Lecture 3: Graph Neural Networks

Relational Learning

İsmail İlkan Ceylan

From Shallow to Deep Node Embeddings

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

Lecture 3 - 8 (Graph neural networks): We will study graph neural networks, which are neural networks that learn representations of nodes that depend on the structure of the graph.

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

Lecture 3 - 8 (Graph neural networks): We will study graph neural networks, which are neural networks that learn representations of nodes that depend on the structure of the graph.

Recall key differences between shallow and deep embedding models:

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

Lecture 3 - 8 (Graph neural networks): We will study graph neural networks, which are neural networks that learn representations of nodes that depend on the structure of the graph.

Recall key differences between shallow and deep embedding models:

- Shallow embedding models are transductive in that they do not apply to novel entities unless we train again on these entities. They are also inherently limited to single-graph tasks.

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

Lecture 3 - 8 (Graph neural networks): We will study graph neural networks, which are neural networks that learn representations of nodes that depend on the structure of the graph.

Recall key differences between shallow and deep embedding models:

- Shallow embedding models are transductive in that they do not apply to novel entities unless we train again on these entities. They are also inherently limited to single-graph tasks.
- We are interested in more elaborate embedding models which can use any feature information available, and can also generalise to novel data points, i.e., inductive models.

From Shallow to Deep Node Embeddings

Lecture 1 - 2 (Knowledge graph embeddings): We discussed shallow node embedding models in the context of knowledge graphs, where the idea is to learn low-dimensional embeddings of the nodes in a graph.

Lecture 3 - 8 (Graph neural networks): We will study graph neural networks, which are neural networks that learn representations of nodes that depend on the structure of the graph.

Recall key differences between shallow and deep embedding models:

- Shallow embedding models are transductive in that they do not apply to novel entities unless we train again on these entities. They are also inherently limited to single-graph tasks.
- We are interested in more elaborate embedding models which can use any feature information available, and can also generalise to novel data points, i.e., inductive models.

The remaining lectures on relational learning focus on graph neural networks: We will cover the basics and widely employed models, while also focusing on some limitations and extensions proposed in the literature.

Overview of the Lecture

Overview of the Lecture

- Motivation and relational inductive bias

Overview of the Lecture

- Motivation and relational inductive bias
- Message passing neural networks

Overview of the Lecture

- Motivation and relational inductive bias
- Message passing neural networks
- Graph representation learning tasks

Overview of the Lecture

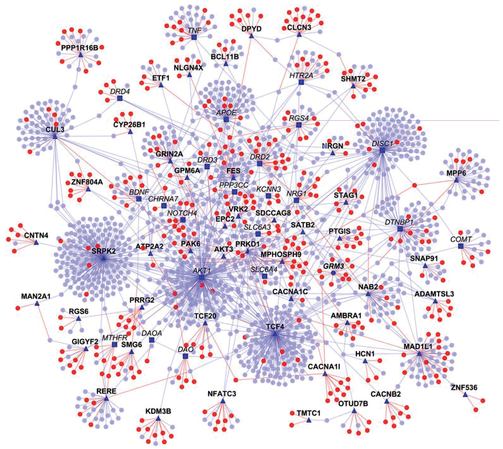
- Motivation and relational inductive bias
- Message passing neural networks
- Graph representation learning tasks
- Perspectives for graph neural networks

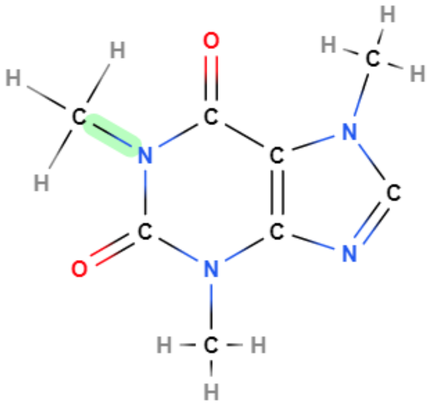
Overview of the Lecture

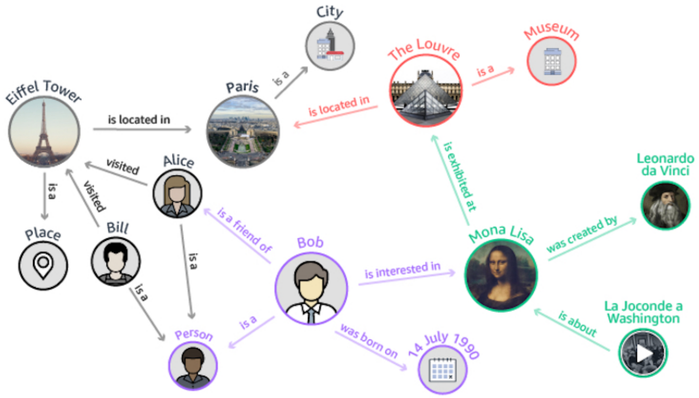
- Motivation and relational inductive bias
- Message passing neural networks
- Graph representation learning tasks
- Perspectives for graph neural networks
- Summary

Motivation and Relational Inductive Bias

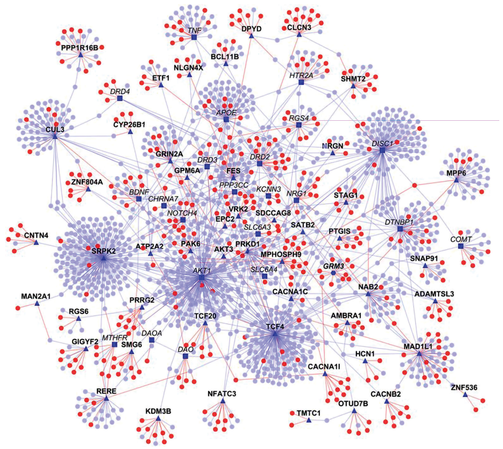
Beyond Euclidian Spaces

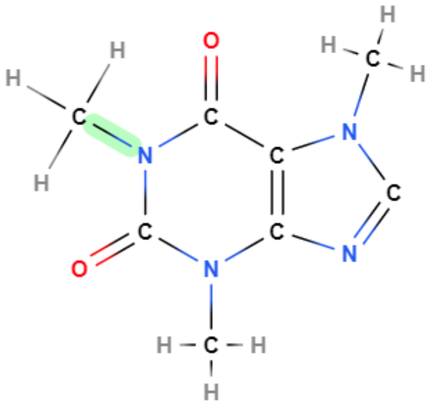


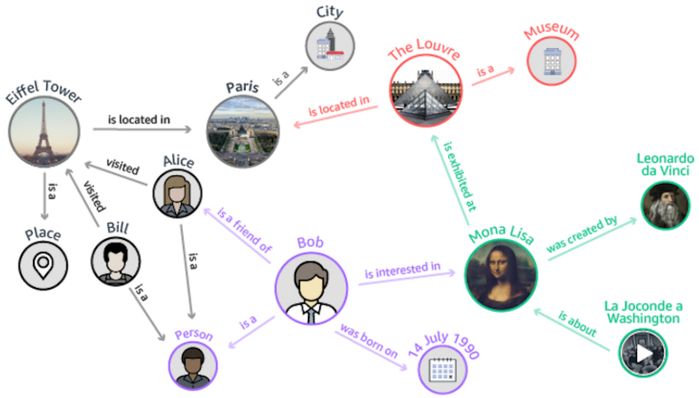




Beyond Euclidian Spaces







Geometric deep learning is an umbrella term for deep learning over non-Euclidian spaces (Bronstein et al.), primarily graphs and manifolds. The focus of this course is on graph representation learning.

What Kind of Graphs?

What Kind of Graphs?

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

What Kind of Graphs?

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

We focus on undirected, unweighted, and unlabelled (single-relation) graphs, and assume deterministic node features, unless explicitly stated otherwise. We sometimes speak about different types of graphs, but this shall become clear from the context.

What Kind of Graphs?

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

We focus on undirected, unweighted, and unlabelled (single-relation) graphs, and assume deterministic node features, unless explicitly stated otherwise. We sometimes speak about different types of graphs, but this shall become clear from the context.

Remark: This choice is partly for brevity, as most of the concepts can be extended to other types of graphs. Not all extensions are straight-forward, though, e.g., extensions to more general graphs such as hyper-graphs are non-trivial and do not presently admit any efficient models.

What Kind of Graphs?

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

We focus on undirected, unweighted, and unlabelled (single-relation) graphs, and assume deterministic node features, unless explicitly stated otherwise. We sometimes speak about different types of graphs, but this shall become clear from the context.

Remark: This choice is partly for brevity, as most of the concepts can be extended to other types of graphs. Not all extensions are straight-forward, though, e.g., extensions to more general graphs such as hyper-graphs are non-trivial and do not presently admit any efficient models.

Notation: We consider graphs of the form $G = (V, E)$ 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.

What Kind of Graphs?

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

We focus on undirected, unweighted, and unlabelled (single-relation) graphs, and assume deterministic node features, unless explicitly stated otherwise. We sometimes speak about different types of graphs, but this shall become clear from the context.

Remark: This choice is partly for brevity, as most of the concepts can be extended to other types of graphs. Not all extensions are straight-forward, though, e.g., extensions to more general graphs such as hyper-graphs are non-trivial and do not presently admit any efficient models.

Notation: We consider graphs of the form $G = (V, E)$ 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.

Intuitively, 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.

What Kind of Graphs?

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

We focus on undirected, unweighted, and unlabelled (single-relation) graphs, and assume deterministic node features, unless explicitly stated otherwise. We sometimes speak about different types of graphs, but this shall become clear from the context.

Remark: This choice is partly for brevity, as most of the concepts can be extended to other types of graphs. Not all extensions are straight-forward, though, e.g., extensions to more general graphs such as hyper-graphs are non-trivial and do not presently admit any efficient models.

Notation: We consider graphs of the form $G = (V, E)$ 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.

Intuitively, 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.

Finally, we write $\mathbf{A}^G$ to denote the adjacency matrix of a graph $G = (V, E)$, and $\mathbf{A}^G_{[i]} \in \mathbb{R}^{V_G}$ to denote the rows of the adjacency matrix.

Learning over Graphs

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

These assumptions are helpful since

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

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

These assumptions are helpful since

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

These assumptions are unrealistic in the context of graphs.

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

These assumptions are helpful since

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

These assumptions are unrealistic in the context of graphs.

Suppose we are given a single graph, and we want to classify the nodes in the given graph with respect to a certain property (i.e., node classification).

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

These assumptions are helpful since

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

These assumptions are unrealistic in the context of graphs.

Suppose we are given a single graph, and we want to classify the nodes in the given graph with respect to a certain property (i.e., node classification).

Such properties depend on the other nodes through edges, e.g., imagine functions that rely on node statistics (e.g., #neighbours), or the overall graph structure (e.g., is the node in a cycle).

Learning over Graphs

Machine learning algorithms make many assumptions about the data, and a common assumption is to assume that the data points are independent and identically distributed (i.i.d.).

