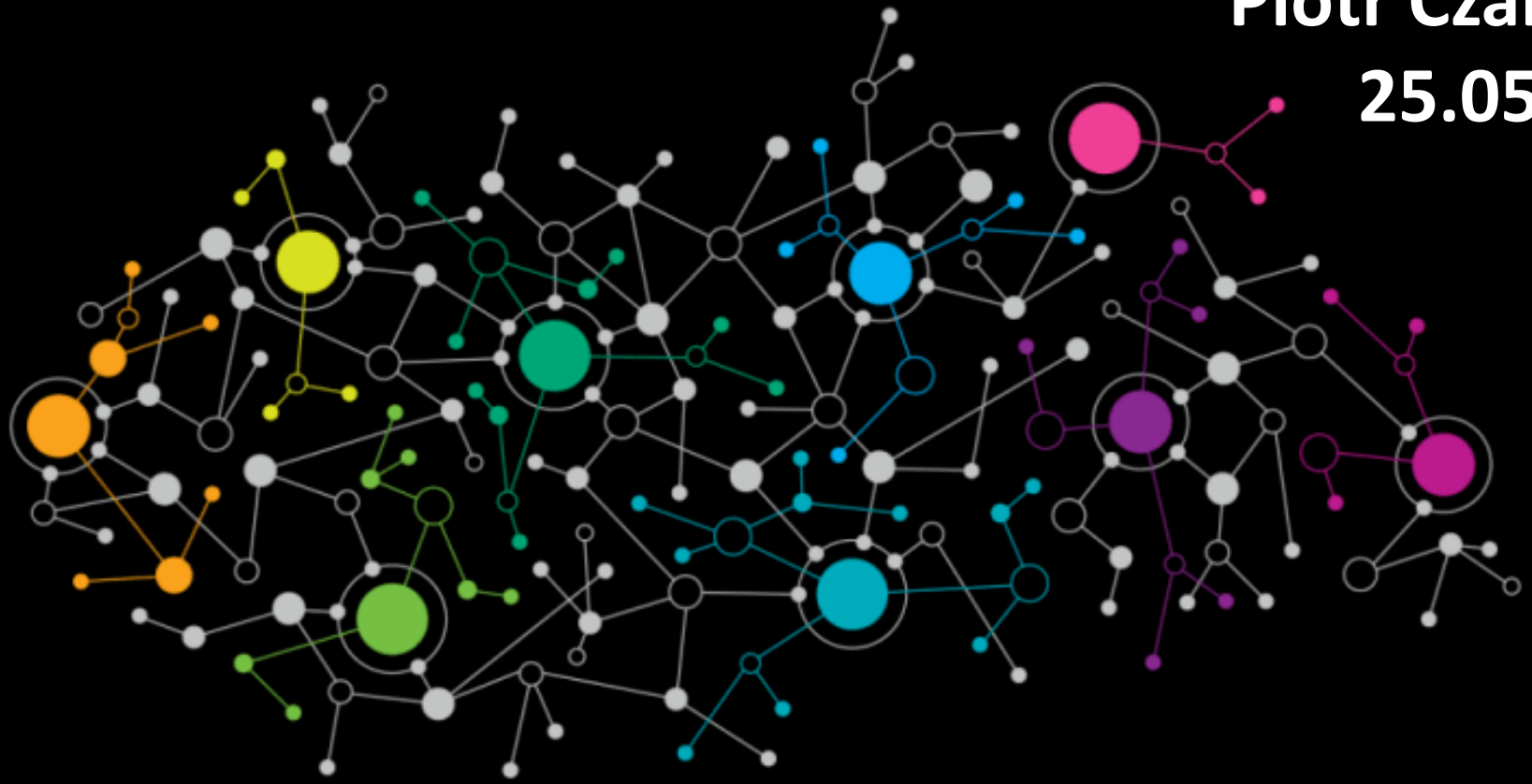
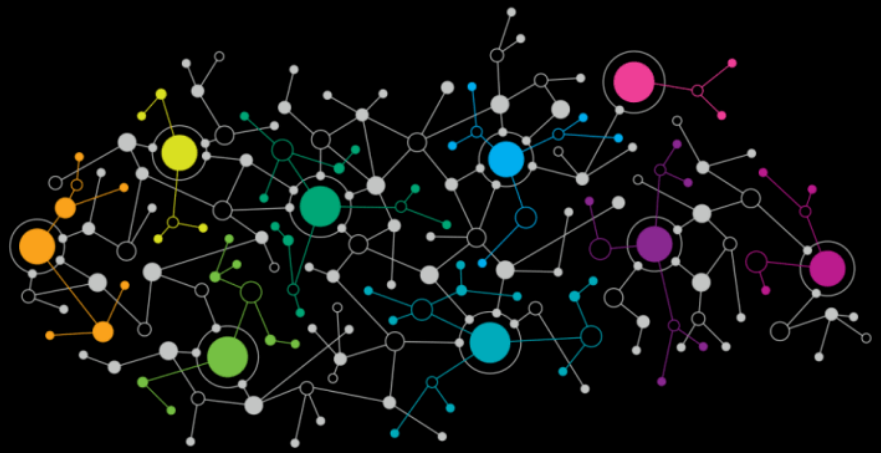
Graph Convolutional Networks

Piotr Czarnecki

25.05.2020

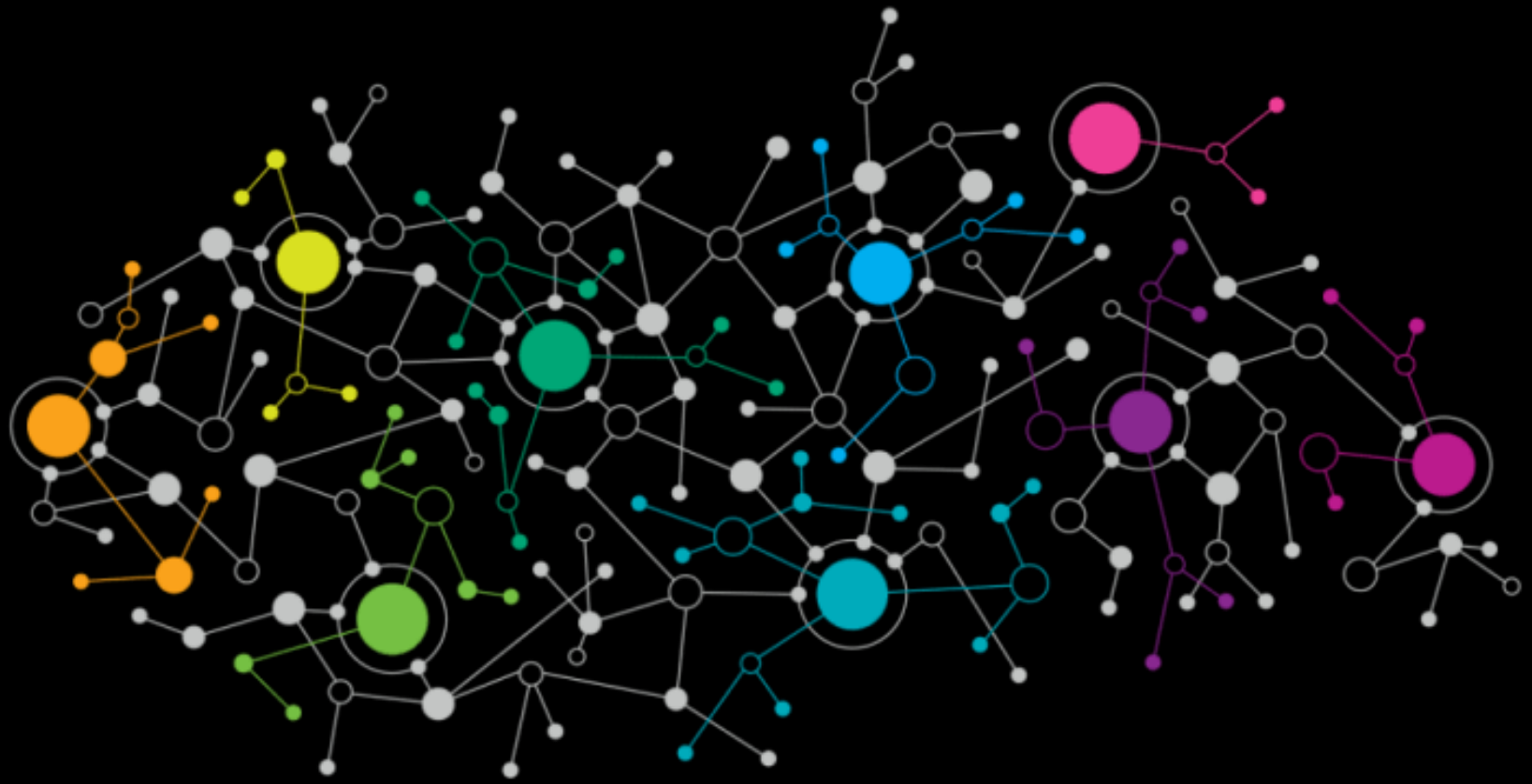




Agenda:

