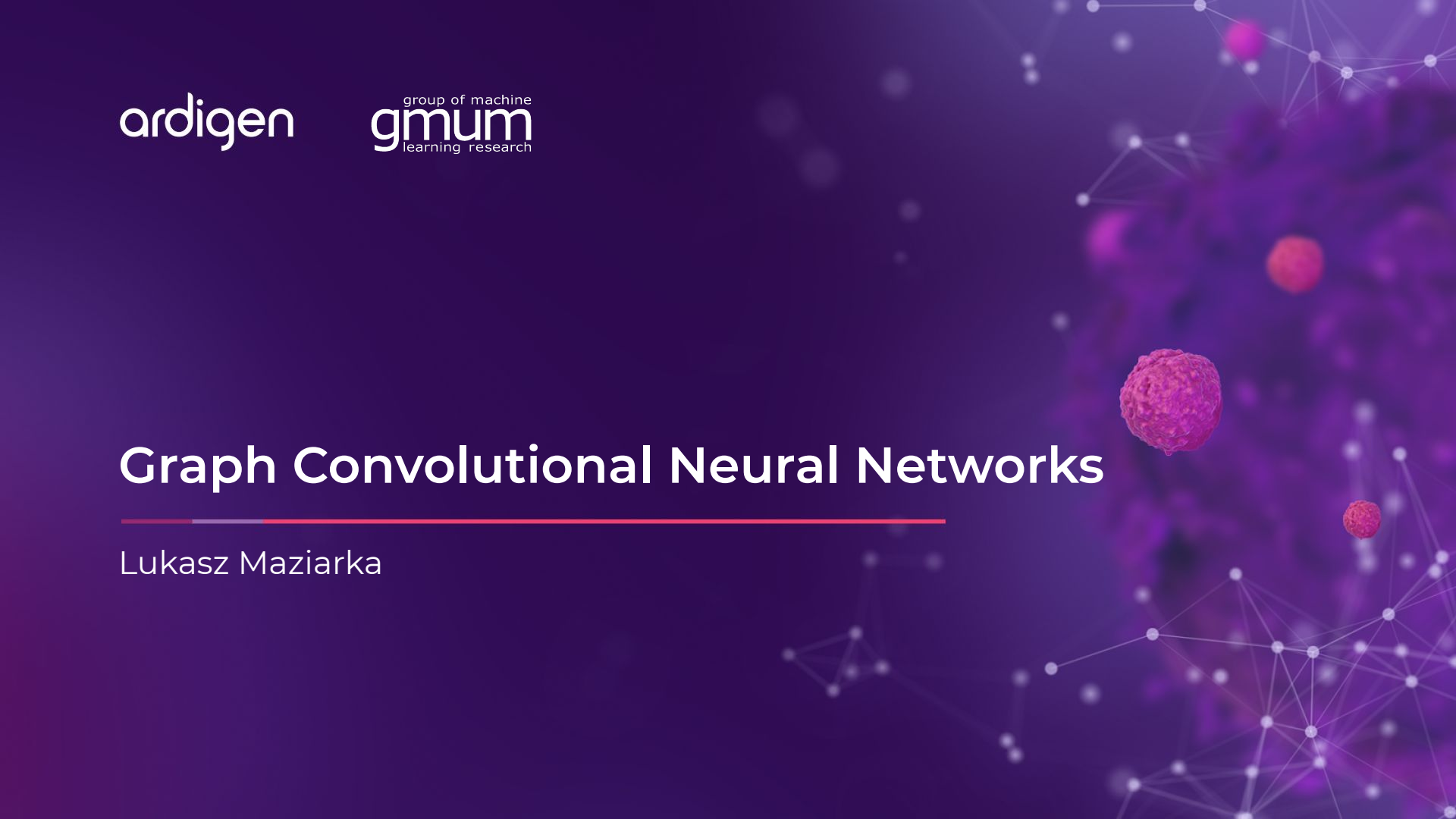
ardigen

group of machine gmum learning research

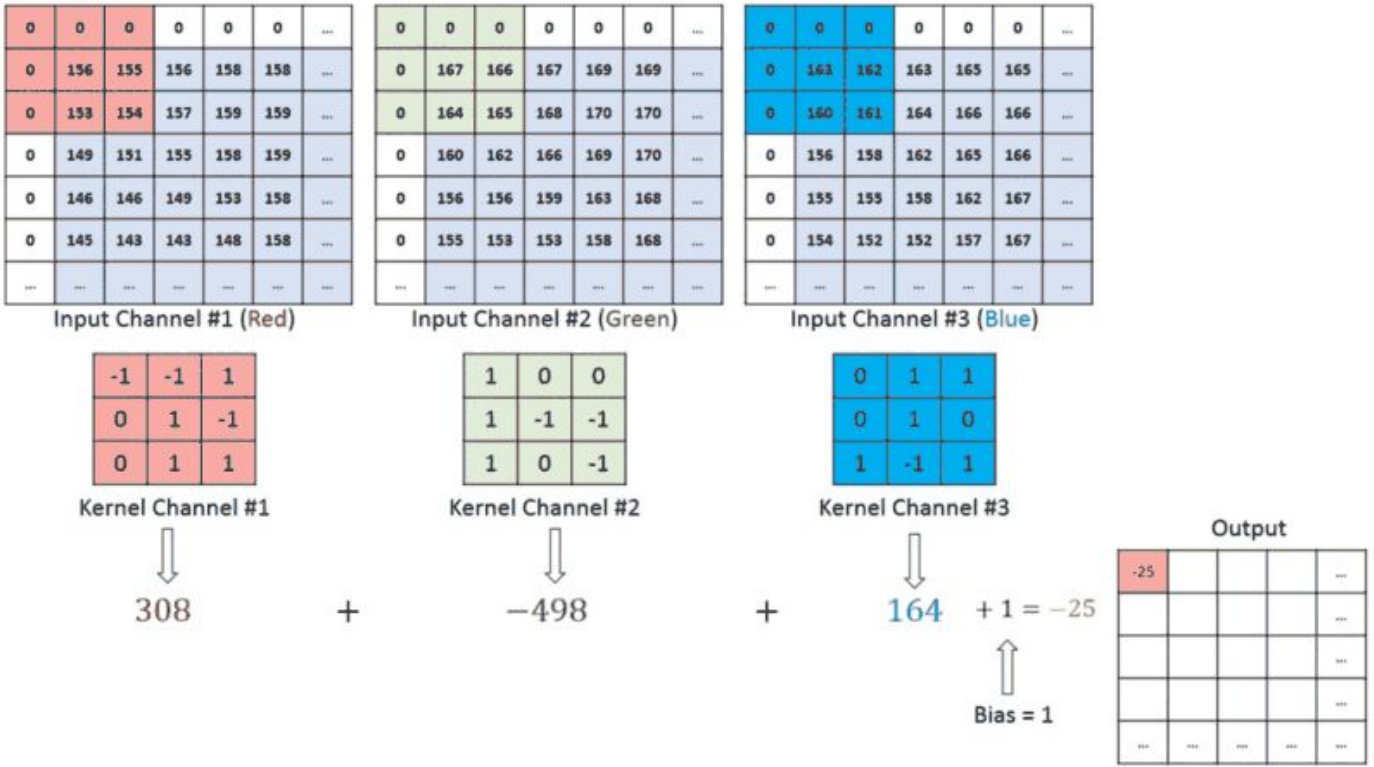
Graph Convolutional Neural Networks

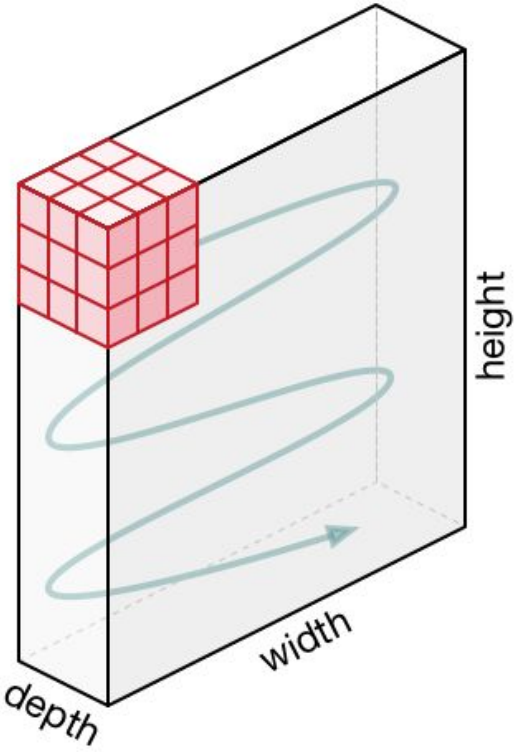
Lukasz Maziarka



Convolutional Neural Networks

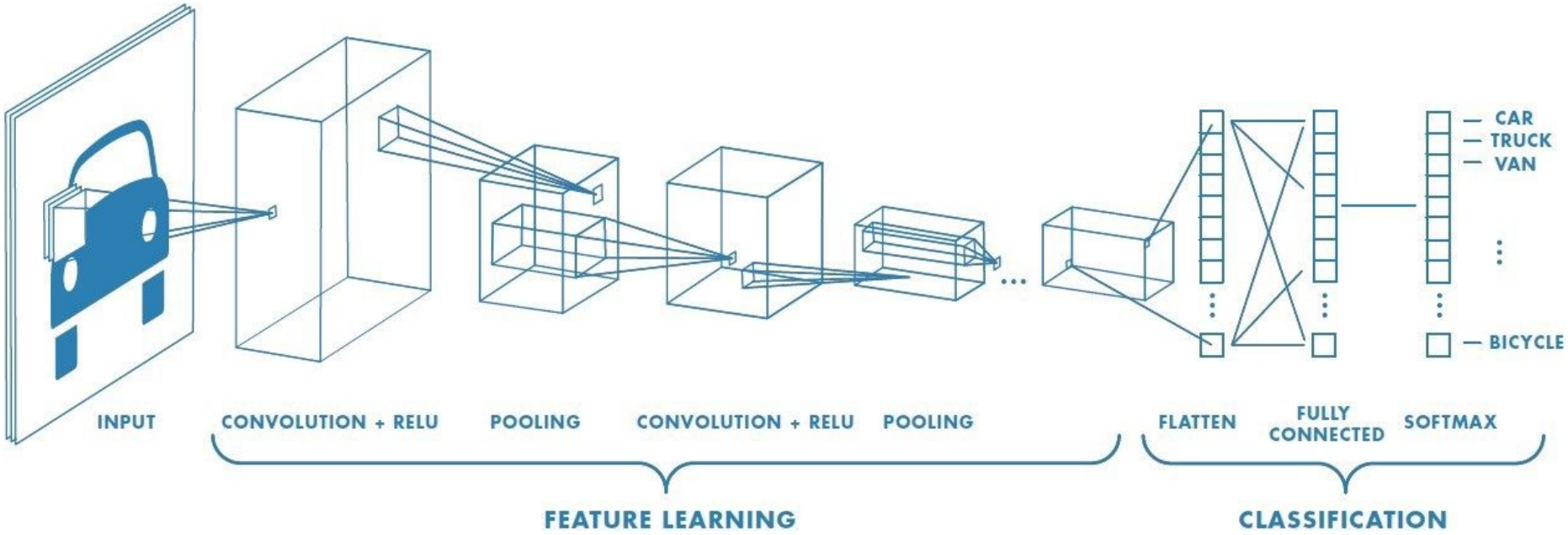
Convolution operation





Source: [towardsdatascience]

Convolutional Neural Network



Source: [towardsdatascience]

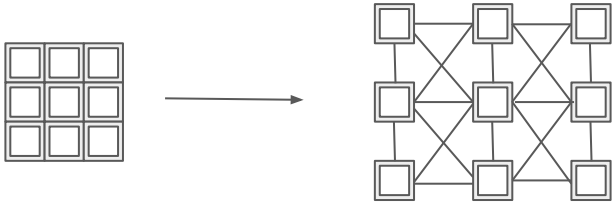
Convolutional Neural Network

Pros of trainable convolutional filters & weight sharing

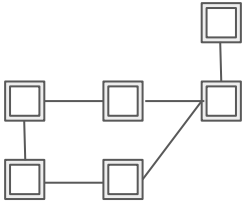
- Custom filters (representation suitable for the given dataset)
- Equivariance for affine transform
- Much less parameters to train
  - faster training
  - less training data needed
  - less prone to overfitting
- CNNs were inspired by how the visual cortex in a human brain works. It can detect simple patterns in small receptive fields

