Graph Learning

IFT6758 - Data Science

Sources:

http://snap.stanford.edu/proj/embeddings-www/

https://jian-tang.com/files/AAAI19/aaai-grltutorial-part2-gnns.pdf

Mila

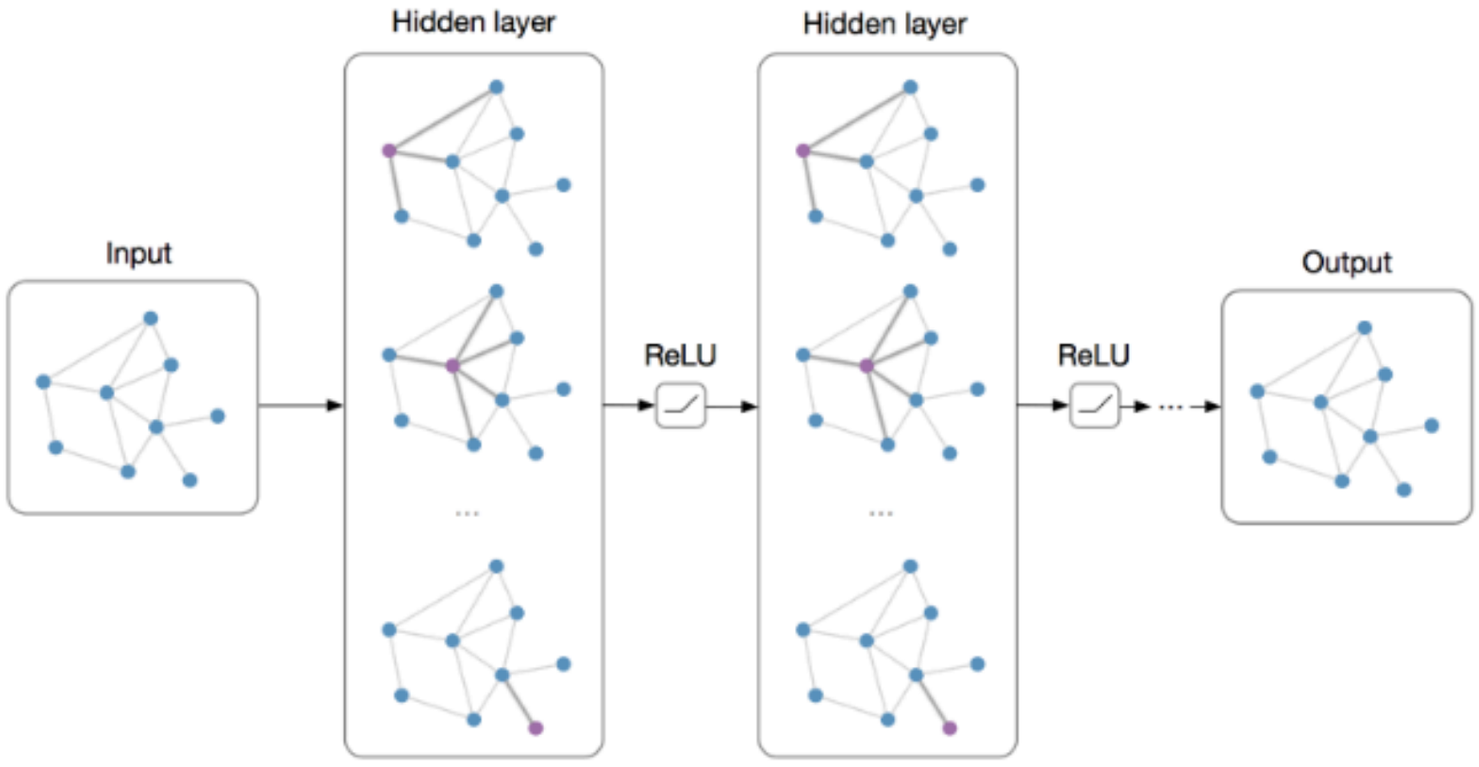
Université de Montréal

The Basics: Graph Neural Networks

Based on material from:

- Hamilton et al. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin on Graph Systems*.
- Scarselli et al. 2005. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*.

Graph Neural Networks

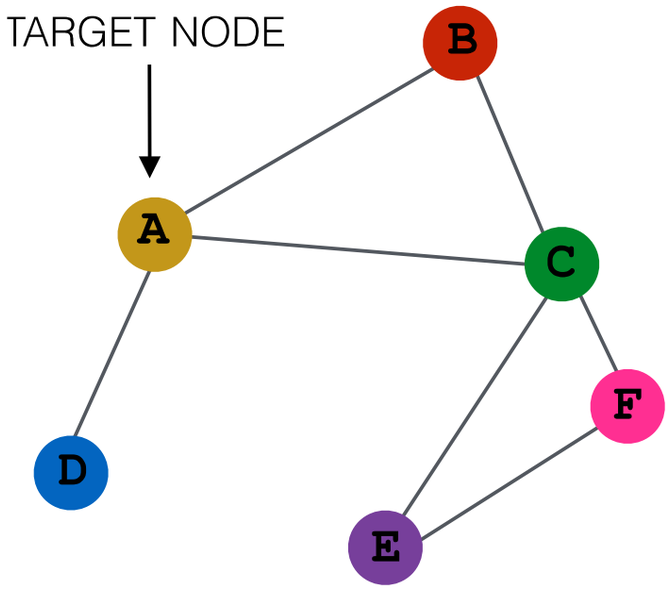


Setup

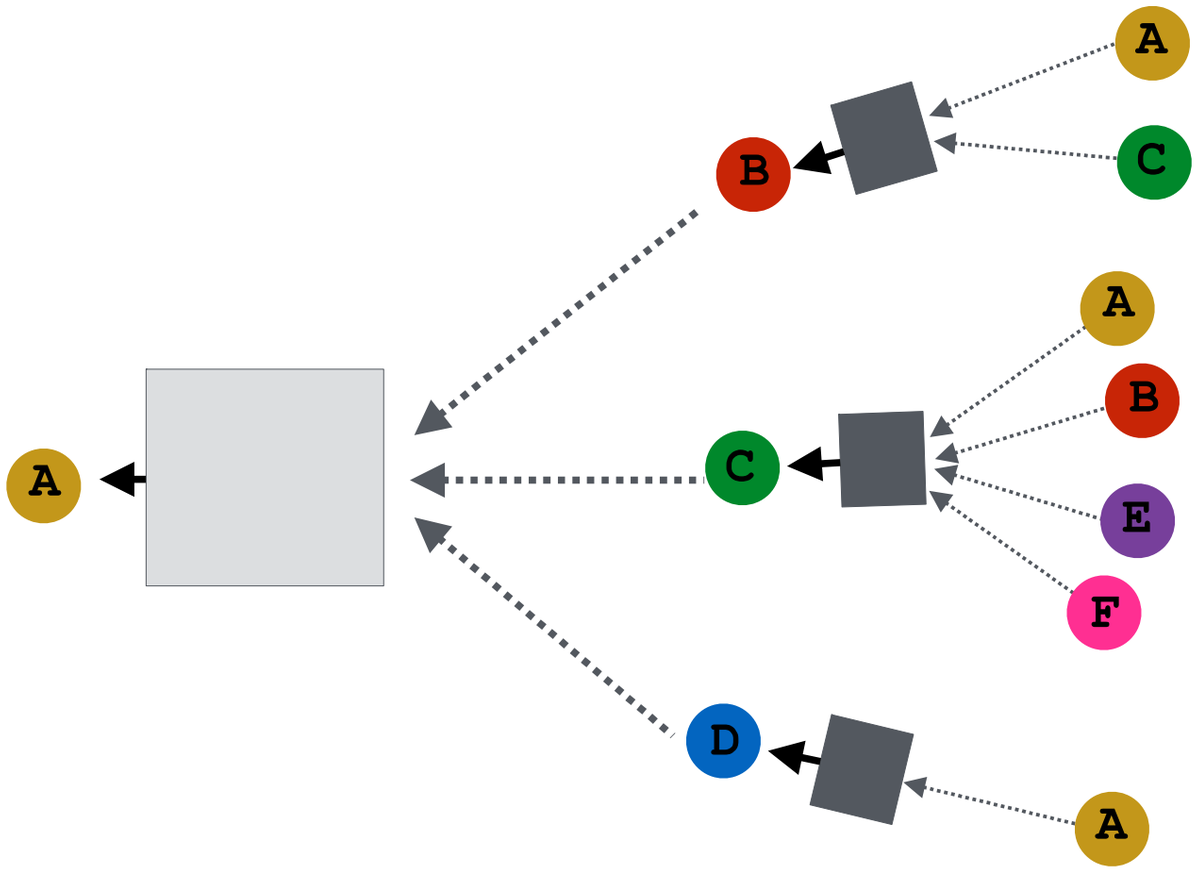
- Assume we have a graph G:
  - V is the vertex set.
  - A is the adjacency matrix (assume binary).
  - X is a matrix of node features.
    - Categorical attributes, text, image data
      - E.g., profile information in a social network.
    - Node degrees, clustering coefficients, etc.
    - Indicator vectors (i.e., one-hot encoding of each node)

