Lecture 3: Graph Neural Networks

Relational Learning

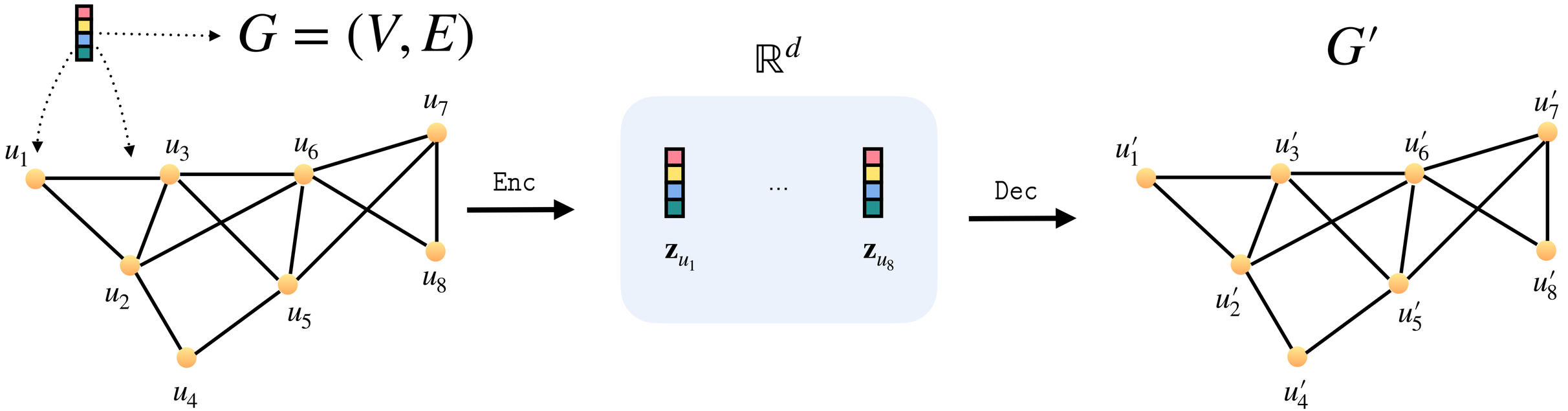
İsmail İlkan Ceylan

Overview of the Lecture

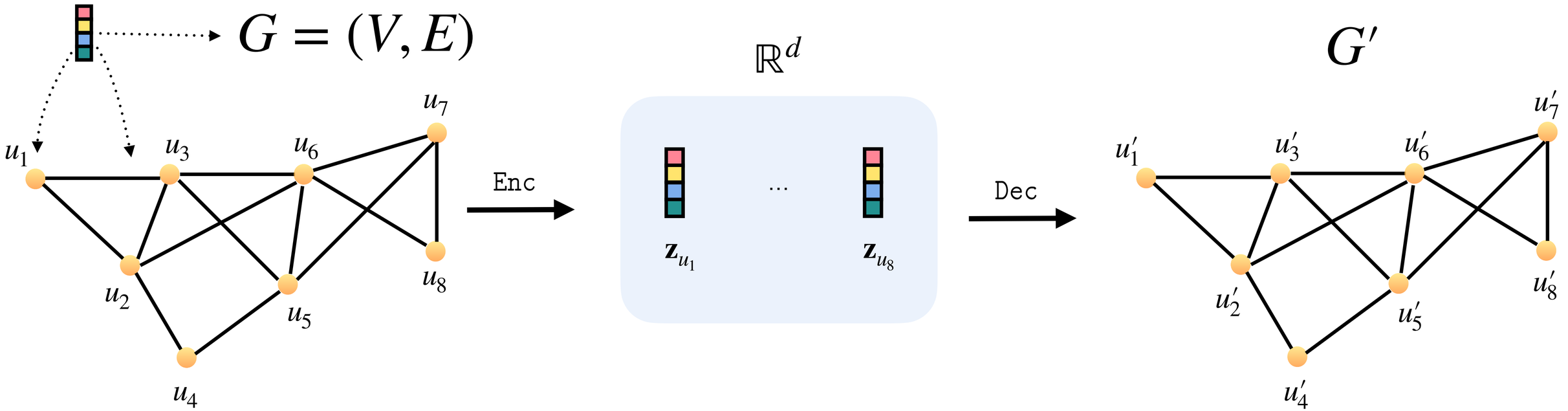
- From shallow to deep embeddings
- Traditional approaches to graph machine learning
- Relational inductive bias
- Message passing neural networks
- Graph representation learning tasks
  - Node-level property predictions
  - Graph-level property prediction
  - Edge-level property prediction
- Summary

From Shallow to Deep Embeddings

From Shallow to Deep Embeddings



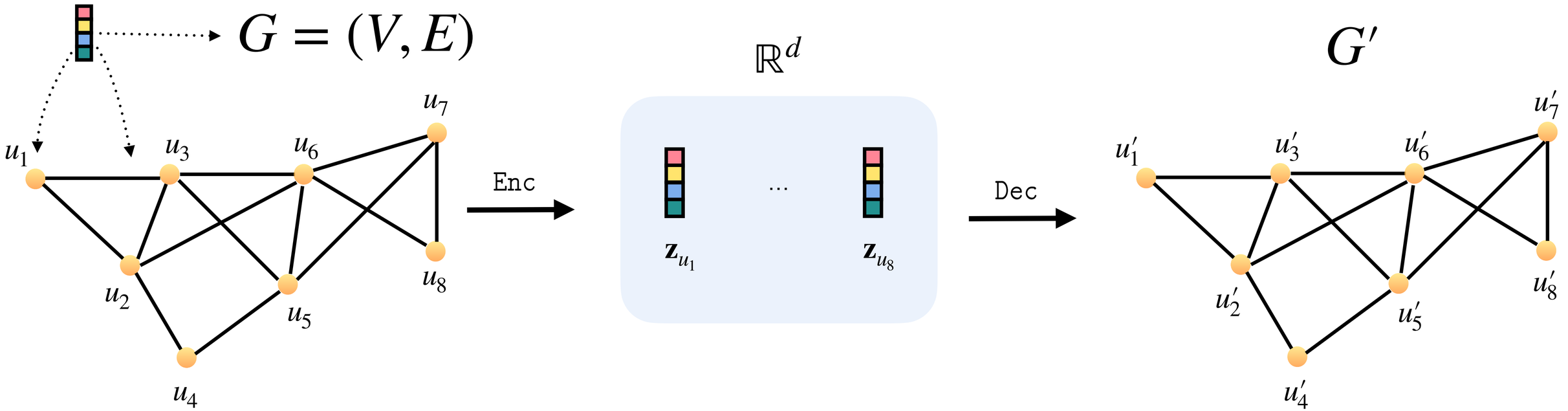
From Shallow to Deep Embeddings



Lecture 1 - 2: Learning with **knowledge graphs** (no features) using shallow embedding models.

Lecture 3 - 9: Learning with (mostly) **undirected graphs** + features using graph neural networks.

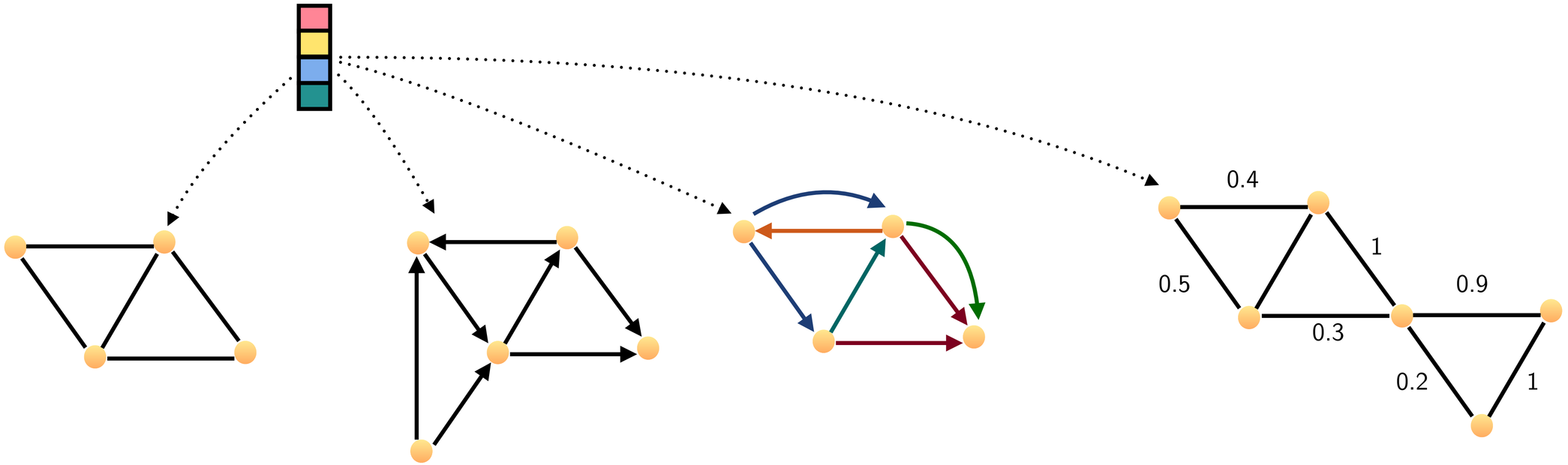
From Shallow to Deep Embeddings



Shallow embeddings are transductive: they do not apply to novel entities and are limited to single-graph tasks.

Graph neural networks are inductive: sophisticated embedding models that can generalize to novel data points.