- **Wprowadzenie**
- **Podstawowa grafowa sieć neuronowa**
- **Aplikacje**
- **Dodatki**

Wprowadzenie



Reference

- Presentation based on:
  - http://videolectures.net/solomon_leskovec_deep_learning/
  - http://snap.stanford.edu/proj/embeddings-www/
  - https://arxiv.org/abs/1709.05584
  - https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780
  - A Comprehensive Survey on Graph Neural Networks: https://arxiv.org/abs/1901.00596

Representation Learning on Graphs: Methods and Applications

William L. Hamilton wleif@stanford.edu
Rex Ying rexying@stanford.edu
Jure Leskovec jure@cs.stanford.edu

NIPS Proceedings$^{\beta}$ Books

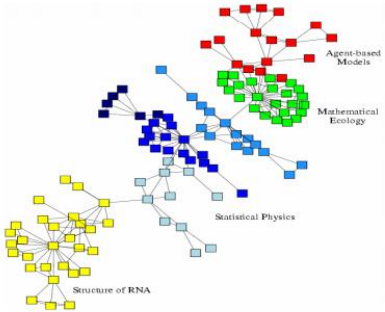
Jure Leskovec

12 Papers

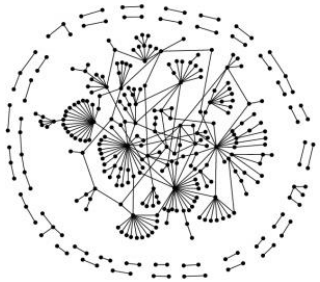
- G2SAT: Learning to Generate SAT Formulas (2019)
- GNNExplainer: Generating Explanations for Graph Neural Networks (2019)
- Hyperbolic Graph Convolutional Neural Networks (2019)
- Dynamic Network Model from Partial Observations (2018)
- Embedding Logical Queries on Knowledge Graphs (2018)
- Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation (2018)
- Hierarchical Graph Representation Learning with Differentiable Pooling (2018)
- Inductive Representation Learning on Large Graphs (2017)
- Confusions over Time: An Interpretable Bayesian Model to Characterize Trends in Decision Making (2016)
- Nonparametric Multi-group Membership Model for Dynamic Networks (2013)
- Learning to Discover Social Circles in Ego Networks (2012)
- On the Convexity of Latent Social Network Inference (2010)

Graph data examples

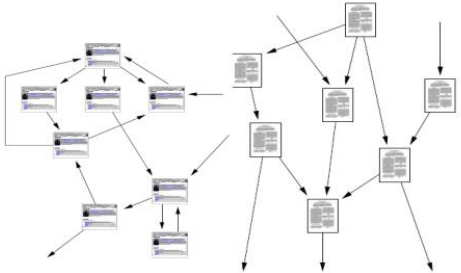
Social networks



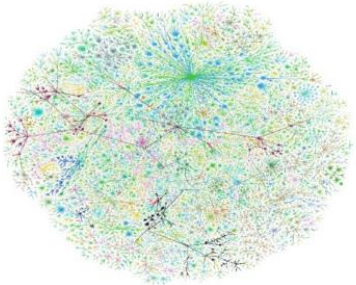
Economic networks



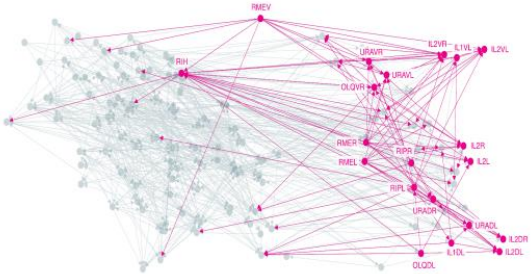
Biomedical networks



Information networks:
Web & citations

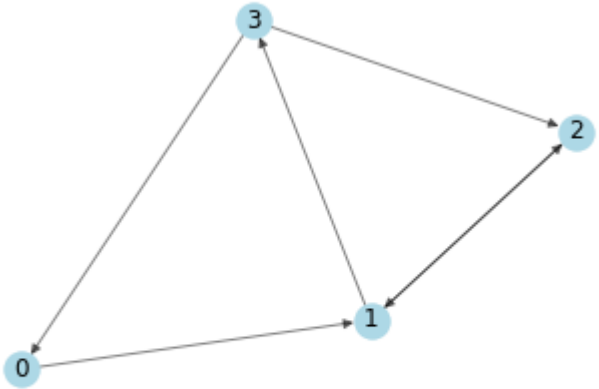


Internet



Networks of neurons

Simple Graph



A simple directed graph.

`numpy` adjacency matrix representation

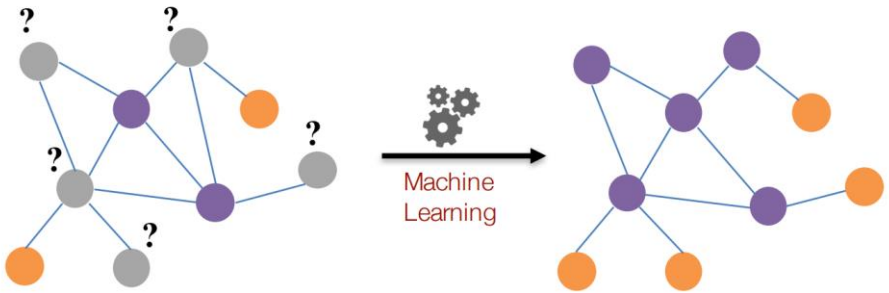
```
A = np.matrix([
    [0, 1, 0, 0],
    [0, 0, 1, 1],
    [0, 1, 0, 0],
    [1, 0, 1, 0]],
    dtype=float
)
```

2 integer features for every node

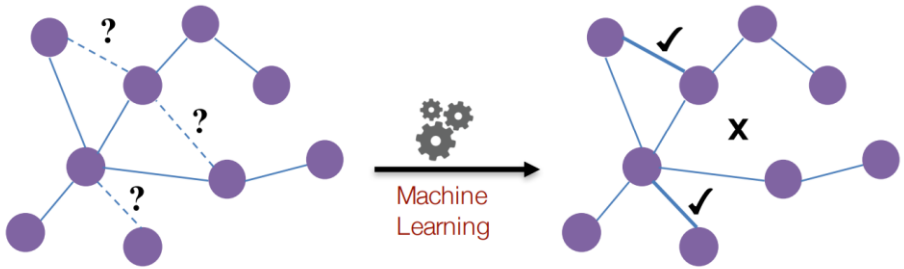
```
matrix([
   [ 0.,  0.],
   [ 1., -1.],
   [ 2., -2.],
   [ 3., -3.]
])
```

Classical ML tasks on graphs

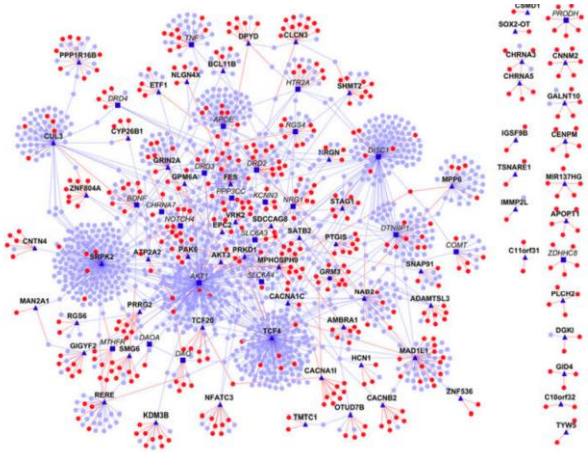
Node classification: Predict a type of a given node



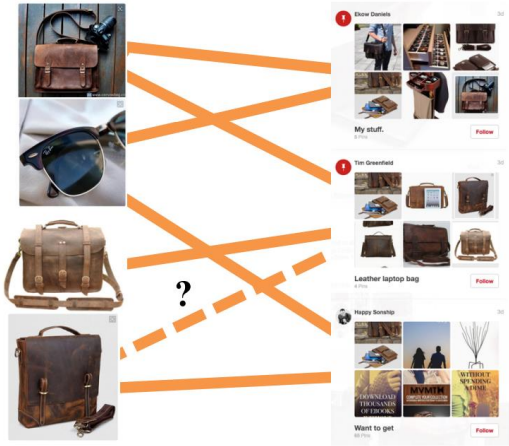
Link prediction: Predict whether two nodes are linked



Classifying the function of proteins in the interactome!



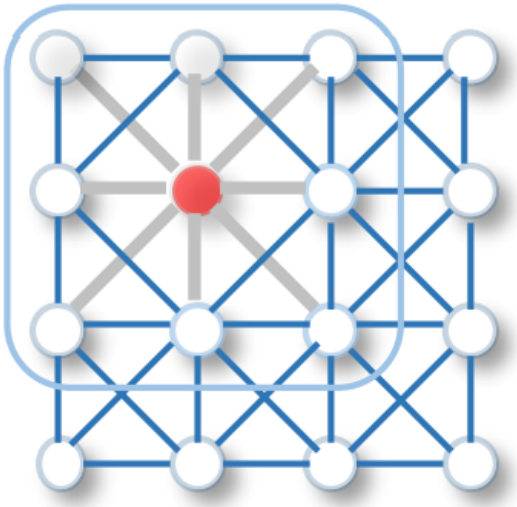
Content recommendation is link prediction!



