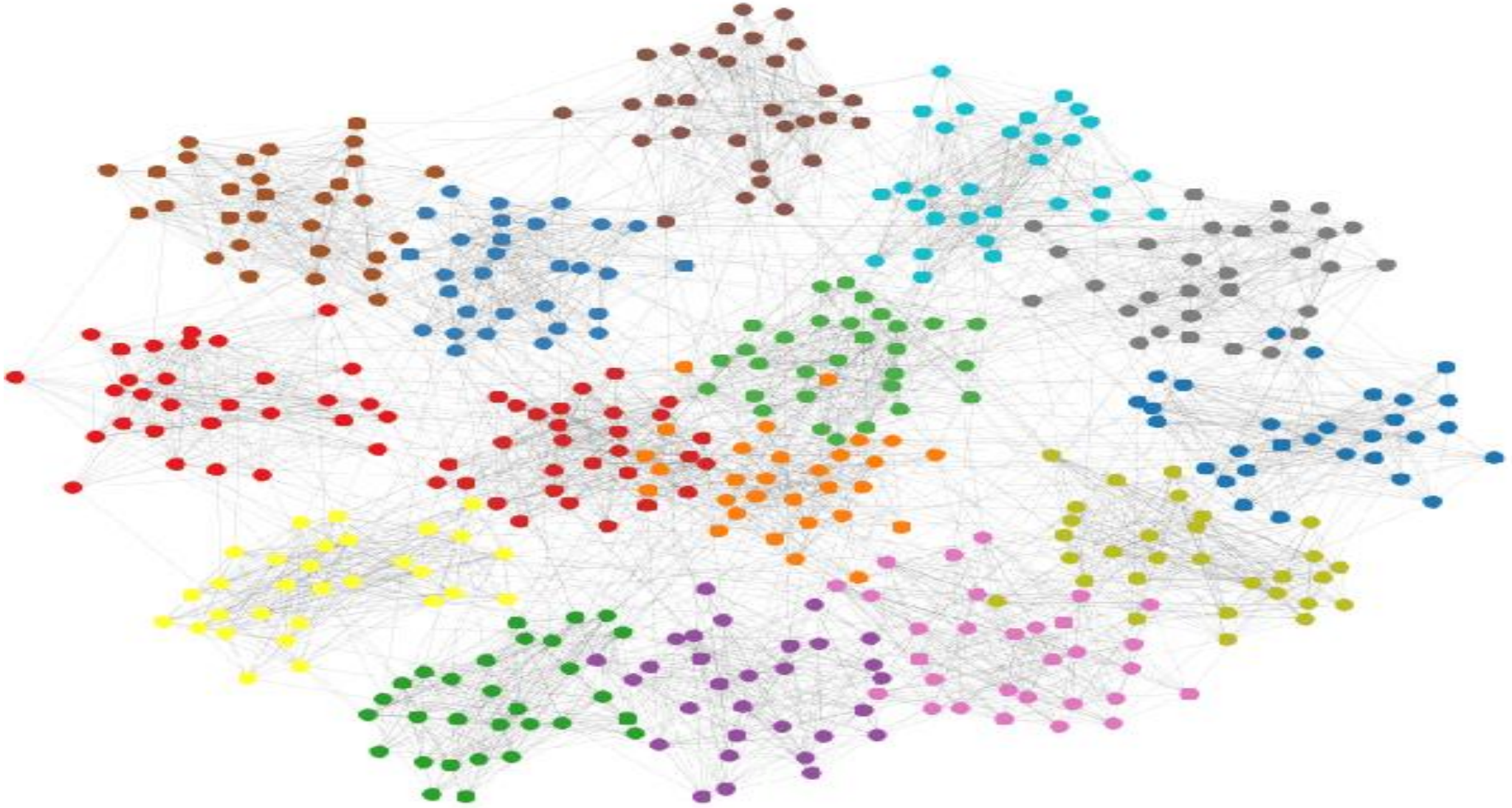


A gentle introduction to graph neural networks

Seongok Ryu, Department of Chemistry @ KAIST

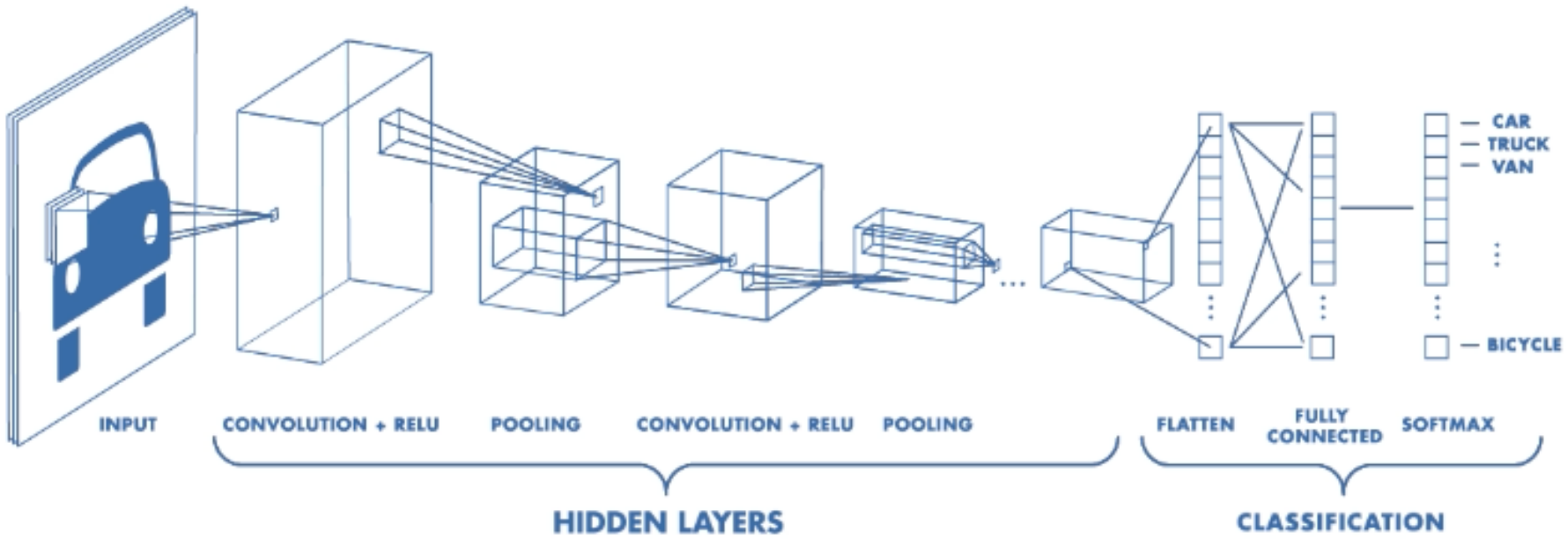
- **Motivation**
- **Graph Neural Networks**
- **Applications of Graph Neural Networks**

Motivation

Non-Euclidean data structure

Successful deep learning architectures

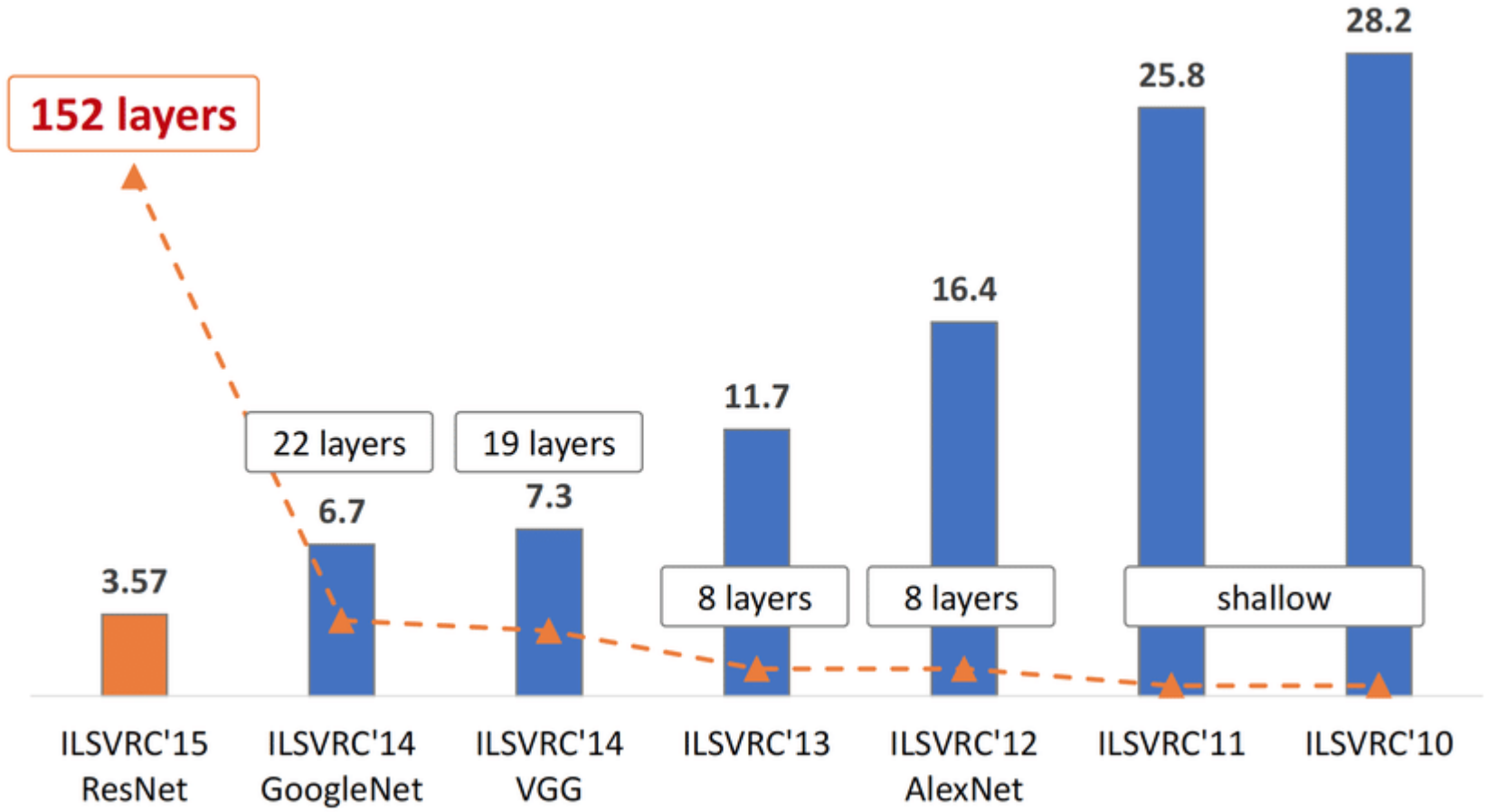
1. Convolutional neural networks



Non-Euclidean data structure

Successful deep learning architectures

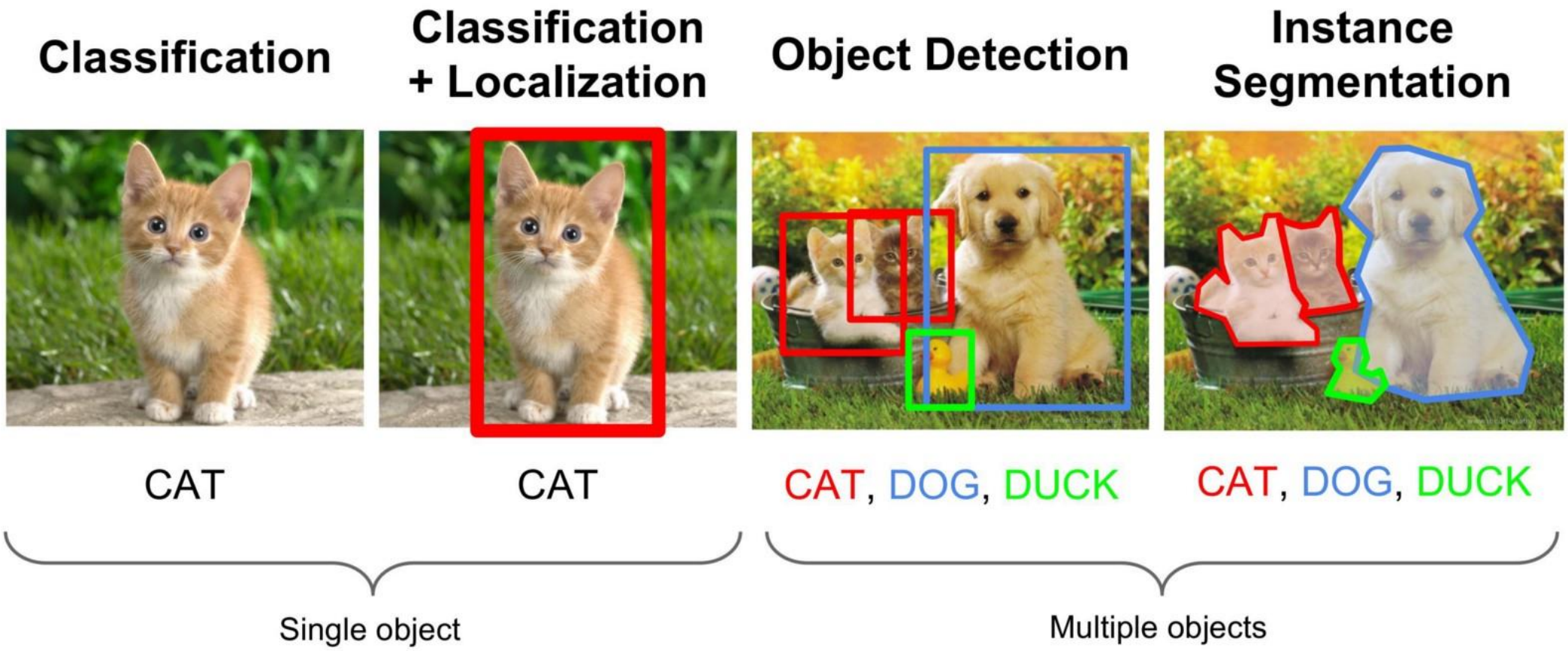
1. Convolutional neural networks



Non-Euclidean data structure

Successful deep learning architectures

1. Convolutional neural networks



https://medium.com/comet-app/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852

Non-Euclidean data structure

Successful deep learning architectures

1. Convolutional neural networks

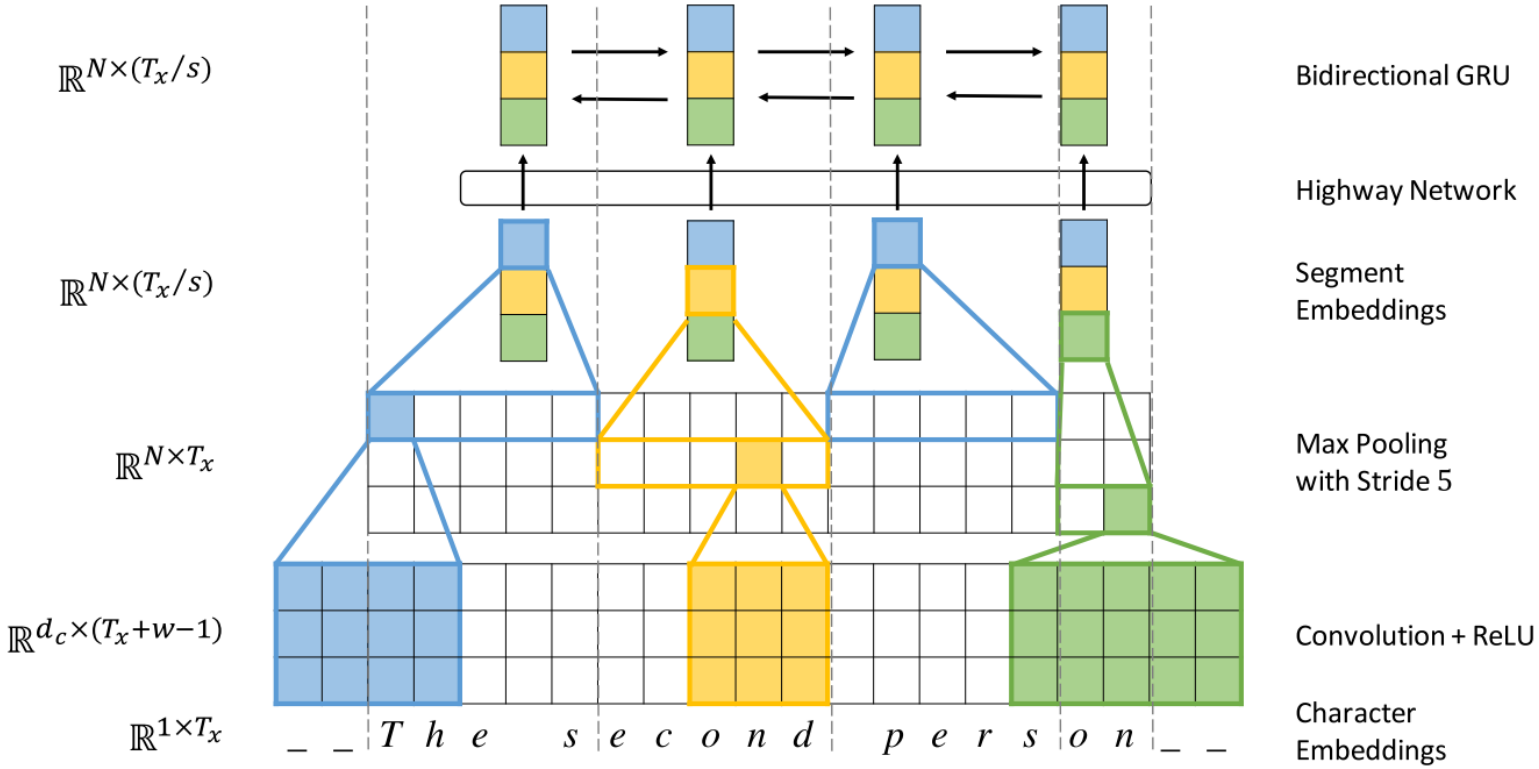


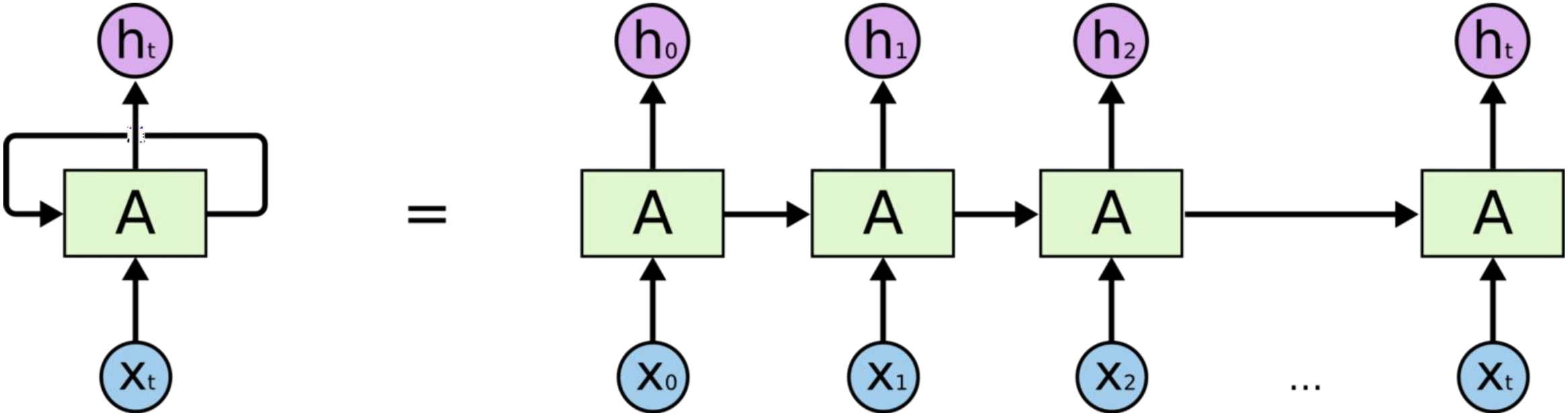
Figure 1: Encoder architecture schematics. Underscore denotes padding. A dotted vertical line delimits each segment.

Lee, Jason, Kyunghyun Cho, and Thomas Hofmann. "Fully character-level neural machine translation without explicit segmentation." *arXiv preprint arXiv:1610.03017* (2016).