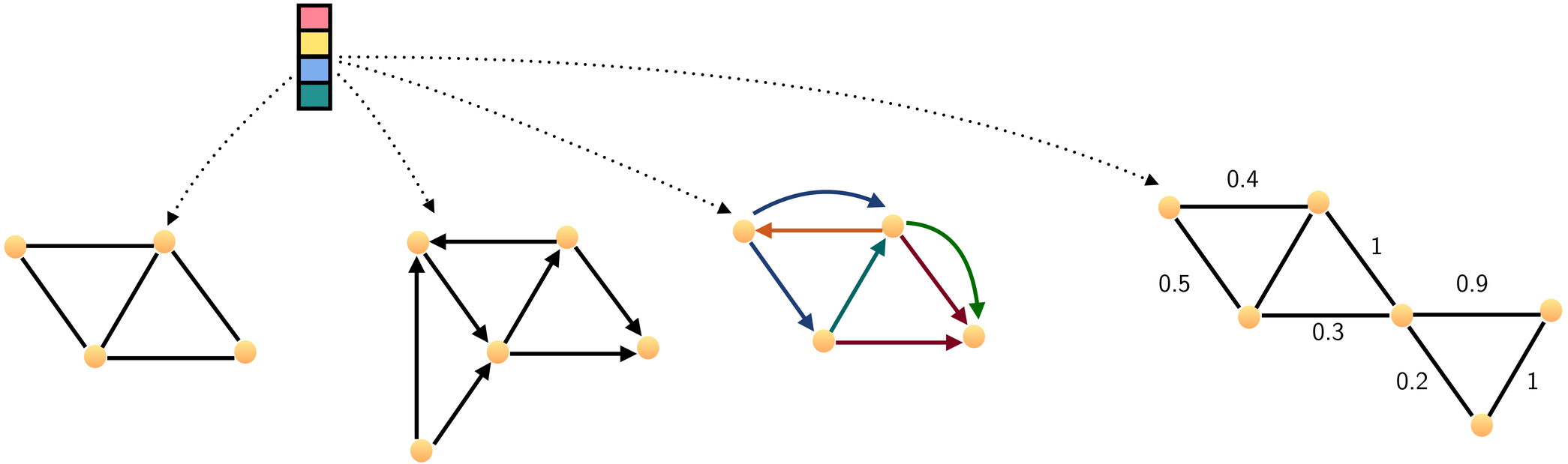
What Kind of Graphs?



The landscape of graphs is rich: Directed, undirected? Weighted graphs? Labelled (multi-relational) graphs? Node/edge features?

We focus on simple, undirected, unweighted, and unlabelled graphs, and assume deterministic node features.

What Kind of Graphs?



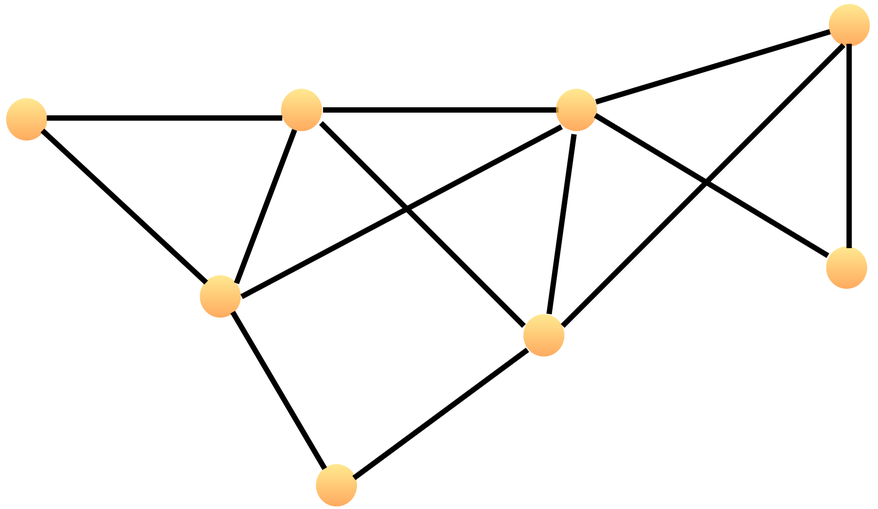
Graphs $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, where d is the embedding dimensionality and V_G denotes the set of vertices/nodes.

For each node u, we have a feature vector $\mathbf{x}_u$ which can be, e.g., domain-specific attributes, or node degrees, or simply one-hot encodings.

$\mathbf{A}^G$ is the adjacency matrix of a graph $G = (V, E)$; $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ the rows of $\mathbf{A}^G$; $\mathbf{A}^G_{[i,j]}$ entries of $\mathbf{A}^G$.

Traditional Approaches

Learning over Graphs

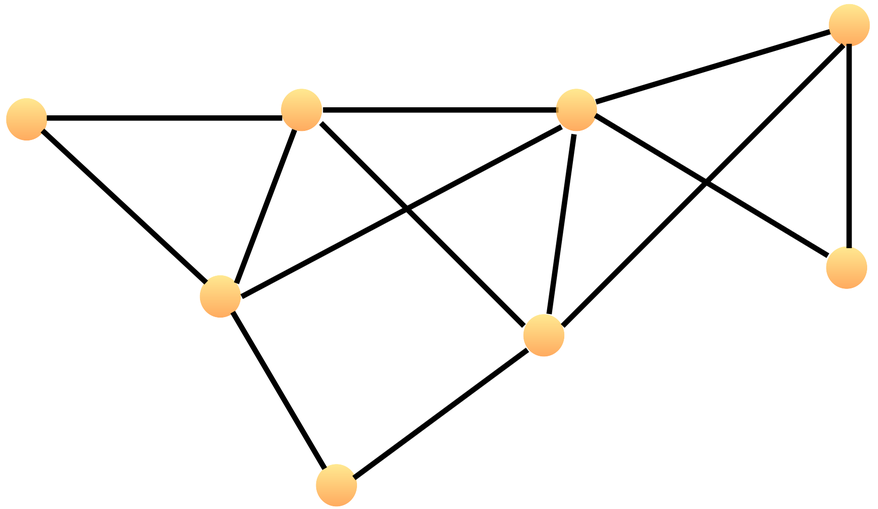


ML algorithms make assumptions about the data, e.g., the data points are independent and identically distributed (i.i.d.):

- independence: no need to model the dependencies,
- identical distribution: generalization guarantees possible to new/unseen data points.

These assumptions are unrealistic in the context of graphs.

Learning over Graphs

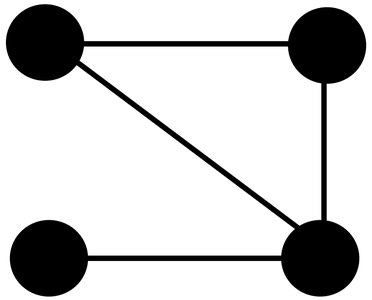


Classify the nodes in the given graph with respect to a certain property (i.e., node classification).

Properties depend on other nodes through edges, e.g., functions that rely on node statistics (e.g., #neighbors), or the overall graph structure (e.g., is the node part of a 5-cycle).

When learning functions over nodes, we cannot treat the nodes independently.

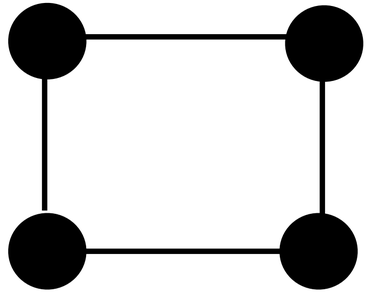
Learning over Graphs



Node degrees?

Contains an odd-length cycle?

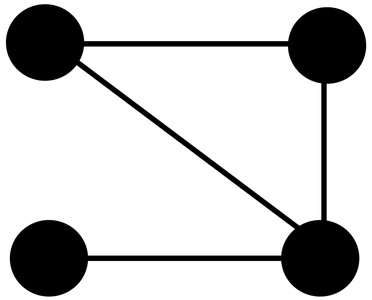
Minimum vertex cover size 1, 2?



Functions over graphs, or nodes, necessarily relate to graph properties, which carry valuable information: needs to be taken into account adequately.

Define similarity measures for nodes/graphs, and then use for the optimization task.

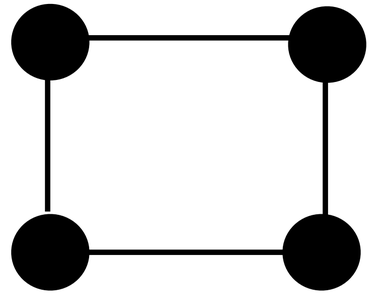
Learning over Graphs: Traditional Methods



Node degrees?

Contains an odd-length cycle?

Minimum vertex cover size 1, 2?



Traditional approaches to graph ML are based on:

- Extract node, edge, or graph-level statistics/features (indicating, e.g., node/edge/graph similarity),
- Using these features as input to standard machine learning classifiers.

Learning over Graphs: Node-Level Statistics

$$d_u = \sum_{v \in V_G} \mathbf{A}[u, v]$$

$$C_u = 3 \times \frac{|\ (v1,v2) \in E : v_1, v_2 \in N(u)\ |}{d_u^2}$$

Simplest node-level statistics given by the node-degrees - nodes with similar degree may be similar.

Another node-level statistics is the local clustering coefficient: ratio of triangles to connected triples.

Learning over Graphs: Graph-Level Statistics

$$D = \sum_{u,v \in V_G} \mathbf{A}[u, v]$$

$$C = 3 \times \frac{\#\text{triangles in the network}}{\#\text{connected triples of vertices}}$$

Any node-level statistic can be used as a graph-level statistics, by aggregating the node-level statistics.

Learning over Graphs: Graph Kernel Methods

Kernel methods: Learning by comparing pairs of data points using similarity measures - *kernels*.

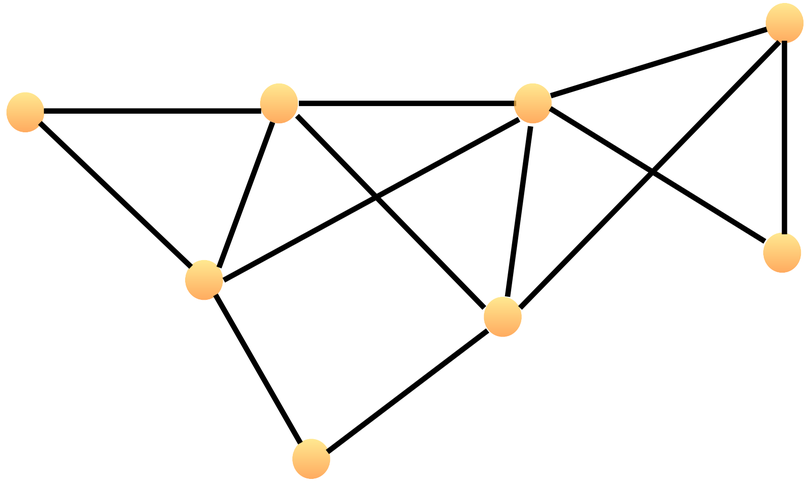
Popular graph similarity functions are studied under the name of graph kernel methods (Kriege et al., 2020):

- 1-dimensional Weisfeiler-Lehman (Weisfeiler and Leman, 1968)
- Shortest path (Borgwardt and Kriegel, 2005)
- Graphlet (Shervashidze et al., 2009)
- Weisfeiler-Lehman Subtree/Edge/Shortest Path (Shervashidze et al., 2011)

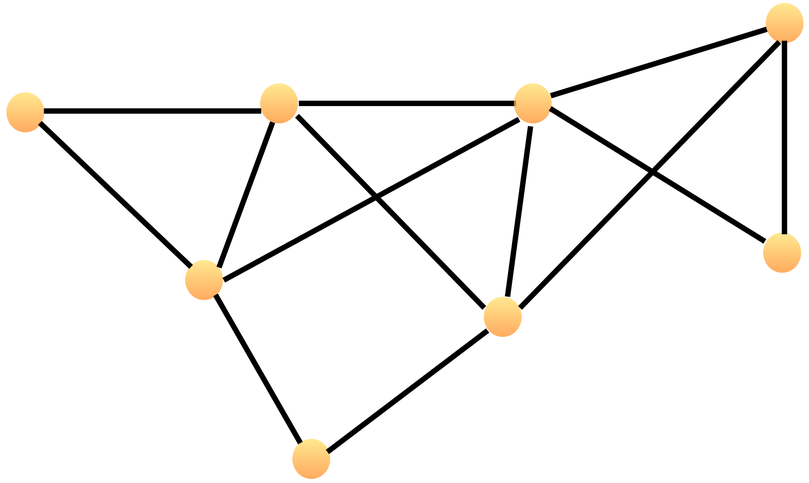
Modern deep learning approaches do not explicitly extract such statistics, but there are strong connections between modern graph representation learning and graph kernel methods, such as 1-WL!

