Lecture 6: Message Passing Neural Networks

İsmail İlkan Ceylan
Fundamentals of Graph Neural Networks

- Overview, motivation, and applications of graph representation learning
  2 lectures: Scope, context, and applications of graph representation learning.

- Shallow node embedding models
  3 lectures: "Simple" node embedding models and their applications in knowledge graphs

- Fundamentals of graph neural networks
  4 lectures: Message passing neural networks, and popular graph neural network variants

- Foundations, limitations, and extensions of graph neural networks
  5 lectures: Limitations of graph neural networks in expressive power and their information bottlenecks.

- Generative graph representation learning
  3 lectures: Graph generation via variational, or autoregressive approaches.
Fundamentals of Graph Neural Networks

L1 - L2: Scope, motivation, and applications of graph representation learning

L3 - L5: Shallow embeddings: ML with undirected graphs and knowledge graphs (no features).

Beyond Shallow Embeddings

The embedding of the nodes do not share any parameters, i.e., hard to model dependencies.

Stronger encoder to capture dependencies?

Better encoder to capture structural properties?

It is hard to capture certain structural similarities, e.g., $u_1$ and $u_{10}$.

Node/graph-level features cannot be utilised effectively.

Encoder which can incorporate node features?

Hard to capture graph-level, global properties, hence worse on graph-level tasks.

Better capturing global properties?

Transductive: No embeddings for new nodes, unseen during training

Inductive models?
Message Passing Neural Networks

**Goal**: Designing neural architectures satisfying the desired desiderata!
Overview of the Lecture

The quest for a new framework
Learning to find paths in a graph is non-trivial!

From shallow to deep embeddings
Incorporating features and structure simultaneously and inductive learning.

Message passing neural networks
Message passing paradigm, a basic graph neural network, and extensions

Graph representation learning tasks
Using MPNNs for graph representation learning
Notation and Background
What Kind of Graphs?

Context: Simple, undirected, unweighted graphs \( G = (V, E, X) \) attributed with node features.

- \( V \): Set of vertices/nodes
- \( E \subseteq V \times V \): Set of edges
- \( X \in \mathbb{R}^{d \times V} \): Node feature matrix, which stores a feature vector \( x_u = X[u]^\top \) for each node \( u \).
What Kind of Graphs?

Representations: We can represent the graph in terms of its adjacency matrix and feature matrix:

- $A$ is the adjacency matrix of a graph $G = (V, E)$; $A_{[i]} \in \mathbb{R}^V$ the rows of $A$; $A_{[i,j]}$ entries of $A$.
- $X \in \mathbb{R}^{\mid V \mid \times d}$ is a feature matrix of a graph $G = (V, E)$ where $d$ is the embedding dimensionality.
- We sometimes write $G = (A, X)$ instead of $G = (V, E, X)$, or alternate between these representations.
What Kind of Graphs?

Features:

• Node features can be, e.g., domain-specific attributes, or node degrees, or simply one-hot encodings.
• We can extend the class of (attributed) graphs to include edge features $G = (V, E, X, Q)$. 
What Kind of Inputs?

$G = (V, E, X)$: Graph with node features.

$(G, u)$: Graph-node pairs, where $u \in V$.

$(G, (u, v))$: Graph-edge pair, where $u, v \in V$.

$G_\mathfrak{v}$: Class of graphs $G = (V, E, X)$.

$G_{nl}$: Class of graph-node pairs $(G, u)$.