Non-Euclidean data structure

Successful deep learning architectures

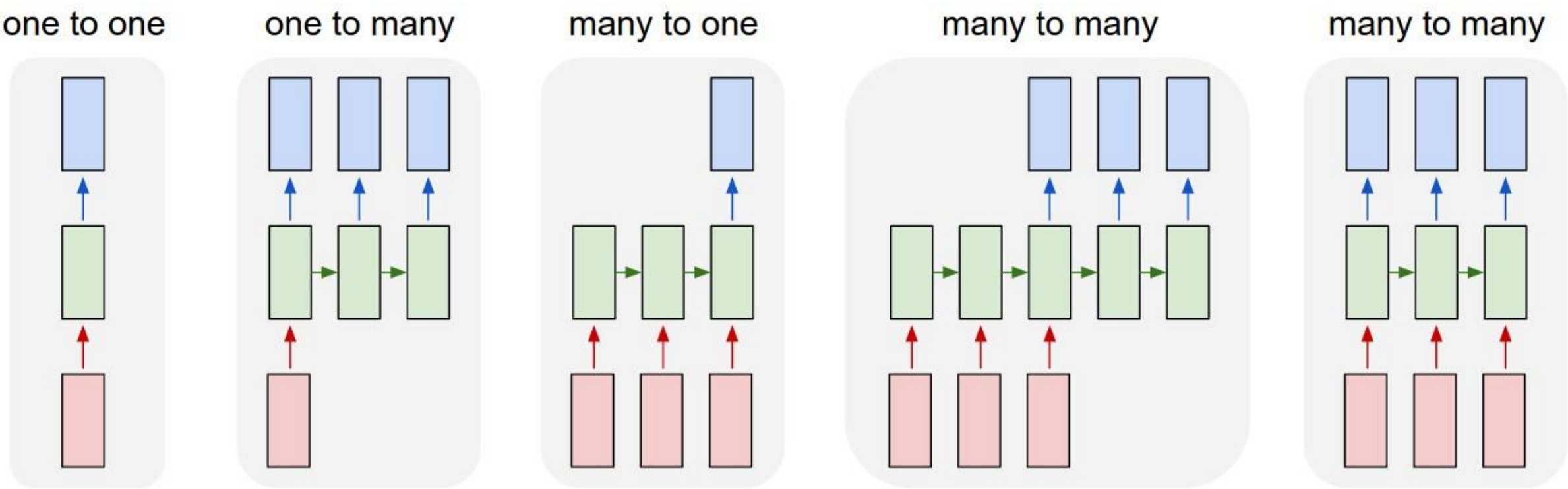
2. Recurrent neural network



Non-Euclidean data structure

Successful deep learning architectures

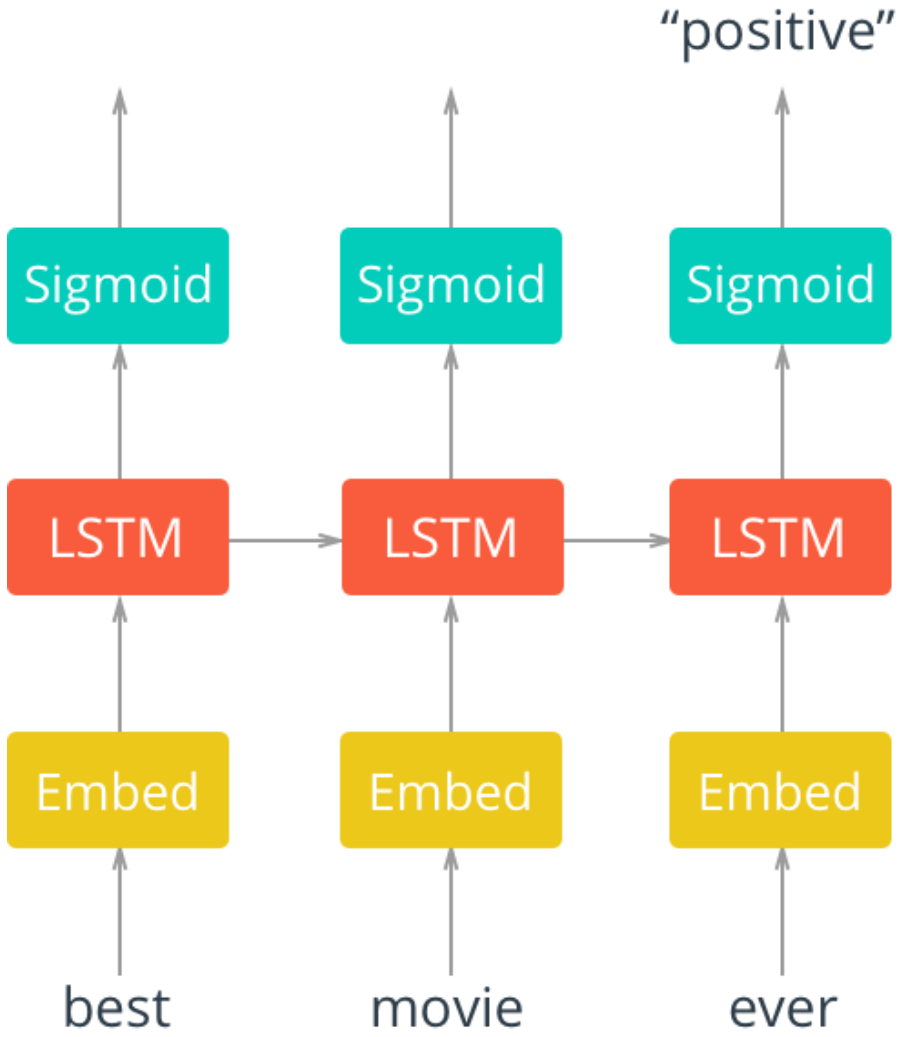
2. Recurrent neural network



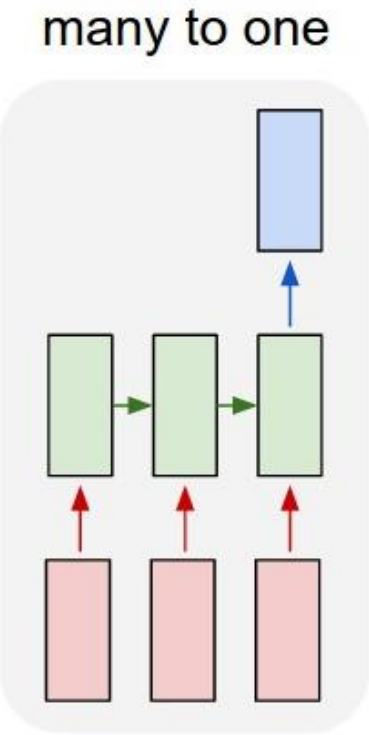
Non-Euclidean data structure

Successful deep learning architectures

2. Recurrent neural network



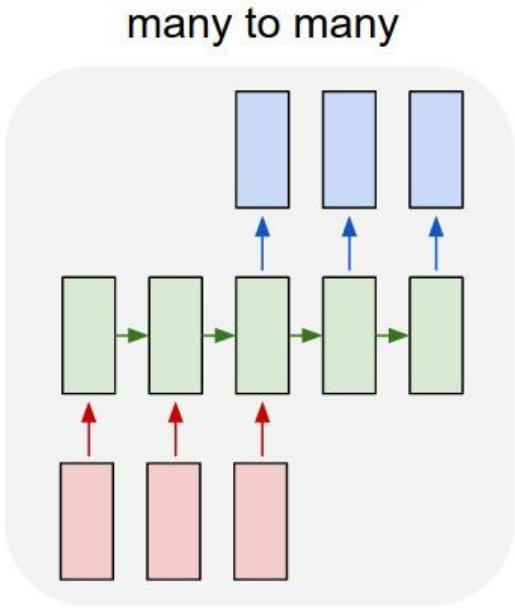
Sentiment analysis

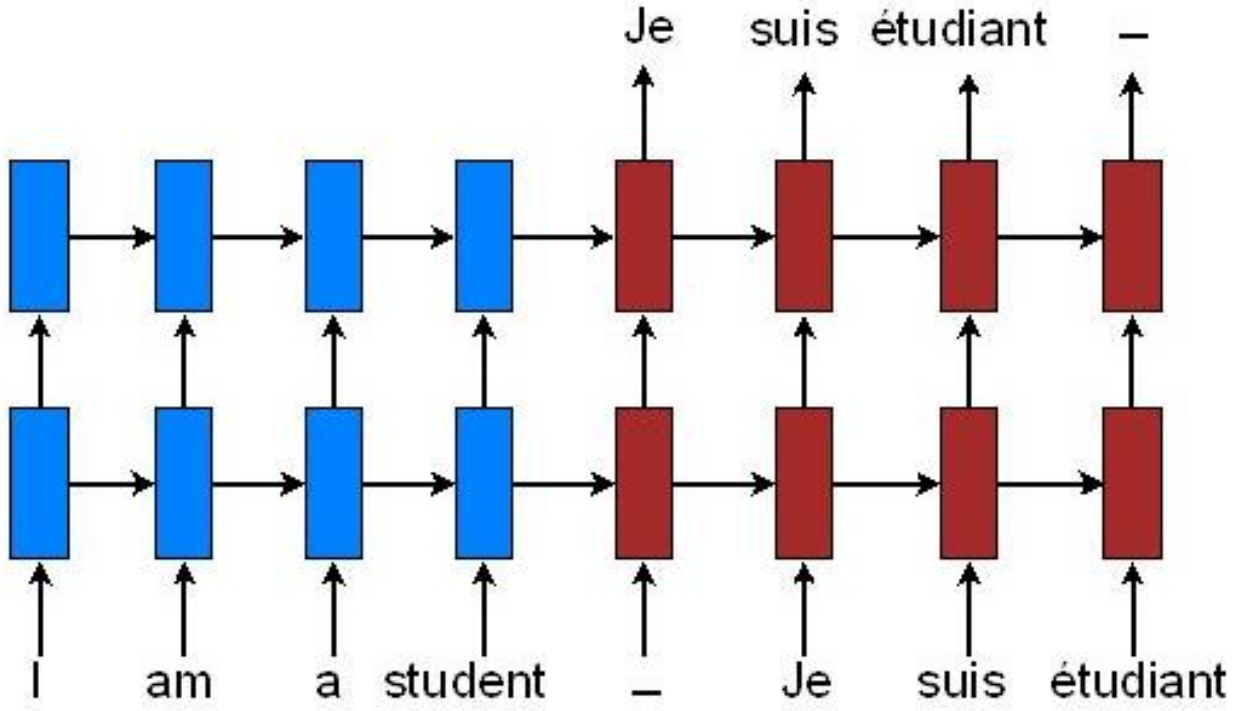


Non-Euclidean data structure

Successful deep learning architectures

2. Recurrent neural network





Neural machine translation

Non-Euclidean data structure

Data structures of shown examples are regular.

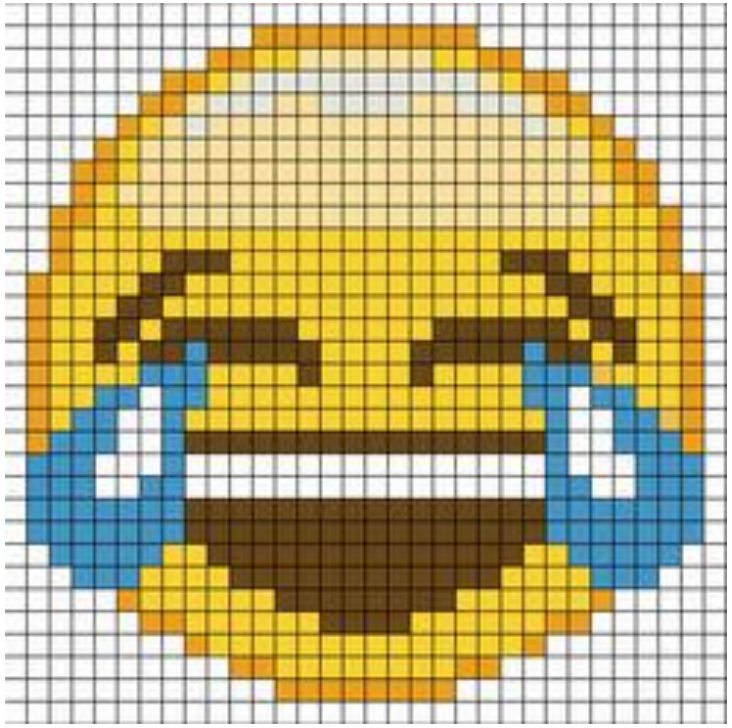
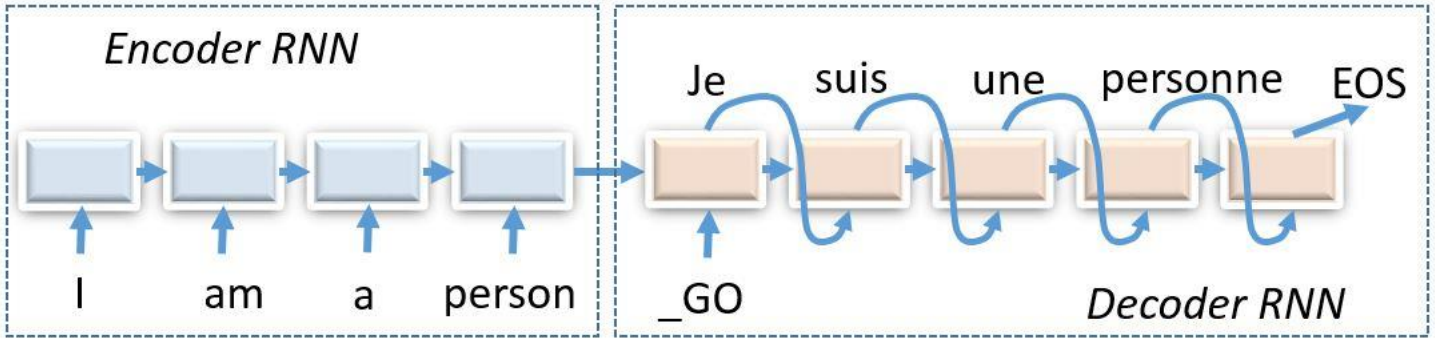


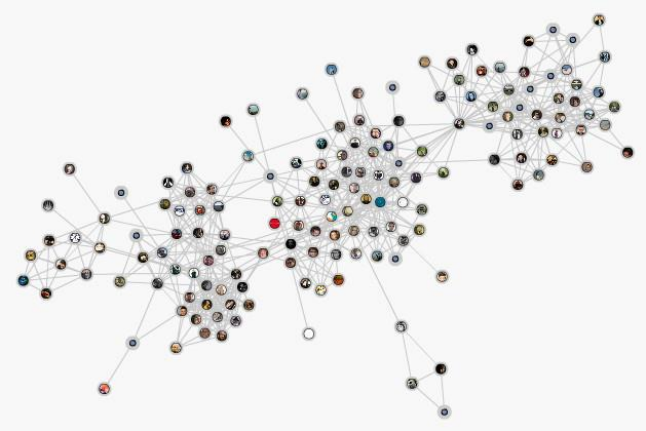
Image – values on pixels (grids)



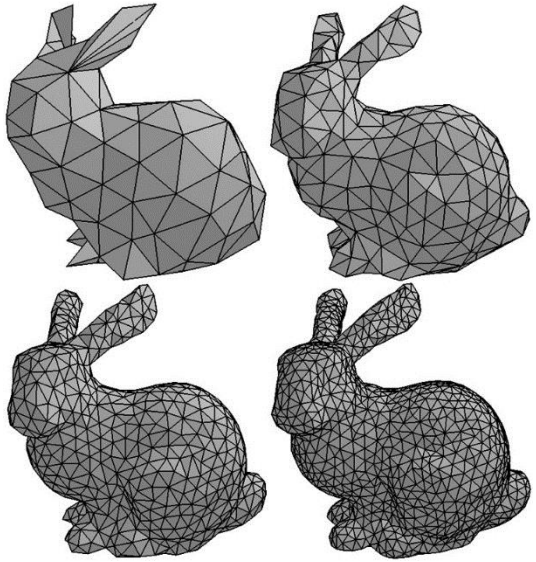
Sentence – sequential structure

Non-Euclidean data structure

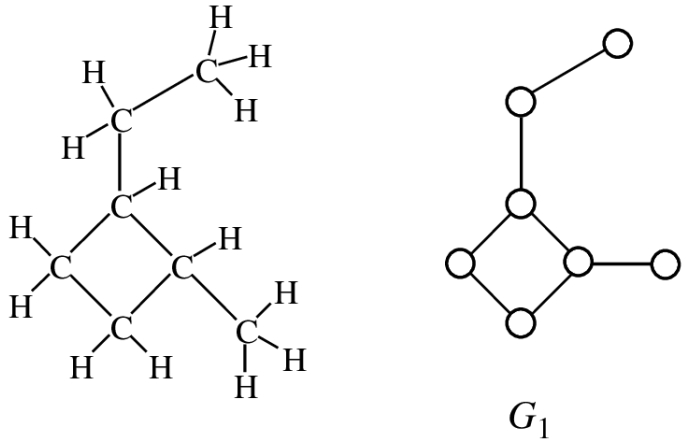
HOWEVER, there are lots of irregular data structure, …



Social Graph
(Facebook, Wikipedia)



3D Mesh



Molecular Graph

All you need is GRAPH!

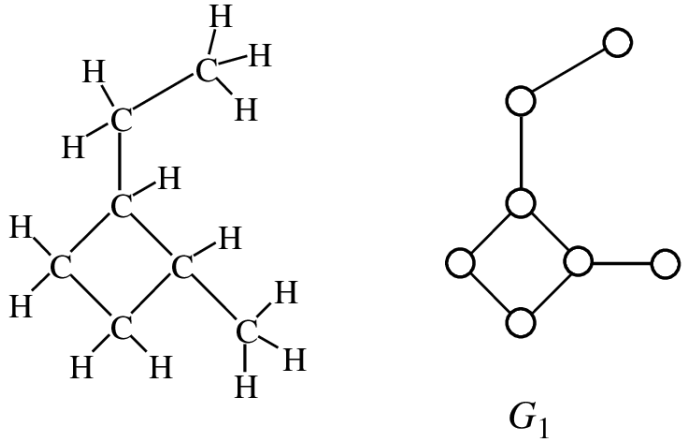
Graph structure

$$\boldsymbol{Graph = G(X, A)}$$

$\boldsymbol{X}$: Node, Vertex

- Individual person in a social network
- Atoms in a molecule

Represent elements of a system

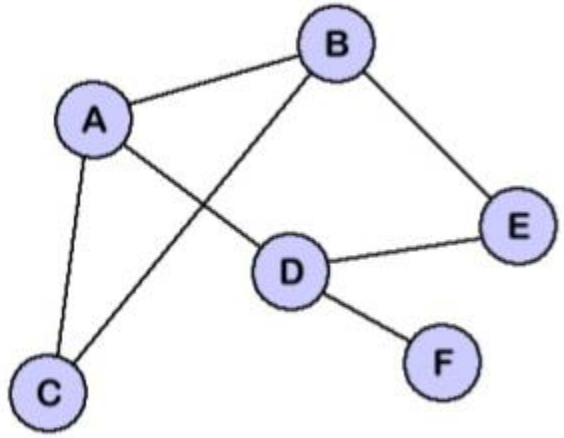


Graph structure

$$\boldsymbol{Graph = G(X, A)}$$

$\boldsymbol{A}$: Adjacency matrix

- Edges of a graph
- Connectivity, Relationship



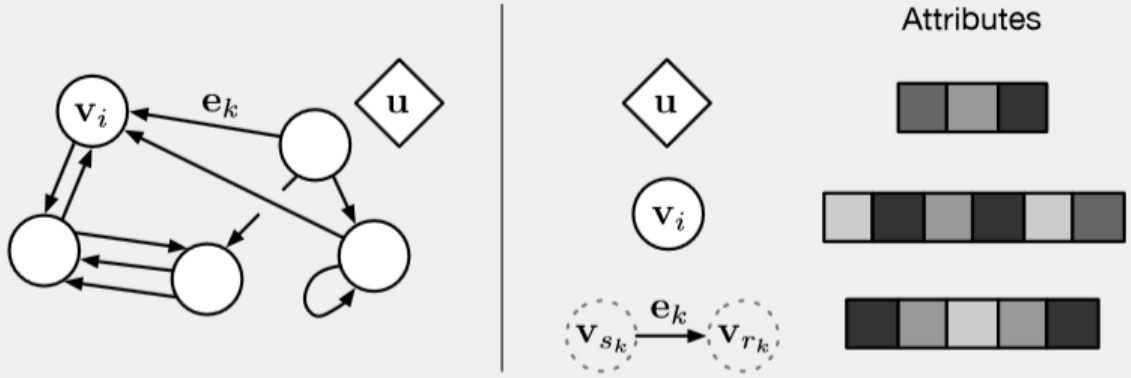
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Represent relationship or interaction between elements of the system

Graph structure

More detail, ...

Box 3: Our definition of "graph"



Here we use "graph" to mean a directed, attributed multi-graph with a global attribute. In our terminology, a node is denoted as $\mathbf{v}_i$, an edge as $\mathbf{e}_k$, and the global attributes as $\mathbf{u}$. We also use s_k and r_k to indicate the indices of the sender and receiver nodes (see below), respectively, for edge k. To be more precise, we define these terms as:

Directed : one-way edges, from a "sender" node to a "receiver" node.
Attribute : properties that can be encoded as a vector, set, or even another graph.
Attributed : edges and vertices have attributes associated with them.
Global attribute : a graph-level attribute.
Multi-graph : there can be more than one edge between vertices, including self-edges.

Figure 2 shows a variety of different types of graphs corresponding to real data that we may be interested in modeling, including physical systems, molecules, images, and text.

Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261* (2018).