These assumptions are helpful since

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

These assumptions are unrealistic in the context of graphs.

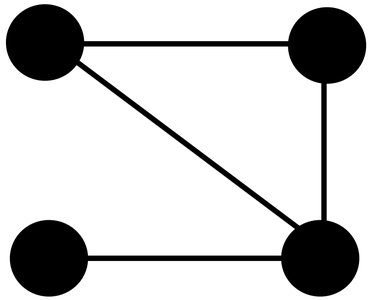
Suppose we are given a single graph, and we want to classify the nodes in the given graph with respect to a certain property (i.e., node classification).

Such properties depend on the other nodes through edges, e.g., imagine functions that rely on node statistics (e.g., #neighbours), or the overall graph structure (e.g., is the node in a cycle).

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

Learning over Graphs

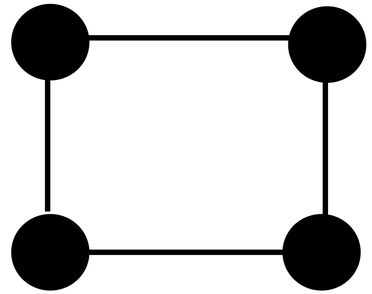
Learning over Graphs



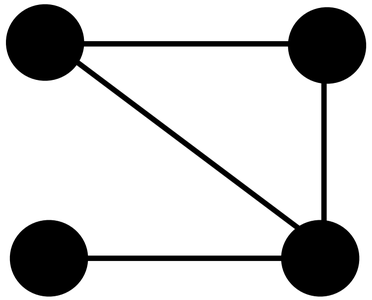
Node degrees?

Contains an odd-length cycle?

Minimum vertex cover size 1, 2?



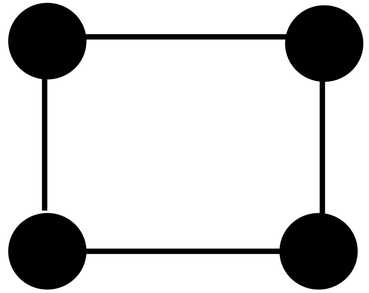
Learning over Graphs



Node degrees?

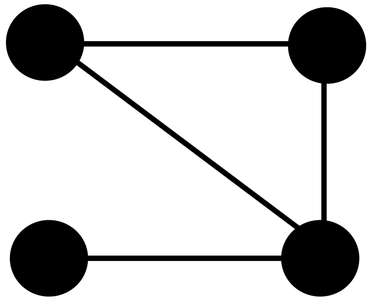
Contains an odd-length cycle?

Minimum vertex cover size 1, 2?



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

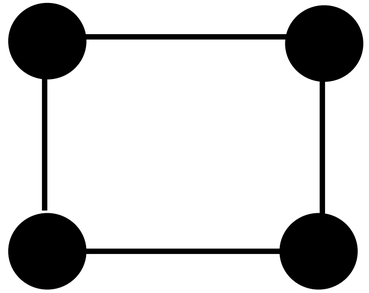
Learning over Graphs



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. This information needs to be taken into account adequately.

Intuitively, this could be achieved by defining similarity measures, for nodes/graphs, which can then be used for the optimisation task.

Learning over Graphs

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

- Extracting node, or graph-level statistics or features (indicating, e.g., node/graph similarity), and

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

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

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

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

Most popular graph similarity functions are studied under the name of graph kernel methods; see, e.g., (Kriege et al., 2020) for a recent survey.

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

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

Most popular graph similarity functions are studied under the name of graph kernel methods; see, e.g., (Kriege et al., 2020) for a recent survey.

Modern deep learning approaches do not explicitly extract such statistics, but there are nevertheless strong connections between modern graph representation learning and some well-known graph kernel methods, such as Weisfeiler Lehman kernel (Weisfeiler and Leman, 1968)!

Learning over Graphs

Traditional approaches to machine learning over graph-structured data are based on:

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

Most popular graph similarity functions are studied under the name of graph kernel methods; see, e.g., (Kriege et al., 2020) for a recent survey.

Modern deep learning approaches do not explicitly extract such statistics, but there are nevertheless strong connections between modern graph representation learning and some well-known graph kernel methods, such as Weisfeiler Lehman kernel (Weisfeiler and Leman, 1968)!

We will discuss the Weisfeiler Lehman algorithm, and its connection to popular graph neural network models, in detail later in the course.

The Quest for a New Framework

The Quest for a New Framework

Earlier deep learning models do not capture graph-structured domains adequately.

Example: Let us consider multi-layer perceptrons. We can define an embedding of a graph G as a multi-layer perceptron:

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

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

The Quest for a New Framework

Earlier deep learning models do not capture graph-structured domains adequately.

Example: Let us consider multi-layer perceptrons. We can define an embedding of a graph G as a multi-layer perceptron:

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

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 representation depends on the ordering of nodes that we used in the adjacency matrix! Learned representation depends on an ordering that is arbitrary!

The Quest for a New Framework

Earlier deep learning models do not capture graph-structured domains adequately.

Example: Let us consider multi-layer perceptrons. We can define an embedding of a graph G as a multi-layer perceptron:

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

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 representation depends on the ordering of nodes that we used in the adjacency matrix! Learned representation depends on an ordering that is arbitrary!

Other models: Convolutional neural networks are well-defined over grids, but not on graphs. Long short-term memory networks process sequential (including, e.g., tree-shaped) data, but not graphs: In fact, the above-mentioned problem persists, as by encoding graphs as simple sequences, we lose valuable information.

The Quest for a New Framework

Earlier deep learning models do not capture graph-structured domains adequately.

Example: Let us consider multi-layer perceptrons. We can define an embedding of a graph G as a multi-layer perceptron:

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

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 representation depends on the ordering of nodes that we used in the adjacency matrix! Learned representation depends on an ordering that is arbitrary!

Other models: Convolutional neural networks are well-defined over grids, but not on graphs. Long short-term memory networks process sequential (including, e.g., tree-shaped) data, but not graphs: In fact, the above-mentioned problem persists, as by encoding graphs as simple sequences, we lose valuable information.

We need a new kind of deep learning framework!

Invariance and Equivariance

Invariance and Equivariance

Given a set of graphs $\mathscr{G}$, we consider functions of the form $f : \mathscr{G} \to \mathbb{R}^{V_G}$.

Invariance and Equivariance

Given a set of graphs $\mathcal{G}$, we consider functions of the form $f : \mathcal{G} \to \mathbb{R}^{V_G}$.

Invariance: We say that a function f is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$.

Invariance and Equivariance

Given a set of graphs $\mathcal{G}$, we consider functions of the form $f : \mathcal{G} \to \mathbb{R}^{V_G}$.

Invariance: We say that a function f is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$.

Equivariance: We say that a function f is permutation-equivariant if it has the property that for every permutation π of V_G, it holds that $f(G^\pi) = f(G)^\pi$.

Invariance and Equivariance

Given a set of graphs $\mathcal{G}$, we consider functions of the form $f : \mathcal{G} \to \mathbb{R}^{V_G}$.

Invariance: We say that a function f is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$.

Equivariance: We say that a function f is permutation-equivariant if it has the property that for every permutation π of V_G, it holds that $f(G^\pi) = f(G)^\pi$.

Intuitively, permutation-invariance implies that the function does not depend on the ordering of the nodes in the graph, and permutation-equivariance implies the output of f is permuted in a consistent way when we permute the nodes in the graph.

Invariance and Equivariance

Given a set of graphs $\mathcal{G}$, we consider functions of the form $f : \mathcal{G} \to \mathbb{R}^{V_G}$.

Invariance: We say that a function f is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$.

Equivariance: We say that a function f is permutation-equivariant if it has the property that for every permutation π of V_G, it holds that $f(G^\pi) = f(G)^\pi$.

Intuitively, permutation-invariance implies that the function does not depend on the ordering of the nodes in the graph, and permutation-equivariance implies the output of f is permuted in a consistent way when we permute the nodes in the graph.

We can also consider functions of different forms, e.g., $f : \mathcal{G} \to \mathbb{R}$. Note that permutation-invariance is still well-defined over these functions, but we cannot speak of permutation-equivariance here.

Invariance and Equivariance

Given a set of graphs $\mathcal{G}$, we consider functions of the form $f : \mathcal{G} \to \mathbb{R}^{V_G}$.

Invariance: We say that a function f is permutation-invariant if for isomorphic graphs $G, H \in \mathcal{G}$ it holds that $f(G) = f(H)$.

Equivariance: We say that a function f is permutation-equivariant if it has the property that for every permutation π of V_G, it holds that $f(G^\pi) = f(G)^\pi$.

Intuitively, permutation-invariance implies that the function does not depend on the ordering of the nodes in the graph, and permutation-equivariance implies the output of f is permuted in a consistent way when we permute the nodes in the graph.

We can also consider functions of different forms, e.g., $f : \mathcal{G} \to \mathbb{R}$. Note that permutation-invariance is still well-defined over these functions, but we cannot speak of permutation-equivariance here.

Argument: These properties are important for graph representation learning tasks, as they provide a strong relational inductive bias! The goal is to develop a deep learning framework enhanced with these properties.

Invariance and Equivariance: Critical Perspective

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Discussion: Let us loosely denote by M a model that does not have these properties and by M^+ a model that has these properties. We aim to learn a rather simple permutation-invariant or equivariant function f:

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Discussion: Let us loosely denote by M a model that does not have these properties and by M^+ a model that has these properties. We aim to learn a rather simple permutation-invariant or equivariant function f:

- Theoretically, it possible to learn f using M, but it is non-trivial to ensure e.g., invariance to orderings, or even approximate this well in practice.

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Discussion: Let us loosely denote by M a model that does not have these properties and by M^+ a model that has these properties. We aim to learn a rather simple permutation-invariant or equivariant function f:

- Theoretically, it possible to learn f using M, but it is non-trivial to ensure e.g., invariance to orderings, or even approximate this well in practice.
- Learning a function f will likely require longer training time for model M as compared to model M^+, as M needs to learn "more".

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Discussion: Let us loosely denote by M a model that does not have these properties and by M^+ a model that has these properties. We aim to learn a rather simple permutation-invariant or equivariant function f:

- Theoretically, it possible to learn f using M, but it is non-trivial to ensure e.g., invariance to orderings, or even approximate this well in practice.
- Learning a function f will likely require longer training time for model M as compared to model M^+, as M needs to learn "more".
- Learning a function f will likely require more training data for model M as compared to model M^+, as, e.g., M likely needs more examples (of orderings) so as to learn invariance to them.

Invariance and Equivariance: Critical Perspective

Question: Wouldn't it be possible to learn properties such as invariance and equivariance from data during training? Why do we need a framework that ensures such properties?

