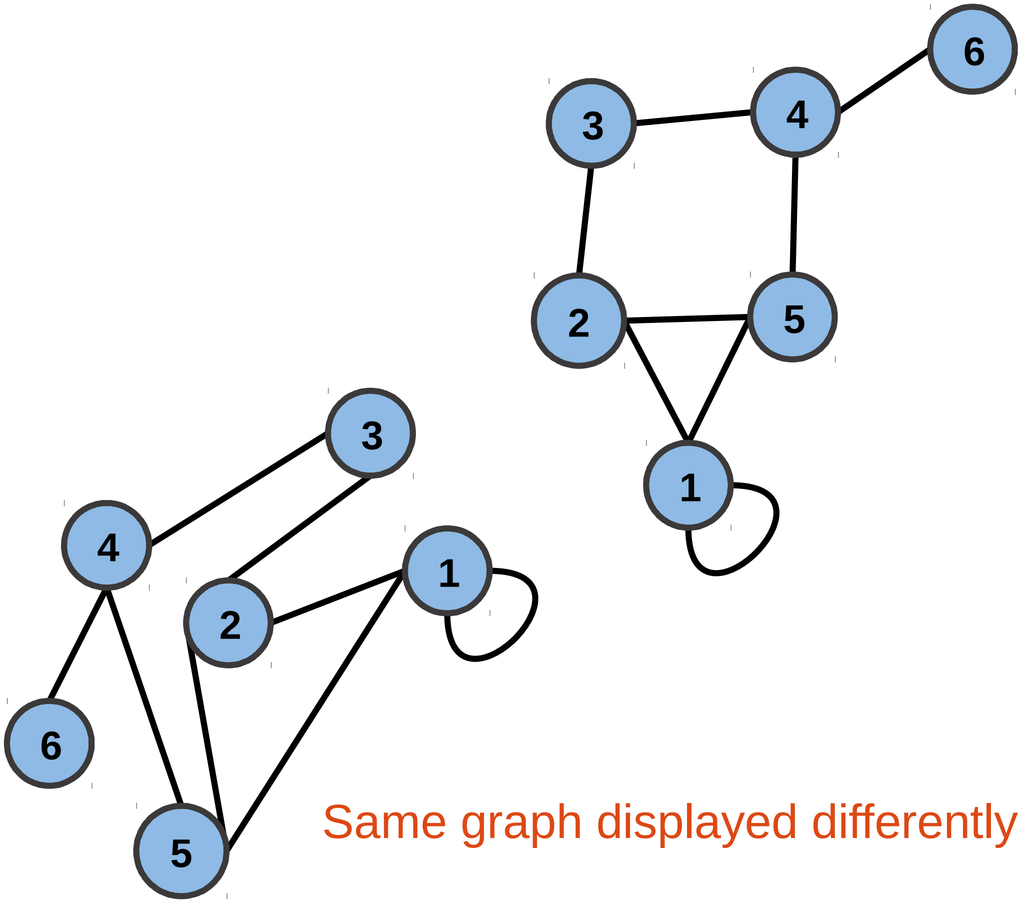
What is a Graph

$$\mathcal{G} = (\mathcal{V}, \mathcal{E})$$

- Graph is ordered pair
  - of nodes $\mathcal{V}$
  - and edges $\mathcal{E}$
- ➢ mainly defined by neighborhood

- Nodes have no order
  - ➢ **Permutational invariance**

- Challenging to visualize!

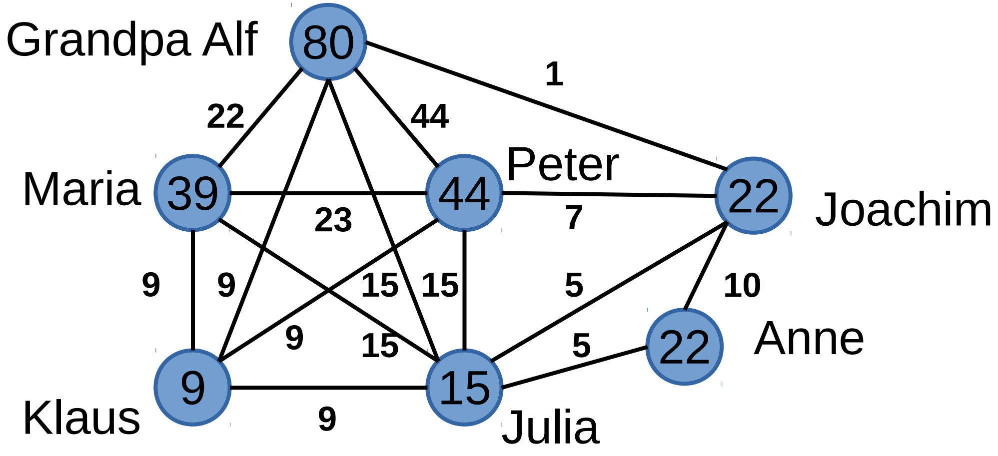


Same graph displayed differently

Example Graph

Social network

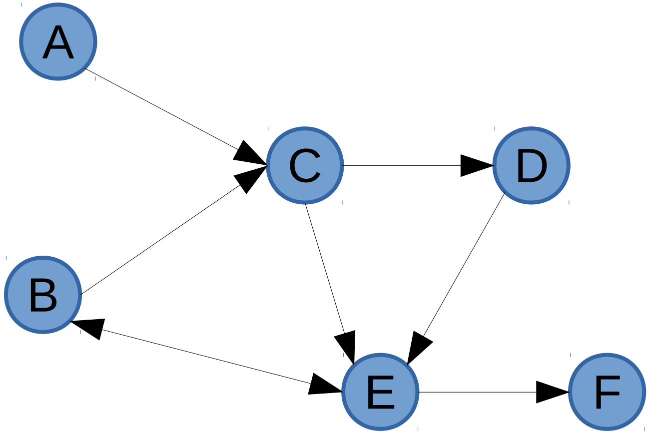
Bidirectional graph,
Including edge information



node = age of person
edge = age of relationship

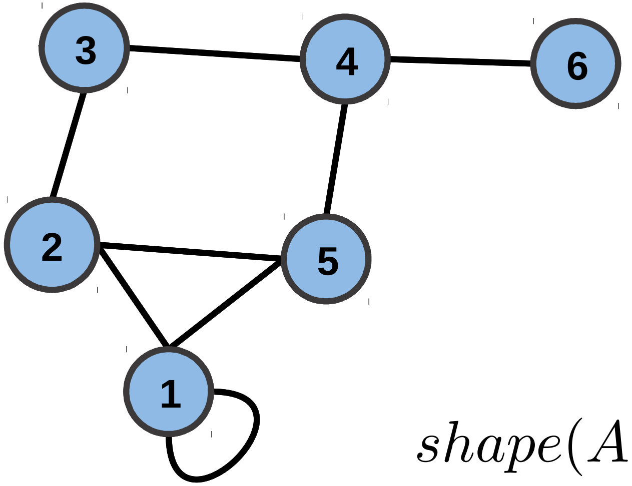
Production Chain

Directed graph



Adjacency Matrix

- Matrix to represent structure of graph
- Elements indicate edges of graph
- Symmetric for undirected graphs
- In general sparse



$$shape(A) = N \times N$$

$$A = \begin{array}{cc} \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} & \\ \begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} & \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} \end{array}$$

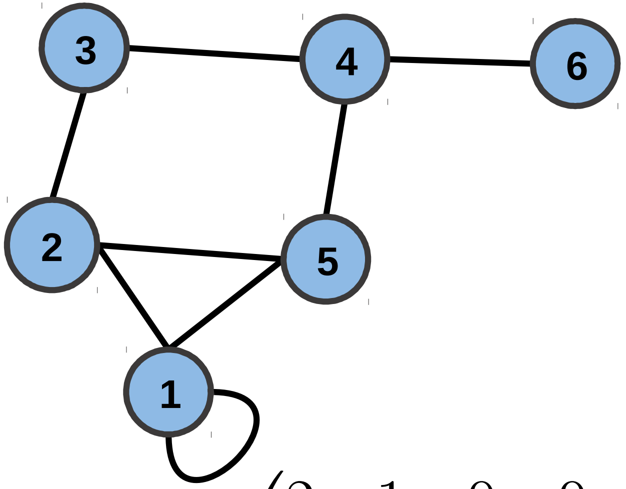
- Used to propagate signals on the graph

$$A \cdot f = \begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

signal

Degree Matrix

- Elements count number of times edges terminate at each node
- Used used to normalize adjacency A
- $shape(D) = N \times N$



$$D = \begin{array}{cc} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} & \\ \begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \end{array}$$

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Laplacian Matrix

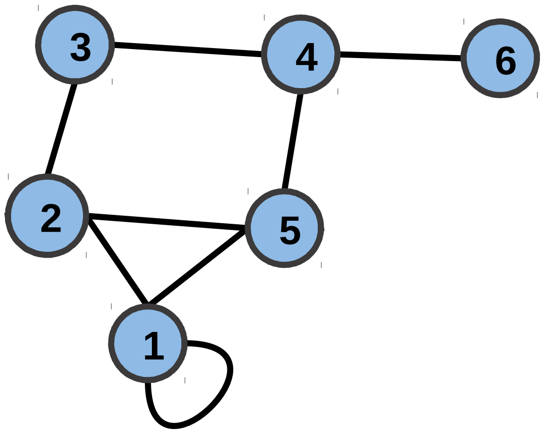
- Laplacian matrix L = normalized adjacency matrix A
  - $L = D - A$

- Difference between f and its local average
- Core operator in spectral graph theory

- Symmetric normalized Laplacian:
  - Eigenvalues do not depend on degree of nodes

$$L^{\text{sym}} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$$

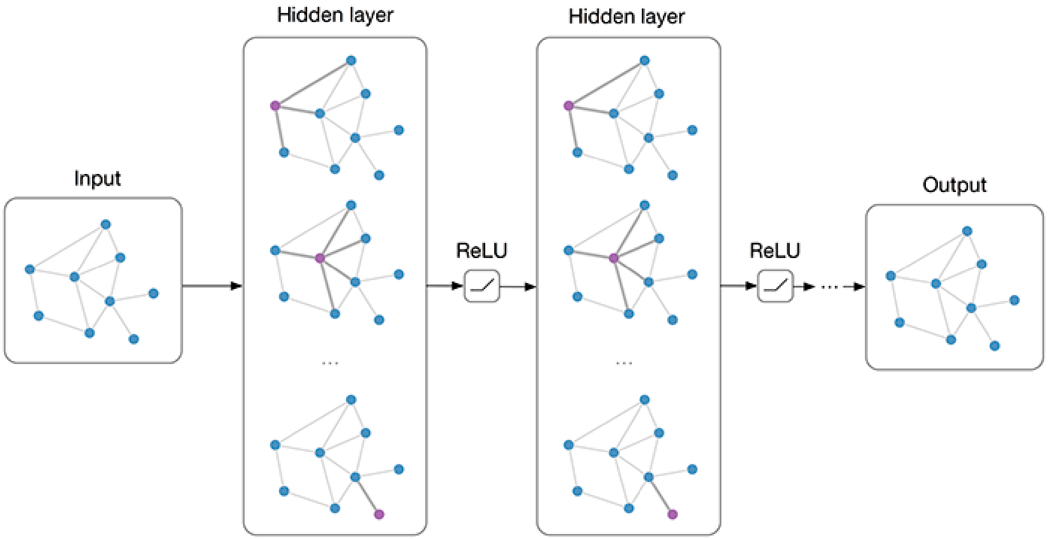
- Discrete version of Laplace operator



f = function acting on the graph

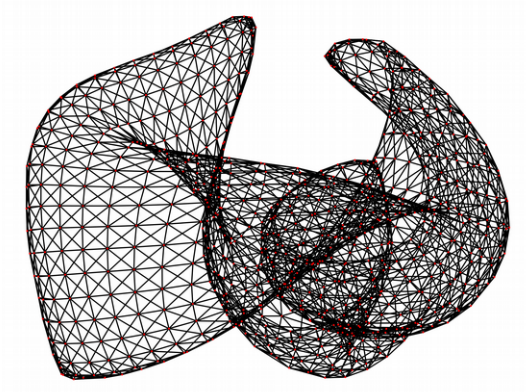
Graph Convolutional Networks

- Propagation rule for GCN
- Connection to euclidean Convolutions
- Semi-supervised classification



Thomas Kipf, Max Welling
arXiv:1609.02907

Natural Images vs. Graphs



- Collection of pixels (node)
  - Node (pixel) holds feature vector
  - Dense (rarely sparse)
  - Discrete, regular (symmetric)

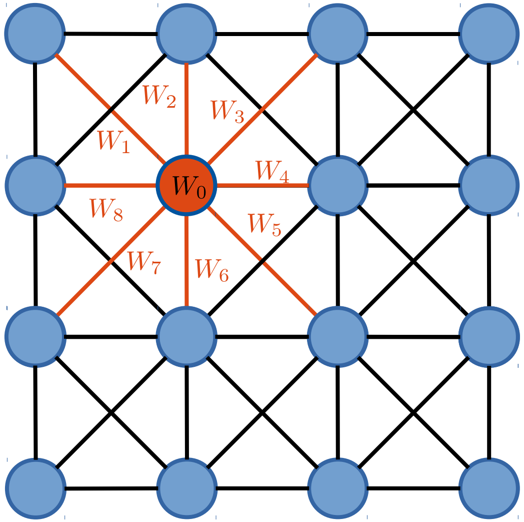
- Images feature euclidean space

- Collection of nodes and edges
  - Node + edge holds feature vector
  - Can be dense or sparse
  - Continuous non-symmetric positions

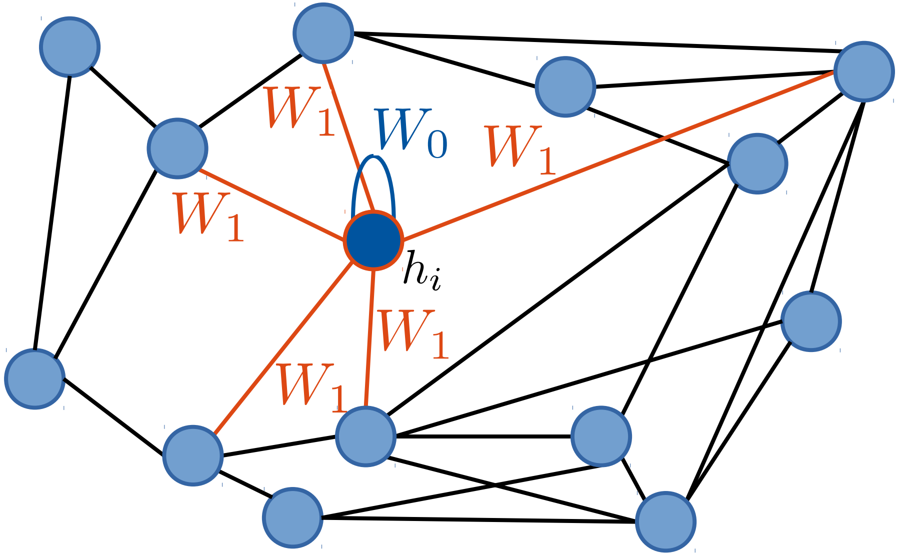
- Graphs can feature “arbitrary” domains

Graph Convolutional Networks

2D Convolution on regular grid



Convolution on Graph

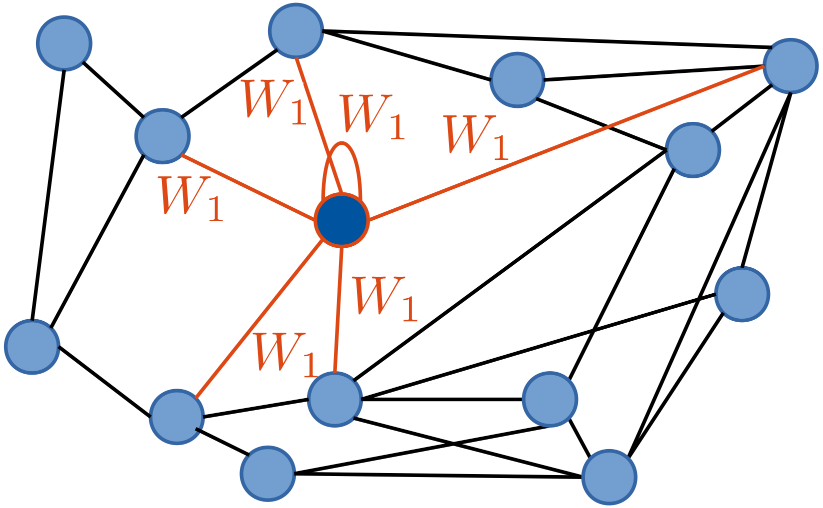


- Feature-map-wise weight-sharing!
- Node-wise weight-sharing!
- Propagation rule for GCN:

$$h_i^{(l+1)} = \sigma(h_i^{(l)} W_0^{(l)} + \underbrace{\sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}}}_{\text{Average over neighbors}} h_j^{(l)} W_1^{(l)})$$

Graph Convolutional Networks

- In general more easy $W_0 = W_1$
- Self coupling: same weight as neighbors
  - Very simple $\rightarrow$ works surprisingly good
- Node-wise weight-sharing!