Graph structure

Node features

Ariana Grande

- 25 years old
- Female
- American
- Singer
- …

Donald Trump

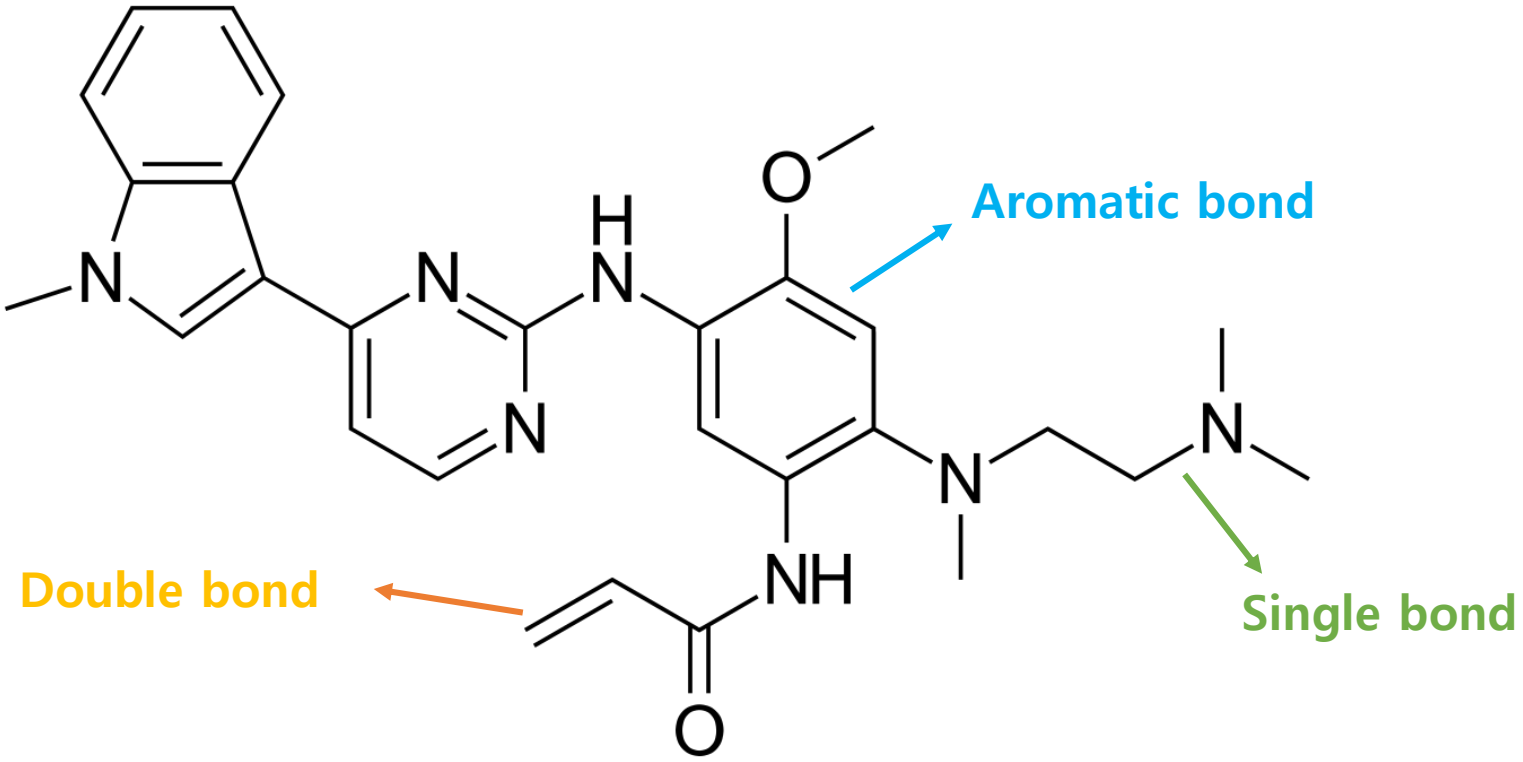
- 72 years old
- Male
- American
- President, business man
- …

Vladimir Putin

- 65 years old
- Male
- Russian
- President
- …

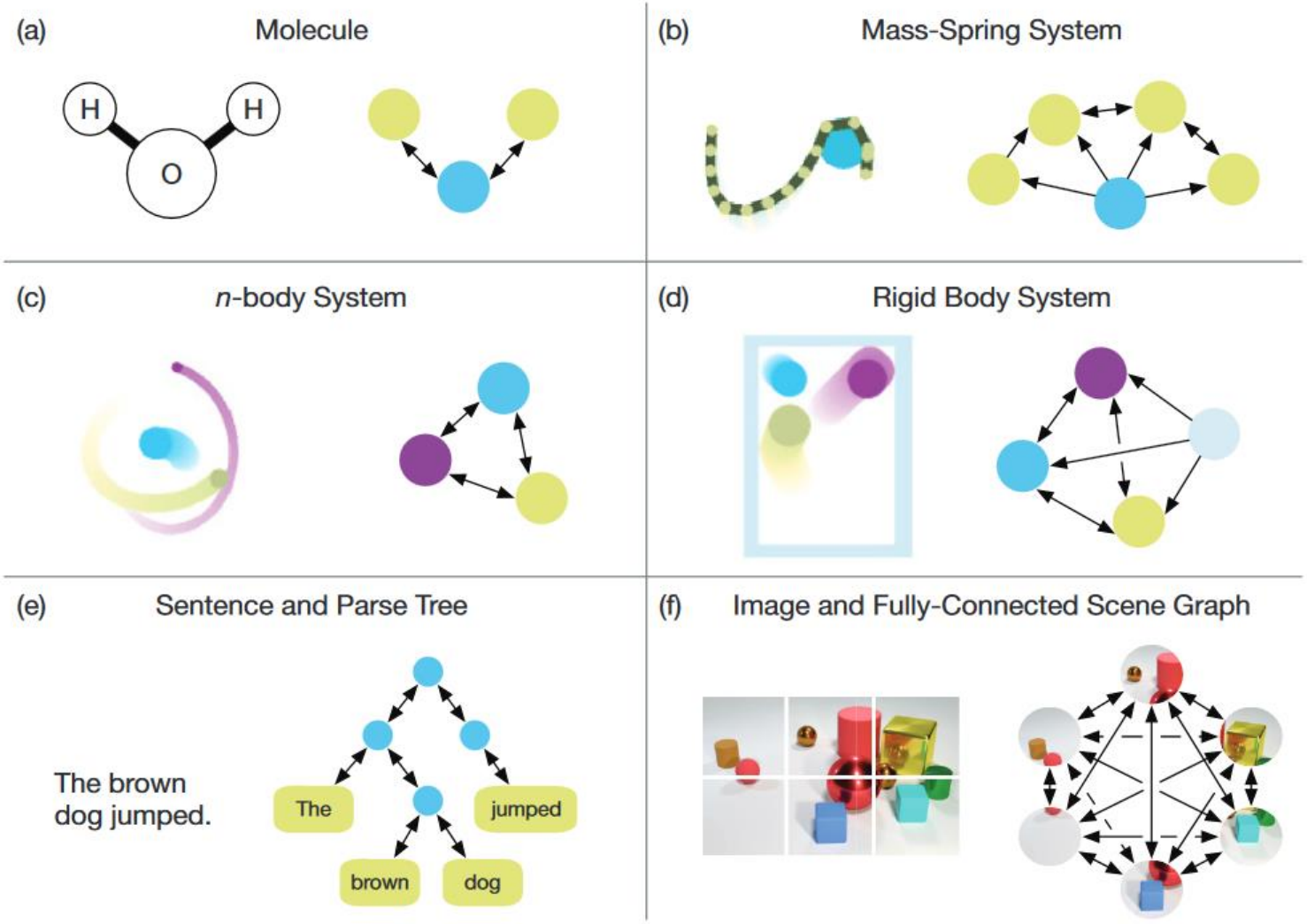
Graph structure

Edge features



Learning relation and interaction

What can we do with graph neural networks?



Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261* (2018).

Learning relation and interaction

What can we do with graph neural networks?

- **Node classification**
- **Link prediction**
- **Node2Vec, Subgraph2Vec, Graph2Vec**

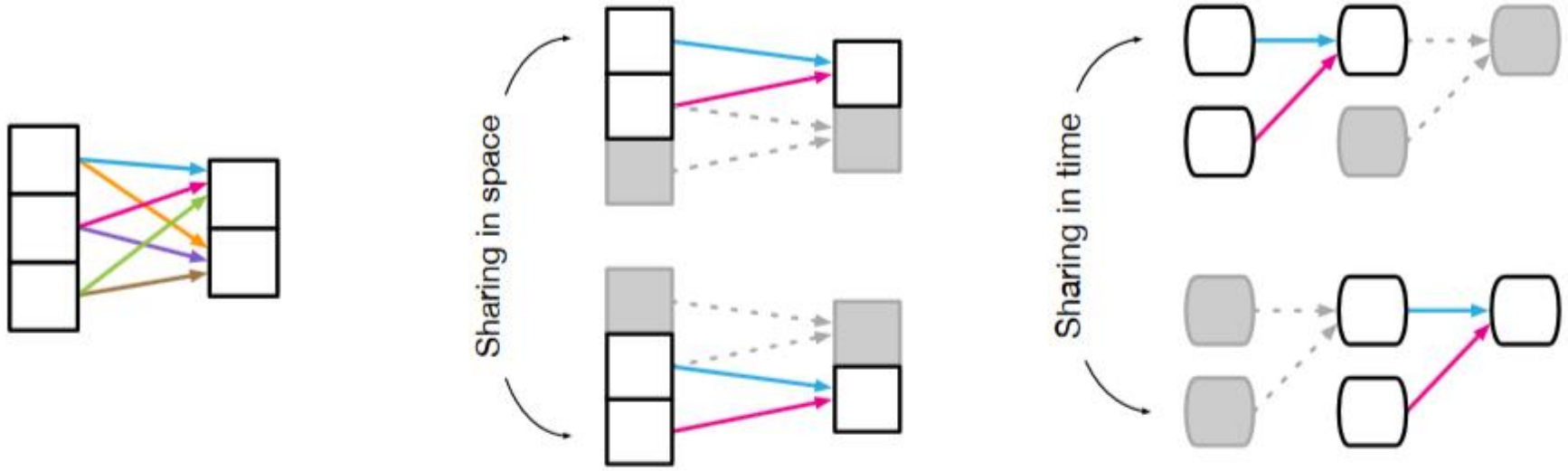
  : Embedding node/substructure/graph structure to a vector
- **Learning physics law from data**
- **And you can do more amazing things with GNN!**

Graph neural networks

- **Overall architecture of graph neural networks**
- **Updating node states**
  - Graph Convolutional Network (GCN)
  - Graph Attention Network (GAT)
  - Gated Graph Neural Network (GGNN)
- **Readout : permutation invariance on changing node orders**
- **Graph Auto-Encoders**
- **Practical issues**
  - Skip connection
  - Inception
  - Dropout

Principles of graph neural network

Weights using in updating hidden states of fully-connected Net, CNN and RNN

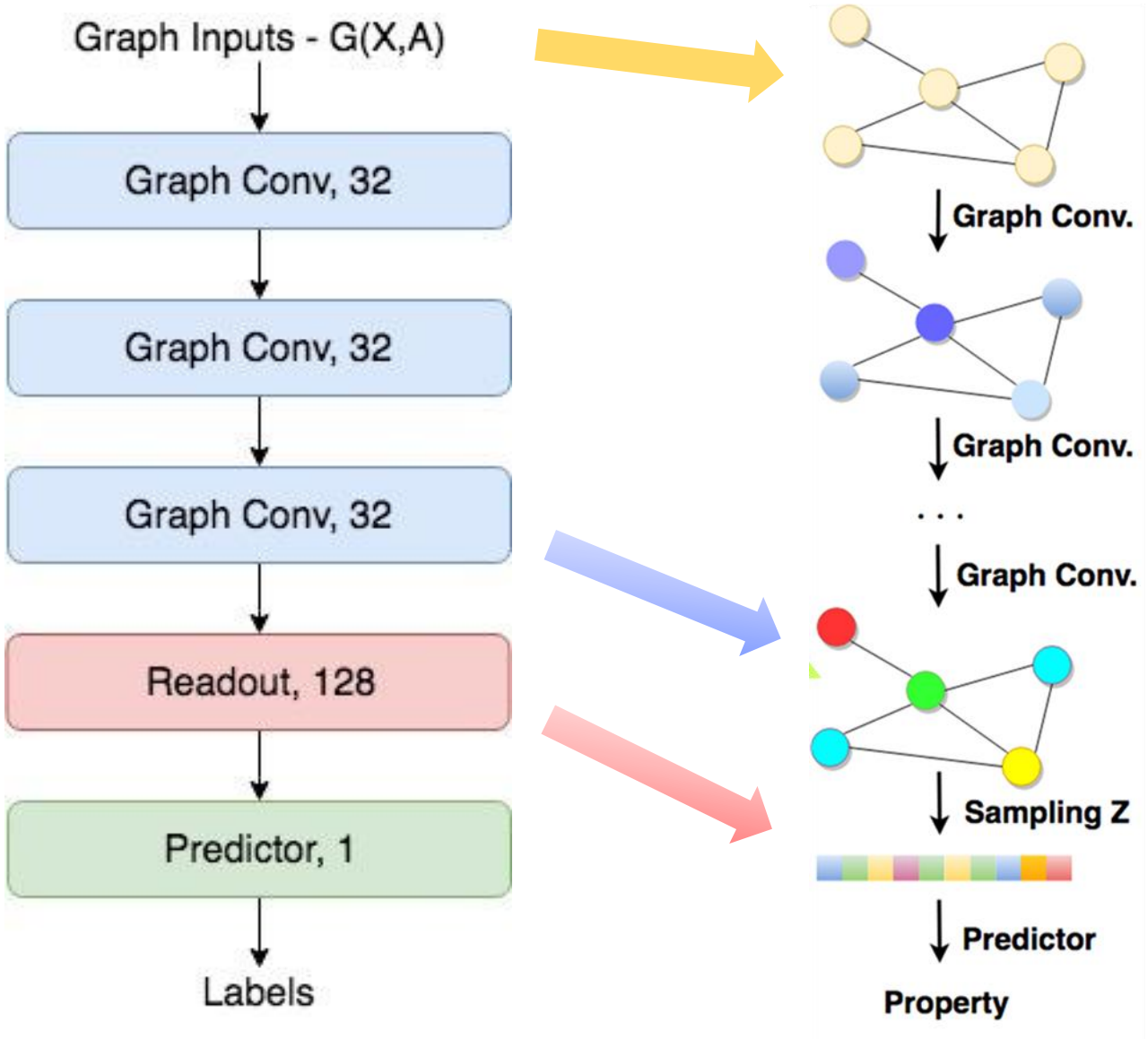


(a) Fully connected (b) Convolutional (c) Recurrent

Figure 1: Reuse and sharing in common deep learning building blocks. (a) Fully connected layer, in which all weights are independent, and there is no sharing. (b) Convolutional layer, in which a local kernel function is reused multiple times across the input. Shared weights are indicated by arrows with the same color. (c) Recurrent layer, in which the same function is reused across different processing steps.

Battaglia, Peter W., et al.
"Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261* (2018).

Overall neural network structure – case of supervised learning



Input node features, $\left\{H_i^{(0)}\right\}$

: Raw node information

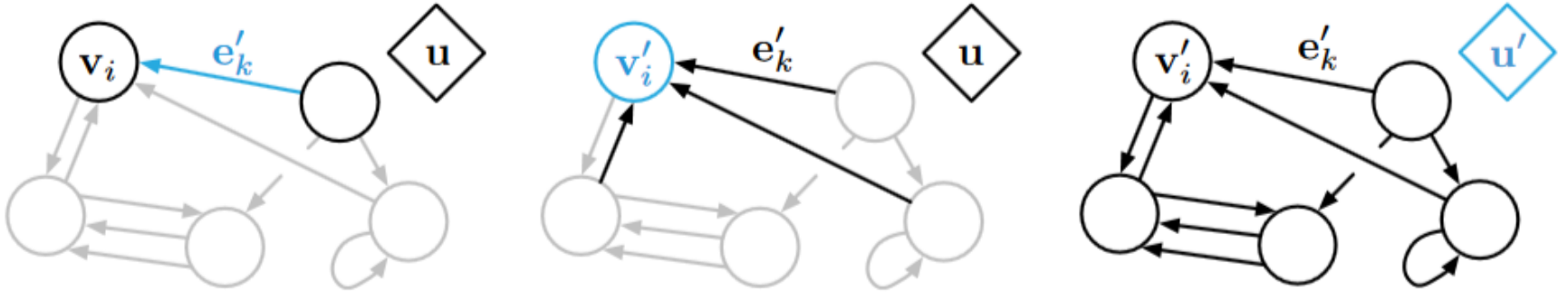
Final node states, $\left\{H_i^{(L)}\right\}$

: How the NN recognizes the nodes

Graph features, Z

Principles of graph neural network

Updates in a graph neural network



(a) Edge update (b) Node update (c) Global update

Figure 3: Updates in a GN block. Blue indicates the element that is being updated, and black indicates other elements which are involved in the update (note that the pre-update value of the blue element is also used in the update). See Equation 1 for details on the notation.

- **Edge update** : relationship or interactions, sometimes called as 'message passing'

  ex) the forces of spring

- **Node update** : aggregates the edge updates and used in the node update

  ex) the forces acting on the ball

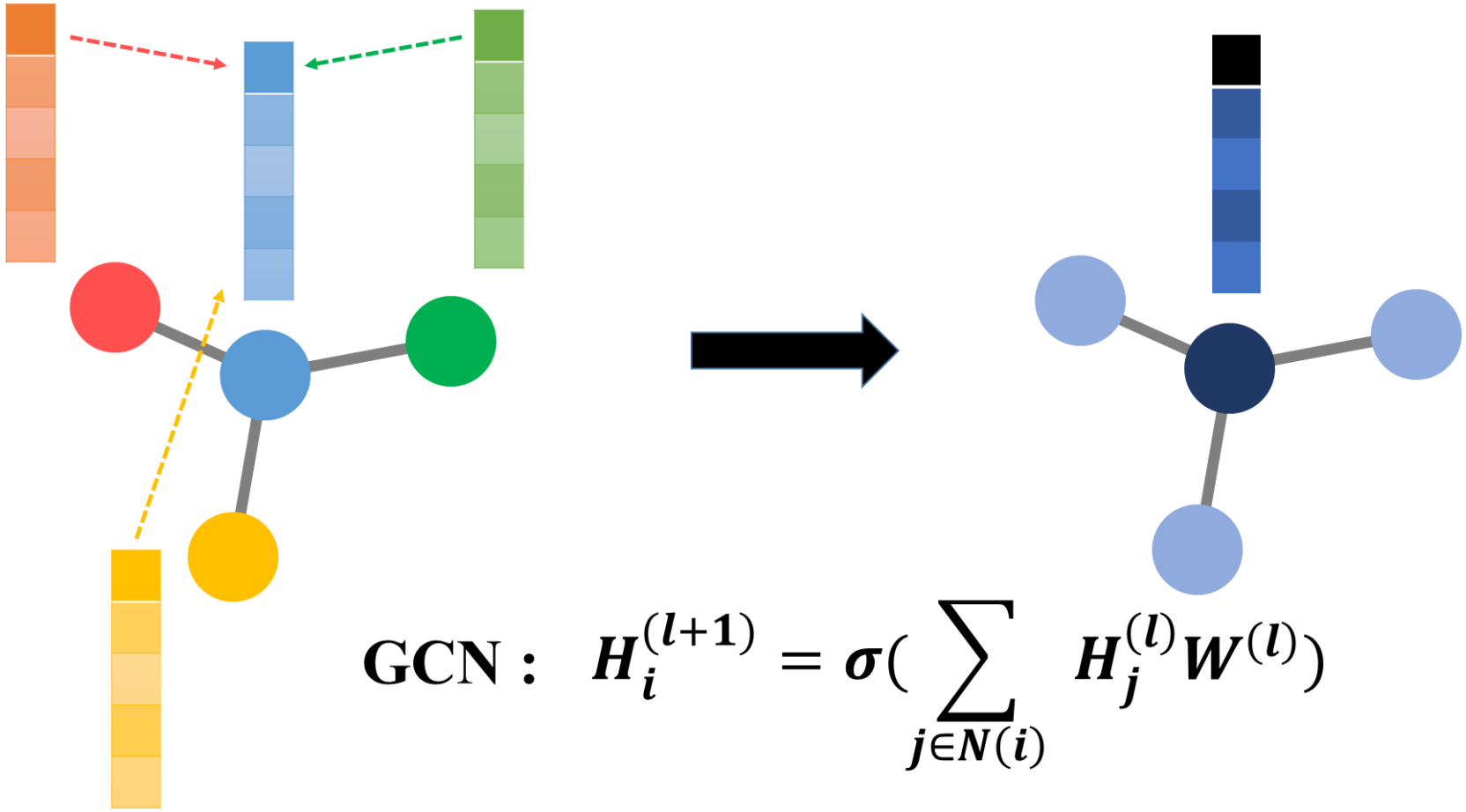
- **Global update** : an update for the global attribute

  ex) the net forces and total energy of the physical system

Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261* (2018).

