Lecture 4: Message Passing Neural Network Architectures

Relational Learning

İsmail İlkan Ceylan

Advanced Topics in Machine Learning, University of Oxford

27.01.2021

Overview

Overview

- A glimpse at graph neural network models

Overview

- A glimpse at graph neural network models
- Graph convolutional networks

Overview

- A glimpse at graph neural network models
- Graph convolutional networks
- Gated graph neural networks

Overview

- A glimpse at graph neural network models
- Graph convolutional networks
- Gated graph neural networks
- Graph isomorphism networks

Overview

- A glimpse at graph neural network models
- Graph convolutional networks
- Gated graph neural networks
- Graph isomorphism networks
- Graph attention networks

Overview

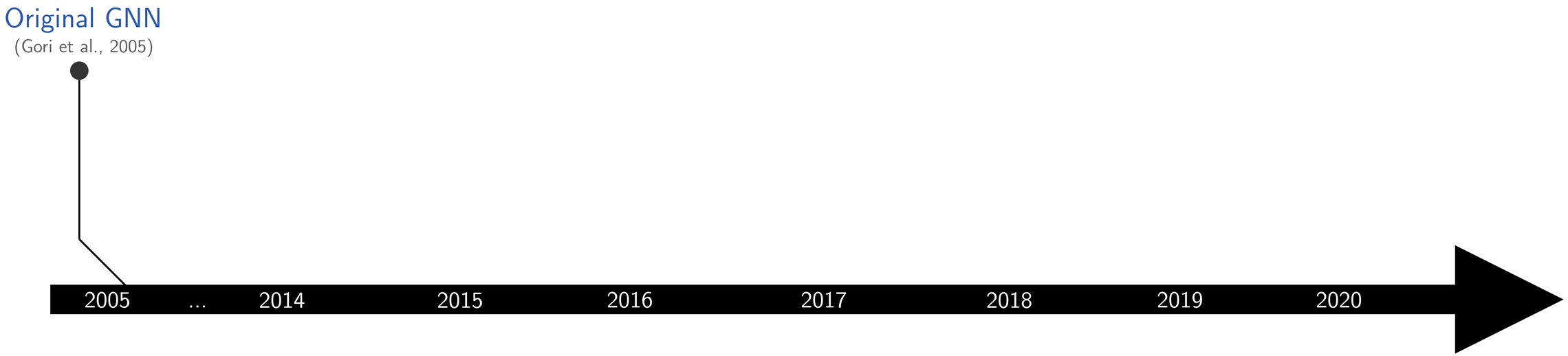
- A glimpse at graph neural network models
- Graph convolutional networks
- Gated graph neural networks
- Graph isomorphism networks
- Graph attention networks
- Discussions and limitations

Overview

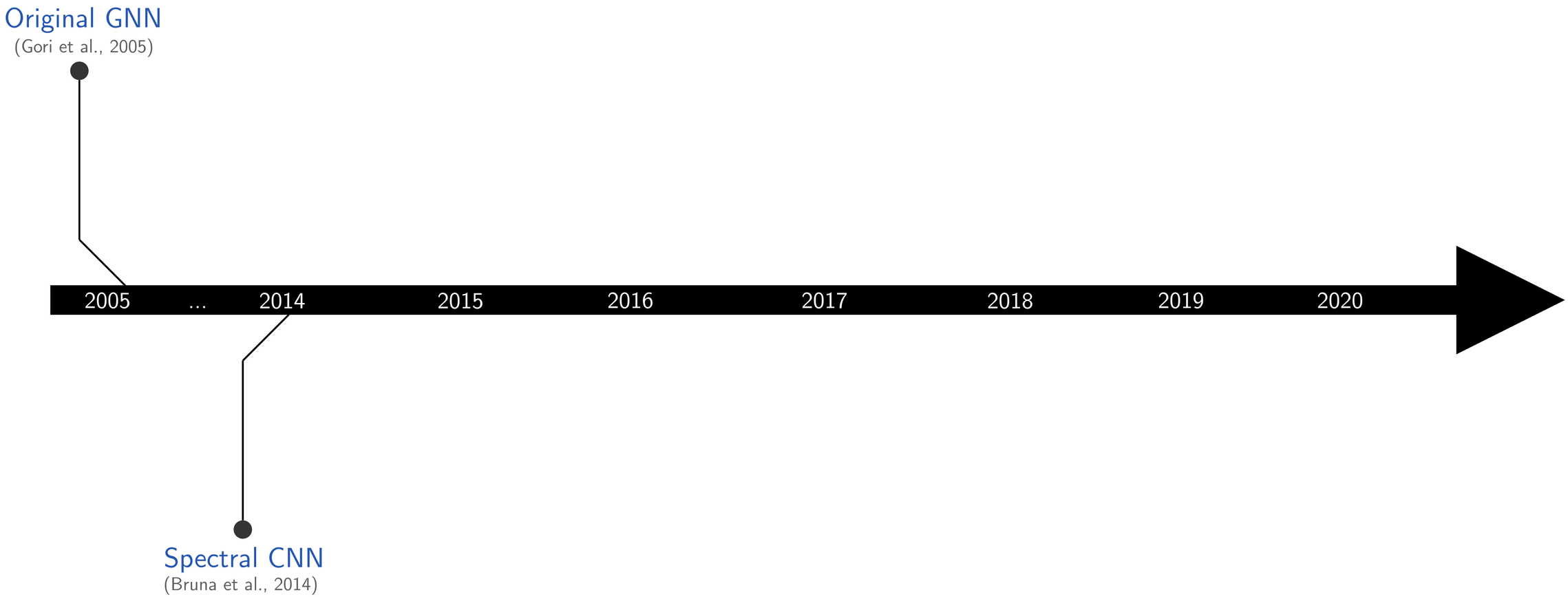
- A glimpse at graph neural network models
- Graph convolutional networks
- Gated graph neural networks
- Graph isomorphism networks
- Graph attention networks
- Discussions and limitations
- Summary

