

Advances in Machine Learning for Molecules

José Miguel Hernández-Lobato

Department of Engineering

University of Cambridge,

Microsoft Research Cambridge,

Alan Turing Institute

<http://jmh1.org>, jmh233@cam.ac.uk

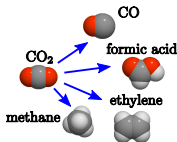
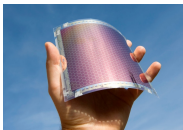
Machine Learning Summer School, Madrid, 2018.

Motivation

Molecules make everything we know of and have a huge impact in our lives.

New molecules and materials can potentially solve important existing challenges:

- Drug and medicine design for health care
- Energy production and storage
- Greenhouse gas conversion

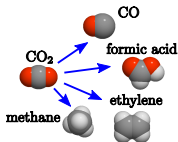
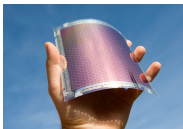


Motivation

Molecules make everything we know of and have a huge impact in our lives.

New molecules and materials can potentially solve important existing challenges:

- Drug and medicine design for health care
- Energy production and storage
- Greenhouse gas conversion



However, progress in drug and material discovery has been **slow** because of the cost of

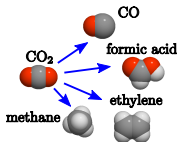
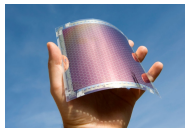
- collecting data and
 - making decisions based on that data,
- which require a lot of human intervention.

Motivation

Molecules make everything we know of and have a huge impact in our lives.

New molecules and materials can potentially solve important existing challenges:

- Drug and medicine design for health care
- Energy production and storage
- Greenhouse gas conversion



However, progress in drug and material discovery has been **slow** because of the cost of

- collecting data and
- making decisions based on that data,

which require a lot of human intervention. **But things are changing...**

More and more data is becoming available

Modern chemical-simulation tools based on **density functional theory** (DFT) can estimate the properties of molecules before they are made in the laboratory.

Extensive datasets are available with properties of real and virtual molecules.

More and more data is becoming available

Modern chemical-simulation tools based on **density functional theory** (DFT) can estimate the properties of molecules before they are made in the laboratory.

Extensive datasets are available with properties of real and virtual molecules.

Harvard Clean Energy Project:

In silico framework to study candidate structures for organic photovoltaics.

3.5M molecules, 30,000 CPU years to estimate molecule properties via DFT.



More and more data is becoming available

Modern chemical-simulation tools based on **density functional theory** (DFT) can estimate the properties of molecules before they are made in the laboratory.

Extensive datasets are available with properties of real and virtual molecules.

Harvard Clean Energy Project:

In silico framework to study candidate structures for organic photovoltaics.

3.5M molecules, 30,000 CPU years to estimate molecule properties via DFT.

QM9 dataset:

130,000 molecules with up to 9 atoms (not counting hydrogen) and with 13 properties estimated by DFT.



'Robot scientist' speeds up drug discovery

BY EMMA STOYE | 5 FEBRUARY 2015



Automated AI lab that learns and formulates hypotheses has identified promising anti-cancer and anti-malarial compounds

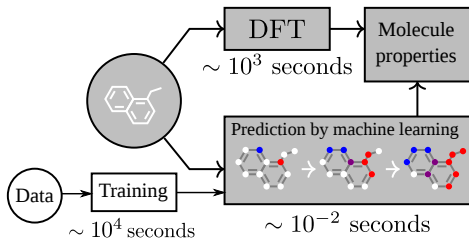
An artificial intelligence system – or ‘robot scientist’ – capable of screening potential drugs almost completely independently could speed up drug development, say the UK researchers who created it. The approach has already identified some promising leads, including an anti-cancer compound which also shows anti-malarial properties.



The role of machine learning

Machine learning (ML) can **accelerate and automate** the discovery process:

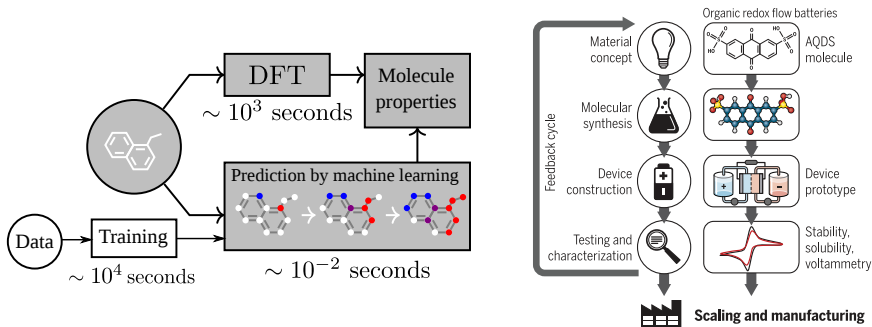
- DFT may take hours, while ML can make much **faster predictions** from data.



The role of machine learning

Machine learning (ML) can **accelerate and automate** the discovery process:

- DFT may take hours, while ML can make much **faster predictions** from data.
- ML can be used to **close the loop**: use collected data to guide new synthesis and characterization experiments or simulations.



Figures source: Gilmer et al. 2017 and Sanchez-Lengeling and Aspuru-Guzik 2018.

Existing libraries

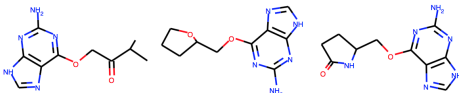
Molecule data can be loaded, analyzed and visualized using public libraries, e.g.

RDKit: <https://www.rdkit.org>



```
In [75]: Draw.MolsToGridImage( mols[:3] )
```

Out[75]:



Once we have loaded the data, we can apply a machine learning (ML) method.

Existing libraries

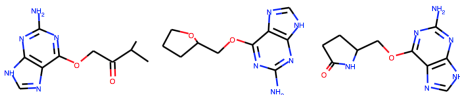
Molecule data can be loaded, analyzed and visualized using public libraries, e.g.

RDKit: <https://www.rdkit.org>



```
In [75]: Draw.MolsToGridImage( mols[:3] )
```

Out[75]:



Once we have loaded the data, we can apply a machine learning (ML) method.

Challenges

Common ML methods only accept **vectorial** input data and

molecules are typically **graphs** with **nodes** being atoms and **edges** chemical bonds!



Outline

We will aim to cover...

- Representations of molecules for machine learning
 - Molecular fingerprints
 - SMILES
 - Graph neural networks (GNNs)

For a more complete review see Sanchez-Lengeling and Aspuru-Guzik 2018.

- Reaction prediction problem
 - GNN-based method
 - seq2seq method

Part I: Representations of molecules

Molecular fingerprints

Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

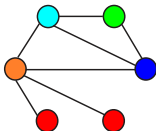
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

→ **Assign integers** to atoms by applying hash function to atom features.

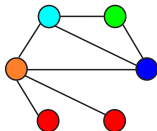
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

→ **Assign integers** to atoms by applying hash function to atom features.

For $r = 1$ to R

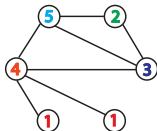
Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Initial hashing



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

→ For $r = 1$ to R

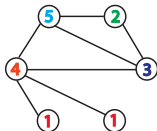
Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Initial hashing



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

For $r = 1$ to R

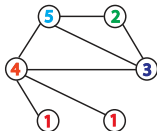
→ **Concatenate** atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Initial hashing



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

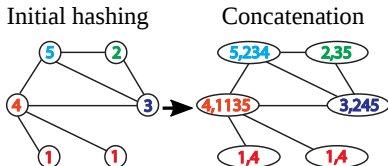
For $r = 1$ to R

→ **Concatenate** atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

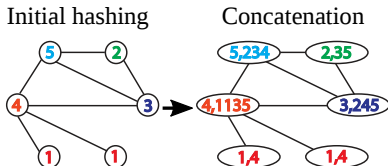
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

→ **Assign new integers** to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

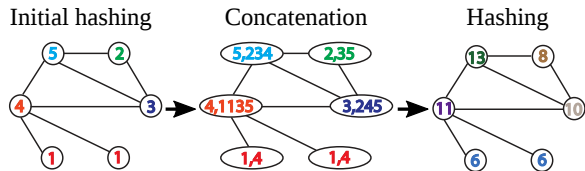
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

→ **Assign new integers** to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

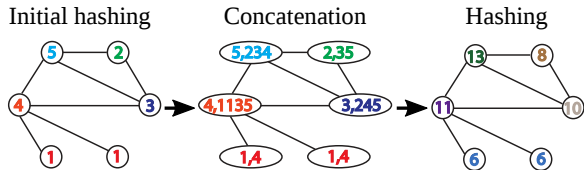
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

→ Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

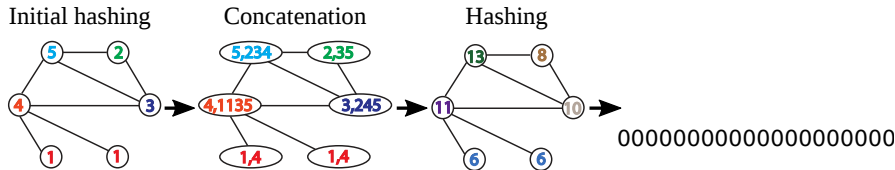
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

→ Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

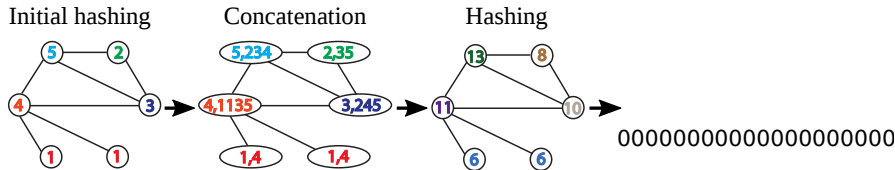
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Crate L -dimensional zero vector \mathbf{f} .

→ **Map generated integers** to an entry in **f** which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

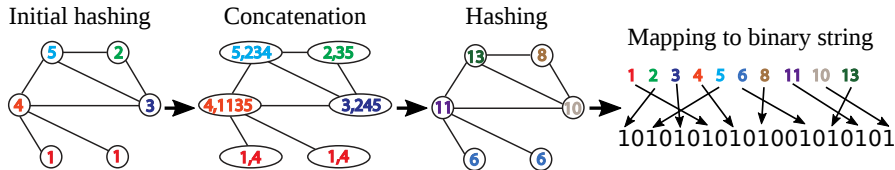
For $r = 1$ to R

Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

→ **Map generated integers** to an entry in \mathbf{f} which is set to 1.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

For $r = 1$ to R

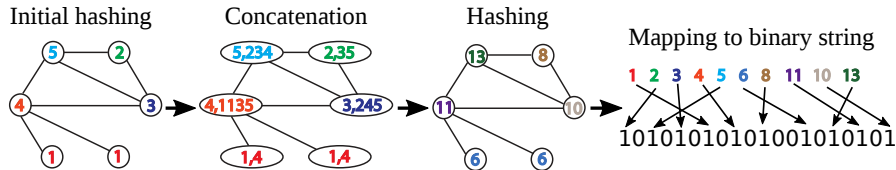
Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Each generated integer represents a **fragment** in the graph.



Molecular fingerprints

An encoding of the molecular graph into a **binary string** [Rogers et al. 2010].

Input: molecular graph, radius parameter R , length L

Assign integers to atoms by applying hash function to atom features.

For $r = 1$ to R

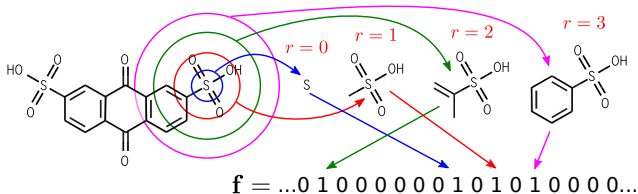
Concatenate atom integers with integers of neighboring atoms.