Principles of graph neural network

Weights using in updating hidden states of GNN



$$\text{GCN}: \quad H_i^{(l+1)} = \sigma\Big(\sum_{j \in N(i)} H_j^{(l)} W^{(l)}\Big)$$

Sharing weights for all nodes in graph,
but nodes are differently updated by reflecting individual node features, $H_j^{(l)}$

GCN : Graph Convolutional Network

Published as a conference paper at ICLR 2017

SEMI-SUPERVISED CLASSIFICATION WITH GRAPH CONVOLUTIONAL NETWORKS

Thomas N. Kipf
University of Amsterdam
T.N.Kipf@uva.nl

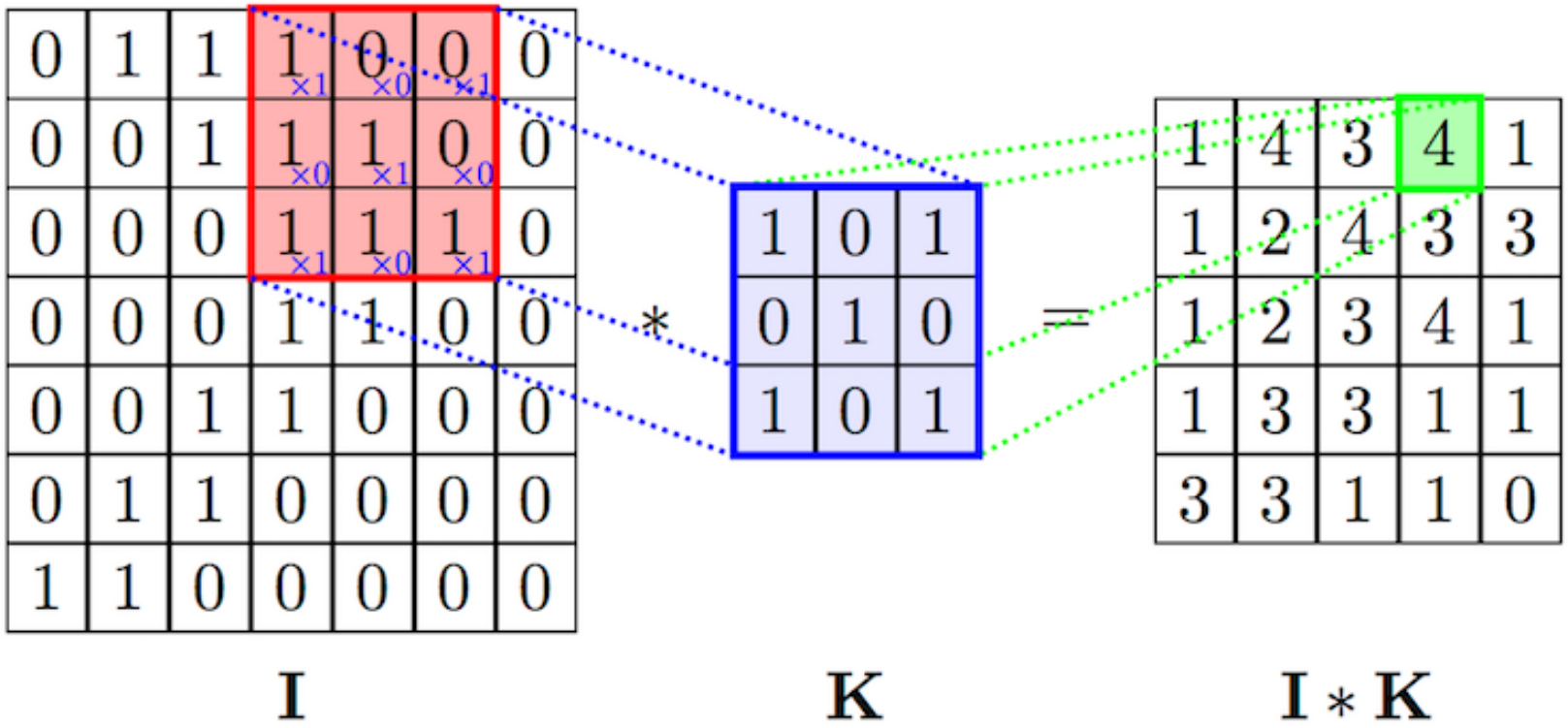
Max Welling
University of Amsterdam
Canadian Institute for Advanced Research (CIFAR)
M.Welling@uva.nl

http://tkipf.github.io/

Famous for variational autoencoder (VAE)

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)

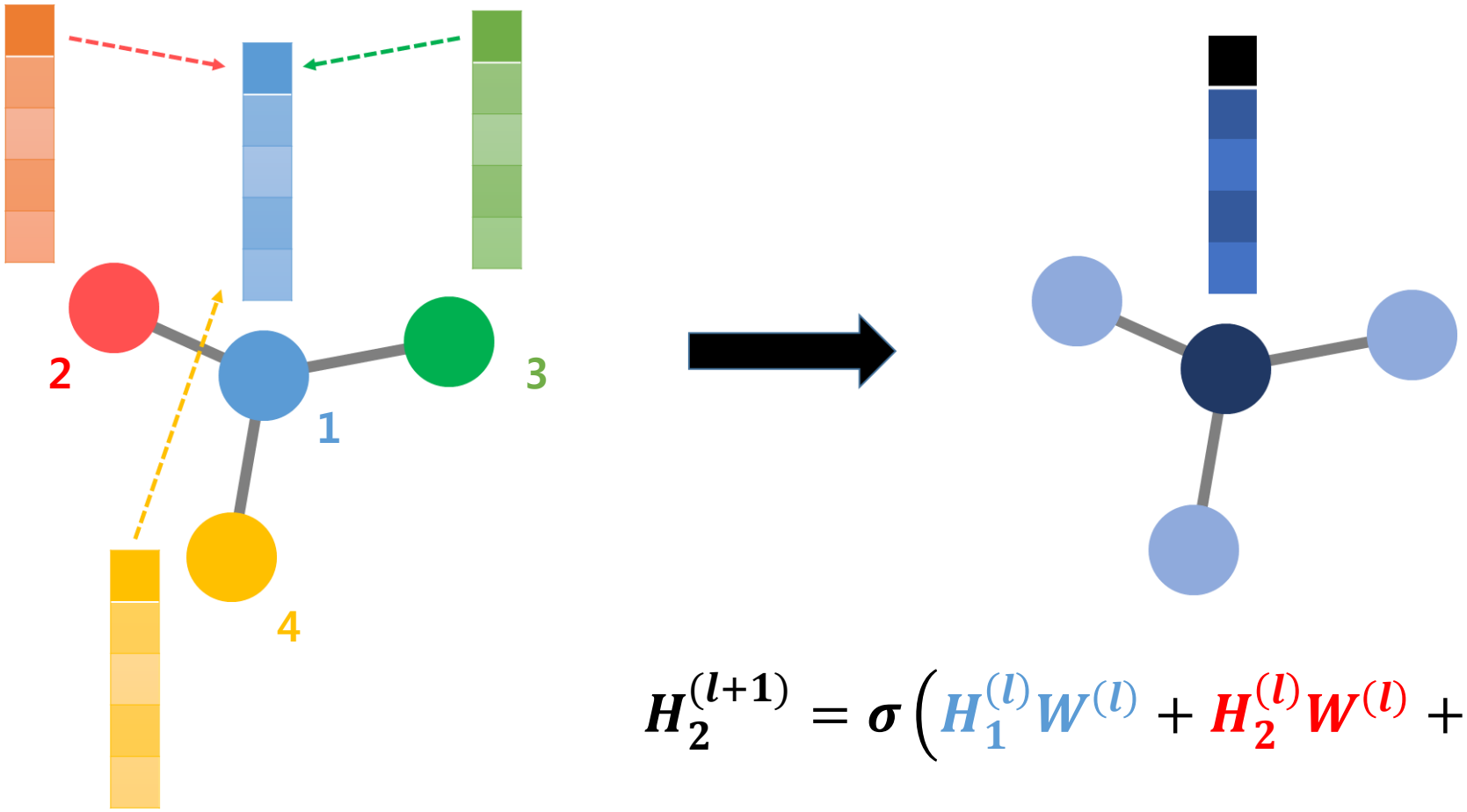
GCN : Graph Convolutional Network



What NN learns

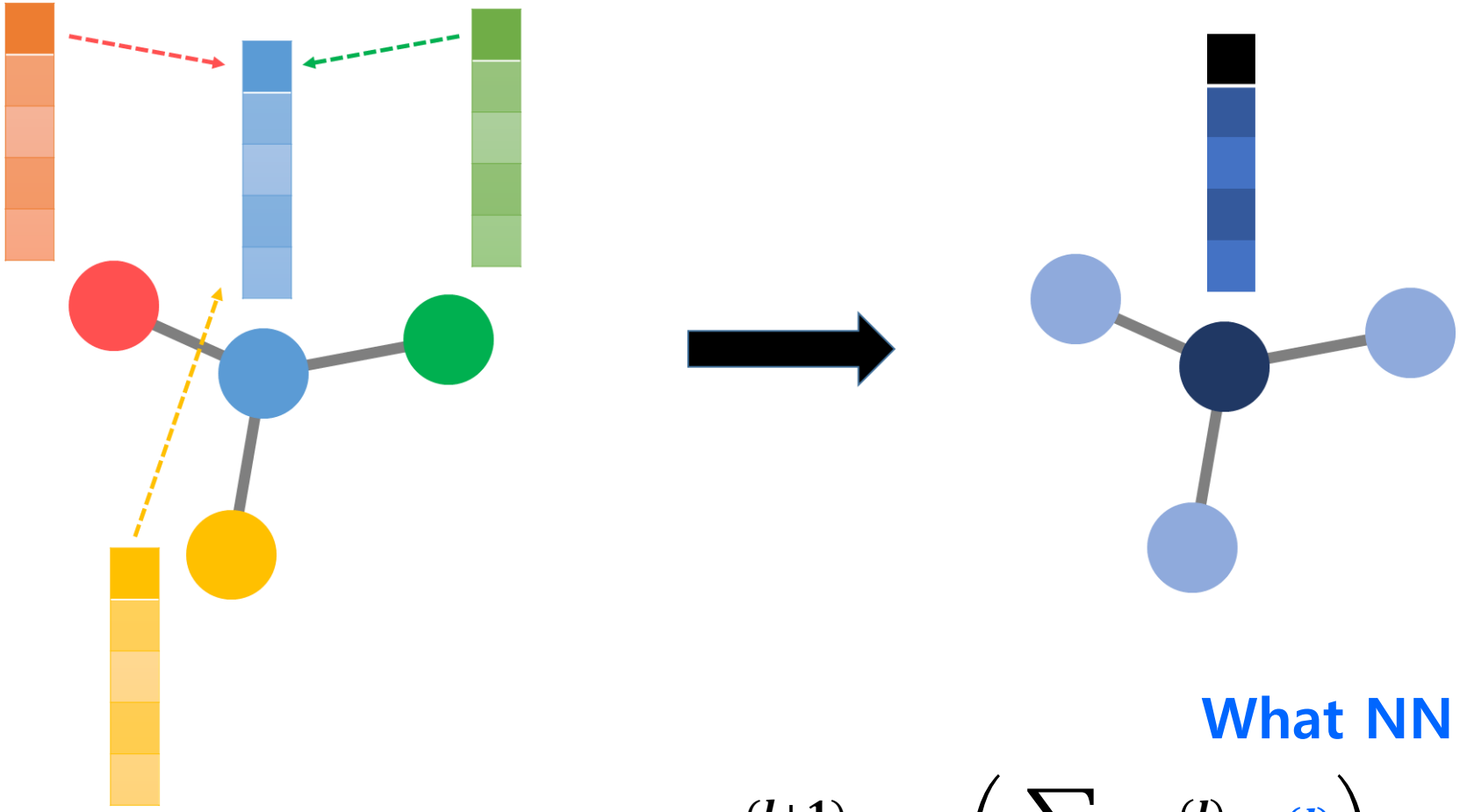
$$X_i^{(l+1)} = \sigma\left(\sum_{j\in[i-k,i+k]} W_j^{(l)} X_j^{(l)} + b_j^{(l)}\right)$$

GCN : Graph Convolutional Network



$$H_2^{(l+1)} = \sigma\left(H_1^{(l)}W^{(l)} + H_2^{(l)}W^{(l)} + H_3^{(l)}W^{(l)} + H_4^{(l)}W^{(l)}\right)$$

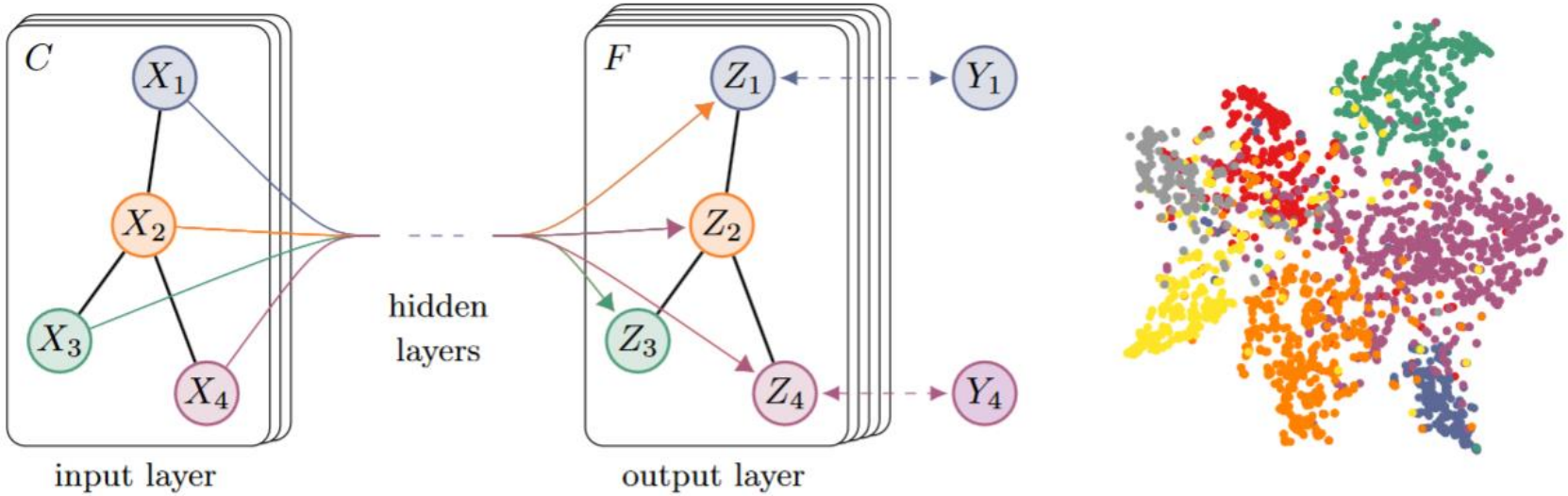
GCN : Graph Convolutional Network



What NN learns

$$H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} H_j^{(l)} W^{(l)}\right) = \sigma\left(A H_j^{(l)} W^{(l)}\right)$$

GCN : Graph Convolutional Network



(a) Graph Convolutional Network

(b) Hidden layer activations

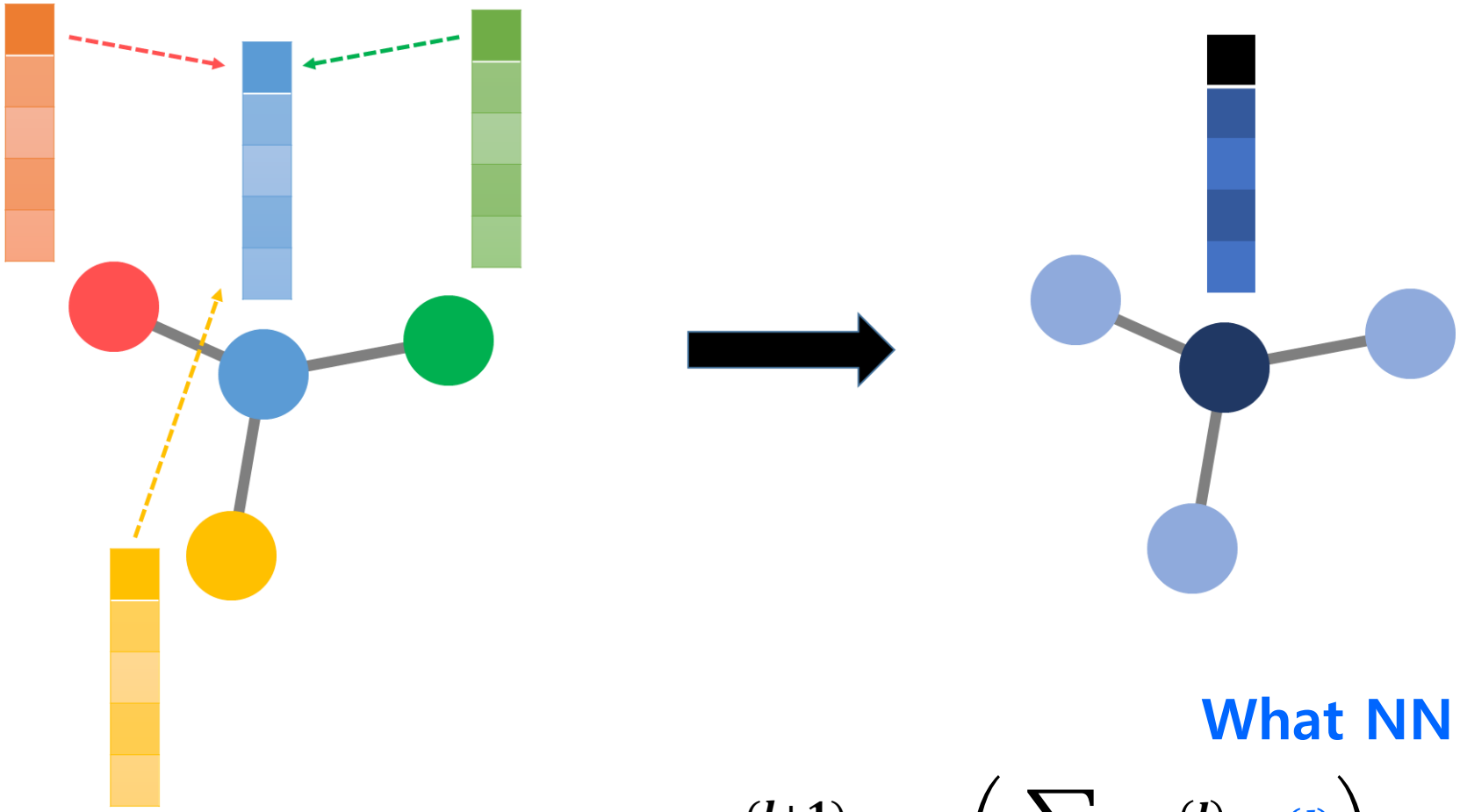
Figure 1: *Left*: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Y_i. *Right*: t-SNE (Maaten & Hinton, 2008) visualization of hidden layer activations of a two-layer GCN trained on the Cora dataset (Sen et al., 2008) using 5% of labels. Colors denote document class.

- Classification of nodes of citation networks and a knowledge graph
- $L = \sum_{l \in y_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$

Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)

GAT : Graph Attention Network

- **Attention revisits**

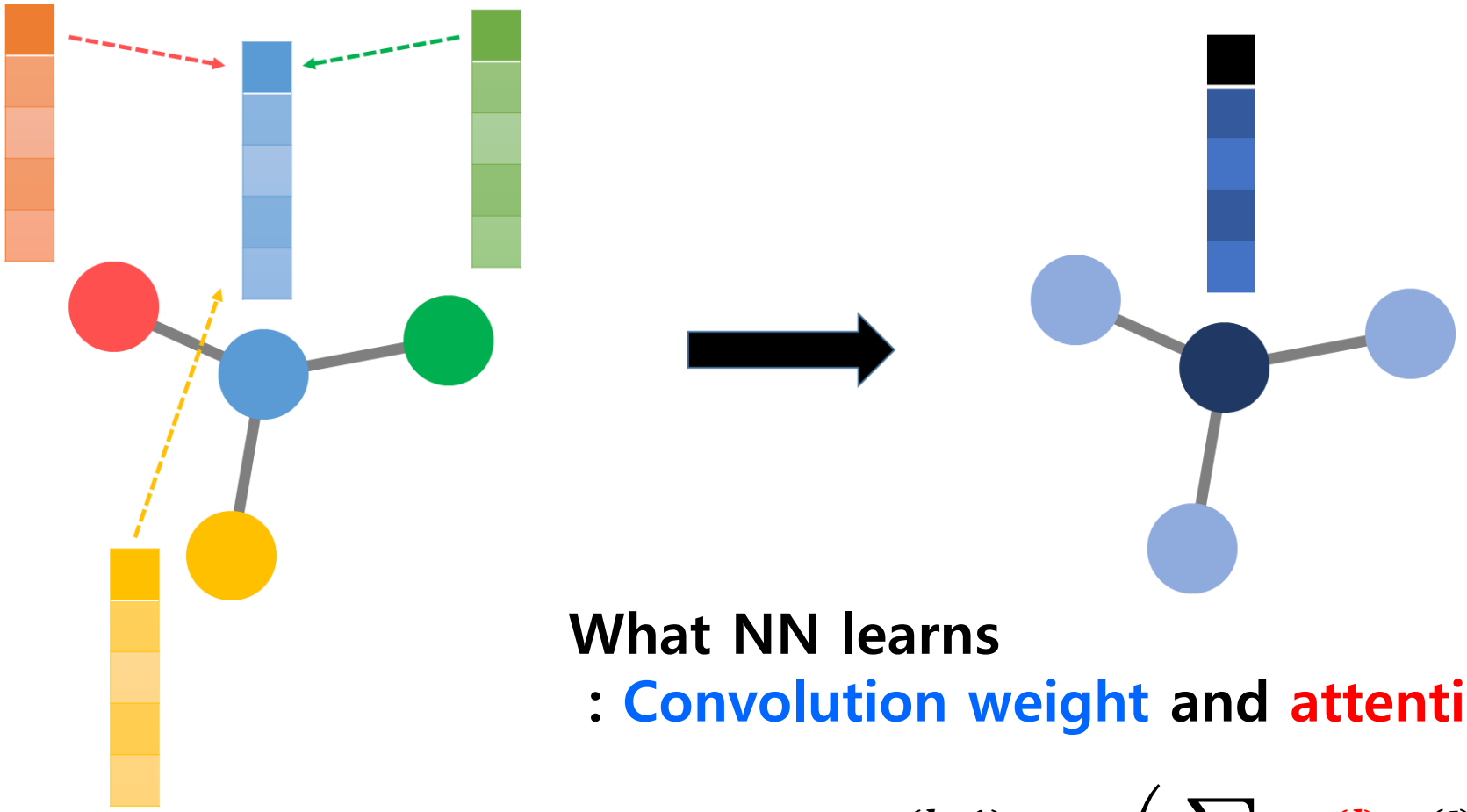


What NN learns

$$\text{GCN}: \quad H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} H_j^{(l)} W^{(l)}\right) = \sigma\left(A H_j^{(l)} W^{(l)}\right)$$