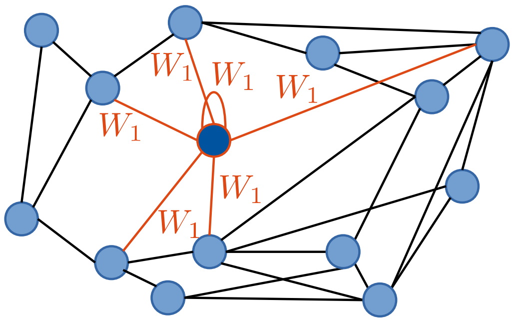
$$h_i^{(l+1)} = \sigma(h_i^{(l)} W_1^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} h_j^{(l)} W_1^{(l)})$$

Mathematical Formulation

- Input $H^{(0)} = X$
  - $shape(X) = N \times D$
- Weight signal with neighborhood using adjacency matrix $A,\ shape(A) = N \times N$
  - $H = f(X, A) \sim AH$

- Apply transformation using weight matrix $W,\ shape(W) = D \times F$
  - $H = \sigma(AXW^{(l)})$

- As A do not include self loops, we have to add them via
  - $\hat{A} = I + A$



Normalization

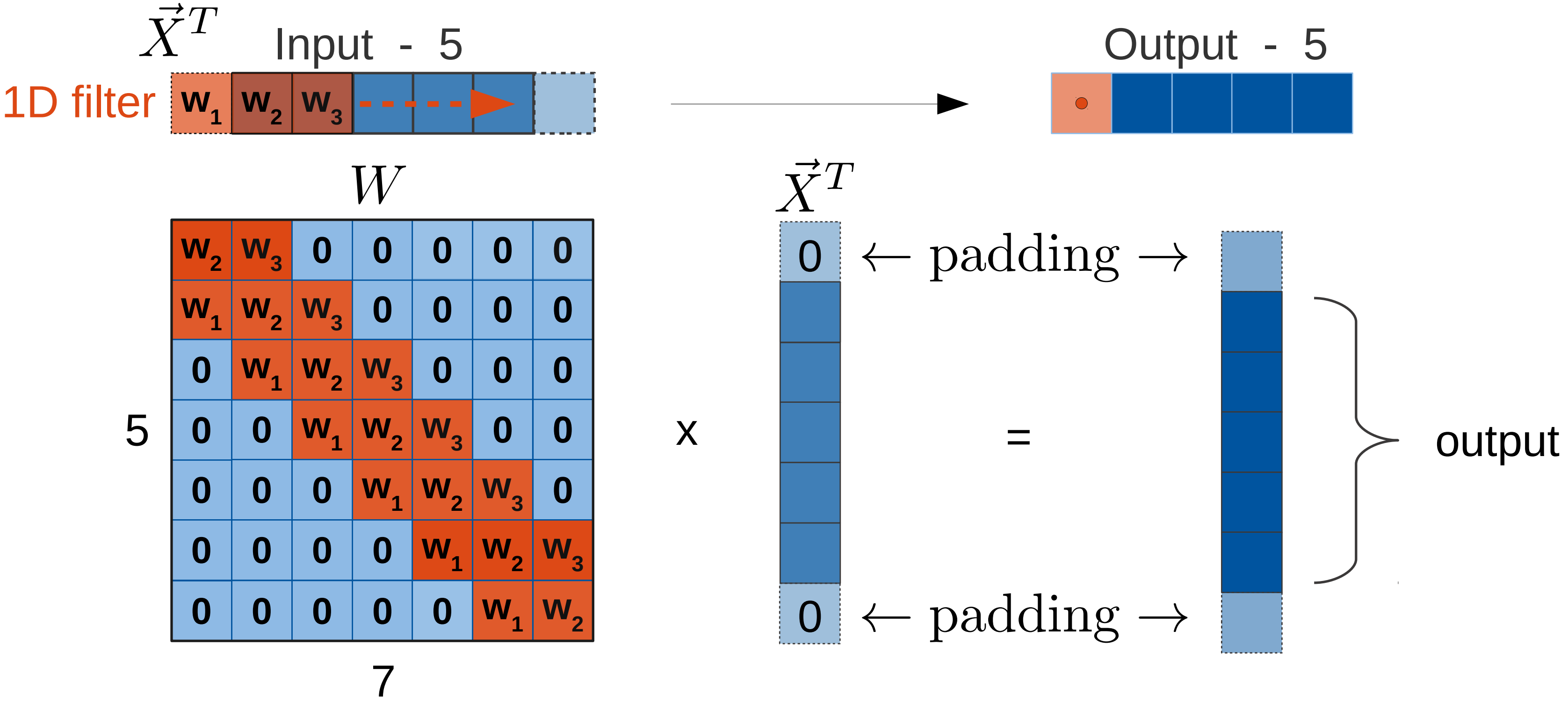
- Normalization needed in deep learning
  - Input / output normalization + batch / feature normalization
  - Weight normalization

- $\hat{A} = I + A$ is not normalized
  - Each multiplication would change feature scale!
- Normalize new adjacency matrix using *degree matrix* $\hat{D}$ of $\hat{A}$ (average over neighbor nodes)
  - $A \rightarrow \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$

- Final propagation rule: $f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}\right)$
  - Can be repeated for each layer, by sharing graph structure A

Convolutional Operation

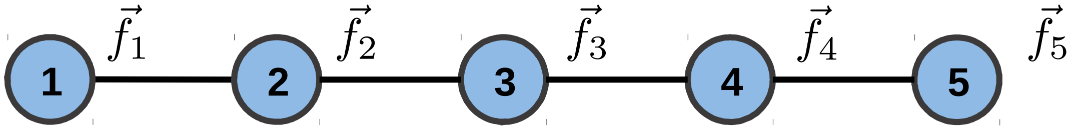
- Fully connected layers are special case of convolutional layers



  - Strong prior on **local correlation** and **translational invariance**

Graph Convolution

- Convolutional layers are special case of Graph convolutional layers



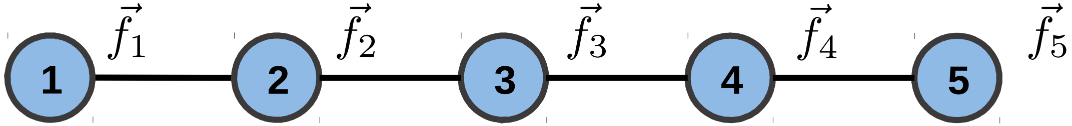
f-dim. feature vector at each node

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

$$\underbrace{\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} \vec{f}_1 \\ \vec{f}_2 \\ \vec{f}_3 \\ \vec{f}_4 \\ \vec{f}_5 \end{bmatrix}}_{H} \times \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}}_{W} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

Graph Convolution

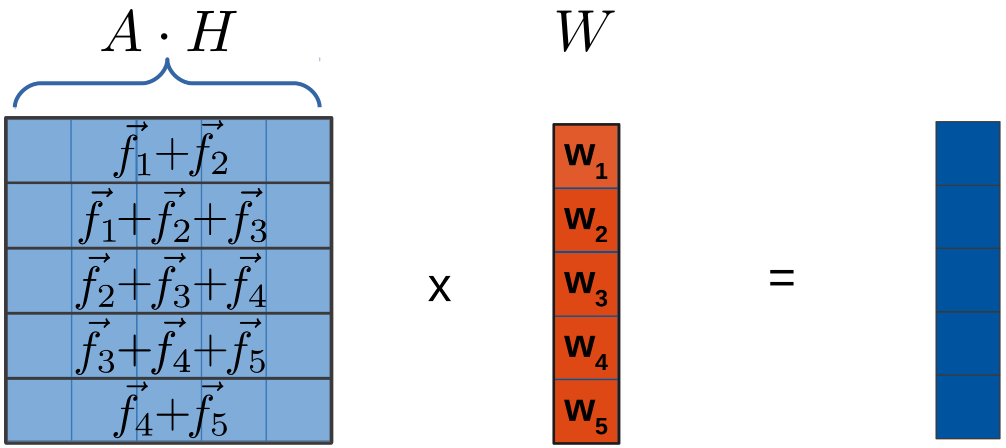
- Convolutional layers are special case of Graph convolutional layers



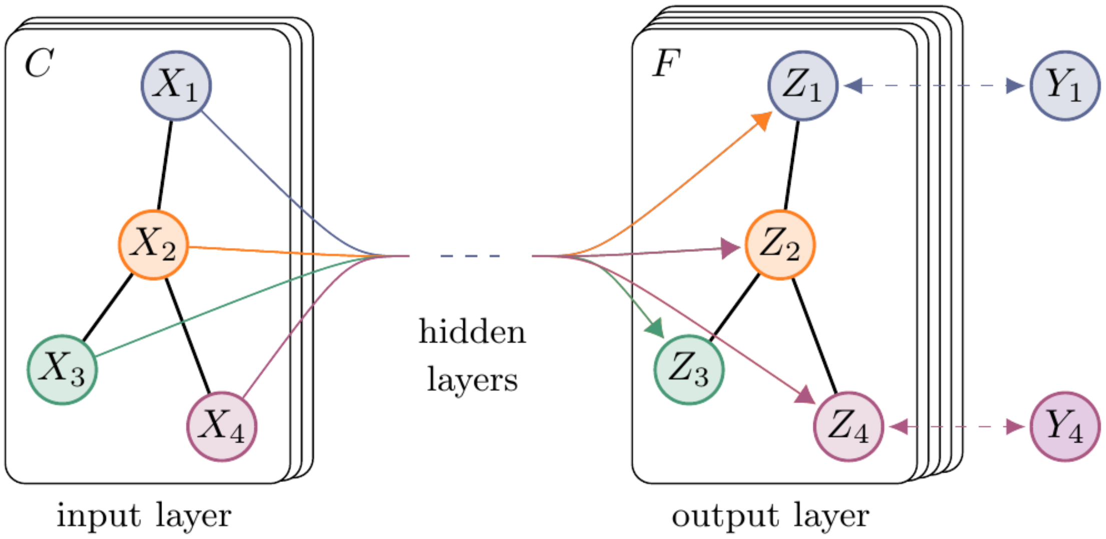
$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

$$shape(H^{(l)}) = N \times F$$
$$shape(A) = N \times N$$
$$shape(W) = D \times F$$

- Output 5 nodes
  - Structure shared over model
- Graph Convolution
  - 5 adaptive weights
- Cartesian Convolution
  - 3 adaptive weights (translational invariance)



Graph Convolutional Network - GCN



(a) Graph Convolutional Network

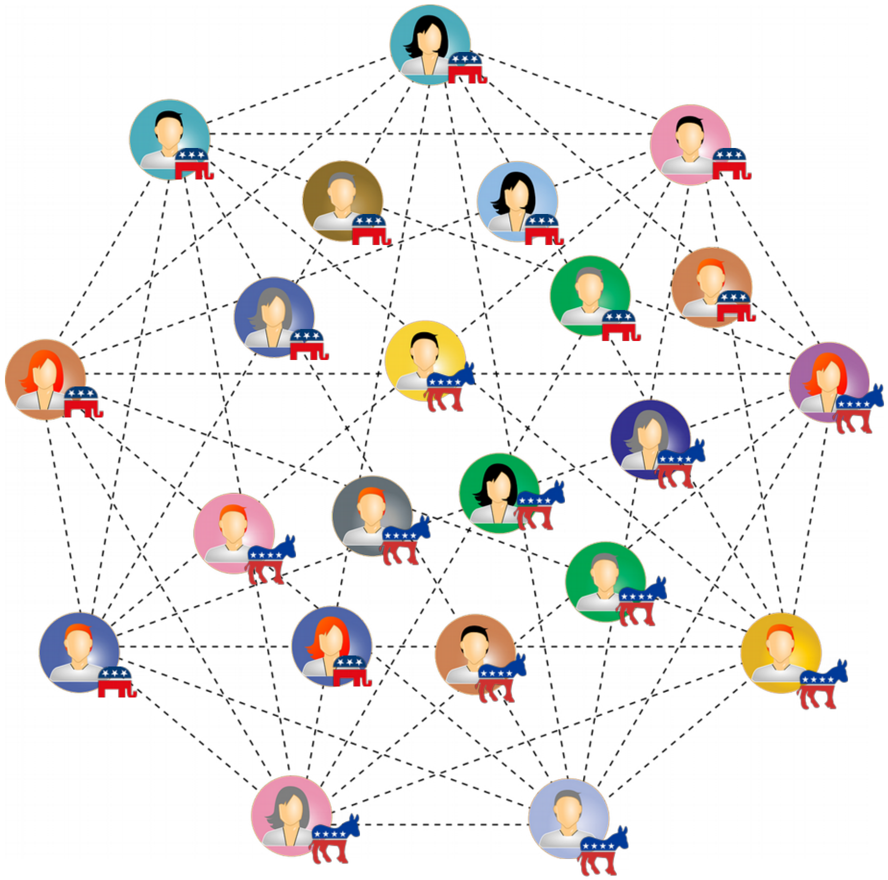
arXiv:1609.02907

- Share graph structure over model
  - Calculate once

    $$A = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$$

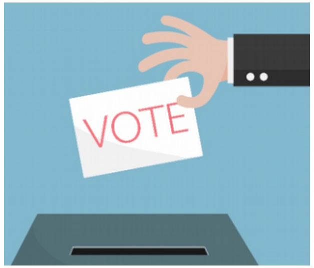
    during pre-processing
- Aggregate neighborhood information in every node
  - $$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$$