$G_{el}$: Class of graph-edge pairs $(G, (u, v))$. 
What Kind of Functions?

\[ f : G \rightarrow B \]
\[ f : G_{nl} \rightarrow B \]
\[ f : G_{el} \rightarrow B \]

\[ f : G \rightarrow \mathbb{R} \]
\[ f : G_{nl} \rightarrow \mathbb{R} \]
\[ f : G_{el} \rightarrow \mathbb{R} \]

\[ f : G \rightarrow \mathbb{R}^d \]
\[ f : G_{nl} \rightarrow \mathbb{R}^d \]
\[ f : G_{el} \rightarrow \mathbb{R}^d \]
The Quest for a New Framework
The Quest for a New Framework
Example: Consider multi-layer perceptrons (MLPs) and embedding of a graph $G$ as:

$$f(G) = \text{MLP}(A_{[1]} \oplus \ldots \oplus A_{[|V|]}),$$

where $\oplus$ is vector concatenation of the rows $A_{[i]} \in \mathbb{R}^V$ of the adjacency matrix $A$. 
Example: Consider multi-layer perceptrons (MLPs) and embedding of a graph $G$ as:

$$f(G) = \text{MLP}(A_{[1]} \oplus \ldots \oplus A_{[|V|]}),$$

where $\oplus$ is vector concatenation of the rows $A_{[i]} \in \mathbb{R}^V$ of the adjacency matrix $A$.

Problem: This encoding depends on the ordering of nodes from the adjacency matrix!
Invariance and Equivariance

\[ G \]

\[ H \]
Invariance: A function $f : \mathbb{G} \to \mathbb{R}$ is permutation-invariant if for isomorphic graphs $G, H \in \mathbb{G}$ 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 $f : \mathbb{G}_{nl} \to \mathbb{R}^{|V|}$ is permutation-equivariant if for every permutation $\pi$ of $V$, 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.

Goal: Develop a deep learning framework which satisfies these properties, for better relational inductive bias.
Relational Inductive Bias: Pathfinding

**bAbI**: A collection of machine learning tasks.

**Background**: 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:

E s A, B n C, E w F, B w E

E s A: Encodes A is reachable from E by going south.

**Task**: Pathfinding on graphs defined over edge types.
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:

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

Inductive bias helps: higher accuracy, and less #samples.
Message Passing Neural Networks
Message passing neural networks (MPNNs) capture popular GNNs (Gilmer et al., 2017).

Idea: Iteratively update initial node features with the information received from their respective neighborhoods.

Notation: The representation of \( u \in V \) at iteration \( t \) is \( h_{u}^{(t)} \), i.e., the initial representation is \( h_{u}^{(0)} = x_{u} = X[u]^T \).
Message Passing Neural Networks

Given a graph $G = (V, E, X)$, an MPNN iteratively computes $h_u^{(t)}$ for every node $u \in V$:

\[
\begin{align*}
    h_u^{(0)} &= x_u, & \text{initialize} \\
    m_u^{(t)} &= \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} | v \in N(u)\}), & \text{aggregate} \\
    h_u^{(t)} &= \phi^{(t)}(h_u^{(t-1)}, m_u^{(t)}), & \text{update/combine}
\end{align*}
\]
Given a graph $G = (V, E, X)$, an MPNN iteratively computes $h_u^{(t)}$ for every node $u \in V$:

$$
\begin{align*}
  h_u^{(0)} &= x_u, \\
  m_u^{(t)} &= \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} \mid v \in N(u)\}), \\
  h_u^{(t)} &= \phi^{(t)}(h_u^{(t-1)}, m_u^{(t)}),
\end{align*}
$$

where $\phi^{(t)}$ and $\psi^{(t)}$ can be any differentiable function!
Given a graph $G = (V, E, X)$, an MPNN defines $\forall u \in V$ the features $h_u^{(0)} = x_u$, and iteratively updates them:

$$h_u^{(t)} = \phi^{(t)}(h_u^{(t-1)}, \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} \mid v \in N(u)\})))$$

where $\phi^{(t)}$ and $\psi^{(t)}$ are differentiable functions.
You may encounter variations, where a message computation function $\text{msg}$ is defined w.r.t the source node:

$$h_u^{(t)} = \phi^{(t)}\left(h_u^{(t-1)}, \psi^{(t)}\left(\left\{\text{msg}(h_u^{(t-1)}, h_{v}^{(t-1)}) \mid v \in N(u)\right\}\right)\right),$$