GAT : Graph Attention Network

- **Attention revisits**



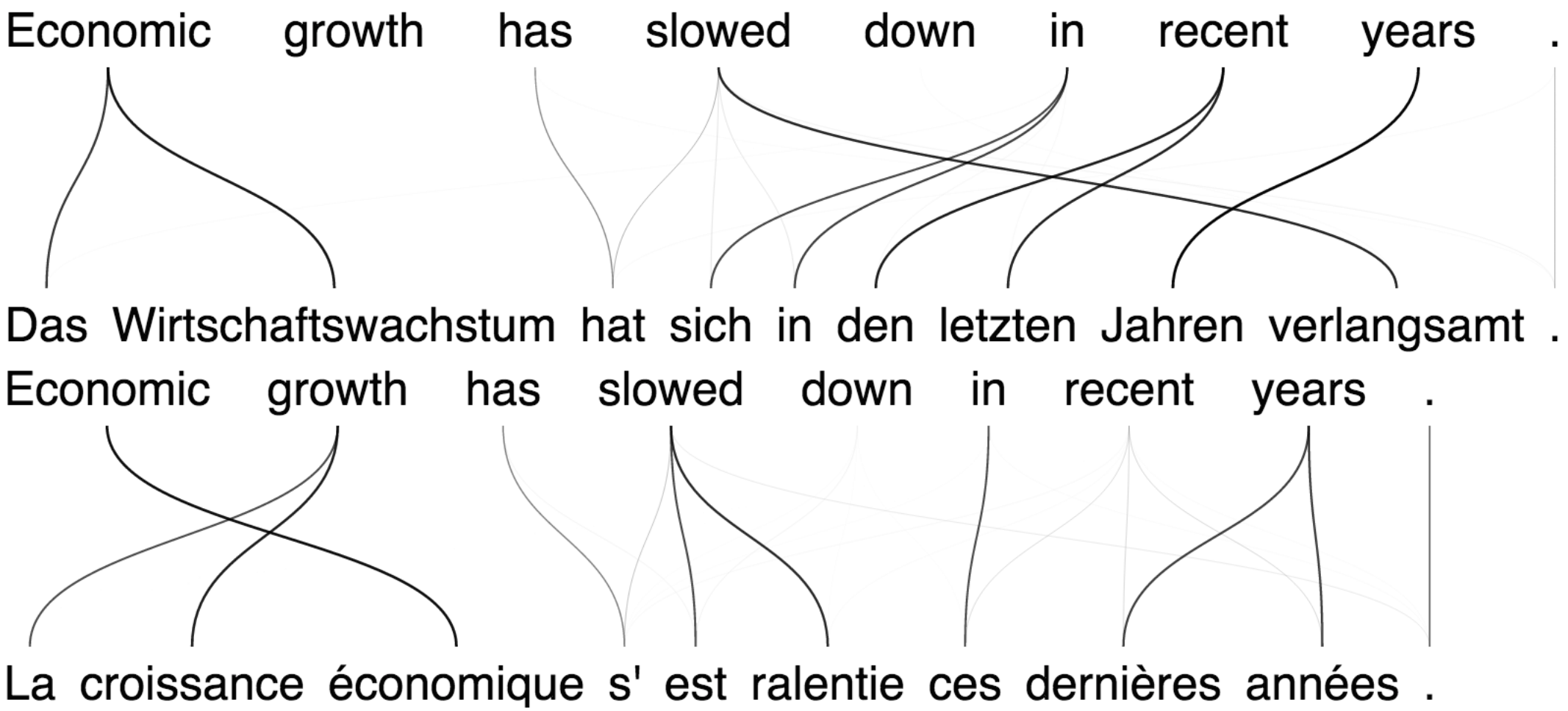
What NN learns
: Convolution weight and attention coefficient

$$\text{GAT}: \quad H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right)$$

Velickovic, Petar, et al.
"Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

GAT : Graph Attention Network

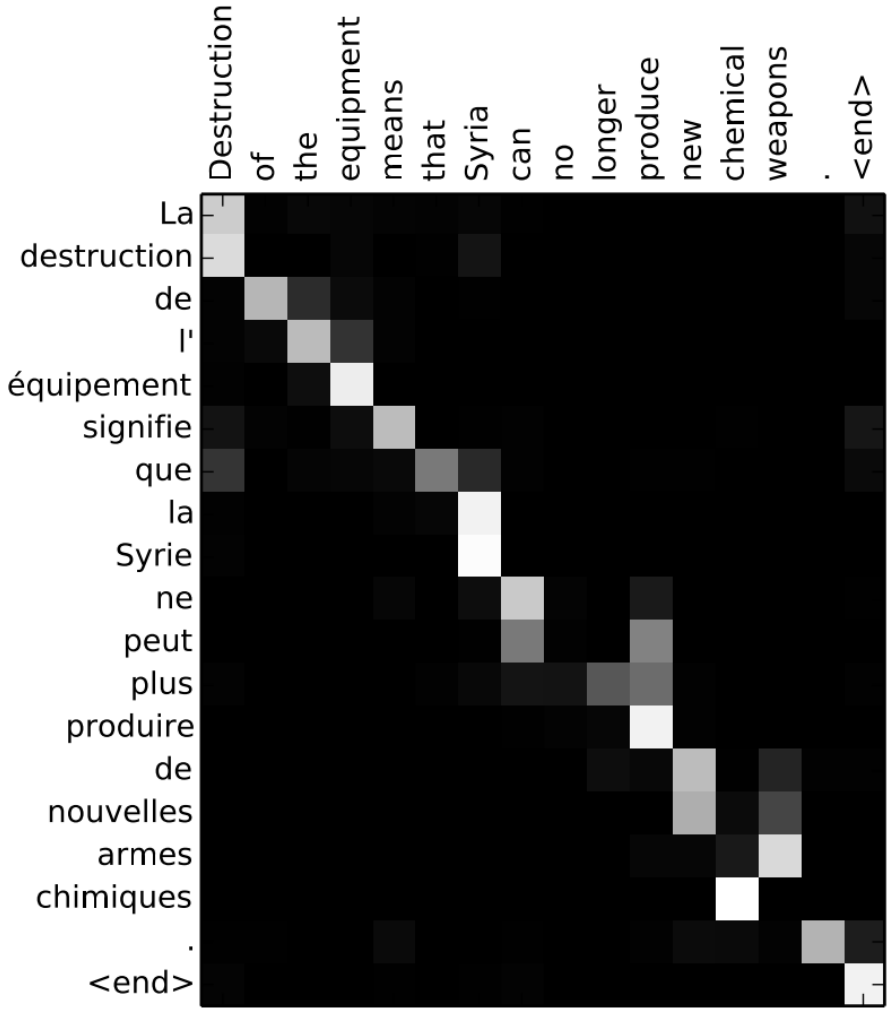
- **Attention mechanism in natural language processing**



https://devblogs.nvidia.com/introduction-neural-machine-translation-gpus-part-3/figure6_sample_translations-2/

GAT : Graph Attention Network

- **Attention mechanism in natural language processing**



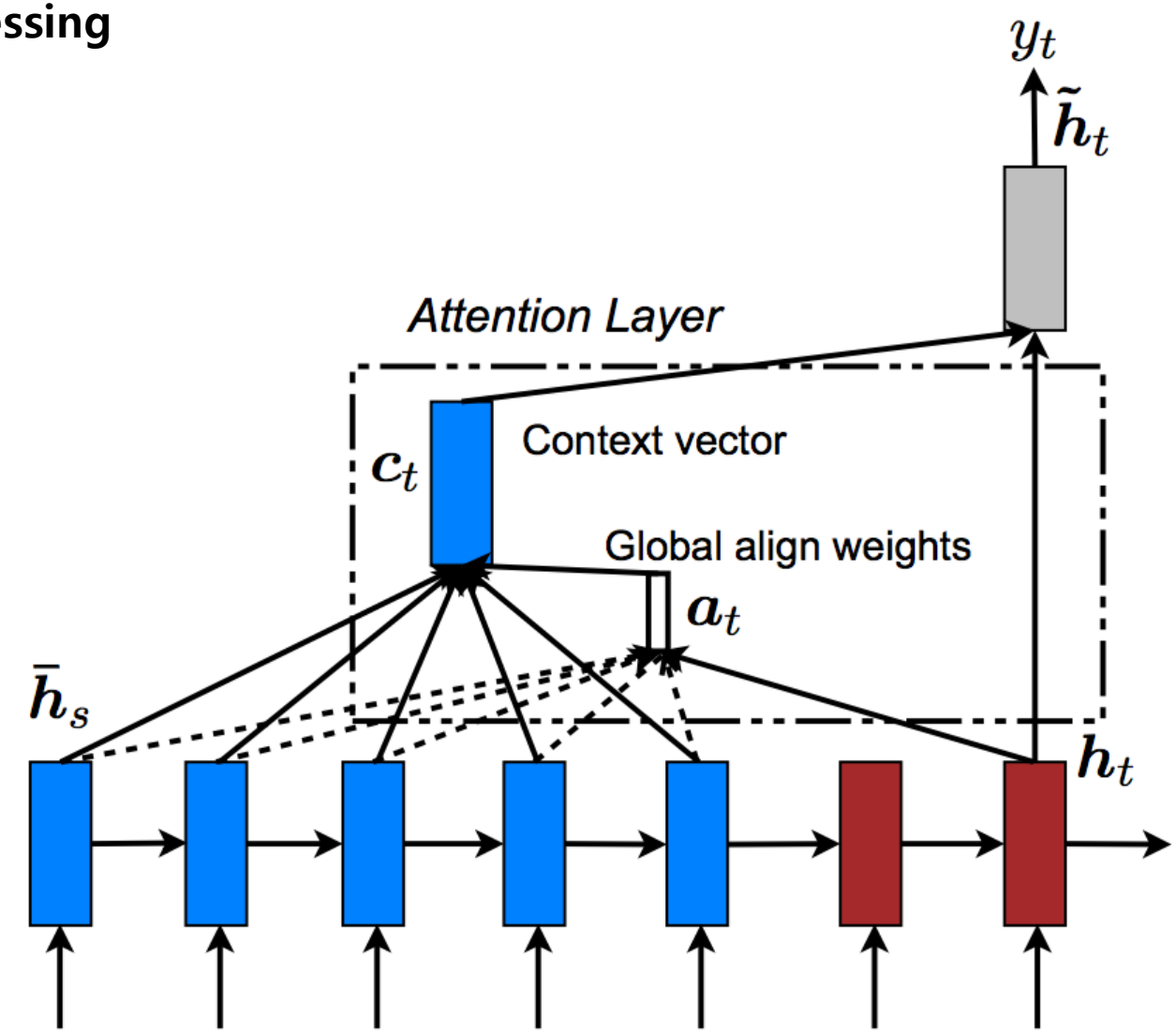
http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/

GAT : Graph Attention Network

- **Attention mechanism in natural language processing**

$$\boldsymbol{a_t}(s) = align(\boldsymbol{h_t}, \bar{\boldsymbol{h}}_{\boldsymbol{s}}) = softmax\left(score(\boldsymbol{h_t}, \bar{\boldsymbol{h}}_{\boldsymbol{s}})\right)$$

$$score(\boldsymbol{h_t}, \bar{\boldsymbol{h}}_{\boldsymbol{s}}) = \begin{cases} \beta \cdot \boldsymbol{h_t^T}\bar{\boldsymbol{h}}_{\boldsymbol{s}} \\ \boldsymbol{h_t^T W_a}\bar{\boldsymbol{h}}_{\boldsymbol{s}} \\ \boldsymbol{v_a^T}\tanh(\boldsymbol{W_a}[\boldsymbol{h_t}, \bar{\boldsymbol{h}}_{\boldsymbol{s}}]) \end{cases}$$



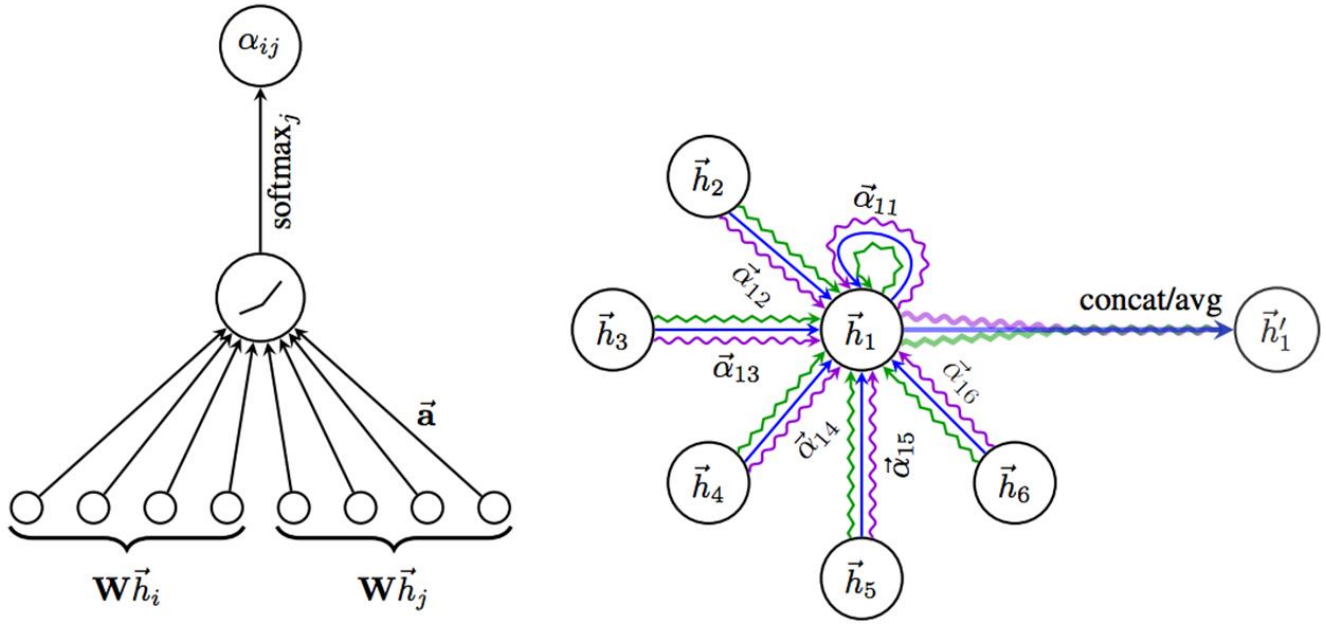
Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." *arXiv preprint arXiv:1508.04025* (2015).

GAT : Graph Attention Network

- **Attention revisits**

What NN learns
: Convolution weight and attention coefficient

$$H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right) \qquad \alpha_{ij} = f(H_i W, H_j W)$$



Velickovic, Petar, et al.
"Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

GAT : Graph Attention Network

- **Attention revisits**

What NN learns
: Convolution weight and attention coefficient

$$H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right) \qquad \alpha_{ij} = f(H_i W, H_j W)$$

- Velickovic, Petar, et al. – network analysis

$$\alpha_{ij} = softmax(e_{ij}) = \frac{e_{ij}}{exp\left(\sum_{k \in N(i)} e_{ik}\right)} \qquad e_{ij} = LearkyReLU(a^T[H_i W, H_j W])$$

- Seongok Ryu, et al. – molecular applications

$$\alpha_{ij} = \tanh\left((H_i W)^T C(H_j W)\right)$$

GAT : Graph Attention Network

- **Multi-head attention**

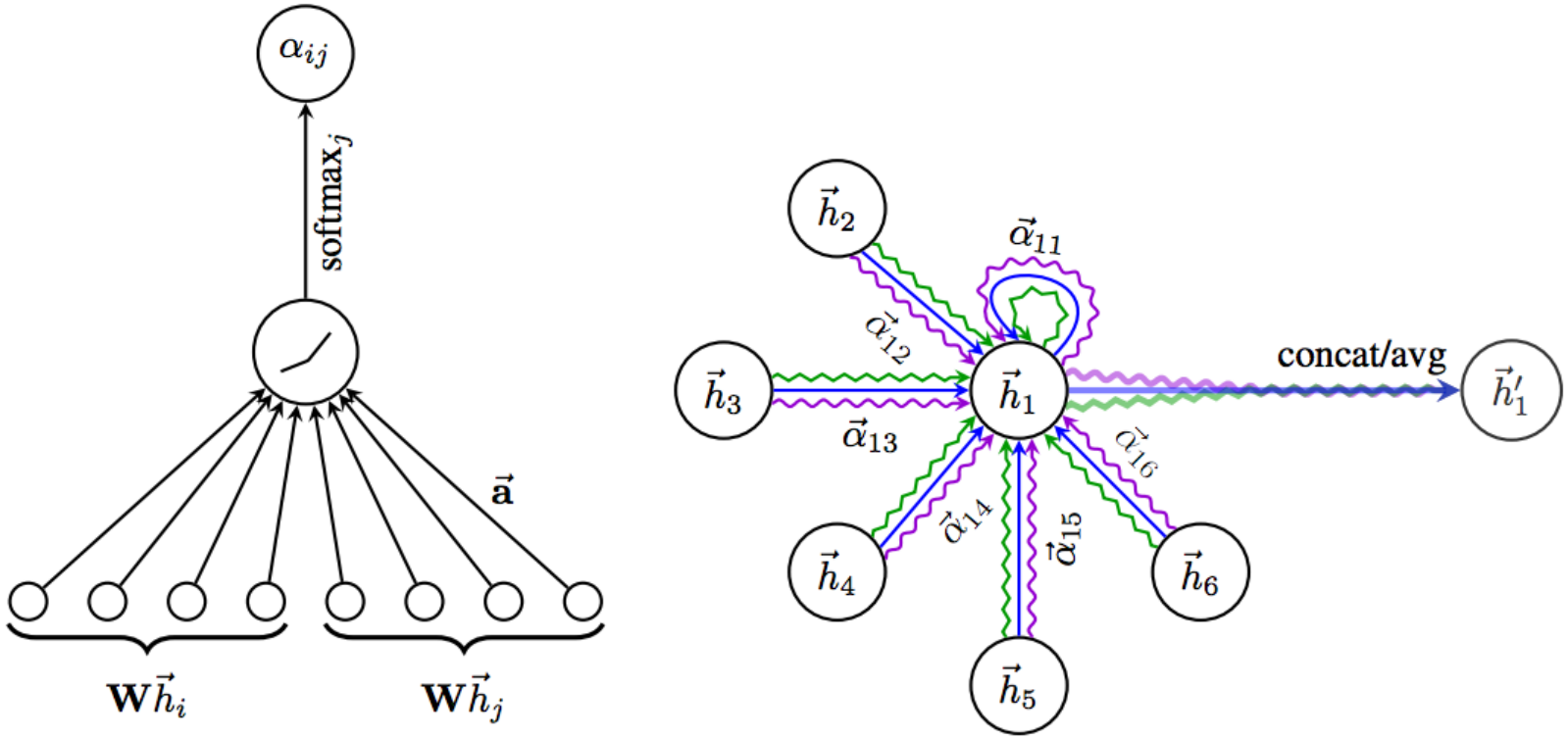


Figure 1: **Left:** The attention mechanism $a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$.

- **Average**

$$H_i^{(l+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right)$$

- **Concatenation**

$$H_i^{(l+1)} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j\in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right)$$

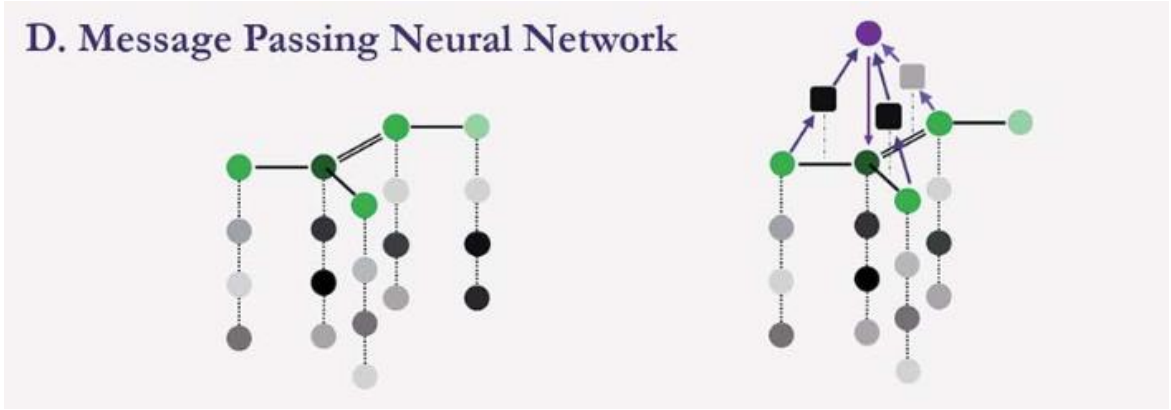
Velickovic, Petar, et al.
"Graph attention networks." *arXiv preprint arXiv:1710.10903* (2017).

GGNN : Gated Graph Neural Network

- **Message Passing Neural Network (MPNN) framework**

Using previous node state and message state to update the i^{th} hidden node state

$$h_i^{(l+1)} = U(h_i^{(l)}, m_i^{(l+1)})$$



The message state is updated by previous neighbor states and the route state, and edge states.

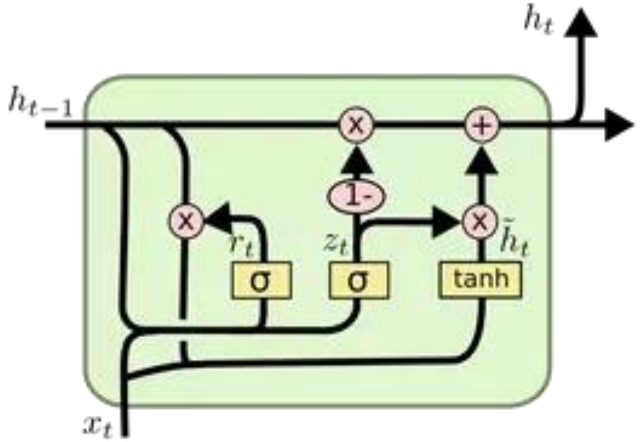
$$m_i^{(l+1)} = \sum_{j \in N(i)} M^{(l+1)}(h_i, h_j, e_{ij})$$

Wu, Zhenqin, et al.
"MoleculeNet: a benchmark for molecular machine learning." *Chemical science* 9.2 (2018): 513-530.