Neighborhood Aggregation

- Key idea: Generate node embeddings based on **local neighborhoods.**

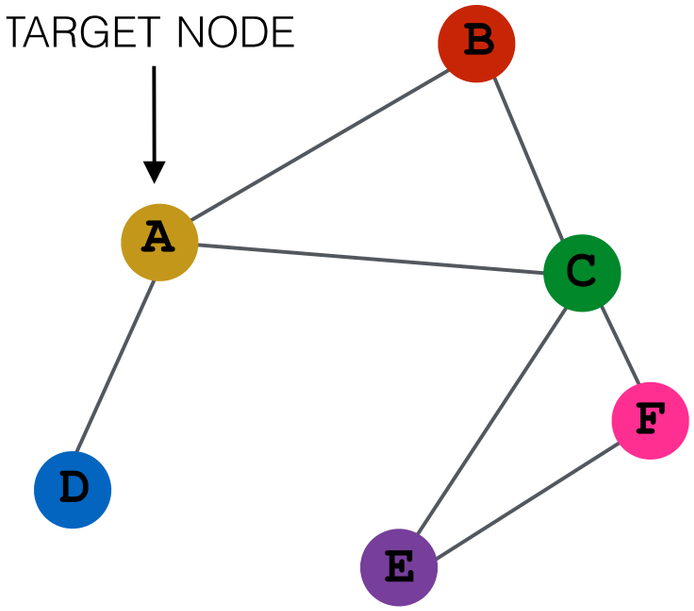


INPUT GRAPH

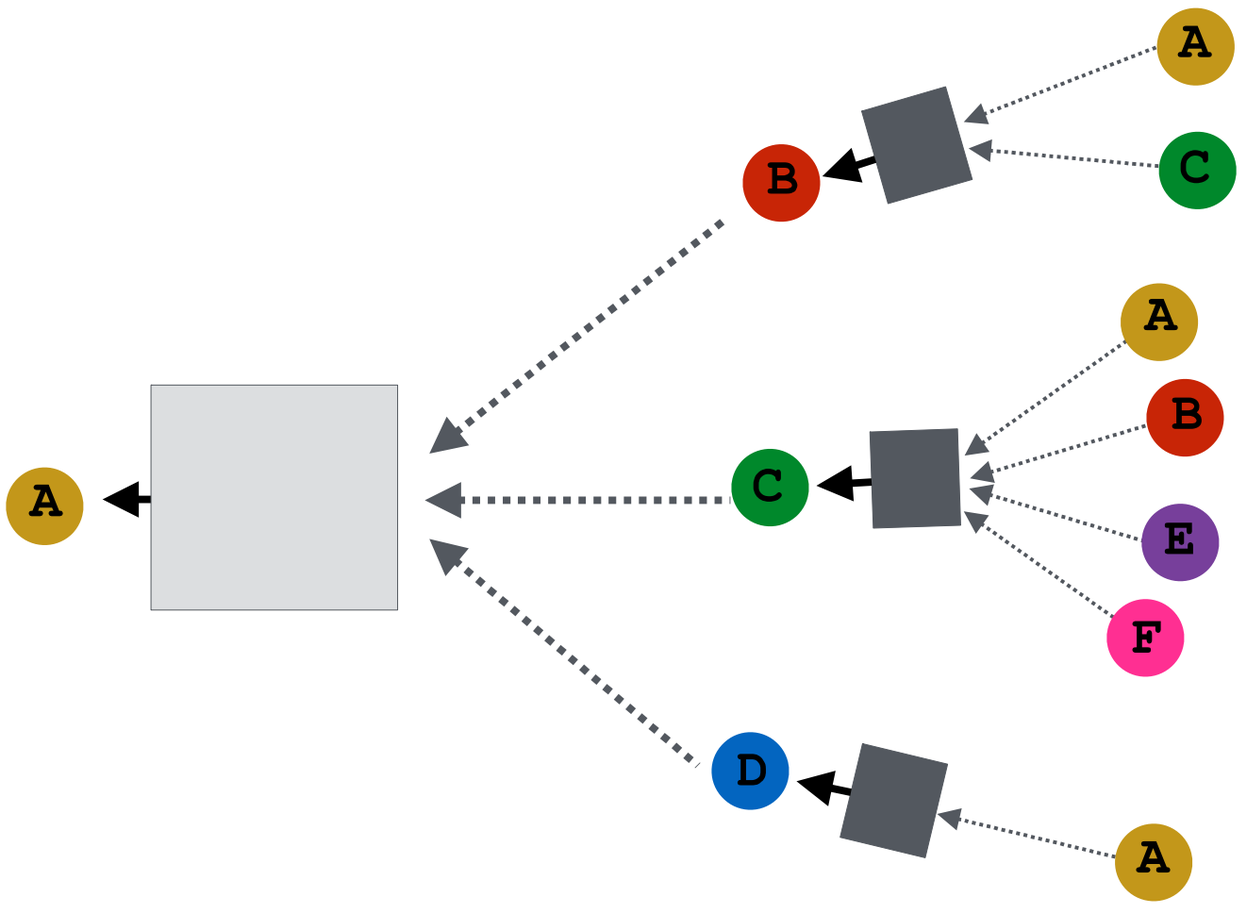


Neighborhood Aggregation

- Intuition: Nodes **aggregate** information from their neighbors using neural networks



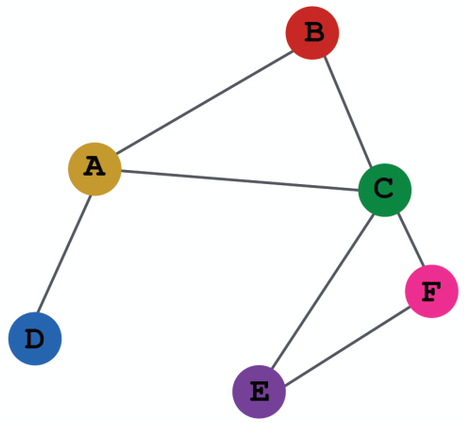
INPUT GRAPH



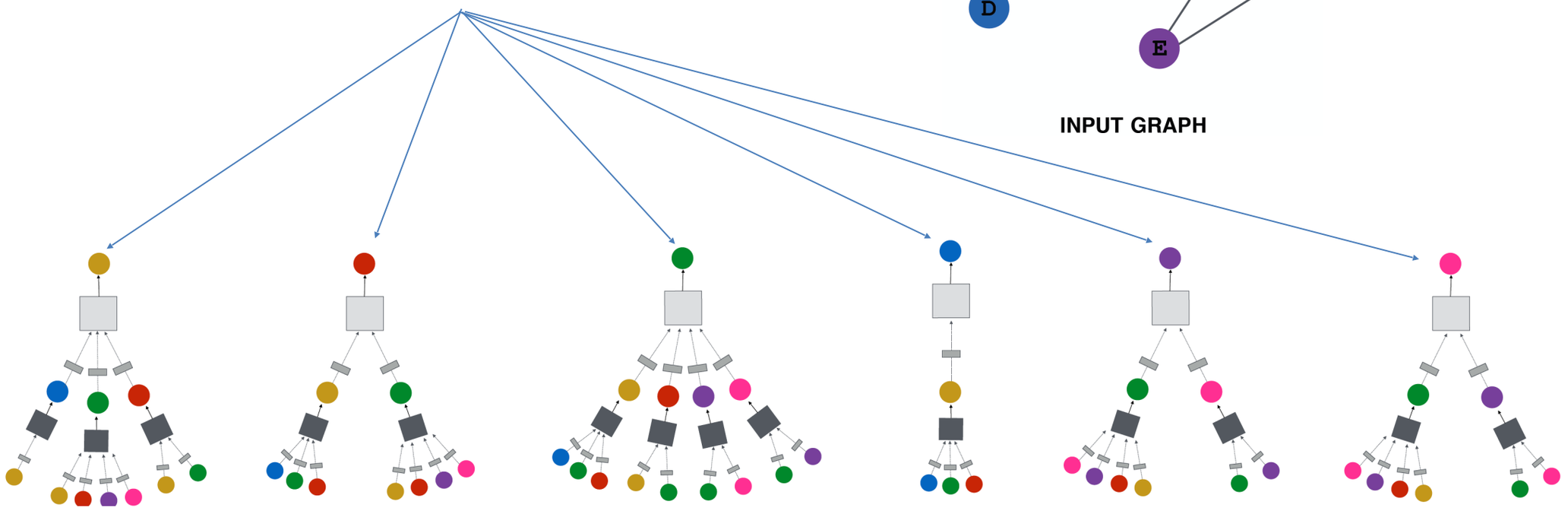
Neighborhood Aggregation

- Intuition: Network neighborhood defines a **computation graph**

Every node defines a unique computation graph!

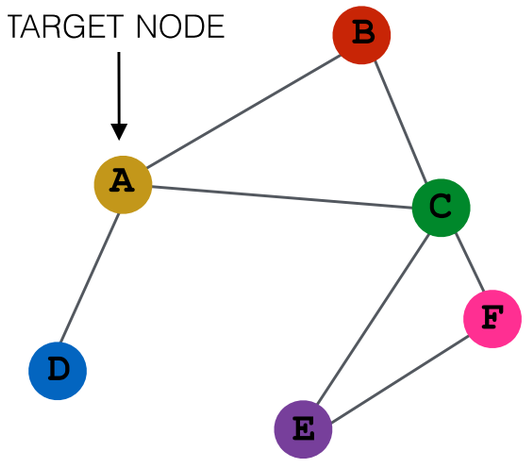


INPUT GRAPH

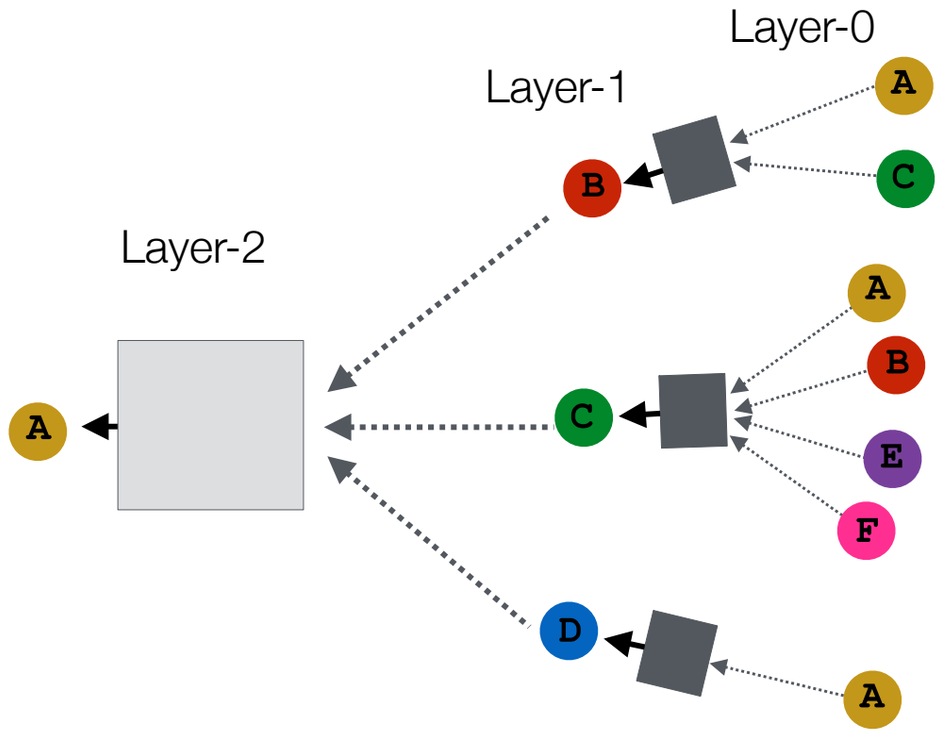


Neighborhood Aggregation

- Nodes have embeddings at each layer.
- Model can be arbitrary depth.
- “layer-0” embedding of node u is its input feature, i.e. x_u.

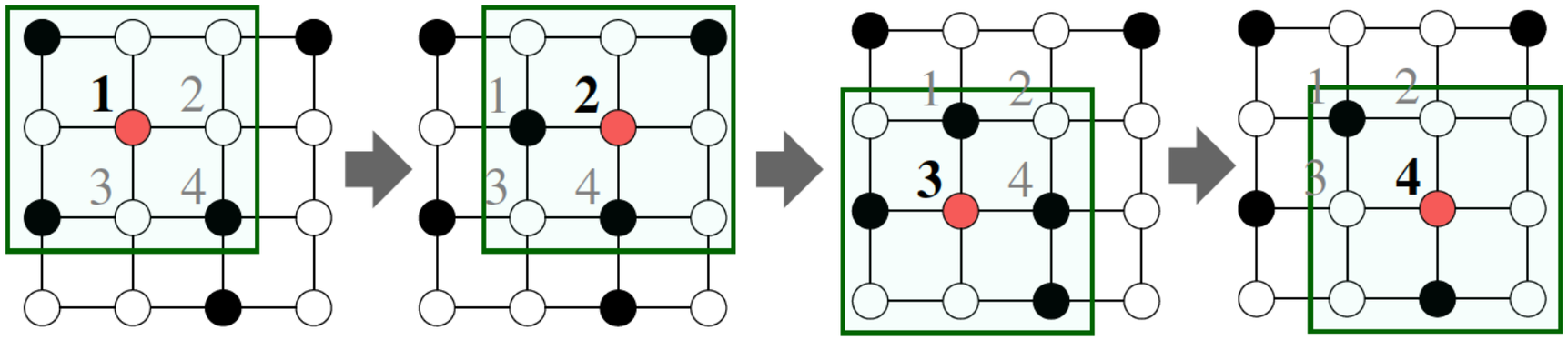


INPUT GRAPH



Neighborhood “Convolutions”

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

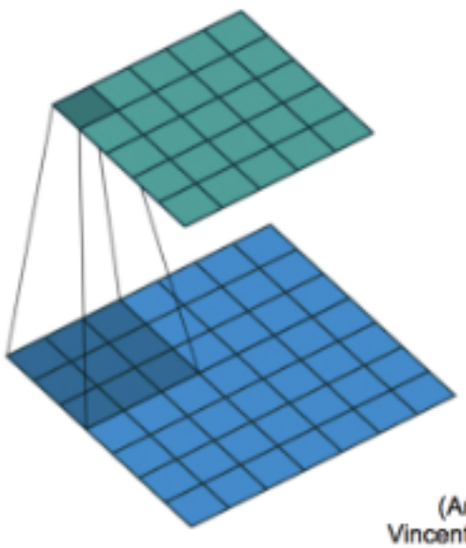


- Mathematically related to spectral graph convolutions (see [Bronstein et al., 2017])