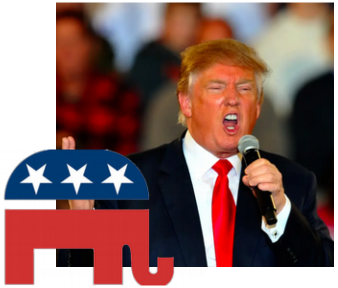
Node Classification – social network



- Node Classification of single graph
  - Social network
- Clustering / classification of nodes
  - Voting behavior of individual persons

- Semi-supervised
  - use few labels || rest of nodes masked
- Unsupervised
  - without label information



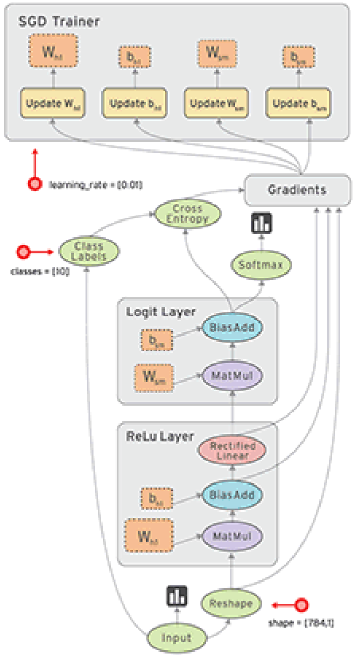


TensorFlow

"Open source software library for numerical computation using data flowing graphs"

- **Nodes** represent mathematical operations
- **Graph edges** represent multi dimensional data arrays (**tensors**) which **flow** through the graph

- Supports:
  - CPUs and **GPUs**
  - Desktops and mobile devices
- Released 2015, stable since Feb. 2017
- Developer: Google Brain



Keras

- Will use Keras in this tutorial (TensorFlow backend) - https://keras.io
  - High-level neural networks API, written in Python
- Concise syntax with many reasonable default settings
- Useful callbacks for monitoring the training procedure
- Nice Documentation & many examples and tutorials + useful extensions
- Ships with TensorFlow

- We use tf.keras 2.2.4-tf // TensorFlow 2.1

Additional Software

- We use Spektral in this tutorial, version 0.2.0
- Python library for deep learning on graphs
- Based on Keras and TensorFlow

Spektral

https://github.com/danielegrattarola/spektral

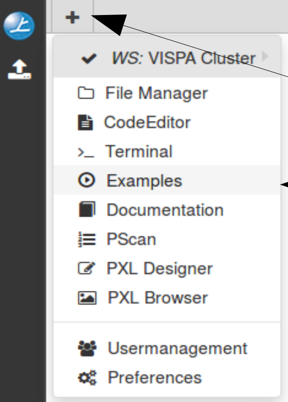
- Alternative for PyTorch users:

PyTorch geometric

https://github.com/rusty1s/pytorch_geometric

NetworkX

- For visualization of graphs we use NetworkX

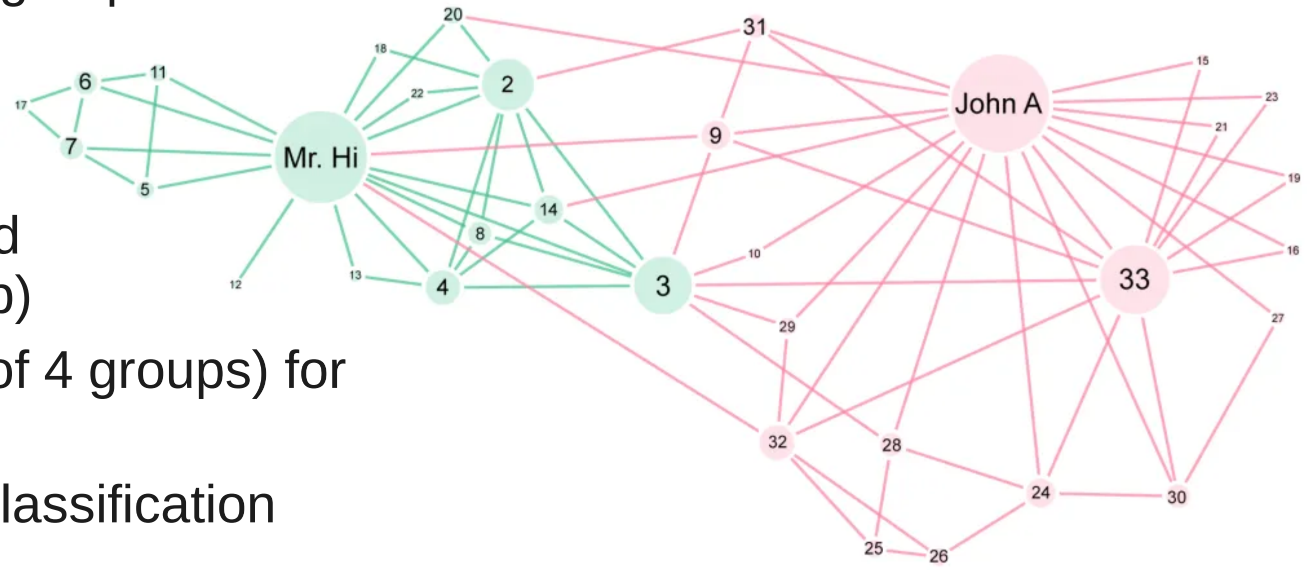


Opens the example page

- Developed in Aachen (group of Martin Erdmann)
- GPU extension
  - 20x NVIDIA GTX 1080
  - 3x RTX 6000, 6x RTX 5000
- Accessible via https://vispa.physik.rwth-aachen.de/

Zachary's Karate Club

- “Historical” Dataset
- Social network of university karate club
  - Edges represent social relationships outside the club
- Conflict between administrator “John. A” and trainer “Mr. Hi”
  - Karate Club splits in 4 groups



Task

- Given a single graph and 4 labels (1 of each group)
- Identify membership (1 of 4 groups) for every person
- Semi-supervised node classification

Embedding

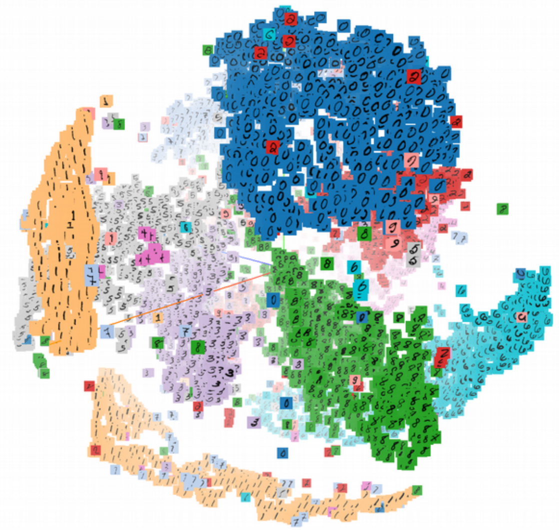
- To visualize machine learning models
- Project vectors of high dimensional space on low dimensional manifold

- Good classifier need high separation capability
  - especially at latest layers

- Most simple embedding
  - Neural network layer with 2 dimensional uttput

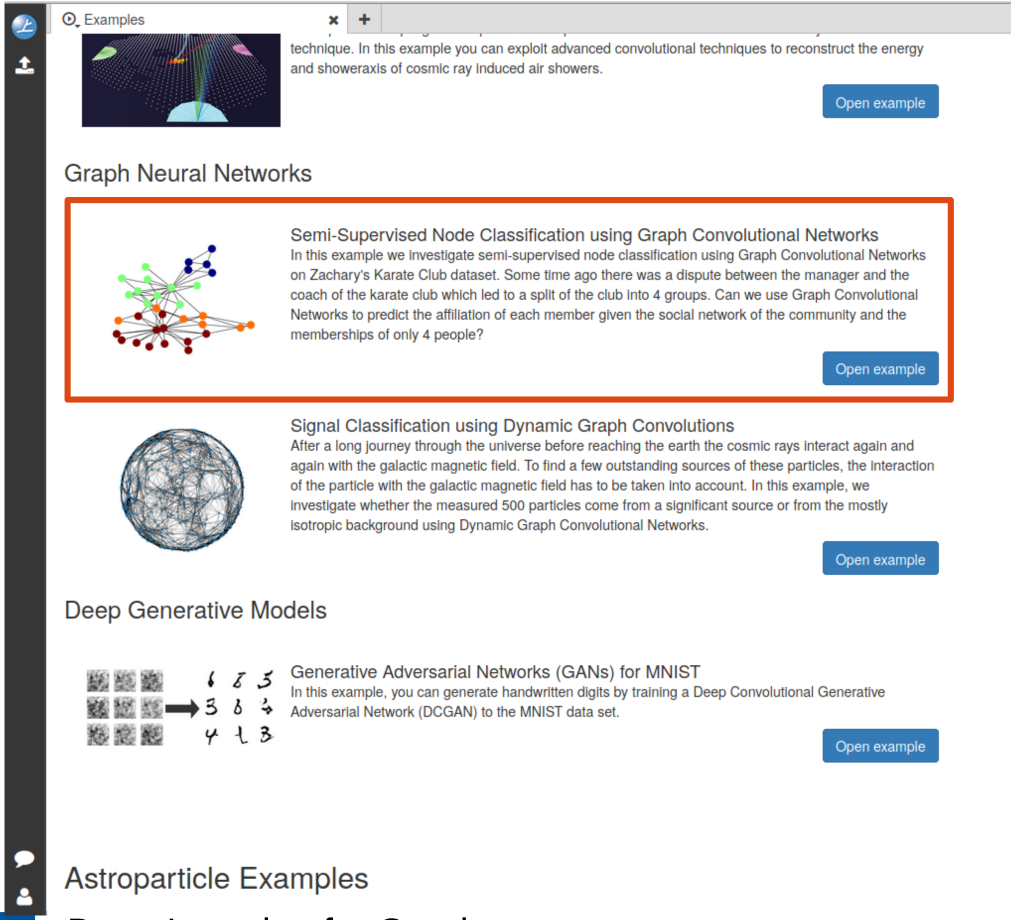
```
x = GraphConv(2, activation='tanh', name="embedding")([x, fltr_in])
```

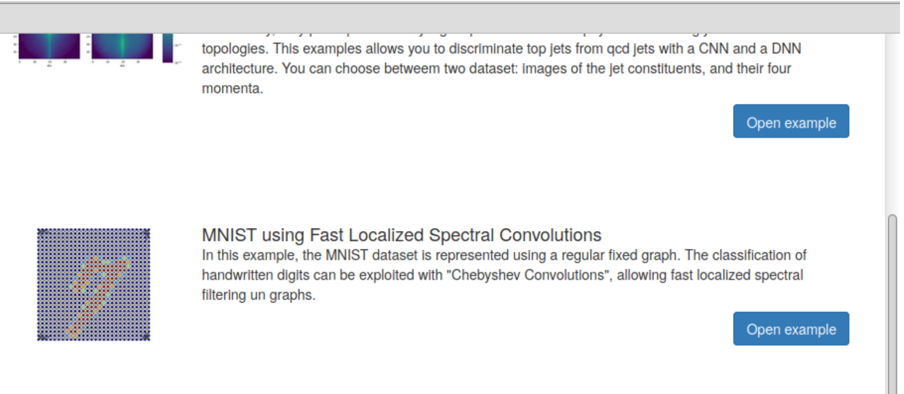
3D embedding of MNIST



https://projector.tensorflow.org/

Practice I

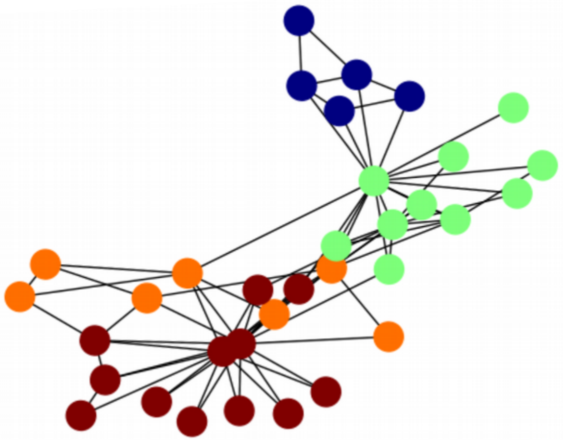


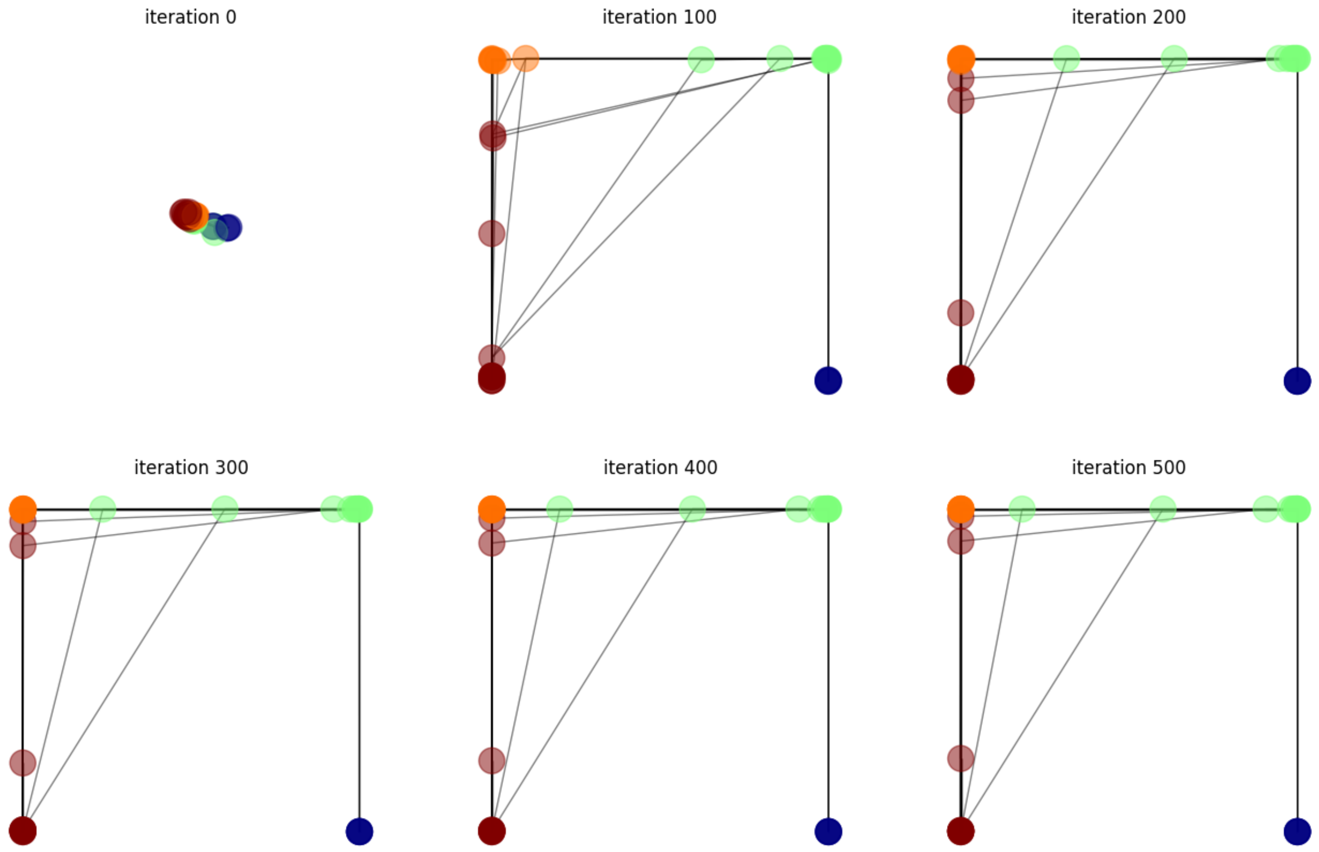


- Reach accuracy > 90%
- Change hyperparameters:
  - Number of features, Learning rate, epochs, layers ...