Assign new integers to atoms by applying hash function to concatenation.

Create L -dimensional zero vector \mathbf{f} .

Map generated integers to an entry in \mathbf{f} which is set to 1.

Each generated integer represents a **fragment** in the graph. Another example:



Pros and cons of fingerprints

Advantages:

- Very fast to compute and widely available (e.g. in RDKit).
- Produce very good predictive performance in practice.
- Easy to interpret: features represent the presence of substructures.

Pros and cons of fingerprints

Advantages:

- Very fast to compute and widely available (e.g. in RDKit).
- Produce very good predictive performance in practice.
- Easy to interpret: features represent the presence of substructures.

Disadvantages:

- The generated features are handcrafted and not data dependent.
- They are not smooth: similar fragments will map to different bits.

SMILES

SMILES

Simplified Molecular Input Line Entry System.

Allow us to represent a molecular graph in line notation.

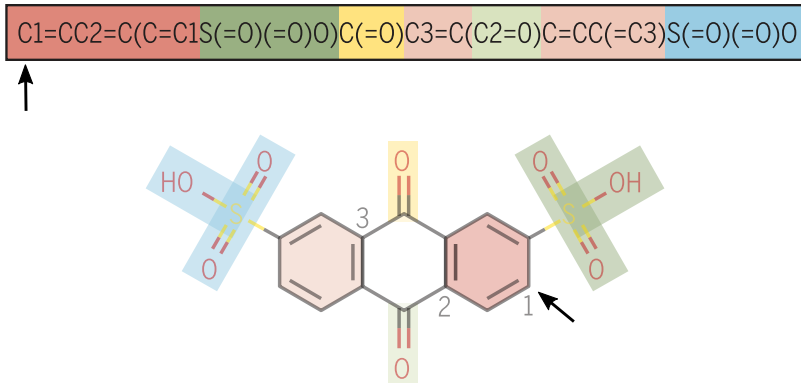
Examples:

- CC represents CH_3CH_3 (ethane)
- CC(=O)O represents CH_3COOH (acetic acid).
- C1CCCCC1 represents C_6H_{12} (cyclohexane).

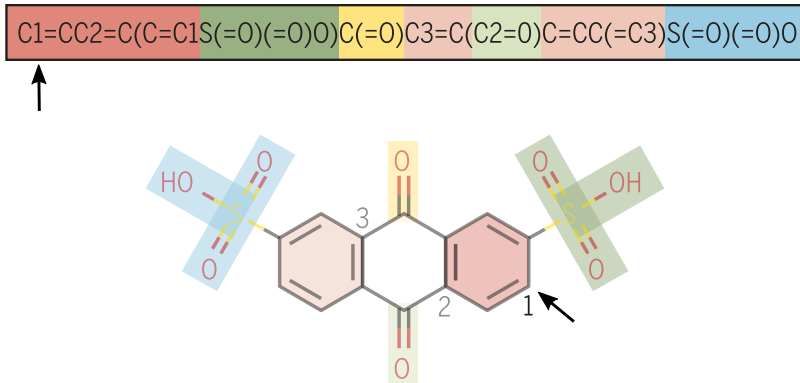
Some of the key elements of the SMILES format:

- Hydrogen atoms are implicit.
- Each atoms is connected to the previous atom in the sequence.
- Parenthesis indicate branches.
- Digits are used to label beginning and end of a cycle.
- Single bonds implicit, = used for double bonds, # for triple bonds.