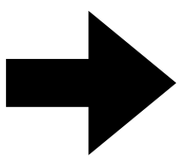
Convolution on Images

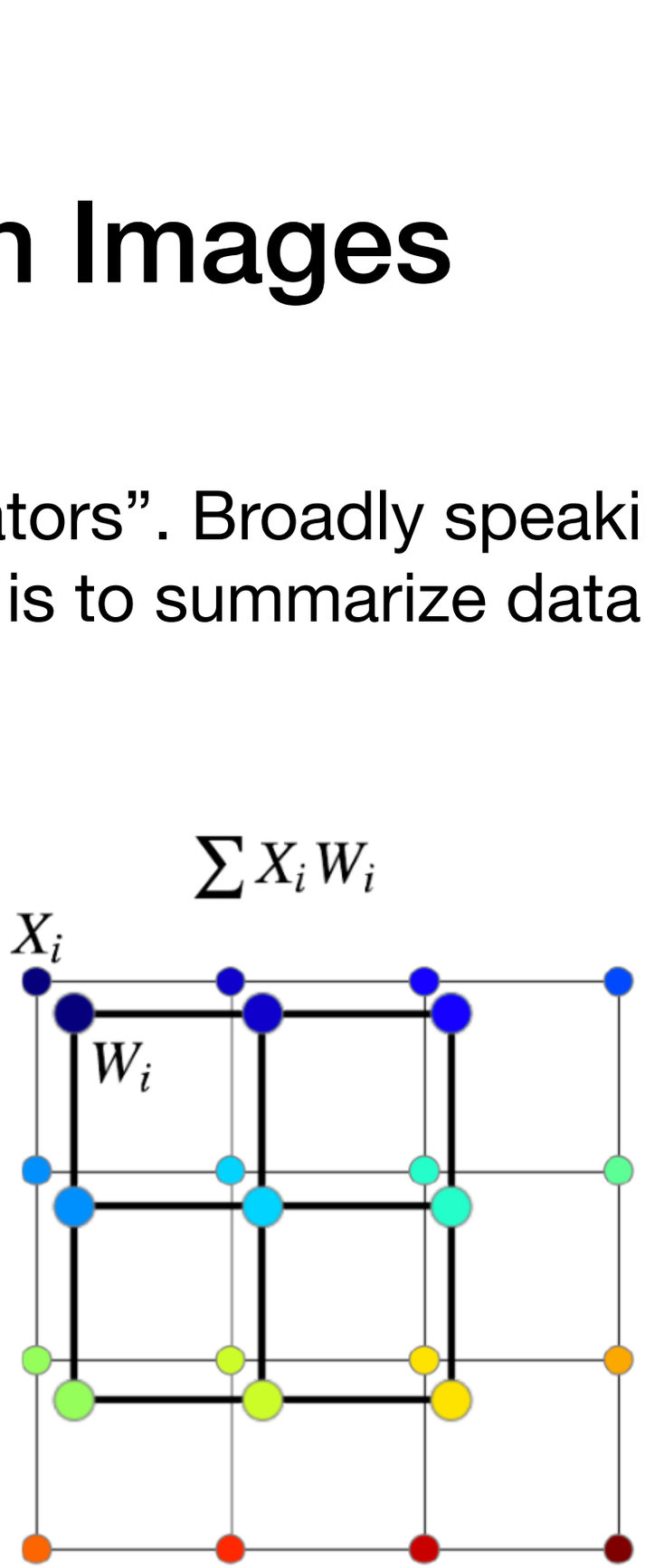
Convolution is a “aggregator operators”. Broadly speaking, the goal of an aggregator operator is to summarize data to a reduced form.

Single CNN layer with 3x3 filter:



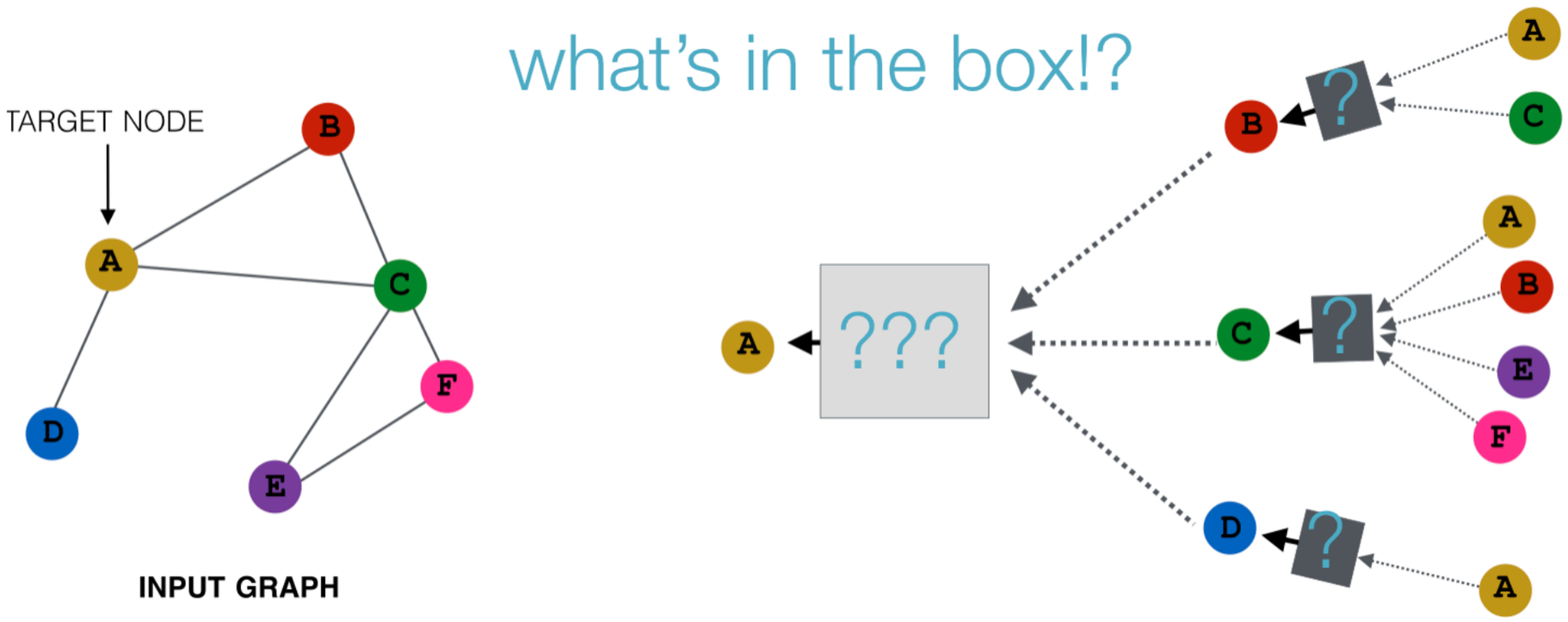
(Animation by Vincent Dumoulin)





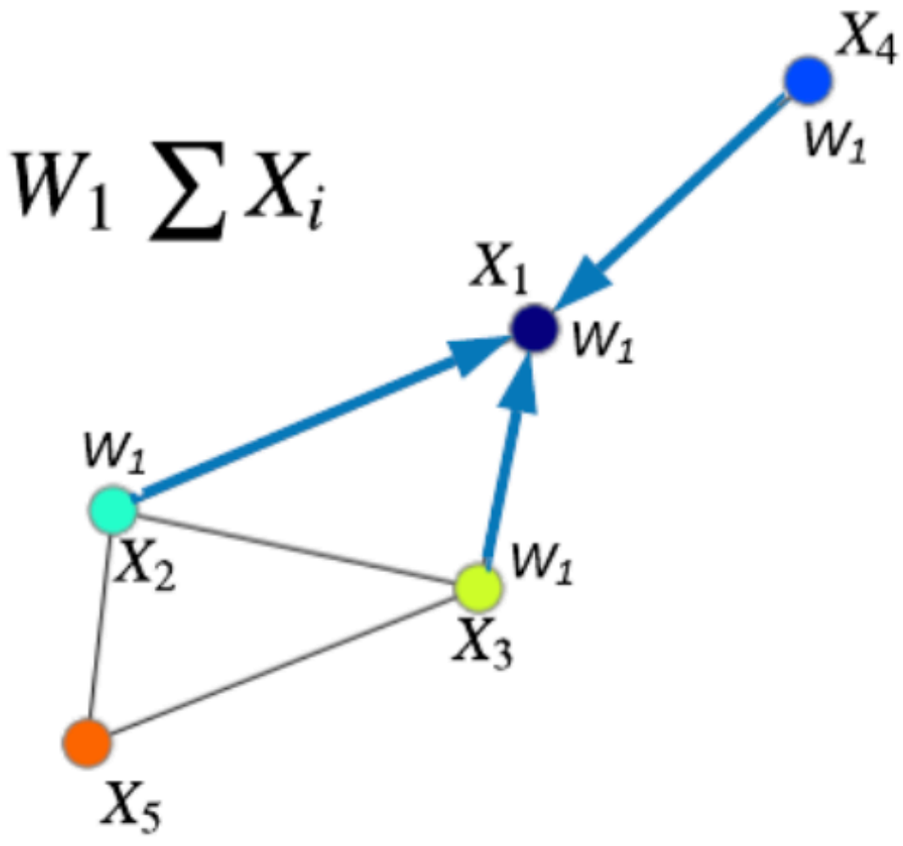
Neighborhood Aggregation

- Key distinctions are in how different approaches aggregate information across the layers.



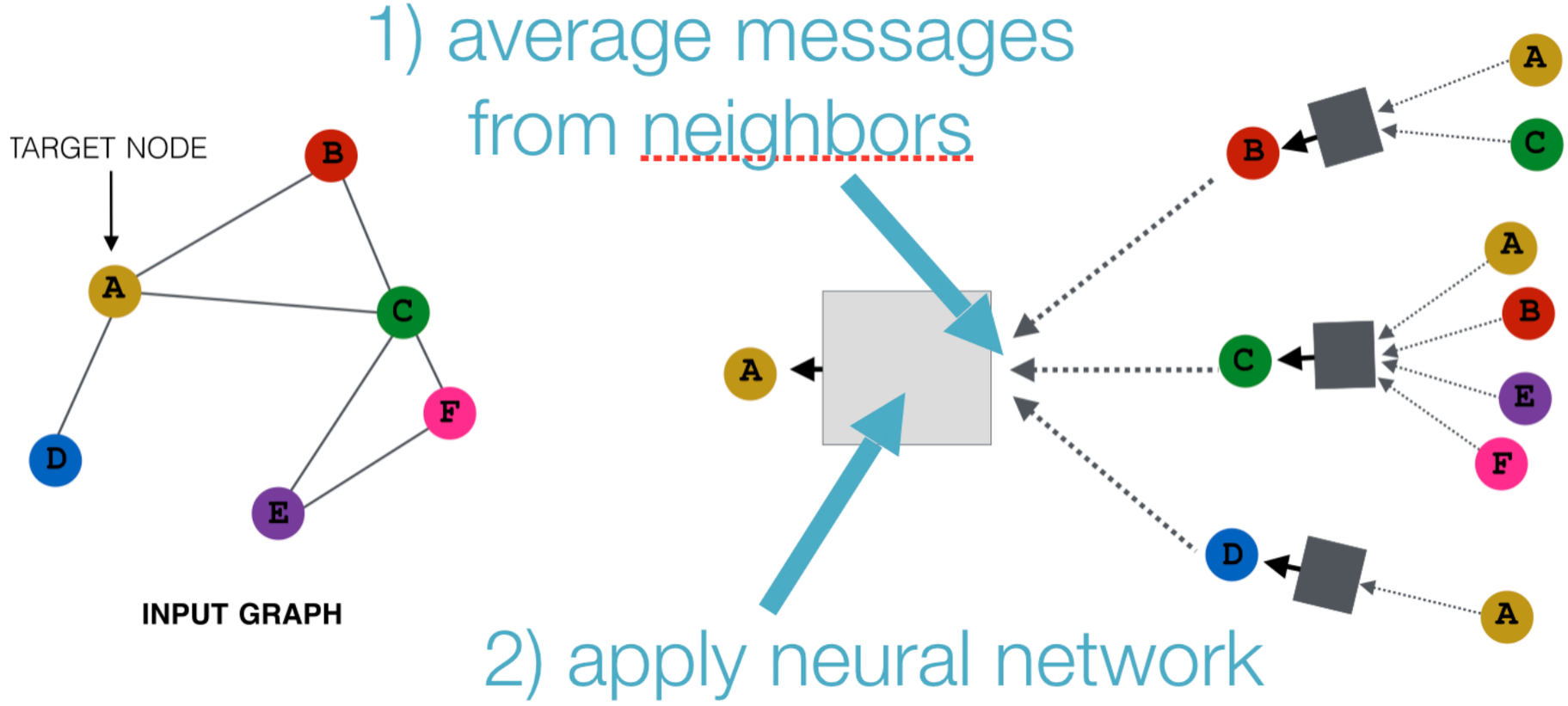
Convolution on Graphs

- The most popular choices of convolution on graphs are averaging \ and summation of **all** neighbors, i.e. sum or mean pooling, followed by projection by a trainable vector *W.*



Neighborhood Aggregation

- Basic approach: Average neighbor information and apply a neural network.