Practice 1 – Karate Club Network

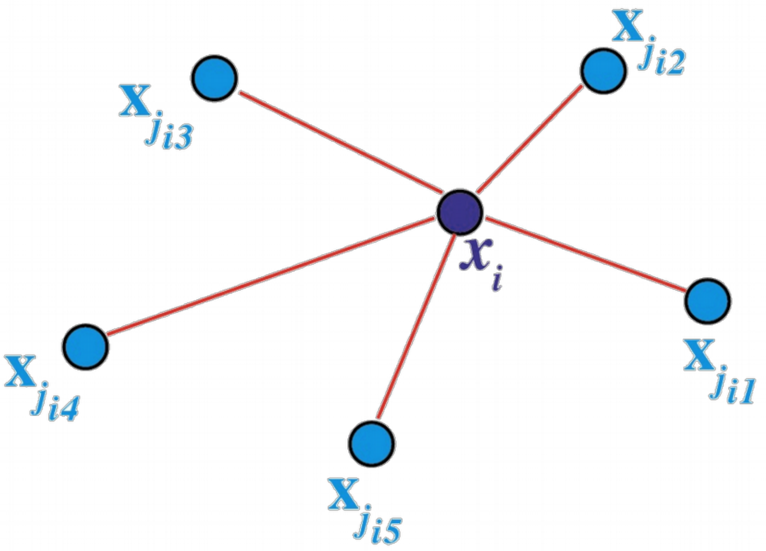
- Tune learning rate
- Increase iterations
- Well connected labels





Convolutions in the Spatial Domain

- Edge-Convolutions
- Dynamic Graph Convolutional Neural Networks
- Physics example



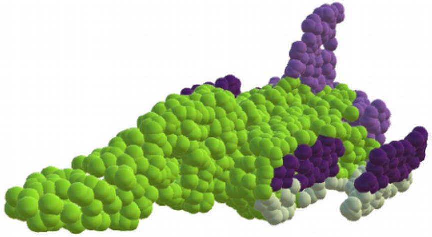
Y. Wang et al.
ArXiv:1801.07829

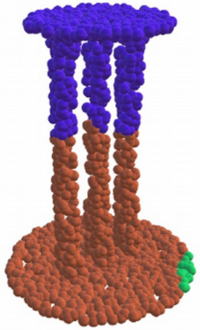
M. Simonovsky, N. Komodakis
ArXiv:1704.02901

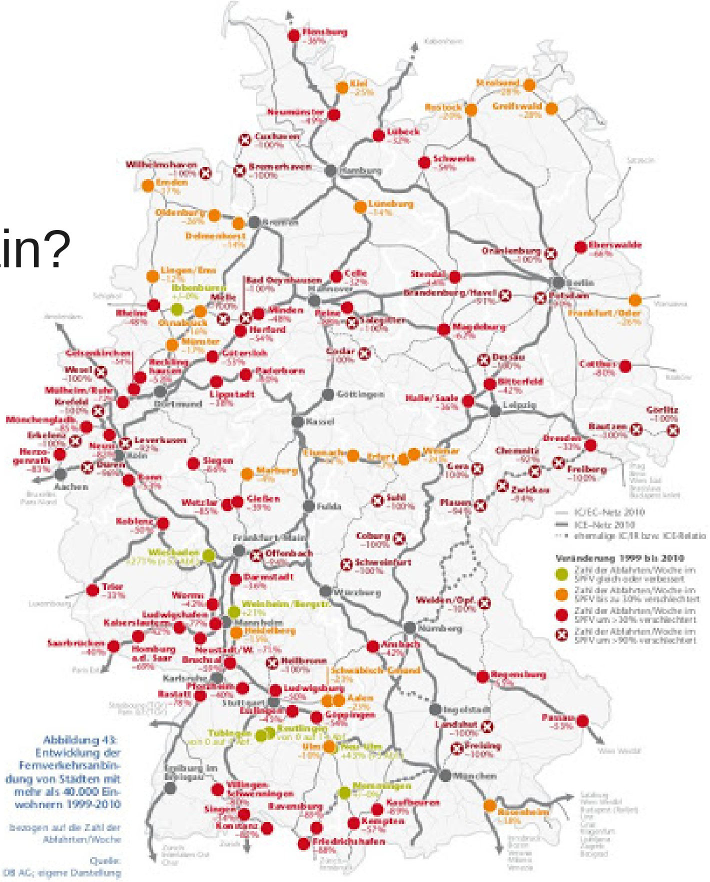
Convolution in Spatial Domain

- Graphs feature permutational invariance of nodes
- Orientation of nodes meaningless

- Whats with networks embedded in the spatial domain?
  - Node position is important!
  - Not only neighborhood relationship!



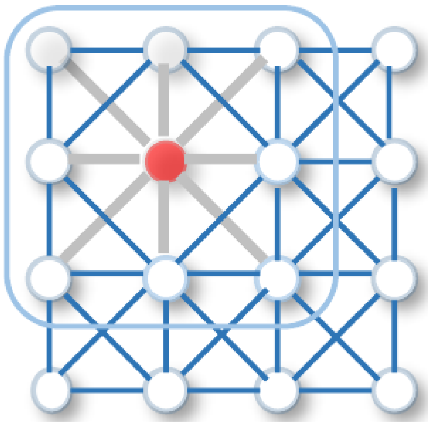




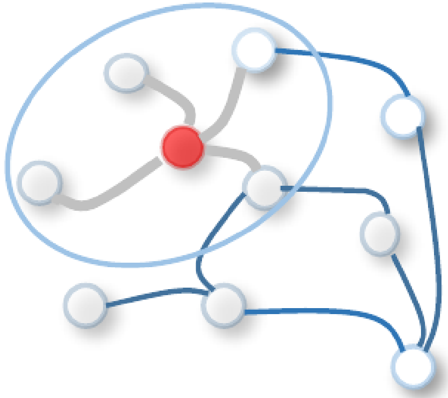
Convolution in Spatial Domain

- Images with discrete and continuous pixel coordinates

Discrete grid positions



Continuous grid positions

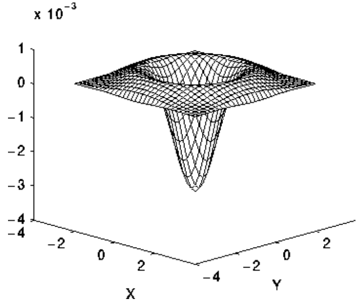


- Learned filter

$$\mathbf{D}^2_{xy} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

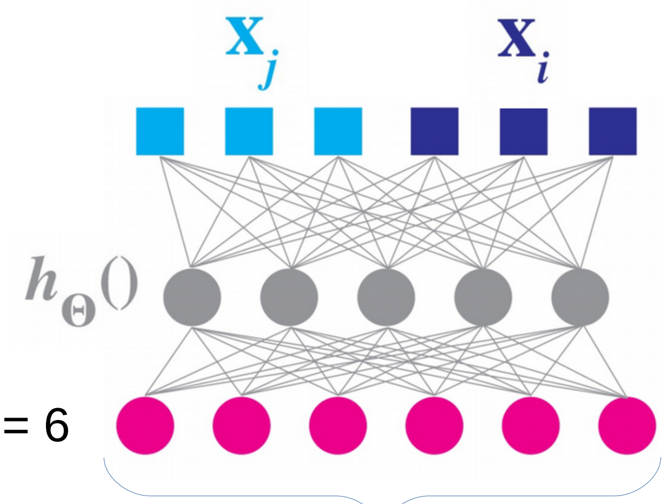
......?

Transition of discrete filter to continuous filter

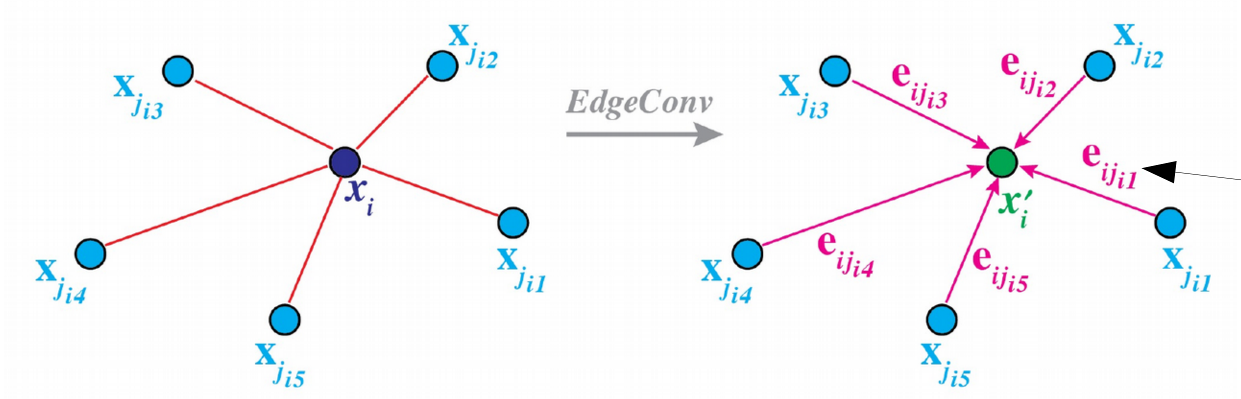


EdgeConvolution

- ✗ For continuous pixelization → matrix become gigantic and sparse
- ➔ Approximate discrete f-dimensional kernel using deep neural network
- Network applied at each pixel using:
  - central pixel x_i
  - relation to neighbor pixels eg. x_j or $x_i - x_j$
- Outputs f-dimensional feature vector



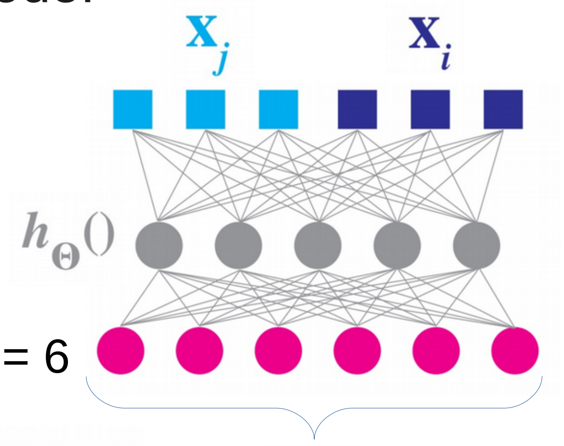
Calculate $\mathbf{e}_{ij}$ for each adjacent node $\mathbf{x}_j$



https://arxiv.org/abs/1801.07829

Edge Convolution

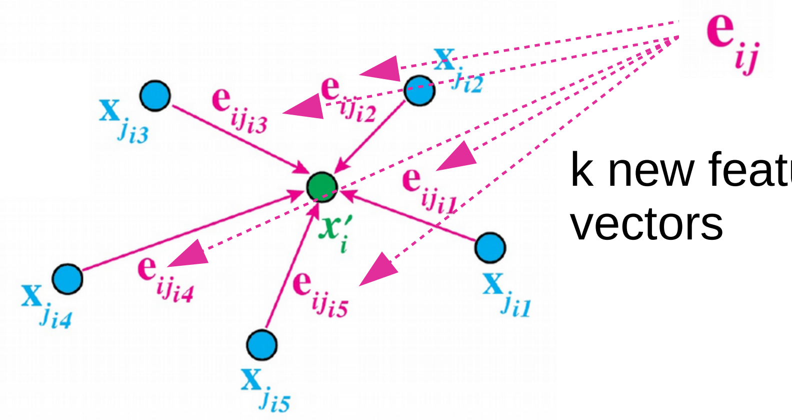
- Convolution acts on neighborhood $\mathbf{x}_i$ yielding for each node:
  - k new features $\mathbf{e}_{ij}$ (one for each neighbor)
  - feature dimension depends on features of $h_\Theta()$
  - **Parameters shared over edges**



- Aggregate neighborhood information
- Aggregation operation flexible:
  - eg.

$$x'_i = \max_{j \in N_i} e_{ij}$$

$$x'_i = \langle e_{ij} \rangle_{j \in N_i}$$

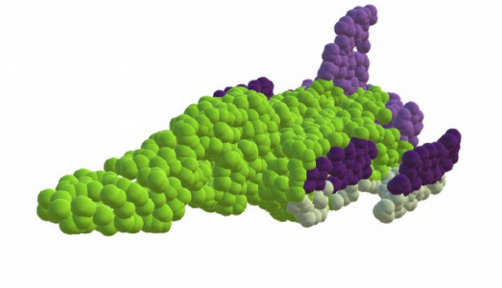


k new feature vectors

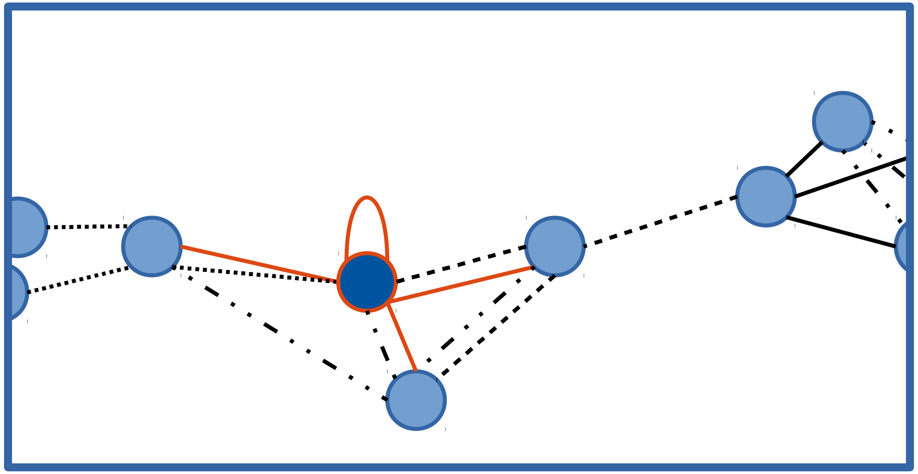
https://arxiv.org/abs/1801.07829

Define Graph with kNN Algorithm

- Before applying EdgeConv
  - Define underlying graph
- Find neighbors using kNN clustering
  - Smallest euclidean distance in feature space
  - Directed graph