**Remark:** The function $\text{msg}$ typically depends on the neighborhood - hard to decouple $\text{msg}$ from $\psi^{(t)}$. Following a common convention, we view the message computation as part of aggregation.
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 $k = 2$. 
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 $k = 2$. 
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 $k = 2$. 
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 $k = 2$. 
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 $k = 2$. 
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 $k = 2$. 

$$A = h_A^{(0)}$$
$$B = h_B^{(0)}$$
$$C = h_C^{(0)}$$
$$D = h_D^{(0)}$$
$$E = h_E^{(0)}$$
$$F = h_F^{(0)}$$
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

\[ t = 3 \quad t = 2 \quad t = 1 \quad t = 0 \]
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.
An MPNN is **homogeneous** if $\psi$ and $\phi$ are the same across all layers, and **non-homogeneous** otherwise:

\[
\begin{align*}
    h_u^{(t)} &= \phi^{(t)} \left( h_u^{(t-1)}, \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} \mid v \in N(u)\}) \right), \\
\end{align*}
\]

If the model is homogeneous, we write $\psi$ and $\phi$, dropping the superscripts.
Node-level final representation: The final node representations are denoted as $z_{u_i} = h_{u_i}^{(k)}$.

Graph-level final representation: A final graph embedding $z_G$ for a graph $G$ through a mapping from the multiset of all the node embeddings $\{z_{u_1}, \ldots, z_{u_n}\}$ to $z_G$, known as relational pooling (Murphy et al., 2019).

Common choices are sum, or mean, which are normalized, e.g., w.r.t. number of the nodes.
Deriving a Basic Graph Neural Network Model

Model design space is very large: many possible choices for aggregate and update.

\[ h_u^{(t)} = \phi^{(t)}(h_u^{(t-1)}, \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} | v \in N(u)\})) \]

\[ = \phi^{(t)}(h_u^{(t-1)}, \sum_{v \in N(u)} h_v^{(t-1)}) \text{ aggregate: sum} \]

\[ = \sigma(W_{\text{self}}^{(t)} h_u^{(t-1)} + W_{\text{neigh}}^{(t)} \sum_{v \in N(u)} h_v^{(t-1)}) \text{ update: linear transformations with a nonlinearity at the end} \]
The Basic Graph Neural Network Model

\[ h_u^{(t)} = \sigma \left( W_{\text{self}}^{(t)} h_u^{(t-1)} + W_{\text{neigh}}^{(t)} \sum_{v \in N(u)} h_v^{(t-1)} + b^{(t)} \right) \]

- \( \psi = \text{sum aggregation} \)
- \( \sigma: \text{element-wise non-linearity} \)
- \( W_{\text{self}}^{(t)}, W_{\text{neigh}}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)} \)
- \( b^{(t)} \in \mathbb{R}^{d(t)} \)
- \( \phi = \text{update} \)
Message Passing With Self-Loops

**Message passing**: Define an aggregate function which treats the source node also as a neighbor:

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

**Self-loop**: This can be thought as (implicitly) adding self-loops to the nodes, hence the name.
Message Passing With Self-Loops

**Message passing:** Define an aggregate function which treats the source node also as a neighbor:

$$h_u^{(t)} = \psi^{(t)} \left( \left\{ \left\{ h_v^{(t-1)} \mid v \in N(u) \right\} \right\} \cup \left\{ \left\{ h_u^{(t-1)} \right\} \right\} \right)$$

**Self-loop:** This can be thought as (implicitly) adding self-loops to the nodes, hence the name.

**Basic model:** Note that this further simplifies the base model:

$$h_u^{(t)} = \sigma \left( W^{(t)} \sum_{v \in N(x)} h_v^{(t-1)} + h_u^{(t-1)} \right)$$

**Expressivity:** This limits the expressivity since the information coming from the node's neighbor's cannot be differentiated from the information from the node itself.
A Limitation of Message Passing

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

Remark: Pooling yields a graph embedding, which is global, but there is still no communication between disjoint subgraphs during message passing, so the node embeddings are “blind” to disjoint subgraphs.