The Math

- Basic approach: Average neighbor messages and apply a neural network.

Initial "layer 0" embeddings are equal to node features

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

The Math

- Basic approach: Average neighbor messages and apply a neural network.

Initial "layer 0" embeddings are equal to node features

previous layer embedding of v

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

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

kth layer embedding of v

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

average of neighbor's previous layer embeddings

The Math

- Basic approach: Average neighbor messages and apply a neural network.

Initial "layer 0" embeddings are equal to node features

previous layer embedding of v

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

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

kth layer embedding of v

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

average of neighbor's previous layer embeddings

Training the Model

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

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

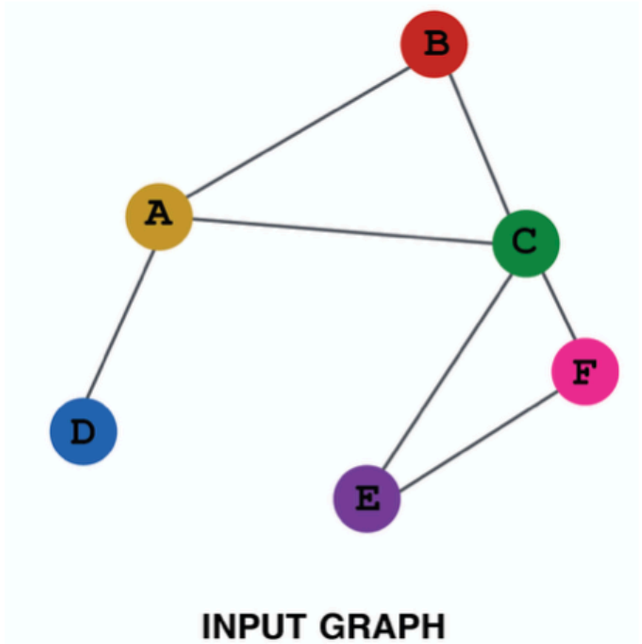
$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right), \quad \forall k \in \{1, ..., K\}$$

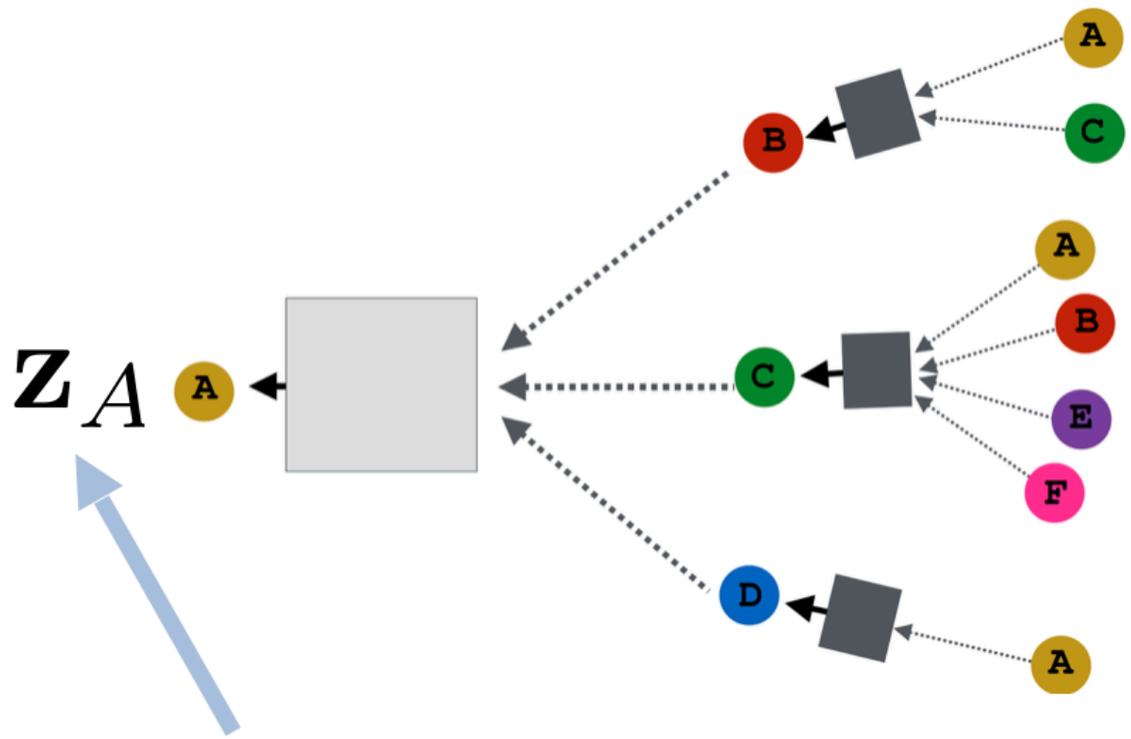
$$\mathbf{z}_v = \mathbf{h}_v^K$$

- After K-layers of neighborhood aggregation, we get output embeddings for each node.

Training the Model

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





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

Training the Model

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

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

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

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

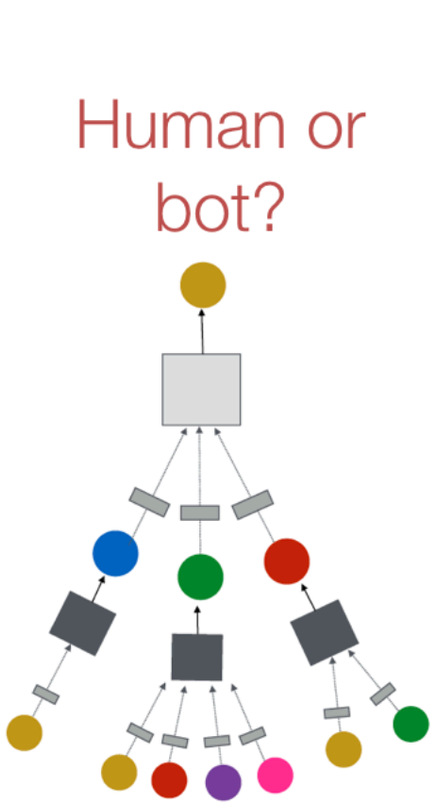
- After K-layers of neighborhood aggregation, we get output embeddings for each node.
- We can feed these embeddings into any loss function and run stochastic gradient descent to train the aggregation parameters.

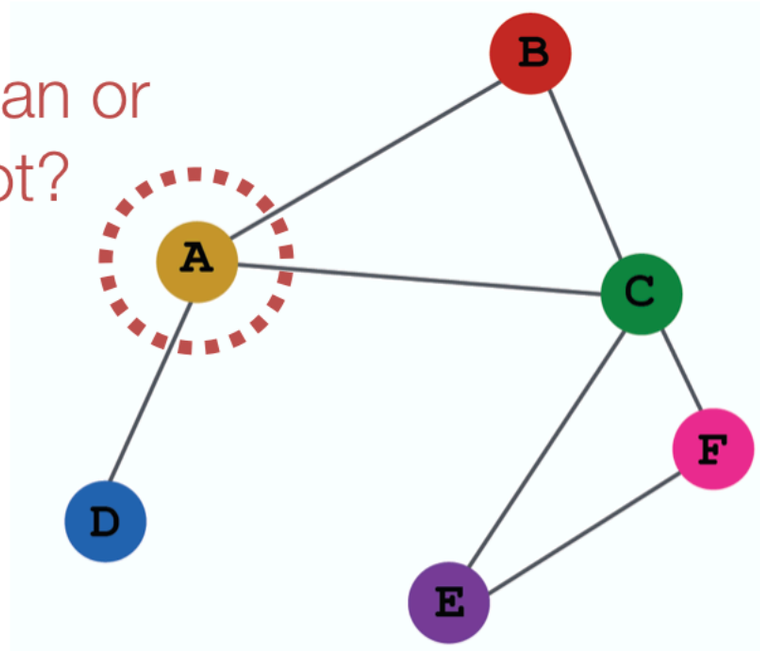
Training the Model

- Train in an **unsupervised** manner using only the graph structure.
- **Unsupervised loss function** can be anything e.g., based on
  - **Random walks** (node2vec, DeepWalk)
  - **Graph factorization**
  - i.e., train the model so that “similar” nodes have similar embeddings.

Training the Model

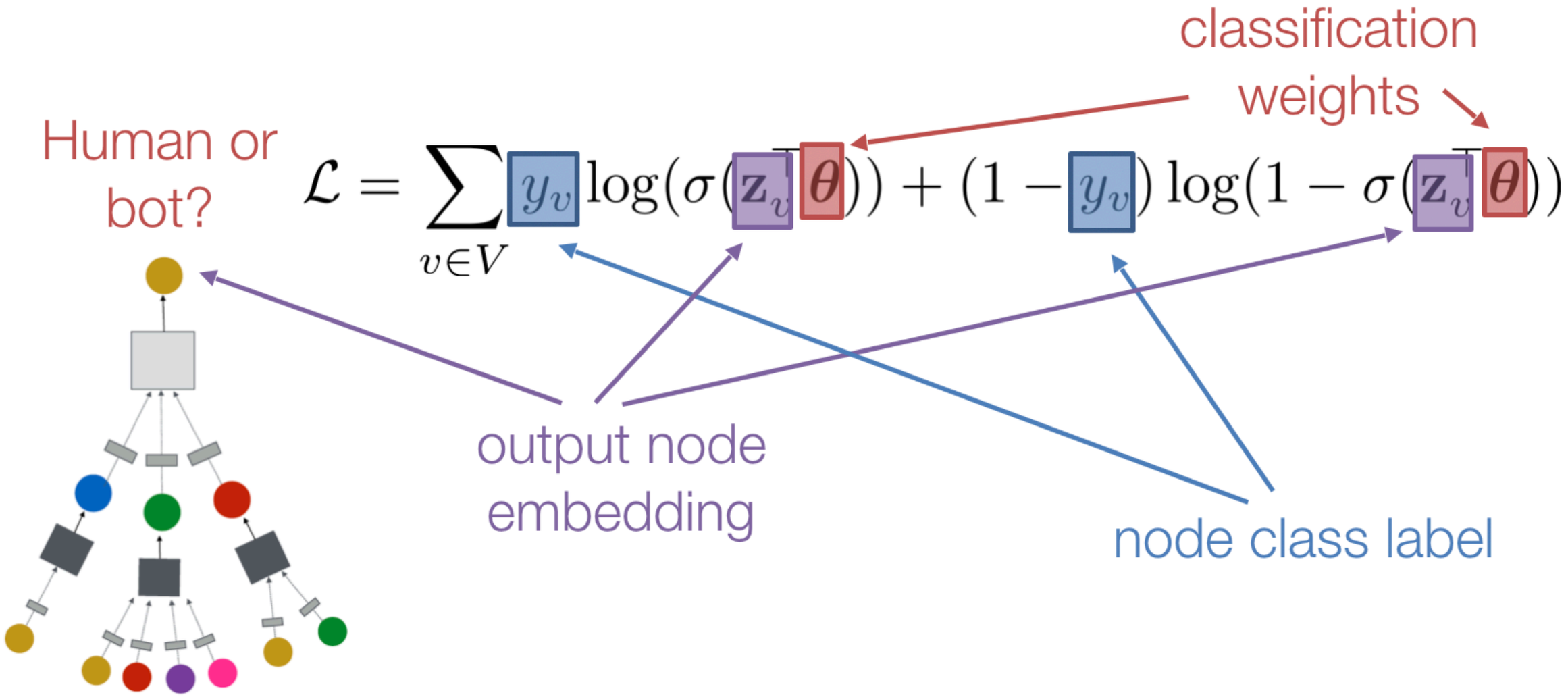
- Alternative: Directly train the model for a **supervised task** (e.g., node classification):