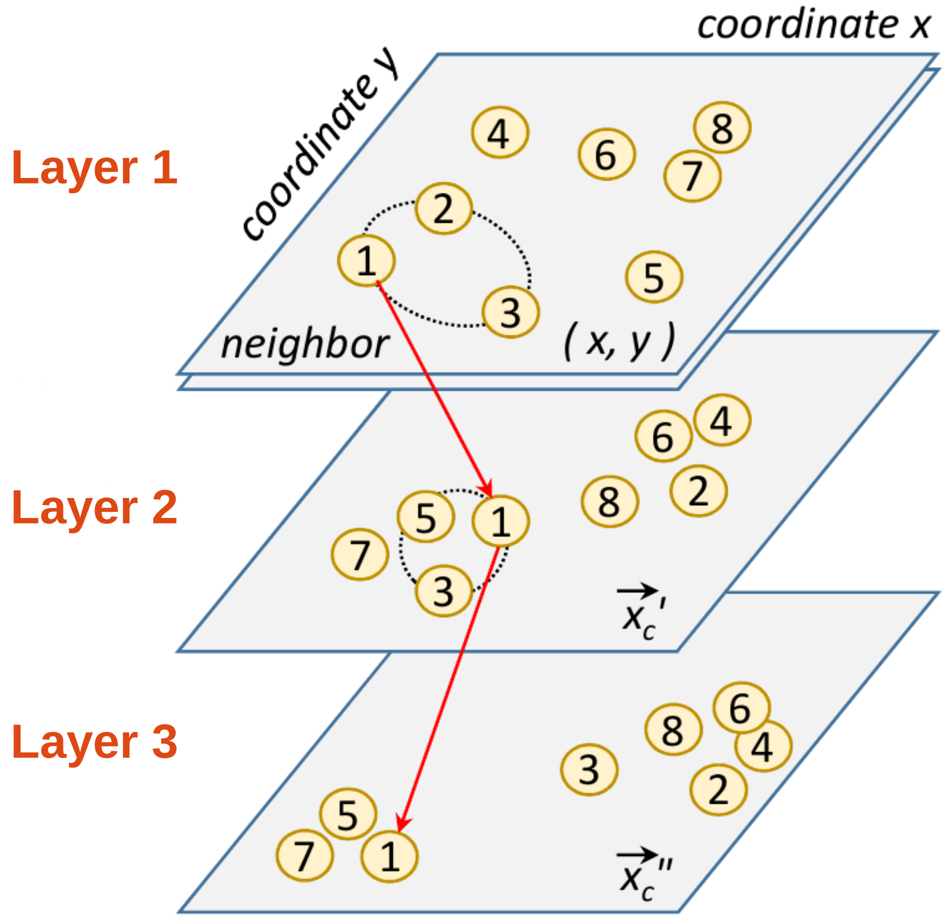
- Edges can be updated in each layer
  - **In feature space neighbors change**
  - *Dynamical* update of graph



Dynamical Graph Update

- In each layer neighbors of nodes change
- Update of graph using kNN
- DNN can not directly learn neighbor relations
  - **kNN has no gradient**

- Implicit clustering of nodes
  - Nodes with same features are embedded similar
    - Become neighbors

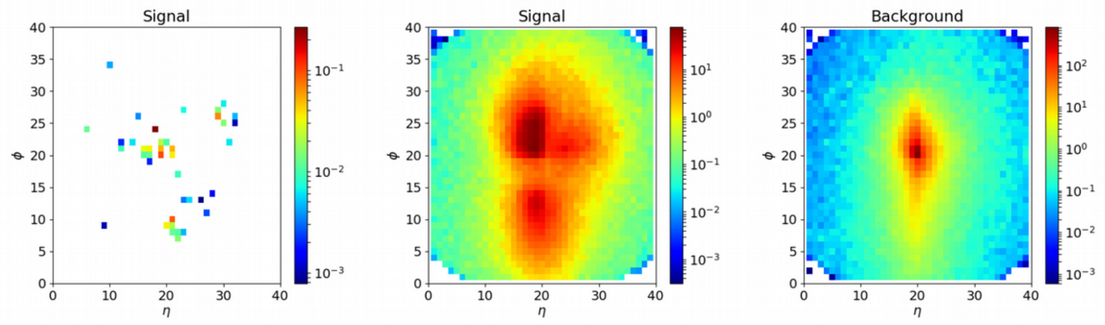


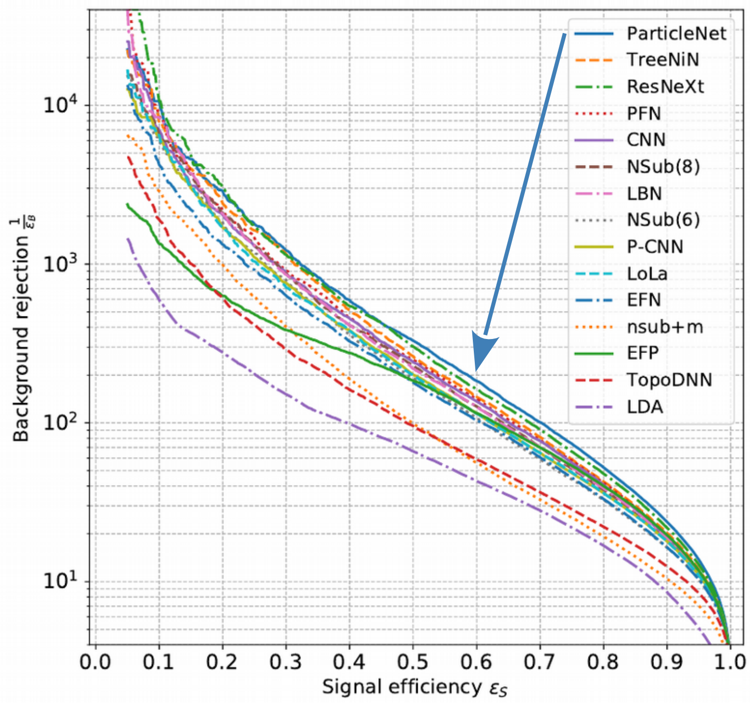
Erdmann et al.: Identification of Patterns in Cosmic-Ray Arrival Directions using Dynamic Graph Convolutional Neural Networks

Example: Jet Tagging via Particle Clouds

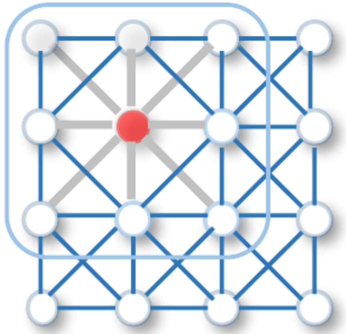
- Challenge in high-energy physics
- Input: Particle cloud
  - Permutational invariance!
- Classify jets into: **1.** top quarks **2.** background

- ParticleNet won championship
  - Using 3 EdgeConv Layer

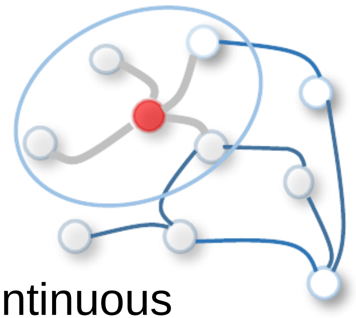




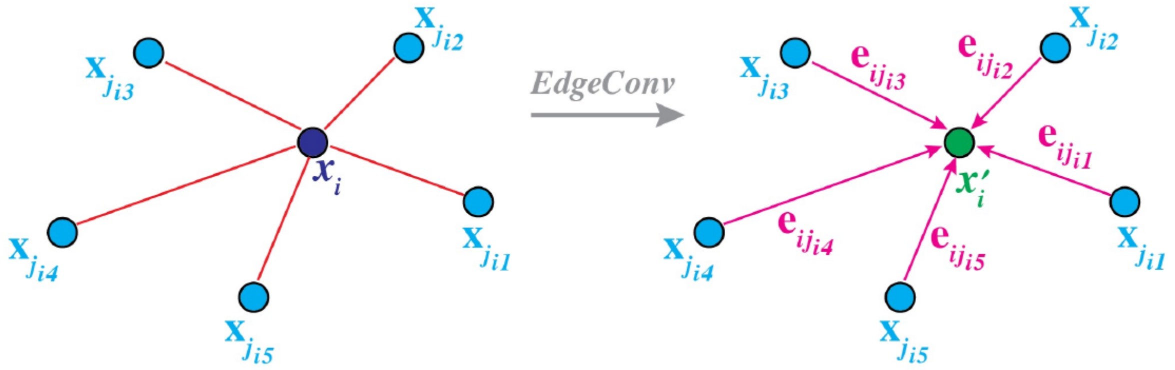
Summary: Dynamical Graph Convolution



Discrete
grid positions



Continuous
grid positions



Input: size x (features)

1. Search **k next neighbors**
2. **Convolve** signals
   → size x (k, channels)
3. **Aggregate** signals
   → size x (channels)

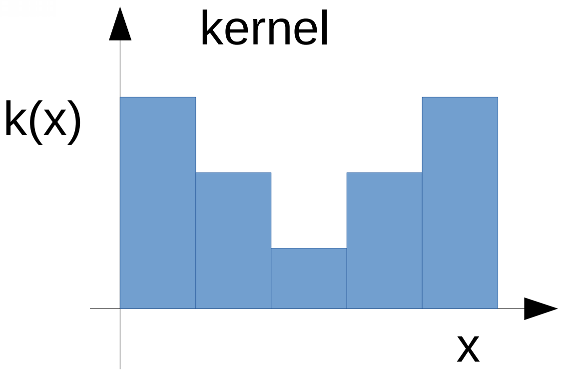
→ Repeat if you want

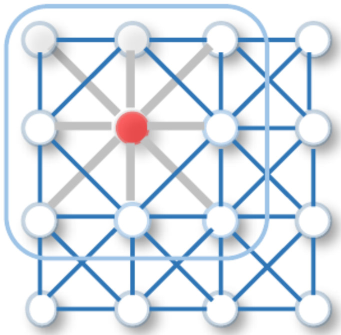
$$\boldsymbol{x}'_i = \mathop{\square}_{j=1}^{k} \boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) = \bar{\boldsymbol{h}}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_{i_j} - \boldsymbol{x}_i),$$

Use DNN

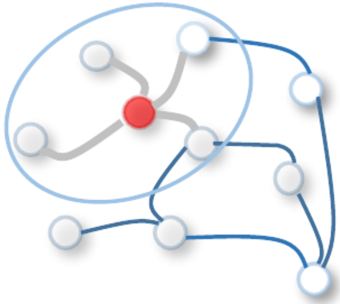
$$x_i = \sum_j \mathrm{ReLU}(\boldsymbol{\theta}_m \cdot (\mathbf{x}_j - \mathbf{x}_i) + \boldsymbol{\phi}_m \cdot \mathbf{x}_i),$$

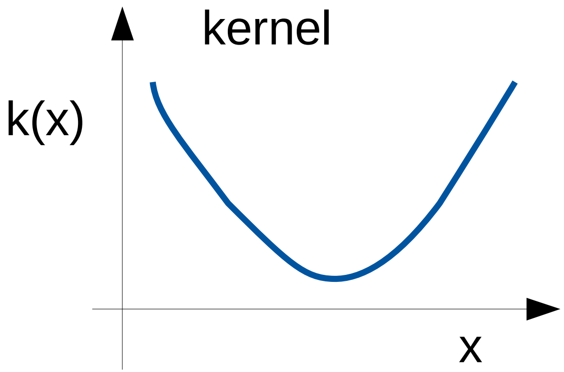
Convolution vs Dynamical Convolution





Discrete grid positions





Continuous grid positions

Similarities:

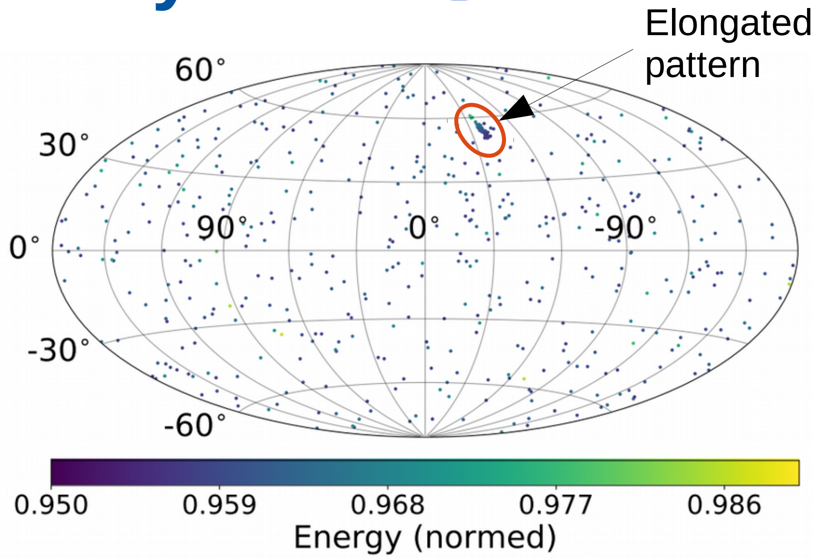
- Localized convolution
- Feature symmetry invariance in data: (translation, rotation, permutation)
  - depends on your chosen $h_\Theta()$
  - → Weight sharing over pixel positions

Differences:

- Image: convolution at positions over features
  - Neighbor points stay neighbors
- Graph: at features over features
  - **Neighbors can change!**

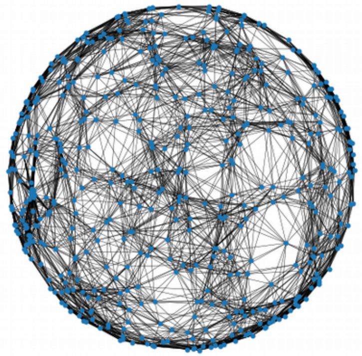
Example Classification of Cosmic Rays

- Ultra-high energy cosmic rays deflected by galactic magnetic field
- Cosmic rays induce characteristic pattern when arriving at the earth



Task

- Given skymap of 500 cosmic rays
- Using EdgeConvs classify if skymap contains
  I. Signal from single significant source
  II. Only isotropic background



Visualize formed graph in each EdgeConv layer in physics space

Helpful Comments on the Code

- Modify kernel network → change $h_\Theta()$ of EdgeConv

```
def kernel_nn(data, nodes=16):
    d1, d2 = data  # get xi ("central" pixel) and xj ("neighborhood" pixels)
    dif = layers.Subtract()([d1, d2])
    x = layers.Concatenate(axis=-1)([d1, dif])
    x = layers.Dense(nodes, use_bias=False, activation="relu")(x)
    x = layers.BatchNormalization()(x)
    return x
```

Dynamic + fixed graph updates

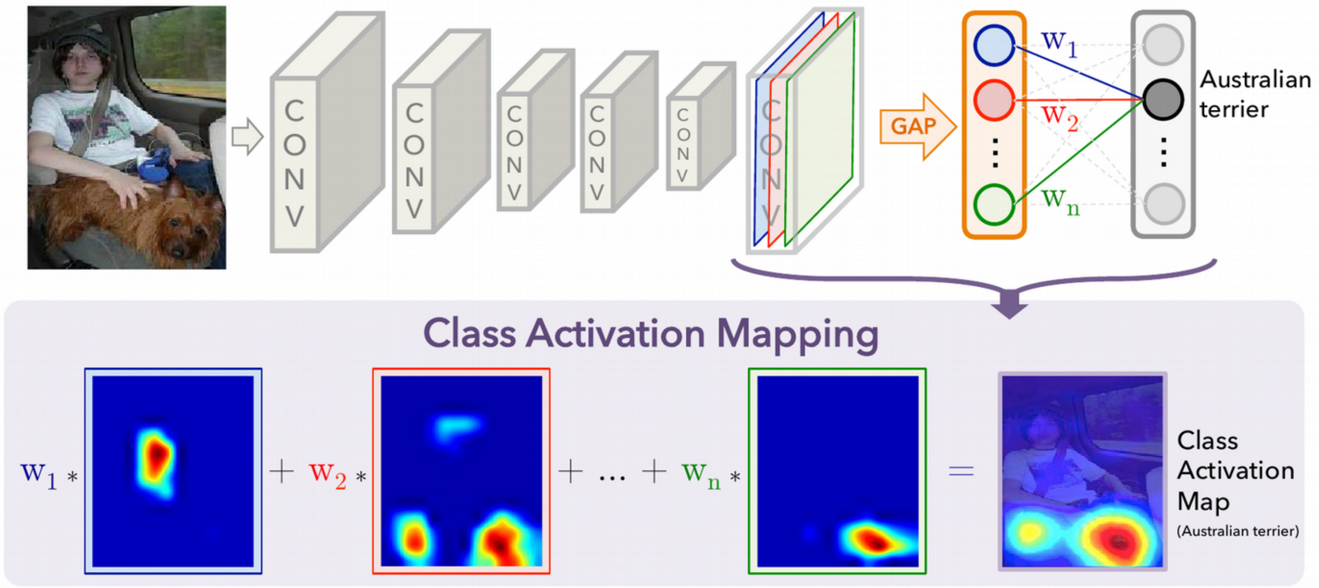
- Used *fixed* graph by passing in each layer the very first points_input

```
x = EdgeConv(lambda a: kernel_nn(a, nodes=8), next_neighbors=5)([points_input, feats_input])
```