2D Convolution vs. Graph convolution

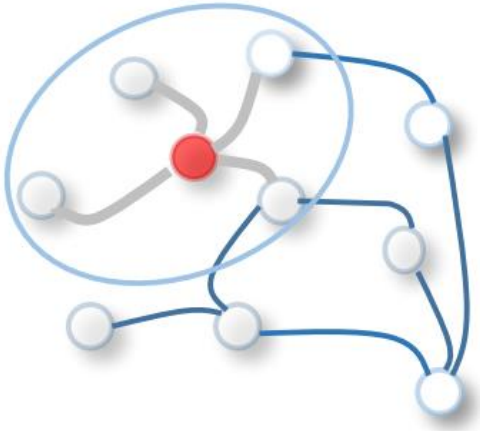
2D Convolution

Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.



Graph Convolution

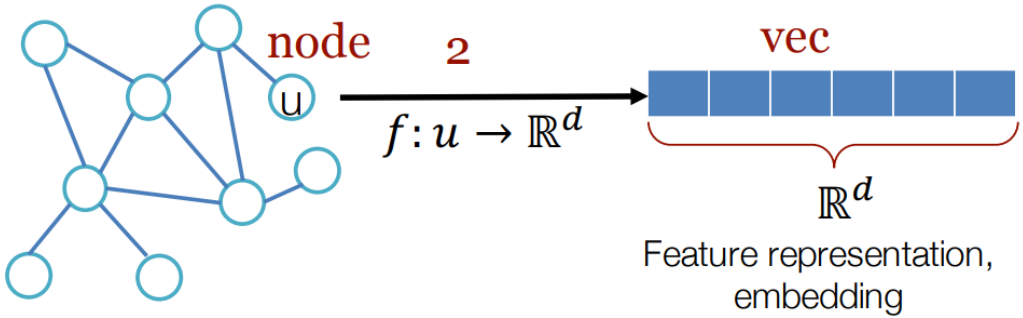
To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size



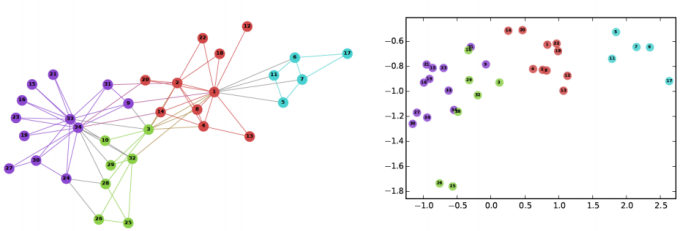
Graph convolution is mathematically related to spectral graph convolutions (Bronstein et al., 2017: Geometric deep learning: going beyond Euclidean data, https://arxiv.org/abs/1611.08097)

Embedding Nodes

Goal is to encode nodes so that similarity in the embedding space approximates similarity in the original network.



Zachary's Karate Club Network:



Input **Output**

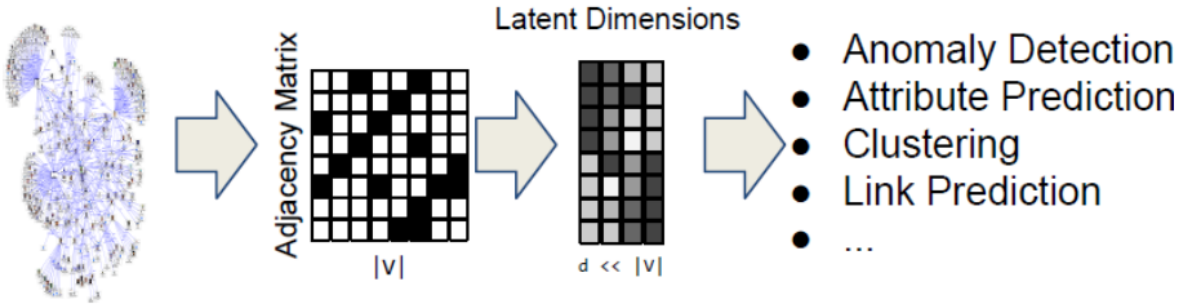
Image from: Perozzi et al. 2014. DeepWalk: Online Learning of Social Representations. *KDD*.

Tutorial:
https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780

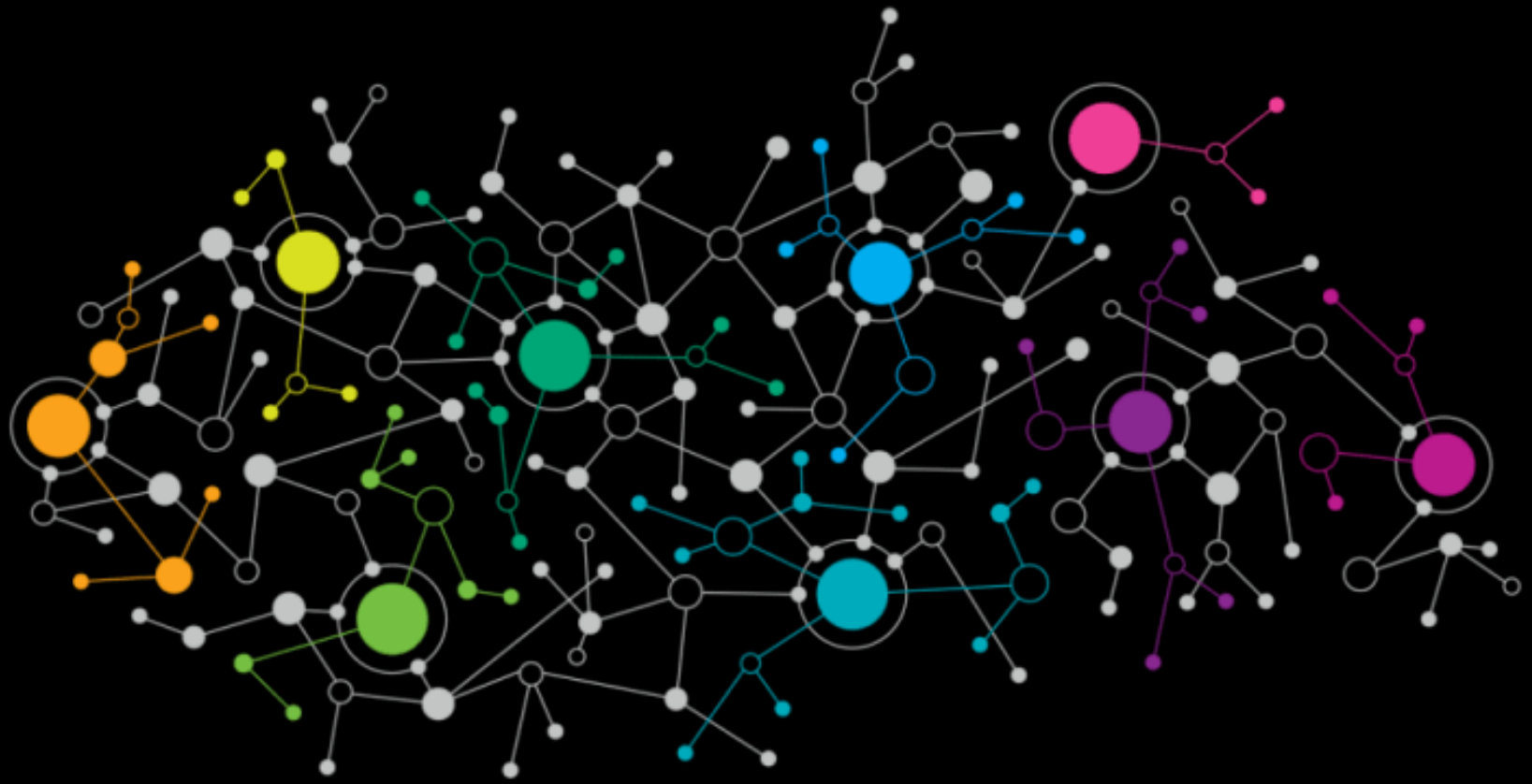
Embedding Nodes

The goal is to map each node into a low-dimensional space

- Distributed representation for nodes
- Similarity between nodes indicates link strength
- Encodes network information and generate node representation