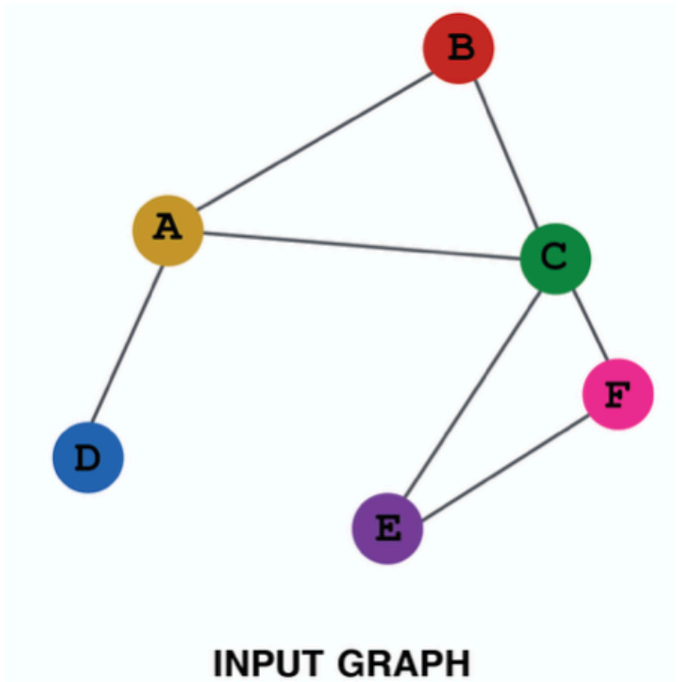
Training the Model

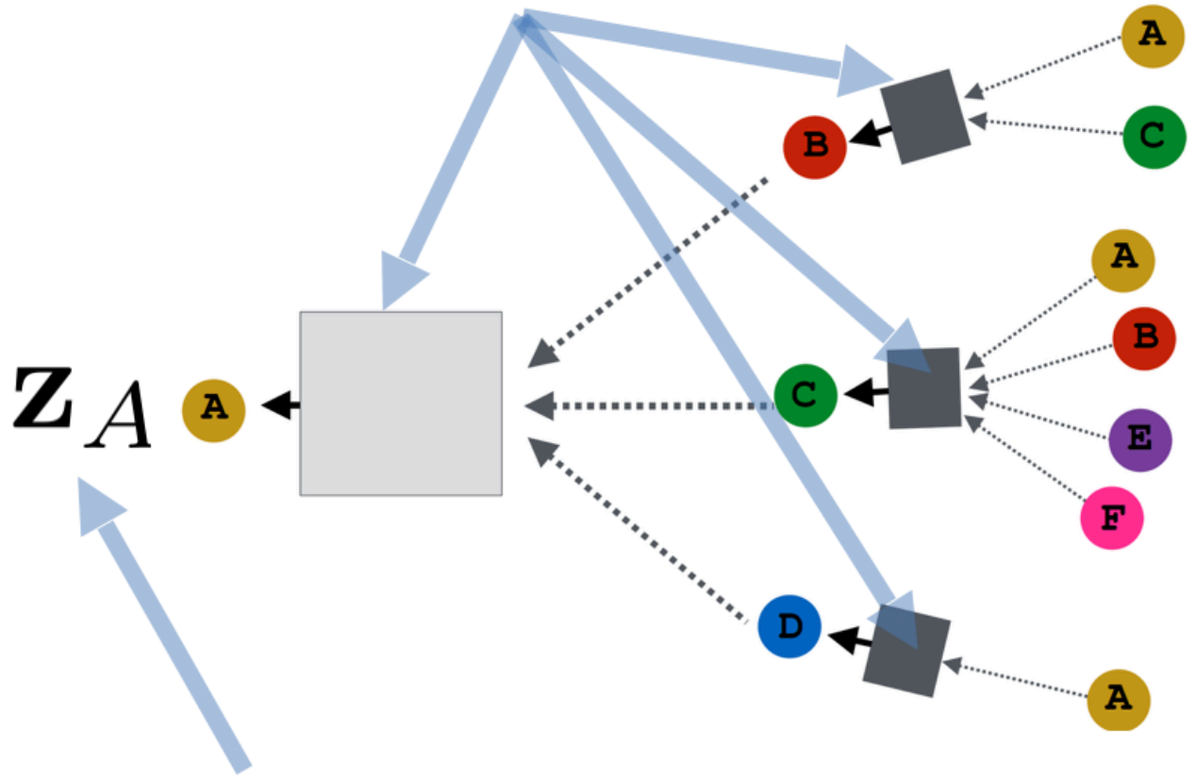
- Alternative: Directly train the model for a supervised task (e.g., node classification):



Overview of Model

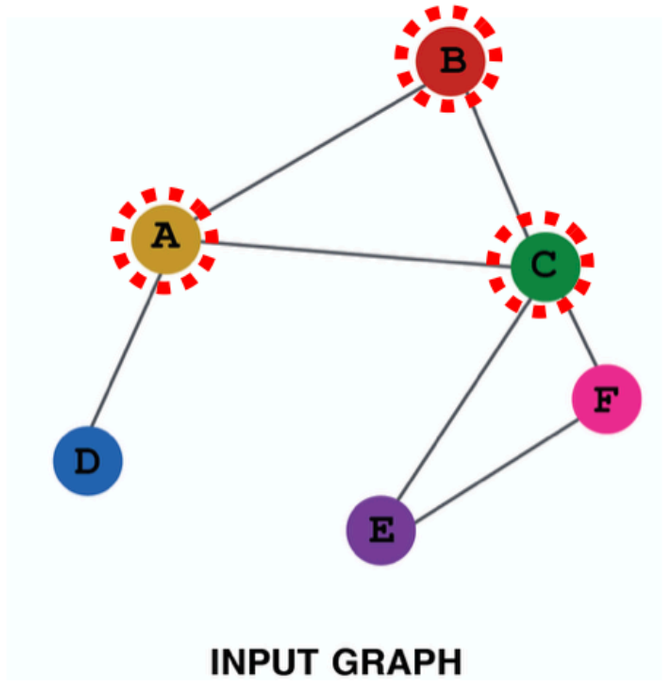
1) Define a neighborhood aggregation function.



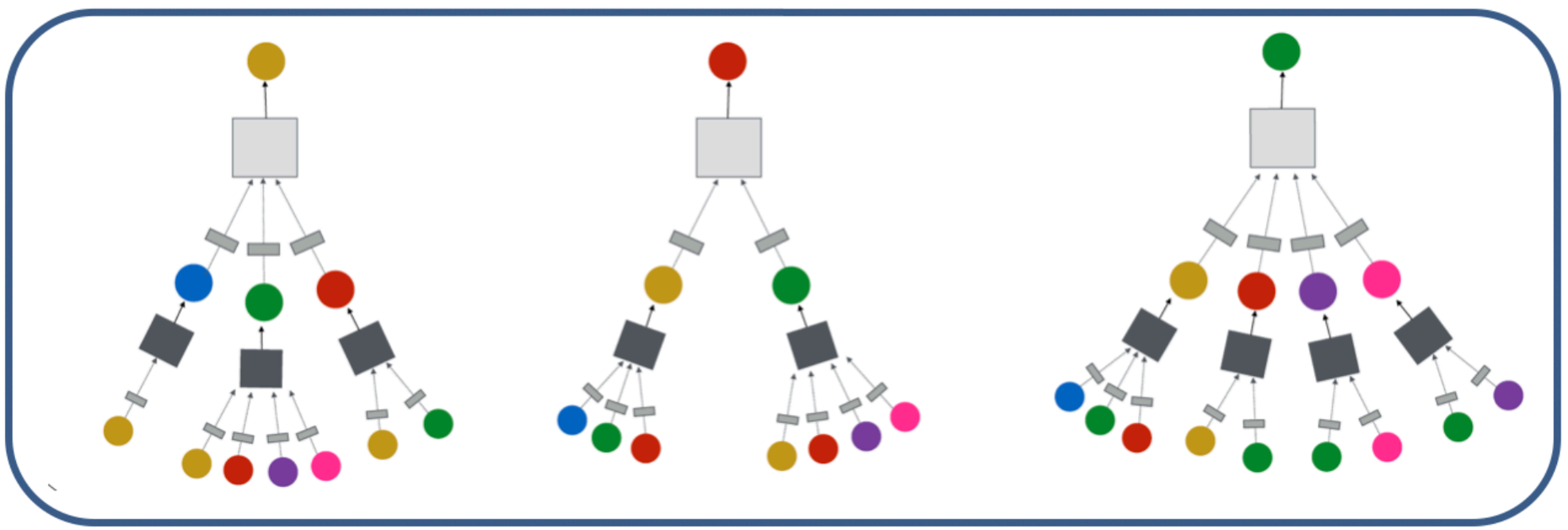


2) Define a loss function on the embeddings, $\mathcal{L}(z_u)$

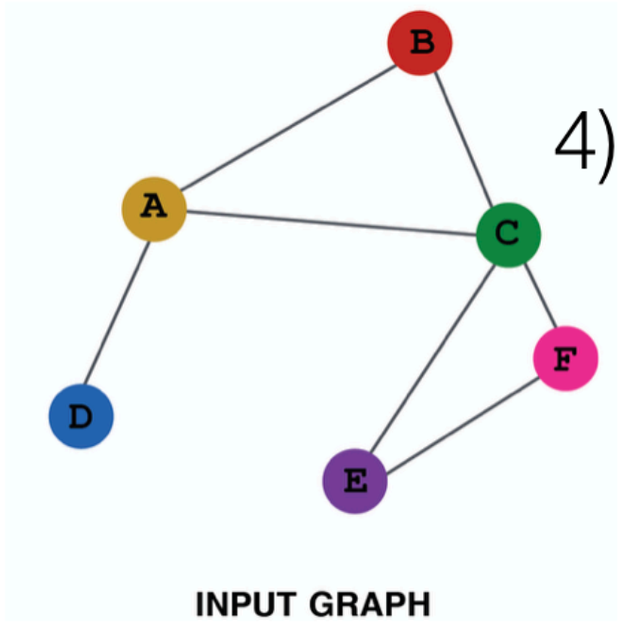
Overview of Model



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

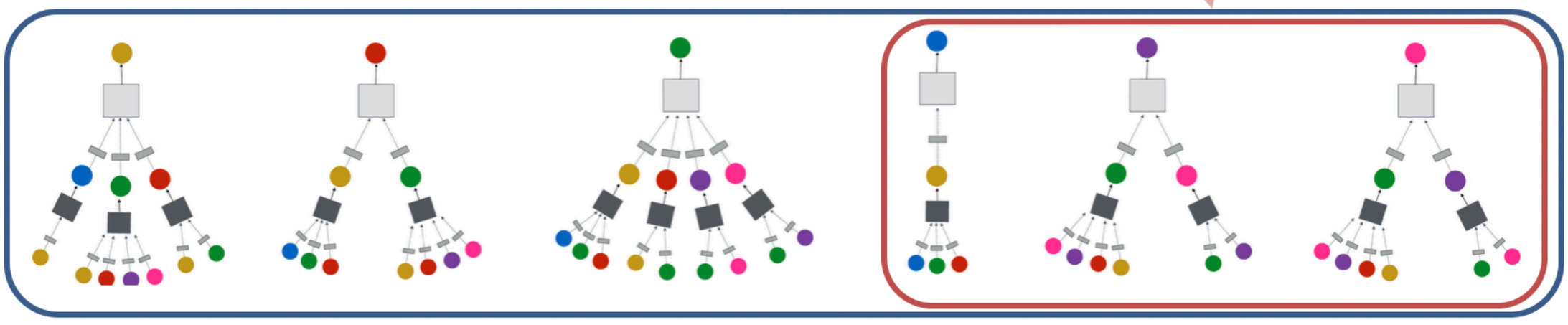


Overview of Model



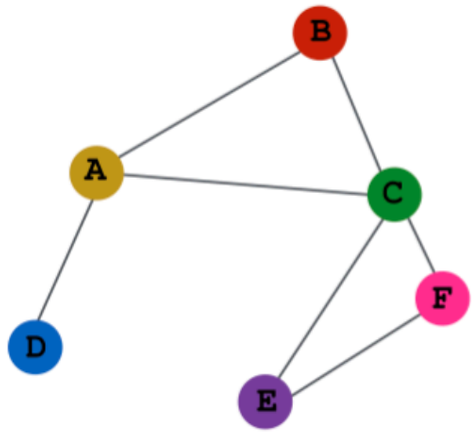
4) Generate embeddings for nodes as needed

Even for nodes we never trained on!!!!