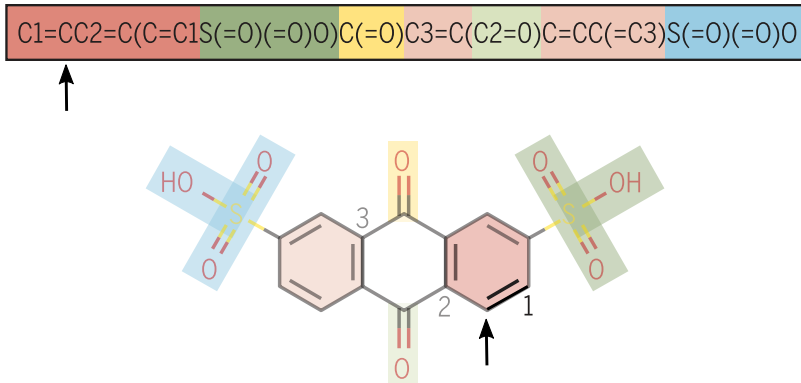
Example



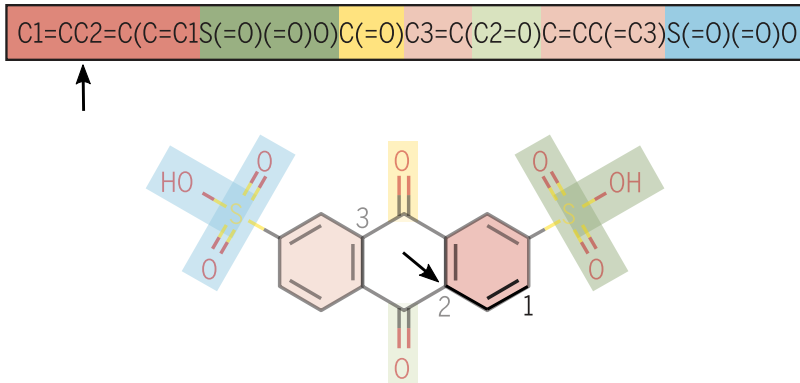
Example



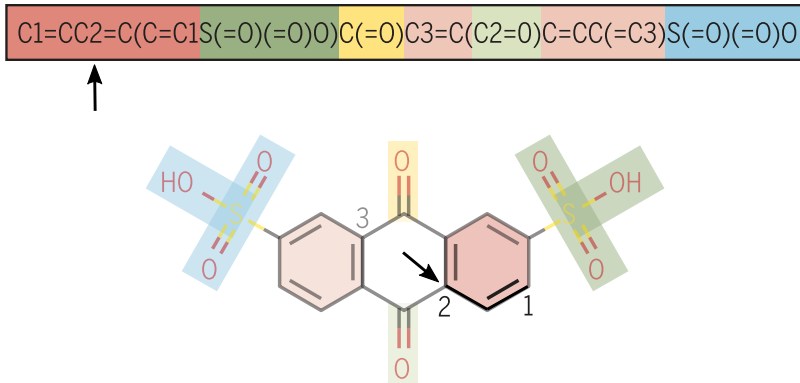
Example



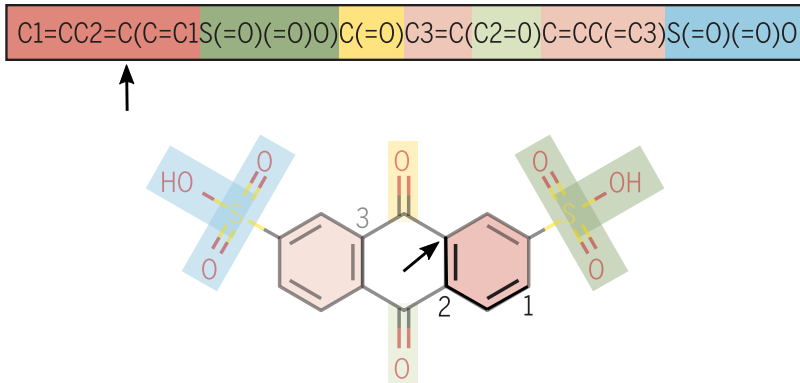
Example



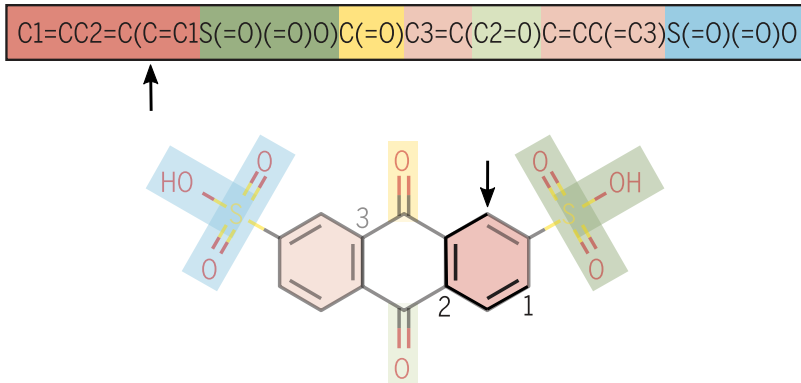
Example



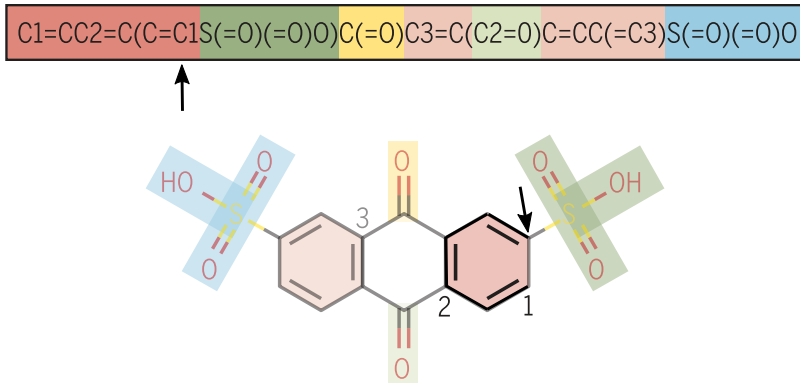
Example



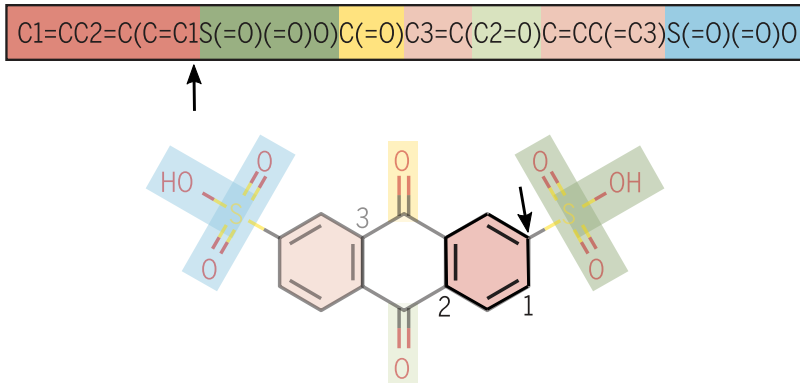
Example



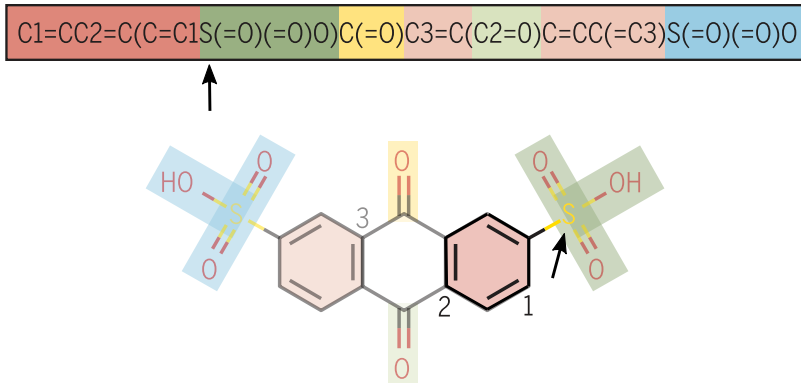
Example



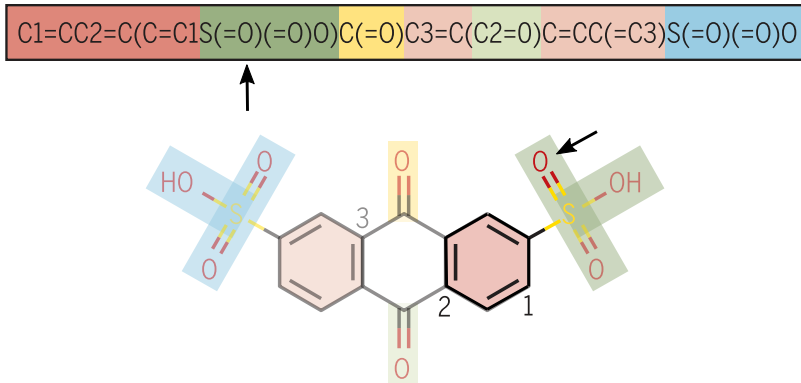
Example



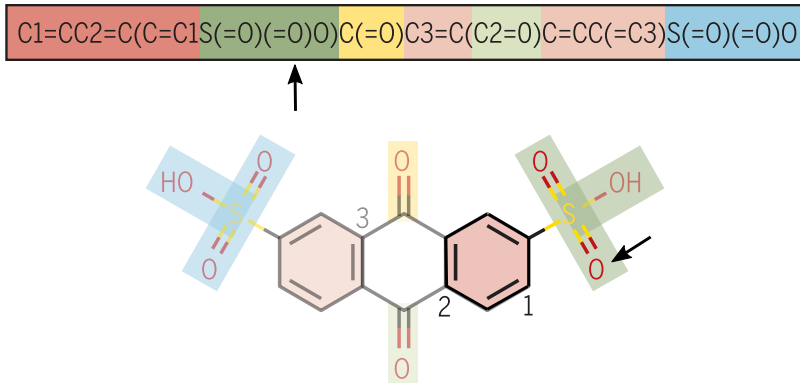
Example



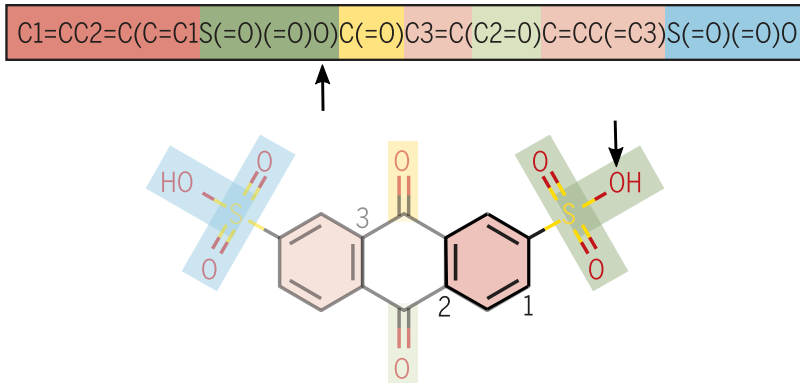
Example



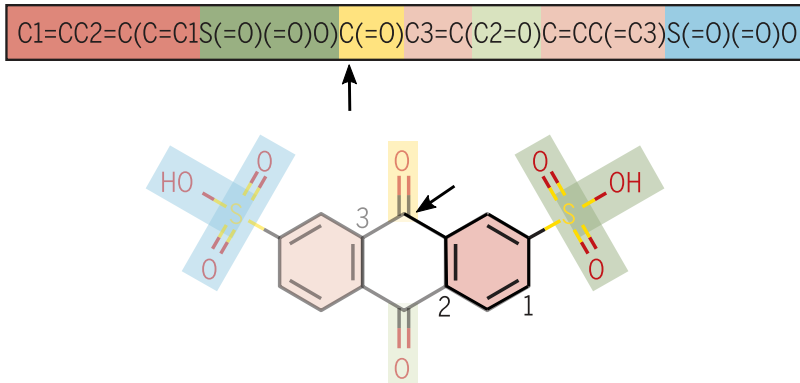
Example



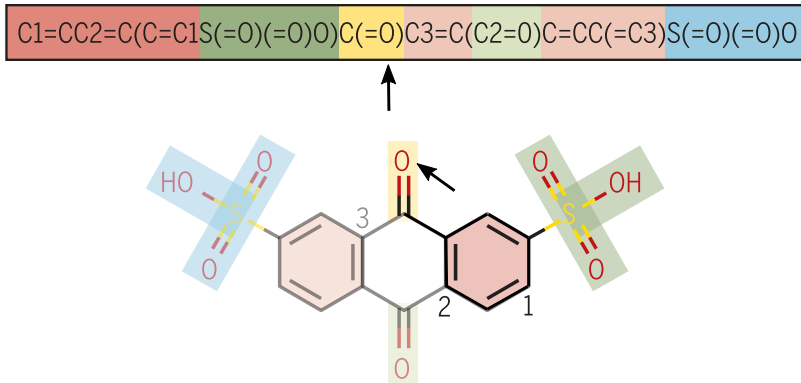
Example



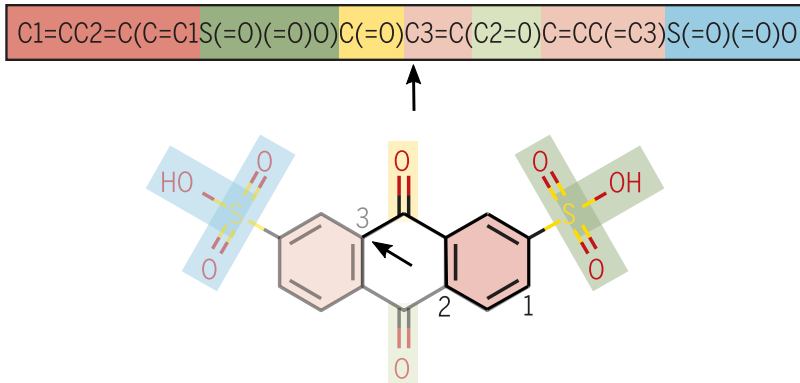
Example



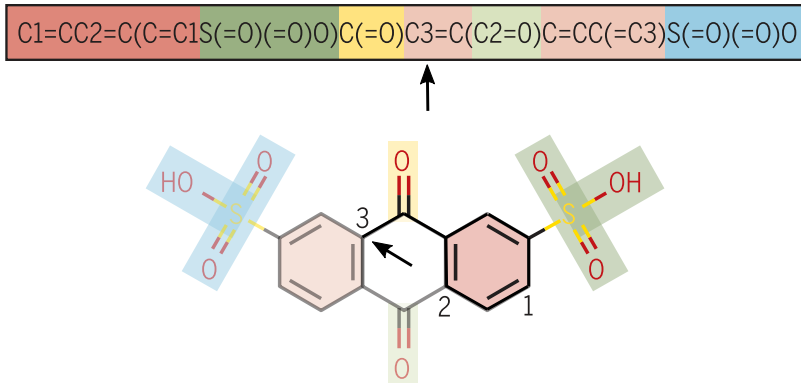
Example



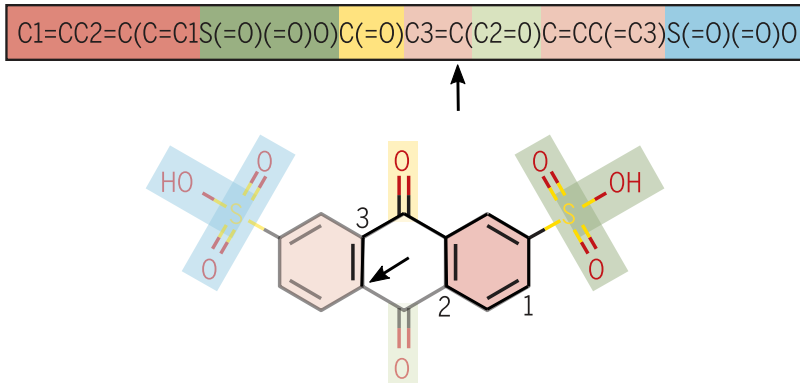
Example



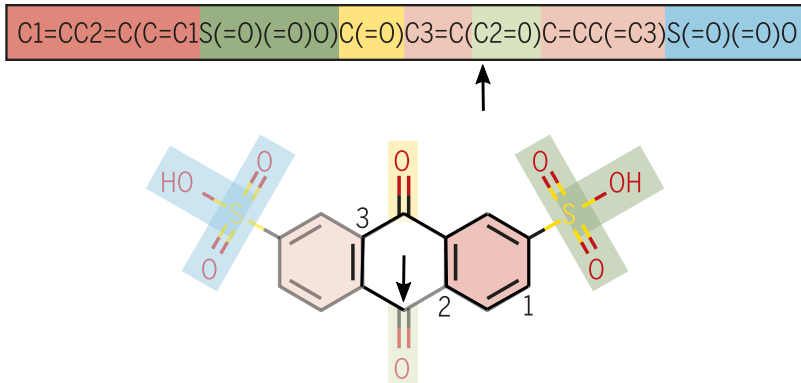
Example



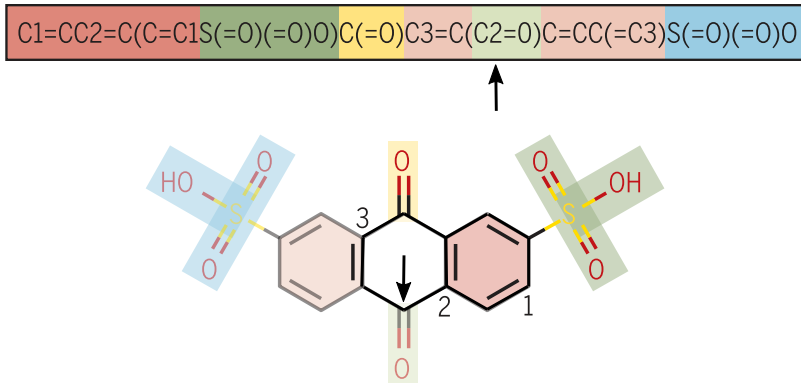
Example



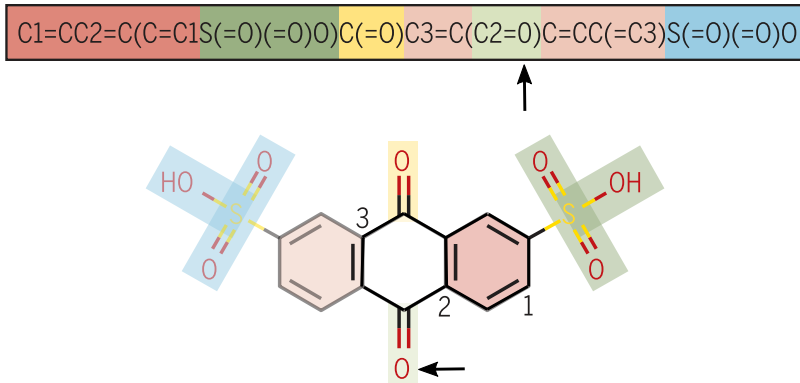
Example



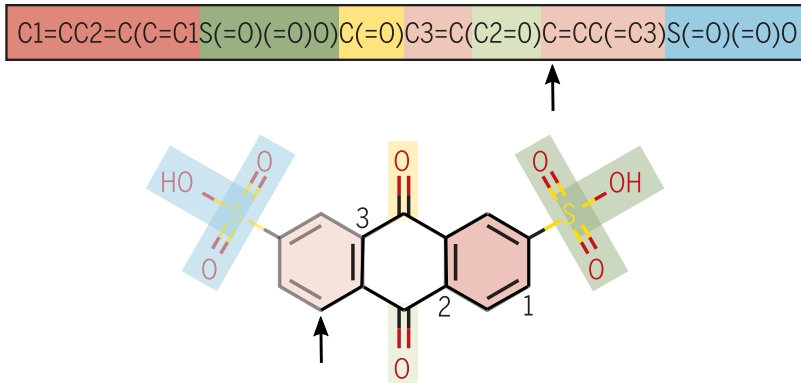
Example



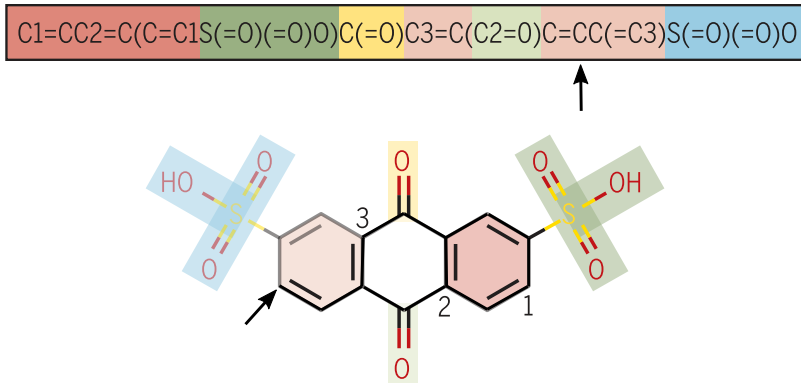
Example



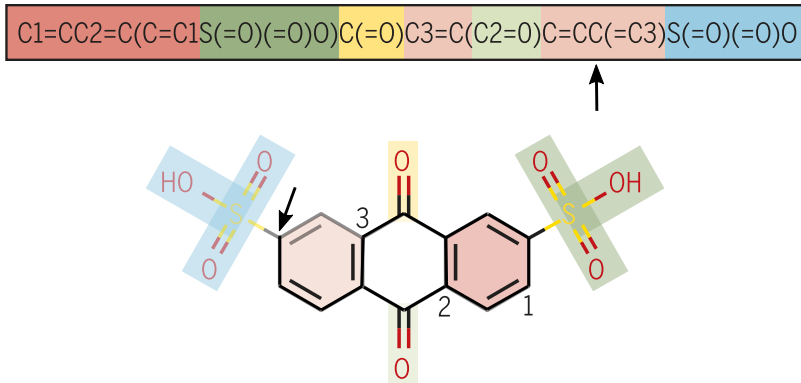
Example



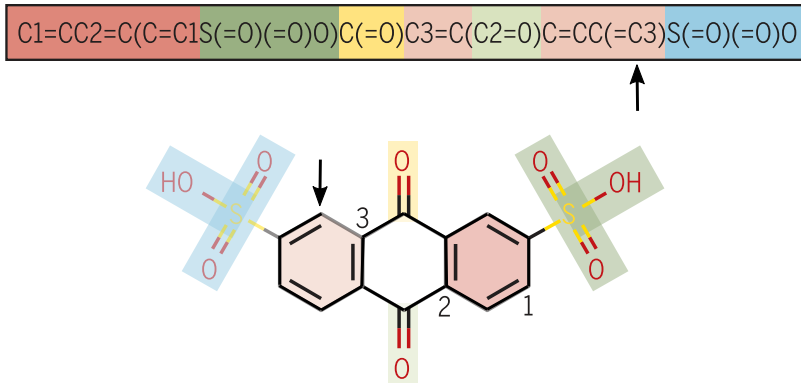
Example



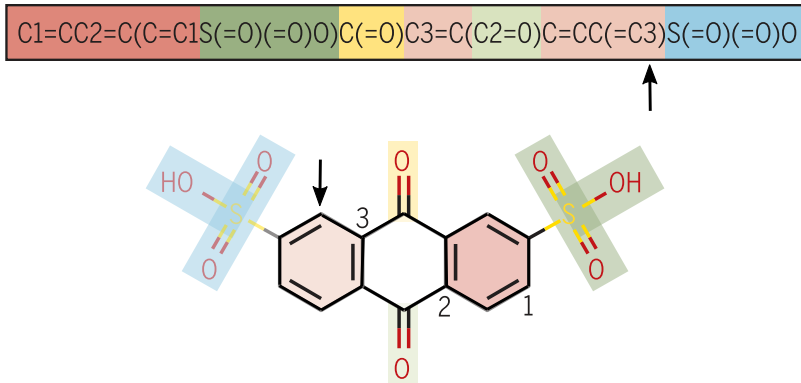
Example



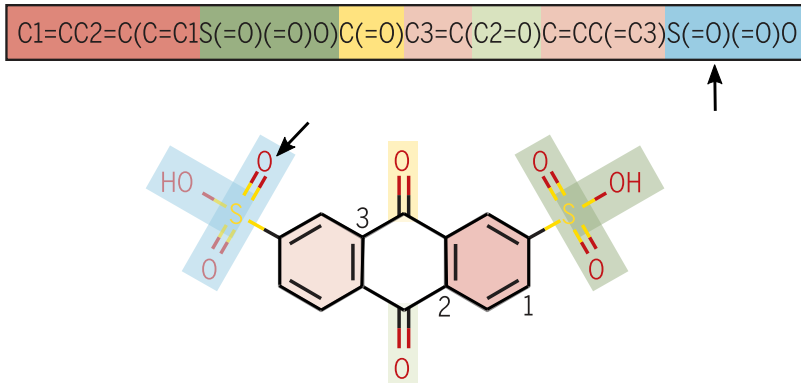
Example



Example



Example



Example

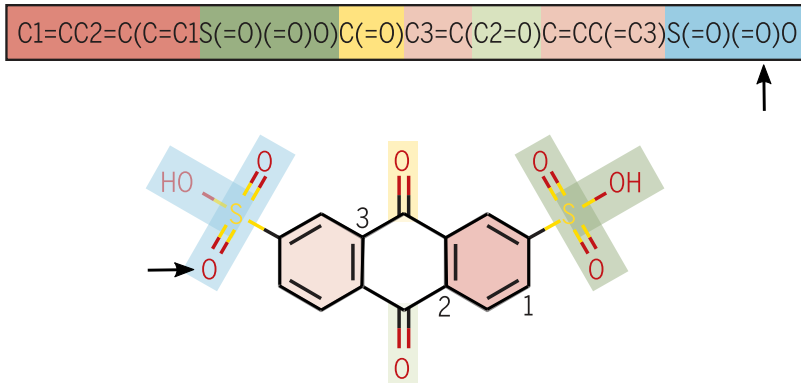
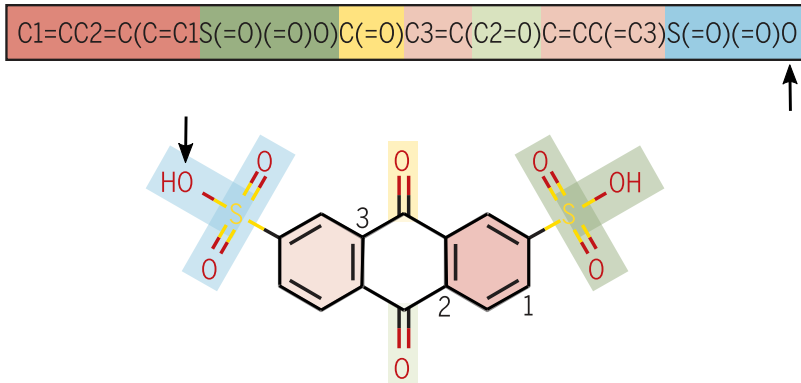


Figure source: Sanchez-Lengeling and Aspuru-Guzik, 2018.

Example

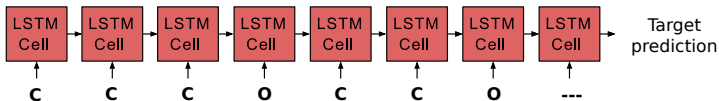


Neural networks and SMILES strings

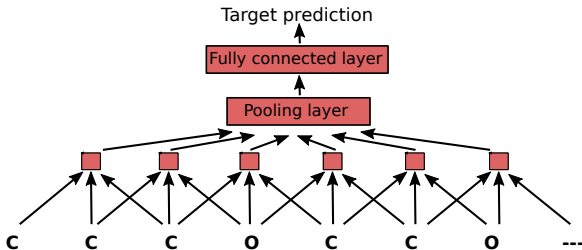
SMILES strings are easily processed by neural networks.

Padding with white spaces can be used to guarantee all strings have equal size.

Recurrent neural networks:



1D convolutional neural networks:



Pros and cons of SMILES

Advantages:

- Molecules are easily encoded as simple text strings.