Discussion: Let us loosely denote by M a model that does not have these properties and by M^+ a model that has these properties. We aim to learn a rather simple permutation-invariant or equivariant function f:

- Theoretically, it possible to learn f using M, but it is non-trivial to ensure e.g., invariance to orderings, or even approximate this well in practice.
- Learning a function f will likely require longer training time for model M as compared to model M^+, as M needs to learn "more".
- Learning a function f will likely require more training data for model M as compared to model M^+, as, e.g., M likely needs more examples (of orderings) so as to learn invariance to them.
- Having the right inductive bias for a domain is very important — This has been observed on other deep learning models, and motivated new architectures (e.g., the use of convolutions which are translation-invariant).

Relational Inductive Bias: Pathfinding

Relational Inductive Bias: Pathfinding

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

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. In this reformulation, we are given a set of connections:

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

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. In this reformulation, we are given a set of connections:

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

where, for instance, "E s A" denotes A is reachable from E by going south. The task is to find a path between, e.g., B and A, such as w,s.

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. In this reformulation, we are given a set of connections:

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

where, for instance, "E s A" denotes A is reachable from E by going south. The task is to find a path between, e.g., B and A, such as w,s.

This is a rather simple pathfinding problem on graphs defined over edge types s, n, e, w. Can we learn to predict such paths?

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. In this reformulation, we are given a set of connections:

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

where, for instance, "E s A" denotes A is reachable from E by going south. The task is to find a path between, e.g., B and A, such as w,s.

This is a rather simple pathfinding problem on graphs defined over edge types s, n, e, w. Can we learn to predict such paths?

Note, for instance, that one would expect the answers to be the same for isomorphic graph instances: permutation-invariance.

Relational Inductive Bias: Pathfinding

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?

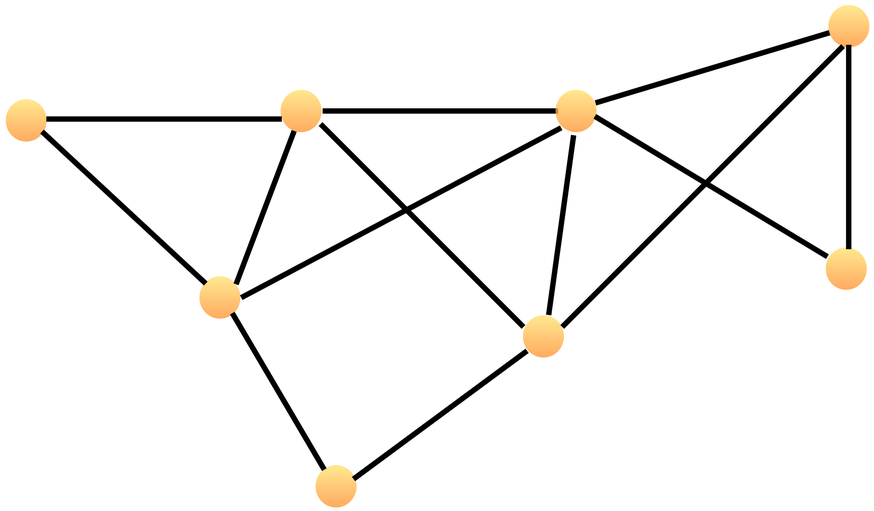
Results: (Li et al., 2016) reports the empirical results relative to LSTMs and gated graph sequence neural networks (GGSNNs), i.e., a graph neural network model proposed in the same paper:

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

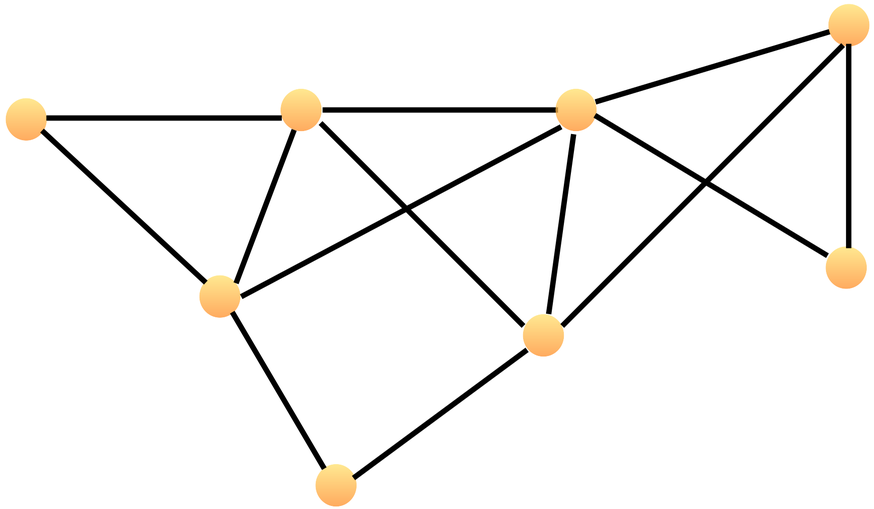
The relational inductive bias helps a lot, both in terms of accuracy, and in the #samples needed Similar phenomenons have been observed in other papers and on other tasks, and we will discuss some of these in more detail later in the course.

Message Passing Neural Networks

Message Passing Neural Networks

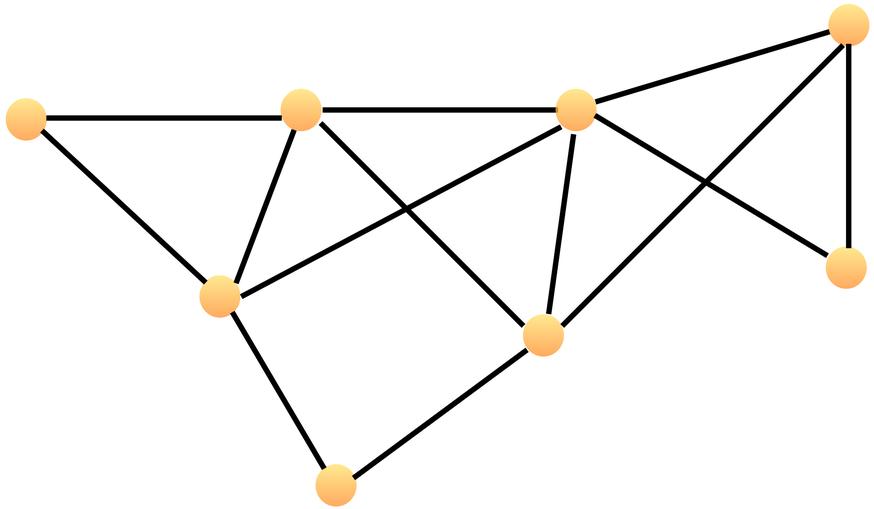


Message Passing Neural Networks



We consider input graphs $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, where the features will be used to initialise the representations of the respective nodes.

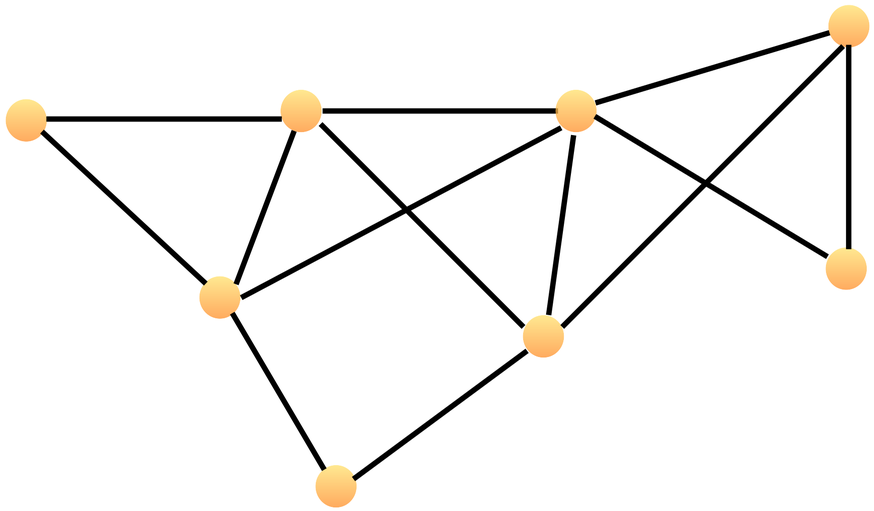
Message Passing Neural Networks



We consider input graphs $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, where the features will be used to initialise the representations of the respective nodes.

These representations are updated with the information received from their respective neighbourhoods (i.e., message passing), and this process continues for k iterations, yielding final node representations.

Message Passing Neural Networks



We consider input graphs $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, where the features will be used to initialise the representations of the respective nodes.

These representations are updated with the information received from their respective neighbourhoods (i.e., message passing), and this process continues for k iterations, yielding final node representations.

(Gilmer et al., 2017) defined a single common framework, called Message Passing Neural Networks (MPNNs), which captures a popular family of graph neural network models proposed in the literature.

Message Passing Neural Networks

Message Passing Neural Networks

Initialisation: A message passing neural network defines, for every node $u \in V$, an initial vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

Message Passing Neural Networks

Initialisation: A message passing neural network defines, for every node $u \in V$, an initial vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

Message passing: Let us denote the representation for each node $u \in V$ at iteration t by $\mathbf{h}_u^{(t)}$. The representation $\mathbf{h}_u$ for each node $u \in V$ is then iteratively updated with the information received from its neighbourhood as:

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

where $aggregate^{(t)}$ and $combine^{(t)}$ are arbitrary differentiable functions (i.e., neural networks), and $aggregate^{(t)}$ is a permutation-invariant function by construction, since its input is a set. Typical aggregate functions are mean, sum, or max.

Message Passing Neural Networks

Initialisation: A message passing neural network defines, for every node $u \in V$, an initial vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

Message passing: Let us denote the representation for each node $u \in V$ at iteration t by $\mathbf{h}_u^{(t)}$. The representation $\mathbf{h}_u$ for each node $u \in V$ is then iteratively updated with the information received from its neighbourhood as:

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

where $aggregate^{(t)}$ and $combine^{(t)}$ are arbitrary differentiable functions (i.e., neural networks), and $aggregate^{(t)}$ is a permutation-invariant function by construction, since its input is a set. Typical aggregate functions are mean, sum, or max.

Observe that we allow different functions at different iterations: if $aggregate^{(t)}$ and $combine^{(t)}$ are the same across all iterations, it is sometimes called a homogeneous MPNN.

Message Passing Neural Networks

Initialisation: A message passing neural network defines, for every node $u \in V$, an initial vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$.