- Use *dynamic* graph update by passing only produced feature dimension x

```
x = EdgeConv(lambda a: kernel_nn(a, nodes=16), next_neighbors=8)(x)
```

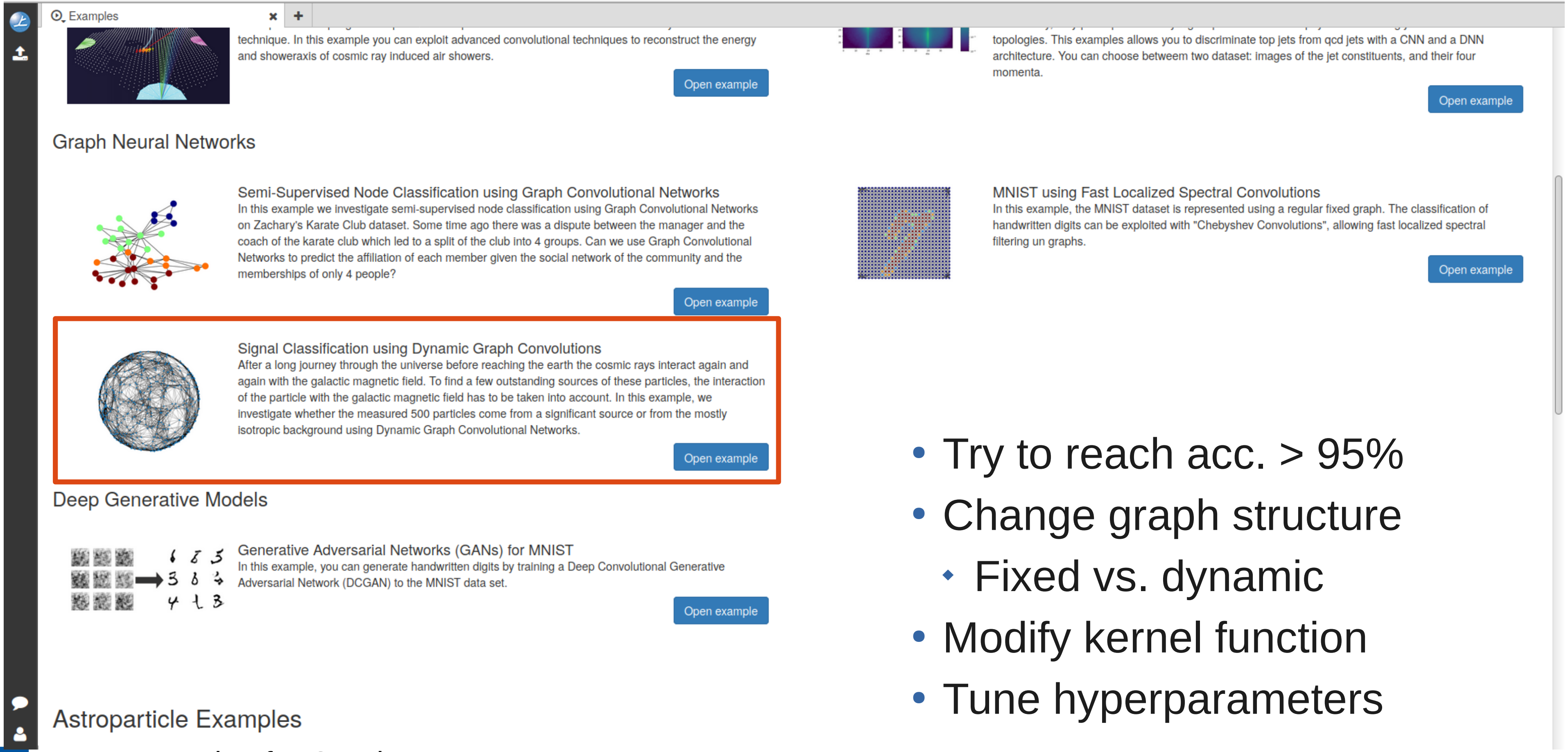
Recap Discriminative Localization



B. Zhou et al. - Learning Deep Features for Discriminative Localization

- Weights of classification layer state importance of respective feature map
  - Use superposition of last feature maps scaled with weights of classification layer
  - Map of activations indicate how the output of the last convolutional layer is used for final classification

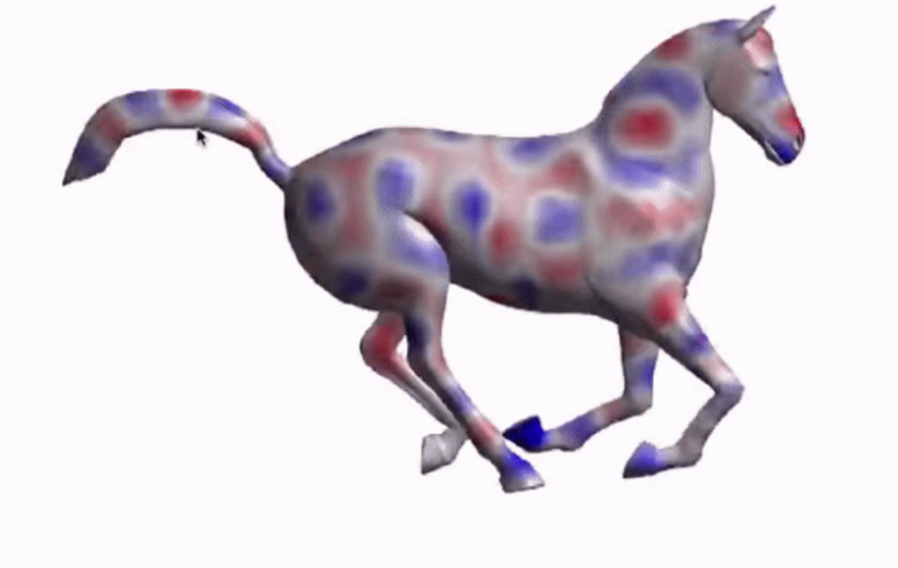
Practice II



- Try to reach acc. > 95%
- Change graph structure
  - Fixed vs. dynamic
- Modify kernel function
- Tune hyperparameters

Convolutions in the Spectral Domain

- Spectral graph theory
- Stable and localized filtering
- Chebychev Convolutions

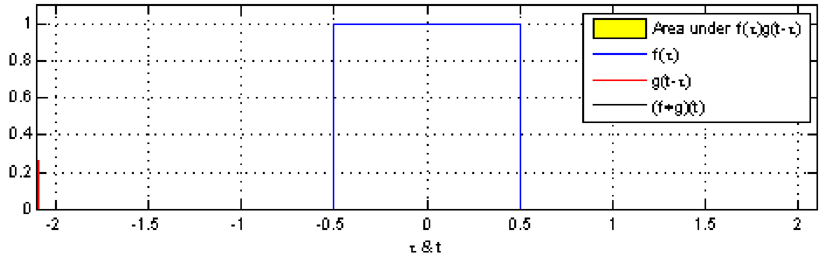


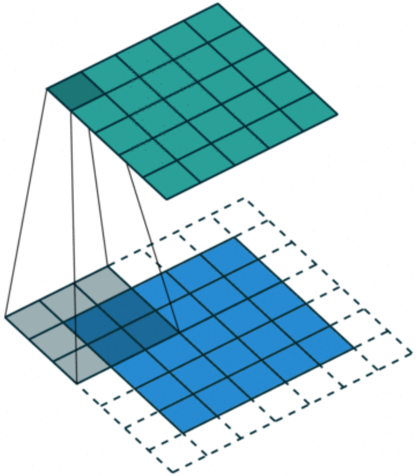
M. Defferrard, X. Bresson, P. Vandergheynst
ArXiv:1606.09375

J. Bruna, W. Zaremba, A. Szlam, Y. LeCun
arXiv:1312.6203

Convolution on Non-Euclidean Manifolds

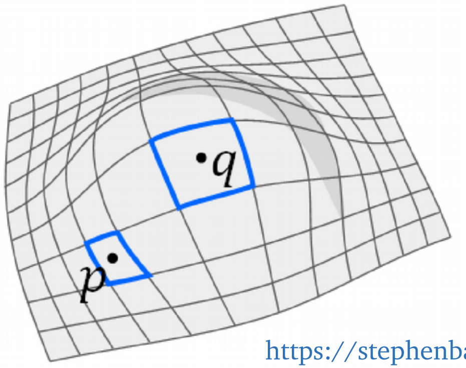
- $$(f * g)(x) := \int_{\mathcal{R}^n} f(\tau) g(x - \tau) \mathrm{d}\tau$$





Paul-Louis Pröve,
Towards Data Science

- Convolution has to include curvature of manifold
  - Filters get distorted



https://stephenbaek.github.io/projects/zernet/

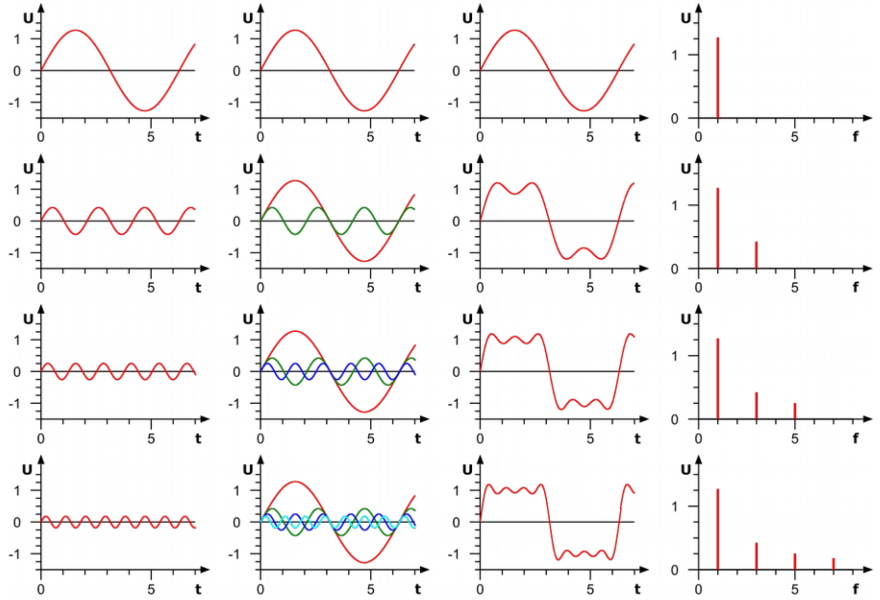
- How to convolve?

- How to make it fast?

Convolutional Theorem

- Convolution acts pointwise in Fourier domain
  - $\mathcal{F}\{f * g\} = \hat{f} \cdot \hat{g} = \hat{g} \cdot \hat{f}$
  - in Fourier domain matrices are diagonal!

$$\mathcal{F}\{f\} = \hat{f} = \Phi^T f$$



- Accelerate computation
  - $f * g = \Phi(\Phi^T f \cdot \Phi^T g) = \Phi \hat{g} \Phi^T f$

- But need to do Fourier transformation!
  - Need eigenvectors of Fourier domain

Laplacian

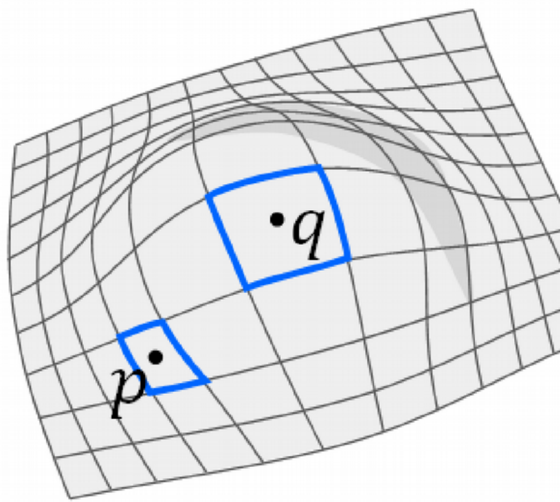
- Laplace matrix L is discrete version of Laplace operator Δ
- Laplace operator encodes smoothness/"curvature" of manifold (2nd derivative)

$$\Delta f = (\nabla \cdot \nabla) f = \mathrm{div}\,(\mathrm{grad}\ f) = \sum_{k=1}^{n} \frac{\partial^2 f}{\partial x_k^2}$$

Λ = matrix of eigenvalues

Φ = matrix of eigenvectors

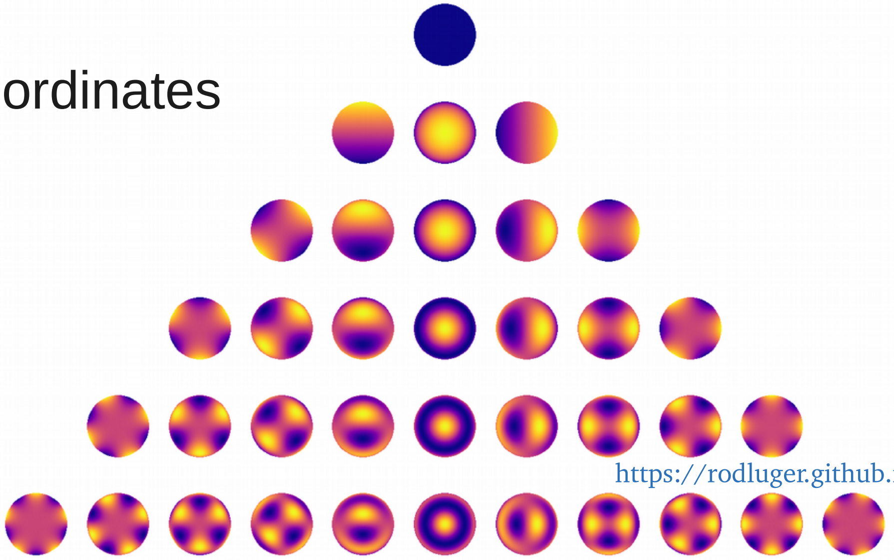
- Eigenfunctions of Laplacian form orthonormal basis
  - $\Delta f = \lambda f$, for graphs $Lf = \lambda f \rightarrow L = \Phi \Lambda \Phi^T$
- Solution directly connected to Fourier space
- Fourier basis = Laplacian eigenvectors/eigenfunctions
  - $-\dfrac{d^2}{dx^2} \exp^{ikx} = k^2 \exp^{ikx}$