- Relatively easy to understand by humans.
- NLP methods can be applied to molecules (data dependent representation).

Pros and cons of SMILES

Advantages:

- Molecules are easily encoded as simple text strings.
- Relatively easy to understand by humans.
- NLP methods can be applied to molecules (data dependent representation).

Disadvantages:

- Same molecule is represented by many different SMILES strings. Lack of invariance to atom ordering! Although **canonical** SMILES somewhat circumvent this by choosing a specific ordering.
- Atoms close in the graph may be far away within a SMILES string: short-range dependencies may be transformed into long-range ones.

Graph neural networks

Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs?

Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.

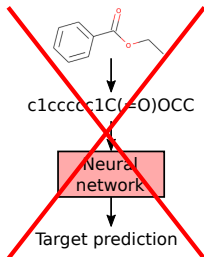
Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.



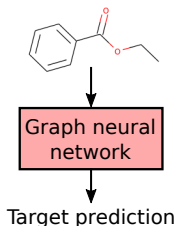
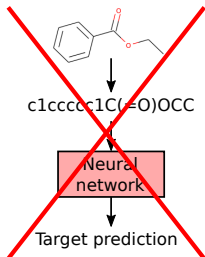
Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.



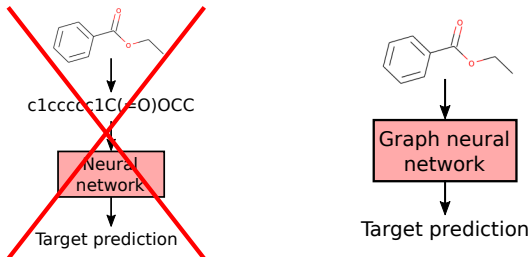
Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.



GNNs have been developed for more than a decade, but with very rapid recent growth.

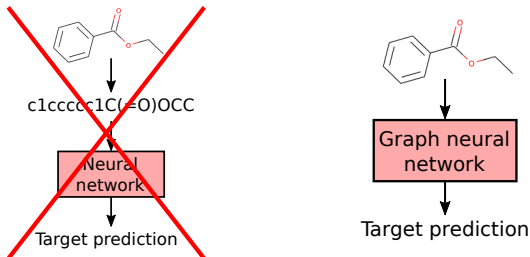
Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.



GNNs have been developed for more than a decade, but with very rapid recent growth.

Many different works: Gori et al. 2005, Scarselli et al. 2005, 2009, Bruna et al. 2014, Duvenaud et al. 2015, Li et al. 2016, Kipf & Welling 2016, Kearnes et al. 2016, Schütt et al. 2017, Jin et al. 2017, Gilmer et al. 2017, etc.

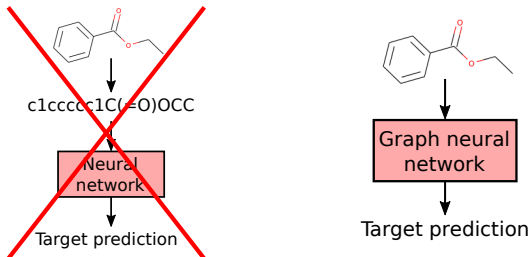
Graph neural networks (GNN)

Unlike SMILES, the **molecular graph** naturally encodes

- Invariance to permutation of nodes.
- Distances between atoms.

Can we directly work with graphs? Yes!