Graph Convolutional Neural Networks

Graph Convolution



Images can be represented as a graph

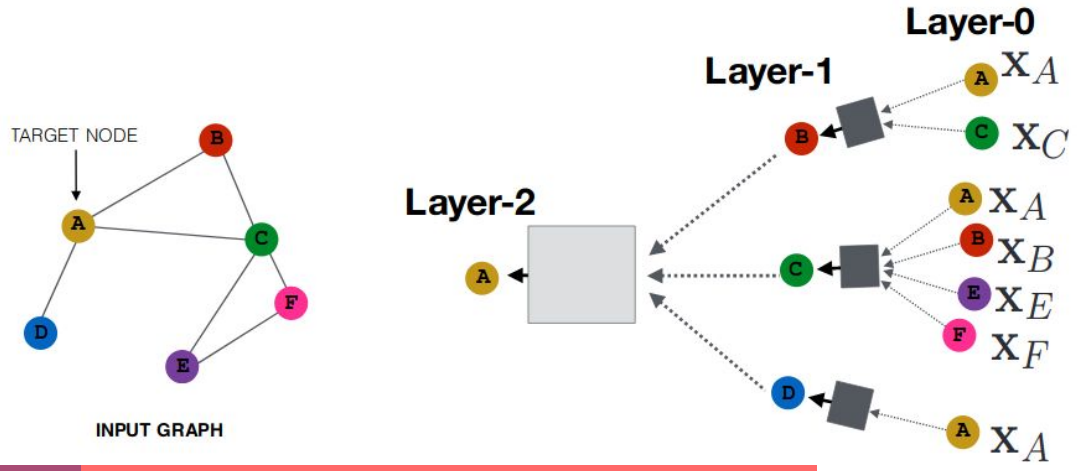


However usually graphs do not have such nice structure

Graph Convolutional Layer

Neighborhood Aggregation

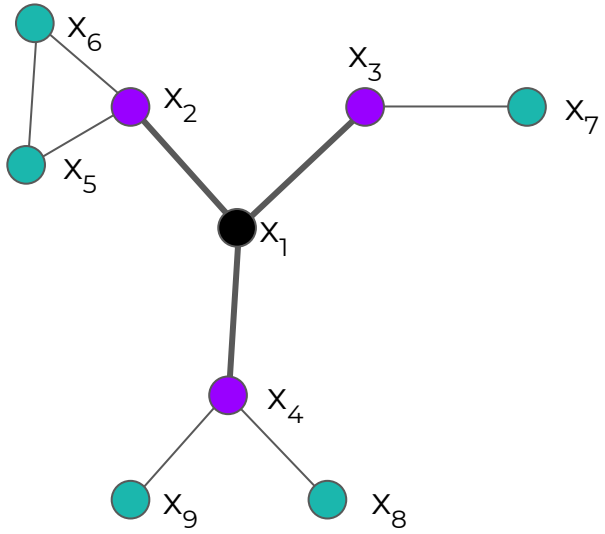
- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- "layer-0" embedding of node $\mathbf{u}$ is its input feature, i.e. $\mathbf{x}_u$.



Source: AAAI workshop

Graph Convolutional Layer

Learning new nodes representation



Calculating new node vectors

$$h_i^{(k)} = ReLU(W^{(k)} x_i)$$

$$x_i^{(k+1)} = \frac{1}{|j : (i,j) \in E|} \sum_{j:\ (i,j)\in E} h_j^{(k)}$$

Calculating graph representation

$$x_{graph} = \frac{1}{|V|} \sum_{v \in V} x_v^{(N)}$$

Message Passing Neural Network

group of machine gmum learning research ardigen

1. Message passing phase

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

2. Readout phase

$$\hat{y} = R(\{h_v^T \mid v \in G\}).$$

The message functions $\mathbf{M_t}$, vertex update functions $\mathbf{U_t}$ and readout function **R** are all learned differentiable functions (Neural networks).

R operates on the set of node states and must be invariant to permutations of the node states in order for the MPNN to be invariant to graph isomorphism.

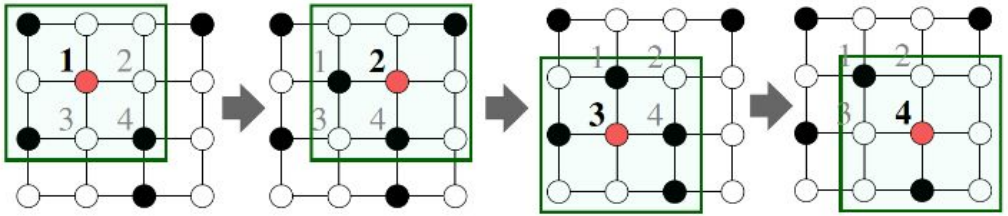
Source: Gilmer et al

Graph Convolutional Layer

group of machine gmum learning research ardigen

Neighborhood "Convolutions"

- Neighborhood aggregation can be viewed as a center-surround filter.



- Mathematically related to spectral graph convolutions (see Bronstein et al., 2017)

Source: AAAI workshop

Input atom representations

Indices	Description
$0-10$	Atomic identity as a one-hot vector of B, N, C, O, F, P, S, Cl, Br, I, other
$11-16$	Number of heavy neighbors as one-hot vector of 0, 1, 2, 3, 4, 5
$17-21$	Number of hydrogen atoms as one-hot vector of 0, 1, 2, 3, 4
22	Formal charge
23	Is in a ring
24	Is aromatic

Source: Coley et al

Possible extensions

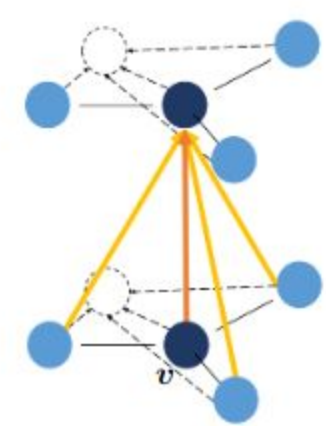
Add edge features

group of machine gmum learning research ardigen

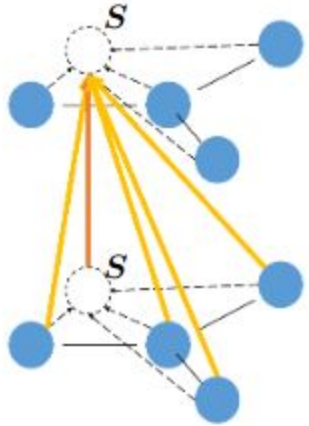
Indices	Description
0-3	Bond order as one-hot vector of 1, 1.5, 2, 3
4	Is aromatic
5	Is conjugated
6	Is in a ring
7	Placeholder, is a bond