A Glimpse at Graph Neural Networks

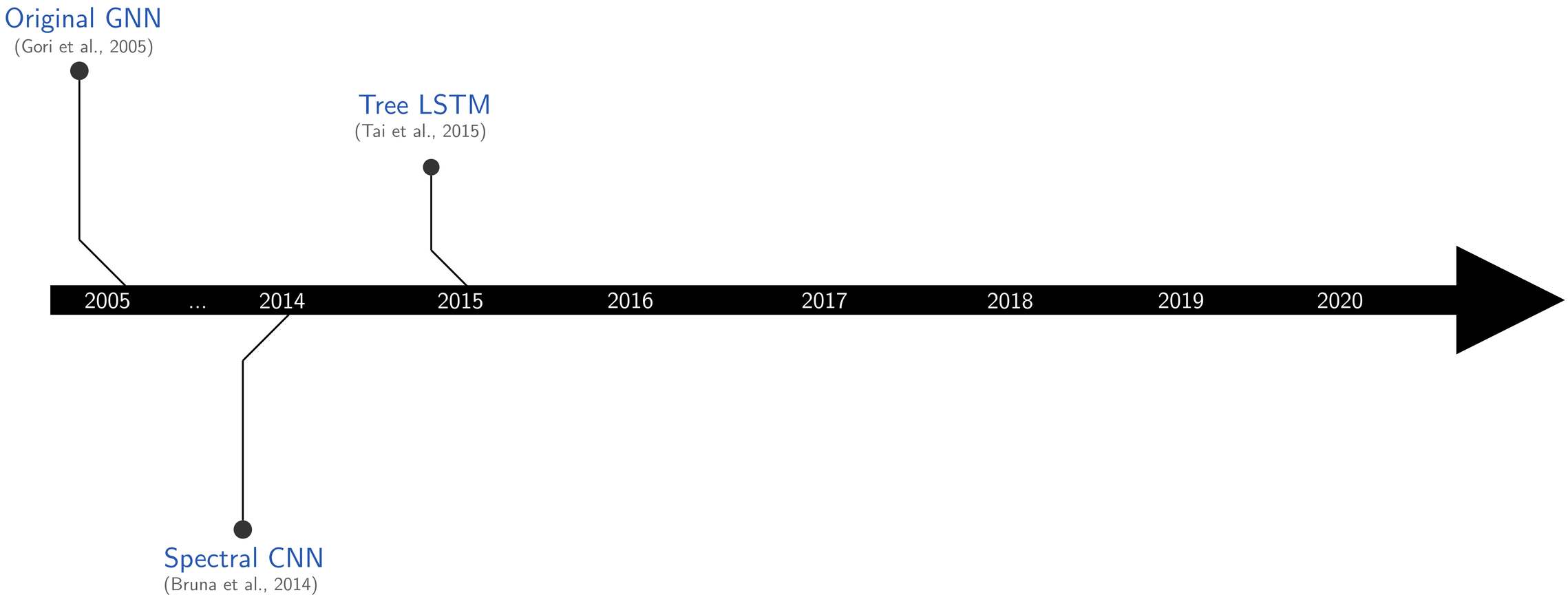
A Glimpse at Graph Neural Networks



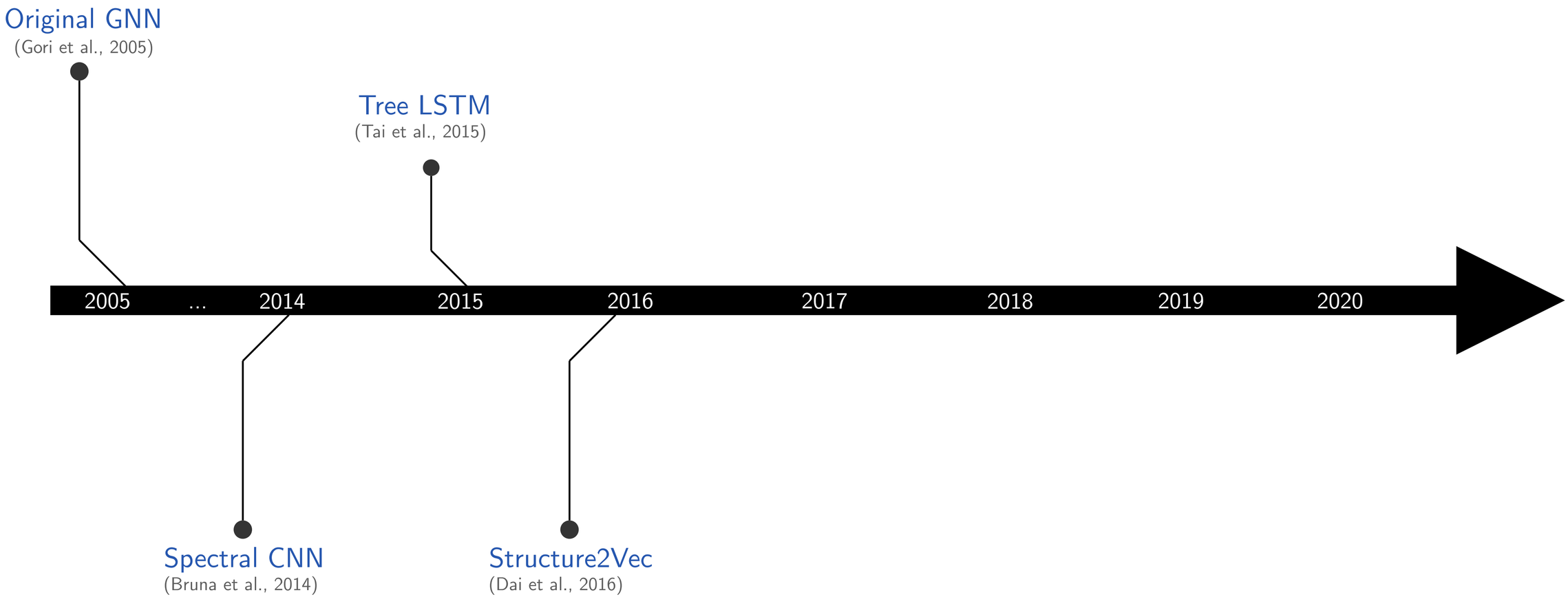
A Glimpse at Graph Neural Networks



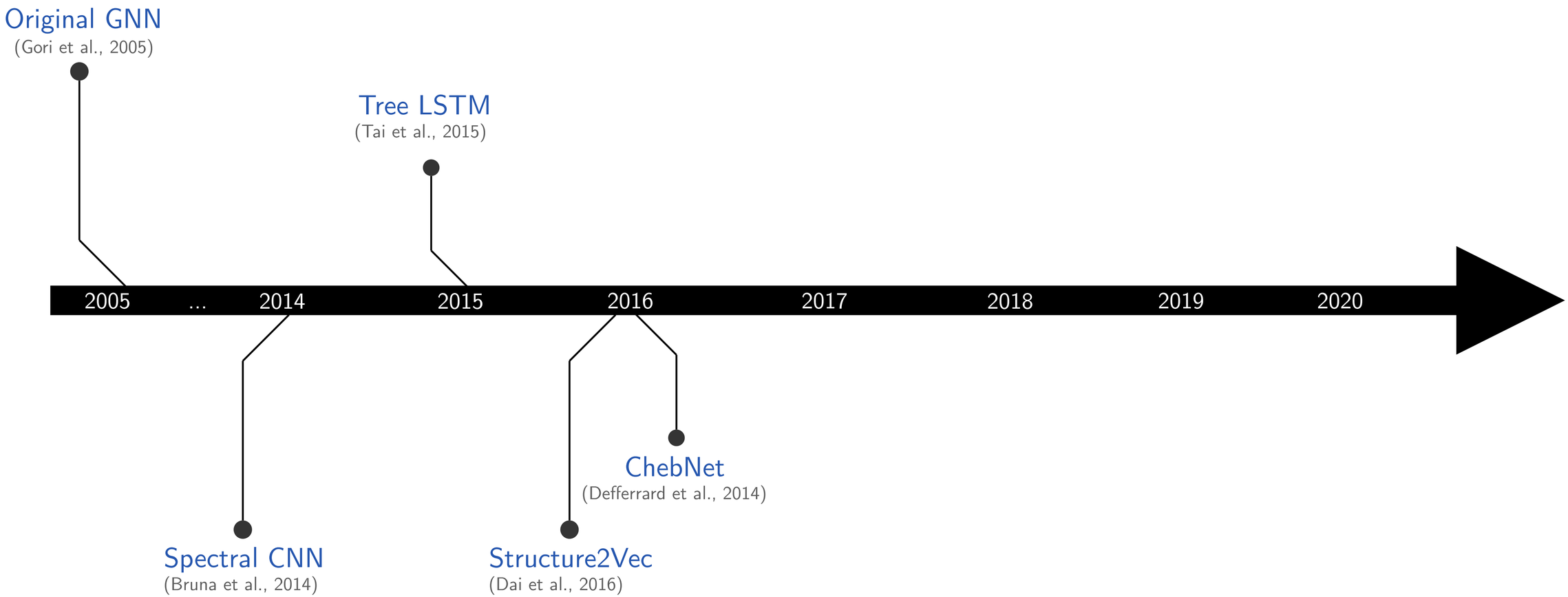
A Glimpse at Graph Neural Networks



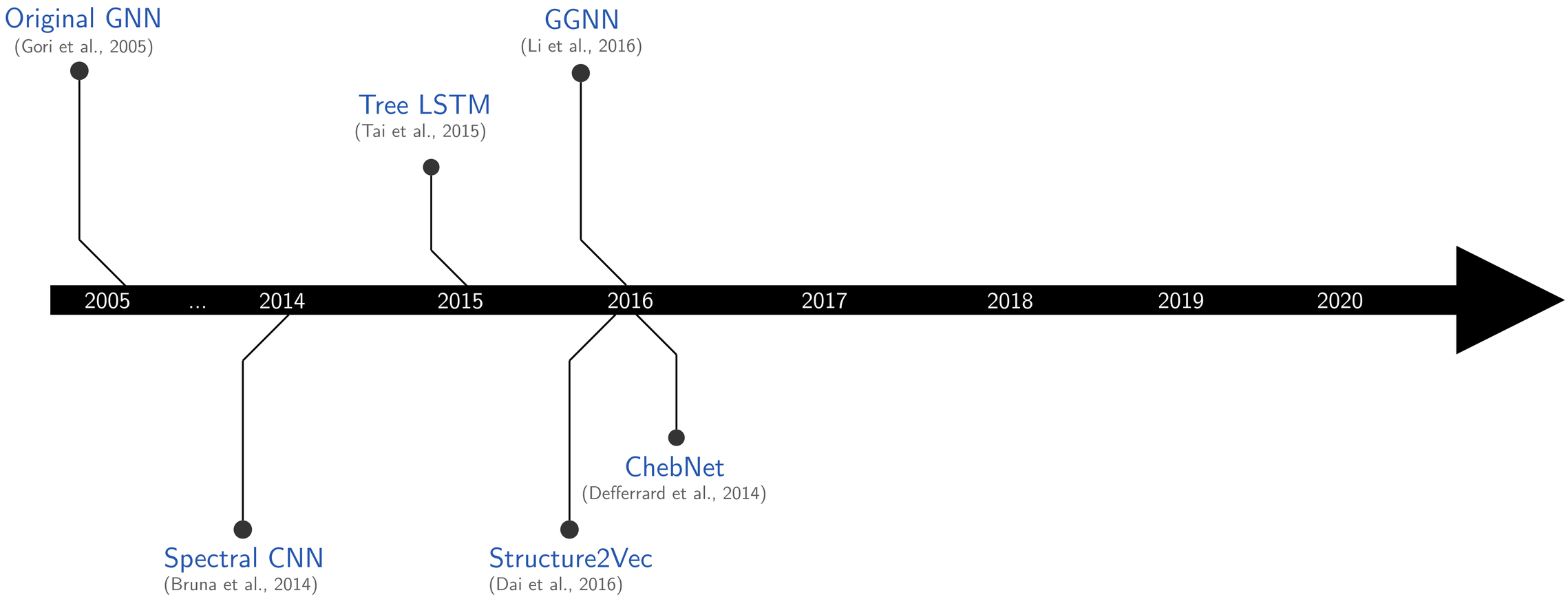
A Glimpse at Graph Neural Networks



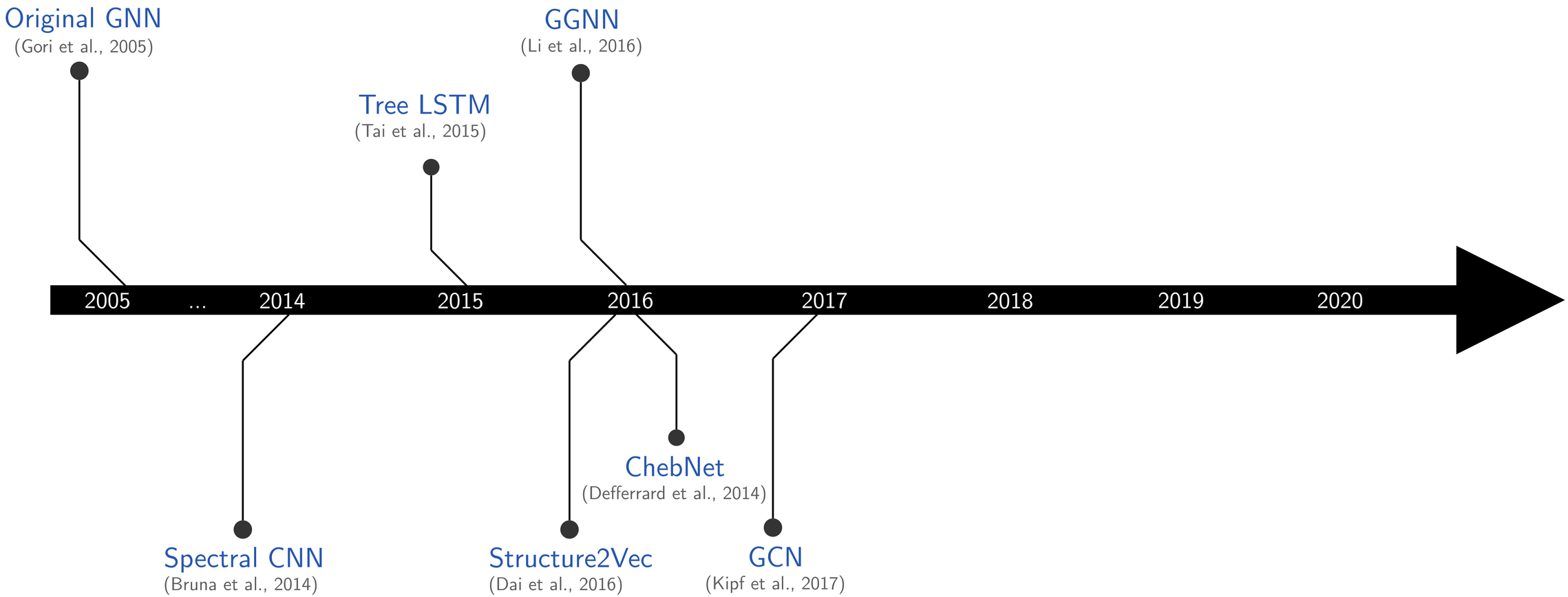
A Glimpse at Graph Neural Networks



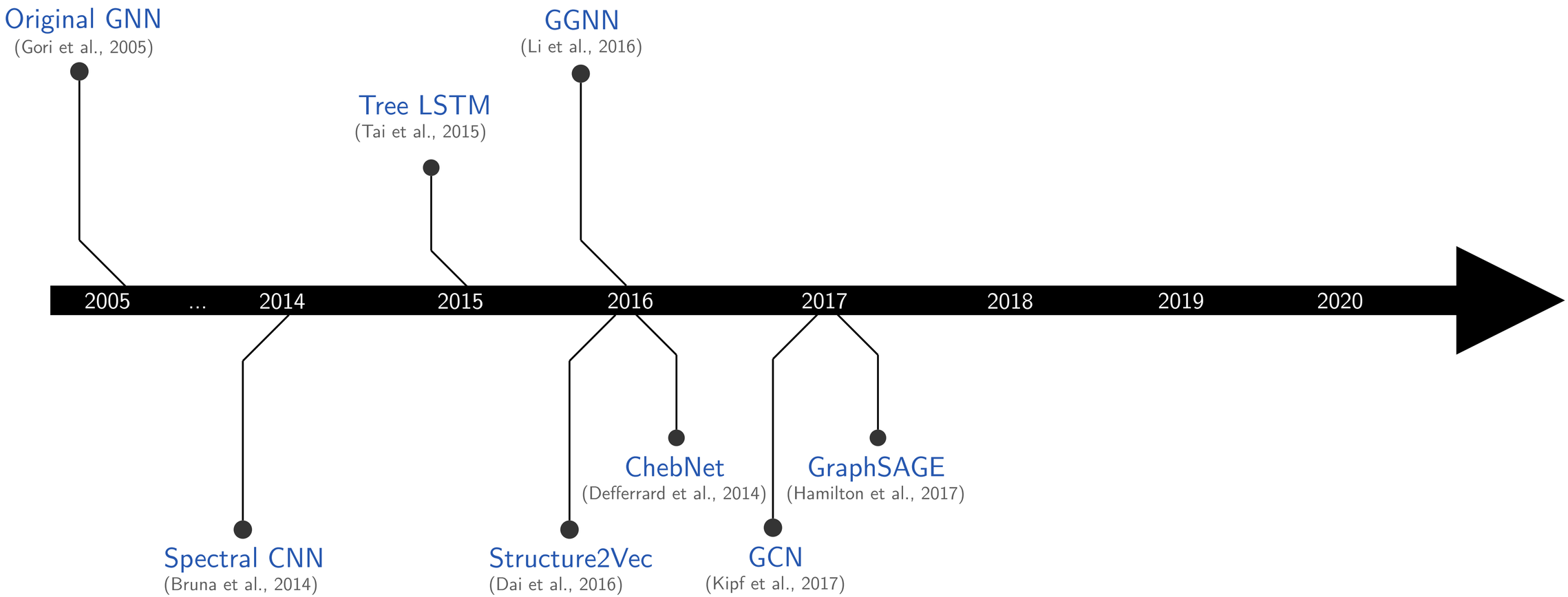
A Glimpse at Graph Neural Networks



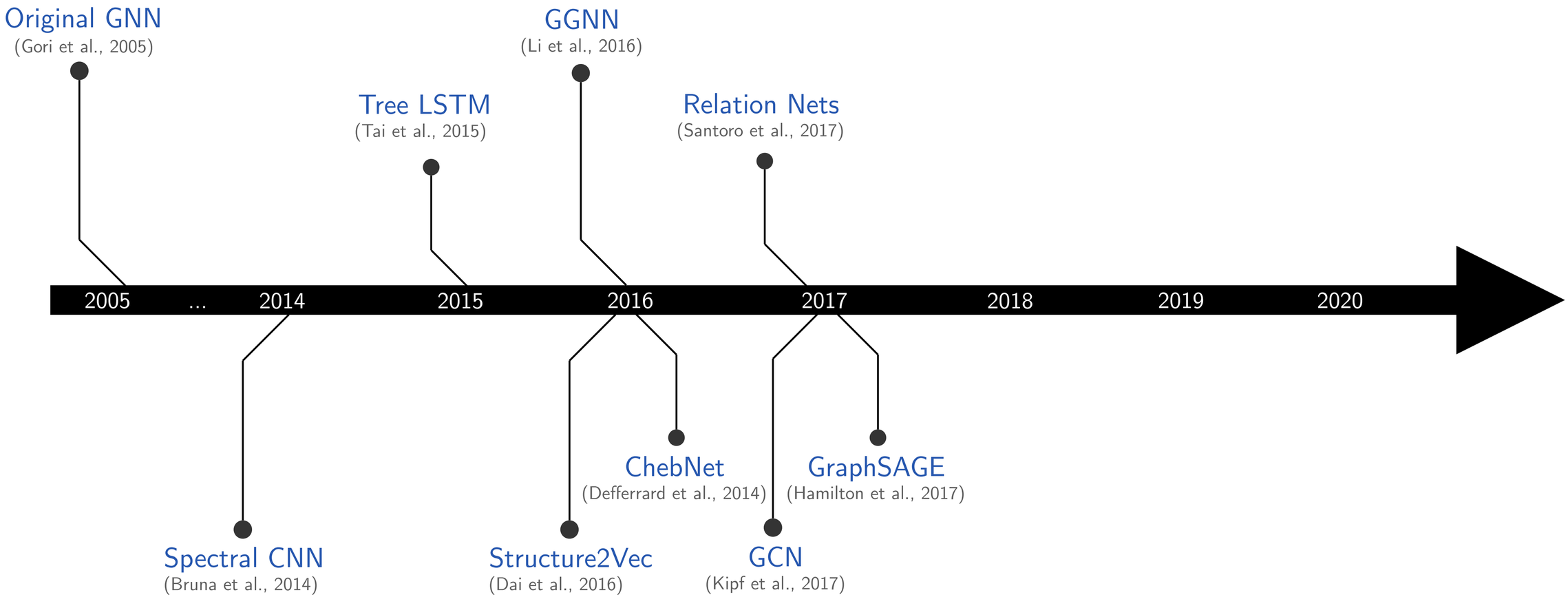
A Glimpse at Graph Neural Networks



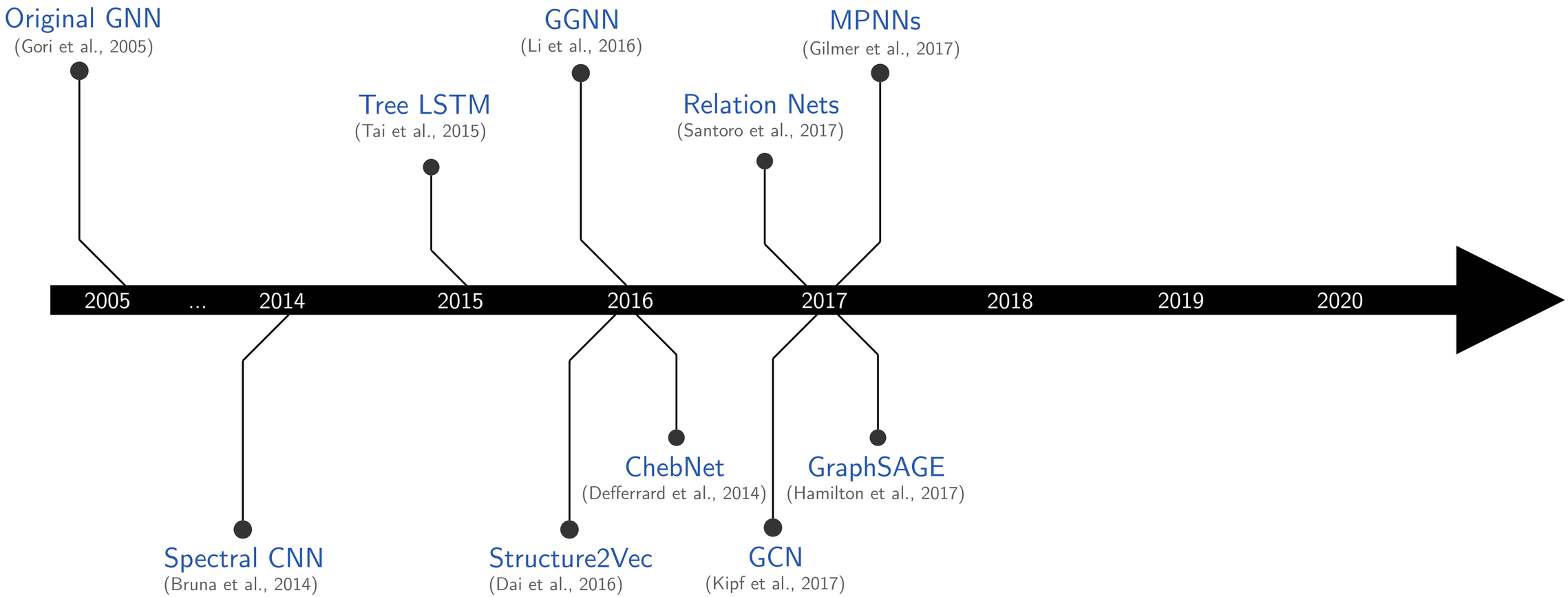
A Glimpse at Graph Neural Networks



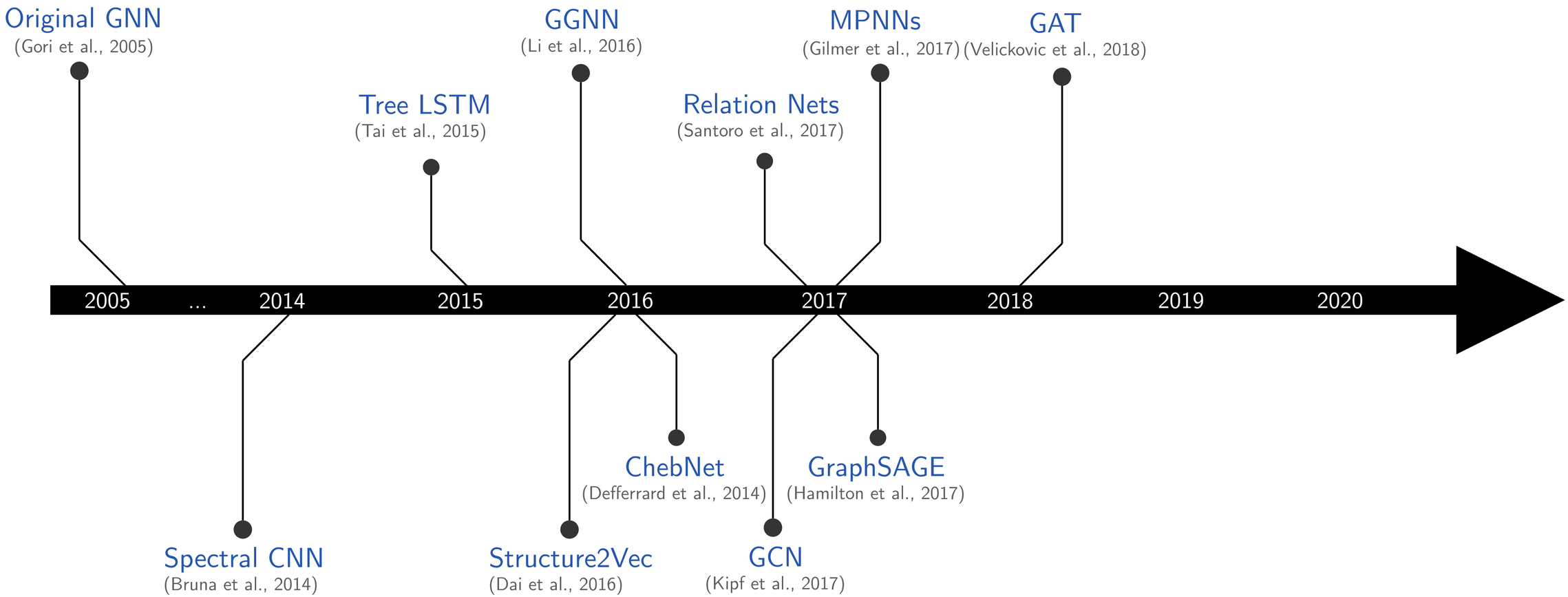
A Glimpse at Graph Neural Networks



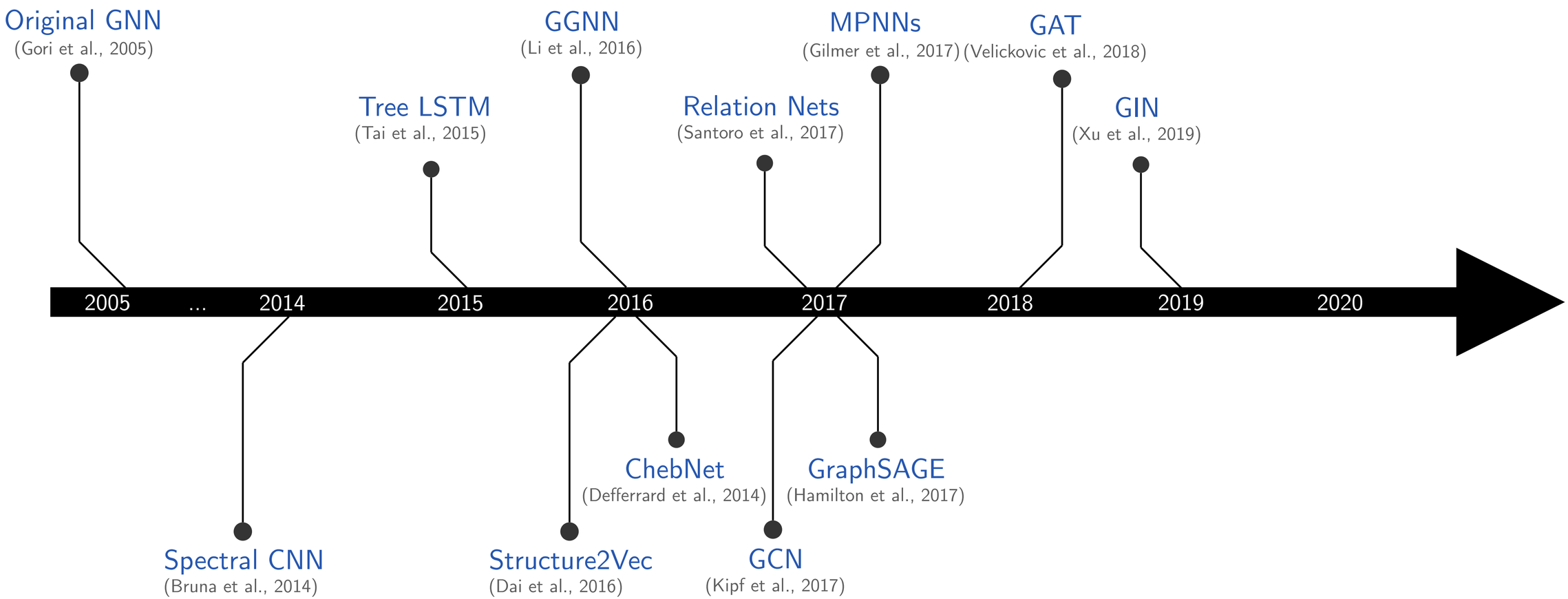
A Glimpse at Graph Neural Networks



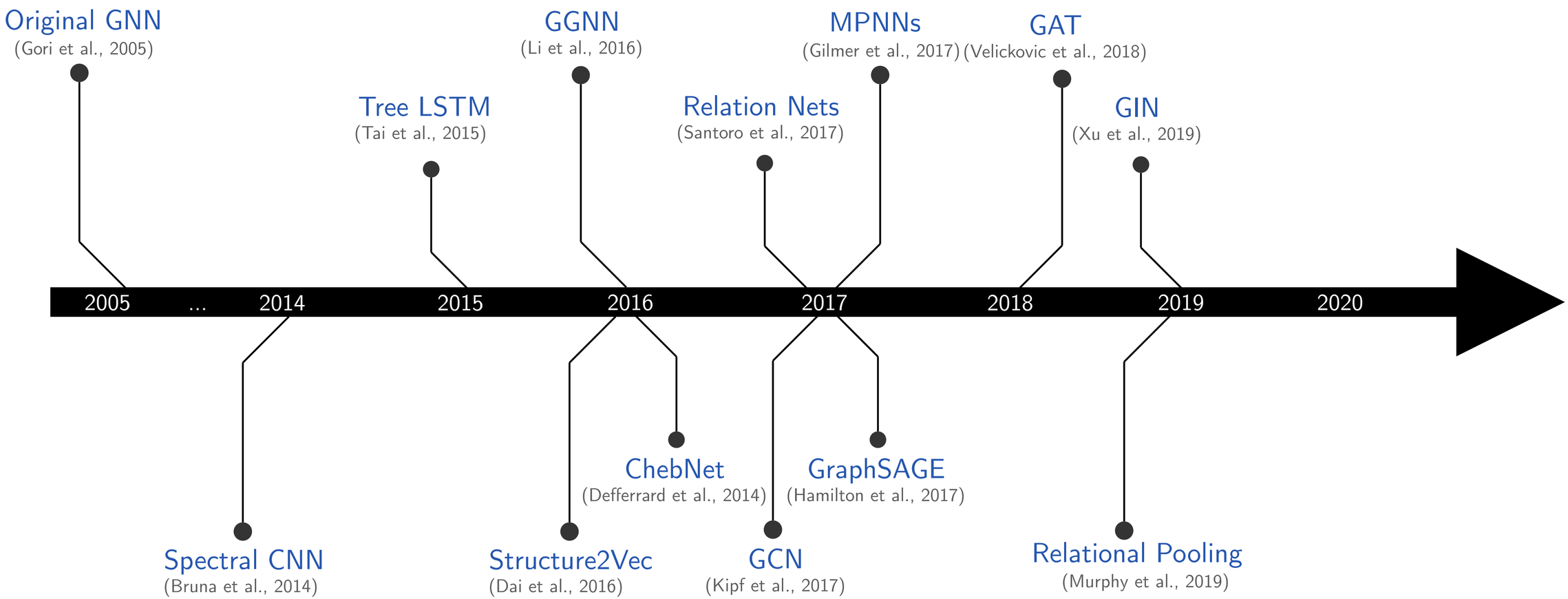
A Glimpse at Graph Neural Networks



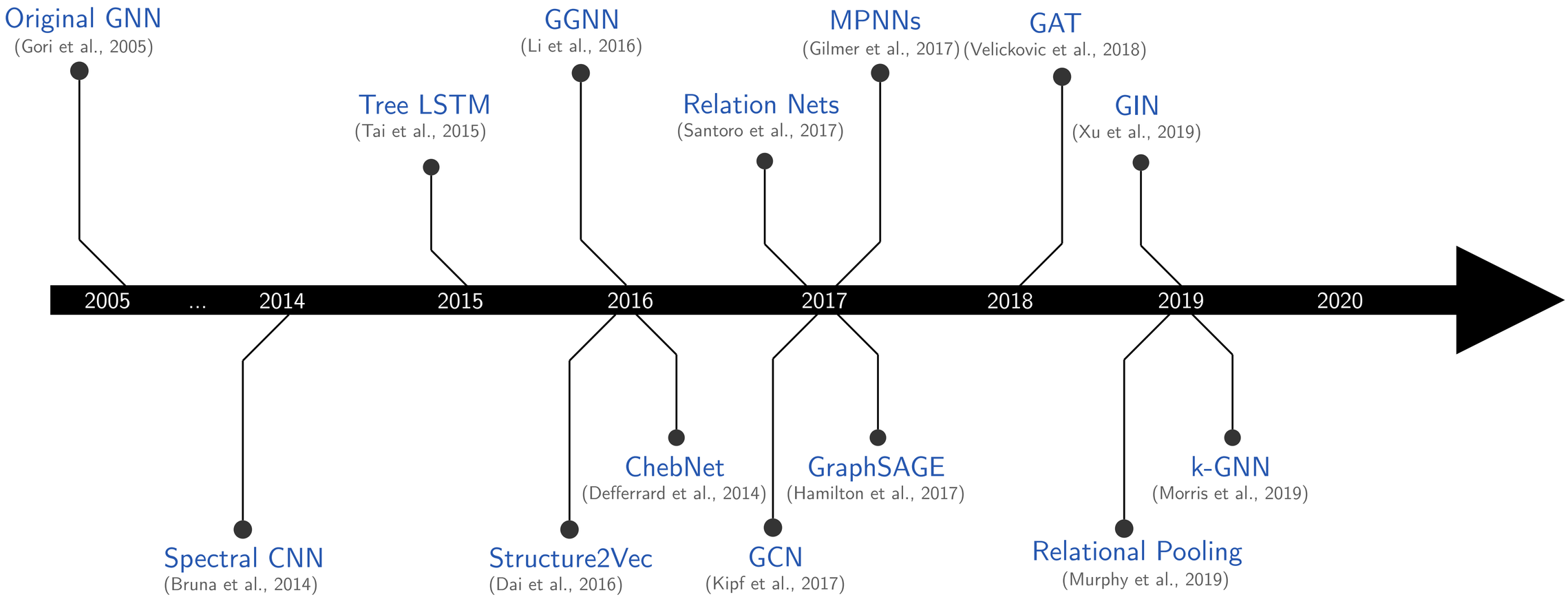
A Glimpse at Graph Neural Networks



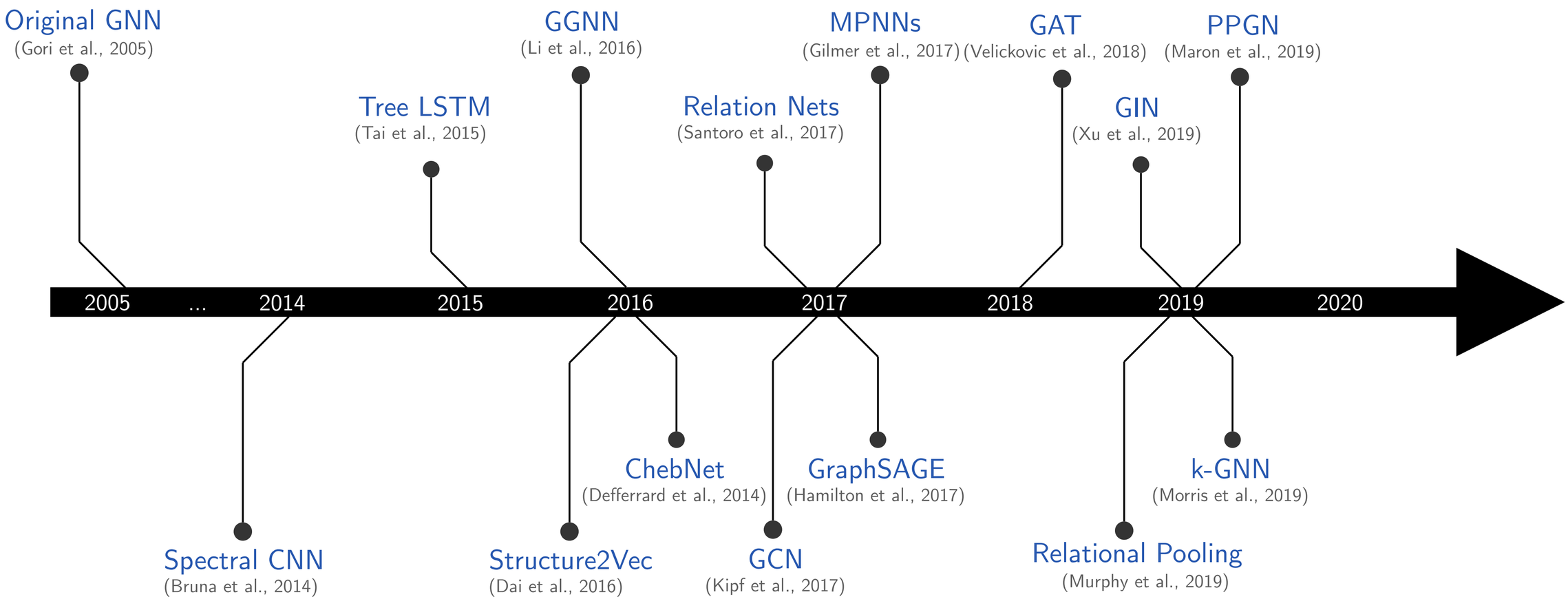
A Glimpse at Graph Neural Networks



A Glimpse at Graph Neural Networks



A Glimpse at Graph Neural Networks



Graph Convolutional Networks

From Convolutions to Graph Convolutions

From Convolutions to Graph Convolutions

Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) are motivated by the popular convolution operator, which is prevalent in signal processing, and is also key to convolutional neural networks (CNNs), and their successes in image processing.

From Convolutions to Graph Convolutions

Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) are motivated by the popular convolution operator, which is prevalent in signal processing, and is also key to convolutional neural networks (CNNs), and their successes in image processing.

Briefly, let f and h be two functions. Then, the convolution operation is defined as follows:

$$(f \star h)(\mathbf{x}) = \int_{\mathbb{R}^d} f(\mathbf{y})\, h(\mathbf{x} - \mathbf{y})\, d\mathbf{y}.$$

Intuitively, this means that the function h "slides" over the function f for every value $\mathbf{x}$, and returns a value reflecting the similarity between the two functions around the value $\mathbf{x}$.

From Convolutions to Graph Convolutions

Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017) are motivated by the popular convolution operator, which is prevalent in signal processing, and is also key to convolutional neural networks (CNNs), and their successes in image processing.

Briefly, let f and h be two functions. Then, the convolution operation is defined as follows:

$$(f \star h)(\mathbf{x}) = \int_{\mathbb{R}^d} f(\mathbf{y})\, h(\mathbf{x} - \mathbf{y})\, d\mathbf{y}.$$

Intuitively, this means that the function h "slides" over the function f for every value $\mathbf{x}$, and returns a value reflecting the similarity between the two functions around the value $\mathbf{x}$.

Convolution can be mapped to the frequency domain, such that the integration operation reduces to point-wise multiplication over the Fourier transforms of both functions:

$$(f \star h)(\mathbf{x}) = \mathscr{F}^{-1}(\mathscr{F}(f(\mathbf{x})) \circ \mathscr{F}(h(\mathbf{x}))),$$

where $\mathscr{F}, \mathscr{F}^{-1}$ denote the Fourier and inverse Fourier transform, respectively.

Properties of Convolution

Properties of Convolution

Convolutions are translation-equivariant:

$$f(t+a) \star g(t) = f(t) \star g(t+a) = (f \star g)(t+a),$$

which means that translating a signal and then convolving it is equivalent to convolving the signal and then translating the result.

Convolution allows for a more localised processing of information, and this offers a strong inductive bias.

This inductive bias is key to the successes for CNNs, as CNNs used convolutional filters to detect localised features, irrespective of their location in the image!