Relational Inductive Bias

The Quest for a New Framework



The Quest for a New Framework

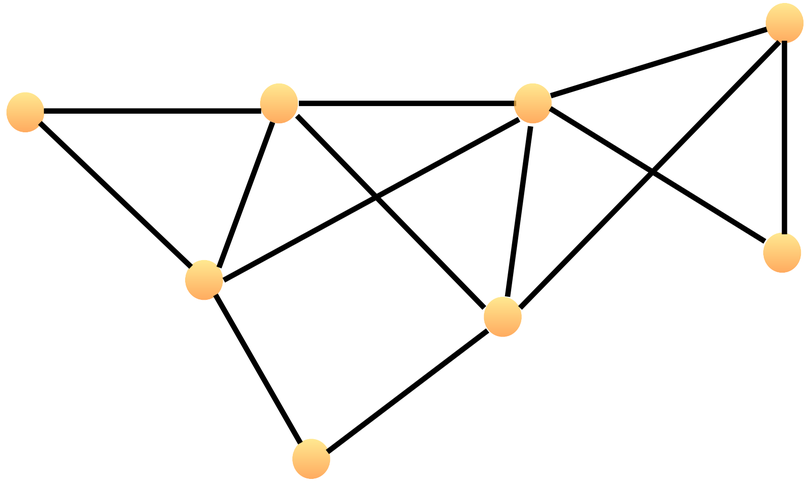


Example: Consider multi-layer perceptrons and embedding of a graph G as:

$$f(G) = \mathrm{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

The Quest for a New Framework



Example: Consider multi-layer perceptrons and embedding of a graph G as:

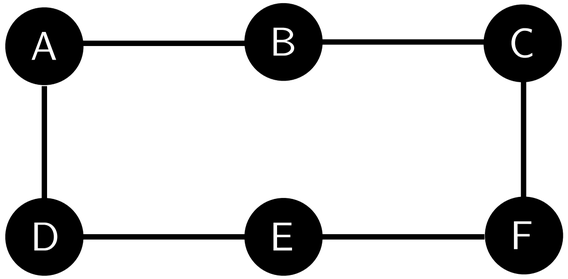
$$f(G) = \text{MLP}(\mathbf{A}^G_{[1]} \oplus \ldots \oplus \mathbf{A}^G_{[|V_G|]}),$$

where $\oplus$ is vector concatenation of the rows $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ of the adjacency matrix $\mathbf{A}^G$.

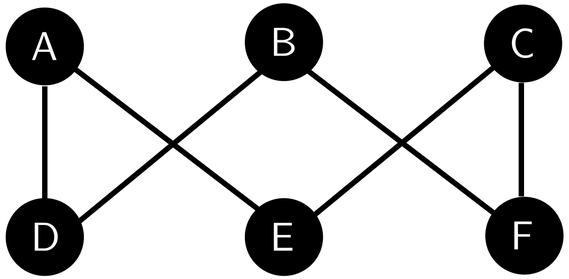
Problem: This depends on the ordering of nodes that we used in the adjacency matrix!

Invariance and Equivariance

G

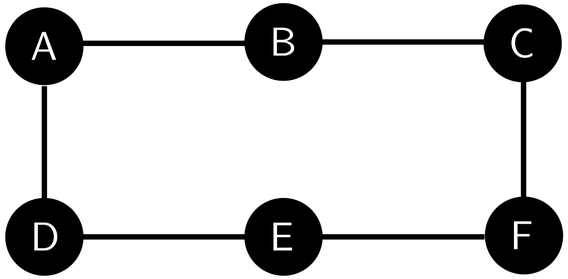


H

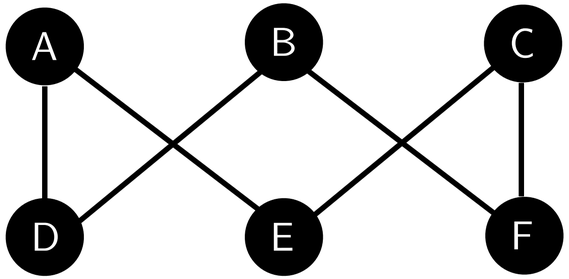


Invariance and Equivariance

G



H

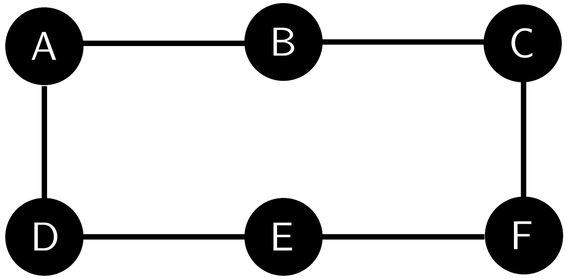


Invariance: A function $f : \mathcal{G} \to \mathbb{R}$ is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$, i.e., the function f does not depend on the ordering of the nodes in the graph.

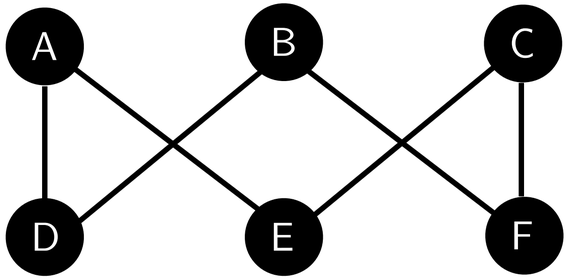
Equivariance: A function $f : \mathcal{G} \to \mathbb{R}^{V_G}$ is permutation-equivariant if for every permutation π of V_G, it holds that $f(G^\pi) = f(G)^\pi$, i.e., the output of f is permuted in a consistent way when we permute the nodes in the graph.

Invariance and Equivariance

G



H



We can also speak of invariance or equivariance for other kinds of functions.

Argument: These properties entail strong relational inductive bias! The goal is to develop a deep learning framework enhanced with these properties.

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data?

Discussion: Suppose we want to learn a rather simple permutation-invariant or equivariant function f:

- It is non-trivial to ensure, e.g., invariance to orderings, or even approximate this well in practice.
- Learning f will likely require longer training time...
- Learning f will likely require more training data, as, e.g., it needs more examples (of orderings) so as to learn invariance to them...
- Inductive bias is proven to be crucial, e.g., the use of convolutions which are translation-invariant.

Relational Inductive Bias: Pathfinding

Relational Inductive Bias: Investigate on a simple function, i.e., on a concrete task of pathfinding.

Background: Weston et al. (2015) proposed a collection of proxy tasks (bAbI) that are aimed at evaluating certain reasoning capabilities in the context of question answering. Li et al. (2016) transformed the "bAbI Task 19", a kind of pathfinding, into a symbolic form, and conducted experiments.

Pathfinding: We are given a set of connections:

$$\text{E s A}, \quad \text{B n C}, \quad \text{E w F}, \quad \text{B w E},$$

where "E s A" denotes A is reachable from E by going south.

The task is simple pathfinding on graphs defined over edge types s, n, e, w.

Relational Inductive Bias: Pathfinding

Pathfinding: How well do earlier deep learning models, e.g., LSTMs, perform on this problem?

Relational Inductive Bias: Pathfinding

Pathfinding: How well do earlier deep learning models, e.g., LSTMs, perform on this problem?

Results: Li et al. (2016) reports the empirical results relative to LSTMs and gated graph sequence neural networks (GGSNNs), i.e., a graph neural network model proposed in the same paper:

Relational Inductive Bias: Pathfinding

Pathfinding: How well do earlier deep learning models, e.g., LSTMs, perform on this problem?

Results: Li et al. (2016) reports the empirical results relative to LSTMs and gated graph sequence neural networks (GGSNNs), i.e., a graph neural network model proposed in the same paper:

LSTM	28.2 ± 1.3 with 950 training samples

Relational Inductive Bias: Pathfinding

Pathfinding: How well do earlier deep learning models, e.g., LSTMs, perform on this problem?

Results: Li et al. (2016) reports the empirical results relative to LSTMs and gated graph sequence neural networks (GGSNNs), i.e., a graph neural network model proposed in the same paper:

LSTM	28.2 ± 1.3 with 950 training samples
GGSNN	71.1 ± 14.7 with 50 training samples
	92.5 ± 5.9 with 100 training samples
	99.0 ± 1.1 with 250 training samples

Relational Inductive Bias: Pathfinding

Pathfinding: How well do earlier deep learning models, e.g., LSTMs, perform on this problem?