Source: Coley et al

Super node



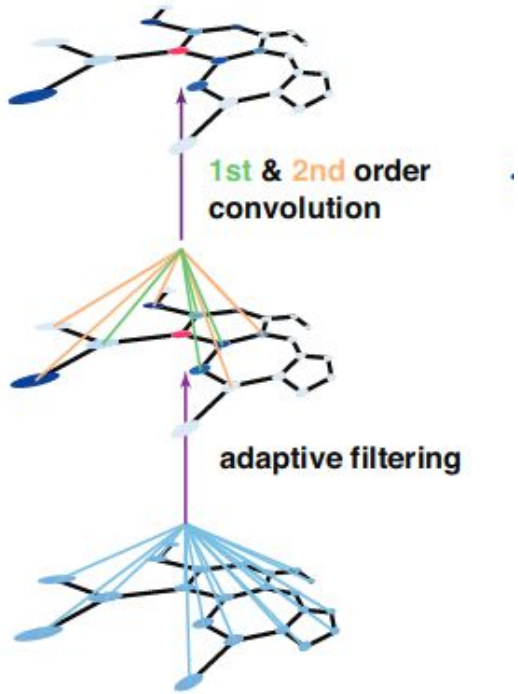
GraphConv



GraphConv(Super Node)

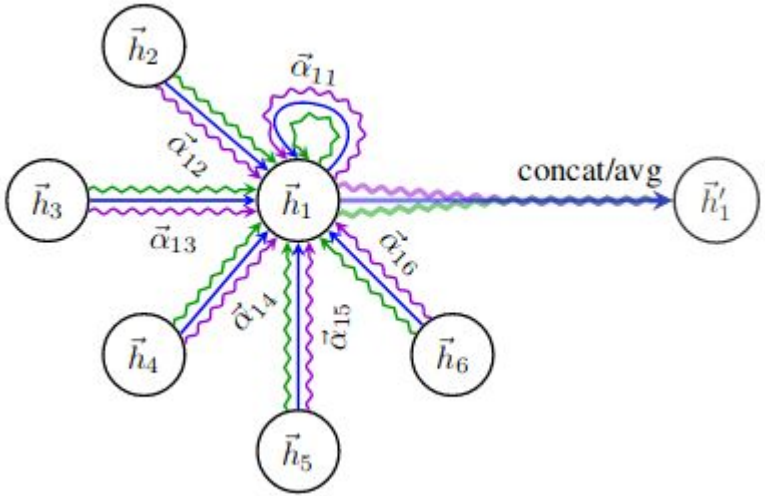
Source: Li et al

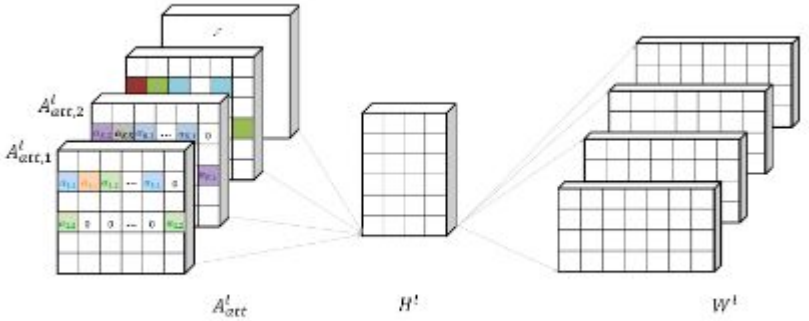
Add higher order convolutions



Source: Zhou et al

Add attention





$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

Source: Veclickovic et al and Shang et al

Self attention & distances

Molecule Transformer

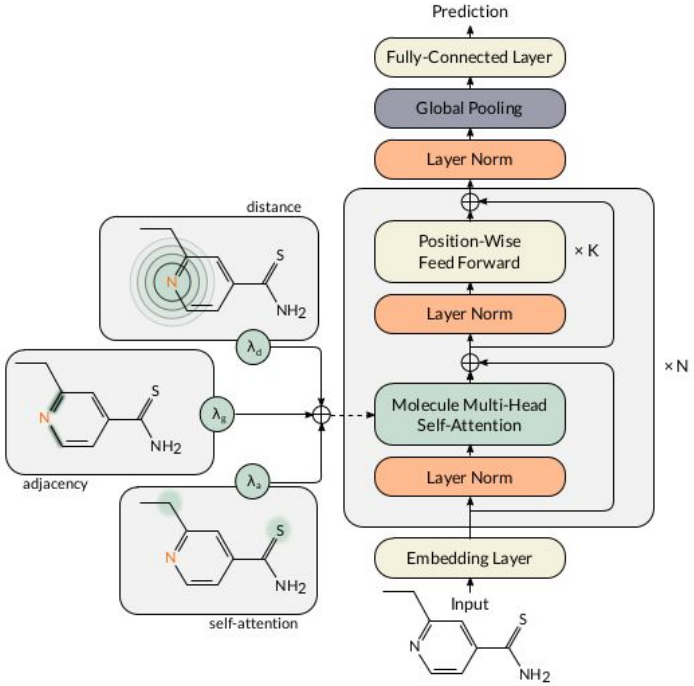
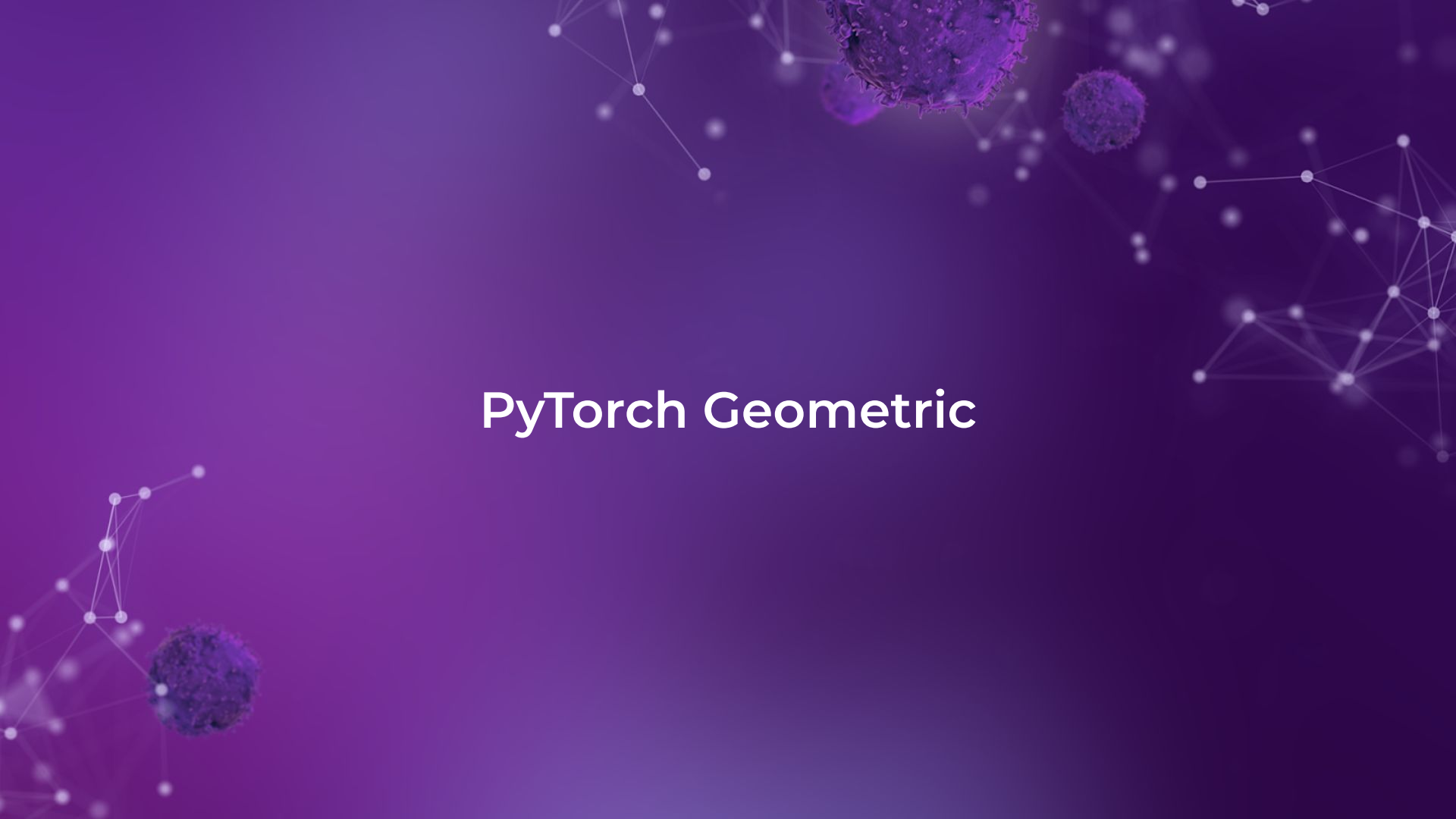


Figure 1: Molecule Transformer architecture. In the first layer we embed each atom using one-hot encoding and atomic features. The main innovation is the Molecule Multi-Head Self-Attention block that augments the self-attention module with distance, and graph structure of the molecule.

Source: Maziarka et al


PyTorch Geometric

PyTorch Geometric

group of machine gmum learning research ardigen

PyTorch Geometric (PyG) is a geometric deep learning extension library for PyTorch.