Message passing: Let us denote the representation for each node $u \in V$ at iteration t by $\mathbf{h}_u^{(t)}$. The representation $\mathbf{h}_u$ for each node $u \in V$ is then iteratively updated with the information received from its neighbourhood as:

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

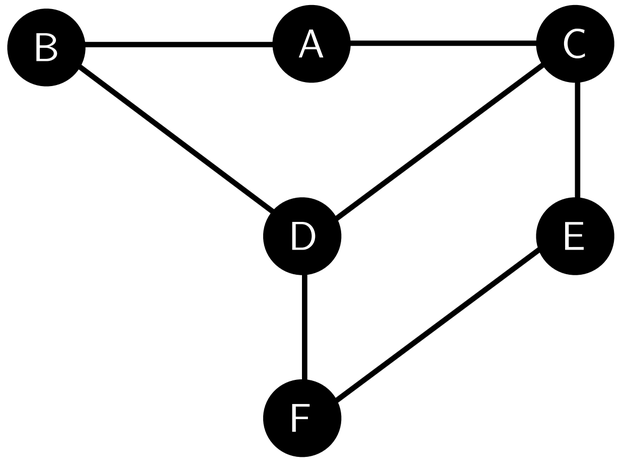
where $aggregate^{(t)}$ and $combine^{(t)}$ are arbitrary differentiable functions (i.e., neural networks), and $aggregate^{(t)}$ is a permutation-invariant function by construction, since its input is a set. Typical aggregate functions are mean, sum, or max.

Observe that we allow different functions at different iterations: if $aggregate^{(t)}$ and $combine^{(t)}$ are the same across all iterations, it is sometimes called a homogeneous MPNN.

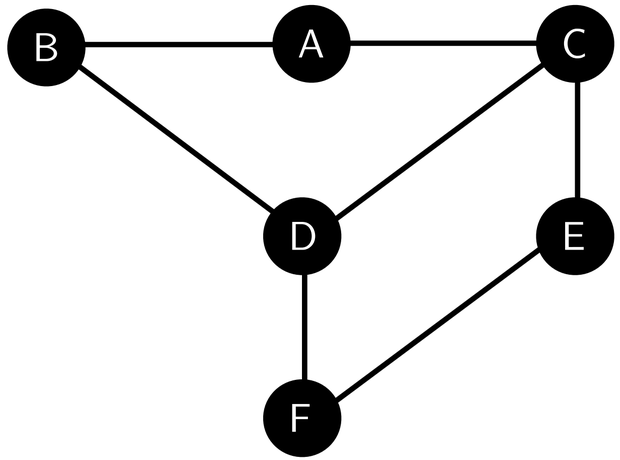
Final representation: Upon termination, the final node representations will be denoted as $\mathbf{z}_u = \mathbf{h}_u^{(k)}$.

Message Passing Neural Networks

Message Passing Neural Networks

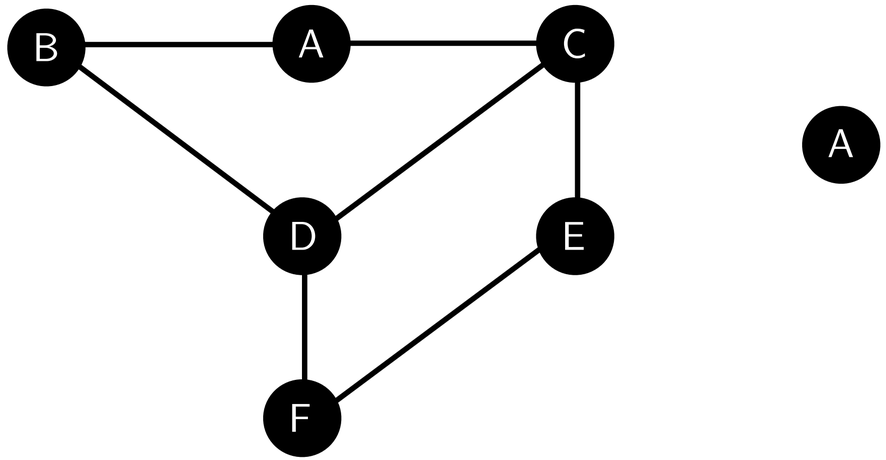


Message Passing Neural Networks



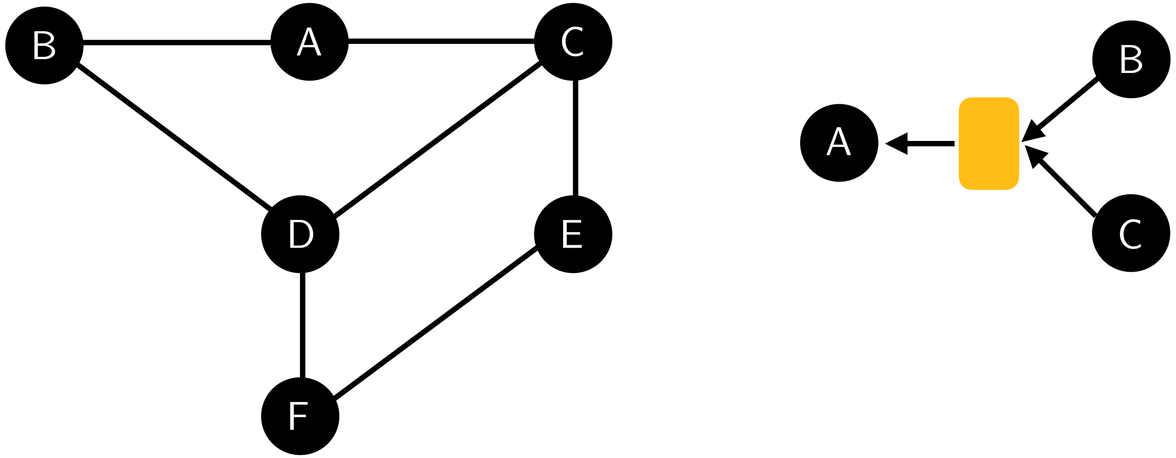
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

Message Passing Neural Networks



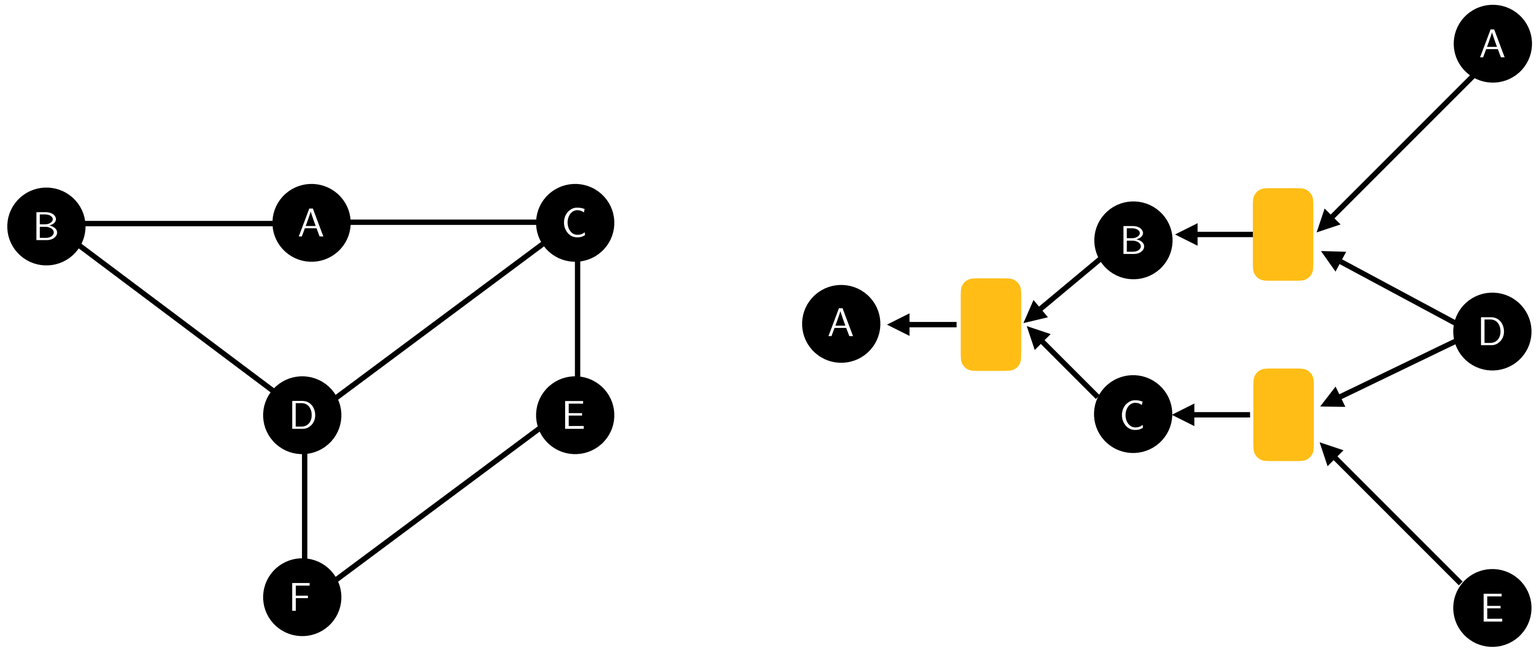
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

Message Passing Neural Networks



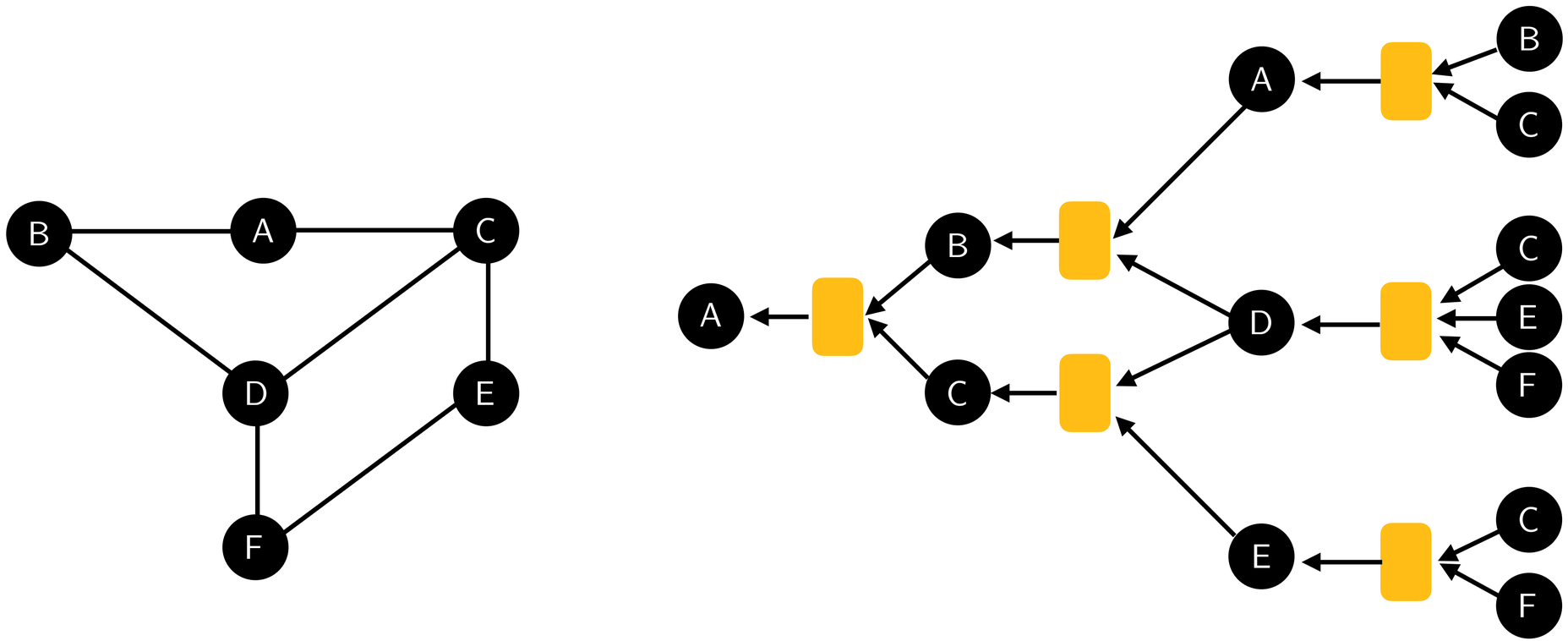
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