Solution: Global feature computation, or global readout, on each layer of the MPNN (Battaglia et al., 2018).
The representation $h_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$h_u^{(t)} = \phi^{(t)}(h_u^{(t-1)}, \psi^{(t)}(h_u^{(t-1)}, \{h_v^{(t-1)} | v \in N(u)\}), \gamma^{(t)}(\{h_w^{(t-1)} | w \in G\})),$$

where $\gamma^{(t)}$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma^{(t)}$. 
The representation $h_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$h_u(t) = \phi(t)(h_u(t-1), \psi(t)(h_u(t-1), \{h_v(t-1) \mid v \in N(u)\}), \gamma(t)(\{h_w(t-1) \mid w \in G\})),$$

where $\gamma(t)$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma(t)$.

An instance of generalized message passing (Battaglia et al., 2018)
The representation $h_u$ for each node $u \in V$ is iteratively updated with the information received from its neighborhood as well as a global feature vector as:

$$h_u^{(t)} = \phi^{(t)}(h_u^{(t-1)}, \psi^{(t)}(h_u^{(t-1), \{h_v^{(t-1)} : v \in N(u)\}}, \{\{h_w^{(t-1)} : w \in G\})),$$

where $\gamma^{(t)}$ is a differentiable function, and all aggregate functions are typical candidates also for $\gamma^{(t)}$. 

An instance of generalized message passing (Battaglia et al., 2018).
Generalized Message Passing

Generalized message passing to account for the representations of edges, nodes, and the graph:

**Edge representation**

\[
\mathbf{h}_{(u,v)}^{(t)} = \phi_{e}^{(t)} \left( \mathbf{h}_{(u,v)}^{(t-1)}, \mathbf{h}_{u}^{(t-1)}, \mathbf{h}_{v}^{(t-1)}, \mathbf{h}_{G}^{(t-1)} \right)
\]

**Node representation**

\[
\mathbf{h}_{u}^{(t)} = \phi_{n}^{(t)} \left( \mathbf{h}_{u}^{(t-1)}, \psi^{(t)} \left( \mathbf{h}_{u}^{(t-1)}, \{ \mathbf{h}_{v}^{(t-1)} \mid v \in N(u) \} \right), \mathbf{h}_{G}^{(t-1)} \right)
\]

**Graph representation**

\[
\mathbf{h}_{G}^{(t)} = \phi_{G}^{(t)} \left( \mathbf{h}_{G}^{(t-1)}, \{ \{ \mathbf{h}_{u}^{(t)} \mid u \in V_{G} \} \}, \{ \{ \mathbf{h}_{(u,v)}^{(t)} \mid (u, v) \in E \} \} \right)
\]

Each update happens in the given equation order!
Graph Representation Learning
Tasks
The learned embeddings can be used for many graph machine learning task, e.g., graph/node classification, graph/node regression, graph/node clustering, depending how they are learned.
Node-Level Tasks

Node classification:

• Given a graph $G$, 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 $v \in V \setminus V_{tr}$.

Example:

• 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).

• Predict the category (or, categories) of the remaining papers.
Node-Level Tasks

**Training**: Based on a loss, e.g., negative log-likelihood:

\[ \mathcal{L} = \sum_{u \in V_{tr}} - \log(\text{softmax}(z_u, y_u)), \]

where \( y_u \) is a one-hot vector indicating the class \( y_u \) of the training node \( u \in V_{tr} \).

\( \text{softmax}(z_u, y_u) \): the probability that \( u \) belongs to the class \( y_u \).