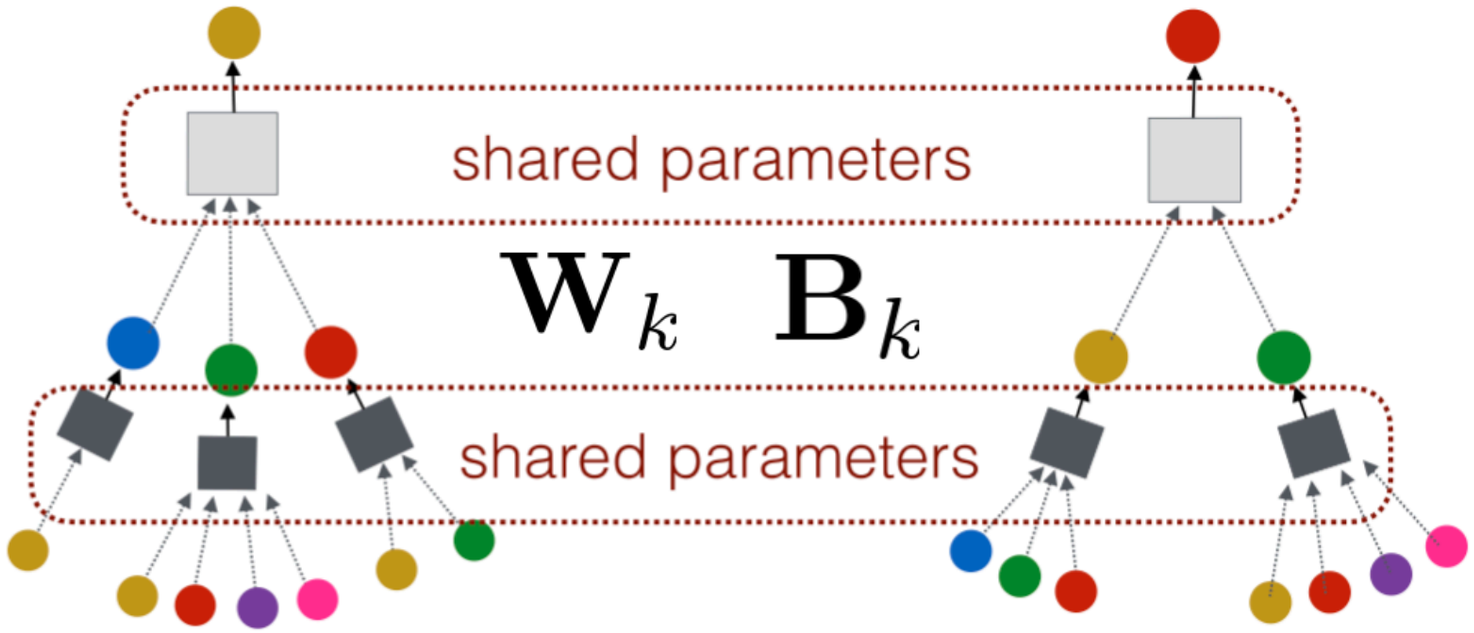


Inductive Capability

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



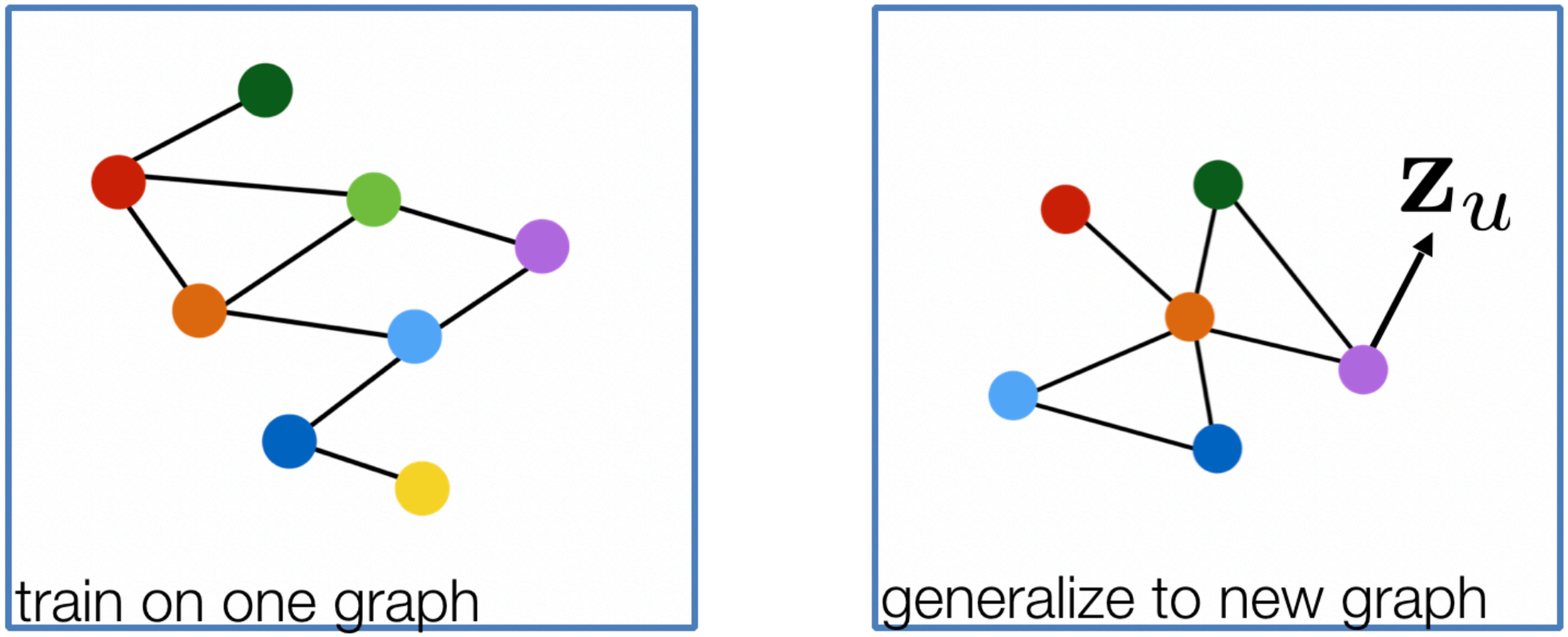
INPUT GRAPH



Compute graph for node A

Compute graph for node B

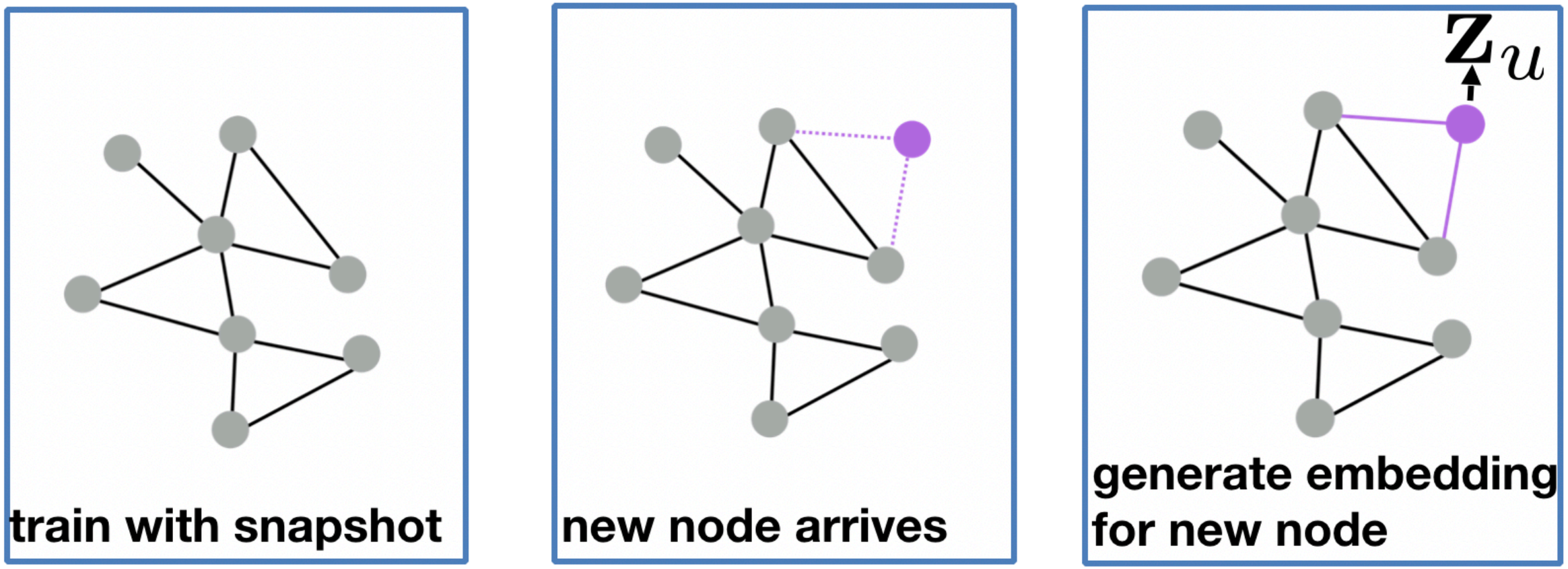
Inductive Capability



Inductive node embedding → generalize to entirely unseen graphs

e.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Inductive Capability



Many application settings constantly encounter previously unseen nodes.

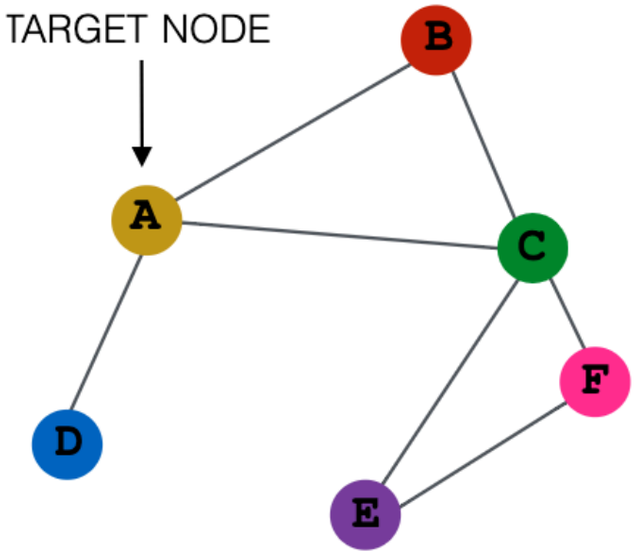
e.g., Reddit, YouTube, GoogleScholar, ….

Need to generate new embeddings “on the fly”

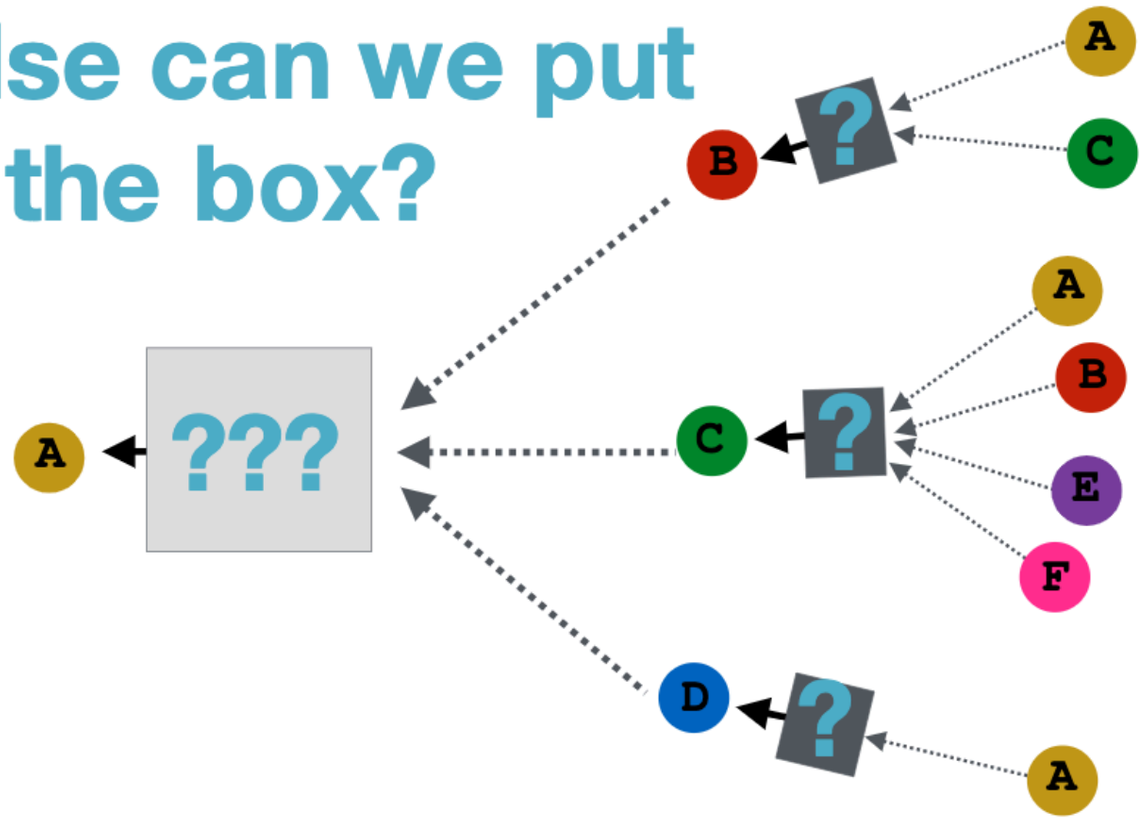
Recap

- Recap: Generate node embeddings by **aggregating neighborhood** information.
  - Allows for **parameter sharing** in the encoder.
  - Allows for **inductive learning**.