Using GNN, which **structure computations according to the graph connectivity**.



GNNs have been developed for more than a decade, but with very rapid recent growth.

Many different works: Gori et al. 2005, Scarselli et al. 2005, 2009, Bruna et al. 2014, Duvenaud et al. 2015, Li et al. 2016, Kipf & Welling 2016, Kearnes et al. 2016, Schütt et al. 2017, Jin et al. 2017, Gilmer et al. 2017, etc.

We will follow the general definition given by Battaglia et al. 2018.

Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- ① $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- ② $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- ③ \mathbf{u} are global features summarizing the graph properties.

Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- ① $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- ② $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- ③ \mathbf{u} are global features summarizing the graph properties.

Approach: $\{\mathbf{e}_{j \sim k}\}$, $\{\mathbf{v}_i\}_{i=1}^N$ and \mathbf{u} are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of \mathbf{u} as input to for example an MLP.

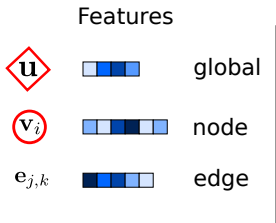
Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- 1 $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- 2 $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- 3 \mathbf{u} are global features summarizing the graph properties.

Approach: $\{\mathbf{e}_{j \sim k}\}$, $\{\mathbf{v}_i\}_{i=1}^N$ and \mathbf{u} are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of \mathbf{u} as input to for example an MLP.



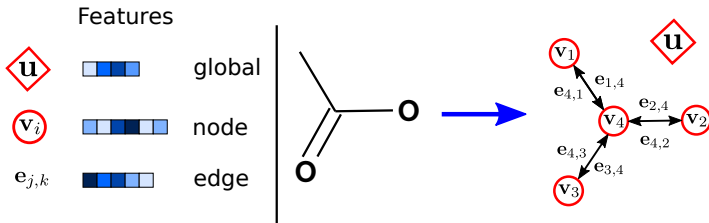
Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- 1 $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- 2 $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- 3 \mathbf{u} are global features summarizing the graph properties.

Approach: $\{\mathbf{e}_{j \sim k}\}$, $\{\mathbf{v}_i\}_{i=1}^N$ and \mathbf{u} are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of \mathbf{u} as input to for example an MLP.



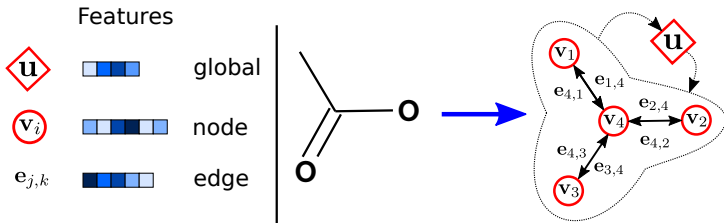
Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- 1 $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- 2 $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- 3 \mathbf{u} are global features summarizing the graph properties.

Approach: $\{\mathbf{e}_{j \sim k}\}$, $\{\mathbf{v}_i\}_{i=1}^N$ and \mathbf{u} are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of \mathbf{u} as input to for example an MLP.



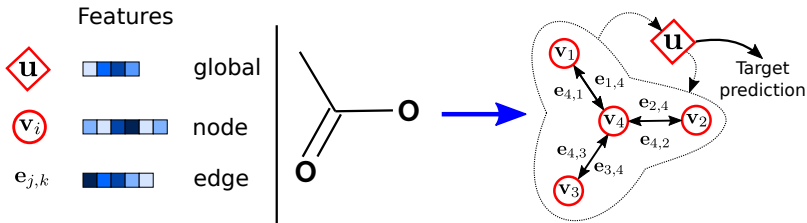
Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- 1 $\{e_{j \sim k}\}$ are features for edges between nodes j and k .
- 2 $\{v_i\}_{i=1}^N$ are node features.
- 3 u are global features summarizing the graph properties.

Approach: $\{e_{j \sim k}\}$, $\{v_i\}_{i=1}^N$ and u are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of u as input to for example an MLP.



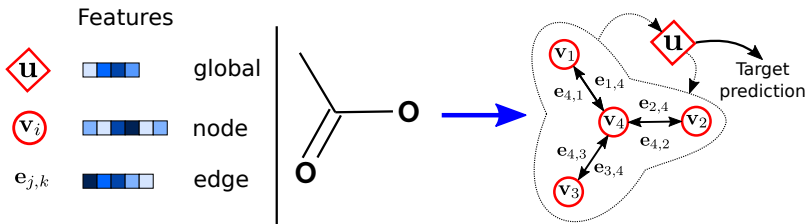
Key elements of GNNs

A GNN includes the following (**vectorial**) variables:

- 1 $\{\mathbf{e}_{j \sim k}\}$ are features for edges between nodes j and k .
- 2 $\{\mathbf{v}_i\}_{i=1}^N$ are node features.
- 3 \mathbf{u} are global features summarizing the graph properties.

Approach: $\{\mathbf{e}_{j \sim k}\}$, $\{\mathbf{v}_i\}_{i=1}^N$ and \mathbf{u} are **iteratively updated** during a **forward pass**.

Predictions: made by using the final value of \mathbf{u} as input to for example an MLP.



$\{\mathbf{e}_{j \sim k}\}$ could be initialized to indicate single, double or triple bond.

$\{\mathbf{v}_i\}_{i=1}^N$ could be initialized to indicate atom type, degree, electronegativity, etc.

Set functions and auxiliary variables

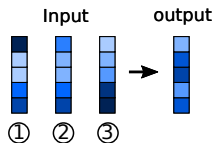
Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.

Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

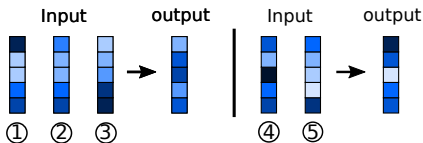
It is **invariant to input permutation** and accepts a **variable number of arguments**.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

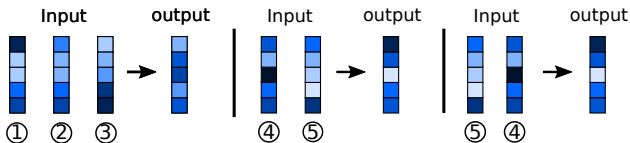
It is **invariant to input permutation** and accepts a **variable number of arguments**.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

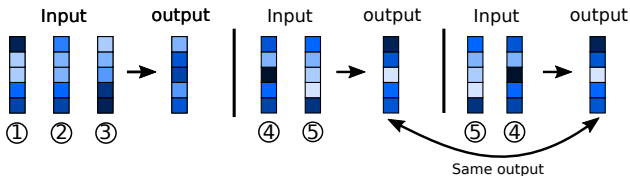
It is **invariant to input permutation** and accepts a **variable number of arguments**.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

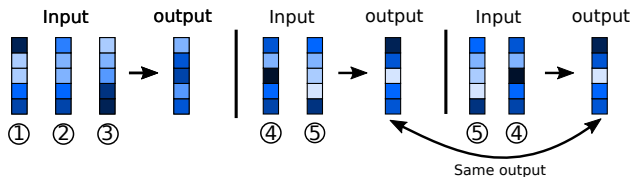
It is **invariant to input permutation** and accepts a **variable number of arguments**.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.

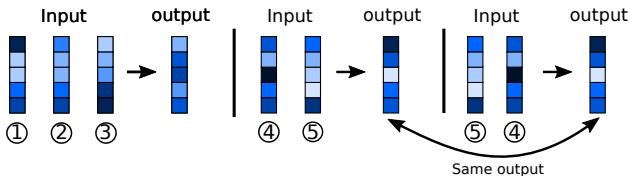


Examples: elementwise summation, mean, maximum, etc.

Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.



Examples: elementwise summation, mean, maximum, etc.

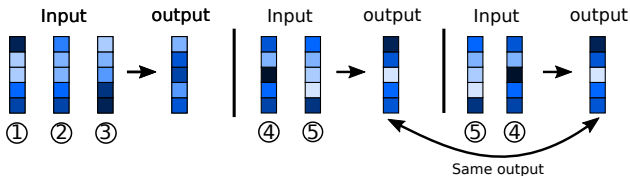
GNNs use **set functions** to summarize...

- edge features of incoming edges to node i :
 $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$
- all edge features: $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$
- all node features: $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.



Examples: elementwise summation, mean, maximum, etc.

GNNs use **set functions** to summarize...

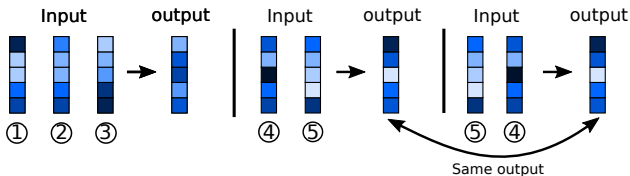
- edge features of incoming edges to node i :
 $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$
- all edge features: $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$
- all node features: $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

This creates **auxiliary features** $\{\bar{\mathbf{e}}_i\}$, $\bar{\mathbf{e}}$ and $\bar{\mathbf{v}}$.

Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.

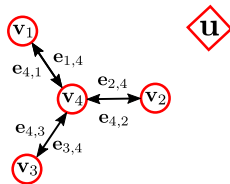


Examples: elementwise summation, mean, maximum, etc.

GNNs use **set functions** to summarize...

- edge features of incoming edges to node i :
 $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$
- all edge features: $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$
- all node features: $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

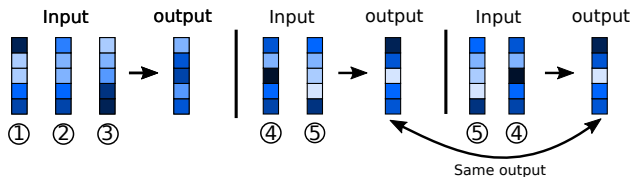
This creates **auxiliary features** $\{\bar{\mathbf{e}}_i\}$, $\bar{\mathbf{e}}$ and $\bar{\mathbf{v}}$.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.

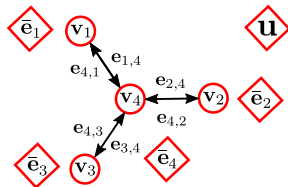


Examples: elementwise summation, mean, maximum, etc.

GNNs use **set functions** to summarize...

- edge features of incoming edges to node i :
 $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$
- all edge features: $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$
- all node features: $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

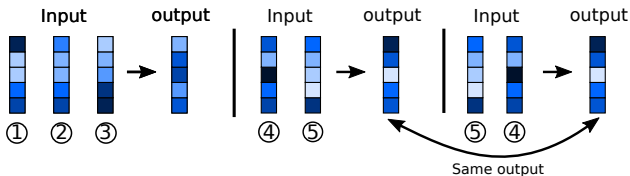
This creates **auxiliary features** $\{\bar{\mathbf{e}}_i\}$, $\bar{\mathbf{e}}$ and $\bar{\mathbf{v}}$.



Set functions and auxiliary variables

Set function: input is a set and output is a single element summarizing the input set.

It is **invariant to input permutation** and accepts a **variable number of arguments**.

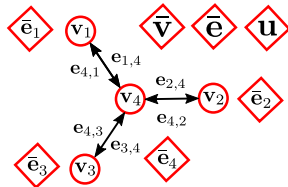


Examples: elementwise summation, mean, maximum, etc.

GNNs use **set functions** to summarize...

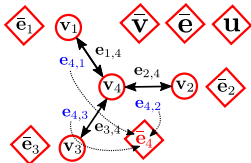
- edge features of incoming edges to node i :
 $\bar{e}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$
- all edge features: $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$
- all node features: $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

This creates **auxiliary features** $\{\bar{e}_i\}$, $\bar{\mathbf{e}}$ and $\bar{\mathbf{v}}$.