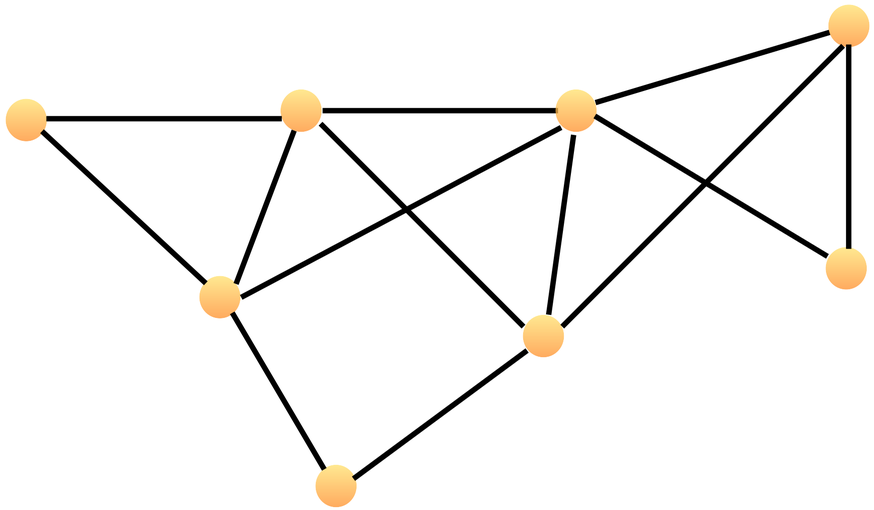
Results: Li et al. (2016) reports the empirical results relative to LSTMs and gated graph sequence neural networks (GGSNNs), i.e., a graph neural network model proposed in the same paper:

LSTM	28.2 ± 1.3 with 950 training samples
GGSNN	71.1 ± 14.7 with 50 training samples
	92.5 ± 5.9 with 100 training samples
	99.0 ± 1.1 with 250 training samples

The relational inductive bias helps, both in terms of accuracy, and in the #samples needed.

Message Passing Neural Networks

Message Passing Neural Networks

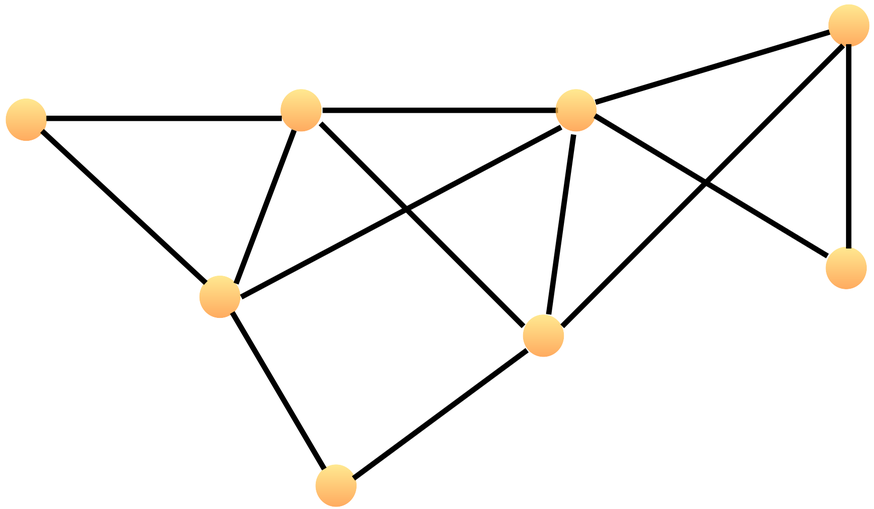


Message passing neural networks (MPNNs) capture popular GNNs (Gilmer et al., 2017).

Idea: Start with initial node features and update them with the information received from their respective neighborhoods (i.e., message passing) for k iterations, yielding final node representations.

Notation: Let the representation of a node $u \in V$ at iteration t be $\mathbf{h}_u^{(t)}$, i.e., $\mathbf{h}_u^{(0)}$ is the initial representation.

Message Passing Neural Networks



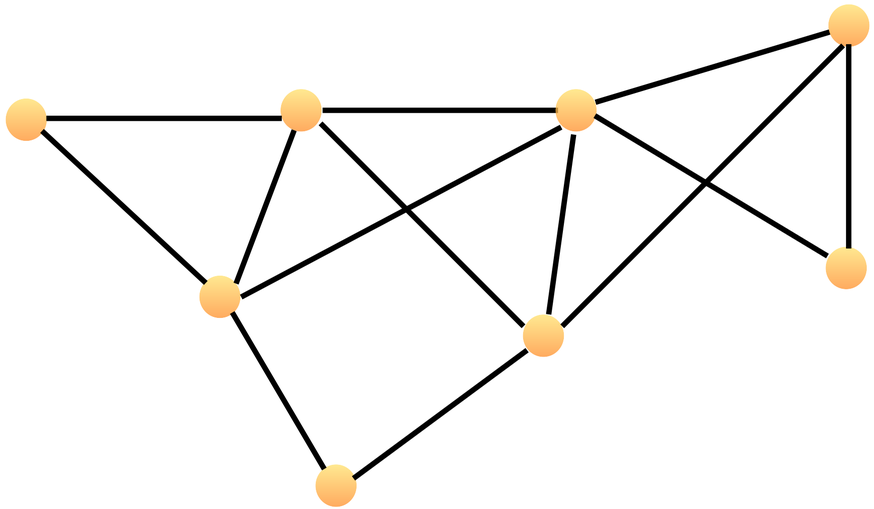
Initialization: A MPNN defines, for every node $u \in V$, an initial representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

Message passing: The representation $\mathbf{h}_u$ for each node $u \in V$ is then iteratively updated as:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right),$$

where $aggregate^{(t)}$ and $combine^{(t)}$ are differentiable functions (i.e., neural networks), and $aggregate^{(t)}$ is a permutation-invariant function (e.g., mean, sum, or max).

Message Passing Neural Networks



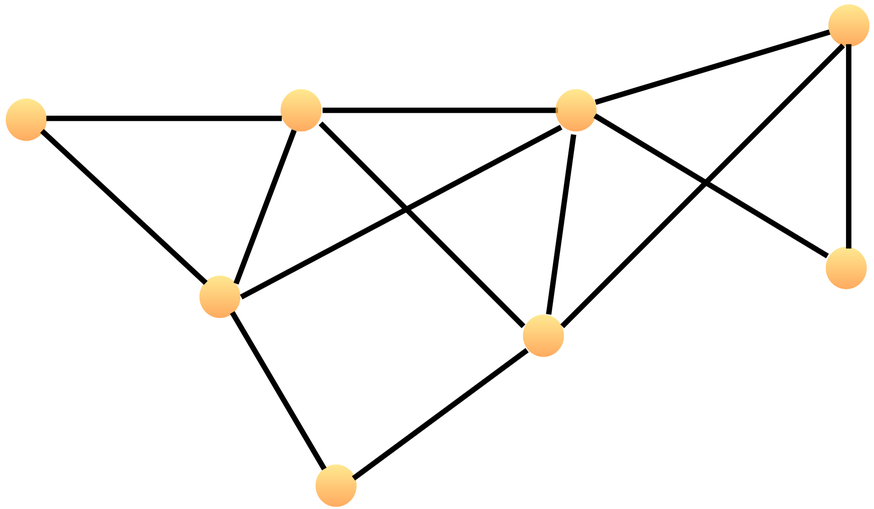
MPNN framework can yield homogeneous or non-homogenous models based on:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right)$$

Non-homogeneous: Different functions at different layers, e.g., $aggregate^{(t)}$ and $aggregate^{(t-1)}$ can differ.

Homogeneous: *aggregate* and *combine* are the same across all layers, superscripts dropped.

Message Passing Neural Networks

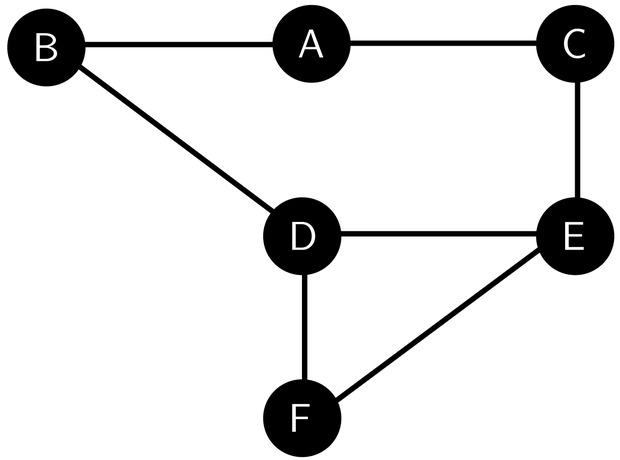


Node-level final representation: The final node representations are denoted as $\mathbf{z}_u = \mathbf{h}_u^{(k)}$.

Graph-level final representation: Define a final graph embedding $\mathbf{z}_G = \mathbf{h}_G^{(k)}$ for a graph G through a mapping from the set of all the node embeddings $\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}$ to $\mathbf{z}_G$:

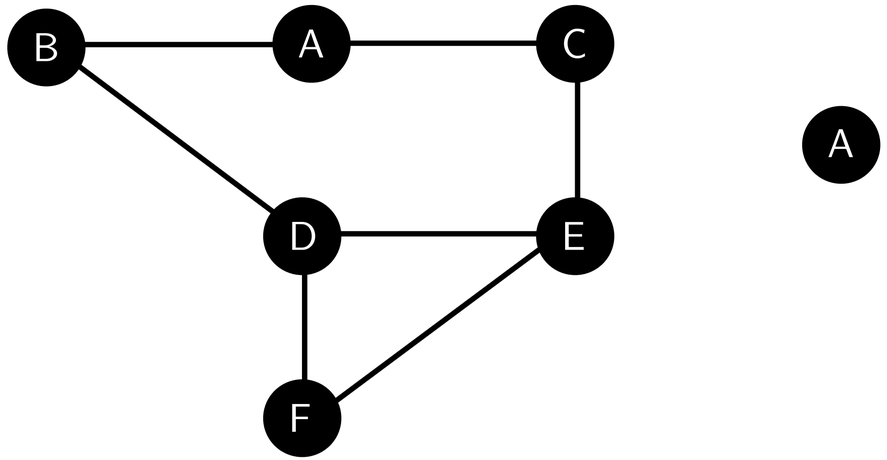
- Common choices are sum, or mean, which are then normalized with respect to, e.g., the size of the nodes.
- There are various methods for relational pooling (Murphy et al., 2019).