Message Passing Neural Networks



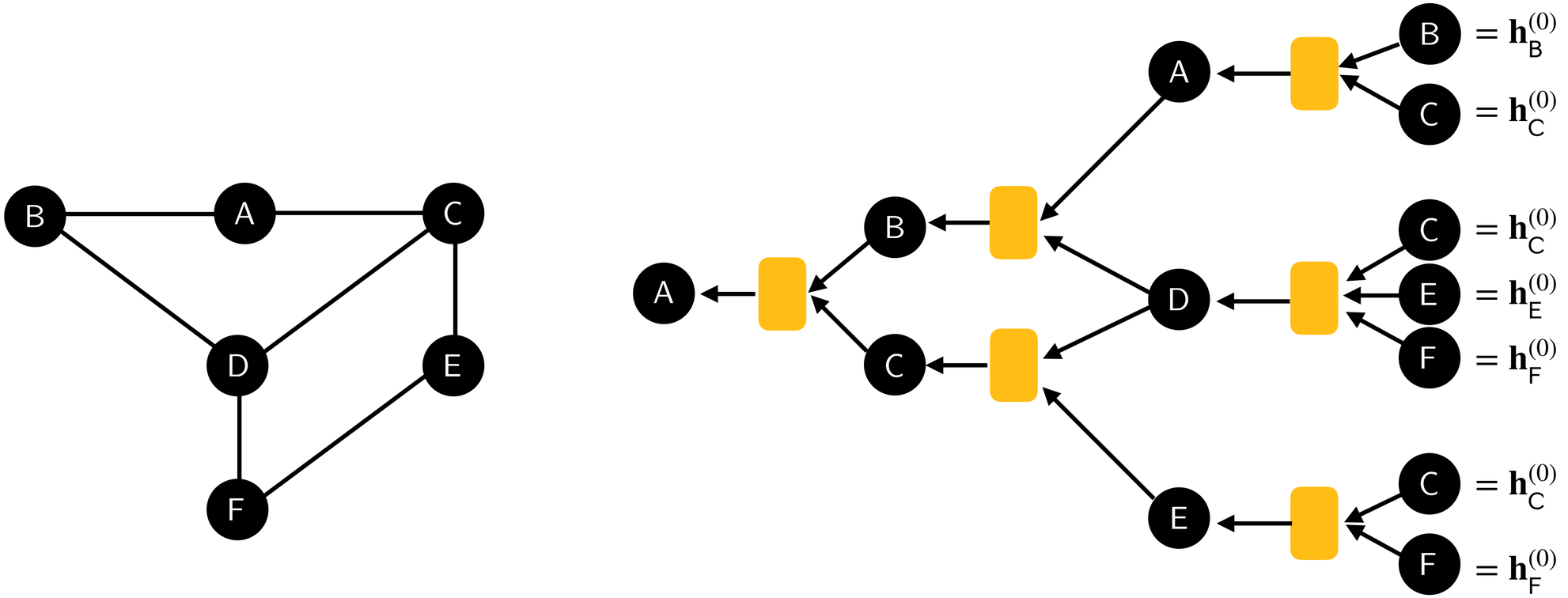
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

Message Passing Neural Networks



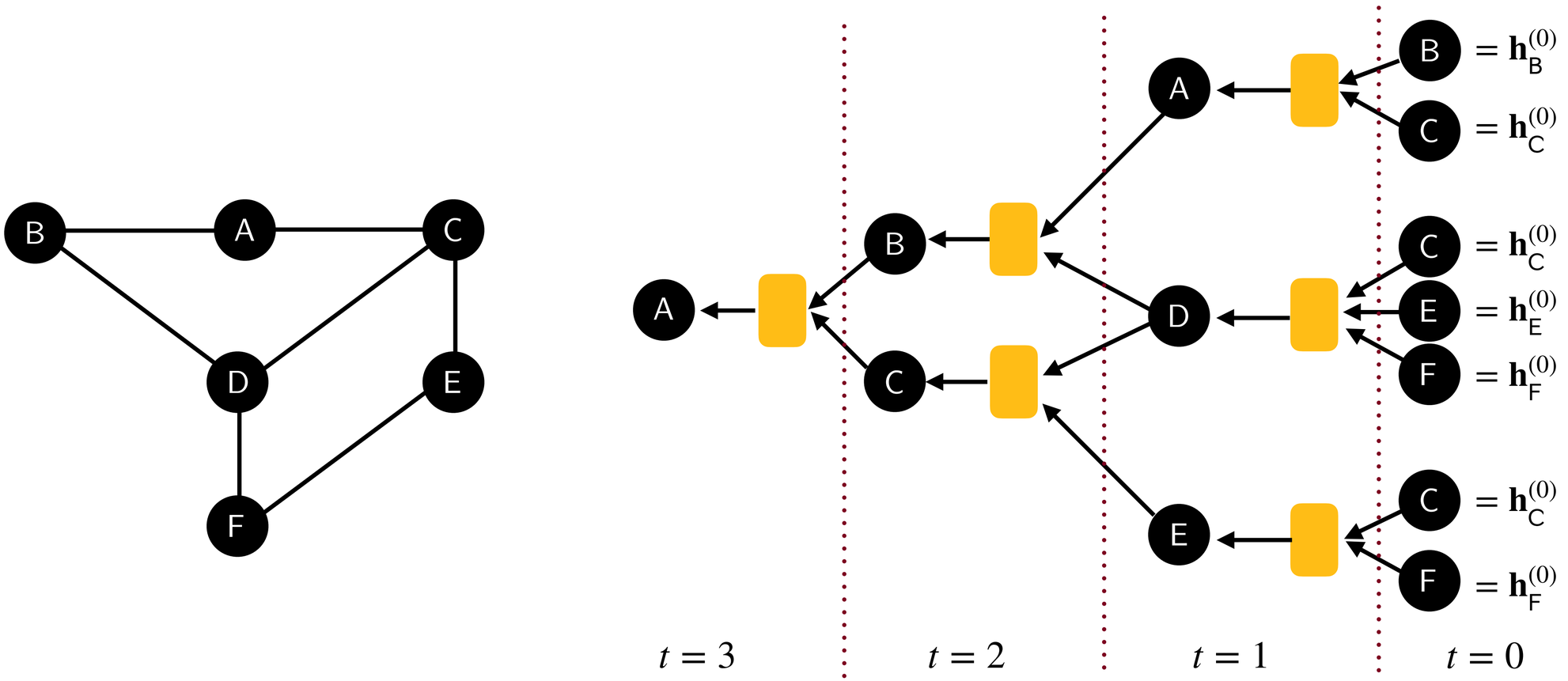
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