GGNN : Gated Graph Neural Network

- **Using recurrent unit for updating the node states, in this case GRU.**

$$h_i^{(l+1)} = U(h_i^{(l)}, m_i^{(l+1)}) \longrightarrow h_i^{(l+1)} = GRU(h_i^{(l)}, m_i^{(l+1)})$$



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

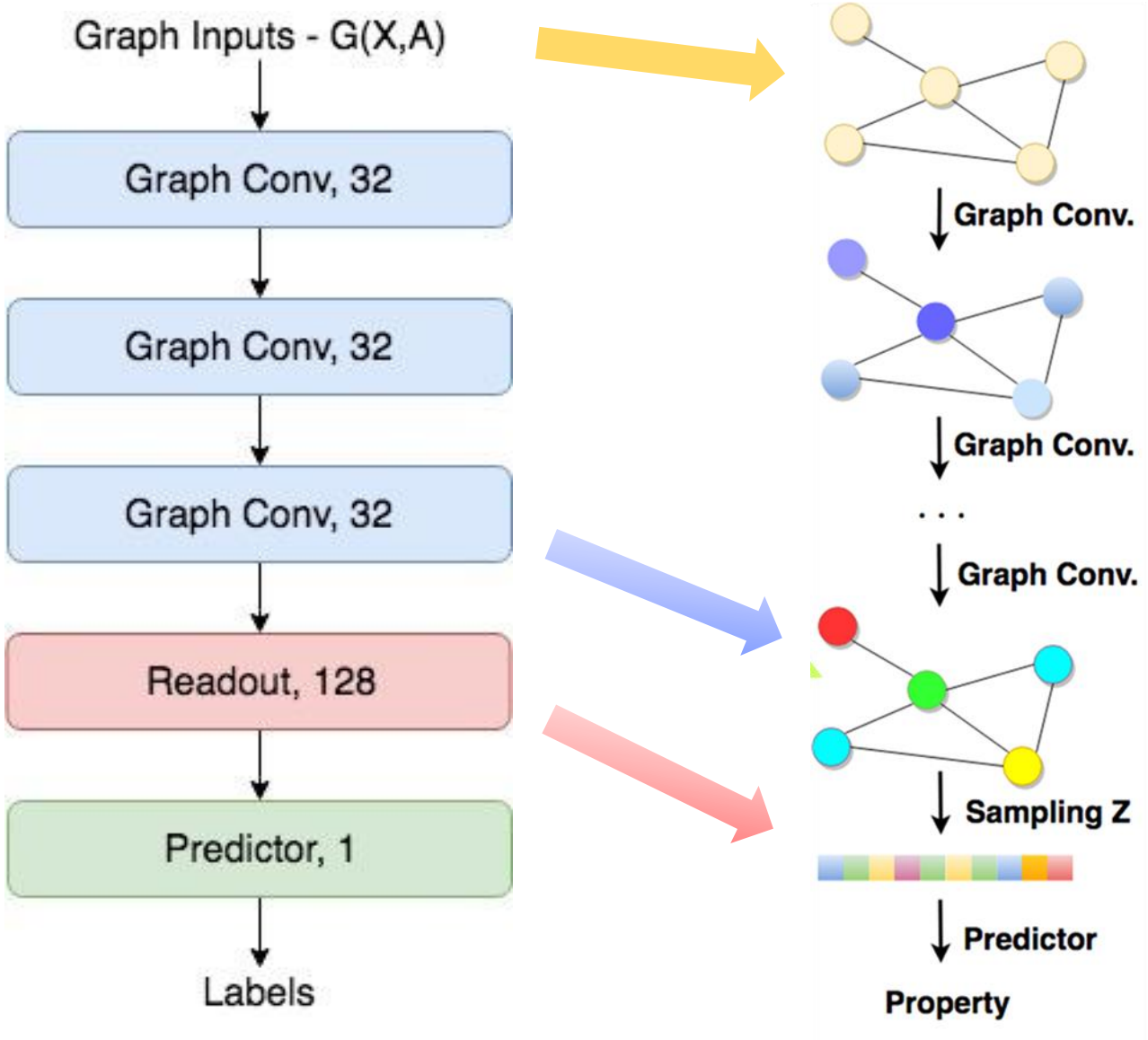
Updating rate of the temporal state

$$h_i^{(l+1)} = z^{(l+1)} * \tilde{h}_i^{(l+1)} + \left(1 - z^{(l+1)}\right) * h_i^{(l)}$$

Temporal node state **Previous node state**

Li, Yujia, et al.
"Gated graph sequence neural networks." *arXiv preprint arXiv:1511.05493* (2015).

Readout : permutation invariance on changing nodes order



Input node features, $\{H_i^{(0)}\}$

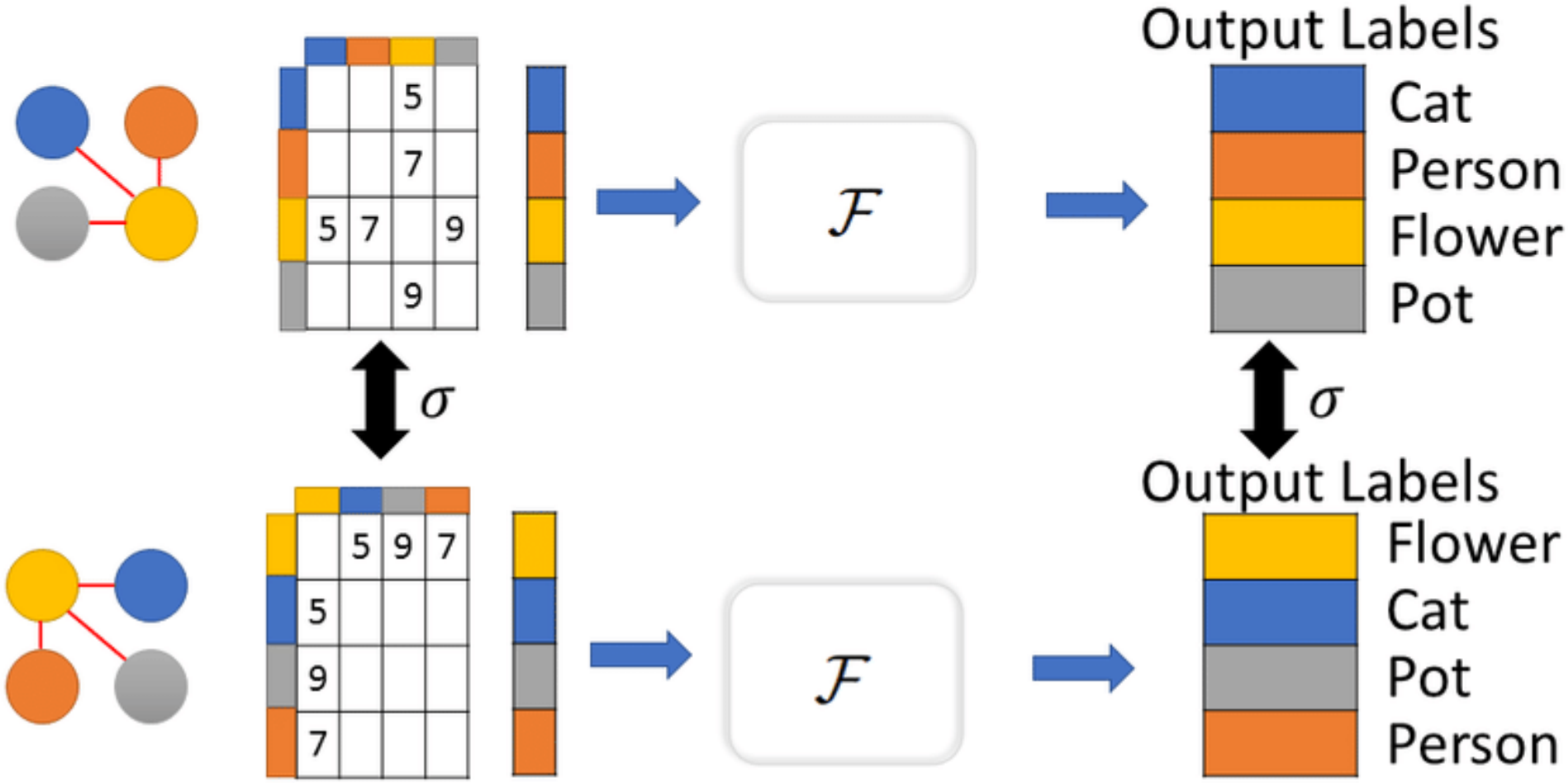
: Raw node information

Final node states, $\{H_i^{(L)}\}$

: How the NN recognizes the nodes

Graph features, Z

Readout : permutation invariance on changing nodes order



Mapping Images to Scene Graphs with Permutation-Invariant Structured Prediction - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/Graph-permutation-invariance-and-structured-prediction-A-graph-labeling-function-F-is_fig1_323217335 [accessed 8 Sep, 2018]

Readout : permutation invariance on changing nodes order

- **Graph feature**

$$z_G = f\left(\left\{H_i^{(L)}\right\}\right)$$

- **Node-wise summation**

$$z_G = \tau\left(\sum_{i \in G} MLP\left(H_i^{(L)}\right)\right)$$

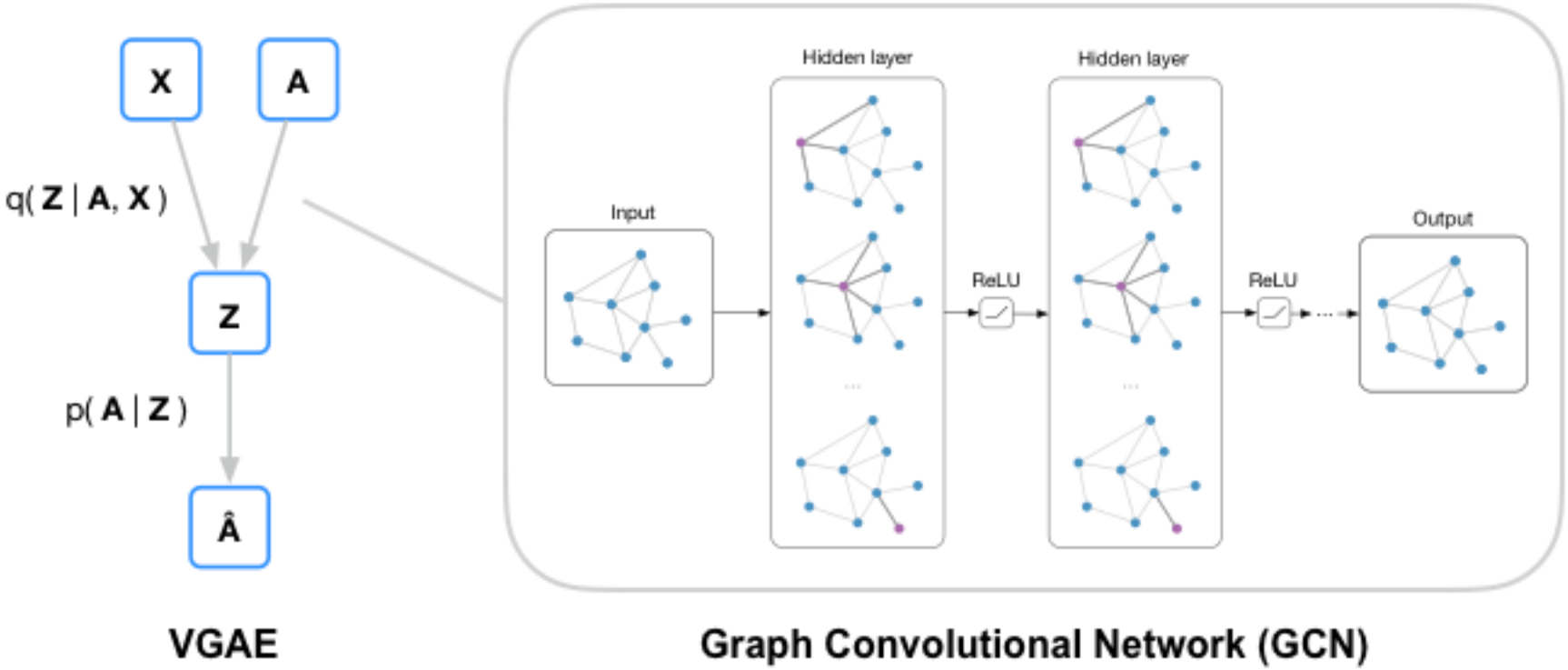
- **Graph gathering**

$$z_G = \tau\left(\sum_{i \in G} \sigma\left(MLP_1\left(H_i^{(L)}, H_i^{(0)}\right)\right) \odot MLP_2\left(H_i^{(L)}\right)\right)$$

- τ : ReLU activation
- σ : sigmoid activation

Gilmer, Justin, et al.
"Neural message passing for quantum chemistry." *arXiv preprint arXiv:1704.01212* (2017).

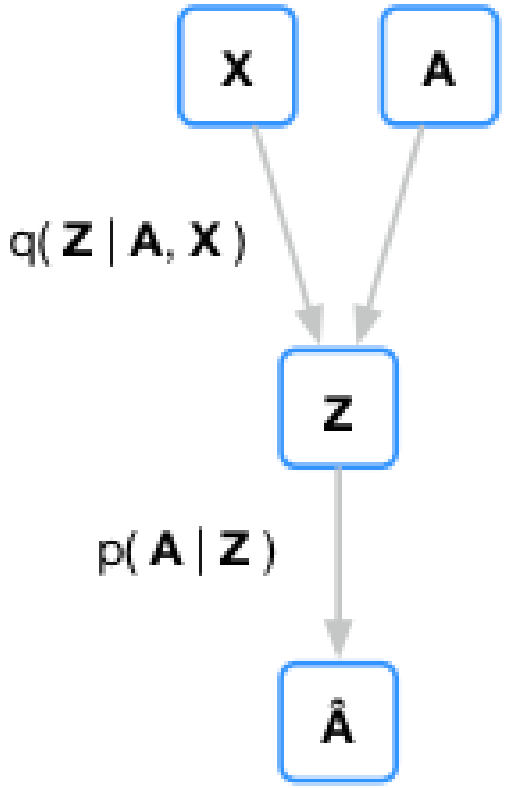
Graph Auto-Encoders (GAE)



- **Clustering**
- **Link prediction**
- **Matrix completion and recommendation**

Kipf, Thomas N., and Max Welling.
"Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Graph Auto-Encoders (GAE)



VGAE

Input graph, $\boldsymbol{G(X, A)}$

↓ **Encoder**

Node features, $q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})$

↓ **Decoder**

Reconstructed
Adjacency matrix, $\hat{A} = \boldsymbol{\sigma(ZZ^T)}$

Kipf, Thomas N., and Max Welling.
"Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Graph Auto-Encoders (GAE)

Encoder - Inference model

- **Two-layer GCN**

$$GCN(\boldsymbol{X}, \boldsymbol{A}) = \tilde{\boldsymbol{A}} ReLU\left(\tilde{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W_0}\right)\boldsymbol{W_1}$$

$\tilde{\boldsymbol{A}} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$: symmetrically normalized adjacency matrix

- **Variational Inference**

$$q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A}) = \prod_{i=1}^{N} q(\boldsymbol{z_i}|\boldsymbol{X}, \boldsymbol{A}) \qquad q(\boldsymbol{z_i}|\boldsymbol{X}, \boldsymbol{A}) = N\left(\boldsymbol{z_i}|\boldsymbol{\mu_i}, diag\left(\boldsymbol{\sigma_i^2}\right)\right)$$

Input graph, $\boldsymbol{G}(\boldsymbol{X}, \boldsymbol{A})$

↓ **Encoder**

Node features, $q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})$

↓ **Decoder**

Reconstructed Adjacency matrix, $\hat{A} = \boldsymbol{\sigma}(\boldsymbol{Z}\boldsymbol{Z}^{\boldsymbol{T}})$

Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Graph Auto-Encoders (GAE)

Decoder - Generative model

- **Inner product between latent vectors**

$$p(\boldsymbol{A}|\boldsymbol{Z}) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(\boldsymbol{A_{ij}}|\boldsymbol{z_i}, \boldsymbol{z_j}) \qquad p(\boldsymbol{A_{ij}}|\boldsymbol{z_i}, \boldsymbol{z_j}) = \sigma(\boldsymbol{z_i^T z_j})$$

A_{ij} : the elements of (reconstructed) A

$\sigma(\cdot)$: sigmoid activation

$\widehat{\boldsymbol{A}} = \boldsymbol{\sigma}(\boldsymbol{Z}\boldsymbol{Z}^{\boldsymbol{T}})$ with $\boldsymbol{Z} = \boldsymbol{GCN}(\boldsymbol{X}, \boldsymbol{A})$

- **Learning**

$$\mathcal{L} = \mathbb{E}_{q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}[\log p(\boldsymbol{A}|\boldsymbol{Z})] - \mathrm{KL}[q(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A}) \,||(p(\boldsymbol{Z})]$$

Reconstruction loss KL-divergence

Input graph, $\boldsymbol{G}(\boldsymbol{X}, \boldsymbol{A})$

↓ **Encoder**

Node features, $q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})$

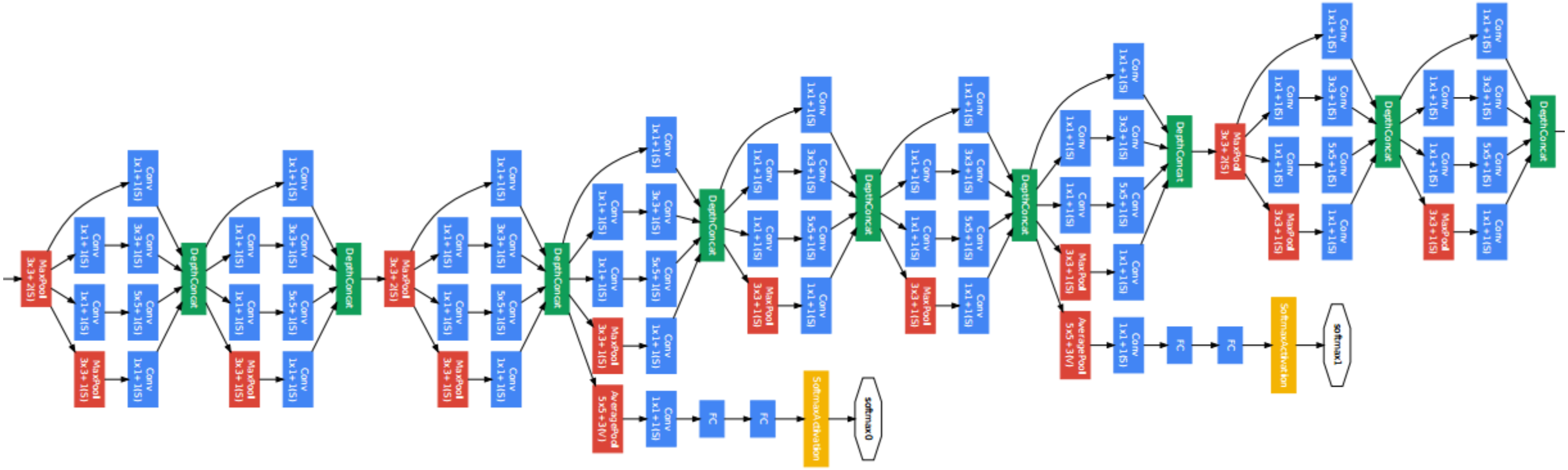
↓ **Decoder**

Reconstructed
Adjacency matrix, $\widehat{\boldsymbol{A}} = \boldsymbol{\sigma}(\boldsymbol{Z}\boldsymbol{Z}^{\boldsymbol{T}})$

Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Practical Issues
: Inception

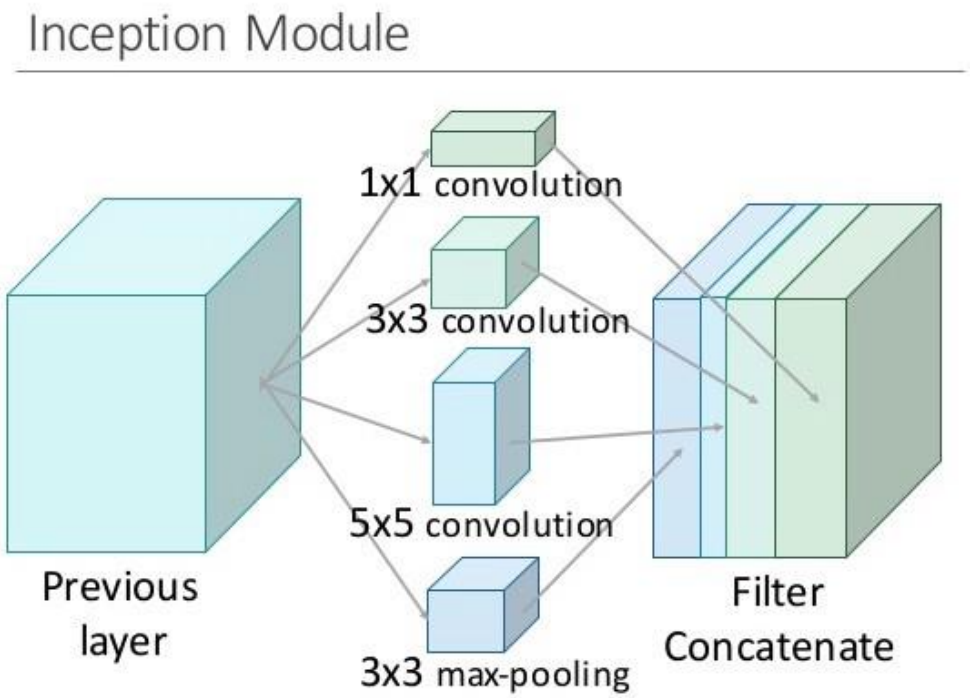
GoogLeNet – Winner of 2014 ImageNet Challenge