Message Passing Neural Networks



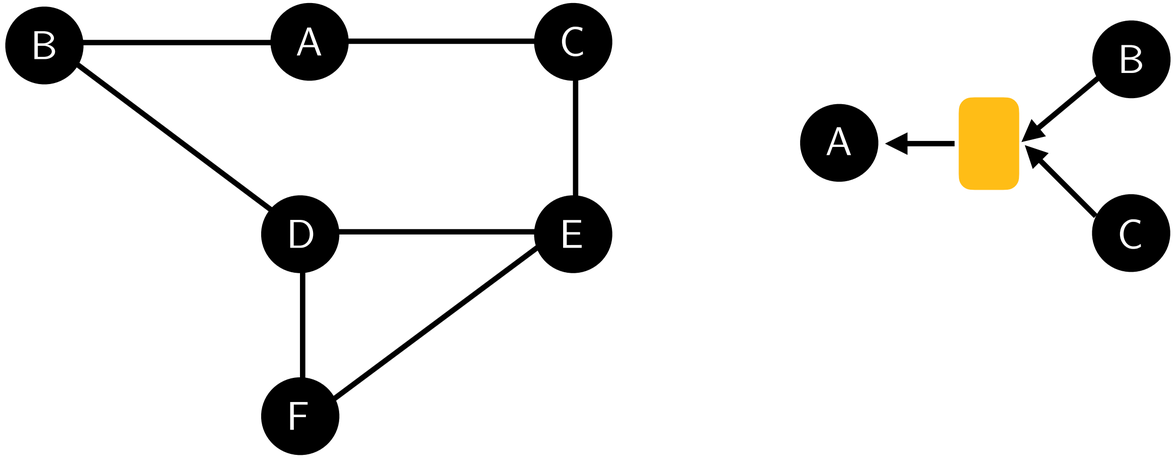
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



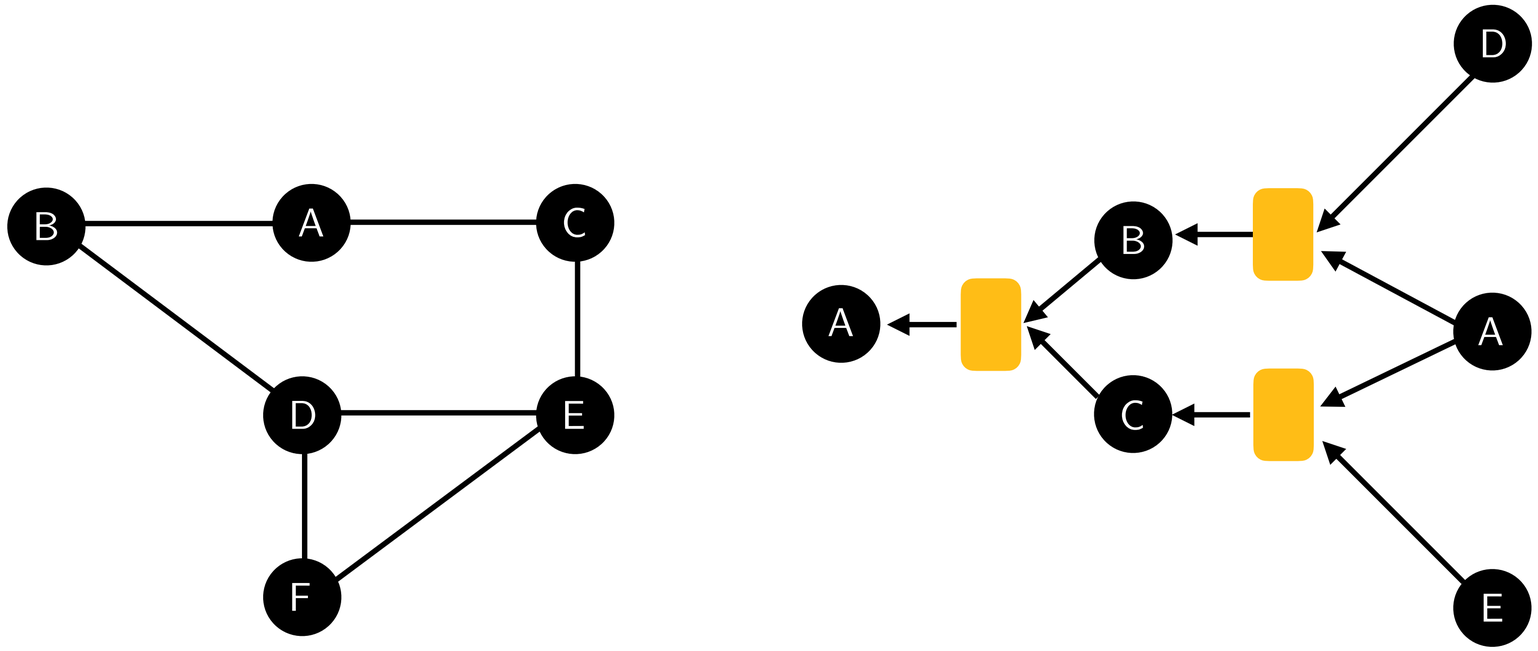
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



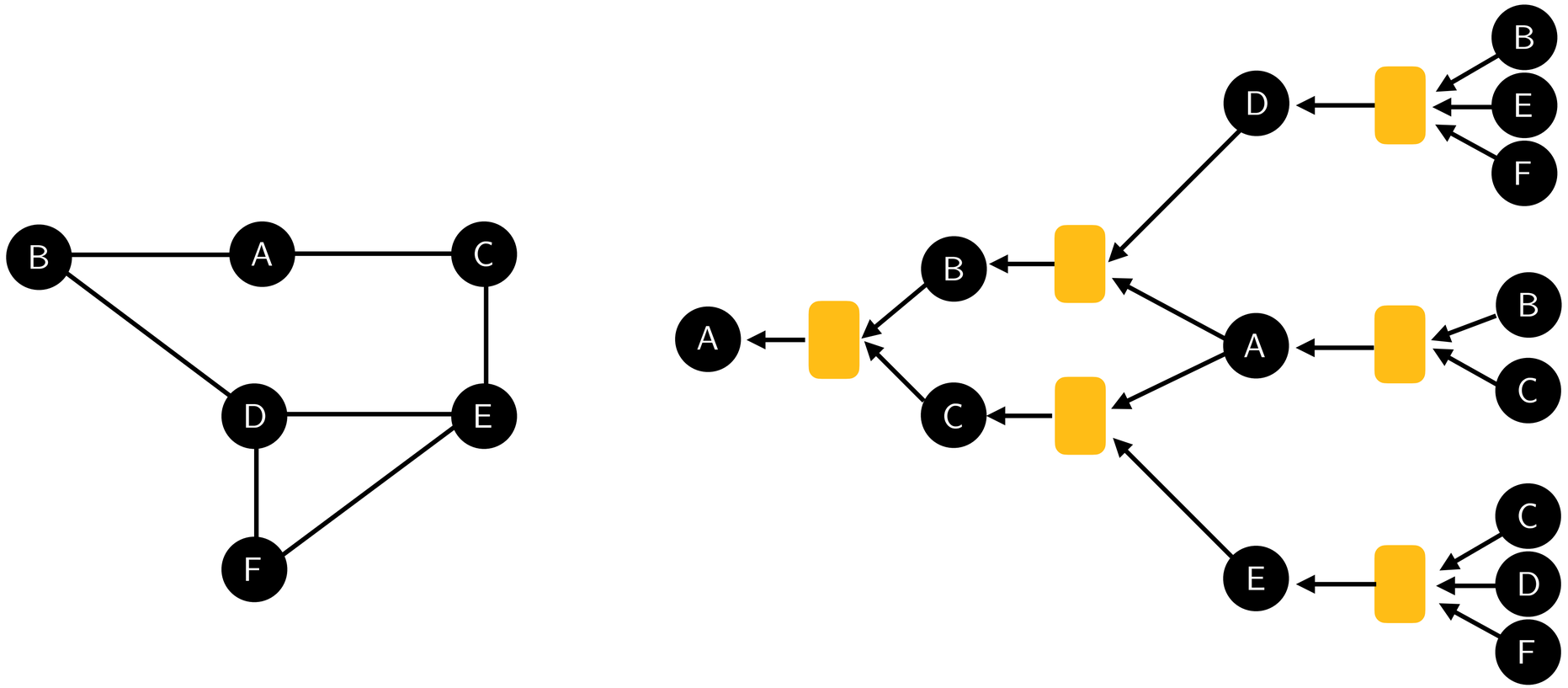
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



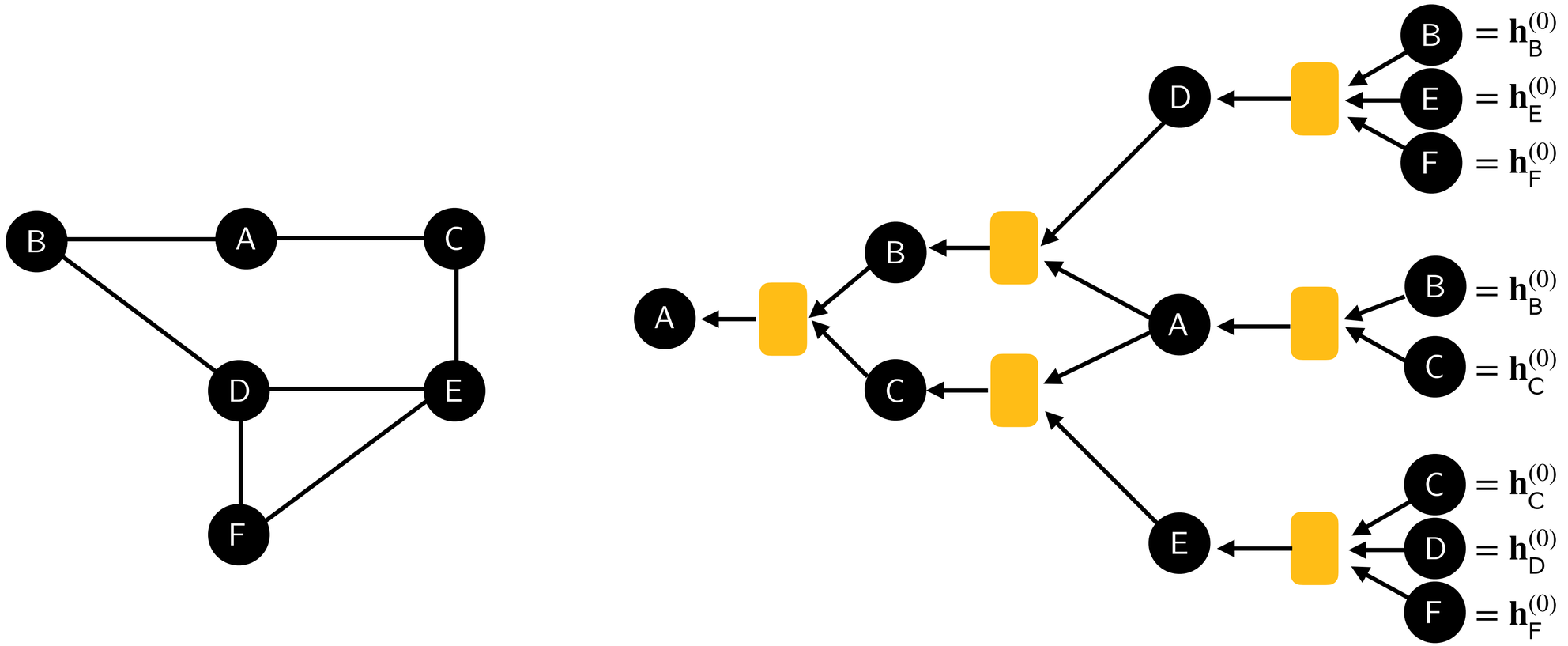
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