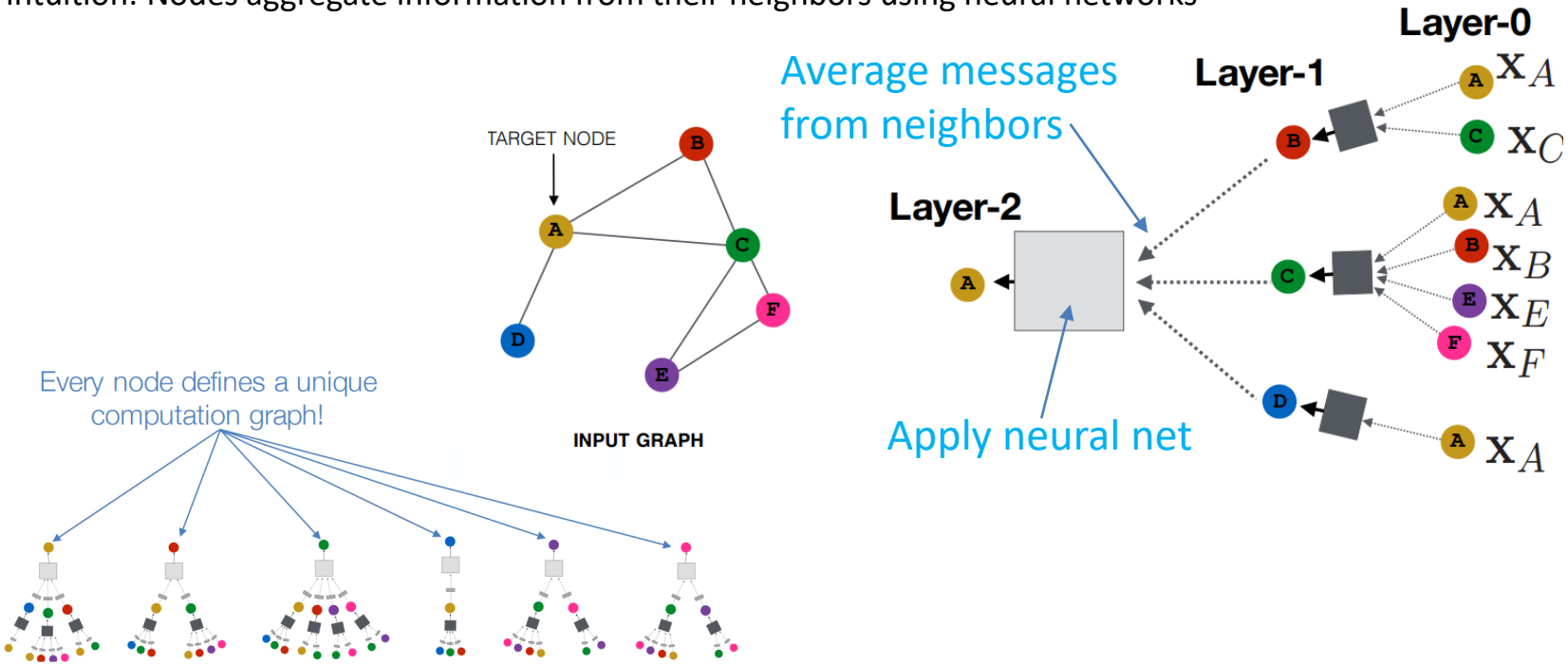


Podstawowa grafowa konwolucyjna sieć neuronowa



Embedding Nodes – basic approach

- Key idea: Generate node embeddings based on local neighborhoods
- Intuition: Nodes aggregate information from their neighbors using neural networks



Embedding Nodes – basic approach

Average neighbor messages and apply a neural network.

Initial "layer 0" embeddings are equal to node features

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

previous layer embedding of v

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k > 0$$

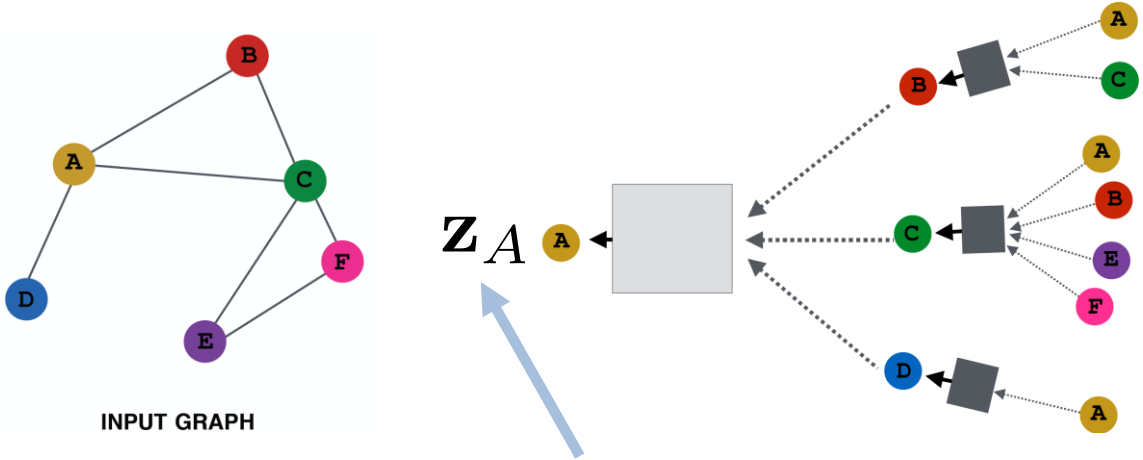
kth layer embedding of v

non-linearity (e.g., ReLU or tanh)

average of neighbor's previous layer embeddings

Training the model

How do we train the model to generate "high-quality" embeddings?



Need to define a loss function on the embeddings, $L(z_u)$!

Training the model

trainable matrices
(i.e., what we learn)

$$\mathbf{h}_v^0 = \mathbf{x}_v$$

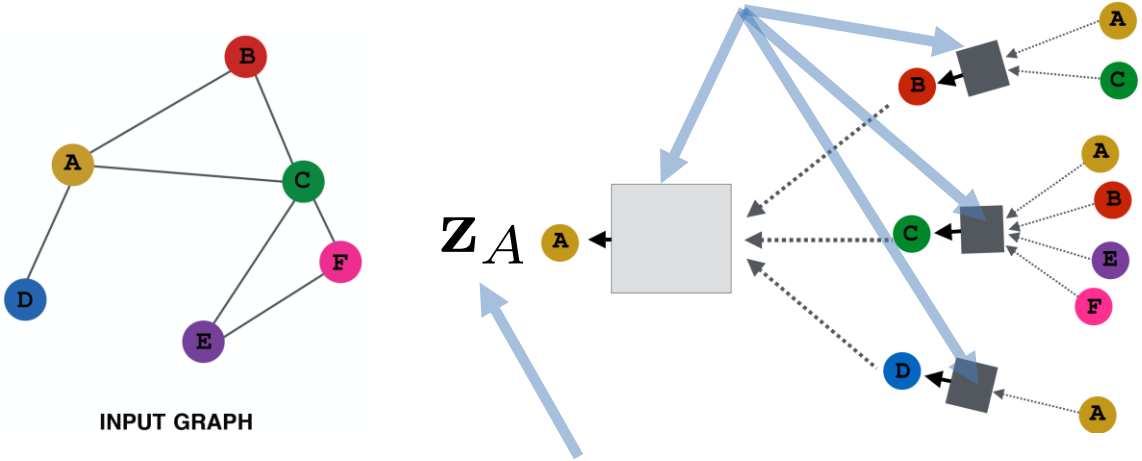
$$\mathbf{h}_v^k = \sigma\left(\boxed{\mathbf{W}_k} \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \boxed{\mathbf{B}_k}\mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, ..., K\}$$

$$\boxed{\mathbf{z}_v = \mathbf{h}_v^K}$$

- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- **We can feed these embeddings into any loss function** and run stochastic gradient descent to train the aggregation parameters.

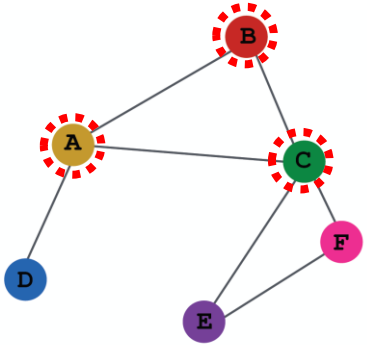
Training the model

1) Define a neighborhood aggregation function.