**Node regression**, or **clustering** (i.e., community detection) are other node-level tasks.
Two regimes: Loss function uses only nodes from the training set $V_{tr}$, but test nodes $V_{test} = V \setminus V_{tr}$ are still observed during training.

- **Transductive**: All nodes are observed during training.
- **Inductive**: Not all nodes are observed during training. For some test nodes, neither the nodes nor their edges are known.

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

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

**Remark**: Standard semi-supervised setting also requires i.i.d. assumption, which does not hold for node classification.

**Inductive node classification** can be seen as supervised learning.
Edge-Level Tasks

Example (Link Prediction):

- OGBL-DDI (Hu et al., 2020) is a drug-drug interaction network: Node's represent drugs and edges interactions between drugs.
- Predict drug-drug interactions: 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.
- Evaluation is similar to standard KG evaluation, but splits are fixed for OGBL-WIKIKG2.
Graph-Level Tasks

Graph classification:

• Given a set of graphs $\mathbb{G} = \{G_1, \ldots, G_n\}$, where a subset of the graphs $\{(G_i, y_{G_i}) | G_i \in \mathbb{G}_{tr} \subset \mathbb{G}\}$ are labeled with a class.

• Predict the labels of the remaining (test) graphs.

Example (Graph classification): IMDB (Morris et al., 2020) consist of 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.
**Graph-Level Tasks**

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

$$
\mathcal{L} = \sum_{G \in \mathcal{G}_{tr}} - \log(\text{softmax}(z_G, y_G))
$$

Graph classification is a supervised learning task: 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, e.g., to predict a molecule’s toxicity.

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., $MLP(z_G)$, for each graph $G$.

**Example (Graph generation):** Generate novel molecules which could be candidates for novel drugs!
Homophily and Heterophily
Homophily: Describes a strong positive correlation between nodes and their neighbors within a graph, i.e., a node is highly likely to share features and attributes with its neighbors in the graph.

Example: Citation networks, where connected papers tend to tackle similar research areas.

Heterophily: Describes negative correlations between nodes and their neighbors, i.e., a node tends to have contrasting features relative to its neighbors.

Example: Protein graphs, as the proteins interacting with each other may differ from a composition perspective.
Homophily and Heterophily

Data-driven inductive bias: Unlike permutation-invariance, the bias does not rely on structural properties of graphs, but on the application domain and the specific input instances.

Practical: These biases are prominent in real-world applications, and are commonly exploited.

Local: MPNNs employ local operations and neighbor aggregation. Easy to capture correlations by simply adjusting combination and aggregation weights.
Summary and Outlook

Overview, motivation, and applications of graph representation learning

2 lectures: Scope, context, and applications of graph representation learning.

Shallow node embedding models

3 lectures: “Simple” node embedding models and their applications in knowledge graphs.

Fundamentals of graph neural networks

4 lectures: Message passing neural networks, and popular graph neural network variants.

Foundations, limitations, and extensions of graph neural networks

4 lectures: Limitations of graph neural networks in expressive power and their information bottlenecks.

Generative graph representation learning

4 lectures: Graph generation via variational, or autoregressive approaches.
Overview, motivation, and applications of graph representation learning

2 lectures: Scope, context, and applications of graph representation learning.

Shallow node embedding models

3 lectures: "Simple" node embedding models and their applications in knowledge graphs

Fundamentals of graph neural networks

4 lectures: Message passing neural networks, and popular graph neural network variants

Foundations, limitations, and extensions of graph neural networks

4 lectures: Limitations of graph neural networks in expressive power and their information bottlenecks.

Generative graph representation learning

4 lectures: Graph generation via variational, or autoregressive approaches.

Summary and Outlook

Lecture 6: Message Passing Neural Networks

- Shallow vs deep embeddings
- Relational inductive bias
- MPNNs as a general framework and variations of the message passing
- MPNNs for graph representation learning tasks
- Homophily and heterophily
References