What else can we put in the box?



INPUT GRAPH



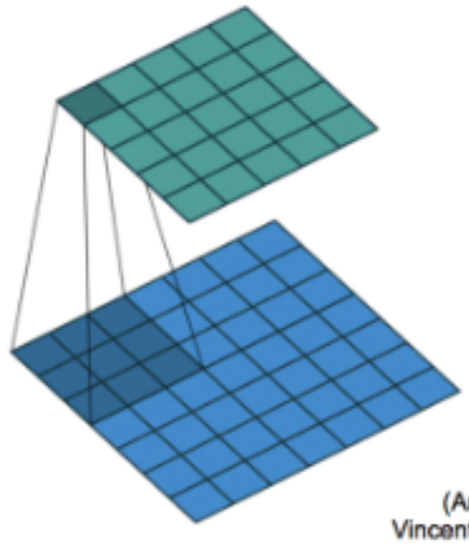
Graph Convolutional Networks

Based on material from:

- Kipf et al., 2017. Semisupervised Classification with Graph Convolutional Networks. *ICLR*.

Recap: Convolutional Neural Networks (CNNs)

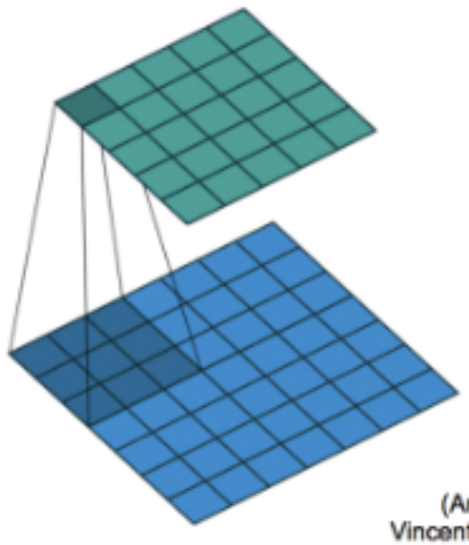
Single CNN layer with 3x3 filter:



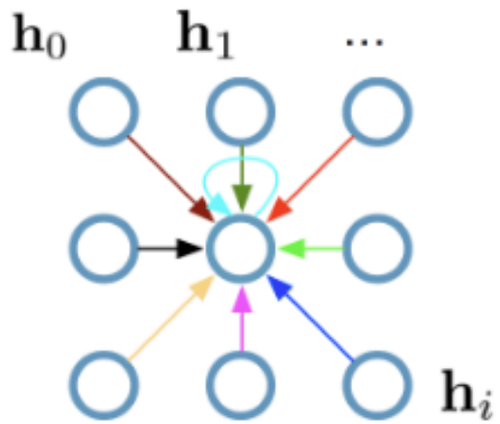
(Animation by Vincent Dumoulin)

Recap: Convolutional Neural Networks (CNNs)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

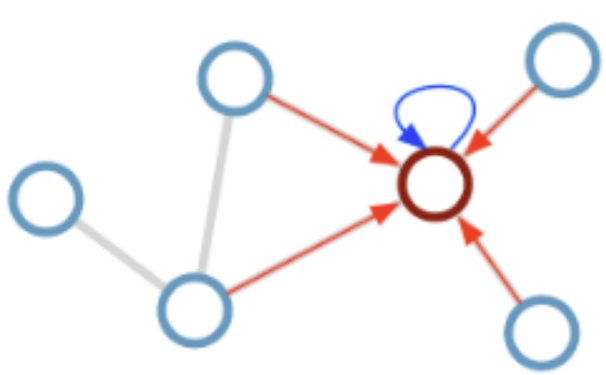
Graph Neural Networks (GNNs)

Consider this undirected graph:

Graph Neural Networks (GNNs)

Consider this undirected graph:

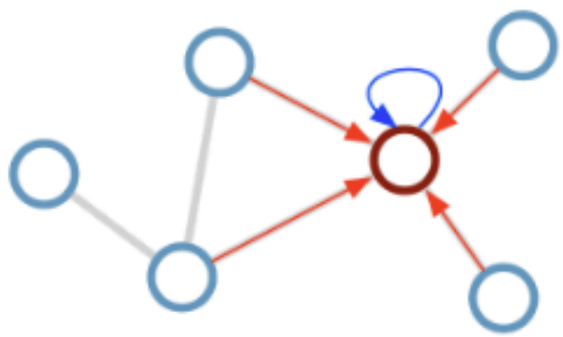
Calculate update for node in red:



Graph Neural Networks (GNNs)

Consider this undirected graph:

Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

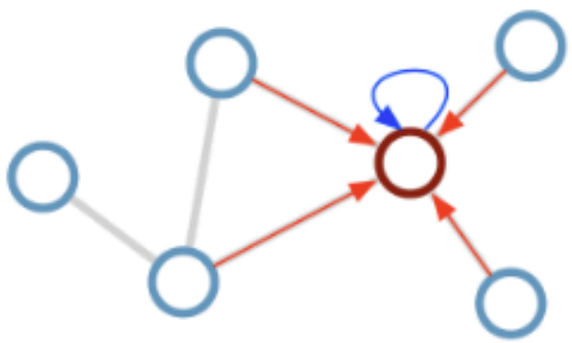
$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

Graph Neural Networks (GNNs)

Consider this undirected graph:

Calculate update for node in red:



Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)
- Applicable both in transductive and inductive settings

Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

Graph Convolutional Networks (GCNs)

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

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

Graph Convolutional Networks

Basic Neighborhood Aggregation

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

VS.

GCN Neighborhood Aggregation

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

same matrix for self and neighbor embeddings

per-neighbor normalization

Graph Convolutional Networks

- Empirically, they found this configuration to give the best results.
  - More parameter sharing.
  - Down-weights high degree neighbors.

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

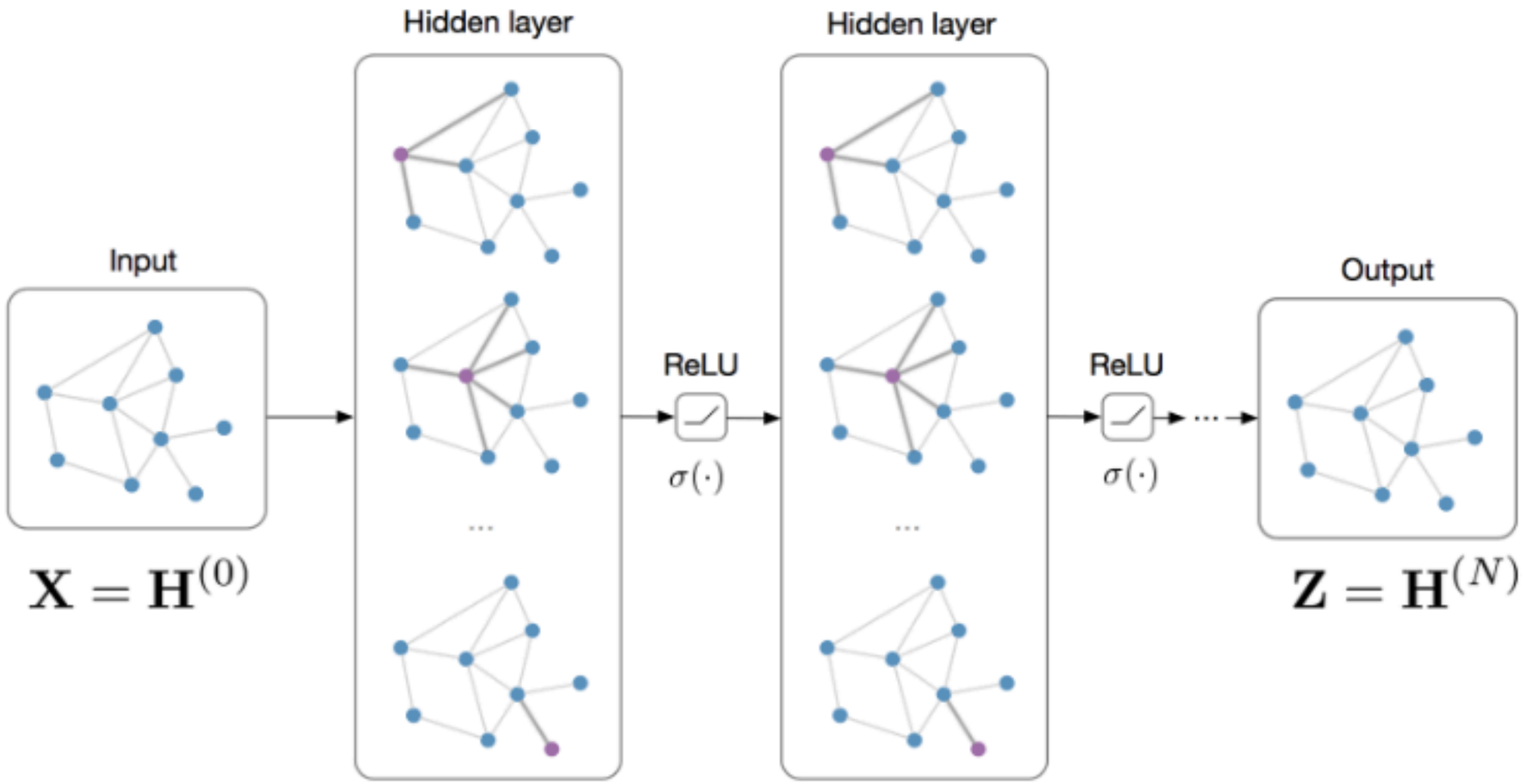
use the same transformation matrix for self and neighbor embeddings