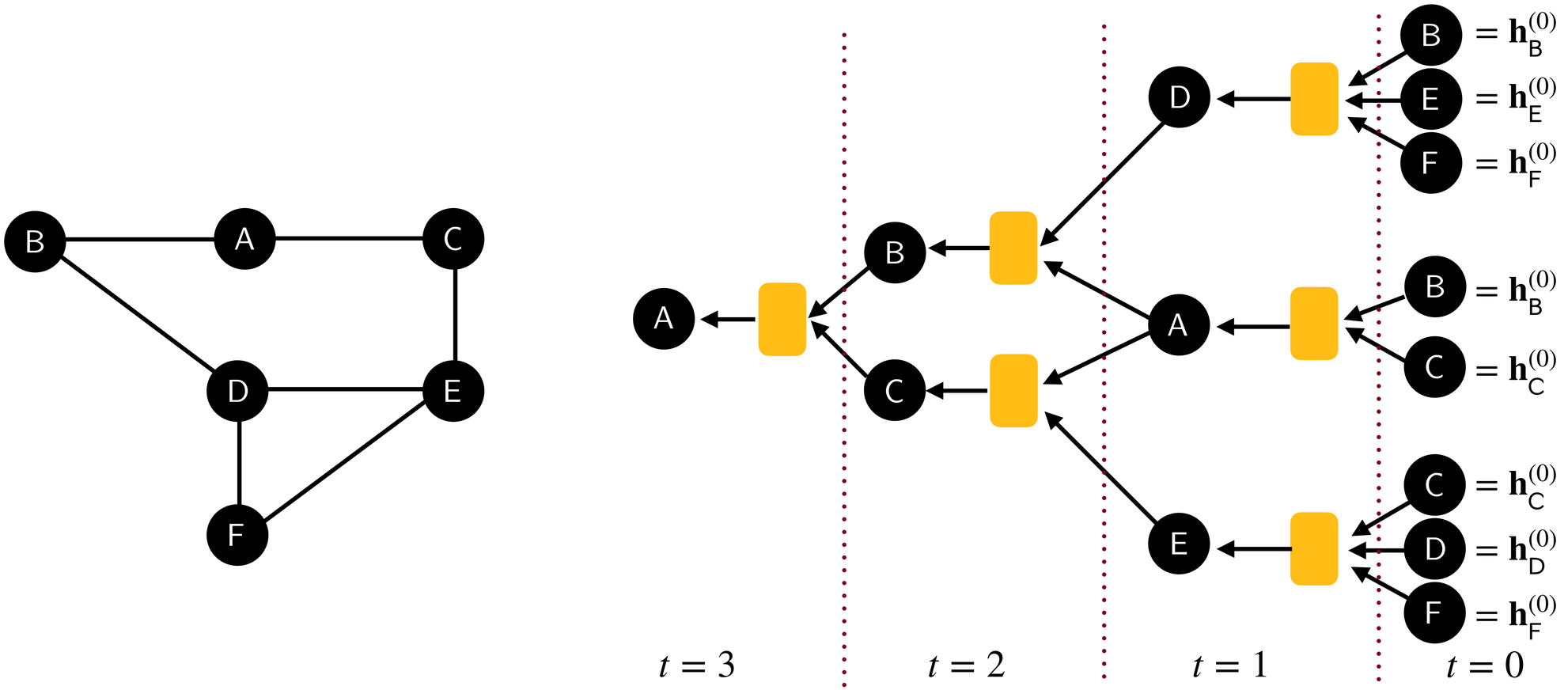
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



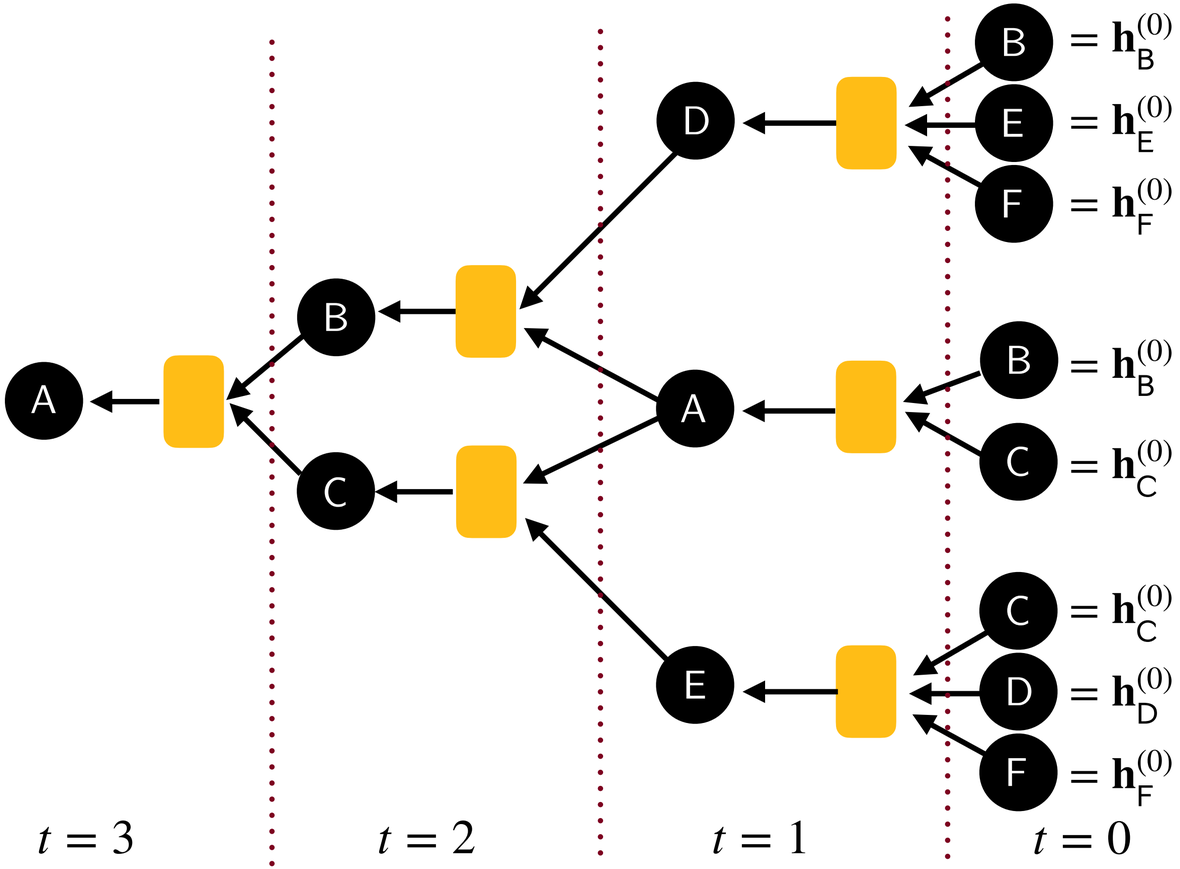
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks

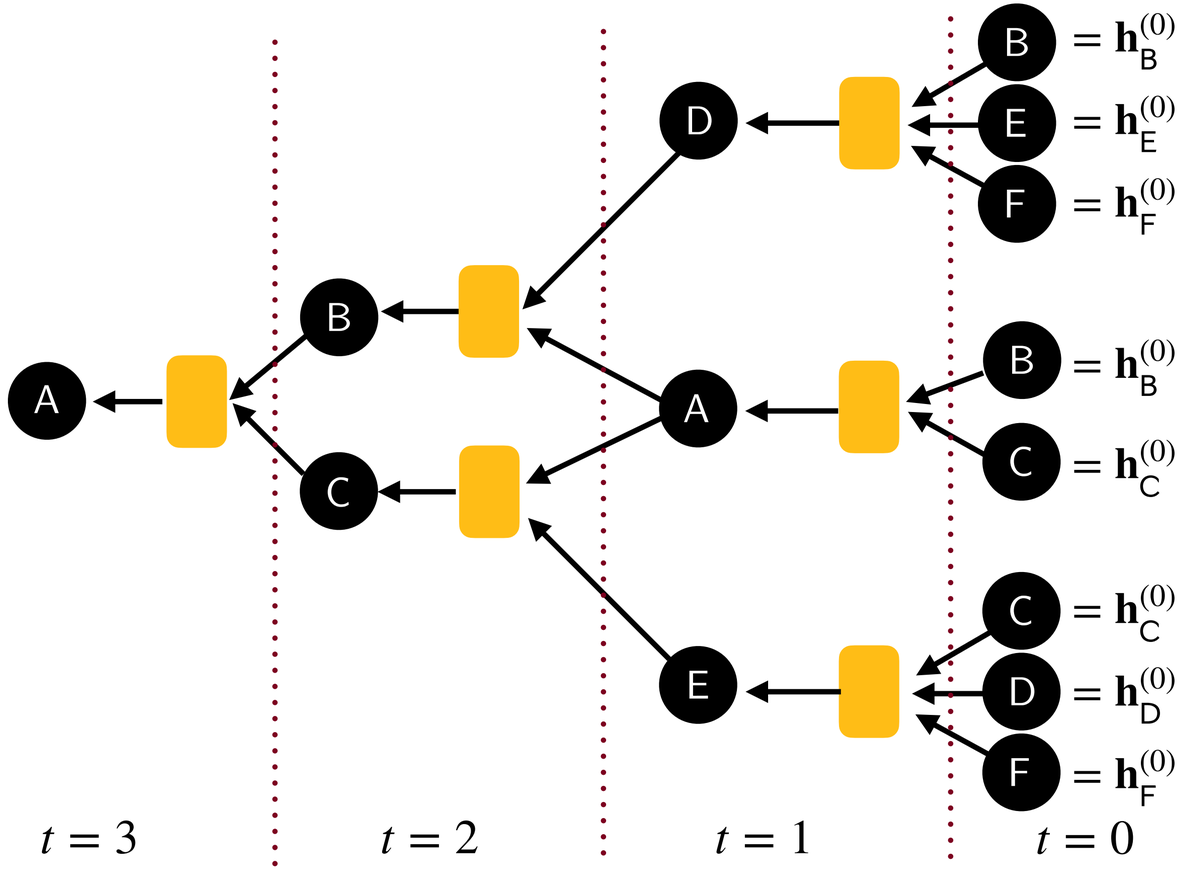


A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least 3 iterations are needed to get information from all nodes, i.e., F will not pass any messages to A with $k = 2$.