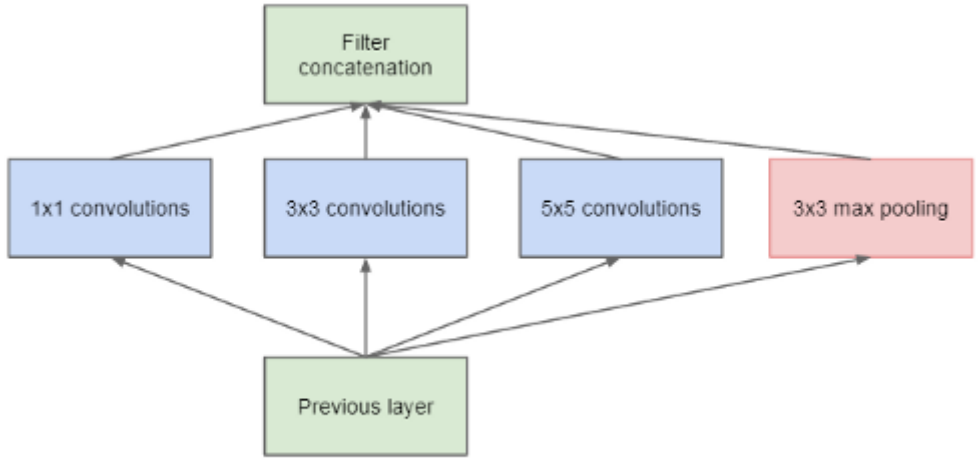


https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

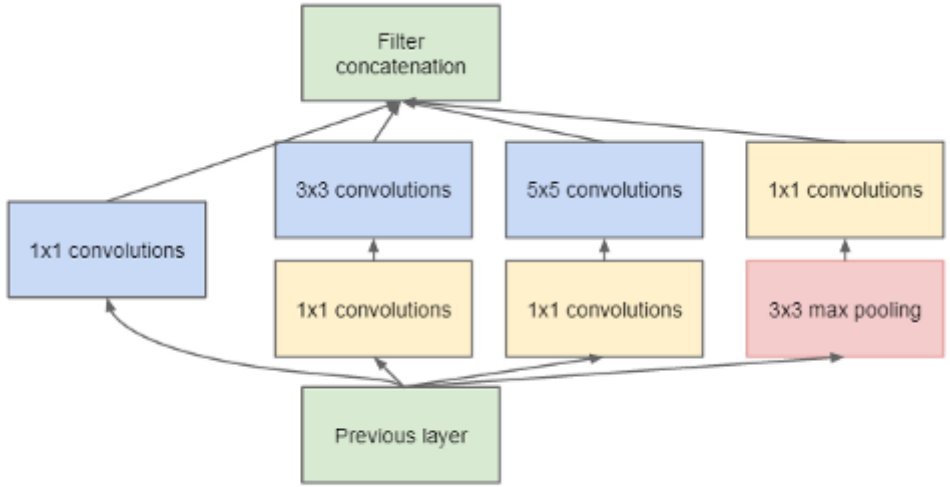
Practical Issues : Inception

Inception module





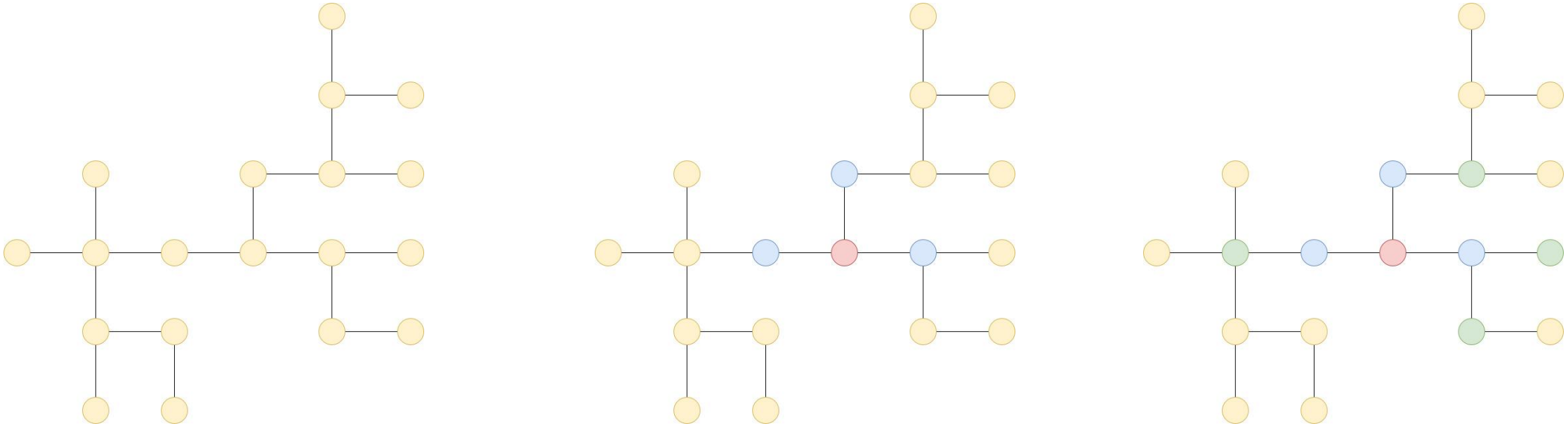
(a) Inception module, naïve version



(b) Inception module with dimension reductions

https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

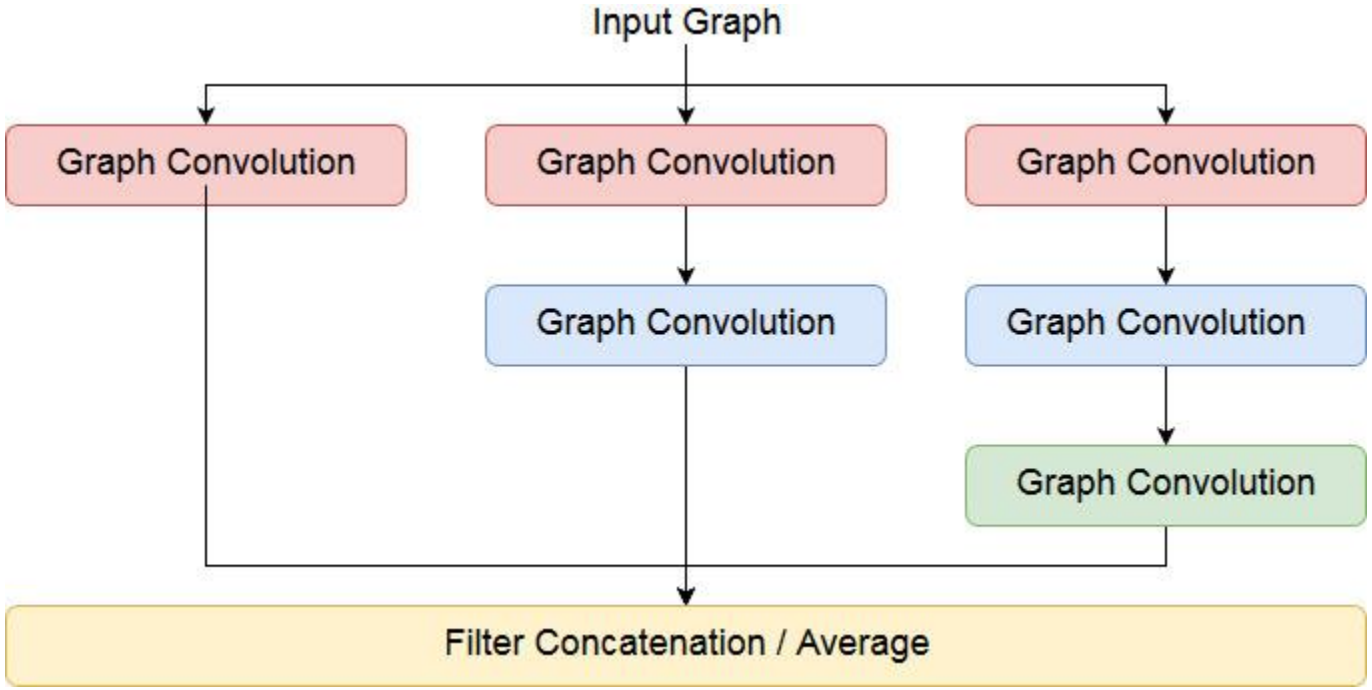
Practical Issues
: Inception

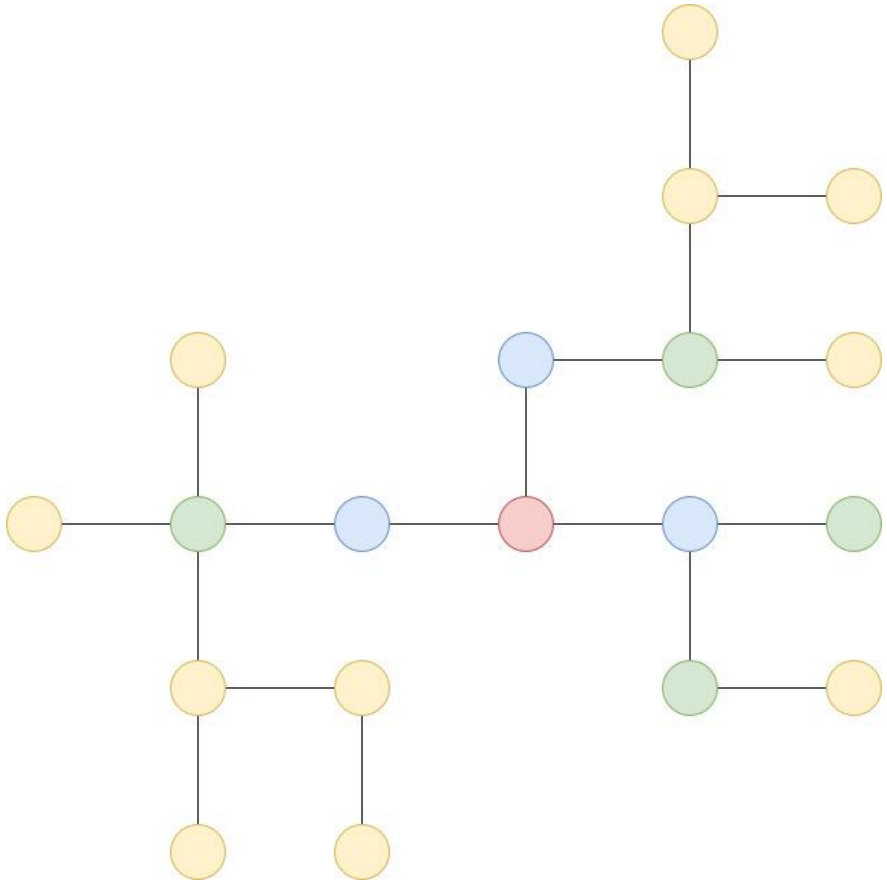


$$H_i^{(1)} = \sigma\left(AH_j^{(0)}W^{(0)}\right)$$

$$H_i^{(2)} = \sigma\left(AH_j^{(1)}W^{(1)}\right)$$

Practical Issues
: Inception

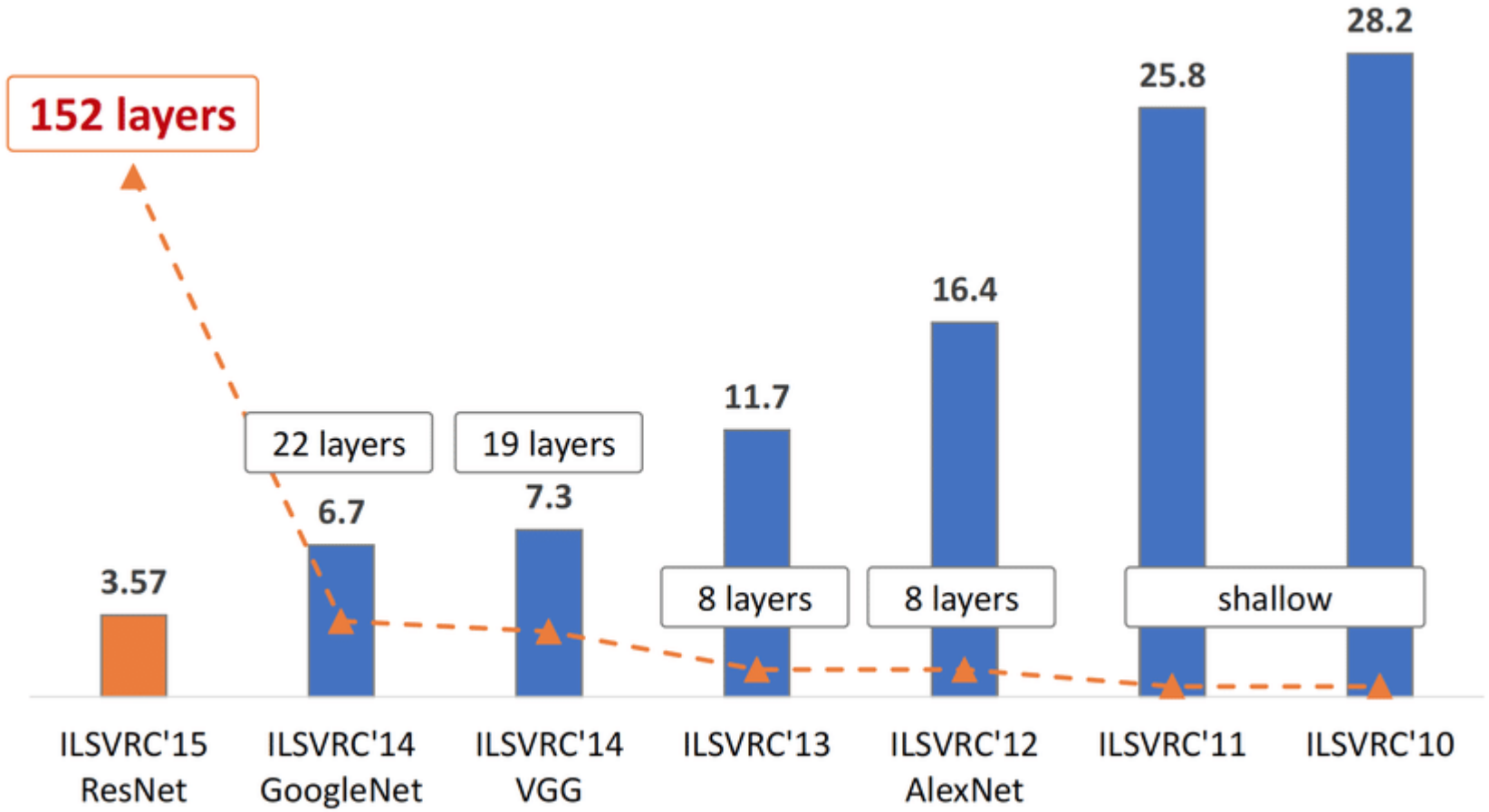




- Make network **wider**
- Avoid **vanishing gradient**

Practical Issues
: Skip-connection

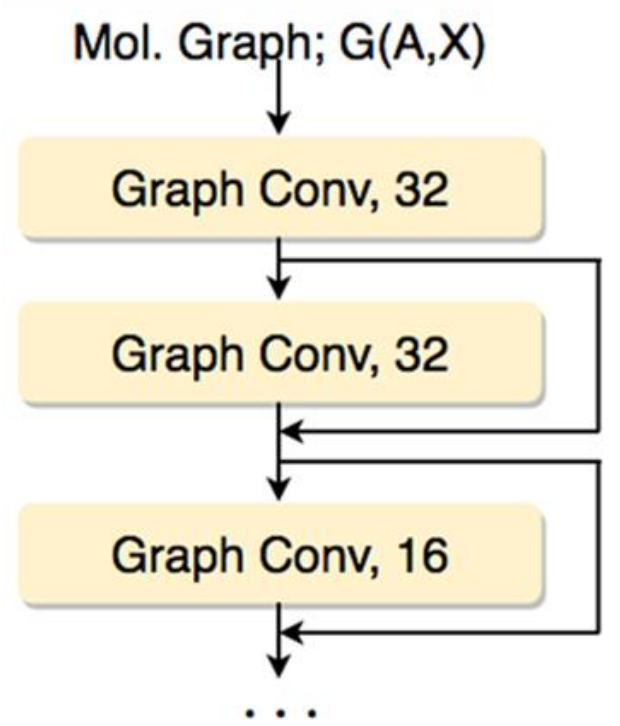
ResNet – Winner of 2015 ImageNet Challenge



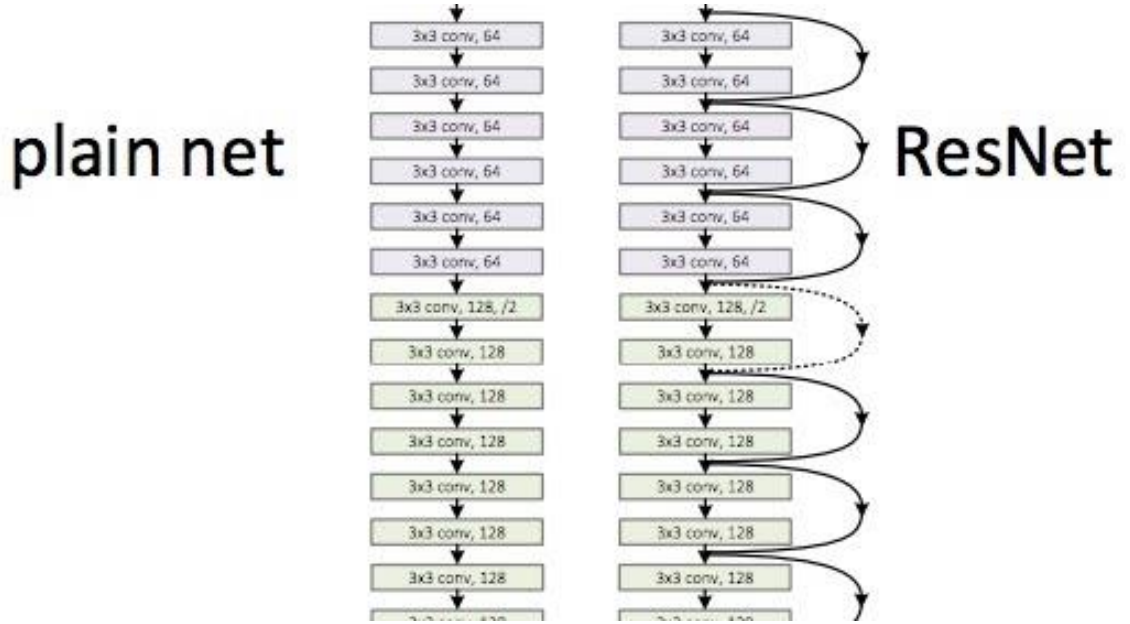
Going Deeeeeeeeper!

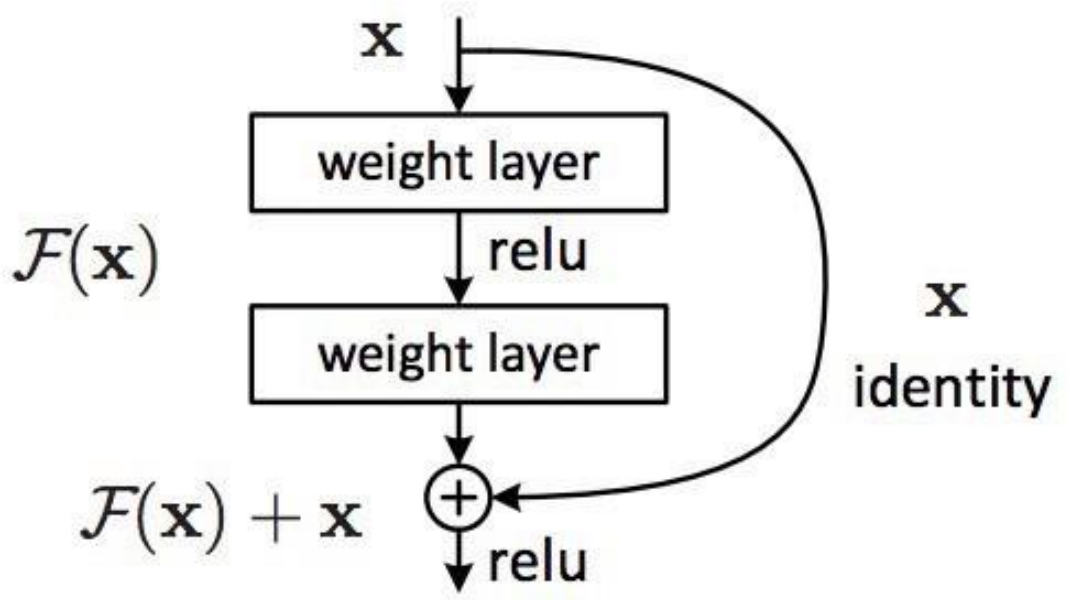
Practical Issues : Skip-connection

ResNet – Winner of 2015 ImageNet Challenge



$$y = H_i^{(l+1)} + H_i^{(l)}$$

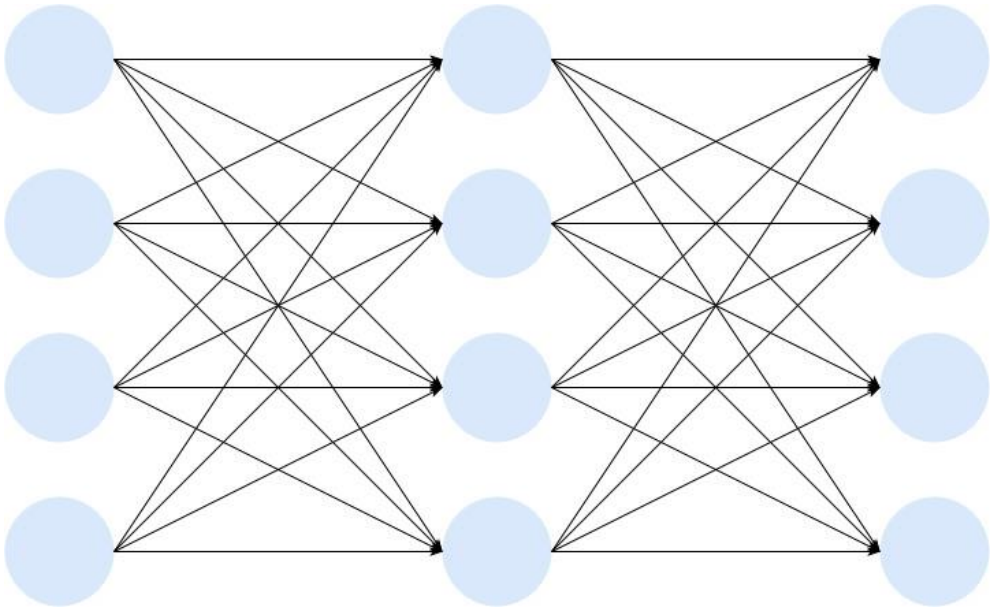


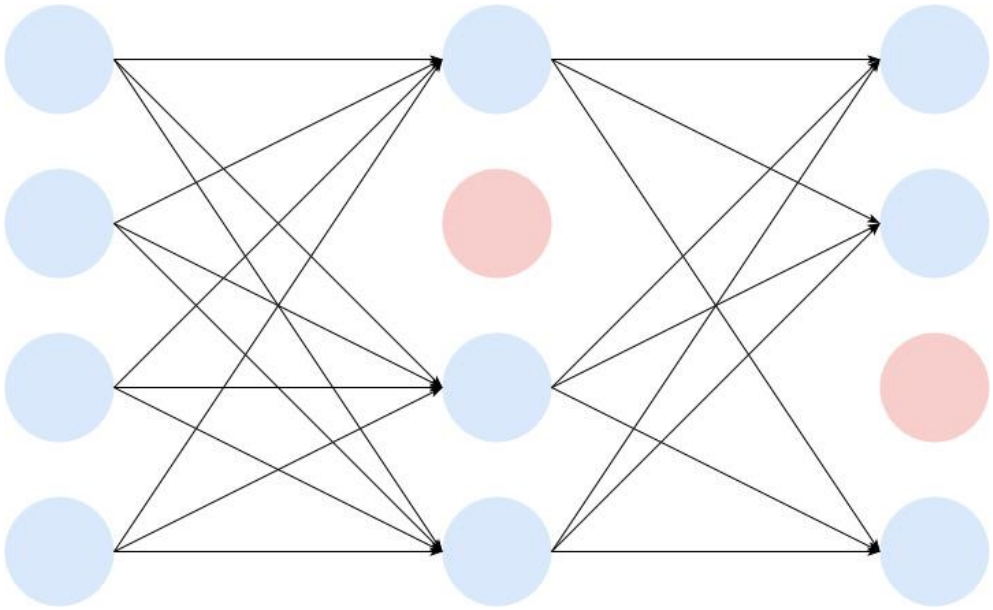


https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

Practical Issues
: Dropout

Dropout rate (p) = 0.25





parameters
: 4x4 = 16

parameters
: 4x4 = 16

parameters
: 16x0.75 = 12

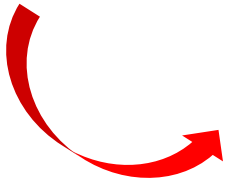
parameters
: 16x0.75x0.75 = 9

For a dense network, the dropout of hidden state reduces the number of parameters from N_w to $N_w(1-p)^2$