Properties of Convolution

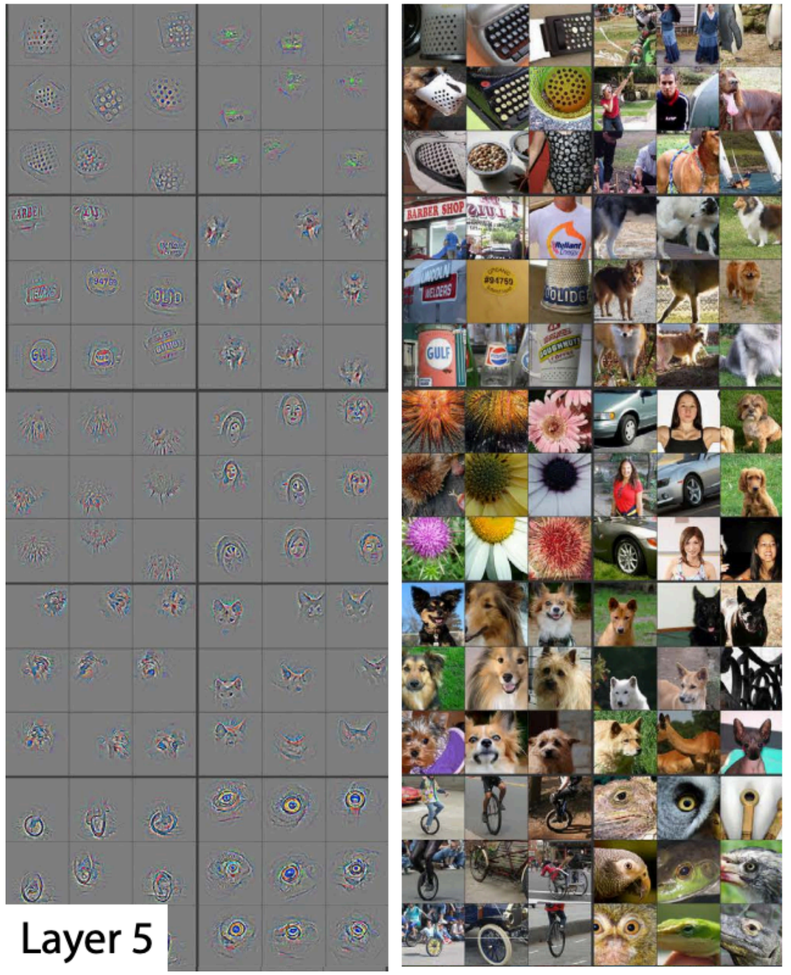
Convolutions are translation-equivariant:

$$f(t+a) \star g(t) = f(t) \star g(t+a) = (f \star g)(t+a),$$

which means that translating a signal and then convolving it is equivalent to convolving the signal and then translating the result.

Convolution allows for a more localised processing of information, and this offers a strong inductive bias.

This inductive bias is key to the successes for CNNs, as CNNs used convolutional filters to detect localised features, irrespective of their location in the image!



Excerpt from Figure 2 of (Zeiler and Fergus, 2014).

Euclidian Convolutions

Euclidian Convolutions

Problem: Convolutions are inherently defined over Euclidian spaces. For signals, they are defined over the real domain, and convolution in CNNs applies over a contiguous tensor.

Euclidian Convolutions

Problem: Convolutions are inherently defined over Euclidian spaces. For signals, they are defined over the real domain, and convolution in CNNs applies over a contiguous tensor.

Graphs are non-Euclidian, and do not admit a smooth structure.

Euclidian Convolutions

Problem: Convolutions are inherently defined over Euclidian spaces. For signals, they are defined over the real domain, and convolution in CNNs applies over a contiguous tensor.

Graphs are non-Euclidian, and do not admit a smooth structure.

Question: Can we lift convolutions to the graph domain, to take advantage of their inductive bias?

Euclidian Convolutions

Problem: Convolutions are inherently defined over Euclidian spaces. For signals, they are defined over the real domain, and convolution in CNNs applies over a contiguous tensor.

Graphs are non-Euclidian, and do not admit a smooth structure.

Question: Can we lift convolutions to the graph domain, to take advantage of their inductive bias?

Recall the basic MPNN model from Lecture 3, defined as:

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

Can we interpret this model in terms of convolutions?

Revisiting the Basic Model

Revisiting the Basic Model

This base MPNN model can be written using a graph-level equation as:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}_{self} + \mathbf{A}^{G}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\Big),$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ denotes the matrix of node representations at layer t and $\mathbf{A}^{G}$ is the adjacency matrix of G, and will omit the superscript G, for brevity.

Revisiting the Basic Model

This base MPNN model can be written using a graph-level equation as:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}_{self} + \mathbf{A}^{G}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\Big),$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ denotes the matrix of node representations at layer t and $\mathbf{A}^G$ is the adjacency matrix of G, and will omit the superscript G, for brevity.

This offers a new perspective: View this base model as a model that applies a convolutional filter at every iteration t, combined with some learnable weight matrices and a non-linearity.

Revisiting the Basic Model

This base MPNN model can be written using a graph-level equation as:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}_{self} + \mathbf{A}^{G}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\Big),$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ denotes the matrix of node representations at layer t and $\mathbf{A}^G$ is the adjacency matrix of G, and will omit the superscript G, for brevity.

This offers a new perspective: View this base model as a model that applies a convolutional filter at every iteration t, combined with some learnable weight matrices and a non-linearity.

Specifically, we define a filter $\mathbf{Q} = \mathbf{I} + \mathbf{A}$, where $\mathbf{I}$ is the identity matrix of G.

Revisiting the Basic Model

This base MPNN model can be written using a graph-level equation as:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}_{self} + \mathbf{A}^{G}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\Big),$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ denotes the matrix of node representations at layer t and $\mathbf{A}^G$ is the adjacency matrix of G, and will omit the superscript G, for brevity.

This offers a new perspective: View this base model as a model that applies a convolutional filter at every iteration t, combined with some learnable weight matrices and a non-linearity.

Specifically, we define a filter $\mathbf{Q} = \mathbf{I} + \mathbf{A}$, where $\mathbf{I}$ is the identity matrix of G.

We can then view the message passing layers in the basic model as a generalisation of the linear filter $\mathbf{Q}$ to more complex non-linear functions.

Revisiting the Basic Model

This base MPNN model can be written using a graph-level equation as:

$$\mathbf{H}^{(t)} = \sigma\Big(\mathbf{H}^{(t-1)}\mathbf{W}^{(t)}_{self} + \mathbf{A}^{G}\ \mathbf{H}^{(t-1)}\mathbf{W}_{neigh}\Big),$$

where $\mathbf{H}^{(t)} \in \mathbb{R}^{|V_G| \times d}$ denotes the matrix of node representations at layer t and $\mathbf{A}^G$ is the adjacency matrix of G, and will omit the superscript G, for brevity.

This offers a new perspective: View this base model as a model that applies a convolutional filter at every iteration t, combined with some learnable weight matrices and a non-linearity.

Specifically, we define a filter $\mathbf{Q} = \mathbf{I} + \mathbf{A}$, where $\mathbf{I}$ is the identity matrix of G.

We can then view the message passing layers in the basic model as a generalisation of the linear filter $\mathbf{Q}$ to more complex non-linear functions.

Convolution can be done based on spectral properties of the graph, and in this case, this is defined via the adjacency matrix!

Laplacian

Laplacian

If we use the adjacency matrix alone as a filter, only node degrees play an important role in the convolution, and this filter is too simple!

Laplacian

If we use the adjacency matrix alone as a filter, only node degrees play an important role in the convolution, and this filter is too simple!

Spectral properties of the graph are captured by other matrices such as the graph Laplacian, and this is another common means to convolve over graphs.

Laplacian

If we use the adjacency matrix alone as a filter, only node degrees play an important role in the convolution, and this filter is too simple!

Spectral properties of the graph are captured by other matrices such as the graph Laplacian, and this is another common means to convolve over graphs.

Recall that the Laplacian of a graph G is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of G.

Laplacian

If we use the adjacency matrix alone as a filter, only node degrees play an important role in the convolution, and this filter is too simple!

Spectral properties of the graph are captured by other matrices such as the graph Laplacian, and this is another common means to convolve over graphs.

Recall that the Laplacian of a graph G is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of G.

Intuitively, the Laplacian allocates different weights to neighbour nodes in a graph based on their overall role in the graph. In this respect, it is more informative than the adjacency matrix alone.

Laplacian

If we use the adjacency matrix alone as a filter, only node degrees play an important role in the convolution, and this filter is too simple!

Spectral properties of the graph are captured by other matrices such as the graph Laplacian, and this is another common means to convolve over graphs.

Recall that the Laplacian of a graph G is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix of G.

Intuitively, the Laplacian allocates different weights to neighbour nodes in a graph based on their overall role in the graph. In this respect, it is more informative than the adjacency matrix alone.

Problem: For general graphs, commutativity with the adjacency matrix (i.e., translation-equivariance) does not necessarily imply commutativity with the Laplacian!

Symmetric Normalised Filters

Symmetric Normalised Filters

Both Laplacian and adjacency variants are typically normalised, to ensure that they have bounded spectra, and thus ensure numerical stability.

Symmetric Normalised Filters

Both Laplacian and adjacency variants are typically normalised, to ensure that they have bounded spectra, and thus ensure numerical stability.

The symmetric normalised Laplacian is defined as: $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$

Symmetric Normalised Filters

Both Laplacian and adjacency variants are typically normalised, to ensure that they have bounded spectra, and thus ensure numerical stability.

The symmetric normalised Laplacian is defined as: $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$

The symmetric normalised adjacency matrix is defined as: $\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

Symmetric Normalised Filters

Both Laplacian and adjacency variants are typically normalised, to ensure that they have bounded spectra, and thus ensure numerical stability.

The symmetric normalised Laplacian is defined as: $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$

The symmetric normalised adjacency matrix is defined as: $\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

These matrices share the same eigenvectors! We observe the following relationship:

$$\mathbf{L}_{sym} = \mathbf{I} - \mathbf{A}_{sym} \Rightarrow \mathbf{L}_{sym} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top} \text{ and } \mathbf{A}_{sym} = \mathbf{U}(\mathbf{I} - \mathbf{\Lambda})\mathbf{U}^{\top},$$

where $\mathbf{U}$ is the shared set of eigenvectors and $\mathbf{\Lambda}$ is the diagonal matrix containing the Laplacian eigenvalues.

Symmetric Normalised Filters

Both Laplacian and adjacency variants are typically normalised, to ensure that they have bounded spectra, and thus ensure numerical stability.

The symmetric normalised Laplacian is defined as: $\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$

The symmetric normalised adjacency matrix is defined as: $\mathbf{A}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

These matrices share the same eigenvectors! We observe the following relationship:

$$\mathbf{L}_{sym} = \mathbf{I} - \mathbf{A}_{sym} \Rightarrow \mathbf{L}_{sym} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\top} \text{ and } \mathbf{A}_{sym} = \mathbf{U}(\mathbf{I} - \mathbf{\Lambda})\mathbf{U}^{\top},$$

where $\mathbf{U}$ is the shared set of eigenvectors and $\mathbf{\Lambda}$ is the diagonal matrix containing the Laplacian eigenvalues.

Defining filters based on one of these matrices implies commutativity with the other, which is a very convenient and desirable property!

Graph Convolutional Networks

Graph Convolutional Networks

Based on graph convolutions, the basic graph convolutional network (GCN) model is defined as follows:

$$\mathbf{H}^{(t)} = \sigma\Big(\hat{\mathbf{A}}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big),$$

where $\hat{\mathbf{A}} = \big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}\big(\mathbf{I}+\mathbf{A}\big)\big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}$ is a normalised version of adjacency matrix with self-loops and $\mathbf{W}^{(t)}$ is a learnable parameter matrix.

Graph Convolutional Networks

Based on graph convolutions, the basic graph convolutional network (GCN) model is defined as follows:

$$\mathbf{H}^{(t)} = \sigma\Big(\hat{\mathbf{A}}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big),$$

where $\hat{\mathbf{A}} = \big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}\big(\mathbf{I}+\mathbf{A}\big)\big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}$ is a normalised version of adjacency matrix with self-loops and $\mathbf{W}^{(t)}$ is a learnable parameter matrix.

Note the similarity to the basic MPNN model with self-loops (written as a graph-level equation):

$$\mathbf{H}^{(t)} = \sigma\Big(\big(\mathbf{A}+\mathbf{I}\big)\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big).$$

Graph Convolutional Networks

Based on graph convolutions, the basic graph convolutional network (GCN) model is defined as follows:

$$\mathbf{H}^{(t)} = \sigma\Big(\hat{\mathbf{A}}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big),$$

where $\hat{\mathbf{A}} = \big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}\big(\mathbf{I}+\mathbf{A}\big)\big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}$ is a normalised version of adjacency matrix with self-loops and $\mathbf{W}^{(t)}$ is a learnable parameter matrix.

Note the similarity to the basic MPNN model with self-loops (written as a graph-level equation):

$$\mathbf{H}^{(t)} = \sigma\Big(\big(\mathbf{A}+\mathbf{I}\big)\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big).$$

The basic GCN model is hence a more elaborate version of the basic MPNN model with self-loops. The basic GCN model implicitly captures the combine and aggregate functions: The adjacency component of $\hat{\mathbf{A}}$ enables messaging between neighbours, and the added identity component makes nodes send messages to themselves.

Graph Convolutional Networks

Based on graph convolutions, the basic graph convolutional network (GCN) model is defined as follows:

$$\mathbf{H}^{(t)} = \sigma\Big(\hat{\mathbf{A}}\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big),$$

where $\hat{\mathbf{A}} = \big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}\big(\mathbf{I}+\mathbf{A}\big)\big(\mathbf{D}+\mathbf{I}\big)^{-\frac{1}{2}}$ is a normalised version of adjacency matrix with self-loops and $\mathbf{W}^{(t)}$ is a learnable parameter matrix.

Note the similarity to the basic MPNN model with self-loops (written as a graph-level equation):

$$\mathbf{H}^{(t)} = \sigma\Big(\big(\mathbf{A}+\mathbf{I}\big)\ \mathbf{H}^{(t-1)}\mathbf{W}^{(t)}\Big).$$

The basic GCN model is hence a more elaborate version of the basic MPNN model with self-loops. The basic GCN model implicitly captures the combine and aggregate functions: The adjacency component of $\hat{\mathbf{A}}$ enables messaging between neighbours, and the added identity component makes nodes send messages to themselves.

In this basic GCN, the node base embedding is treated identically to messages from other nodes, and the combine and aggregate functions are intertwined. Recall that this can hinder the expressiveness of the model, and there are variants of GCNs without self-loops.

Gated Graph Neural Networks

Node Embeddings as a Sequence

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Message passing can be seen to fit to another natural perspective:

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Message passing can be seen to fit to another natural perspective:

- In a graph, every node has a state, and this state is updated after every message passing iteration to reflect the new information it receives.

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Message passing can be seen to fit to another natural perspective:

- In a graph, every node has a state, and this state is updated after every message passing iteration to reflect the new information it receives.
- This update is based on the current state of the node, as well as its previous states.

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Message passing can be seen to fit to another natural perspective:

- In a graph, every node has a state, and this state is updated after every message passing iteration to reflect the new information it receives.
- This update is based on the current state of the node, as well as its previous states.
- This process repeats until the end of message passing.

Node Embeddings as a Sequence

GCNs are motivated from convolutions and primarily by the success of CNNs. From this perspective, we viewed graph convolutions as a generalisation of, e.g., image convolution.

Message passing can be seen to fit to another natural perspective:

- In a graph, every node has a state, and this state is updated after every message passing iteration to reflect the new information it receives.
- This update is based on the current state of the node, as well as its previous states.
- This process repeats until the end of message passing.

This mode of operation suggests yet another abstraction: View node embedding updates during message passing as a sequential process!

Sequence Modelling: Refresher

Sequence Modelling: Refresher

Let us take a step back and look at sequence modelling.

Sequence Modelling: Refresher

Let us take a step back and look at sequence modelling.

- Consider, e.g., spam detection, where the task is to identify whether an email is spam or not. In this context, the input sentences are processed word by word, and the state of the computation is updated based on the most recently viewed word, as well as a state, which stores information about all earlier words, to yield a final representation for the overall sentence.

Sequence Modelling: Refresher

Let us take a step back and look at sequence modelling.

- Consider, e.g., spam detection, where the task is to identify whether an email is spam or not. In this context, the input sentences are processed word by word, and the state of the computation is updated based on the most recently viewed word, as well as a state, which stores information about all earlier words, to yield a final representation for the overall sentence.
- This is typically done with recurrent neural models, and a popular model is the Gated Recurrent Unit (GRU) defined as:

$$
\begin{aligned}
\mathbf{R}^t &= \sigma(\mathbf{X}^t \mathbf{W}_{xr} + \mathbf{H}^{(t-1)} \mathbf{W}_{hr} + \mathbf{b}_r), \\
\mathbf{Z}^t &= \sigma(\mathbf{X}^t \mathbf{W}_{xz} + \mathbf{H}^{(t-1)} \mathbf{W}_{hz} + \mathbf{b}_z), \\
\tilde{\mathbf{H}}^t &= \tanh(\mathbf{X}^t \mathbf{W}_{xh} + \left(\mathbf{R}^t \odot \mathbf{H}^{(t-1)}\right) \mathbf{W}_{hh} + \mathbf{b}_h), \\
\mathbf{H}^t &= \mathbf{Z}^t \odot \mathbf{H}^{t-1} + (1 - \mathbf{Z}^t) \odot \tilde{\mathbf{H}}^t .
\end{aligned}
$$

Sequence Modelling: Refresher

Let us take a step back and look at sequence modelling.