Example: Spherical Harmonics

$$\left(\frac{\partial^2}{\partial \vartheta^2} + \frac{\cos\vartheta}{\sin\vartheta}\frac{\partial}{\partial\vartheta} + \frac{1}{\sin^2\vartheta}\frac{\partial^2}{\partial\varphi^2} \right) Y_{lm}(\vartheta,\varphi) = -l(l+1)Y_{lm}(\vartheta,\varphi)$$

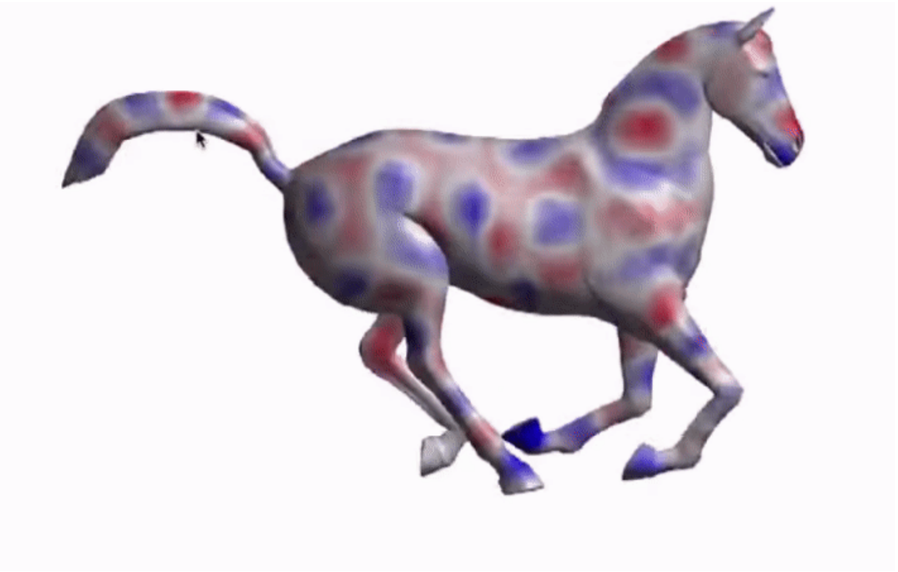
- eg. Schrödinger's equation for hydrogen atom
  - angular component breaks down to $\hat{\mathbf{L}}^2 = -\hbar^2 \Delta_{\theta,\varphi}$
- Eigenfunctions of Laplacian in spherical coordinates
  - $\Delta_{\theta,\phi} f = \lambda f$
- Spherical harmonics
  - complete and orthonormal set of eigenfunctions of angular component



Spectral Convolutions

- We can perform the convolution in the spectral domain
  - Signal $X^{(l)}$
  - Weight matrix $W^{(l)}$
- $X^{(l+1)} = \Phi(\Phi^T X^{(l)} \cdot \Phi^T W^{(l)})$
  $= \Phi \hat{W}_\theta^{(l)} \Phi^T X^{(l)}$
- $\hat{W}_\theta^{(l)} = \mathrm{diag}(\theta_1, ..., \theta_n)$

Adaptive parameters
in Fourier domain



NIPS2017: M. Bronstein, J. Bruna, A. Szlam, X. Bresson, Y. LeCun

Problems:

- Weights scale with number of graph nodes
  - Act global! No prior on local features!
- $\hat{W}_\theta^{(l)}$ strongly depends on L (Λ, Φ)
  - Bad generalization performance!

Smoothing in Spectral domain

- Approximate $\hat{W}_\theta$ in spectral domain $\tau(L)f = \Phi\tau(\Lambda)\Phi^T f$

$$\Phi(\hat{W}_\theta \Phi^T f) = \Phi \begin{pmatrix} \tau_\theta(\lambda_1) & & \\ & \ddots & \\ & & \tau_\theta(\lambda_n) \end{pmatrix} \Phi^T f$$

- $\hat{W}_\theta \approx \tau_\theta(\lambda) = \sum_{k=1}^{K} \theta_k f_k(\lambda)$

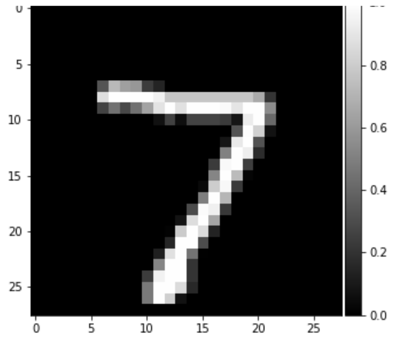
some function

adaptive parameters

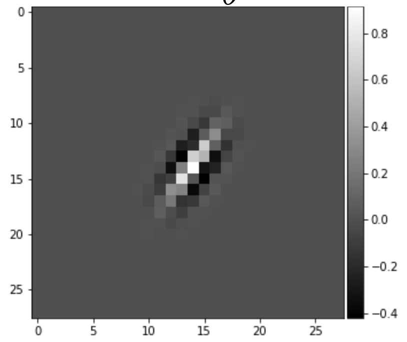
- Learn only K parameters → parameter reduction
- For $K << N$, $\hat{W}_\theta$ gets smooth in spectral domain
  - Spectral theory: filter become local!

proposed by Bruna et al. https://arxiv.org/abs/1312.6203

MNIST image$\hat{=}$Signal f

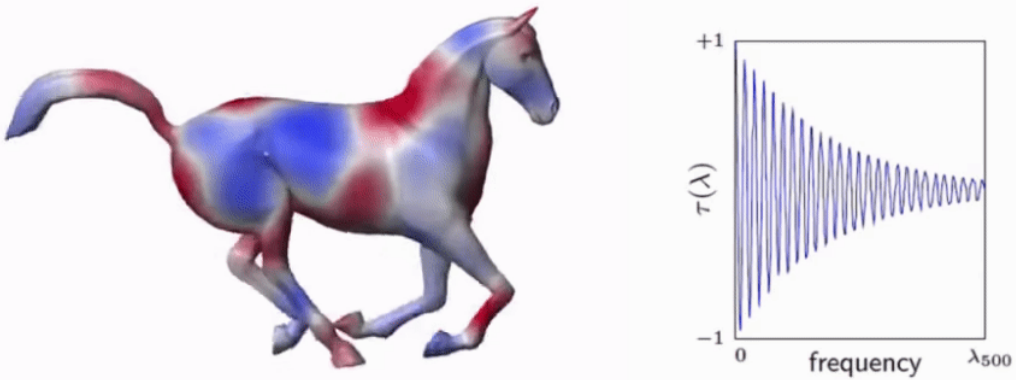


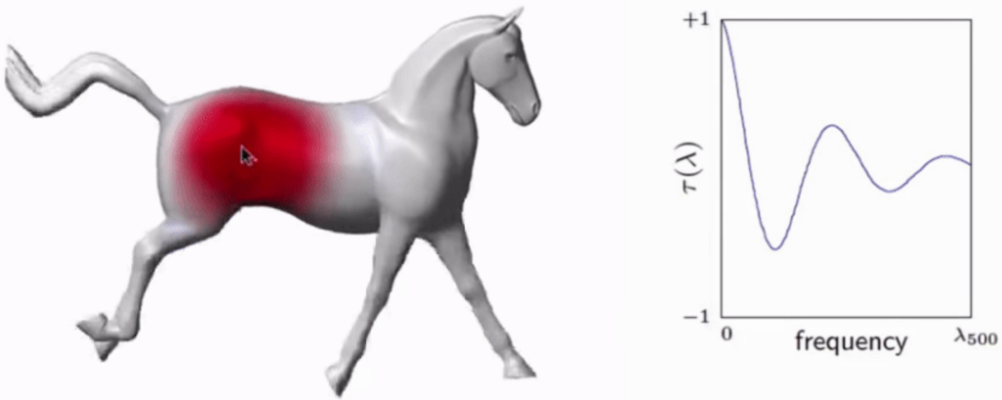
$\Phi\hat{W}_\theta$



Boris Knyazev, Towards data science

Stable and Localized Filters





NIPS2017: M. Bronstein, J. Bruna, A. Szlam, X. Bresson, Y. LeCun

- Non-smooth spectral filter
  - Not stable and delocalized
- Smooth spectral filter
  - stable and localized

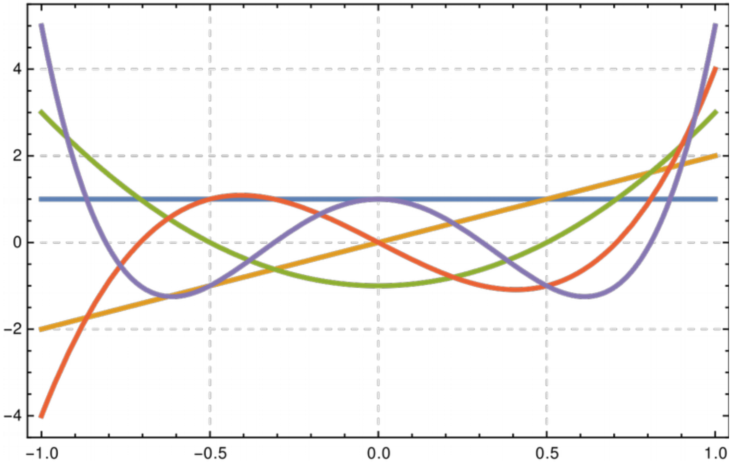
Chebychev Convolution

- Use "Chebychev polynomials" for approximation in spectral domain

$$\Phi(\hat{W}_\theta \Phi^T f) = \Phi \hat{W}_\theta(\Lambda) \Phi^T f = \hat{W}_\theta(L) f$$

$$\hat{W}_\theta(L) f \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) f \qquad \tilde{L} = \frac{2}{\lambda_{\max}} L - I$$

- Chebychev polynomials are recursively defined
  - $T_{n+1}(x) = 2x\, T_n(x) - T_{n-1}(x)$

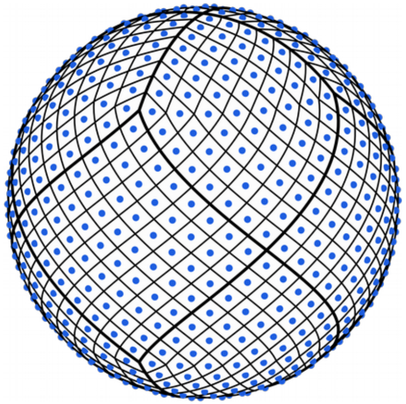


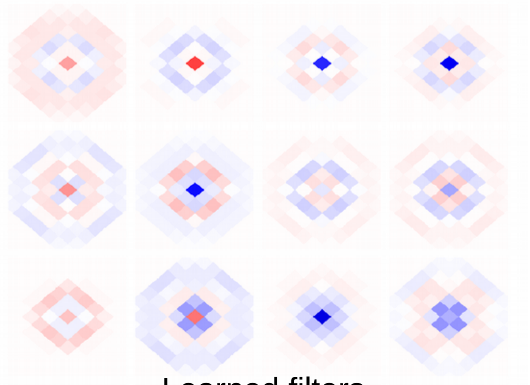
- As $T_0(L) = I,\ T_1(L) = L$
  - Calculate approximation recursive
  - No need for expensive decomposition!

Example: DeepSphere

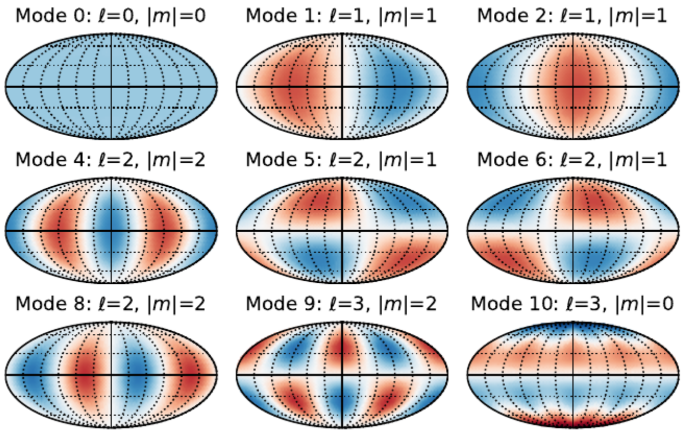
- Convolution on sphere
  - Use pixelization of HEAPix
  - Defines adjacency matrix

- Convolution via Chebychev expansion
  - Framework allows to process spherical data
  - Several properties can changed
  - Not very modular





Learned filters



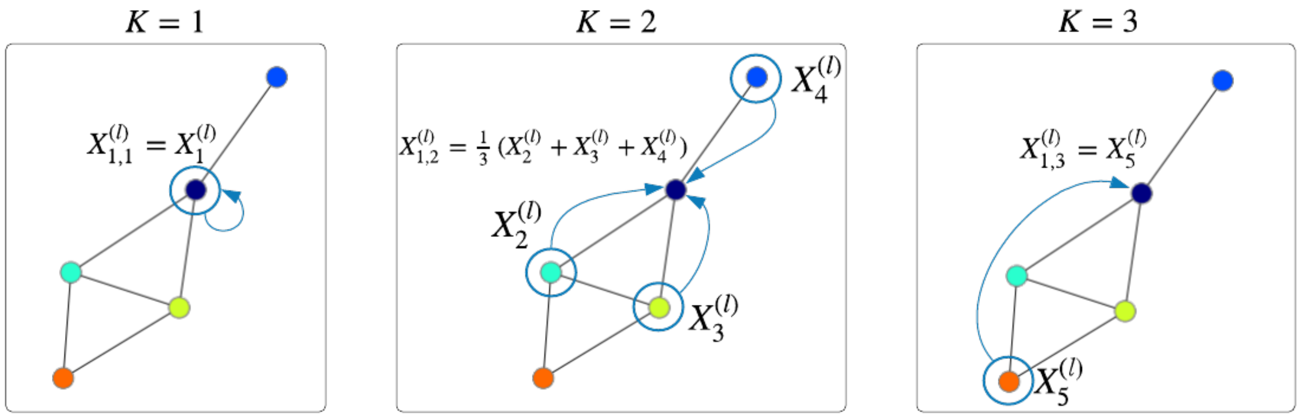
Crosscheck: eigenvectors of Laplacian

https://github.com/SwissDataScienceCenter/DeepSphere

https://arxiv.org/abs/1810.12186

Illustrative Chebychev expansion

- Using the Chebychev expansion can be seen as
  - $\sum_{k=0}^{K-1} \theta_k A^k$, weighting the neighborhood with the adjacency matrix
- Precise A^k: element ij = number of walks of length n from node i to node j



Result of the Chebyshev convolution: $X_1^{(l+1)} = [X_{1,1}^{(l)}, X_{1,2}^{(l)}, X_{1,3}^{(l)}]W^{(l)}$

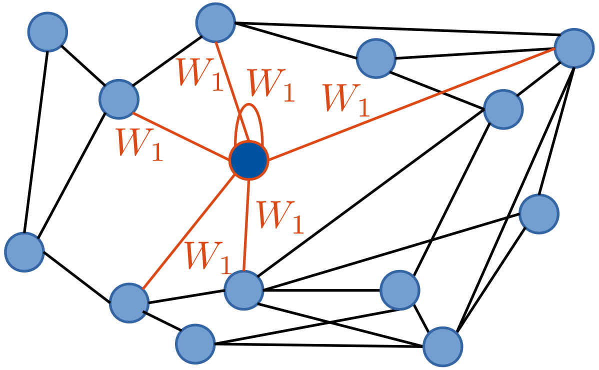
Boris Knyazev, Towards Data Science

First Order Approximation

- Approximation of Chebychev:

$$g * x = \Phi \hat{g}_\theta \Phi^T x \approx \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L}) x$$