instead of simple average, normalization varies across neighbors

GNN/GCN applications

Classification and Link Prediction with GNNs / GCNs

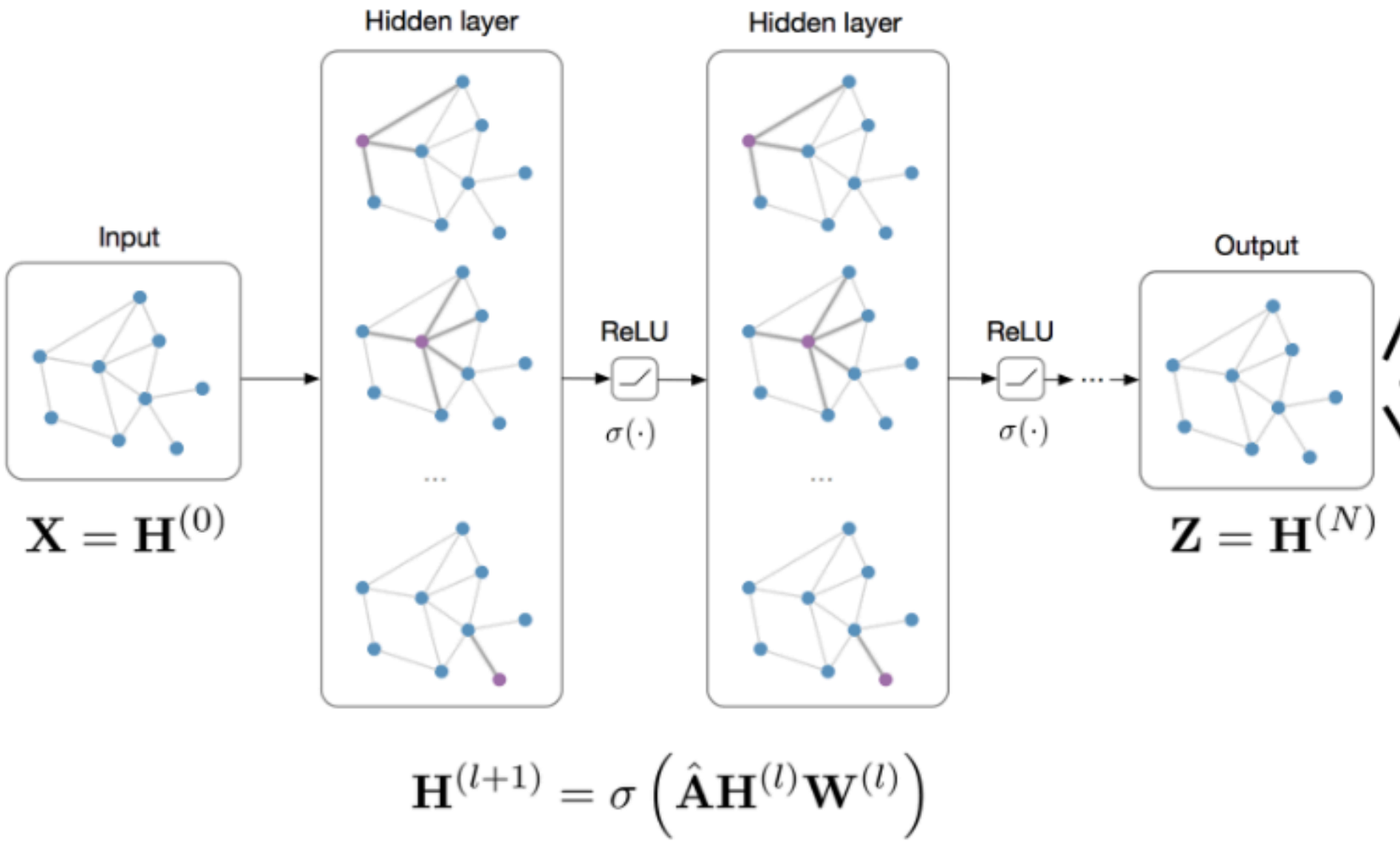
Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



$$\mathbf{H}^{(l+1)} = \sigma\left(\hat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\right)$$

Classification and Link Prediction with GNNs / GCNs

Input: Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\hat{\mathbf{A}}$



Node classification:

$\mathrm{softmax}(\mathbf{z_n})$

e.g. Kipf & Welling (ICLR 2017)

Graph classification:

$\mathrm{softmax}(\sum_n \mathbf{z_n})$

e.g. Duvenaud et al. (NIPS 2015)

Link prediction:

$p(A_{ij}) = \sigma(\mathbf{z_i^T z_j})$

Kipf & Welling (NIPS BDL 2016)

"Graph Auto-Encoders"

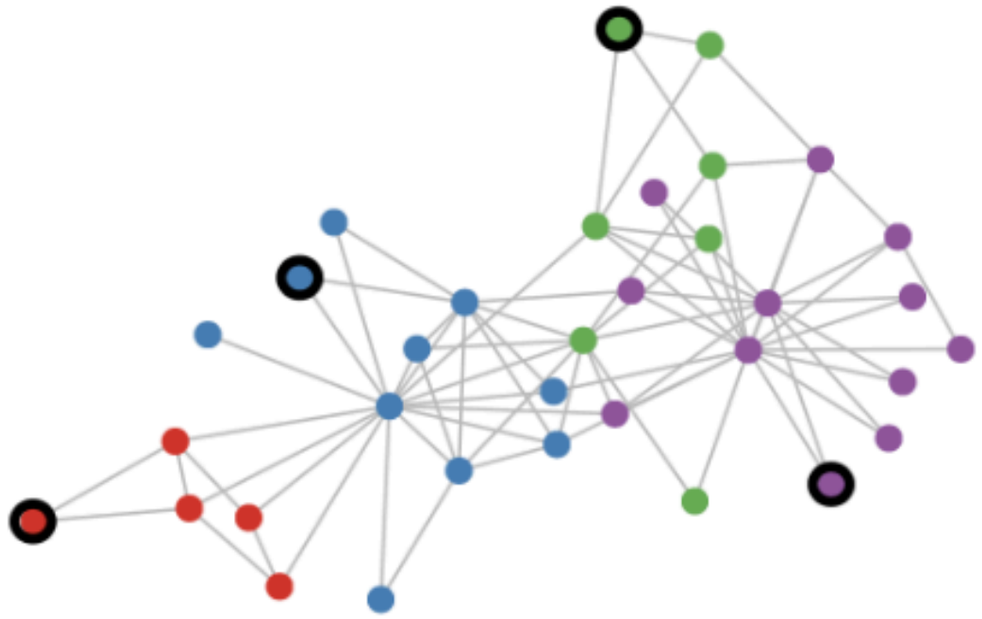
Semi-supervised Classification on Graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:

Predict node label of unlabeled nodes

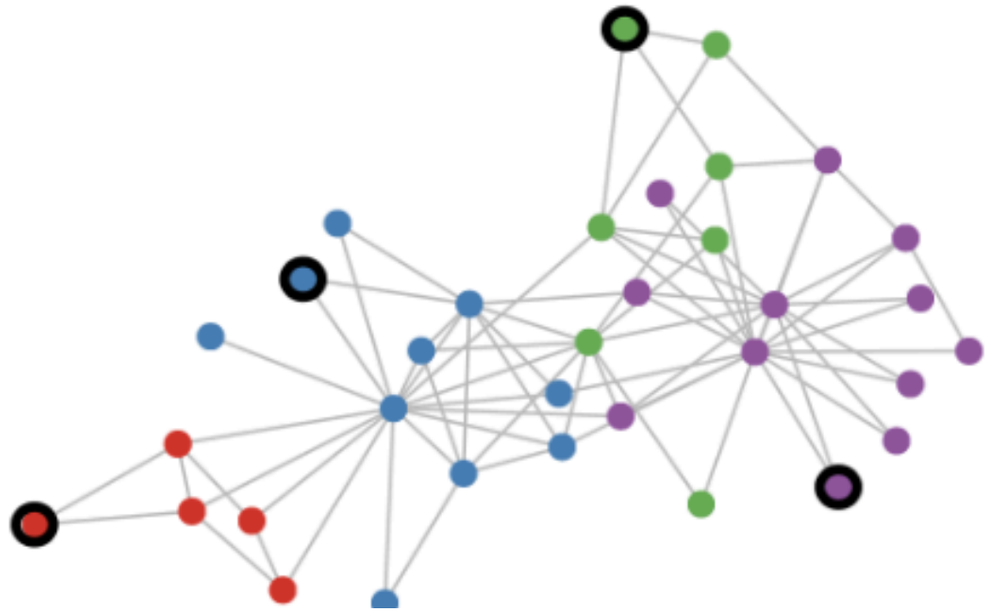


Semi-supervised Classification on Graphs

Setting:

Some nodes are labeled (black circle)
All other nodes are unlabeled

Task:
Predict node label of unlabeled nodes



Evaluate loss on labeled nodes only:

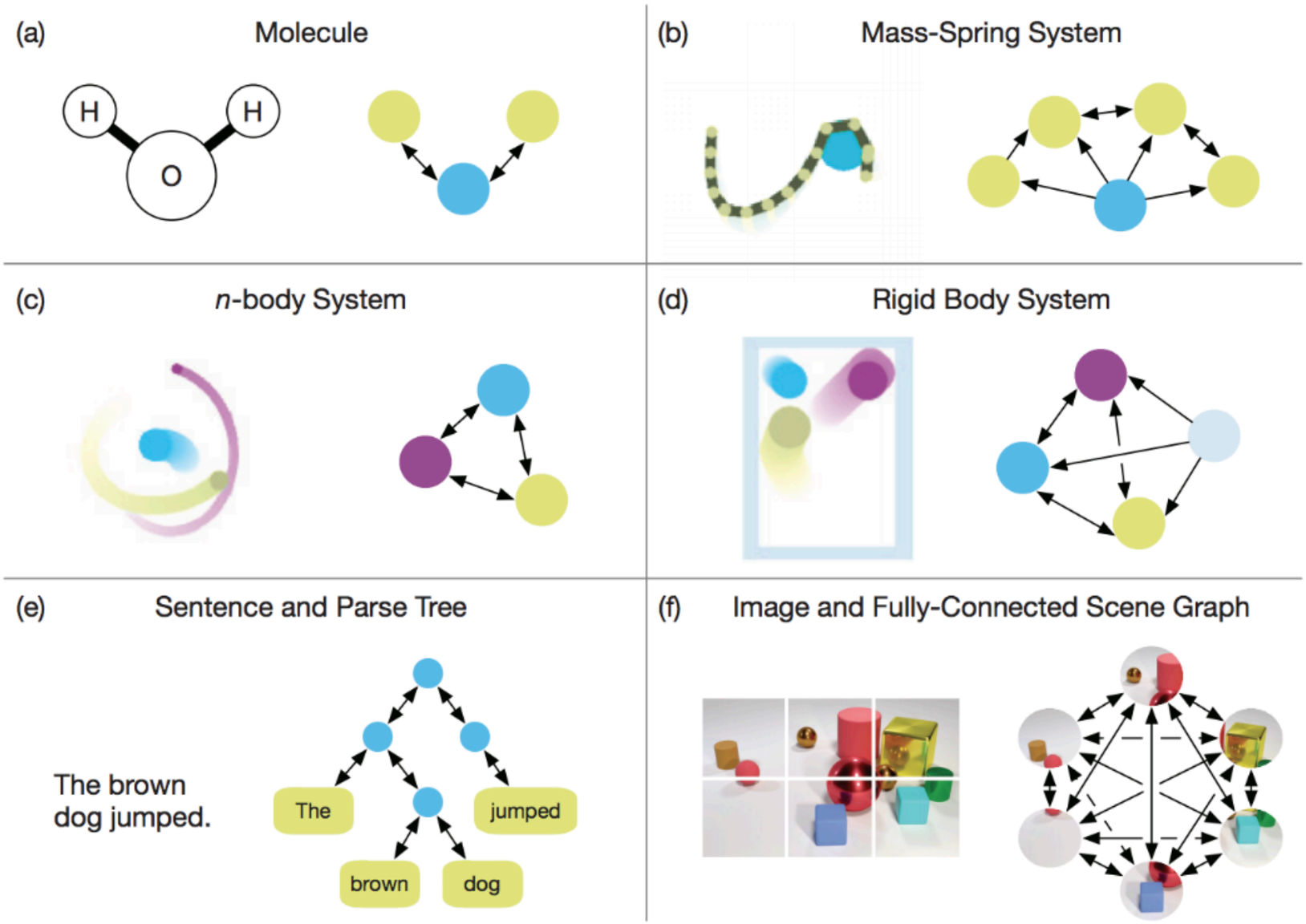
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

Universality of Graph Representations



The universality of Graph Representations https://arxiv.org/pdf/1806.01261.pdf

Conclusion

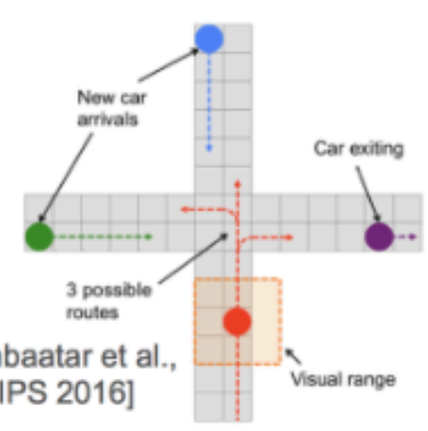
— **Deep learning on graphs works and is very effective!**

— Exciting area: lots of new applications and extensions (hard to keep up)

Relational reasoning

[Santoro et al., NIPS 2017]

Multi-Agent RL



[Sukhbaatar et al., NIPS 2016]

GCN for recommendation on 16 billion edge graph!

[Leskovec lab, Stanford]

Conferences focusing on Graphs

- **WWW:** The Web Conference
  https://www2020.thewebconf.org/

- **ASONAM:** The IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining
  http://asonam.cpsc.ucalgary.ca/2019/

- **ICML**: International Conference on Machine Learning
  https://icml.cc/

- **KDD**: ACM SIGKDD CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING
  https://www.kdd.org/kdd2020/

- **ICDM**: International Conference on Data Mining
  https://waset.org/data-mining-conference-in-july-2020-in-istanbul