- Consider, e.g., spam detection, where the task is to identify whether an email is spam or not. In this context, the input sentences are processed word by word, and the state of the computation is updated based on the most recently viewed word, as well as a state, which stores information about all earlier words, to yield a final representation for the overall sentence.
- This is typically done with recurrent neural models, and a popular model is the Gated Recurrent Unit (GRU) defined as:

$$
\begin{aligned}
\mathbf{R}^t &= \sigma(\mathbf{X}^t\mathbf{W}_{xr} + \mathbf{H}^{(t-1)}\mathbf{W}_{hr} + \mathbf{b}_r),\\
\mathbf{Z}^t &= \sigma(\mathbf{X}^t\mathbf{W}_{xz} + \mathbf{H}^{(t-1)}\mathbf{W}_{hz} + \mathbf{b}_z),\\
\tilde{\mathbf{H}}^t &= \tanh(\mathbf{X}^t\mathbf{W}_{xh} + \left(\mathbf{R}^t \odot \mathbf{H}^{(t-1)}\right)\mathbf{W}_{hh} + \mathbf{b}_h),\\
\mathbf{H}^t &= \mathbf{Z}^t \odot \mathbf{H}^{t-1} + (1 - \mathbf{Z}^t) \odot \tilde{\mathbf{H}}^t.
\end{aligned}
$$

- Briefly, the idea it to maintain a state in memory, and then based on the new state and input, decide to retain or update your state.

Sequential Node Embeddings

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

We can view nodes as having states which are being updated based on an aggregation of all incoming messages from their neighbours, for a given number of message passing iterations.

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

We can view nodes as having states which are being updated based on an aggregation of all incoming messages from their neighbours, for a given number of message passing iterations.

Therefore, the state abstraction for nodes in a graph gives rise to three separate computations:

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

We can view nodes as having states which are being updated based on an aggregation of all incoming messages from their neighbours, for a given number of message passing iterations.

Therefore, the state abstraction for nodes in a graph gives rise to three separate computations:

1. **Message computation**: Based on a node's current state

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

We can view nodes as having states which are being updated based on an aggregation of all incoming messages from their neighbours, for a given number of message passing iterations.

Therefore, the state abstraction for nodes in a graph gives rise to three separate computations:

1. **Message computation**: Based on a node's current state
2. **Message aggregation**: Node level aggregation

Sequential Node Embeddings

How to apply this sequential state abstraction to node embeddings and hence graphs?

We can view nodes as having states which are being updated based on an aggregation of all incoming messages from their neighbours, for a given number of message passing iterations.

Therefore, the state abstraction for nodes in a graph gives rise to three separate computations:

1. **Message computation**: Based on a node's current state
2. **Message aggregation**: Node level aggregation
3. **State update**: A recurrent unit, such as GRU, that accepts the current state, the aggregation of messages, and yields an update.

Gated Graph Neural Networks

Gated Graph Neural Networks

This yields a model known as the gated graph neural networks (Li et al., 2016), where the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

$$\mathbf{h}_u^{(t)} = GRU\Big(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{W}^{(t)} \mathbf{h}_v^{(t-1)}\Big),$$

That is, in the original base model message computation happens through a multiplication by a shared weight matrix, and aggregation is set to sum.

Gated Graph Neural Networks

This yields a model known as the gated graph neural networks (Li et al., 2016), where the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

$$\mathbf{h}_u^{(t)} = GRU\Big(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{W}^{(t)} \mathbf{h}_v^{(t-1)}\Big),$$

That is, in the original base model message computation happens through a multiplication by a shared weight matrix, and aggregation is set to sum.

Several variants to the base GGNN model have been developed by simply modifying any of the three aforementioned components.

Gated Graph Neural Networks

This yields a model known as the gated graph neural networks (Li et al., 2016), where the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

$$\mathbf{h}_u^{(t)} = GRU\Big(\mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{W}^{(t)} \mathbf{h}_v^{(t-1)}\Big),$$

That is, in the original base model message computation happens through a multiplication by a shared weight matrix, and aggregation is set to sum.

Several variants to the base GGNN model have been developed by simply modifying any of the three aforementioned components.

Multi-layer perceptrons have been used for message computation; other gated units, such as LSTM, Layer-Norm LSTM, have also been applied instead of GRU; finally, we have the usual choices for different aggregation functions.

Graph Isomorphism Networks

A Closer Look at Aggregation

A Closer Look at Aggregation

In earlier models, we discussed different choices of aggregation, but did not investigate the impact these had on the discrimination ability of GNNs.

A Closer Look at Aggregation

In earlier models, we discussed different choices of aggregation, but did not investigate the impact these had on the discrimination ability of GNNs.

More concretely, consider a simple node classification task, with basic node types red, green and yellow, where the features are simply the RGB values. We now consider a red node, and want to analyse how different functions aggregate neighbour messages:

A Closer Look at Aggregation

In earlier models, we discussed different choices of aggregation, but did not investigate the impact these had on the discrimination ability of GNNs.

More concretely, consider a simple node classification task, with basic node types red, green and yellow, where the features are simply the RGB values. We now consider a red node, and want to analyse how different functions aggregate neighbour messages:

- **Sum**: Can discern between neighbourhoods based on their sizes, e.g., between 2-red neighbourhood and 3-red neighbourhood. However, summation can lead to false equality: In this example, sum would not be able to distinguish between a 2-yellow and a red-green neighbourhood.

A Closer Look at Aggregation

In earlier models, we discussed different choices of aggregation, but did not investigate the impact these had on the discrimination ability of GNNs.

More concretely, consider a simple node classification task, with basic node types red, green and yellow, where the features are simply the RGB values. We now consider a red node, and want to analyse how different functions aggregate neighbour messages:

- **Sum**: Can discern between neighbourhoods based on their sizes, e.g., between 2-red neighbourhood and 3-red neighbourhood. However, summation can lead to false equality: In this example, sum would not be able to distinguish between a 2-yellow and a red-green neighbourhood.
- **Mean**: Useful for bounding the range of aggregate messages, but cannot distinguish between neighbour sets such as 2-red and 3-red, as the mean operation eliminates cardinality.

A Closer Look at Aggregation

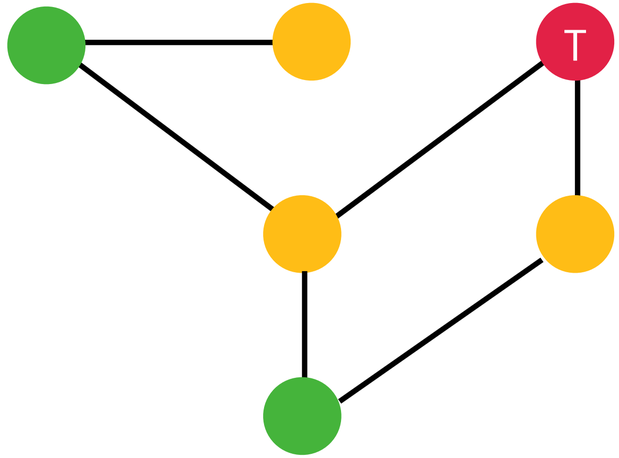
In earlier models, we discussed different choices of aggregation, but did not investigate the impact these had on the discrimination ability of GNNs.

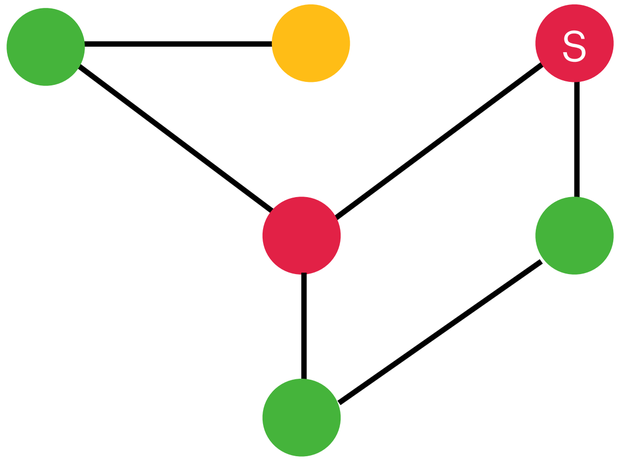
More concretely, consider a simple node classification task, with basic node types red, green and yellow, where the features are simply the RGB values. We now consider a red node, and want to analyse how different functions aggregate neighbour messages:

- **Sum**: Can discern between neighbourhoods based on their sizes, e.g., between 2-red neighbourhood and 3-red neighbourhood. However, summation can lead to false equality: In this example, sum would not be able to distinguish between a 2-yellow and a red-green neighbourhood.
- **Mean**: Useful for bounding the range of aggregate messages, but cannot distinguish between neighbour sets such as 2-red and 3-red, as the mean operation eliminates cardinality.
- **Max**: Allows to highlight a relevant element in the neighbourhood, but has very limited discriminative ability. If we consider an ordering red $<$ yellow $<$ green, then green will be returned for any neighborhood involving at least 1 green node.

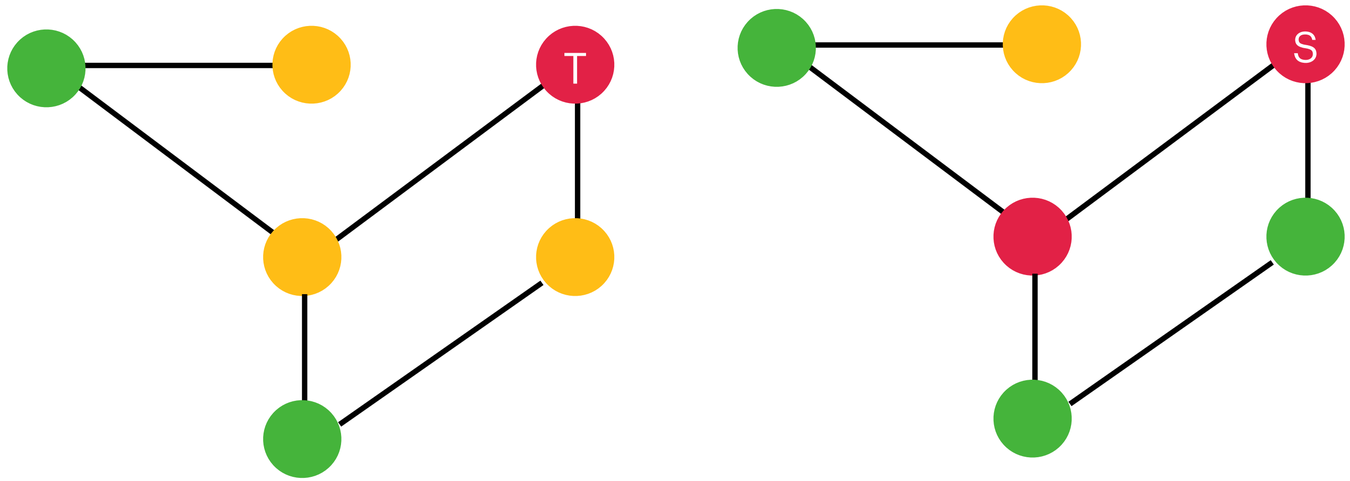
Aggregation and Expressiveness

Aggregation and Expressiveness



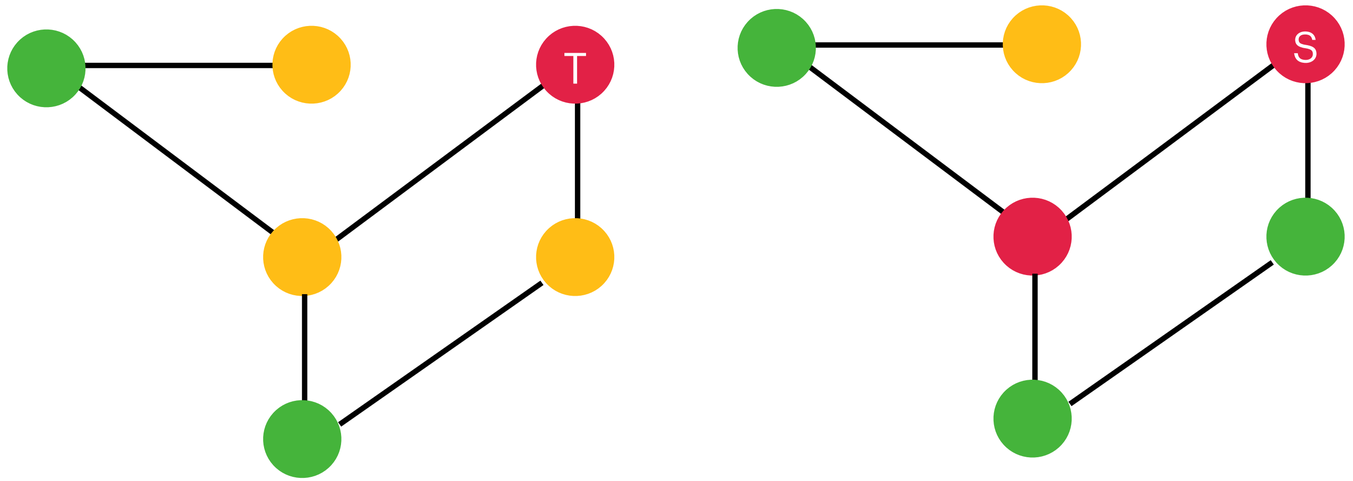


Aggregation and Expressiveness



In order to be more generally applicable, an aggregation function must be able to distinguish between distinct neighbourhoods, and return different results given different neighbourhood multisets.

Aggregation and Expressiveness



In order to be more generally applicable, an aggregation function must be able to distinguish between distinct neighbourhoods, and return different results given different neighbourhood multisets.

More formally, the aggregation function must be injective relative to the neighbourhood. In fact, it is shown that MPNNs are at their maximal expressiveness if the aggregation function being used is injective (Xu et al.,2019).

Aggregation and Expressiveness

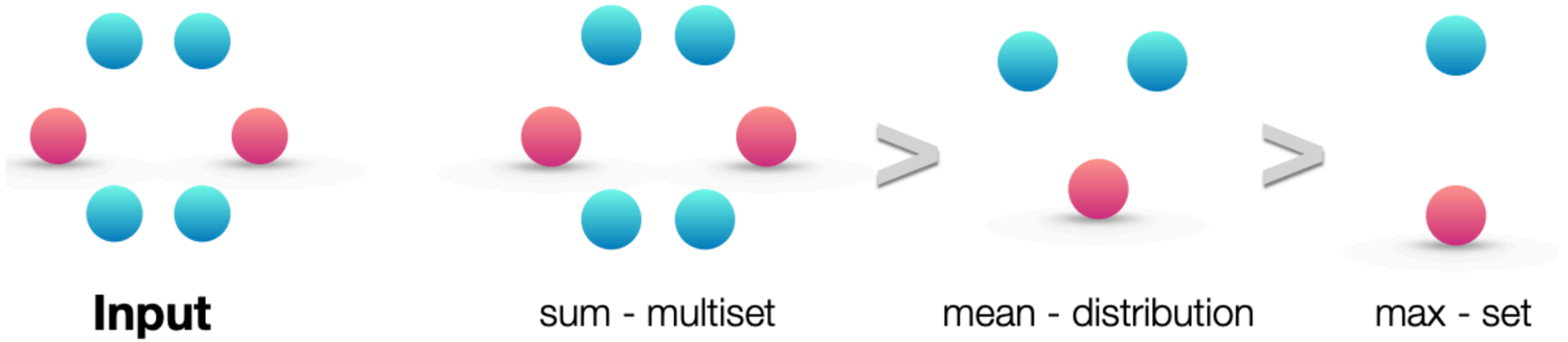
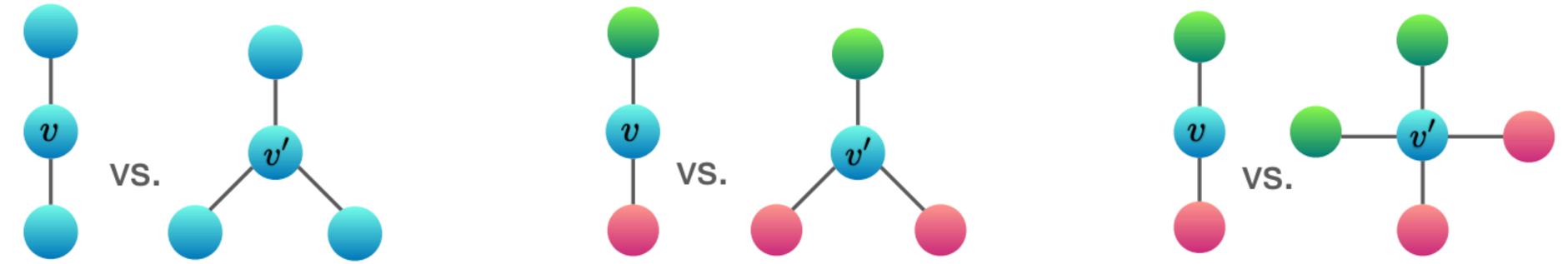


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, *i.e.*, the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set).