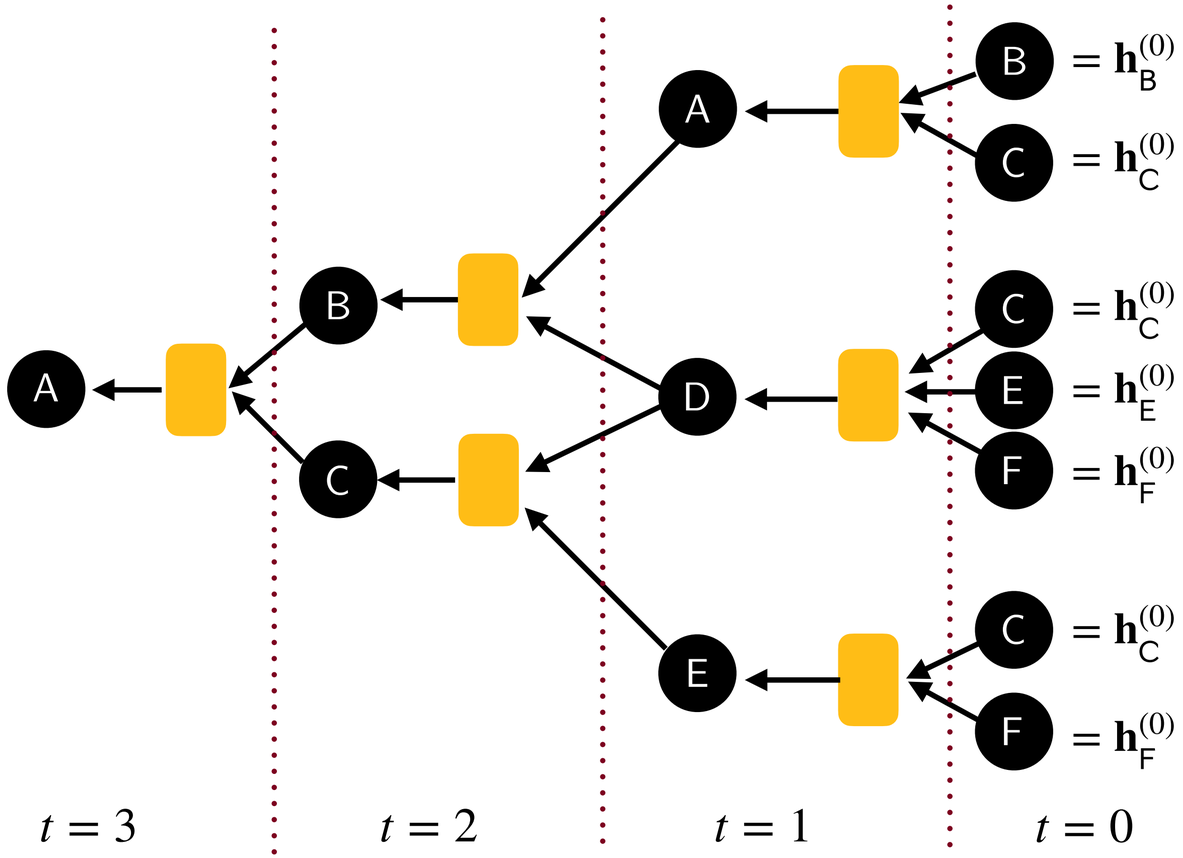
Message Passing Neural Networks



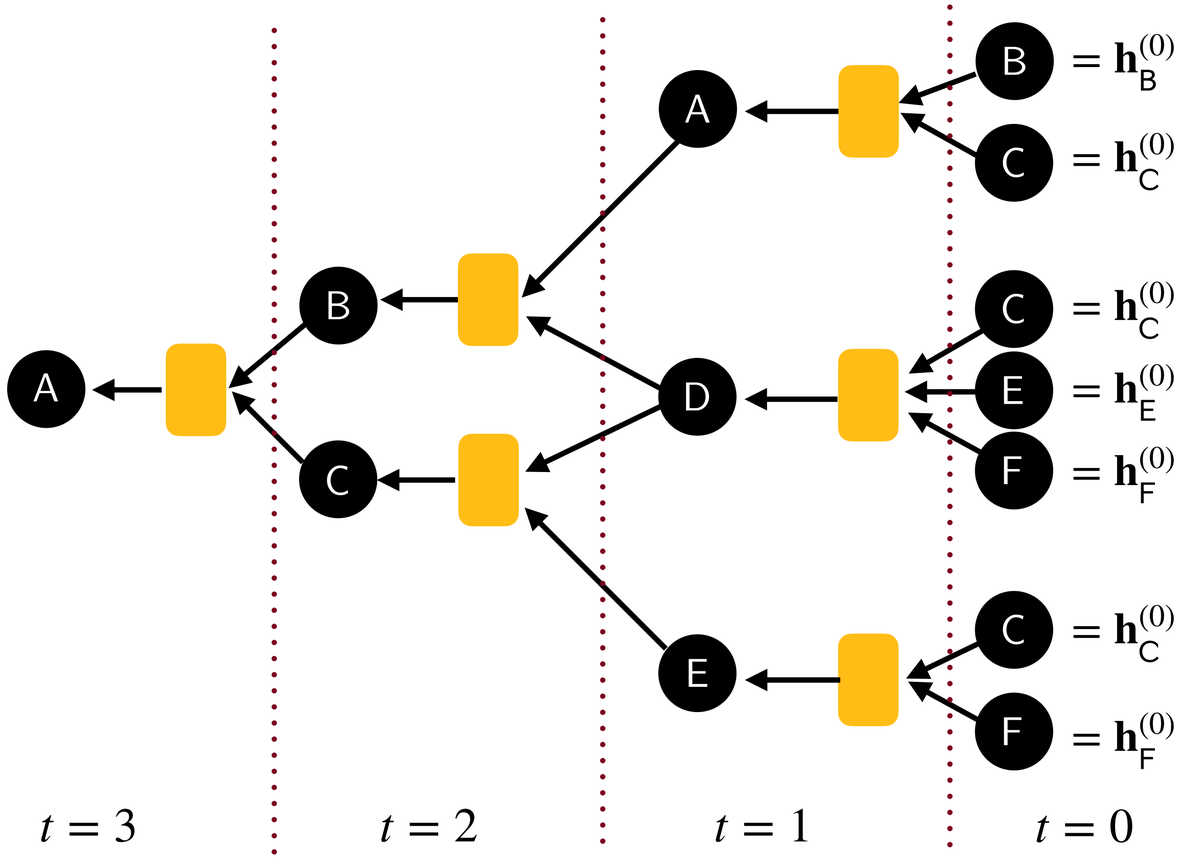
A graph (left) and an illustration of message passing on this graph with respect to the target node A for 3 iterations (right). Directed arrows depict the messages, and yellow boxes denote aggregation. At least $k = 3$ iterations are needed to get information from all nodes, i.e., F will not pass any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

Message Passing Neural Networks

Message Passing Neural Networks



Message Passing Neural Networks



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

The Basic Graph Neural Network Model

The Basic Graph Neural Network Model

We describe a simple, concrete base model. The idea is again to initialise every node $u \in V$ to a vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Then, the representation updates are given as:

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

where $\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)}$ are trainable parameter matrices and σ is an element-wise non-linear function (e.g., ReLU), and $\mathbf{b}^{(\mathbf{t})} \in \mathbb{R}^{d(t)}$ is a bias term (which we will omit in the sequel).

The Basic Graph Neural Network Model

We describe a simple, concrete base model. The idea is again to initialise every node $u \in V$ to a vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Then, the representation updates are given as:

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

where $\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)}$ are trainable parameter matrices and σ is an element-wise non-linear function (e.g., ReLU), and $\mathbf{b^{(t)}} \in \mathbb{R}^{d(t)}$ is a bias term (which we will omit in the sequel).

Intuitively, this base model can be seen as an instance of message passing neural networks, where the aggregation function is the sum, and the combine function updates node embeddings using a linear combination with an element-wise non-linearity applied at the end:

The Basic Graph Neural Network Model

We describe a simple, concrete base model. The idea is again to initialise every node $u \in V$ to a vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Then, the representation updates are given as:

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

where $\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)}$ are trainable parameter matrices and σ is an element-wise non-linear function (e.g., ReLU), and $\mathbf{b^{(t)}} \in \mathbb{R}^{d(t)}$ is a bias term (which we will omit in the sequel).

Intuitively, this base model can be seen as an instance of message passing neural networks, where the aggregation function is the sum, and the combine function updates node embeddings using a linear combination with an element-wise non-linearity applied at the end:

$$aggregate^{(t)}\Big(\big\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\big\}\Big) = \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}$$

The Basic Graph Neural Network Model

We describe a simple, concrete base model. The idea is again to initialise every node $u \in V$ to a vector representation $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Then, the representation updates are given as:

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

where $\mathbf{W}_{self}^{(t)}, \mathbf{W}_{neigh}^{(t)} \in \mathbb{R}^{d(t) \times d(t-1)}$ are trainable parameter matrices and σ is an element-wise non-linear function (e.g., ReLU), and $\mathbf{b^{(t)}} \in \mathbb{R}^{d(t)}$ is a bias term (which we will omit in the sequel).

Intuitively, this base model can be seen as an instance of message passing neural networks, where the aggregation function is the sum, and the combine function updates node embeddings using a linear combination with an element-wise non-linearity applied at the end:

$$aggregate^{(t)}\Big(\big\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\big\}\Big) = \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}$$

$$combine^{(t)}\Big(\mathbf{h}_u^{(t-1)}, aggregate^{(t)}\Big(\big\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\big\}\Big)\Big) = \sigma\Big(\mathbf{W}_{self}^{(t)}\mathbf{h}_u^{(t-1)} + \mathbf{W}_{neigh}^{(t)} aggregate^{(t)}\Big(\big\{\mathbf{h}_v^{(t-1)} \mid v \in N(u)\big\}\Big)\Big)$$

Message Passing With Self-Loops

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

That is, we can define message passing solely based on aggregate function as:

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

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

That is, we can define message passing solely based on aggregate function as:

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

Here, the aggregation is not only over the set the node's neighbours, but also the node itself.

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

That is, we can define message passing solely based on aggregate function as:

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

Here, the aggregation is not only over the set the node's neighbours, but also the node itself.

This simplifies the message passing. For example, the base model will then simplify to a model where we do not distinguish between trainable matrices any more:

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

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

That is, we can define message passing solely based on aggregate function as:

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

Here, the aggregation is not only over the set the node's neighbours, but also the node itself.

This simplifies the message passing. For example, the base model will then simplify to a model where we do not distinguish between trainable matrices any more:

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

However, this also severely limits the expressivity of the MPNN, as the information coming from the node's neighbours cannot be differentiated from the information from the node itself!

Message Passing With Self-Loops

Conceptually, it might appear more convenient to unify combine and aggregate functions, by (implicitly) adding self-loops to the nodes.

That is, we can define message passing solely based on aggregate function as:

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

Here, the aggregation is not only over the set the node's neighbours, but also the node itself.

This simplifies the message passing. For example, the base model will then simplify to a model where we do not distinguish between trainable matrices any more:

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

However, this also severely limits the expressivity of the MPNN, as the information coming from the node's neighbours cannot be differentiated from the information from the node itself!

We focus on the message passing approach without self-loops, unless explicitly mentioned otherwise.

Graph Pooling

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Final representation: For a given graph G, let us denote the final graph representation as $\mathbf{z}_G = \mathbf{h}_G^{(k)}$. There are various ways of defining $\mathbf{z}_G$ which eventually lead to different models.

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Final representation: For a given graph G, let us denote the final graph representation as $\mathbf{z}_G = \mathbf{h}_G^{(k)}$. There are various ways of defining $\mathbf{z}_G$ which eventually lead to different models.

Specifically, we need a mapping from the set of all the node embeddings $\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}$ to $\mathbf{z}_G$. This is very similar to the aggregate function (whose domain is also a set of embeddings) with the only difference being that aggregation operates over the local neighbourhood only.

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Final representation: For a given graph G, let us denote the final graph representation as $\mathbf{z}_G = \mathbf{h}_G^{(k)}$. There are various ways of defining $\mathbf{z}_G$ which eventually lead to different models.

Specifically, we need a mapping from the set of all the node embeddings $\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}$ to $\mathbf{z}_G$. This is very similar to the aggregate function (whose domain is also a set of embeddings) with the only difference being that aggregation operates over the local neighbourhood only.

In principle, any aggregation function can also be used to generate $\mathbf{z}_G$, and common choices are sum, or mean, which are then typically normalised with respect to, e.g., the size of the nodes.

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Final representation: For a given graph G, let us denote the final graph representation as $\mathbf{z}_G = \mathbf{h}_G^{(k)}$. There are various ways of defining $\mathbf{z}_G$ which eventually lead to different models.

Specifically, we need a mapping from the set of all the node embeddings $\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}$ to $\mathbf{z}_G$. This is very similar to the aggregate function (whose domain is also a set of embeddings) with the only difference being that aggregation operates over the local neighbourhood only.

In principle, any aggregation function can also be used to generate $\mathbf{z}_G$, and common choices are sum, or mean, which are then typically normalised with respect to, e.g., the size of the nodes.

There are various methods for graph, or more generally, relational pooling (Murphy et al., 2019).

Graph Pooling

We have defined a message passing approach to produce a set of node embeddings, but we are also interested in making predictions at the graph level.

Embedding for the entire graph: This task is often referred to as graph pooling, since we "pool" together the node embeddings in order to learn an embedding of the entire graph (Hamilton, 2020).

Final representation: For a given graph G, let us denote the final graph representation as $\mathbf{z}_G = \mathbf{h}_G^{(k)}$. There are various ways of defining $\mathbf{z}_G$ which eventually lead to different models.