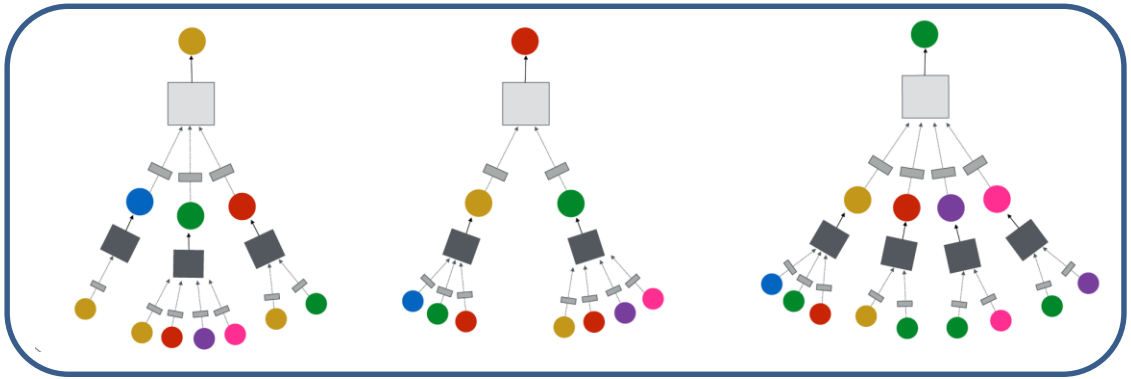
2) Define a loss function on the embeddings, $L(z_u)$

Training the model

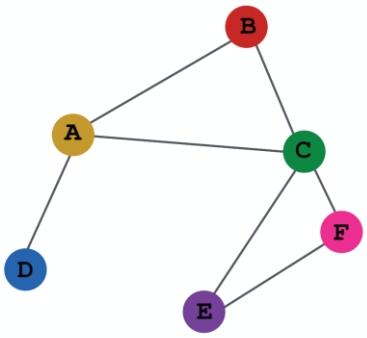


INPUT GRAPH

3) Train on a set of nodes, i.e., a batch of compute graphs



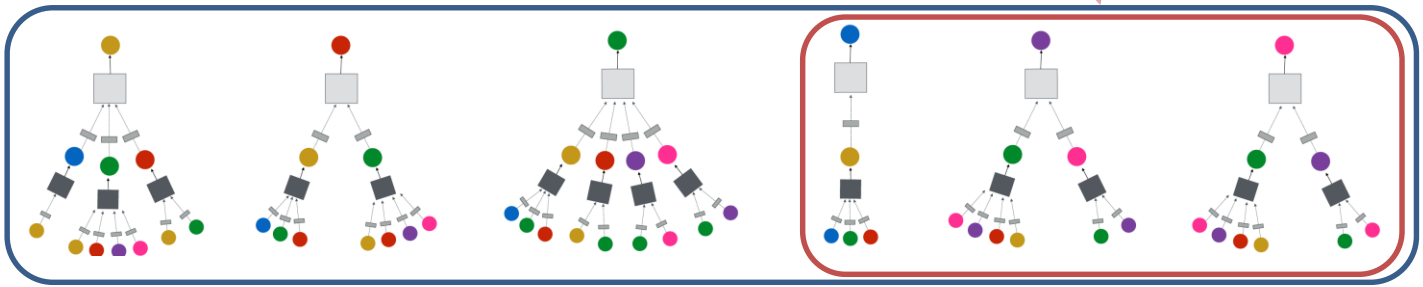
Training the model



INPUT GRAPH

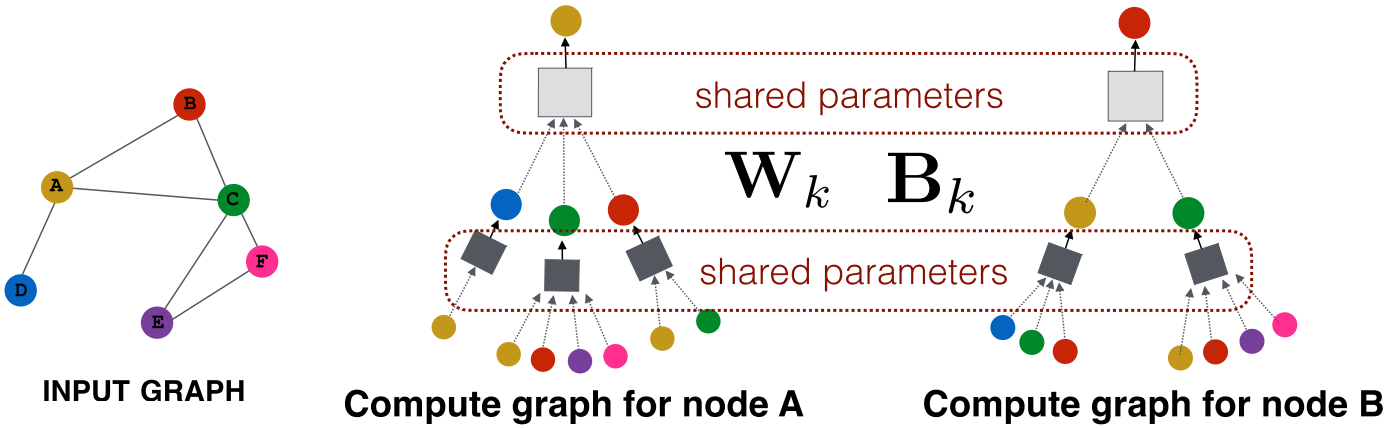
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

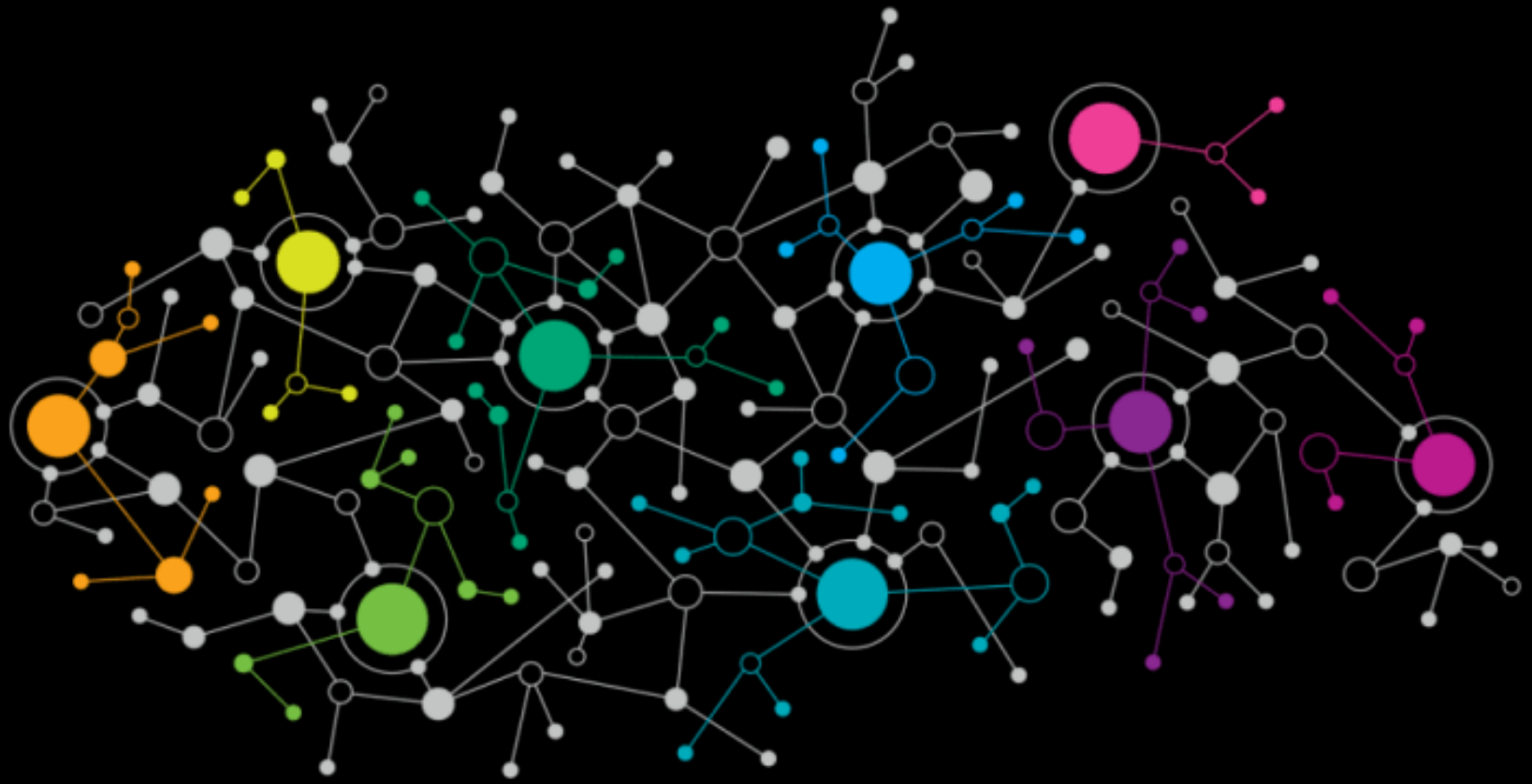


Training the model

- The same aggregation parameters are shared for all nodes.
- The number of model parameters is sublinear in |V| and we can generalize to unseen nodes!