— In terms of injectiveness sum > mean > max. (Xu et al., 2019)

Aggregation and Expressiveness



(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.** Between the two graphs, nodes v and v' get the same embedding even though their corresponding graph structures differ. Figure 2 gives reasoning about how different aggregators "compress" different multisets and thus fail to distinguish them.

(Xu et al., 2019)

Graph Isomorphism Network: Injective Aggregation

Graph Isomorphism Network: Injective Aggregation

Graph isomorphism networks (GINs) propose a novel aggregation scheme that is injective, thereby allowing maximal expressiveness for aggregation.

Graph Isomorphism Network: Injective Aggregation

Graph isomorphism networks (GINs) propose a novel aggregation scheme that is injective, thereby allowing maximal expressiveness for aggregation.

[**Lemma 5 & Corollary 6,** (Xu et al., 2019)] For a countable set $\mathcal{X}$, there exists a function $f : X \to \mathbb{R}^n$ such that for any choice of ϵ, the function

$$h(c, X) = (1 + \epsilon) \cdot f(c) \sum_{x \in X} f(x)$$

is unique for each pair (c, X), where $c \in X$ and $X \subset \mathcal{X}$ is a multiset of bounded size. Moreover, any function g over such pair (c, X) can be decomposed as:

$$g(c, X) = \psi\Big((1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)\Big)$$

for some function ψ.

Graph Isomorphism Network: Injective Aggregation

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

GINs apply an MLP on all nodes, to model and learn $f \circ \psi$, yielding an injective aggregation function, which is possible thanks to universal approximation theorem given for MLPs (Hornik et al., 1989).

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

GINs apply an MLP on all nodes, to model and learn $f \circ \psi$, yielding an injective aggregation function, which is possible thanks to universal approximation theorem given for MLPs (Hornik et al., 1989).

Overall, the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

GINs apply an MLP on all nodes, to model and learn $f \circ \psi$, yielding an injective aggregation function, which is possible thanks to universal approximation theorem given for MLPs (Hornik et al., 1989).

Overall, the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

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

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

GINs apply an MLP on all nodes, to model and learn $f \circ \psi$, yielding an injective aggregation function, which is possible thanks to universal approximation theorem given for MLPs (Hornik et al., 1989).

Overall, the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

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

Intuitively, in GINs, this MLP in fact computes the update for node representation, based on the summation of neighbourhood node representation, plus the current node representation, scaled up by a factor $1+\epsilon$.

Graph Isomorphism Network: Injective Aggregation

The idea is to learn $f \circ \psi$ to obtain an injective aggregation function.

GINs apply an MLP on all nodes, to model and learn $f \circ \psi$, yielding an injective aggregation function, which is possible thanks to universal approximation theorem given for MLPs (Hornik et al., 1989).

Overall, the representation $\mathbf{h}_u$ for each node $u \in V$ is iteratively updated as:

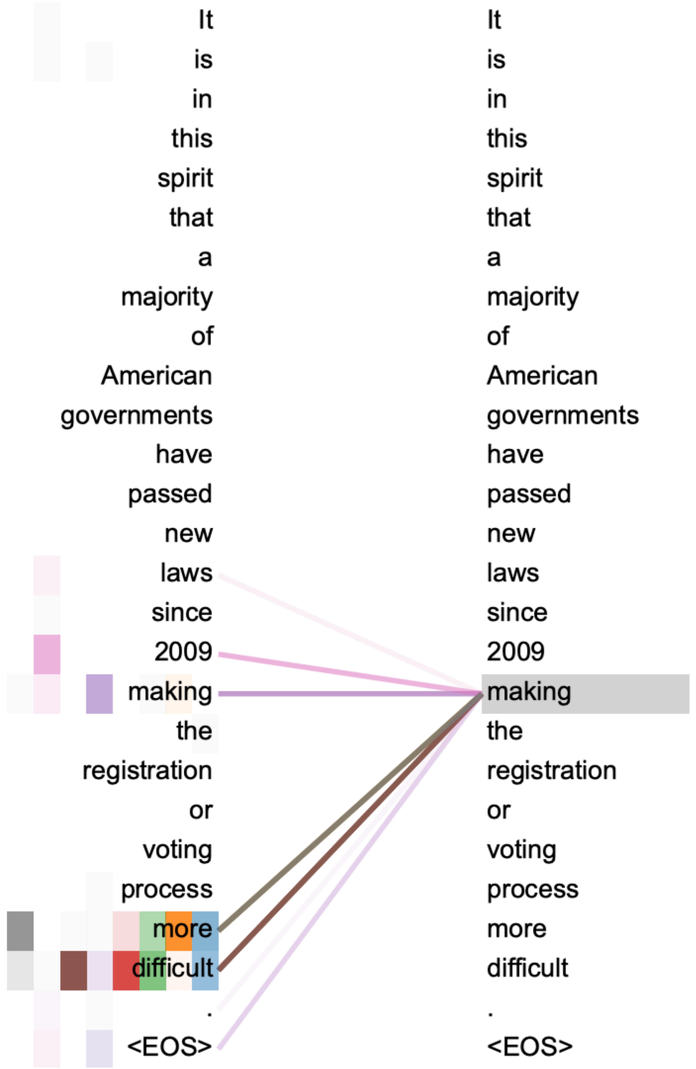
$$\mathbf{h}_u^{(t)} = MLP\Big((1+\epsilon) \cdot \mathbf{h}_u^{(t-1)}, \sum_{v \in N(u)} \mathbf{h}_v^{(t-1)}\Big)$$

Intuitively, in GINs, this MLP in fact computes the update for node representation, based on the summation of neighbourhood node representation, plus the current node representation, scaled up by a factor $1+\epsilon$.

The scaling up of the current node representation distinguishes this node from the neighbour nodes.

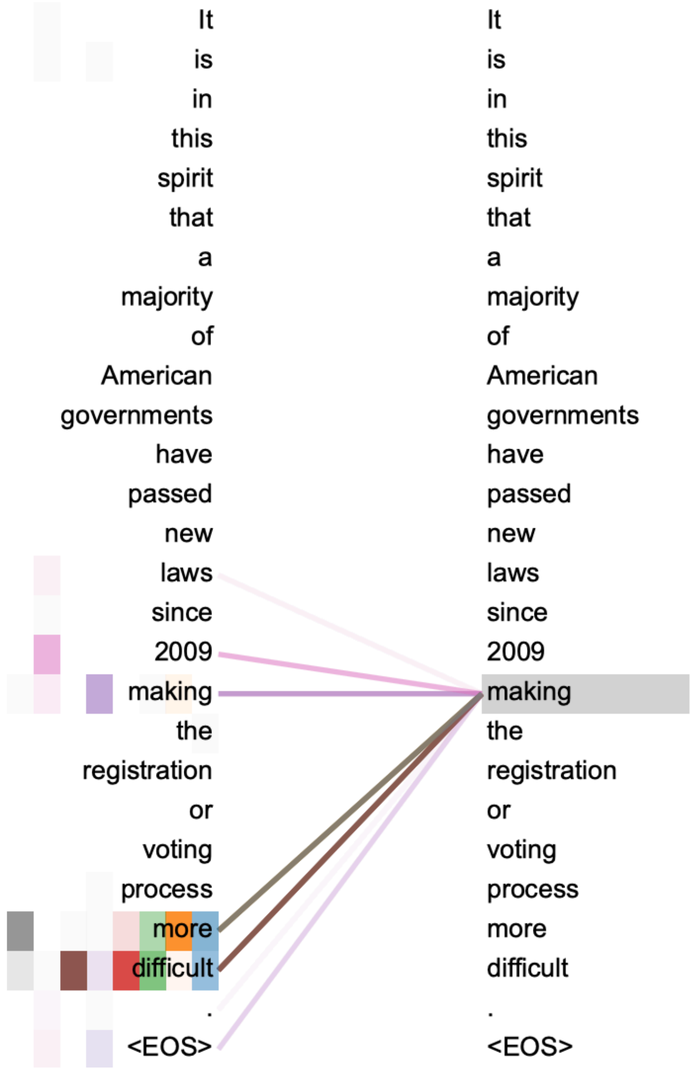
Graph Attention Networks

Attention: Refresher



Attention: Refresher

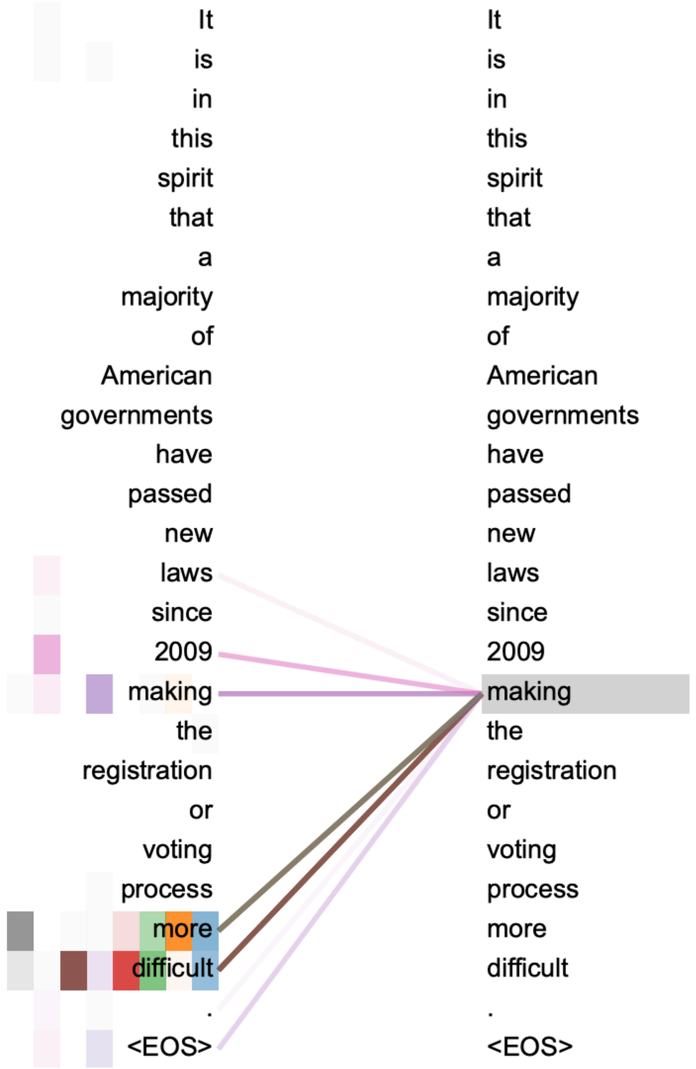
Attention is a mechanism through which neural networks dynamically allocate different weights to distinct inputs, based on their relevance to the learned task.



Attention: Refresher

Attention is a mechanism through which neural networks dynamically allocate different weights to distinct inputs, based on their relevance to the learned task.

Attention has proven key to achieving significant improvements in sequential tasks such as machine translation and language understanding; see, e.g., (Bahdanau et al., 2015).

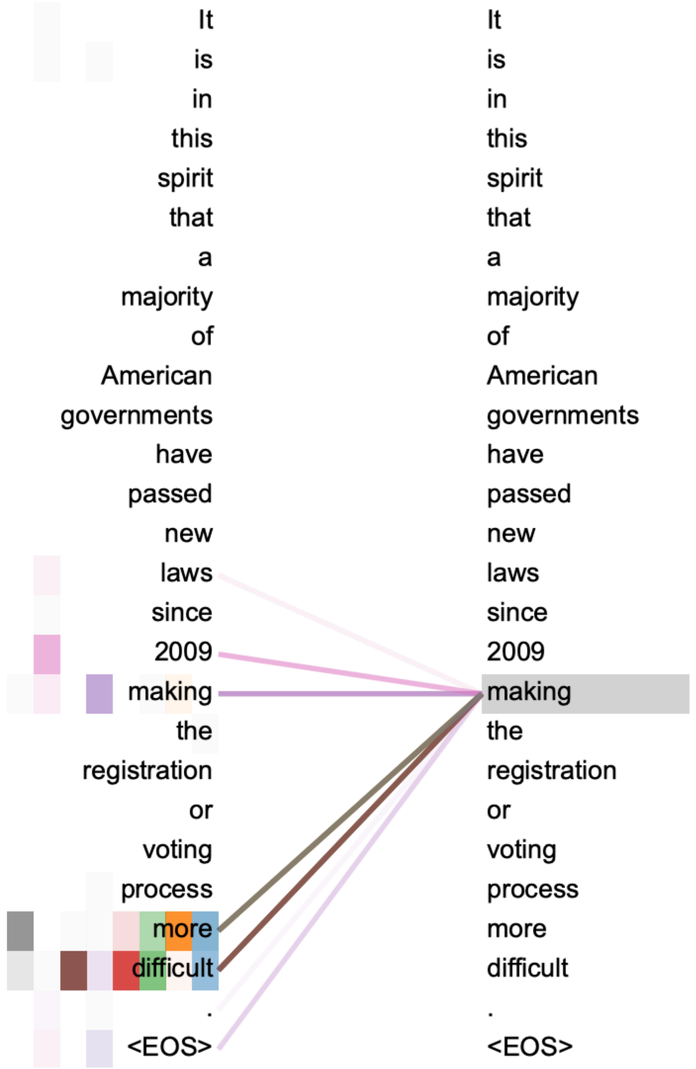


Attention: Refresher

Attention is a mechanism through which neural networks dynamically allocate different weights to distinct inputs, based on their relevance to the learned task.

Attention has proven key to achieving significant improvements in sequential tasks such as machine translation and language understanding; see, e.g., (Bahdanau et al., 2015).

An example of the attention mechanism following long-distance dependencies in the encoder self-attention (from Figure 3 of (Vasvani et al., 2017)) is shown on the right hand side.



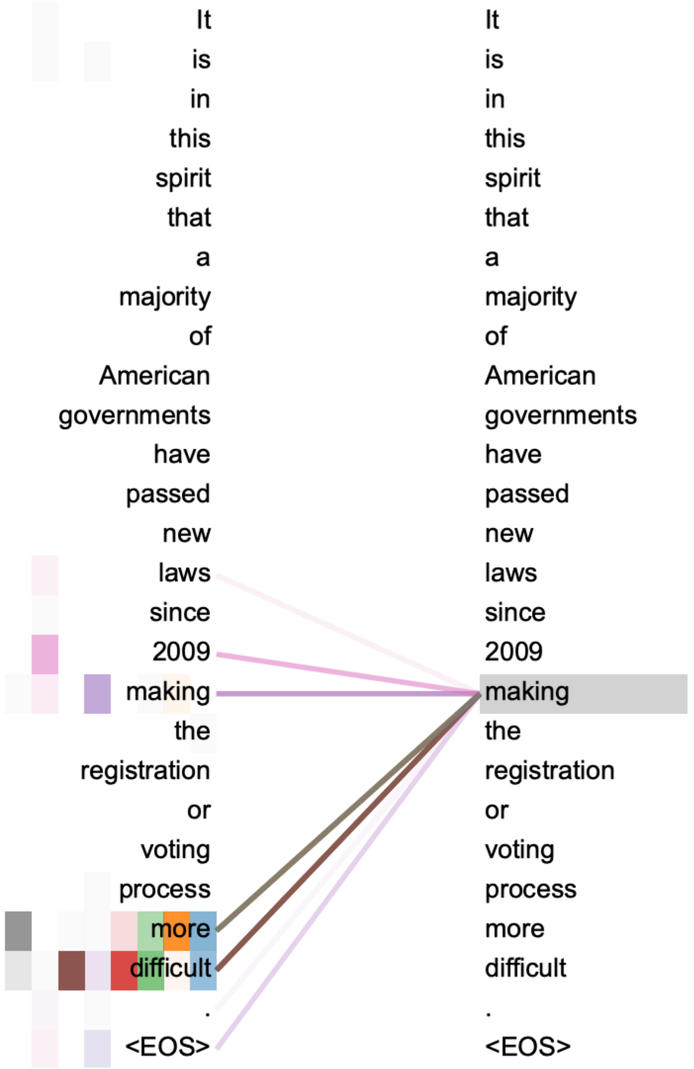
Attention: Refresher

Attention is a mechanism through which neural networks dynamically allocate different weights to distinct inputs, based on their relevance to the learned task.