Message Passing Neural Networks



Message Passing Neural Networks



The i-th iteration is the i-th layer of the MPNN, since each iteration can be seen as an "unrolling" of the network. The #layers defines the depth, and the embedding dimensionality the width of the network.

Deriving a Basic Graph Neural Network Model

Deriving a Basic Graph Neural Network Model

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right)$$

Deriving a Basic Graph Neural Network Model

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right)$$

$$= combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

Deriving a Basic Graph Neural Network Model

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right)\right)$$

$$= combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)$$

$$= \sigma\left(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\right)\right)$$

The Basic Graph Neural Network Model

The basic graph neural network model updates the representations as:

$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}_{self}^{(t)} \mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)} + \mathbf{b}^{(t)} \right)$$

$\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)}$: trainable parameter matrices

σ: an element-wise non-linear function (e.g., ReLU),

$\mathbf{b}^{(t)} \in \mathbb{R}^{d(t)}$: a bias term (which we will omit).

Message Passing With Self-Loops

Message Passing With Self-Loops

What if we unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes?

We can define an aggregate function which also aggregates over the node itself:

$$\mathbf{h}_u^{(t)} = aggregate^{(t)}\left(\left\{ \mathbf{h}_v^{(t-1)} \mid v \in N(u) \right\} \cup \left\{ \mathbf{h}_u^{(t-1)} \right\} \right)$$

Message Passing With Self-Loops

What if we unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes?

We can define an aggregate function which also aggregates over the node itself:

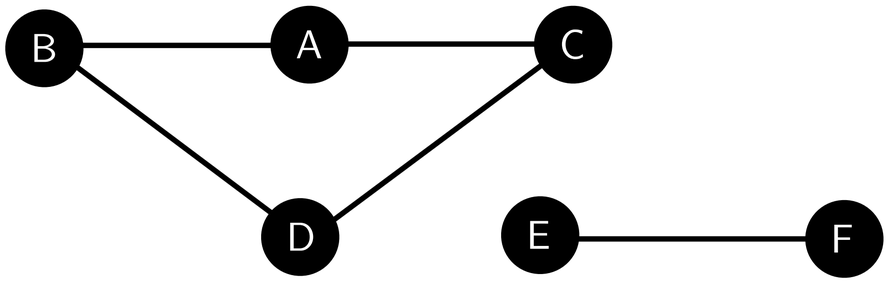
$$\mathbf{h}_u^{(t)} = aggregate^{(t)}\left(\left\{ \mathbf{h}_v^{(t-1)} \mid v \in N(u) \right\} \cup \left\{ \mathbf{h}_u^{(t-1)} \right\} \right)$$

This simplifies the base model:

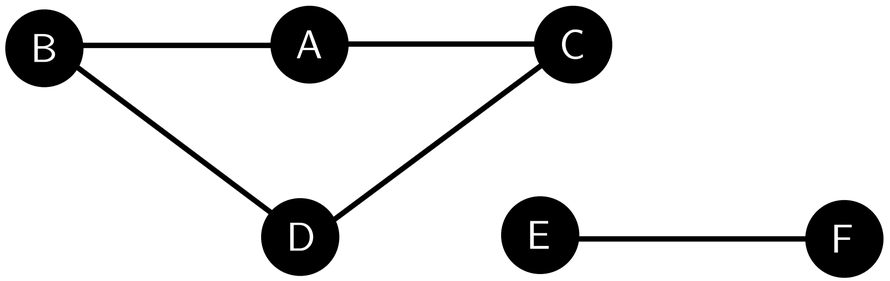
$$\mathbf{h}_u^{(t)} = \sigma\left(\mathbf{W}^{(t)} \sum_{v \in N(x)} \mathbf{h}_v^{(t-1)} + \mathbf{h}_u^{(t-1)} \right)$$

This limits the expressivity of the MPNN: the information coming from the node's neighbor's cannot be differentiated from the information from the node itself.

A Limitation of Message Passing



A Limitation of Message Passing



Problem: The presented message passing approach is local: no information flows across disjoint subgraphs.

Remark: A graph embedding is global since it is composed of all nodes, but during message passing there are still no communication between disjoint subgraphs and the node embeddings are "blind" to other embeddings in disjoint subgraphs.

Solution: Global feature computation, or global readout, on each layer of the MPNN (Battaglia et al., 2018).

Message Passing with Global Readout

Message passing with global readout: The representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right), read^{(t)}\left(\left\{\mathbf{h}_w^{(t-1)} \mid w \in G\right\}\right)\right),$$

where $read^{(t)}$ is a differentiable function. Similarly to $aggregate^{(t)}$, $read^{(t)}$ is permutation-invariant by construction, and all aggregate functions are typical candidates also for $read^{(t)}$.

Message Passing with Global Readout

Message passing with global readout: The representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$\mathbf{h}_u^{(t)} = combine^{(t)}\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right), read^{(t)}\left(\left\{\mathbf{h}_w^{(t-1)} \mid w \in G\right\}\right)\right),$$

where $read^{(t)}$ is a differentiable function. Similarly to $aggregate^{(t)}$, $read^{(t)}$ is permutation-invariant by construction, and all aggregate functions are typical candidates also for $read^{(t)}$.

Battaglia et al., (2018) defines a generalized message passing framework for relational reasoning over graph representations, and message passing with global readout can be seen as a special case of this framework.

This reformulation makes an important difference in the expressive power of MPNNs (Barcelo et al., 2020).

Generalized Message Passing

Generalized message passing (Battaglia et al., 2018): A message passing protocol that takes into account the internal representations for edges, nodes, and the graph, respectively:

$$\mathbf{h}_{(u,v)}^{(t)} = combine_e\left(\mathbf{h}_{(u,v)}^{(t-1)}, \mathbf{h}_u^{(t-1)}, \mathbf{h}_v^{(t-1)}, \mathbf{h}_G^{(t-1)}\right),$$

$$\mathbf{h}_u^{(t)} = combine_n\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right), \mathbf{h}_G^{(t-1)}\right),$$

$$\mathbf{h}_G^{(t)} = combine_G\left(\mathbf{h}_G^{(t-1)}, \left\{\mathbf{h}_u^{(t)} \mid u \in V_G\right\}, \left\{\mathbf{h}_{(u,v)}^{(t)} \mid (u,v) \in E\right\}\right),$$

where, at each iteration, each update happens in the given equation order.

Generalized Message Passing

Generalized message passing (Battaglia et al., 2018): A message passing protocol that takes into account the internal representations for edges, nodes, and the graph, respectively:

$$\mathbf{h}_{(u,v)}^{(t)} = combine_e\left(\mathbf{h}_{(u,v)}^{(t-1)}, \mathbf{h}_u^{(t-1)}, \mathbf{h}_v^{(t-1)}, \mathbf{h}_G^{(t-1)}\right),$$

$$\mathbf{h}_u^{(t)} = combine_n\left(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\left(\left\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\right\}\right), \mathbf{h}_G^{(t-1)}\right),$$

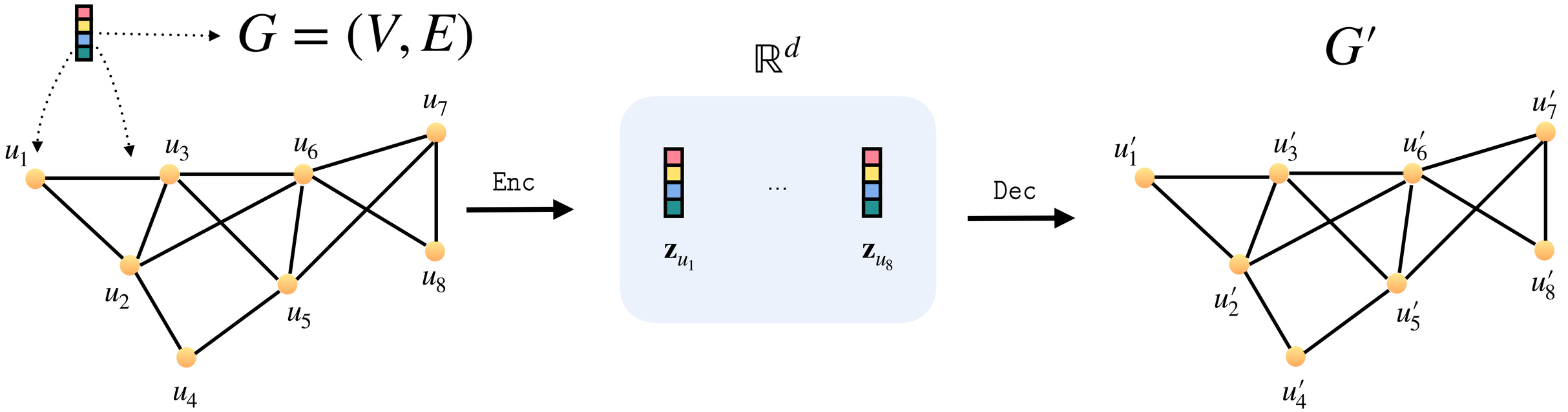
$$\mathbf{h}_G^{(t)} = combine_G\left(\mathbf{h}_G^{(t-1)}, \left\{\mathbf{h}_u^{(t)} \mid u \in V_G\right\}, \left\{\mathbf{h}_{(u,v)}^{(t)} \mid (u,v) \in E\right\}\right),$$

where, at each iteration, each update happens in the given equation order.

Generate embeddings $\mathbf{h}_{(u,v)}$ for each edge $(u,v) \in G$ as well as an embedding $\mathbf{h}_G$ for the entire graph: more easily integrate edge and graph-level features, especially in the multi-relational context.

Graph Representation Learning Tasks

Encoder-Decoder



The learned embeddings can be used for any standard machine learning task, e.g., classification, regression, clustering, and so they have natural counterparts both on node and graph-level tasks.