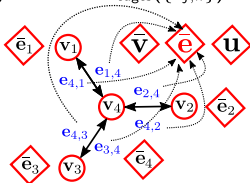


Examples of set functions for summarizing incoming edges, all edges and all nodes:

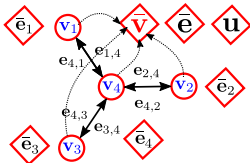
• $\bar{e}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$



• $\bar{e} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$

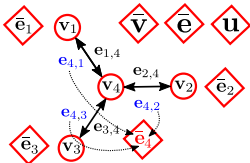


• $\bar{v} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$

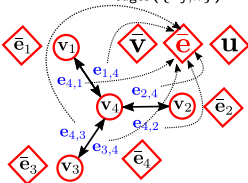


Examples of set functions for summarizing incoming edges, all edges and all nodes:

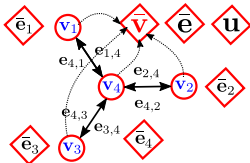
• $\bar{e}_i \leftarrow \mathcal{S}_{e2n}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$



• $\bar{e} \leftarrow \mathcal{S}_{edges}(\{\mathbf{e}_{j,k}\})$



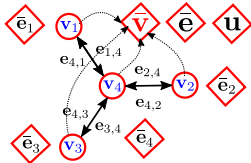
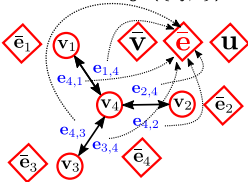
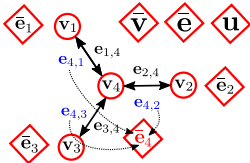
• $\bar{v} \leftarrow \mathcal{S}_{nodes}(\{\mathbf{v}_i\})$



The **set functions** are the equivalent of **pooling** in CNNs.

Examples of set functions for summarizing incoming edges, all edges and all nodes:

$$\bullet \bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\}) \quad \bullet \bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\}) \quad \bullet \bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$$

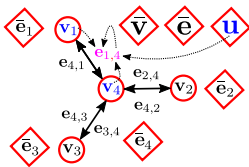


The **set functions** are the equivalent of **pooling** in CNNs.

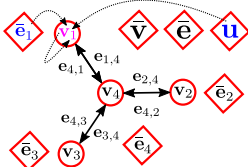
Update functions

GNNs use the following **update functions** to update edge, node and global features:

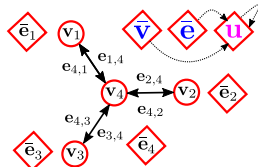
$$\bullet \mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$$



$$\bullet \mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$$

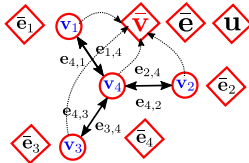
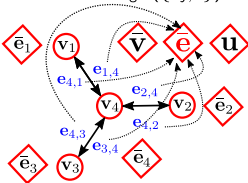
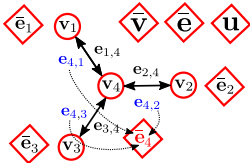


$$\bullet \mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$$



Examples of set functions for summarizing incoming edges, all edges and all nodes:

$$\bullet \bar{\mathbf{e}}_j \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\}) \quad \bullet \bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\}) \quad \bullet \bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$$

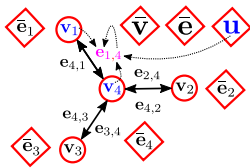


The **set functions** are the equivalent of **pooling** in CNNs.

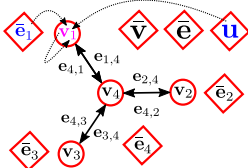
Update functions

GNNs use the following **update functions** to update edge, node and global features:

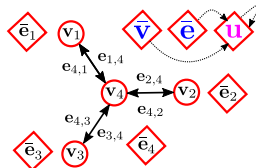
$$\bullet \mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$$



$$\bullet \mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$$



$$\bullet \mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$$

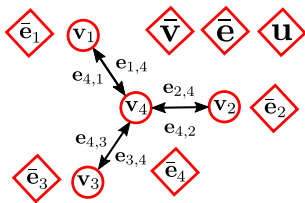


As in CNNs, the same **update functions** are **reused** across all nodes and edges.

The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

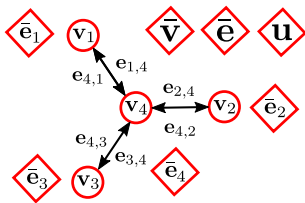
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$            # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                                # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$                # Update global features  
return MLP( $\mathbf{u}$ )                                                  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

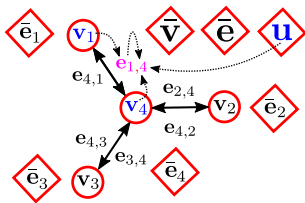
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

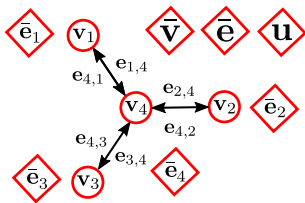
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return MLP( $\mathbf{u}$ ) # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

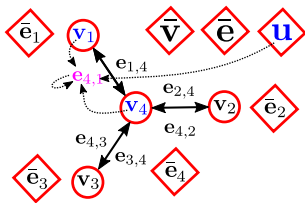
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

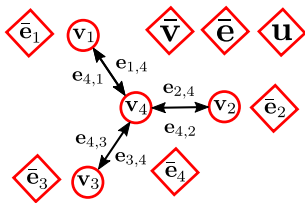
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return  $\text{MLP}(\mathbf{u})$  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

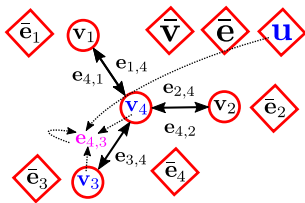
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

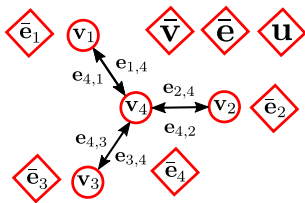
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return  $\text{MLP}(\mathbf{u})$  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

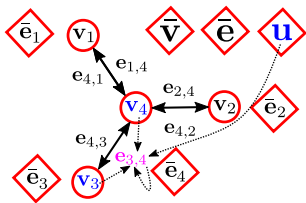
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
    return MLP( $\mathbf{u}$ )                                         # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

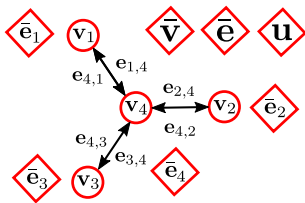
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return  $\text{MLP}(\mathbf{u})$  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

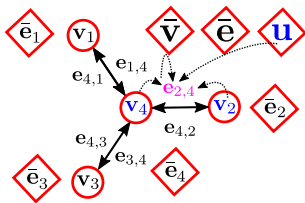
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

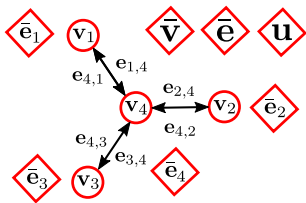
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return  $\text{MLP}(\mathbf{u})$  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

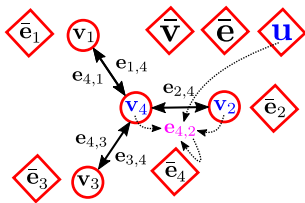
```
→ for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
    for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
         $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
         $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$   
     $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features  
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

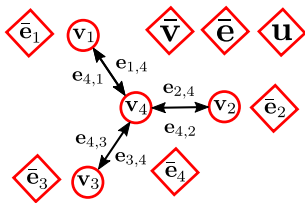
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
→  $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$  # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$  # Update features for node  $i$   
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$  # Summarize all edges  
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$  # Summarize all nodes  
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features  
return MLP( $\mathbf{u}$ ) # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
→ for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                             # Compute prediction from features  $\mathbf{u}$ 
```

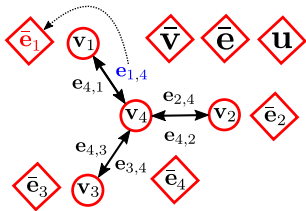


The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```

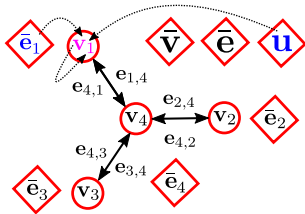
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$          # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                              # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$              # Update global features
return  $\text{MLP}(\mathbf{u})$                                              # Compute prediction from features  $\mathbf{u}$ 
    
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

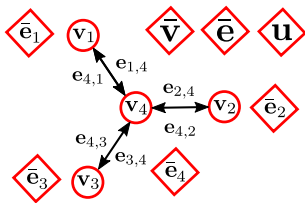
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$            # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$   
→  $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                            # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$              # Update global features  
return MLP( $\mathbf{u}$ )                                                  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
→ for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                    # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```

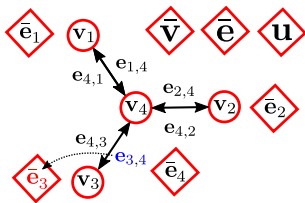


The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```

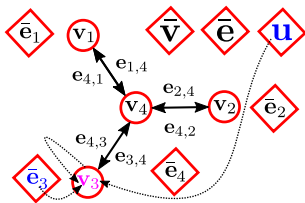
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$          # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                              # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$              # Update global features
return  $\text{MLP}(\mathbf{u})$                                              # Compute prediction from features  $\mathbf{u}$ 
    
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

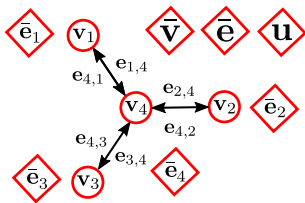
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do  
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features  
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do  
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$            # Summarize incoming edges to  $i$   
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$   
→  $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges  
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                            # Summarize all nodes  
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$            # Update global features  
return MLP( $\mathbf{u}$ )                                                  # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

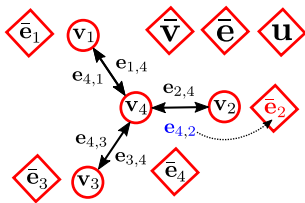
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
→ for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

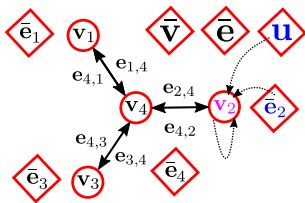
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

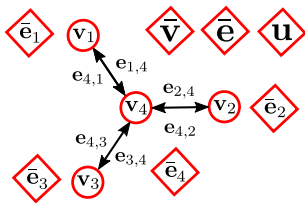
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
→  $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                  # Summarize all nodes
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
→ for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                      # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```

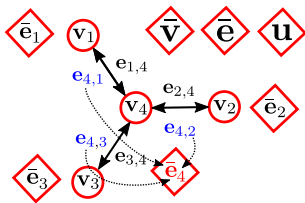


The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```

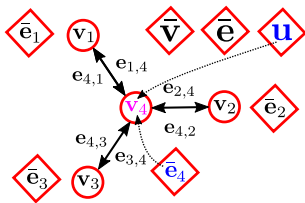
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$          # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                              # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                            # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$            # Update global features
return  $\text{MLP}(\mathbf{u})$                                            # Compute prediction from features  $\mathbf{u}$ 
    
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

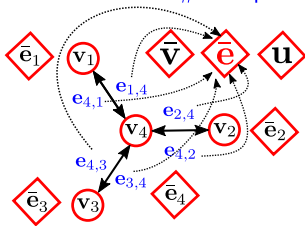
```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
→  $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                        # Summarize all edges
     $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                  # Summarize all nodes
     $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$  # Update global features
return MLP( $\mathbf{u}$ )                                           # Compute prediction from features  $\mathbf{u}$ 
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```
for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$  # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$            # Update features for node  $i$ 
 $\rightarrow \bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                      # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                        # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$        # Update global features
return MLP( $\mathbf{u}$ )                                             # Compute prediction from features  $\mathbf{u}$ 
```