Attention has proven key to achieving significant improvements in sequential tasks such as machine translation and language understanding; see, e.g., (Bahdanau et al., 2015).

An example of the attention mechanism following long-distance dependencies in the encoder self-attention (from Figure 3 of (Vasvani et al., 2017)) is shown on the right hand side.

"Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colours represent different heads."



Attention over Graphs

Attention over Graphs

Question: Can attention be lifted to the graph setting?

Attention over Graphs

Question: Can attention be lifted to the graph setting?

In a sequence, attention allows a token-level computation to identify other relevant tokens, rather than uniformly considering all possible tokens.

Attention over Graphs

Question: Can attention be lifted to the graph setting?

In a sequence, attention allows a token-level computation to identify other relevant tokens, rather than uniformly considering all possible tokens.

An analogous idea could be for graphs: A node can use attention to identify neighbours that affect its embedding more relative to other neighbours.

Attention over Graphs

Question: Can attention be lifted to the graph setting?

In a sequence, attention allows a token-level computation to identify other relevant tokens, rather than uniformly considering all possible tokens.

An analogous idea could be for graphs: A node can use attention to identify neighbours that affect its embedding more relative to other neighbours.

One can easily imagine tasks where neighbour attention can be highly important, e.g., node classification based on having a neighbour of a given type, when all nodes have very large degrees.

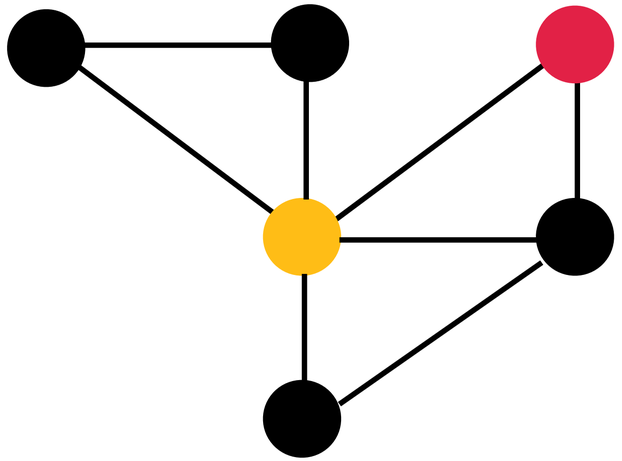
Attention over Graphs

Question: Can attention be lifted to the graph setting?

In a sequence, attention allows a token-level computation to identify other relevant tokens, rather than uniformly considering all possible tokens.

An analogous idea could be for graphs: A node can use attention to identify neighbours that affect its embedding more relative to other neighbours.

One can easily imagine tasks where neighbour attention can be highly important, e.g., node classification based on having a neighbour of a given type, when all nodes have very large degrees.



Graph Attention Networks

Graph Attention Networks

Attention can therefore produce a richer weighing of a node's neighbours, which results in potentially more descriptive and task-specific aggregation schemes.

Graph Attention Networks

Attention can therefore produce a richer weighing of a node's neighbours, which results in potentially more descriptive and task-specific aggregation schemes.

The basic idea is to assign an attention weight to each neighbour, which corresponds to having weighted aggregation functions e.g., weighted sum, weighted average, etc.

Graph Attention Networks

Attention can therefore produce a richer weighing of a node's neighbours, which results in potentially more descriptive and task-specific aggregation schemes.

The basic idea is to assign an attention weight to each neighbour, which corresponds to having weighted aggregation functions e.g., weighted sum, weighted average, etc.

This intuition underlies the development of graph attention networks (GAT) (Velickovic et al., 2018), which uses weighted sum aggregation, and a pairwise node attention mechanism during message passing;

$$\mathbf{h}_u^{(t)} = \sigma\Big(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \alpha_{(u,v)}\, \mathbf{h}_v^{(t-1)}\Big),$$

where $\alpha_{(u,v)}$ denotes the attention on a node in $v \in N(u) \cup \{u\}$ when we are aggregating information at node u.

Graph Attention Networks

Attention can therefore produce a richer weighing of a node's neighbours, which results in potentially more descriptive and task-specific aggregation schemes.

The basic idea is to assign an attention weight to each neighbour, which corresponds to having weighted aggregation functions e.g., weighted sum, weighted average, etc.

This intuition underlies the development of graph attention networks (GAT) (Velickovic et al., 2018), which uses weighted sum aggregation, and a pairwise node attention mechanism during message passing;

$$\mathbf{h}_u^{(t)} = \sigma\Big(\mathbf{W}^{(t)} \sum_{v \in N(u) \cup \{u\}} \alpha_{(u,v)}\, \mathbf{h}_v^{(t-1)}\Big),$$

where $\alpha_{(u,v)}$ denotes the attention on a node in $v \in N(u) \cup \{u\}$ when we are aggregating information at node u.

This can be seen as an extension of the base MPNN model with self-loops. Having self-loops is fine here, as the discrimination of nodes is happening through the attention mechanism. We can of course generalise this:

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

Attention for Graphs: Properties and Other Uses

Attention for Graphs: Properties and Other Uses

GATs allow nodes to assign relative importance to their neighbours throughout message passing, and compute updates less uniformly across the graph. Attention also slightly improves information flow in GNNs.

Attention for Graphs: Properties and Other Uses

GATs allow nodes to assign relative importance to their neighbours throughout message passing, and compute updates less uniformly across the graph. Attention also slightly improves information flow in GNNs.

Furthermore, attention makes the updates more injective: This ensures that GATs are closer to the inherent 1-WL limit (which we will discuss later in the course).

Attention for Graphs: Properties and Other Uses

GATs allow nodes to assign relative importance to their neighbours throughout message passing, and compute updates less uniformly across the graph. Attention also slightly improves information flow in GNNs.

Furthermore, attention makes the updates more injective: This ensures that GATs are closer to the inherent 1-WL limit (which we will discuss later in the course).

Though attention naturally fits in during aggregation, one can also imagine other more conventional uses of the mechanism to improve GNNs:

Attention for Graphs: Properties and Other Uses

GATs allow nodes to assign relative importance to their neighbours throughout message passing, and compute updates less uniformly across the graph. Attention also slightly improves information flow in GNNs.

Furthermore, attention makes the updates more injective: This ensures that GATs are closer to the inherent 1-WL limit (which we will discuss later in the course).

Though attention naturally fits in during aggregation, one can also imagine other more conventional uses of the mechanism to improve GNNs:

- Attention for computing the global readout following, or during, message passing.

Attention for Graphs: Properties and Other Uses

GATs allow nodes to assign relative importance to their neighbours throughout message passing, and compute updates less uniformly across the graph. Attention also slightly improves information flow in GNNs.

Furthermore, attention makes the updates more injective: This ensures that GATs are closer to the inherent 1-WL limit (which we will discuss later in the course).

Though attention naturally fits in during aggregation, one can also imagine other more conventional uses of the mechanism to improve GNNs:

- Attention for computing the global readout following, or during, message passing.
- Attention across different message passing iterations, similarly to sequence models. For instance, a GGNN model can compute its update by attending to its previous states at earlier iterations, much like standard sequence models.

A Relation to Transformer Model

A Relation to Transformer Model

GNN models can easily be extended to multi-head attention (Velickovic et al., 2018).

A Relation to Transformer Model

GNN models can easily be extended to multi-head attention (Velickovic et al., 2018).

Question: Is there a connection between Transformer and GNNs with multi-head attention?

A Relation to Transformer Model

GNN models can easily be extended to multi-head attention (Velickovic et al., 2018).

Question: Is there a connection between Transformer and GNNs with multi-head attention?

Briefly, the transformer architecture (Vaswani et al., 2017) defines neural network layers entirely based on the attention operation, and generates, at each layer, a representation for every position in the input data by using multiple attention heads to compute attention weights between all pairs of positions in the input, which are then aggregated with weighted sums based on these attention weights.

A Relation to Transformer Model

GNN models can easily be extended to multi-head attention (Velickovic et al., 2018).

Question: Is there a connection between Transformer and GNNs with multi-head attention?

Briefly, the transformer architecture (Vaswani et al., 2017) defines neural network layers entirely based on the attention operation, and generates, at each layer, a representation for every position in the input data by using multiple attention heads to compute attention weights between all pairs of positions in the input, which are then aggregated with weighted sums based on these attention weights.

This is exactly what happens in GNNs with multi-head attention, except that the input graphs are not necessarily fully connected!

A Relation to Transformer Model

GNN models can easily be extended to multi-head attention (Velickovic et al., 2018).

Question: Is there a connection between Transformer and GNNs with multi-head attention?

Briefly, the transformer architecture (Vaswani et al., 2017) defines neural network layers entirely based on the attention operation, and generates, at each layer, a representation for every position in the input data by using multiple attention heads to compute attention weights between all pairs of positions in the input, which are then aggregated with weighted sums based on these attention weights.

This is exactly what happens in GNNs with multi-head attention, except that the input graphs are not necessarily fully connected!

We can therefore view the basic Transformer model as a GNN model with multi-head attention, if we further assume that the GNN receives a fully connected graph as input.

Discussions and Limitations

Over-smoothing

Over-smoothing

Over-smoothing (Li et al., 2018) is a phenomenon where the representations of the nodes in the graph become indistinguishable after several message passing iterations.

Over-smoothing

Over-smoothing (Li et al., 2018) is a phenomenon where the representations of the nodes in the graph become indistinguishable after several message passing iterations.

Over-smoothing hence makes it very hard for the model to make meaningful predictions — especially for deep GNN models, where the goal is to pass information across many layers so as to capture long-range dependencies.

Over-smoothing

Over-smoothing (Li et al., 2018) is a phenomenon where the representations of the nodes in the graph become indistinguishable after several message passing iterations.

Over-smoothing hence makes it very hard for the model to make meaningful predictions — especially for deep GNN models, where the goal is to pass information across many layers so as to capture long-range dependencies.

Intuitively, this typically happens when messages aggregated from the neighbours are too prominent, rendering the effect of the embeddings from the previous layers less and less important.

Over-smoothing

Over-smoothing (Li et al., 2018) is a phenomenon where the representations of the nodes in the graph become indistinguishable after several message passing iterations.

Over-smoothing hence makes it very hard for the model to make meaningful predictions — especially for deep GNN models, where the goal is to pass information across many layers so as to capture long-range dependencies.

Intuitively, this typically happens when messages aggregated from the neighbours are too prominent, rendering the effect of the embeddings from the previous layers less and less important.

Significant performance degradation has been observed when stacking many layers on GNNs (Kipf & Welling, 2017); especially for GCNs, quoting from (Li et al., 2018):

Over-smoothing

Over-smoothing (Li et al., 2018) is a phenomenon where the representations of the nodes in the graph become indistinguishable after several message passing iterations.

Over-smoothing hence makes it very hard for the model to make meaningful predictions — especially for deep GNN models, where the goal is to pass information across many layers so as to capture long-range dependencies.

Intuitively, this typically happens when messages aggregated from the neighbours are too prominent, rendering the effect of the embeddings from the previous layers less and less important.

Significant performance degradation has been observed when stacking many layers on GNNs (Kipf & Welling, 2017); especially for GCNs, quoting from (Li et al., 2018):

"*If a GCN is deep with many convolutional layers, the output features may be over-smoothed and vertices from different clusters may become indistinguishable.*"

Over-smoothing

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

For example, one theoretical justification is given by (Xu et al., 2018) for GCN-like models with self-loop:

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

For example, one theoretical justification is given by (Xu et al., 2018) for GCN-like models with self-loop:

With a k-layer GCN, the influence of a node u on node v is proportional the probability of reaching node v on a k-step random walk starting from node u.

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

For example, one theoretical justification is given by (Xu et al., 2018) for GCN-like models with self-loop:

With a k-layer GCN, the influence of a node u on node v is proportional the probability of reaching node v on a k-step random walk starting from node u.

Note that over-smoothing is a fundamental limitation for many models!

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

For example, one theoretical justification is given by (Xu et al., 2018) for GCN-like models with self-loop:

With a k-layer GCN, the influence of a node u on node v is proportional the probability of reaching node v on a k-step random walk starting from node u.

Note that over-smoothing is a fundamental limitation for many models!

To partially alleviate over-smoothing, a typical strategy is to define a more general update procedure (Hamilton et al., 2017) that concatenates each node's previous embedding with the output of the combine function, so as to preserve as much information from previous rounds of message passing as possible.

Over-smoothing

There are various theoretical studies trying to pinpoint the exact effect of over-smoothing in different models.

For example, one theoretical justification is given by (Xu et al., 2018) for GCN-like models with self-loop:

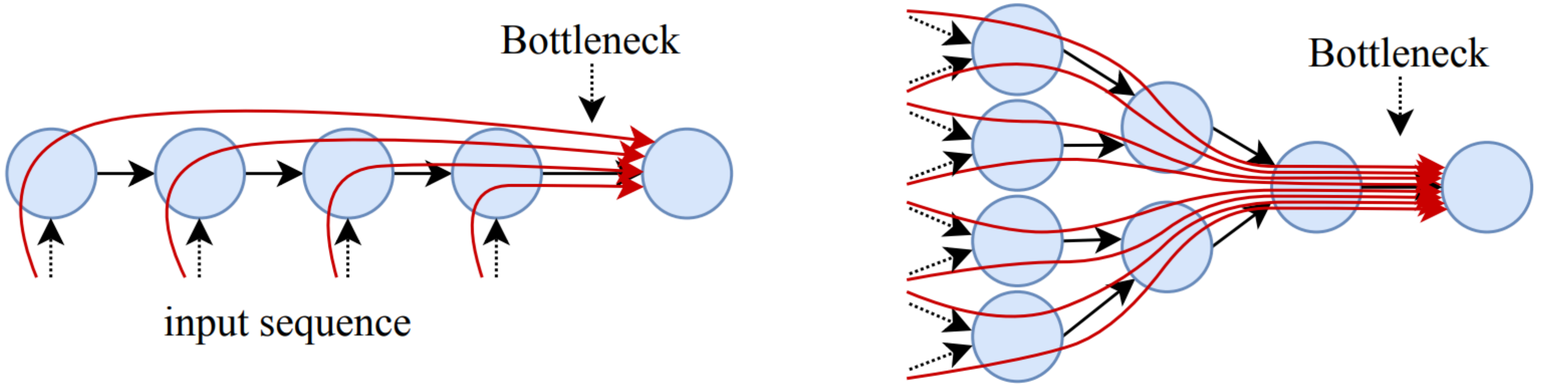
With a k-layer GCN, the influence of a node u on node v is proportional the probability of reaching node v on a k-step random walk starting from node u.

Note that over-smoothing is a fundamental limitation for many models!

To partially alleviate over-smoothing, a typical strategy is to define a more general update procedure (Hamilton et al., 2017) that concatenates each node's previous embedding with the output of the combine function, so as to preserve as much information from previous rounds of message passing as possible.

This is not a solution to the problem, but rather a way to alleviate the problem in practice.