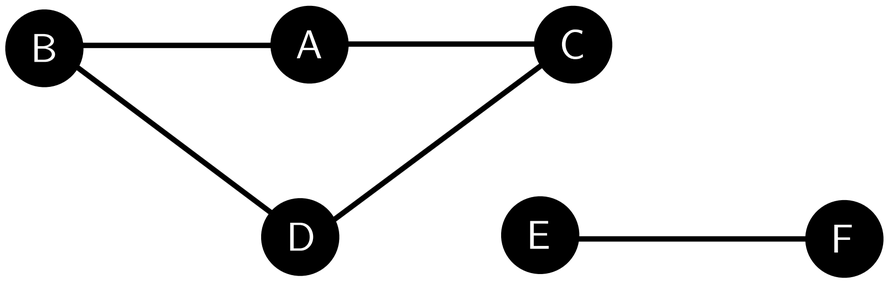
Specifically, we need a mapping from the set of all the node embeddings $\{\mathbf{z}_{u_1} \ldots \mathbf{z}_{u_n}\}$ to $\mathbf{z}_G$. This is very similar to the aggregate function (whose domain is also a set of embeddings) with the only difference being that aggregation operates over the local neighbourhood only.

In principle, any aggregation function can also be used to generate $\mathbf{z}_G$, and common choices are sum, or mean, which are then typically normalised with respect to, e.g., the size of the nodes.

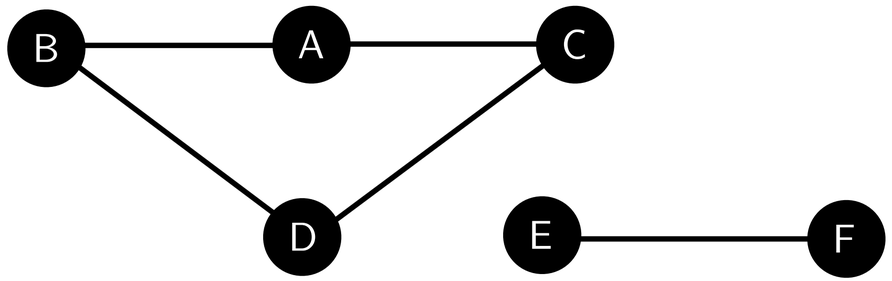
There are various methods for graph, or more generally, relational pooling (Murphy et al., 2019).

A Limitation of Message Passing

A Limitation of Message Passing

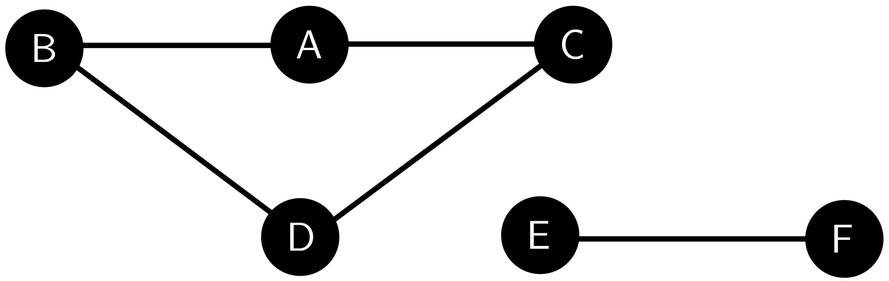


A Limitation of Message Passing



Problem: We have defined MPNNs which learn embeddings of nodes and graphs.The presented message passing approach, however, is local, e.g., for a disconnected graph no information will flow across disjoint subgraphs. This is a serious limitation!

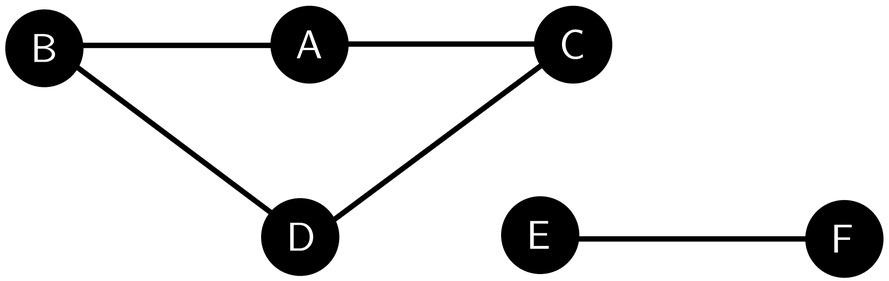
A Limitation of Message Passing



Problem: We have defined MPNNs which learn embeddings of nodes and graphs.The presented message passing approach, however, is local, e.g., for a disconnected graph no information will flow across disjoint subgraphs. This is a serious limitation!

Remark: A graph embedding is necessarily global in the sense that it is composed of all nodes, but it is not a solution to the above-mentioned problem: During message passing there are still no messages between disjoint subgraphs and so the learned node embeddings are "blind" to other embeddings in disjoint subgraphs. The graph embedding, which is based on node embeddings, will necessarily have induced limitations.

A Limitation of Message Passing



Problem: We have defined MPNNs which learn embeddings of nodes and graphs.The presented message passing approach, however, is local, e.g., for a disconnected graph no information will flow across disjoint subgraphs. This is a serious limitation!

Remark: A graph embedding is necessarily global in the sense that it is composed of all nodes, but it is not a solution to the above-mentioned problem: During message passing there are still no messages between disjoint subgraphs and so the learned node embeddings are "blind" to other embeddings in disjoint subgraphs. The graph embedding, which is based on node embeddings, will necessarily have induced limitations.

Solution: To break this behaviour, a standard approach is to use a global feature computation on each layer of the MPNN (Battaglia et al., 2018), also called a global attribute computation, or global readout.

Message Passing with Global Readout

Message Passing with Global Readout

We present a simple extension to MPNNs by allowing global readouts, where in each layer we also compute a feature vector for the whole graph G and combine it with local aggregations.

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

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

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

We present a simple extension to MPNNs by allowing global readouts, where in each layer we also compute a feature vector for the whole graph G and combine it with local aggregations.

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

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

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 generalised 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.

Message Passing with Global Readout

We present a simple extension to MPNNs by allowing global readouts, where in each layer we also compute a feature vector for the whole graph G and combine it with local aggregations.

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

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

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 generalised 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 seemingly simple reformulation makes a significant difference in terms of the expressive power of MPNNs (Barcelo et al., 2020), as we shall see later in the course.

Generalised Message Passing

Generalised Message Passing

Generalised message passing (Battaglia et al., 2018): The main idea is to define 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.

Generalised Message Passing

Generalised message passing (Battaglia et al., 2018): The main idea is to define 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.

The idea is to generate hidden embeddings $\mathbf{h}_{(u,v)}$ for each edge (u,v) in the graph G, as well as an embedding $\mathbf{h}_G$ corresponding to the entire graph.

Generalised Message Passing

Generalised message passing (Battaglia et al., 2018): The main idea is to define 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.

The idea is to generate hidden embeddings $\mathbf{h}_{(u,v)}$ for each edge (u, v) in the graph G, as well as an embedding $\mathbf{h}_G$ corresponding to the entire graph.

This allows the message passing model to easily integrate edge and graph-level features, especially in the multi-relational context, where each edge can be labelled differently.

Graph Representation Learning Tasks

Node Classification

Node Classification

Task: Given a single graph $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, 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 $v \in V \backslash V_{tr}$.

Node Classification

Task: Given a single graph $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, 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 $v \in V \backslash V_{tr}$.

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

Node Classification

Task: Given a single graph $G = (V, E)$ with a feature matrix $\mathbf{X} \in \mathbb{R}^{d \times V_G}$, 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 $v \in V \backslash V_{tr}$.

Example (Kipf and Welling, 2017): Consider citation networks such as Citeseer, where nodes represent papers, and edges denote citation links, and a subset of the nodes/papers 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 $\mathbf{y}_u$.

As usual, the gradient of the loss is backpropagated through the parameters of the GNN using stochastic gradient descent or one of its variants.

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.

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 for learning (Hamilton, 2020):

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 for learning (Hamilton, 2020):

- Transductive: In this regime 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.

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 for learning (Hamilton, 2020):

- Transductive: In this regime 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: In this regime, not all test nodes are observed during training. This means that 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 for learning (Hamilton, 2020):

- Transductive: In this regime 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: In this regime, not all test nodes are observed during training. This means that for some test nodes, neither the nodes themselves nor their edges (and hence their relation to other nodes) are known.

Transductive classification example: Suppose we have access to the full citation graph at training time in the citation networks task, and define a subset of the nodes as the test nodes. This is a transductive task, as the structural information of the nodes is available, and is used during the message passing.

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 for learning (Hamilton, 2020):

- Transductive: In this regime 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: In this regime, not all test nodes are observed during training. This means that for some test nodes, neither the nodes themselves nor their edges (and hence their relation to other nodes) are known.

Transductive classification example: Suppose we have access to the full citation graph at training time in the citation networks task, and define a subset of the nodes as the test nodes. This is a transductive task, as the structural information of the nodes is available, and is used during the message passing.

Inductive classification example: Suppose we only see a subgraph of the citation graph at training time in the citation networks task, and we define a set of nodes from a disjoint subgraph as test nodes. This is an inductive task, as we have not observed the test nodes during training.

Node Classification: Supervised or Semi-supervised?

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

- We train using training labels, as standard in supervised learning

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

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

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

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

This means that we trained using both labelled and unlabelled data.

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

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

This means that we trained using both labelled and unlabelled data.

In transductive node classification, which is by far the most common node classification task on graphs, we typically have access to the full graph, including all the test nodes (and, e.g., their neighbourhood), and this makes the term semi-supervised somewhat more appropriate (though, the standard semi-supervised setting also requires i.i.d. assumption, which doesn't hold for node classification).

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

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

This means that we trained using both labelled and unlabelled data.

In transductive node classification, which is by far the most common node classification task on graphs, we typically have access to the full graph, including all the test nodes (and, e.g., their neighbourhood), and this makes the term semi-supervised somewhat more appropriate (though, the standard semi-supervised setting also requires i.i.d. assumption, which doesn't hold for node classification).

Inductive node classification, on the other hand, can be seen as supervised learning.

Node Classification: Supervised or Semi-supervised?

Node classification is usually viewed as a semi-supervised learning task, and this should be understood in reference to transductive node classification. It is called semi-supervised since:

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

This means that we trained using both labelled and unlabelled data.

In transductive node classification, which is by far the most common node classification task on graphs, we typically have access to the full graph, including all the test nodes (and, e.g., their neighbourhood), and this makes the term semi-supervised somewhat more appropriate (though, the standard semi-supervised setting also requires i.i.d. assumption, which doesn't hold for node classification).

Inductive node classification, on the other hand, can be seen as supervised learning.

This can be noted as another instance where machine learning on graphs tends to differ from standard practices and terminology in machine learning.

Graph Classification

Graph Classification

Task: Given a set of graphs $\mathcal{G} = \{G_1, \ldots G_n\}$, where a subset of the graphs $\{(G_i, y_i) \mid G_i \in \mathcal{G}_{tr} \subset \mathcal{G}\}$ are labeled with a class, predict the labels of the remaining (test) graphs, i.e., y_j for all $G_j \in \mathcal{G} \backslash \mathcal{G}_{tr}$.