- Github
- Documentation
- Tutorial
- Paper

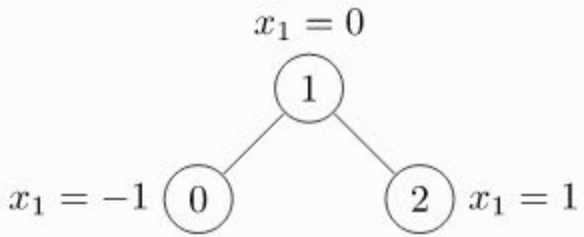
Representing Graphs

group of machine gmum learning research ardigen

```
import torch
from torch_geometric.data import Data

edge_index = torch.tensor([[0, 1, 1, 2],
                           [1, 0, 2, 1]], dtype=torch.long)
x = torch.tensor([[-1], [0], [1]], dtype=torch.float)

data = Data(x=x, edge_index=edge_index)
>>> Data(edge_index=[2, 4], x=[3, 1])
```



Source: Geometric Tutorial

Working with data

- Merging them into batches
- Create Datasets
- Create Data Loaders
- Making Data Transformations

Many included datasets

For both graph and node classification

- KarateClub
- TUDataset
- Planetoid
- CoraFull
- Coauthor
- Amazon
- PPI
- Reddit
- QM7b
- QM9
- Entities
- GEDDataset
- MNISTSuperpixels
- FAUST
- PascalVOCKeypoints
- DynamicFAUST
- ShapeNet
- ModelNet
- CoMa
- SHREC2016
- TOSCA
- PCPNetDataset
- S3DIS
- GeometricShapes
- BitcoinOTC
- ICEWS18
- GDELT
- DBP15K
- WILLOWObjectClass
- PascalPF

Creating Message Passing Networks

“MessagePassing” is a base class in Torch Geometric

class **MessagePassing**(*aggr='add', flow='source_to_target', node_dim=0*) [source]

Base class for creating message passing layers

$$\mathbf{x}_i' = \gamma_{\mathbf{\Theta}} \left(\mathbf{x}_i, \square_{j \in \mathcal{N}(i)} \, \phi_{\mathbf{\Theta}} \left(\mathbf{x}_i, \mathbf{x}_j, \mathbf{e}_{i,j} \right) \right),$$

where $\square$ denotes a differentiable, permutation invariant function, *e.g.*, sum, mean or max, and $\gamma_{\mathbf{\Theta}}$ and $\phi_{\mathbf{\Theta}}$ denote differentiable functions such as MLPs. See here for the accompanying tutorial.

Parameters:

- **aggr** (*string, optional*) – The aggregation scheme to use (`"add"`, `"mean"` or `"max"`). (default: `"add"`)
- **flow** (*string, optional*) – The flow direction of message passing (`"source_to_target"` or `"target_to_source"`). (default: `"source_to_target"`)
- **node_dim** (*int, optional*) – The axis along which to propagate. (default: `0`)

Source: Geometric Tutorial

Creating Message Passing Networks

Implementing GCN layer

group of machine learning research gmum ardigen

$$\mathbf{x}_i^{(k)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{deg(j)}} \cdot \left(\mathbf{\Theta} \cdot \mathbf{x}_j^{(k-1)} \right)$$

```python
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add')  # "Add" aggregation.
        self.lin = torch.nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate(edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]

        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]

        # Step 5: Return new node embeddings.
        return aggr_out
```

Source: Geometric Tutorial

And many more implemented layers!