Aplikacje

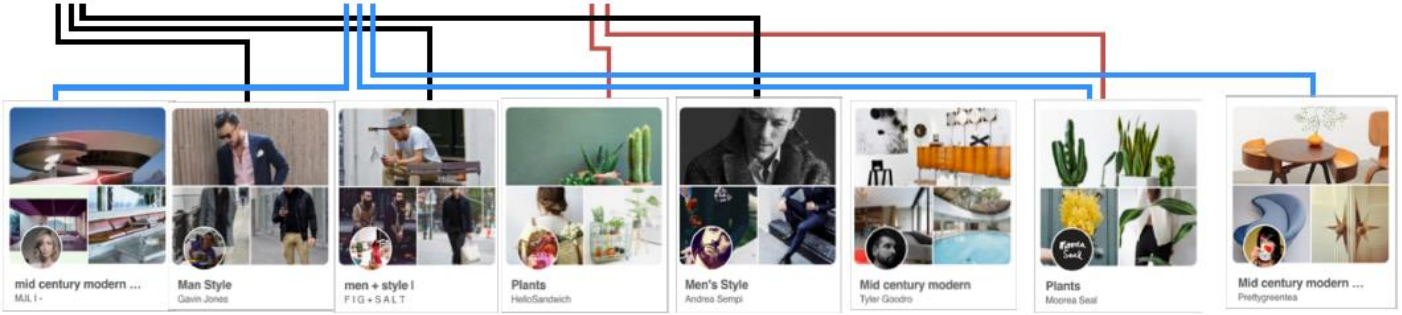


Pinterest

Human curated collection of pins

Pins: Visual bookmarks someone has saved from the internet to a board they've created.

Pin features: Image, text, links



Boards

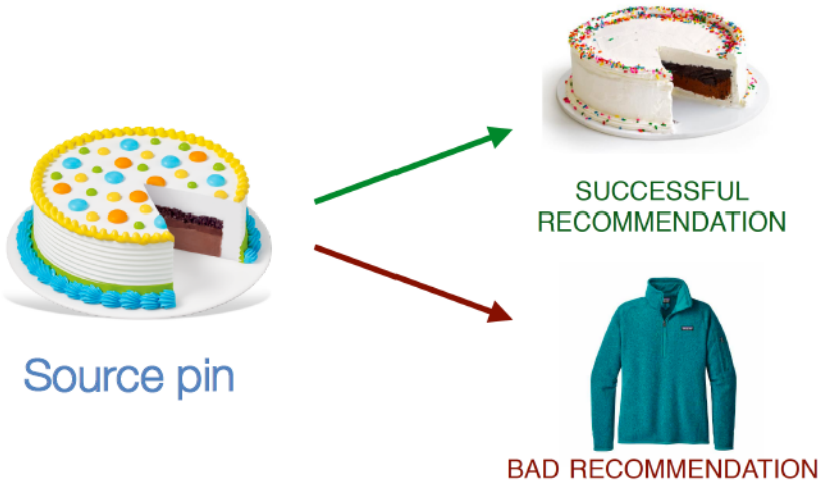
Pinterest

- PinSage graph convolutional network:
  - **Goal:** Generate embeddings for nodes (e.g., Pins/images) in a web-scale Pinterest graph containing billions of objects
  - **Key Idea:** Borrow information from nearby nodes
    - E.g., bed rail Pin might look like a garden fence, but gates and beds are rarely adjacent in the graph

    - Pin embeddings are essential to various tasks like recommendation of Pins, classification, ranking
      - Services like “Related Pins”, “Search”, “Shopping”, “Ads”

Pinterest

Task: Recommend related pins to users.

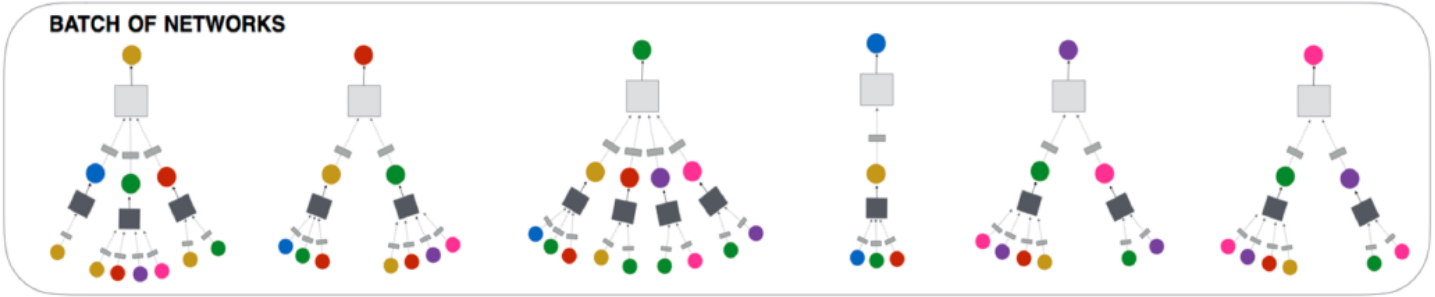


Task: Learn node embeddings z_i such that $d(z_{cake1}, z_{cake2}) < d(z_{cake1}, z_{sweater})$

- Challenges:
  - Massive size: 3 billion nodes, 20 billion edges
  - Heterogeneous data: Rich image and text features

Pinterest

- Leverage inductive capability, and train on individual subgraphs
  - 300 million nodes, 1 billion edges, 1.2 billion pin pairs (Q, X)