Graph Classification

Task: Given a set of graphs $\mathcal{G} = \{G_1, \ldots G_n\}$, where a subset of the graphs $\{(G_i, y_i) \mid G_i \in \mathcal{G}_{tr} \subset \mathcal{G}\}$ are labeled with a class, predict the labels of the remaining (test) graphs, i.e., y_j for all $G_j \in \mathcal{G} \backslash \mathcal{G}_{tr}$.

Example: The IMDB datasets (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.

Graph Classification

Task: Given a set of graphs $\mathcal{G} = \{G_1, \ldots G_n\}$, where a subset of the graphs $\{(G_i, y_i) \mid G_i \in \mathcal{G}_{tr} \subset \mathcal{G}\}$ are labeled with a class, predict the labels of the remaining (test) graphs, i.e., y_j for all $G_j \in \mathcal{G} \backslash \mathcal{G}_{tr}$.

Example: The IMDB datasets (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.

Training is similar to node classification, except that we use the final embedding of the graph instead of nodes.

Graph Classification

Task: Given a set of graphs $\mathcal{G} = \{G_1, \ldots G_n\}$, where a subset of the graphs $\{(G_i, y_i) \mid G_i \in \mathcal{G}_{tr} \subset \mathcal{G}\}$ are labeled with a class, predict the labels of the remaining (test) graphs, i.e., y_j for all $G_j \in \mathcal{G} \backslash \mathcal{G}_{tr}$.

Example: The IMDB datasets (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.

Training is similar to node classification, except that we use the final embedding of the graph instead of nodes.

Note that 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 Representation Learning Beyond Classification

Graph Representation Learning Beyond Classification

Intuitively, graph neural networks can be seen as encoders, which, for a given graph (or set of graphs), learn an embedding for each node (respectively, each graph).

Graph Representation Learning Beyond Classification

Intuitively, graph neural networks can be seen as encoders, which, for a given graph (or set of graphs), learn an embedding for each node (respectively, each graph).

Taking this broader perspective, we have the liberty to choose how to use the learned embeddings, i.e., they can be used for any standard machine learning task, e.g., classification, regression, clustering, etc. and so they have natural counterparts both on node and graph-level tasks.

Graph Representation Learning Beyond Classification

Intuitively, graph neural networks can be seen as encoders, which, for a given graph (or set of graphs), learn an embedding for each node (respectively, each graph).

Taking this broader perspective, we have the liberty to choose how to use the learned embeddings, i.e., they can be used for any standard machine learning task, e.g., classification, regression, clustering, etc. and so they have natural counterparts both on node and graph-level tasks.

Example (Gilmer et al., 2017): Given a graph representing the structure of a molecule, an interesting application is to build a regression model that could predict that molecule's toxicity or solubility, which is an application of graph regression.

Graph Representation Learning Beyond Classification

Intuitively, graph neural networks can be seen as encoders, which, for a given graph (or set of graphs), learn an embedding for each node (respectively, each graph).

Taking this broader perspective, we have the liberty to choose how to use the learned embeddings, i.e., they can be used for any standard machine learning task, e.g., classification, regression, clustering, etc. and so they have natural counterparts both on node and graph-level tasks.

Example (Gilmer et al., 2017): Given a graph representing the structure of a molecule, an interesting application is to build a regression model that could predict that molecule's toxicity or solubility, which is an application of graph regression.

Example: Similarly, if we are interested in applications such as community detection, this can be framed as a node clustering problem. In a similar vein, we may want to cluster a set of graphs, i.e., graph clustering.

Graph Representation Learning Beyond Classification

Intuitively, graph neural networks can be seen as encoders, which, for a given graph (or set of graphs), learn an embedding for each node (respectively, each graph).

Taking this broader perspective, we have the liberty to choose how to use the learned embeddings, i.e., they can be used for any standard machine learning task, e.g., classification, regression, clustering, etc. and so they have natural counterparts both on node and graph-level tasks.

Example (Gilmer et al., 2017): Given a graph representing the structure of a molecule, an interesting application is to build a regression model that could predict that molecule's toxicity or solubility, which is an application of graph regression.

Example: Similarly, if we are interested in applications such as community detection, this can be framed as a node clustering problem. In a similar vein, we may want to cluster a set of graphs, i.e., graph clustering.

Graph clustering is an unsupervised learning task, a natural extension of standard clustering.

Perspectives on Graph Neural Networks

From Convolutions to Graph Convolutions

From Convolutions to Graph Convolutions

Convolutional neural networks (CNNs) have been very successful in machine learning problems on grid-structured data, especially for data that requires translational equivariance/invariance with respect to this grid.

From Convolutions to Graph Convolutions

Convolutional neural networks (CNNs) have been very successful in machine learning problems on grid-structured data, especially for data that requires translational equivariance/invariance with respect to this grid.

This is one of the historical perspectives behind the development of graph neural networks: The main idea is to generalise Euclidean convolutions to the graph domain (Bruna et al., 2014)!

From Convolutions to Graph Convolutions

Convolutional neural networks (CNNs) have been very successful in machine learning problems on grid-structured data, especially for data that requires translational equivariance/invariance with respect to this grid.

This is one of the historical perspectives behind the development of graph neural networks: The main idea is to generalise Euclidean convolutions to the graph domain (Bruna et al., 2014)!

This line of research has led to graph convolutional networks (GCNs), a very popular GNN model (Kipf and Welling, 2016), which also falls into the class of MPNNs.

From Convolutions to Graph Convolutions

Convolutional neural networks (CNNs) have been very successful in machine learning problems on grid-structured data, especially for data that requires translational equivariance/invariance with respect to this grid.

This is one of the historical perspectives behind the development of graph neural networks: The main idea is to generalise Euclidean convolutions to the graph domain (Bruna et al., 2014)!

This line of research has led to graph convolutional networks (GCNs), a very popular GNN model (Kipf and Welling, 2016), which also falls into the class of MPNNs.

Intuitively, the graph Laplacian allows a natural link between graphs and vector spaces, and so the structure of the graph can be exploited with the spectrum of its graph-Laplacian. The main idea is then to use this to generalise the convolution operator to graph domains.

From Convolutions to Graph Convolutions

Convolutional neural networks (CNNs) have been very successful in machine learning problems on grid-structured data, especially for data that requires translational equivariance/invariance with respect to this grid.

This is one of the historical perspectives behind the development of graph neural networks: The main idea is to generalise Euclidean convolutions to the graph domain (Bruna et al., 2014)!

This line of research has led to graph convolutional networks (GCNs), a very popular GNN model (Kipf and Welling, 2016), which also falls into the class of MPNNs.

Intuitively, the graph Laplacian allows a natural link between graphs and vector spaces, and so the structure of the graph can be exploited with the spectrum of its graph-Laplacian. The main idea is then to use this to generalise the convolution operator to graph domains.

We will discuss and introduce graph convolutional networks in Lecture 4!

Graph Isomorphism Test

Graph Isomorphism Test

Another historical perspective is offered through the connection to the fundamental problem of graph isomorphism.

Graph Isomorphism Test

Another historical perspective is offered through the connection to the fundamental problem of graph isomorphism.

Intuitively, classifying graph-structured data requires the ability to distinguish graphs; and message passing neural networks cannot distinguish all graphs, and so they have limited expressive power!

Graph Isomorphism Test

Another historical perspective is offered through the connection to the fundamental problem of graph isomorphism.

Intuitively, classifying graph-structured data requires the ability to distinguish graphs; and message passing neural networks cannot distinguish all graphs, and so they have limited expressive power!

The connection to graph isomorphism testing offers many interesting and theoretical insights, and has contributed a great deal to our understanding of graph neural networks, especially in terms of their expressive power (Morris et al., 2018, Xu et al, 2019).

Graph Isomorphism Test

Another historical perspective is offered through the connection to the fundamental problem of graph isomorphism.

Intuitively, classifying graph-structured data requires the ability to distinguish graphs; and message passing neural networks cannot distinguish all graphs, and so they have limited expressive power!

The connection to graph isomorphism testing offers many interesting and theoretical insights, and has contributed a great deal to our understanding of graph neural networks, especially in terms of their expressive power (Morris et al., 2018, Xu et al, 2019).

We will cover these aspects in Lecture 5.

Probabilistic Graphical Models

Probabilistic Graphical Models

It is also possible to draw connections from traditional machine learning models, i.e., through probabilistic graphical models.

Probabilistic Graphical Models

It is also possible to draw connections from traditional machine learning models, i.e., through probabilistic graphical models.

Message passing algorithms are used in the context of probabilistic graphical models, and can be dated back to belief propagation in Bayesian networks (Pearl, 82).

Probabilistic Graphical Models

It is also possible to draw connections from traditional machine learning models, i.e., through probabilistic graphical models.

Message passing algorithms are used in the context of probabilistic graphical models, and can be dated back to belief propagation in Bayesian networks (Pearl, 82).

(Dai et al., 2016) offers an alternative perspective for graph neural networks based on probabilistic graphical models: The message passing operation that underlies MPNNs can be viewed as a neural network analogue of certain message passing algorithms that are commonly used for variational inference in probabilistic models to infer distributions over latent variables.

Probabilistic Graphical Models

It is also possible to draw connections from traditional machine learning models, i.e., through probabilistic graphical models.

Message passing algorithms are used in the context of probabilistic graphical models, and can be dated back to belief propagation in Bayesian networks (Pearl, 82).

(Dai et al., 2016) offers an alternative perspective for graph neural networks based on probabilistic graphical models: The message passing operation that underlies MPNNs can be viewed as a neural network analogue of certain message passing algorithms that are commonly used for variational inference in probabilistic models to infer distributions over latent variables.

We will not cover this aspect, but you will encounter, e.g., probabilistic graphical models, or variational inference as part of the Bayesian machine learning lectures.

Summary

Summary

- Relational inductive bias is crucial

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions
- Node and graph classification and beyond

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions
- Node and graph classification and beyond
- Historical perspectives for graph neural networks

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions
- Node and graph classification and beyond
- Historical perspectives for graph neural networks
- We have not covered any concrete model beyond the very basic one: Lecture 4!

Summary

- Relational inductive bias is crucial
- Message passing neural networks as a framework
- A basic message passing neural network model and its extensions
- Node and graph classification and beyond
- Historical perspectives for graph neural networks
- 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

- M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond Euclidean data, *IEEE Signal Processing Magazine,* 2017.
- N. Kriege, F. Johansson, and C. Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 2020.
- 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.
- 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, C¸ aglar 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.

References

- C. Morris, M. Ritzert, M. Fey, W. Hamilton,J. E. Lenssen, G. Rattan, and M. Grohe, Weisfeiler and Leman go neural: Higher-order graph neural networks. *AAAI*, 2018.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.
- 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.