Over-squashing

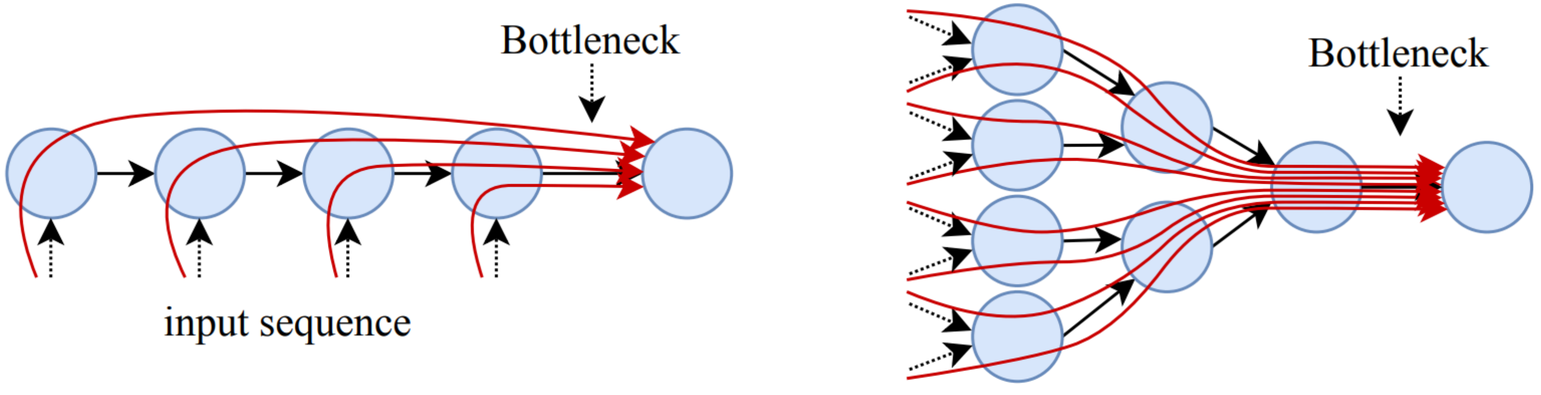


(a) The bottleneck of RNN seq2seq models

(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Over-squashing



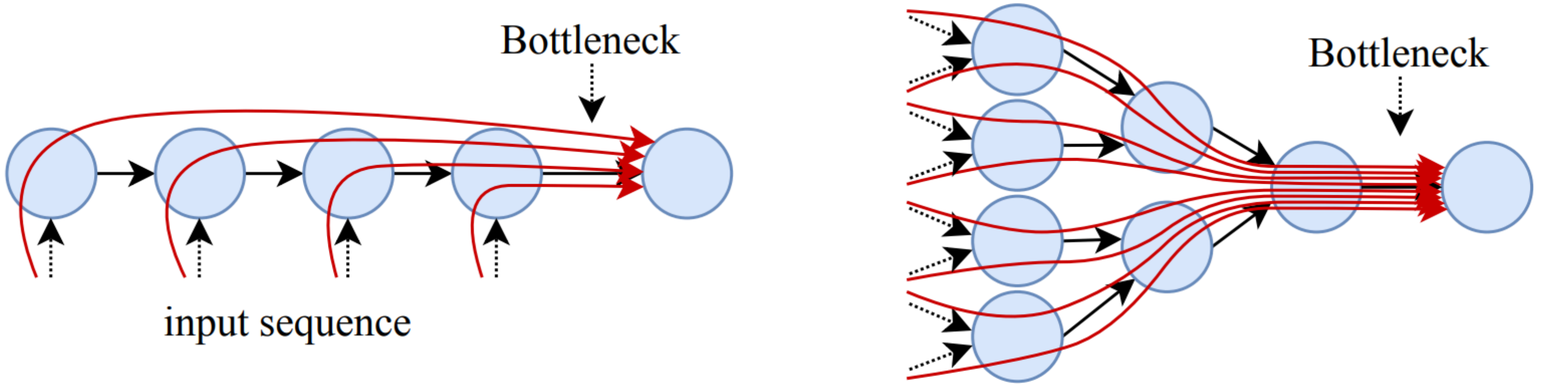
(a) The bottleneck of RNN seq2seq models

(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Over-squashing is a closely related problem; quoting (Alon and Yahav, 2021):

Over-squashing



(a) The bottleneck of RNN seq2seq models

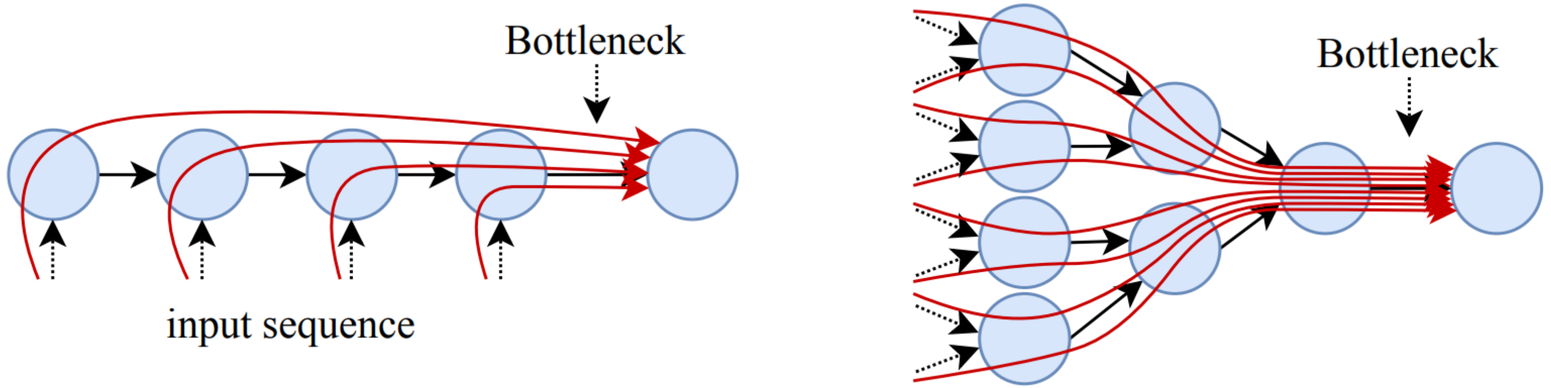
(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Over-squashing is a closely related problem; quoting (Alon and Yahav, 2021):

"As the number of layers increases, the number of nodes in each node's receptive field grows exponentially. This causes over-squashing: information from the exponentially-growing receptive field is compressed into fixed-length node vectors. Consequently, the graph fails to propagate messages flowing from distant nodes; the model learns only short-range signals from the training data; and overall, generalises poorly at test time."

Over-squashing

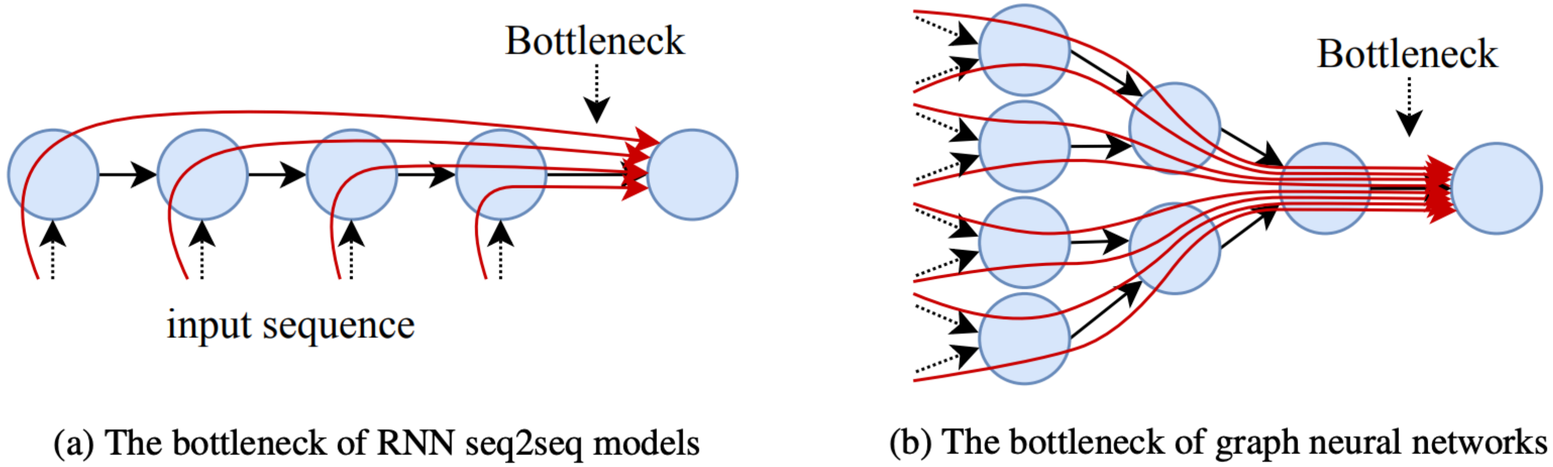


(a) The bottleneck of RNN seq2seq models

(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

Over-squashing



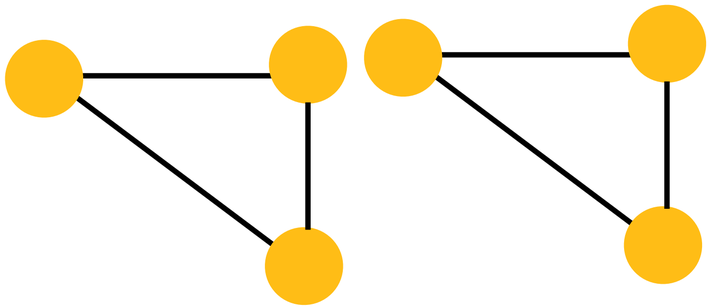
(a) The bottleneck of RNN seq2seq models

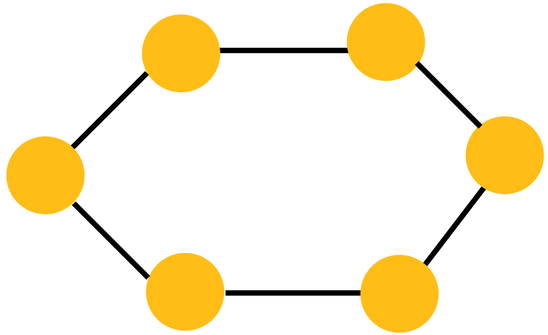
(b) The bottleneck of graph neural networks

Figure 1: The bottleneck that existed in RNN seq2seq models (before attention) is strictly more harmful in GNNs: information from a node's exponentially-growing receptive field is compressed into a fixed-size vector. Black arrows are graph edges; red curved arrows illustrate information flow.

The model then performs poorly when the prediction task depends on long-range interactions, and it is easy to imagine tasks that require long-range dependencies: consider the reachability task on graphs, which requires as many iterations as the diameter of the graph, as otherwise it will suffer from under-reaching (i.e., not receiving information from some nodes).

Expressiveness

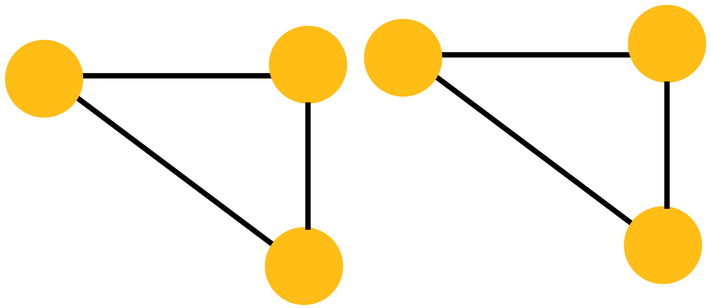


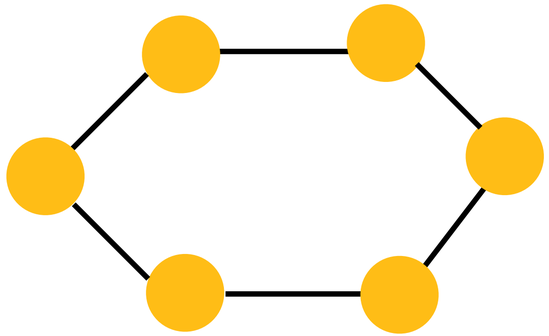


Expressiveness

We have already mentioned that the representation/expressive power of MPNNs are limited:

Expressiveness

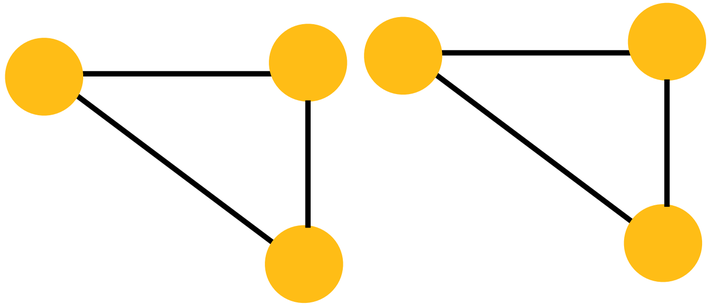


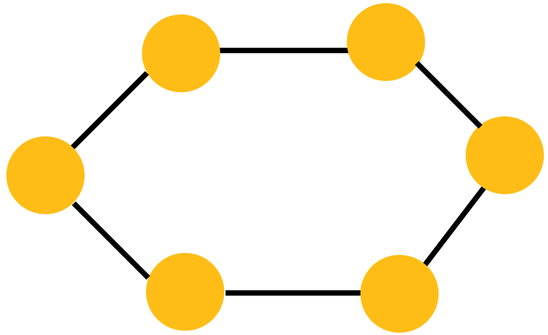


We have already mentioned that the representation/expressive power of MPNNs are limited:

Theoretically, their expressive power is the same as the 1-dimensional Weisfeiler-Lehman graph isomorphism heuristic (1-WL) in terms of distinguishing non-isomorphic (sub-)graphs.

Expressiveness





We have already mentioned that the representation/expressive power of MPNNs are limited:

Theoretically, their expressive power is the same as the 1-dimensional Weisfeiler-Lehman graph isomorphism heuristic (1-WL) in terms of distinguishing non-isomorphic (sub-)graphs.

This implies for example that the embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side.

Expressiveness

We have already mentioned that the representation/expressive power of MPNNs are limited:

Theoretically, their expressive power is the same as the 1-dimensional Weisfeiler-Lehman graph isomorphism heuristic (1-WL) in terms of distinguishing non-isomorphic (sub-)graphs.

This implies for example that the embedding learned for the graph on the left-hand side will be exactly the same as the embedding of the graph on the right-hand side.

This is an important limitation, which will be the topic of the next lecture.

Summary

Summary

- An historical overview of graph neural networks:

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation
  - Graph attention networks: distinguish messages from nodes via attention

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation
  - Graph attention networks: distinguish messages from nodes via attention
- Each of these models fall into the MPNN framework of (Gilmer et al, 2017).

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation
  - Graph attention networks: distinguish messages from nodes via attention
- Each of these models fall into the MPNN framework of (Gilmer et al, 2017).
- Additional reading material: This lecture is partially based on Chapters 5 - 7 of (Hamilton, 2020).

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation
  - Graph attention networks: distinguish messages from nodes via attention
- Each of these models fall into the MPNN framework of (Gilmer et al, 2017).
- Additional reading material: This lecture is partially based on Chapters 5 - 7 of (Hamilton, 2020).
- We have not identified the expressive power of MPNNs: Lecture 5.

Summary

- An historical overview of graph neural networks:
  - Graph convolutional networks: each iteration of message passing is a convolution.
  - Gated graph neural network: graphs as sequences — gated units as the combine function.
  - Graph isomorphism network: aggregation
  - Graph attention networks: distinguish messages from nodes via attention
- Each of these models fall into the MPNN framework of (Gilmer et al, 2017).
- Additional reading material: This lecture is partially based on Chapters 5 - 7 of (Hamilton, 2020).
- We have not identified the expressive power of MPNNs: Lecture 5.
- There are a plethora of other GNN models; some cannot be classified as MPNNs: Lecture 6.

References

- T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017.
- M. D. Zeiler, R. Fergus. Visualizing and Understanding Convolutional Networks, *ECCV*, 2014.
- Y. Li, D. Tarlow, M. Brockschmidt, and R.S. Zemel. Gated graph sequence neural networks. *ICLR*, 2016.
- P. Velickovic, G. Cucurull, A. Casanova, A. Romero,P. Lio, and Y. Bengio. Graph attention networks. *ICLR* 2018.
- K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.
- W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NIPS*, 2017.
- J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. *ICML*, 2017.
- M.Defferrard, X. Bresson, P. Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. *NIPS*, 2016.
- J. Bruna, W. Zaremba, A. Szlam, Y. LeCun. Spectral Networks and Locally Connected Networks on Graphs. *ICLR*, 2014.

References

- K. S. Tai, R. Socher, and C. D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *IJCNLP*, 2015.
- H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. *ICML*, 2016.
- A. Santoro, D. Raposo, D.G.T.Barrett, M. Malinowski, R. Pascanu, P.W. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *NIPS*, 2017.
- R.L. Murphy, B. Srinivasan, V.A. Rao, and B. Ribeiro. Relational Pooling for Graph Representations. *ICML*, 2019.
- H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. *NeurIPS*, 2019.
- 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*, 2019.
- W. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 2017.

References

- M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. *IJCNN*, 2005.
- F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks* 20(1):61–80, 2009.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Uri Alon, Eran Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications, *ICLR*, 2021.
- Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI*, 2018.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NIPS*, 2017.