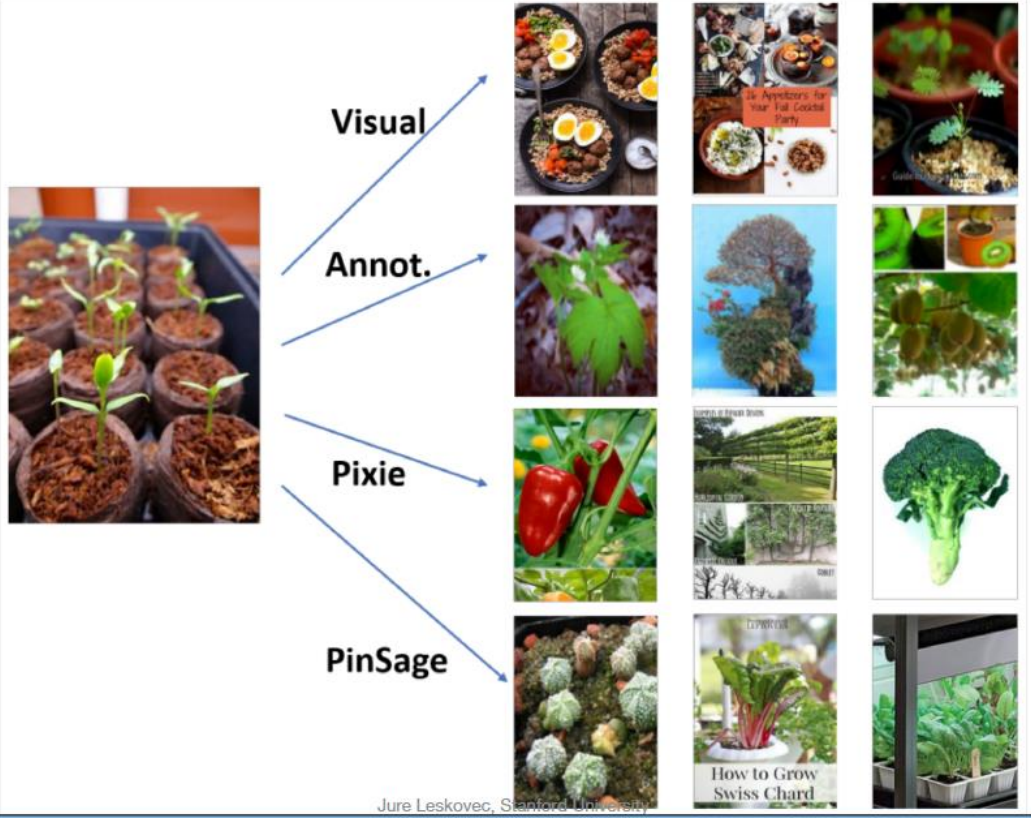
- Large batch size: 2048 per minibatch

Pinterest

Related Pin recommendations

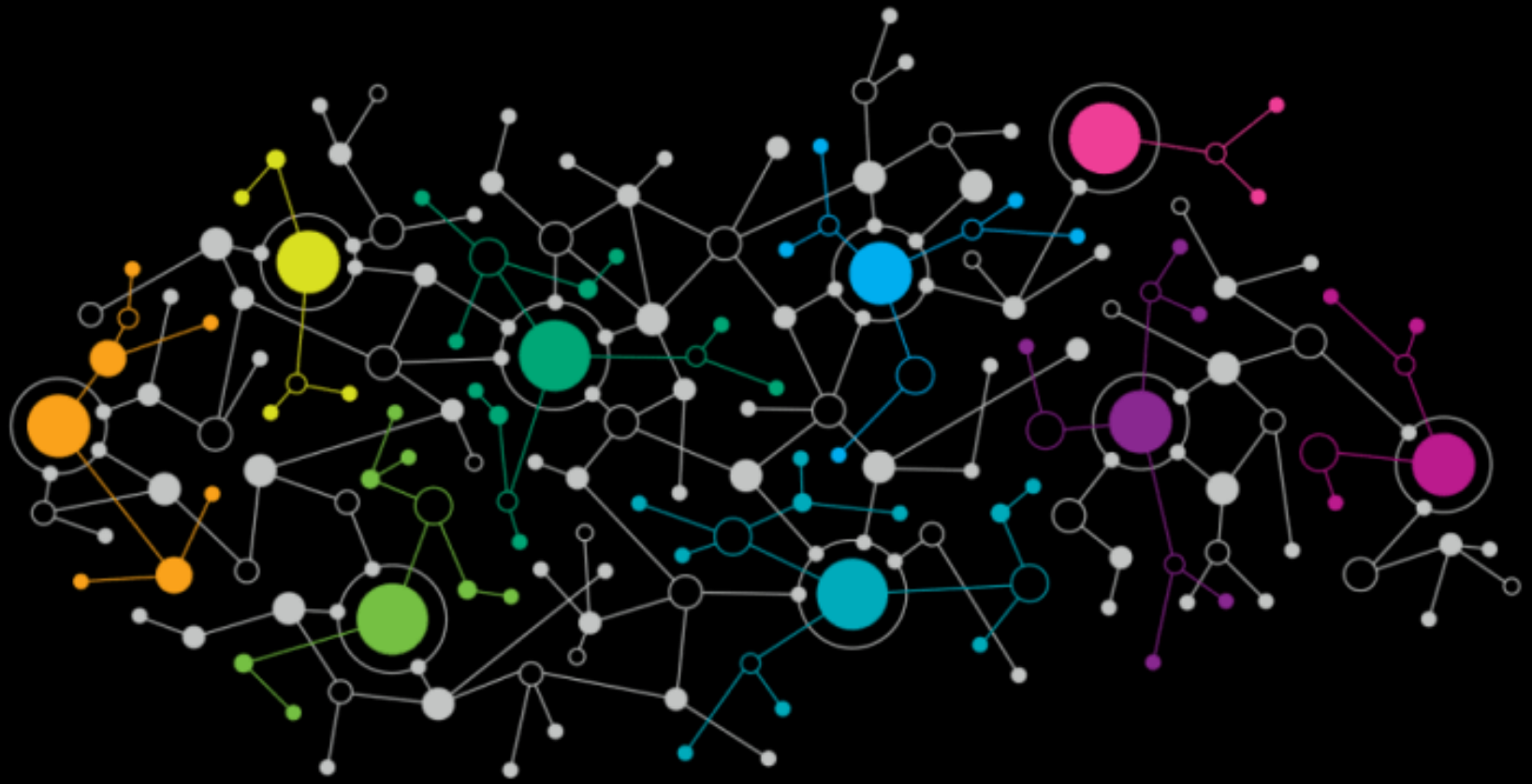
- Given user is looking at pin Q, predict what pin X are they going to save next
- Baselines for comparison
  - Visual: VGG-16 visual features
  - Annotation: Word2Vec model
  - Pixie: Random-walk based algorithm
  - PinSage
- Setup: Embed 3B pins, perform nearest neighbor to generate recommendations

Pinterest



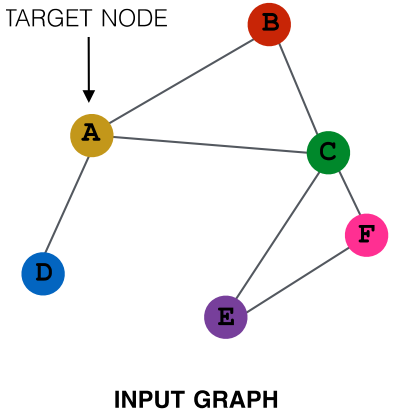
Jure Leskovec, Stanford University

Dodatki

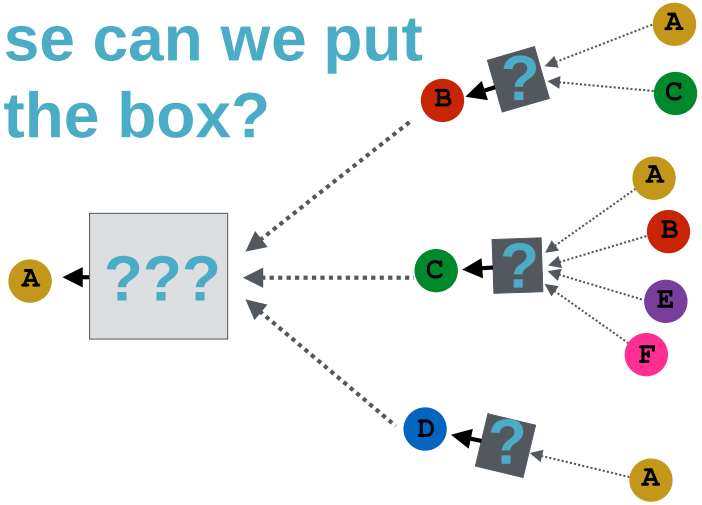


Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate messages



What else can we put in the box?



Graph Convolutional Networks

- Kipf et al.'s **Graph Convolutional Networks (GCNs)** are a slight variation on the neighborhood aggregation idea:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v) \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|N(u)||N(v)|}}\right)$$

GraphSAGE Variants

- **Mean:**

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

- **Pool**
  - Transform neighbor vectors and apply symmetric vector function.

element-wise mean/max

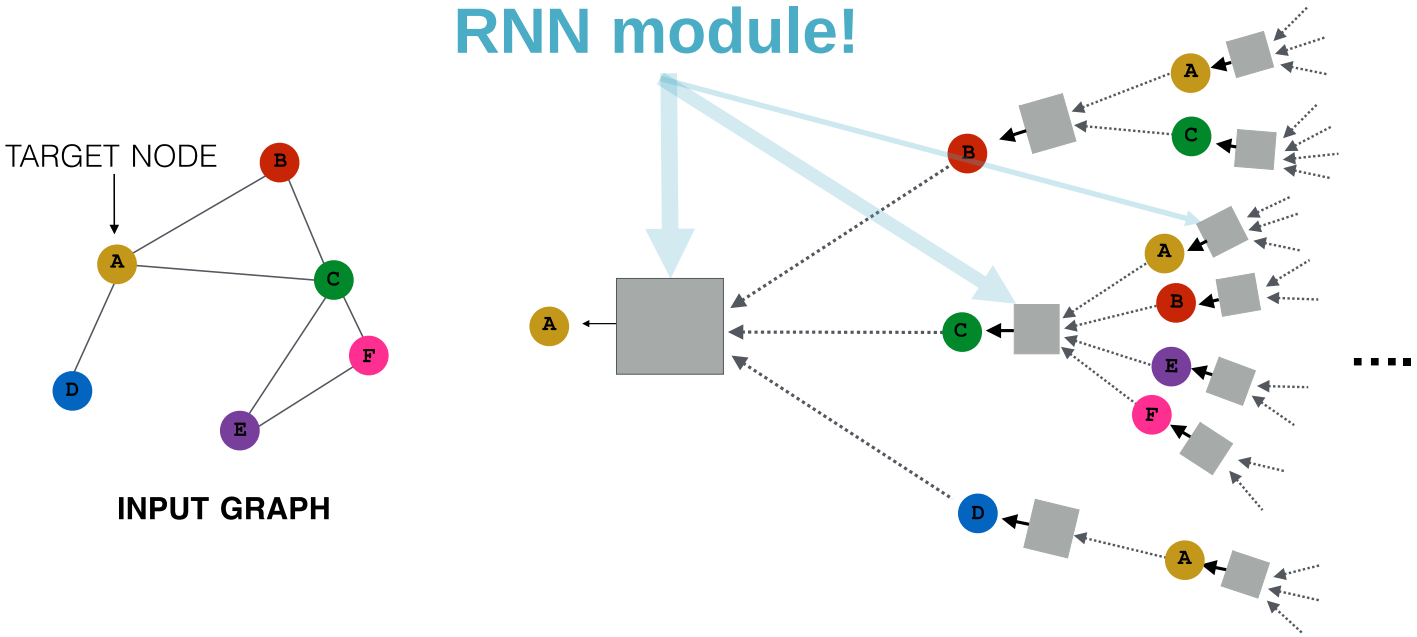
$$\text{AGG} = \boxed{\gamma}\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

- **LSTM:**
  - Apply LSTM to random permutation of neighbors.

$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

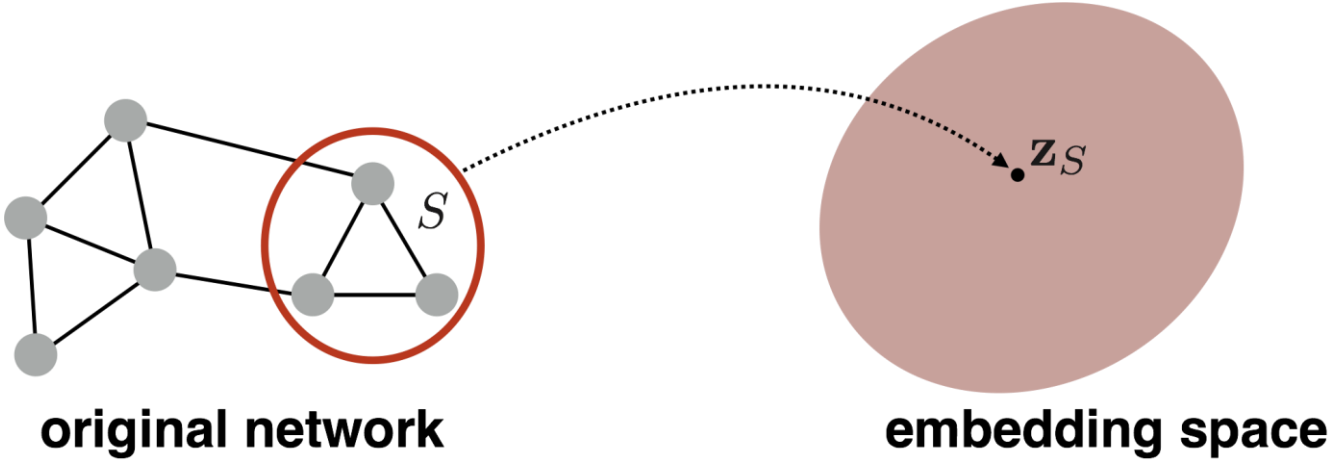
Gated Graph Neural Networks

- **Idea 2:** Recurrent state update.