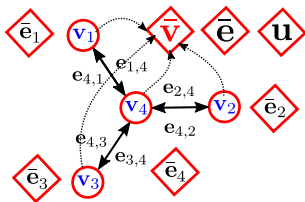


The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```

for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$          # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$                  # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                                # Summarize all edges
 $\rightarrow \bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                          # Summarize all nodes
 $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$              # Update global features
return  $\text{MLP}(\mathbf{u})$                                              # Compute prediction from features  $\mathbf{u}$ 
    
```

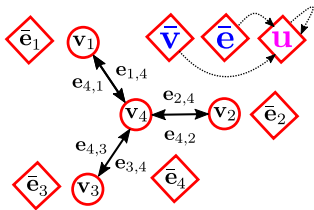


The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

```

for  $\mathbf{e}_{j \sim k}$  in  $\{\mathbf{e}_{j \sim k}\}$  do
     $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$            # Update edge features
for  $\mathbf{v}_i$  in  $\{\mathbf{v}_i\}$  do
     $\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$          # Summarize incoming edges to  $i$ 
     $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$              # Update features for node  $i$ 
 $\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$                          # Summarize all edges
 $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$                        # Summarize all nodes
→  $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$          # Update global features
return  $\text{MLP}(\mathbf{u})$                                          # Compute prediction from features  $\mathbf{u}$ 
    