- Evaluate for k=1
  - $g * x \approx \theta_0 x + \theta_1 (L - I) x$, setting $\lambda_{\max} \approx 2$
    $= \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x$
- Setting $\theta_0 = -\theta_1$ and remembering $\hat{A} = I + A$
  - $A = \theta_1 \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} x$

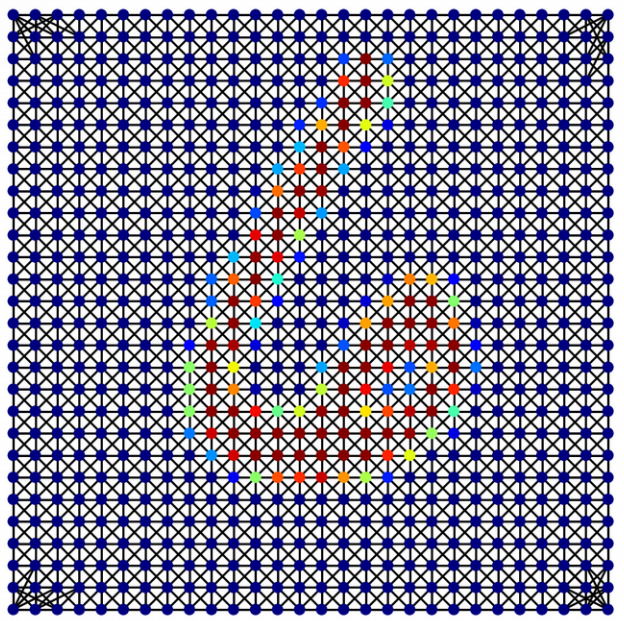


add self connection

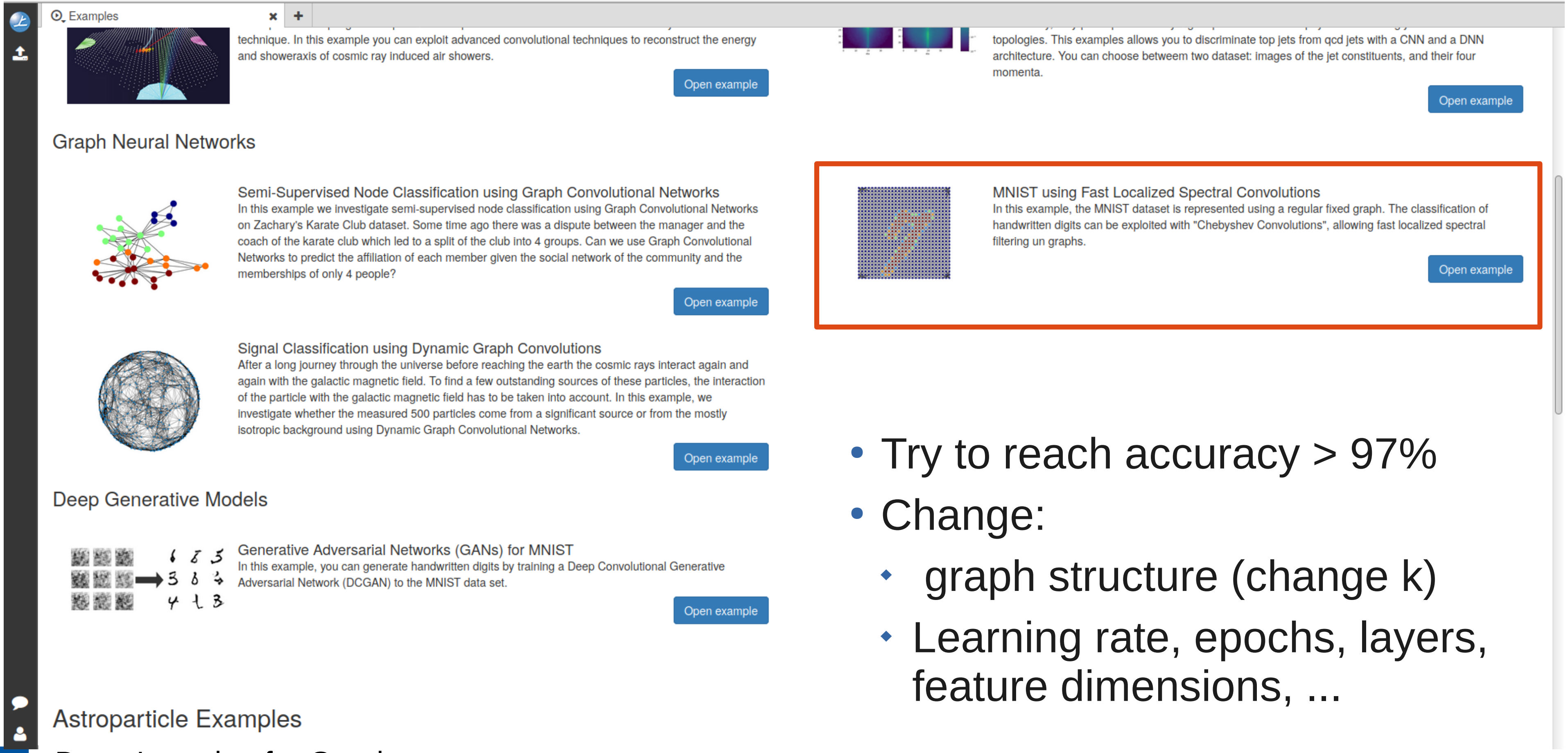
- Propagation rule of GCN (Part I.) $f(X, A) = \sigma \left(\hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}} X W \right)$
  - **GCN is first order approximation of ChebNet!**

Example MNIST

- Projection of MNIST on graph
  - Each nodes has 8 neighbors (kNN clustering)
  - Fixed domain (adjacency matrix fixed)
- Use ChebNet to classify handwritten digits
- MNIST
  - 10 classes
  - Training 50k samples
  - Testing 10 samples



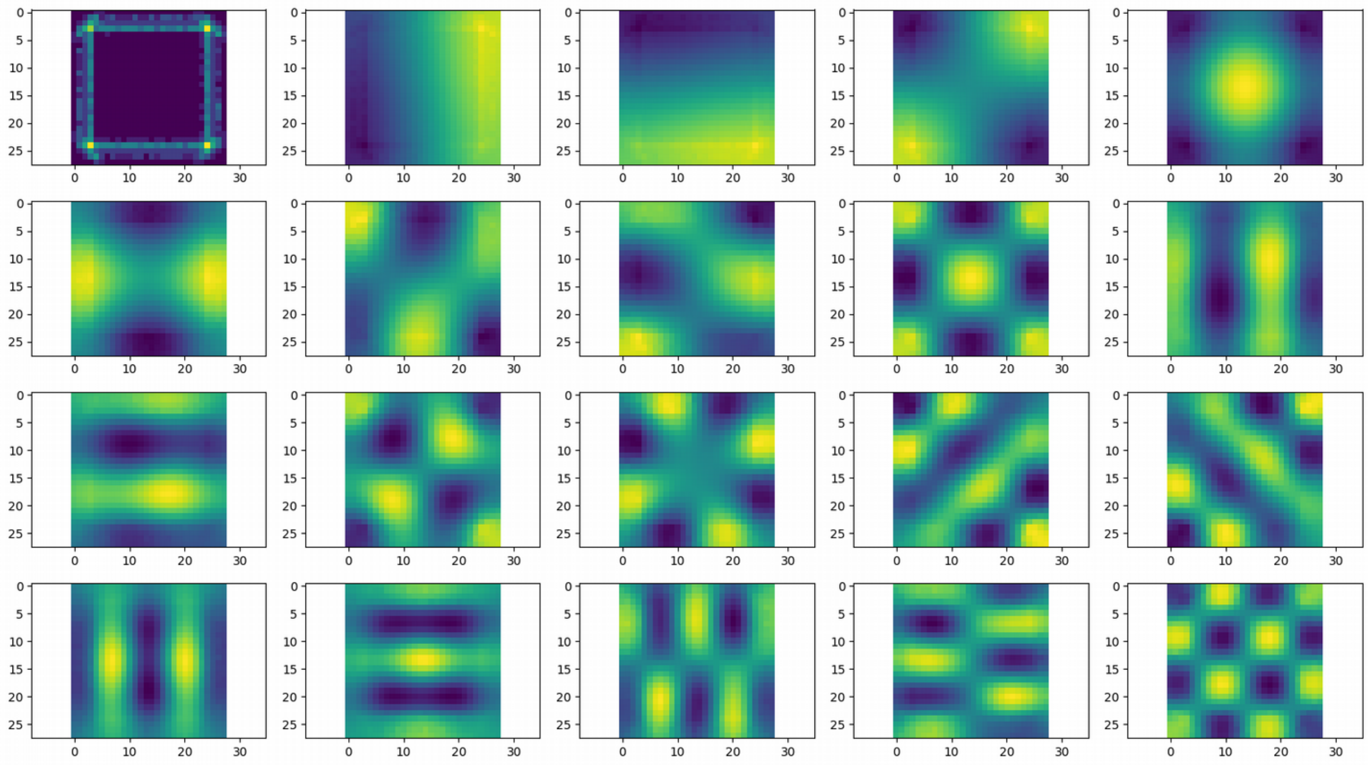
Practice III

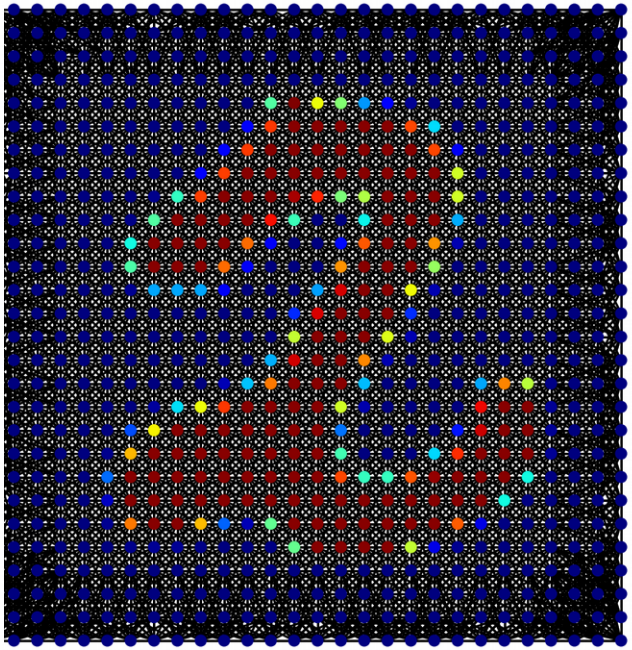


- Try to reach accuracy > 97%
- Change:
  - graph structure (change k)
  - Learning rate, epochs, layers, feature dimensions, ...

Eigenvectors of Graph Laplacian

- 20 first eigenvectors of L





- MNIST sample Graph k=20

Summary

Contra

- GCNN can be very slow
  - Euclidean convolution much faster!
- Many versions and implementations
- Very good implementations are rare
  - Hard to say which one performs best

Pro

- Very flexible
- Can more intuitive on structured data than euclidean convolutions
- Able to exploit many symmetries
  - Also on euclidean manifolds!
- No pixelisation effects

- Using standard methods → Is your model able to exploit all symmetries in data?
  - ➢ Choose architecture which best fits for your symmetry!

"In AI, 'system' should be understood as including the human engineers. Most of the 'data → generalization' conversion happens during model design." - F. Chollet

Links & Resources

- Deep Learning (Goodfellow, Bengio, Courville), MIT Press, ISBN: 0262035618
- Erdmann, Glombitza, Klemradt: Deep Learning in Physics Research", lecture series at RWTH Aachen
- Francois Chollet: Deep Learning with Python, MANNING PUBLICATIONS
- An Introduction to different Types of Convolutions in Deep Learning, Paul-Louis Pröve
  https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
- Michael M. Bronstein et al. : Geometric deep learning: going beyond Euclidean data: ArXiv:1611.08097
- Thomas Kipf, Max Welling: ArXiv:1609.02907
- M. Defferrard, X. Bresson, P. Vandergheynst: ArXiv:1606.09375
- J. Bruna, W. Zaremba, A. Szlam, Y. LeCun: ArXiv:1312.6203
- Y. Wang et al.: ArXiv:1801.07829
- M. Simonovsky, N. Komodakis: ArXiv:1704.02901
- E. Hoogeboom, J. Peters, T. Cohen, M. Welling: ArXiv/1803.02108
- Boris Knyazev, Towards data science, Tutorial on Graph Neural Networks for Computer Vision and Beyond
- M. Bronstein, J. Bruna, A .Szlam, X. Bresson, Y. LeCun: Tutorial Geometric Deep Learning on Graphs and Manifolds, https://www.youtube.com/watch?v=LvmjbXZyoP0&t=3813s, NIPS2017

Graph Neural Networks

Backup

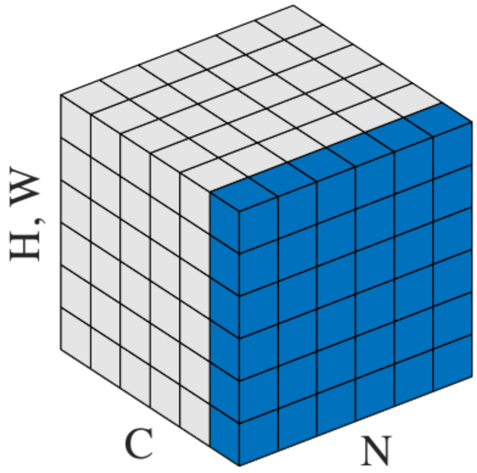
Jonas Glombitza, Martin Erdmann

Big Data Science in Astroparticle Research, Aachen, 17-19 February 2020

Batch Normalization

- Calculate batch-wise for each channel:
  - Mean: μ_B
  - Variance: σ_B^2
  - Add free parameters $\gamma,\ \beta$ to change scale and mean

➢ $$y = \frac{x - \mu_B}{\sigma_B}\gamma + \beta$$

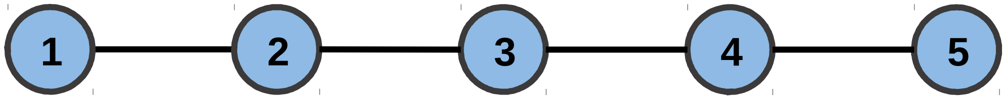


- Makes DNN robust against poor initializations
- Helps with vanishing gradient / less sensitive to high learning rates
- Has regularizing effect (no large weights, noise because of batch dependency)
- Reduce internal covariate shift

➢ **Very successful for convolutional architectures**

Graph Convolution

- Convolutional layers are special case of Graph convolutional layers



$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

$$shape(H^{(l)}) = N \times F$$
$$shape(A) = N \times N$$
$$shape(W) = D \times F$$

A

0	0	0	0	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	0	0	1	1	0
0	0	0	0	0	0	0

x

H

0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	0
0	0	0	0	0	0	0

x

W

w_1
w_2
w_3
w_4
w_5
w_6
w_7

=