group of machine learning research gmum ardigen

- **SplineConv** from Fey *et al.*: SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels (CVPR 2018)
- **GCNConv** from Kipf and Welling: Semi-Supervised Classification with Graph Convolutional Networks (ICLR 2017)
- **ChebConv** from Defferrard *et al.*: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering (NIPS 2016)
- **NNConv** from Gilmer *et al.*: Neural Message Passing for Quantum Chemistry (ICML 2017)
- **CGConv** from Xie and Grossman: Crystal Graph Convolutional Neural Networks for an Accurate and Interpretable Prediction of Material Properties (Physical Review Letters 120, 2018)
- **ECConv** from Simonovsky and Komodakis: Edge-Conditioned Convolution on Graphs (CVPR 2017)
- **GATConv** from Veličković *et al.*: Graph Attention Networks (ICLR 2018)
- **SAGEConv** from Hamilton *et al.*: Inductive Representation Learning on Large Graphs (NIPS 2017)
- **GraphConv** from, *e.g.*, Morris *et al.*: Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks (AAAI 2019)
- **GatedGraphConv** from Li *et al.*: Gated Graph Sequence Neural Networks (ICLR 2016)
- **GINConv** from Xu *et al.*: How Powerful are Graph Neural Networks? (ICLR 2019)
- **ARMAConv** from Bianchi *et al.*: Graph Neural Networks with Convolutional ARMA Filters (CoRR 2019)
- **SGConv** from Wu *et al.*: Simplifying Graph Convolutional Networks (CoRR 2019)
- **APPNP** from Klicpera *et al.*: Predict then Propagate: Graph Neural Networks meet Personalized PageRank (ICLR 2019)
- **AGNNConv** from Thekumparampil *et al.*: Attention-based Graph Neural Network for Semi-Supervised Learning (CoRR 2017)
- **TAGConv** from Du *et al.*: Topology Adaptive Graph Convolutional Networks (CoRR 2017)
- **RGCNConv** from Schlichtkrull *et al.*: Modeling Relational Data with Graph Convolutional Networks (ESWC 2018)
- **SignedConv** from Derr *et al.*: Signed Graph Convolutional Network (ICDM 2018)
- **DNAConv** from Fey: Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks (ICLR-W 2019)
- **EdgeConv** from Wang *et al.*: Dynamic Graph CNN for Learning on Point Clouds (CoRR, 2018)
- **PointConv** (including **Iterative Farthest Point Sampling**, dynamic graph generation based on **nearest neighbor** or **maximum distance**, and **k-NN interpolation** for upsampling) from Qi *et al.*: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation (CVPR 2017) and PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space (NIPS 2017)
- **XConv** from Li *et al.*: PointCNN: Convolution On X-Transformed Points (official implementation) (NeurIPS 2018)
- **PPFConv** from Deng *et al.*: PPFNet: Global Context Aware Local Features for Robust 3D Point Matching (CVPR 2018)
- **GMMConv** from Monti *et al.*: Geometric Deep Learning on Graphs and Manifolds using Mixture Model CNNs (CVPR 2017)
- **FeaStConv** from Verma *et al.*: FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis (CVPR 2018)
- **HypergraphConv** from Bai *et al.*: Hypergraph Convolution and Hypergraph Attention (CoRR 2019)
- A **MetaLayer** for building any kind of graph network similar to the TensorFlow Graph Nets library from Battaglia *et al.*: Relational Inductive Biases, Deep Learning, and Graph Networks (CoRR 2018)
- **GlobalAttention** from Li *et al.*: Gated Graph Sequence Neural Networks (ICLR 2016)
- **Set2Set** from Vinyals *et al.*: Order Matters: Sequence to Sequence for Sets (ICLR 2016)
- **Sort Pool** from Zhang *et al.*: An End-to-End Deep Learning Architecture for Graph Classification (AAAI 2018)
- **Dense Differentiable Pooling** from Ying *et al.*: Hierarchical Graph Representation Learning with Differentiable Pooling (NeurIPS 2018)
- **Graclus Pooling** from Dhillon *et al.*: Weighted Graph Cuts without Eigenvectors: A Multilevel Approach (PAMI 2007)
- **Voxel Grid Pooling** from, *e.g.*, Simonovsky and Komodakis: Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs (CVPR 2017)
- **Top-K Pooling** from Gao and Ji: Graph U-Nets (ICML 2019), Cangea *et al.*: Towards Sparse Hierarchical Graph Classifiers (NeurIPS-W 2018) and Knyazev *et al.*: Understanding Attention and Generalization in Graph Neural Networks (ICLR-W 2019)
- **SAG Pooling** from Lee *et al.*: Self-Attention Graph Pooling (ICML 2019) and Knyazev *et al.*: Understanding Attention and Generalization in Graph Neural Networks (ICLR-W 2019)
- **Edge Pooling** from Diehl *et al.*: Towards Graph Pooling by Edge Contraction (ICML-W 2019) and Diehl: Edge Contraction Pooling for Graph Neural Networks (CoRR 2019)
- **Local Degree Profile** from Cai and Wang: A Simple yet Effective Baseline for Non-attribute Graph Classification (CoRR 2018)
- **Jumping Knowledge** from Xu *et al.*: Representation Learning on Graphs with Jumping Knowledge Networks (ICML 2018)
- **Node2Vec** from Grover and Leskovec: node2vec: Scalable Feature Learning for Networks (KDD 2016)
- **Deep Graph Infomax** from Veličković *et al.*: Deep Graph Infomax (ICLR 2019)
- All variants of **Graph Auto-Encoders** from Kipf and Welling: Variational Graph Auto-Encoders (NIPS-W 2016) and Pan *et al.*: Adversarially Regularized Graph Autoencoder for Graph Embedding (IJCAI 2018)
- **RENet** from Jin *et al.*: Recurrent Event Network for Reasoning over Temporal Knowledge Graphs (ICLR-W 2019)
- **GraphUNet** from Gao and Ji: Graph U-Nets (ICML 2019)
- **NeighborSampler** from Hamilton *et al.*: Inductive Representation Learning on Large Graphs (NIPS 2017)

Source: Geometric Github

ardigen

Artificial Intelligence & Bioinformatics
for Precision Medicine

group of machine
gmum
learning research

Thank You!