```



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

for $\mathbf{e}_{j \sim k}$ in $\{\mathbf{e}_{j \sim k}\}$ do

$\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$ # Update edge features

for \mathbf{v}_i in $\{\mathbf{v}_i\}$ do

$\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$ # Summarize incoming edges to i

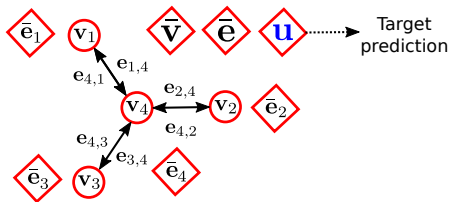
$\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$ # Update features for node i

$\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$ # Summarize all edges

$\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$ # Summarize all nodes

$\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$ # Update global features

→ return $\text{MLP}(\mathbf{u})$ # Compute prediction from features \mathbf{u}



The forward pass in a GNN

Input: initial $\{\mathbf{v}_i\}_{i=1}^N$, $\{\mathbf{e}_{j \sim k}\}$ and \mathbf{u} .

for $\mathbf{e}_{j \sim k}$ in $\{\mathbf{e}_{j \sim k}\}$ do

$\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u})$ # Update edge features

for \mathbf{v}_i in $\{\mathbf{v}_i\}$ do

$\bar{\mathbf{e}}_i \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\})$ # Summarize incoming edges to i

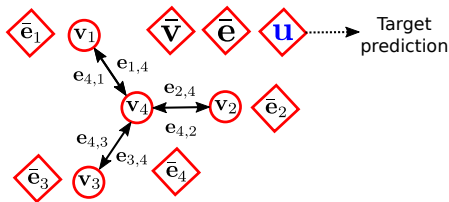
$\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u})$ # Update features for node i

$\bar{\mathbf{e}} \leftarrow \mathcal{S}_{\text{edges}}(\{\mathbf{e}_{j,k}\})$ # Summarize all edges

$\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$ # Summarize all nodes

$\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u})$ # Update global features

→ return MLP(\mathbf{u}) # Compute prediction from features \mathbf{u}

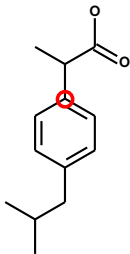


What if the number of layers L is larger than 1?

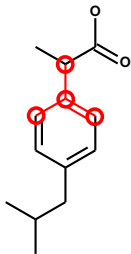
GNNs and message passing

Without \mathbf{u} , the information that a node has after completing the forward pass for the l -th layer is given by the **nodes and edges that are at most l hops away**.

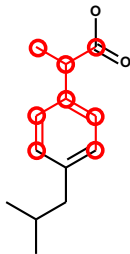
Example for a single initial node:



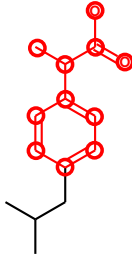
$l = 0$



$l = 1$



$l = 2$



$l = 3$

Nodes and edges further away than L hops will not share any information without \mathbf{u} .

\mathbf{u} will allow nodes and edges to access a summary of the current overall graph state.

Specific implementations of GNNs

- **Message passing neural network (MPNN)**. Gilmer et al. 2017.

- $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u}) \equiv \mathbf{A}(\mathbf{e}_{j,k}^{\text{initial}}) \mathbf{v}_k$.
- $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u}) \equiv \text{RNN}(\bar{\mathbf{e}}_i, \mathbf{v}_i)$.
- $\mathbf{u}^{\text{new}} \leftarrow \mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u}) \equiv \bar{\mathbf{v}}$.
- $\bar{\mathbf{e}}_i^{\text{new}} \leftarrow \mathcal{S}_{\text{e2n}}(\{\mathbf{e}_{i,k} : k = 1, \dots, N\}) \equiv \sum_k \mathbf{e}_{i,k}^{\text{T}}$.
- $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\})$ is given by the set2set model from Vinyals et al. (2015).
- $\bar{\mathbf{e}}$ and $\mathcal{S}_{\text{edges}}$ are not used.
- Includes a fully connected *master* node.

- **Gated graph neural network (GGNN)**. Li et al. 2016.

Like MPNN, but with **soft attention** to obtain $\bar{\mathbf{v}}$:

- $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\}) \equiv \sum_i \sigma(f_{\text{gate}}(\mathbf{v}_i)) f_{\text{up}}(\mathbf{v}_i)$,
where $\sigma(\cdot)$ is the logistic function and f_{gate} and f_{up} are linear functions.

- **Weisfeiler-Lehman network (WLN)**. Jin et al. 2017.

Like MPNN, but with the following changes:

- $\mathbf{e}_{j,k}^{\text{new}} \leftarrow \mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u}) \equiv \text{NN}(\mathbf{v}_k, \mathbf{e}_{j,k}^{\text{initial}})$.
- $\mathbf{v}_i^{\text{new}} \leftarrow \mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u}) \equiv \text{NN}(\bar{\mathbf{e}}_i, \mathbf{v}_i)$ when $l = 1, \dots, L - 1$ and
 $\mathbf{v}_i^{\text{new}} \leftarrow \sum_{j \in N(i)} \text{Linear}(\mathbf{v}_j) \odot \text{Linear}(\mathbf{e}_{i,j}^{\text{initial}}) \odot \text{Linear}(\mathbf{v}_i)$ otherwise.
- $\bar{\mathbf{v}} \leftarrow \mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\}) \equiv \sum_i \mathbf{v}_i$.

- **Neural graph fingerprints (NGFs)**. Duvenaud et al. 2015.

Like MPNN, but with $\mathcal{U}_{\text{edge}}(\mathbf{e}_{j,k}, \mathbf{v}_j, \mathbf{v}_k, \mathbf{u}) \equiv \mathbf{v}_k$, $\mathcal{U}_{\text{node}}(\bar{\mathbf{e}}_i, \mathbf{v}_i, \mathbf{u}) \equiv \text{NN}(\bar{\mathbf{e}}_i)$,
 $\mathcal{S}_{\text{nodes}}(\{\mathbf{v}_i\}) \equiv \sum_i \text{softmax_layer}_l(\mathbf{v}_i)$ and $\mathcal{U}_{\text{global}}(\bar{\mathbf{e}}, \bar{\mathbf{v}}, \mathbf{u}) \equiv \mathbf{u} + \bar{\mathbf{v}}$

Publicly available code

- Neural graph fingerprints: <https://github.com/HIPS/neural-fingerprint/>
Python, autograd, no GPU support.
- Message passing neural networks (and many other graph neural networks):
<https://http://moleculenet.ai/>
Python, tensorflow.
- Gated graph neural network:
<https://github.com/Microsoft/gated-graph-neural-network-samples>
Python, tensorflow.
- Weisfeiler-Lehman network:
<https://github.com/wengong-jin/nips17-rexgen>
Python, tensorflow.

GPUs give about 4x speed up when working with small molecules (~ 20 atoms).

The speed up will increase when working with larger molecules.

Results neural graph fingerprints

Solubility dataset: aqueous solubility of 1144 molecules.

Drug efficacy: half-maximal effective concentration (EC 50) against malaria parasite.

Organic photovoltaics: DFT simulations of photovoltaic efficiency of molecules.

Dataset Units		Solubility [4] log Mol/L	Drug efficacy [5] EC ₅₀ in nM	Photovoltaic efficiency [8] percent
hand-engineered features {	Predict mean	4.29 ± 0.40	1.47 ± 0.07	6.40 ± 0.09
	Circular FPs + linear layer	1.71 ± 0.13	1.13 ± 0.03	2.63 ± 0.09
	Circular FPs + neural net	1.40 ± 0.13	1.36 ± 0.10	2.00 ± 0.09
	Neural FPs + linear layer	0.77 ± 0.11	1.15 ± 0.02	2.58 ± 0.18
	Neural FPs + neural net	0.52 ± 0.07	1.16 ± 0.03	1.43 ± 0.09

Slide source: Duvenaud et al. 2015

Results message passing neural networks

QM-9 dataset with 130462 molecules.

Targets: 13 properties approximated by quantum mechanical simulations (DFT).

Performance: ratio of MAE and estimate of chemical accuracy for target.

	hand engineered features					other graph-based methods			
Target	BAML	BOB	CM	ECFP4	HDAD	GC	GG-NN	DTNN	MPNN
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	-	0.30
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	-	0.92
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	1.17	-	0.99
LUMO	2.76	2.74	4.26	3.10	1.96	1.10	1.08	-	0.87
gap	3.28	3.41	5.32	3.86	2.49	1.78	1.70	-	1.60
R2	3.25	0.80	2.83	90.68	1.35	4.73	3.99	-	0.15
ZPVE	3.31	3.40	4.80	241.58	1.91	9.75	2.52	-	1.27
U0	1.21	1.43	2.98	85.01	0.58	3.02	0.83	-	0.45
U	1.22	1.44	2.99	85.59	0.59	3.16	0.86	-	0.45
H	1.22	1.44	2.99	86.21	0.59	3.19	0.81	-	0.39
G	1.20	1.42	2.97	78.36	0.59	2.95	0.78	.84 ²	0.44
Cv	1.64	1.83	2.36	30.29	0.88	1.45	1.19	-	0.80
Omega	0.27	0.35	1.32	1.47	0.34	0.32	0.53	-	0.19
Average	2.17	2.08	3.37	53.97	1.35	2.59	1.36	-	0.68

SMILES vs. GNNs

Datasets:

ZINC: 250,000 drug-like commercially available molecules from the ZINC database.

QM9: Subset of size 108,000 among molecules with 9 atoms (not counting hydrogens).

SMILES: 1D CNN. Strings padded with spaces up to length 120 for ZINC and 34 for QM9.

GNNs: similar to neural graph fingerprints by Duvenaud et al. 2015.

Results:

Database/Property	Mean	ECFP	GNNs	SMILES
ZINC250k/logP	1.14	0.38	0.05	0.16
ZINC250k/QED	0.112	0.045	0.017	0.041
QM9/HOMO, eV	0.44	0.20	0.12	0.12
QM9/LUMO, eV	1.05	0.20	0.15	0.11
QM9/Gap, eV	1.07	0.30	0.18	0.16

SMILES good on QM9 molecules, but much worse results on larger ZINC molecules!

Table source: Gómez-Bombarelli et al. 2017

Pros and cons of graph neural networks

Advantages:

- Invariant to order in which atom and bond information is provided.
- Data driven approach for generating graph features.
- Empirically, they seem to have very good predictive performance.

Pros and cons of graph neural networks

Advantages:

- Invariant to order in which atom and bond information is provided.
- Data driven approach for generating graph features.
- Empirically, they seem to have very good predictive performance.

Disadvantages:

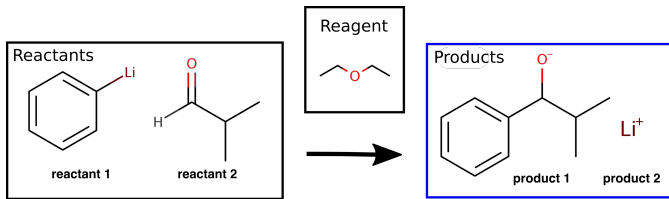
- Higher computational cost than other neural network models.
- Propagation of local information somewhat limited by depth of network.
- Set functions prevent us from knowing the individual contributions of nodes or edges in the update functions.

Part II: Reaction prediction

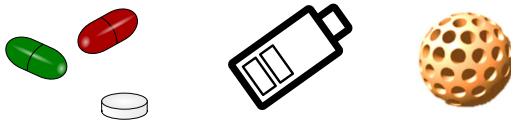
Definition and motivation

Chemical reaction: a process by which a set of **input molecules** called **reactants or reagents** is transformed into another set of **output molecules** called **products**.

The reaction changes positions of electrons, forming and breaking bonds between atoms.



Predicting products from input reactants and reagents is key to automate the fabrication of new medicines, energy capturing devices, nanomaterials, etc.

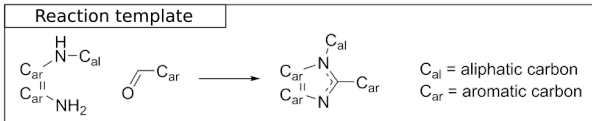
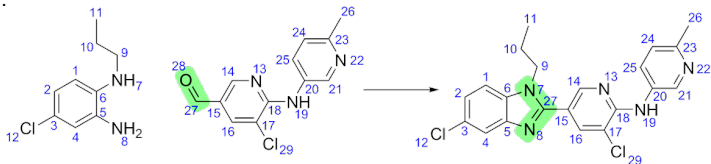



Reaction templates

They specify a molecular subgraph pattern and a corresponding graph transformation.

Can be generated automatically from reaction databases.

Example:



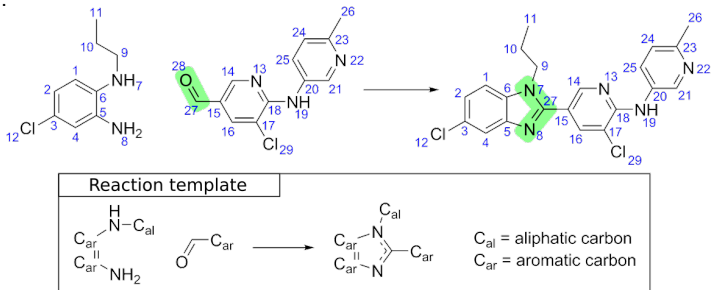
 Bonds that change during the reaction (**reaction center**)


Reaction templates

They specify a molecular subgraph pattern and a corresponding graph transformation.

Can be generated automatically from reaction databases.

Example:



 Bonds that change during the reaction (**reaction center**)

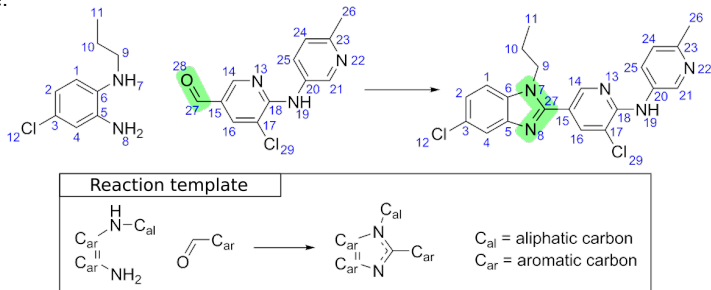
Since multiple templates can result in a match, another supervised learning method is used to filter candidate products. Reaction templates are **computationally expensive**.

Reaction templates

They specify a molecular subgraph pattern and a corresponding graph transformation.

Can be generated automatically from reaction databases.

Example:



Bonds that change during the reaction (**reaction center**)

Since multiple templates can result in a match, another supervised learning method is used to filter candidate products. Reaction templates are **computationally expensive**.

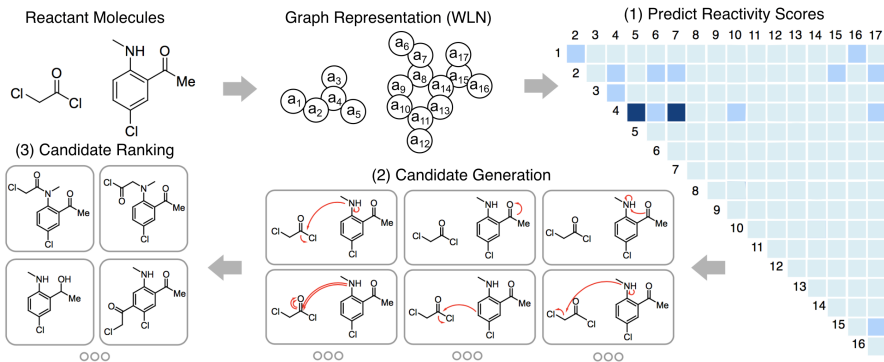
Templates **fail to take into account context** far away from reaction center.

Reaction prediction using GNNs

Jin et al. 2017 use GNNs to predict the **reaction center** (RC), that is, **the set of nodes and edges where graph edits occur**.

The probability of the bond between atoms i and j belonging to the RC is $\sigma(\text{NN}(\mathbf{v}_i, \mathbf{v}_j))$ where \mathbf{v}_i and \mathbf{v}_j are node features learned by a WLN and σ is the sigmoid function.

Top-K bonds selected and all possible products generated and then ranked by another WLN.



Results of GNNs in reaction prediction

Data:

USPTO: 480K chemical reactions extracted from the US patent database.

Atoms have unique ID to easily match them before and after the reaction.

Baseline: **template based approach** described by Coley et al. 2017.

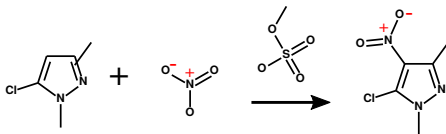
Accuracy in product identification				
USPTO-15K				
Method	Cov.	P@1	P@3	P@5
Coley et al.	100.0	72.1	86.6	90.7
WLN	90.1	76.7	85.6	86.8

Human and model performance on 80 reactions randomly selected:

Chemist	56.3	50.0	72.0	63.8	66.3	65.0	40.0	58.8	25.0	16.3
WLN	69.1									

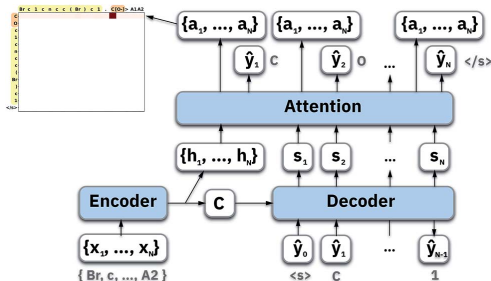
SMILES based approach for reaction prediction

Schwaller et al. 2018 use a **seq2seq model** to solve the reaction prediction problem.



Input sequence: Cc1cc(Cl)n(C)n1 . O=[N+](O)O > O=S(=O)(O)O

Target sequence: Cc1nn(C)c(Cl)c1[N+](=O)[O-]



Based on existing technology for **language translation** problems.

The model is formed by two LSTMs (encoder and decoder) using an attention mechanism.

Results seq2seq method in reaction prediction

Architecture and hyperparameters selected by a **Bayesian optimization** method.

Beam search of width 10 is used for selecting most probable decoding sequences.

Baseline: WLN method described by Jin et al. 2017 and based on **reaction centers**.

Evaluation on the full **USPTO** dataset:

Jin's USPTO test set, ²⁰ accuracies in [%]				
Method	Top-1	Top-2	Top-3	Top-5
WLN ²⁰	79.6		87.7	89.2
Our model	80.3	84.7	86.2	87.5

Limitations of WLN and seq2seq methods

WLN:

- Jin et al. 2017 assumes independence of bonds in reaction center (RC).
- Inefficient two stage training process: RC prediction and product ranking.

seq2seq:

- Same as SMILES-based machine learning methods.
- Sampled sequences are not guaranteed to be valid SMILES strings.

Take home messages

Machine learning (ML) can **accelerate and automate** the molecule discovery process.

Molecules are different from typical data and create their **own challenges for ML**.

Molecule representations...

- 1 Fingerprints are **fast** and **accurate** but **handcrafted** and **not data dependent**.
- 2 SMILES enable **NLP methods** but **lack invariance** and create **long-range dependencies**.
- 3 Graph neural networks are a **state-of-the-art** method with few limitations.

Reaction prediction...

- 1 Machine learning methods can achieve very **good** predictive **performance**.
- 2 Some of the best performing methods are based on **SMILES** or **GNNs** encodings.
- 3 But they still have **limitations**.

References I

- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. and Dahl, G. E. (2017, July). Neural Message Passing for Quantum Chemistry. In International Conference on Machine Learning (pp. 1263-1272).
- Sanchez-Lengeling, Benjamin, and Alán Aspuru-Guzik. "Inverse molecular design using machine learning: Generative models for matter engineering." *Science* 361.6400 (2018): 360-365.
- Rogers, David, and Mathew Hahn. "Extended-connectivity fingerprints." *Journal of chemical information and modeling* 50.5 (2010): 742-754.
- Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 729734. IEEE.
- Scarselli, F., Yong, S. L., Gori, M., Hagenbuchner, M., Tsoi, A. C., and Maggini, M. (2005). Graph neural networks for ranking web pages. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 666672. IEEE.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009b). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):6180.
- Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2014). Spectral networks and locally connected networks on graphs. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

References II

- Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224-2232.
- Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2016). Gated graph sequence neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595-608.
- Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., and Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8, 13890.
- Jin, W., Coley, C., Barzilay, R., and Jaakkola, T. (2017). Predicting organic reaction outcomes with weisfeiler-lehman network. In *Advances in Neural Information Processing Systems* (pp. 2607-2616).
- Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, Gulcehre C. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*. 2018 Jun 4.

References III

- Vinyals, Oriol, Samy Bengio, and Manjunath Kudlur. "Order matters: Sequence to sequence for sets." arXiv preprint arXiv:1511.06391 (2015).
- Gómez-Bombarelli R., Wei J., Duvenaud D., Hernández-Lobato J. M., Sánchez-Lengeling B., Sheberla D., Aguilera-Iparraguirre J., Hirzel T., Adam R. P. and Aspuru-Guzik A. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules, ACS Central Science, 2018
- Jiang, Bo, Cuiling Li, Ömer Dag, Hideki Abe, Toshiaki Takei, Tsubasa Imai, Md Shahriar A. Hossain et al. "Mesoporous metallic rhodium nanoparticles." Nature communications 8 (2017): 15581.
- Coley, C.W., Barzilay, R., Jaakkola, T.S., Green, W.H. and Jensen, K.F., 2017. Prediction of organic reaction outcomes using machine learning. ACS central science, 3(5), pp.434-443.
- Schwaller, P., Gaudin, T., Lanyi, D., Bekas, C. and Laino, T. (2018). Found in Translation: predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. Chemical science, 9(28), 6091-6098.

Thanks!