Node-Level Tasks

Node-Level Tasks

Node classification: Given a graph $G = (V, E)$, where a subset of the nodes $\{(u, y_u) \mid u \in V_{tr} \subset V\}$ are labeled with a class, predict the labels of the remaining nodes, i.e., test nodes in the graph, i.e., y_v for all $v \in V \backslash V_{tr}$.

Node-Level Tasks

Node classification: Given a graph $G = (V, E)$, where a subset of the nodes $\{(u, y_u) \mid u \in V_{tr} \subset V\}$ are labeled with a class, predict the labels of the remaining nodes, i.e., test nodes in the graph, i.e., y_v for all $v \in V \backslash V_{tr}$.

Example (Kipf and Welling, 2017): Citeseer is a citation network, where nodes represent papers, and edges denote citation links, and a subset of the nodes are labelled with a paper category (e.g., AI, ML). The task is to predict the category (or, categories) of the remaining papers.

Node-Level Tasks

Node classification: Given a graph $G = (V, E)$, where a subset of the nodes $\{(u, y_u) \mid u \in V_{tr} \subset V\}$ are labeled with a class, predict the labels of the remaining nodes, i.e., test nodes in the graph, i.e., y_v for all $v \in V \backslash V_{tr}$.

Example (Kipf and Welling, 2017): Citeseer is a citation network, where nodes represent papers, and edges denote citation links, and a subset of the nodes are labelled with a paper category (e.g., AI, ML). The task is to predict the category (or, categories) of the remaining papers.

Training: Based on an appropriate loss function, e.g., negative log-likelihood loss:

$$\mathscr{L} = \sum_{u \in V_{tr}} - \log(\text{softmax}(\mathbf{z}_u, \mathbf{y}_u)),$$

where $\mathbf{y}_u$ is a one-hot vector indicating the class y_u of the training node $u \in V_{tr}$, and the $\text{softmax}(\mathbf{z}_u, \mathbf{y}_u)$ function denotes the predicted probability that the node u belongs to the class $\mathbf{y}_u$.

Node-Level Tasks

Node classification: Given a graph $G = (V, E)$, where a subset of the nodes $\{(u, y_u) \mid u \in V_{tr} \subset V\}$ are labeled with a class, predict the labels of the remaining nodes, i.e., test nodes in the graph, i.e., y_v for all $v \in V \backslash V_{tr}$.

Example (Kipf and Welling, 2017): Citeseer is a citation network, where nodes represent papers, and edges denote citation links, and a subset of the nodes are labelled with a paper category (e.g., AI, ML). The task is to predict the category (or, categories) of the remaining papers.

Training: Based on an appropriate loss function, e.g., negative log-likelihood loss:

$$\mathscr{L} = \sum_{u \in V_{tr}} - \log(\text{softmax}(\mathbf{z}_u, \mathbf{y}_u)),$$

where $\mathbf{y}_u$ is a one-hot vector indicating the class y_u of the training node $u \in V_{tr}$, and the $\text{softmax}(\mathbf{z}_u, \mathbf{y}_u)$ function denotes the predicted probability that the node u belongs to the class $\mathbf{y}_u$.

Node regression, or **clustering** (i.e., community detection) are other node-level tasks.

Node Classification: Inductive vs Transductive

Node Classification: Inductive vs Transductive

Test nodes: The given loss function uses only nodes from the training set V_{tr} , but test nodes $V_{test} = V \backslash V_{tr}$ may still be observed during training, i.e., they can take part in the message passing.

This yields two regimes (Hamilton, 2020):

- Transductive: All nodes, including test nodes, are observed during training, i.e., their representations are computed during message passing and they also affect the representation of other nodes.
- Inductive: Not all test nodes are observed during training. For some test nodes, neither the nodes themselves nor their edges (and hence their relation to other nodes) are known.

Node Classification: Inductive vs Transductive

Test nodes: The given loss function uses only nodes from the training set V_{tr} , but test nodes $V_{test} = V \backslash V_{tr}$ may still be observed during training, i.e., they can take part in the message passing.

This yields two regimes (Hamilton, 2020):

- Transductive: All nodes, including test nodes, are observed during training, i.e., their representations are computed during message passing and they also affect the representation of other nodes.
- Inductive: Not all test nodes are observed during training. For some test nodes, neither the nodes themselves nor their edges (and hence their relation to other nodes) are known.

Transductive classification example: Access to the full citation graph at training time and define a subset of the nodes as the test nodes.

Inductive classification example: Access to only a subgraph of the citation graph at training time in and define a set of nodes from a disjoint subgraph as test nodes.

Node Classification: Supervised or Semi-supervised?

Node Classification: Supervised or Semi-supervised?

Transductive node classification can be viewed as a semi-supervised learning task:

- We train using training labels: standard in supervised learning
- We additionally have access to the structural information of unlabelled test nodes

We trained using both labelled and unlabelled data.

Node Classification: Supervised or Semi-supervised?

Transductive node classification can be viewed as a semi-supervised learning task:

- We train using training labels: standard in supervised learning
- We additionally have access to the structural information of unlabelled test nodes

We trained using both labelled and unlabelled data.

While semi-supervised is more appropriate for transductive node classification, the standard semi-supervised setting also requires i.i.d. assumption, which doesn't hold for node classification.

Inductive node classification can be seen as supervised learning.

Machine learning on graphs tends to differ from standard practices and terminology in machine learning.

Edge-Level Tasks

Link prediction and **knowledge graph completion** are the most fundamental edge-level tasks.

Remark: KG completion refers to relation prediction, whereas link prediction applies to single-relational graphs, though these terms are sometimes used interchangeable when the context is clear.

Example (Link Prediction): OGBL-DDI (Hu et al., 2020) is a drug-drug interaction network: node's represent drugs and edges interactions between drugs. The task is to predict drug-drug interactions given information on already known drug-drug interactions, i.e., to rank true drug interactions higher than non-interacting drug pairs.

Example (KG completion): OGBL-BIOKG and OGBL-WIKIKG2 (Hu et al., 2020) are KGs proposed as part of OGB. The evaluation protocol is similar to standard KG evaluation, but splits are fixed for OGBL-WIKIKG2.

Training: Pairwise node embedding loss function to decode edges, e.g., one of shallow embedding models.

KG completion requires dedicated GNN architectures with better relational inductive bias.

Graph-Level Tasks

Graph classification: Given a set of graphs $\mathcal{G} = \{G_1, \ldots G_n\}$, where a subset of the graphs $\{(G_i, y_{G_i}) \mid G_i \in \mathcal{G}_{tr} \subset \mathcal{G}\}$ are labeled with a class, predict the labels of the remaining (test) graphs.

Example (Graph classification): IMDB (Morris et al., 2020) consist of the so-called ego-networks for each movie, and contains information such as actor collaborations for each movie. The task is to predict the genre (e.g., action, horror) of the movie.

Example (Graph classification): OGBG-MOLHIV (Hu et al., 2020) is a molecular property prediction dataset: each graph represents a molecule, where nodes are atoms, and edges are chemical bonds. The task is to predict the target molecular properties, e.g., whether a molecule inhibits HIV virus replication or not.

Training is similar to node classification, except that we use the final embedding of the graphs instead, e.g.:

$$\mathcal{L} = \sum_{G \in \mathcal{G}_{tr}} - \log(\text{softmax}(\mathbf{z}_G, \mathbf{y}_G))$$

Graph-Level Tasks

Graph classification is a supervised learning task since each graph is an i.i.d. data point associated with a label, and the goal is to use a labeled set of graphs to learn a mapping from graphs to class labels.

Graph clustering and graph regression are other graph-level supervised learning tasks.

Example (Graph regression): QM9 is a molecular dataset, where each graph represents the structure of a molecule (Gilmer et al., 2017), and the task is to predict that molecule's toxicity or solubility.

Training for graph regression can be done in various ways, e.g., by minimizing a squared-error loss between a target value $y_G \in \mathbb{R}$ and the predicted value, e.g., $MLP(\mathbf{z}_G)$, for each graph G.

Graph generation is a different graph-level tasks which falls under "generative models":

Example (Graph generation): Generate novel molecules which could be candidates for novel drugs!

We will discuss graph generation in more detail later in the course.

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions
- Graph representation learning tasks
- We have not covered any concrete model beyond the very basic one: Lecture 4!
- Additional reading material: This lecture is partially based on Chapters 5 - 7 of (Hamilton, 2020)

References

- B. Weisfeiler and A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 1968.
- Borgwardt, KM, Kriegel HP. Shortest-path kernels on graphs In: IEEE International Conference on Data Mining, 2005.
- Shervashidze, N, Vishwanathan SVN, Petri TH, Mehlhorn K, Borgwardt KM. Efficient graphlet kernels for large graph comparison. AISTATS, 2009.
- Shervashidze, N, Schweitzer P, van Leeuwen EJ, Mehlhorn K, Borgwardt KM. Weisfeiler-Lehman graph kernels. JMLR, 2011
- N. Kriege, F. Johansson, and C. Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 2020.
- J. Weston, A. Bordes, S. Chopra, and T. Mikolov. Towards AI-complete question answering: a set of prerequisite toy tasks. arXiv preprint arXiv:1502.05698, 2015.
- J. Gilmer, S.S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. 2017. Neural message passing for Quantum chemistry. *ICML*, 2017.
- C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *ICML Workshop on Graph Representation Learning and Beyond*, 2020.
- R.L.Murphy, B. Srinivasan, V.A. Rao, and B. Ribeiro, Relational Pooling for Graph Representations. *ICML*, 2019.

References

- Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vin´ıcius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulc¸ehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish ¨ Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. CoRR, abs/1806.01261, 2018.
- Neural MP: J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, Neural message passing for quantum chemistry. *ICML*, 2017.
- J. Bruna, W. Zaremba, A. Szlam, Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *ICLR*, 2014.
- T. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
- Y. Li, D. Tarlow, M. Brockschmidt, and R.S. Zemel, Gated graph sequence neural networks. *ICLR*, 2016.
- P. Barcelo, E. Kostylev, M. Monet, J. Perez, J. Reutter, and J. Silva. The logical expressiveness of graph neural networks. *ICLR*, 2020.
- Pearl, Judea. Reverend Bayes on inference engines: A distributed hierarchical approach. *AAAI*, 1982.
- Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec, Open Graph Benchmark: Datasets for Machine Learning on Graphs, *NeurIPS*, 2020.