(Sub)graph Embeddings

- **But what about subgraph embeddings?**



Approach 1

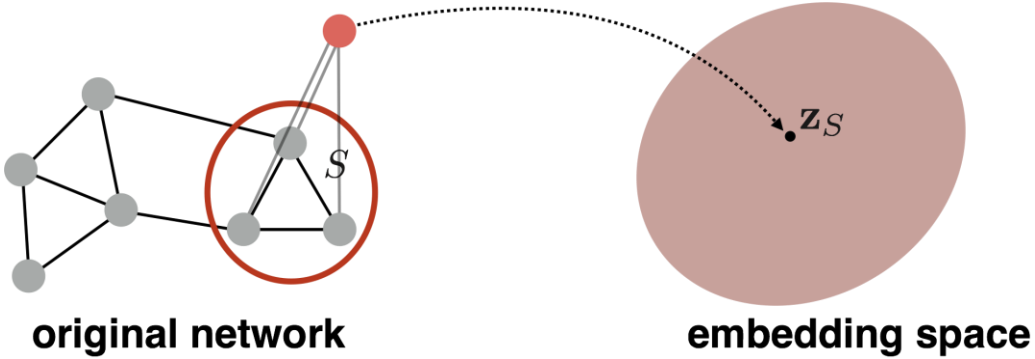
- **Simple idea:** Just sum (or average) the node embeddings in the (sub)graph

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$

- Used by Duvenaud et al., 2016 to classify molecules based on their graph structure.

Approach 2

- **Idea:** Introduce a **"virtual node"** to represent the subgraph and run a standard graph neural network.



- Proposed by Li et al., 2016 as a general technique for subgraph embedding.

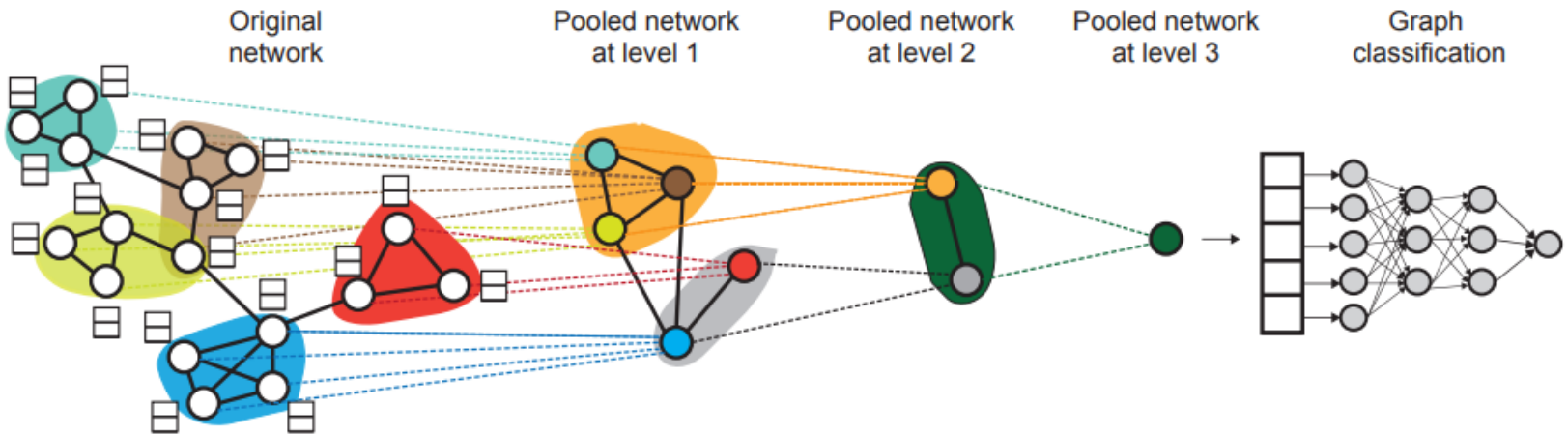
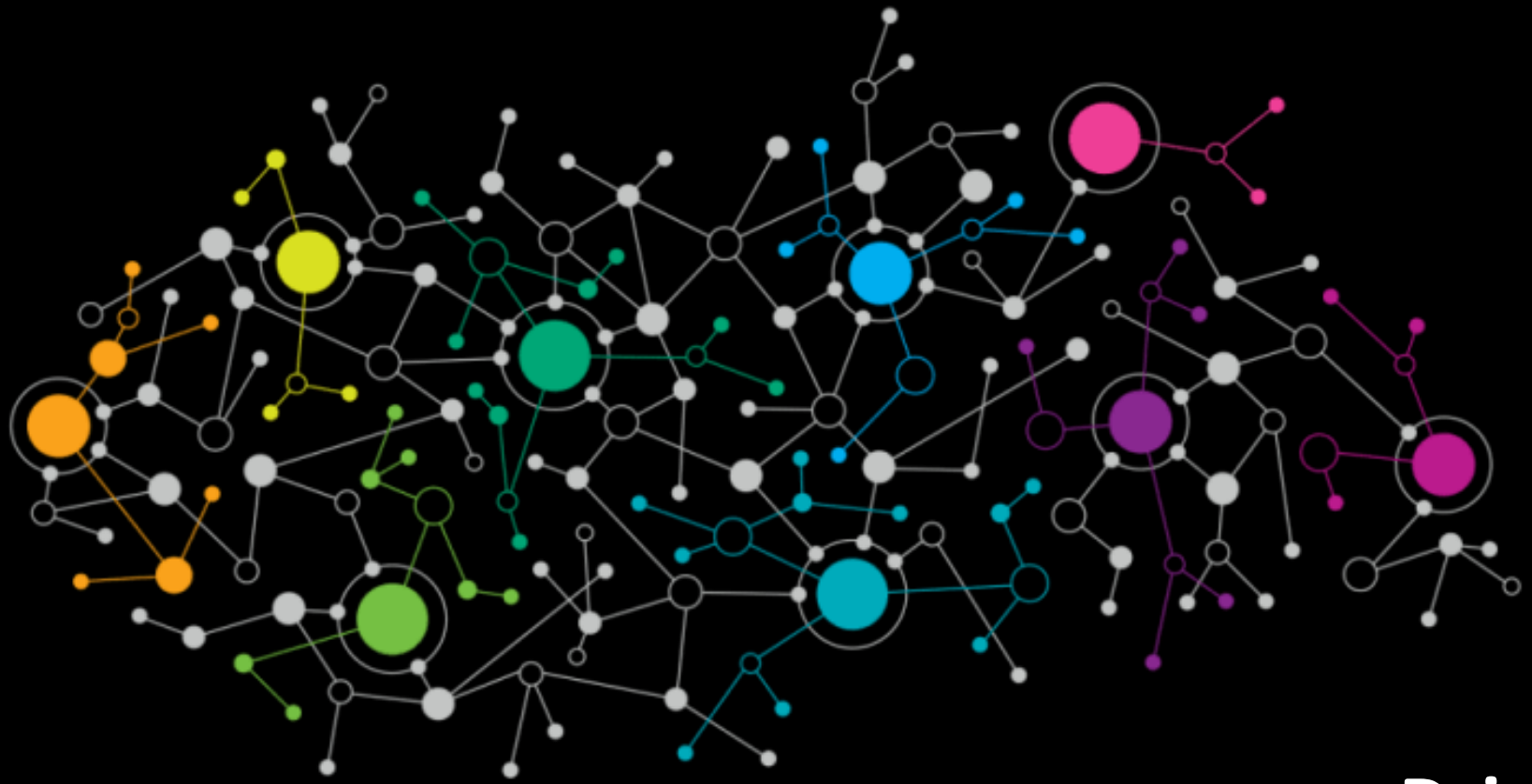


Figure 1: High-level illustration of our proposed method DIFFPOOL. At each hierarchical layer, we run a GNN model to obtain embeddings of nodes. We then use these learned embeddings to cluster nodes together and run another GNN layer on this coarsened graph. This whole process is repeated for L layers and we use the final output representation to classify the graph.



Dziękuję !