Practical Issues
: Dropout

Hidden states in a graph neural network

Graph Conv. : $H_i^{(l+1)} = A\boldsymbol{H}^{(l)}W$



	Node 1	Node 2	Node 3	…
Age	28	40	36	…
Sex	M	F	F	…
Nationality	Korean	American	French	…
Job	Student	Medical Doctor	Politician	…
	$\boldsymbol{H_1^{(l)}}$	$\boldsymbol{H_2^{(l)}}$	$\boldsymbol{H_3^{(l)}}$	$\boldsymbol{H_4^{(l)}}$

Practical Issues
: Dropout

Hidden states in a graph neural network

	Node 1	Node 2	Node 3	…
Age	28	40	36	…
Sex	M	F	F	…
Nationality	Korean	American	French	…
Job	Student	Medical Doctor	Politician	…
	$H_1^{(l)}$	$H_2^{(l)}$	$H_3^{(l)}$	$H_4^{(l)}$

Mask individual person

	Node 1	Node 2	Node 3	…
Age	28	40	36	…
Sex	M	F	F	…
Nationality	Korean	American	French	…
Job	Student	Medical Doctor	Politician	…
	$H_1^{(l)}$	$H_2^{(l)}$	$H_3^{(l)}$	$H_4^{(l)}$

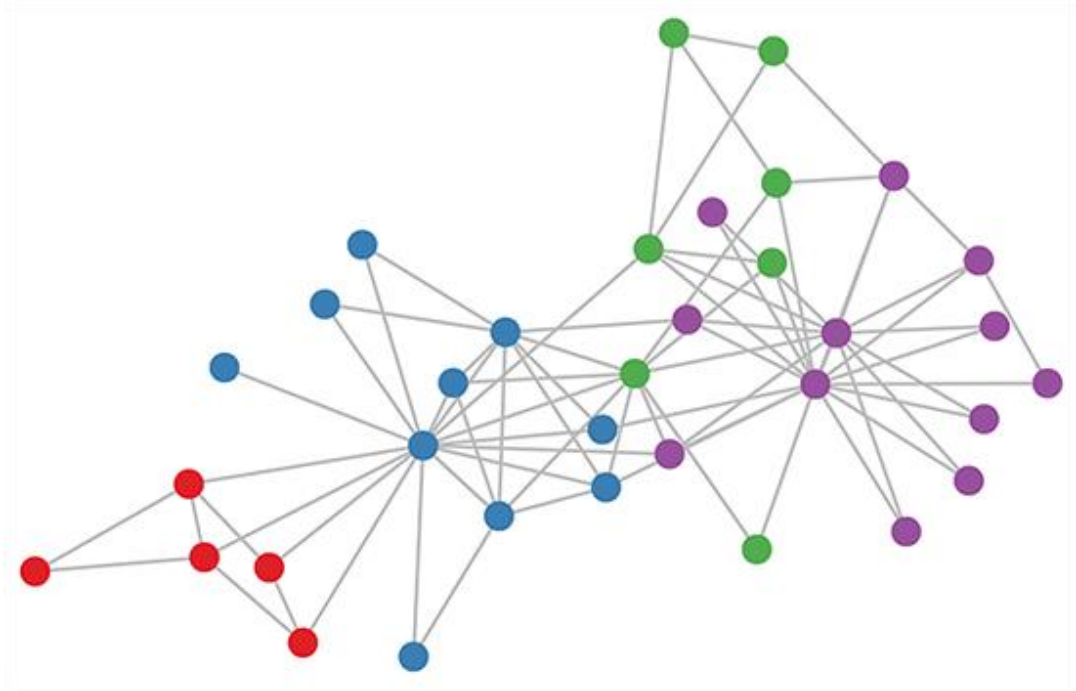
Mask information (features) of node states

And many other options are possible. The proper method depends on your task.

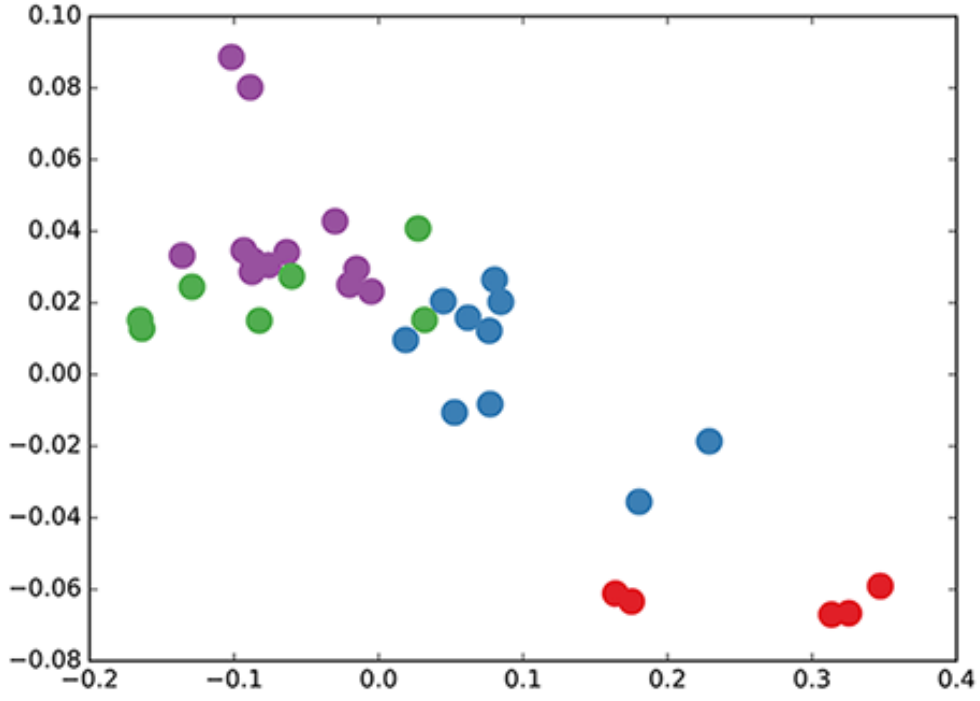
Applications of graph neural networks

- **Network Analysis**
  1. Node classification
  2. Link prediction
  3. Matrix completion
- **Molecular Applications**
  1. Neural molecular fingerprint
  2. Quantitative Structure-Property Relationship (QSPR)
  3. Molecular generative model
- **Interacting physical system**

Network analysis
1. Node classification – karate club network



Karate club graph, colors denote communities obtained via modularity-based clustering (Brandes et al., 2008).



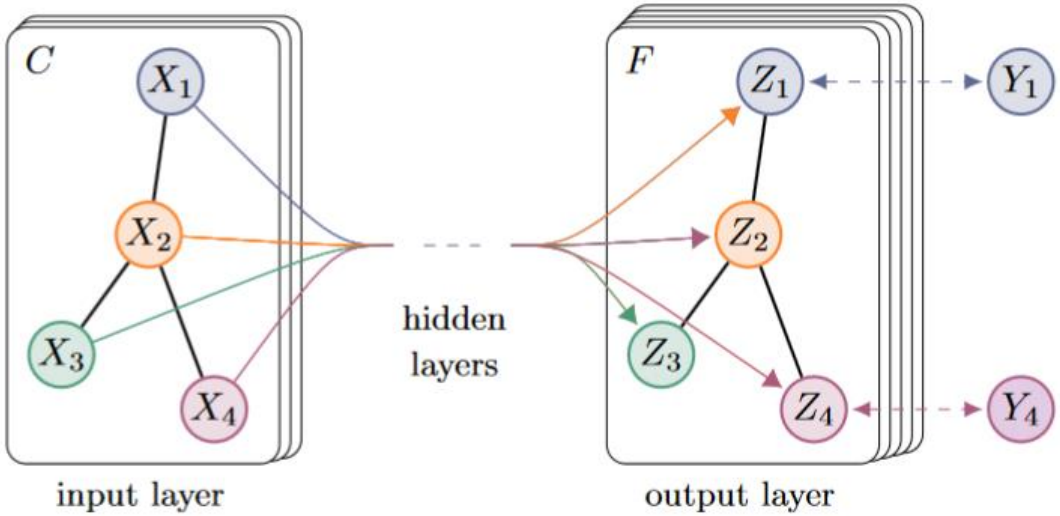
GCN embedding (with random weights) for nodes in the karate club network.

- All figures and descriptions are taken from Thomas N. Kipf's blog.
- Watch video on his blog.

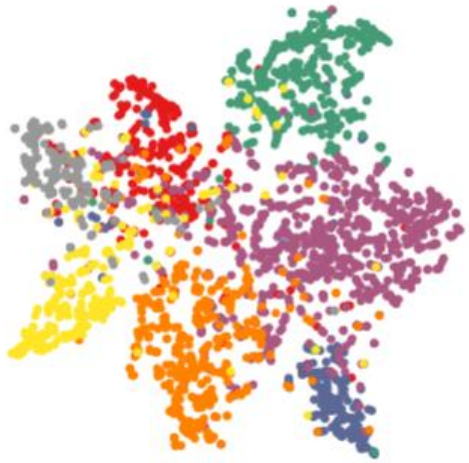
Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Good node features → Good node classification results



(a) Graph Convolutional Network



(b) Hidden layer activations

Figure 1: *Left*: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Y_i. *Right*: t-SNE (Maaten & Hinton, 2008) visualization of hidden layer activations of a two-layer GCN trained on the Cora dataset (Sen et al., 2008) using 5% of labels. Colors denote document class.

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Semi-supervised learning – low label rate
- Citation network – Citeseer, Cora, Pubmed / Bipartite graph - NELL

Table 1: Dataset statistics, as reported in Yang et al. (2016).

Dataset	Type	Nodes	Edges	Classes	Features	Label rate
Citeseer	Citation network	3,327	4,732	6	3,703	0.036
Cora	Citation network	2,708	5,429	7	1,433	0.052
Pubmed	Citation network	19,717	44,338	3	500	0.003
NELL	Knowledge graph	65,755	266,144	210	5,414	0.001

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Outperforms classical machine learning methods

Table 2: Summary of results in terms of classification accuracy (in percent).

Method	Citeseer	Cora	Pubmed	NELL
ManiReg [3]	60.1	59.5	70.7	21.8
SemiEmb [28]	59.6	59.0	71.1	26.7
LP [32]	45.3	68.0	63.0	26.5
DeepWalk [22]	43.2	67.2	65.3	58.1
ICA [18]	69.1	75.1	73.9	23.1
Planetoid* [29]	64.7 (26s)	75.7 (13s)	77.2 (25s)	61.9 (185s)
GCN (this paper)	**70.3** (7s)	**81.5** (4s)	**79.0** (38s)	**66.0** (48s)
GCN (rand. splits)	67.9 ± 0.5	80.1 ± 0.5	78.9 ± 0.7	58.4 ± 1.7

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Spectral graph convolution

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \qquad \tilde{A} = A + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

- Spectral graph filtering

$$g_\theta \star x = U g_{\theta'}(\Lambda) U^T x$$

U : the matrix of eigenvectors of the normalized graph Laplacian

$$L = I_N - \tilde{D}^{-\frac{1}{2}} A \tilde{D}^{-\frac{1}{2}} = U \Lambda U^T$$

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda})$$

Polynomial approximation (In this case, Chebyshev polynomial)

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Spectral graph convolution

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \qquad \tilde{A} = A + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

- Spectral graph filtering

$$g_\theta \star x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x$$

Linear approx.

$$g_\theta \star x \approx \theta'_0 x + \theta'_1 (L - I_N)x = \theta'_0 x + \theta'_1 D^{-1/2}AD^{-1/2}x$$

Use a single parameter $\theta = \theta'_0 = \theta'_1$

$$g_\theta \star x \approx \theta\left(I_N + D^{-1/2}AD^{-1/2}\right)x$$

Renormalization trick

$$I_N + D^{-1/2}AD^{-1/2} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$$

$$g_\theta \star x \approx \theta\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\right)x$$

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

Network analysis
1. Node classification

- Spectral graph convolution

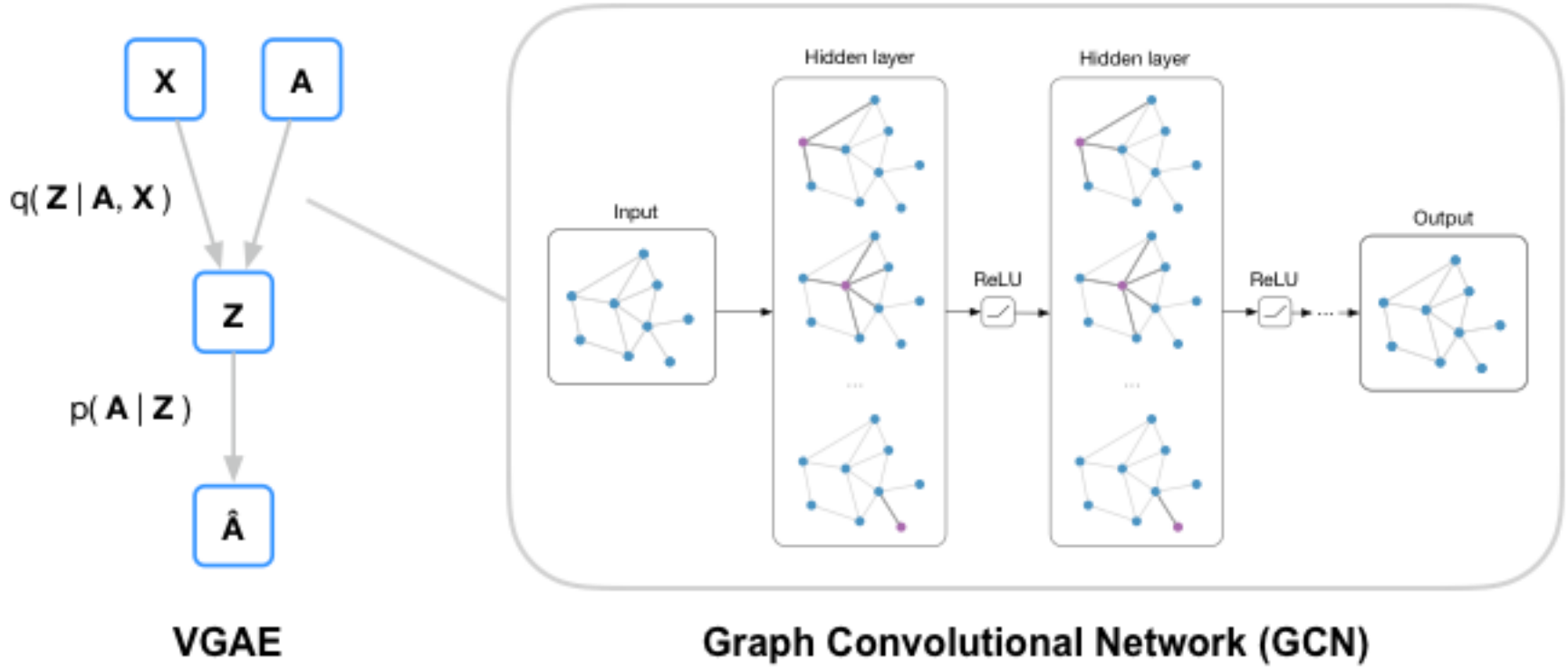
$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right) \qquad \tilde{A} = A + I_N, \quad \tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$$

Table 3: Comparison of propagation models.

Description		Propagation model	Citeseer	Cora	Pubmed
Chebyshev filter (Eq. 5)	$K = 3$	$\sum_{k=0}^{K} T_k(\tilde{L})X\Theta_k$	69.8	79.5	74.4
	$K = 2$		69.6	81.2	73.8
1st-order model (Eq. 6)		$X\Theta_0 + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta_1$	68.3	80.0	77.5
Single parameter (Eq. 7)		$(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}})X\Theta$	69.3	79.2	77.4
Renormalization trick (Eq. 8)		$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta$	**70.3**	**81.5**	**79.0**
1st-order term only		$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}X\Theta$	68.7	80.5	77.8
Multi-layer perceptron		$X\Theta$	46.5	55.1	71.4

Kipf, Thomas N., and Max Welling.
"Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016)
http://tkipf.github.io/graph-convolutional-networks/

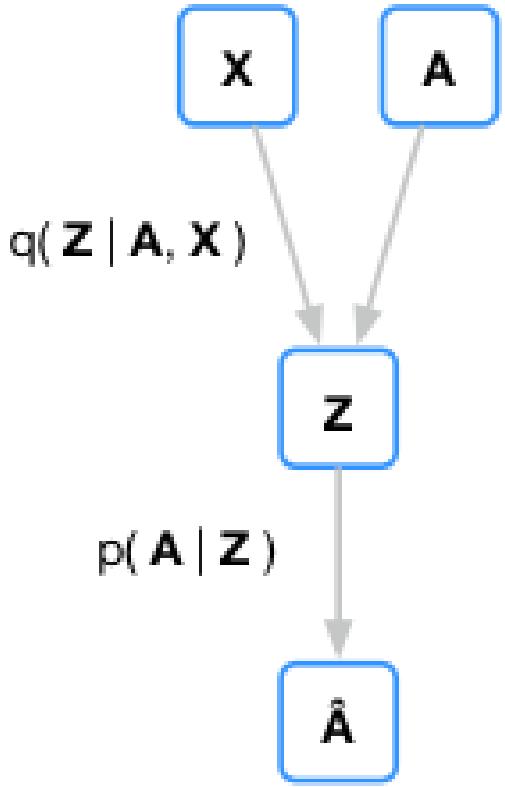
Network analysis
2. Link prediction



- **Clustering**
- **Link prediction**
- **Matrix completion and recommendation**

Kipf, Thomas N., and Max Welling.
"Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Network analysis
2. Link prediction



VGAE

Input graph, $\boldsymbol{G(X, A)}$

↓ **Encoder**

Node features, $q(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})$

↓ **Decoder**

Reconstructed
Adjacency matrix, $\hat{A} = \boldsymbol{\sigma(ZZ^T)}$

Kipf, Thomas N., and Max Welling.
"Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Network analysis
2. Link prediction

- Trained on an incomplete version of {Cora, Citeseer, Pubmed} datasets where parts of the citation links (edges) have been removed, while all node features are kept.
- Form validation and test sets from previously removed edges and the same number of randomly sampled pairs of unconnected nodes (non-edges).

Table 1: Link prediction task in citation networks. See [1] for dataset details.

Method	Cora		Citeseer		Pubmed	
	AUC	AP	AUC	AP	AUC	AP
SC [5]	84.6 ± 0.01	88.5 ± 0.00	80.5 ± 0.01	85.0 ± 0.01	84.2 ± 0.02	87.8 ± 0.01
DW [6]	83.1 ± 0.01	85.0 ± 0.00	80.5 ± 0.02	83.6 ± 0.01	84.4 ± 0.00	84.1 ± 0.00
GAE*	84.3 ± 0.02	88.1 ± 0.01	78.7 ± 0.02	84.1 ± 0.02	82.2 ± 0.01	87.4 ± 0.00
VGAE*	84.0 ± 0.02	87.7 ± 0.01	78.9 ± 0.03	84.1 ± 0.02	82.7 ± 0.01	87.5 ± 0.01
GAE	91.0 ± 0.02	92.0 ± 0.03	89.5 ± 0.04	89.9 ± 0.05	$\mathbf{96.4} \pm 0.00$	$\mathbf{96.5} \pm 0.00$
VGAE	$\mathbf{91.4} \pm 0.01$	$\mathbf{92.6} \pm 0.01$	$\mathbf{90.8} \pm 0.02$	$\mathbf{92.0} \pm 0.02$	94.4 ± 0.02	94.7 ± 0.02

Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." *arXiv preprint arXiv:1611.07308* (2016).
https://github.com/tkipf/gae

Network analysis
3. Matrix completion

- Matrix completion → Can be applied for recommending system

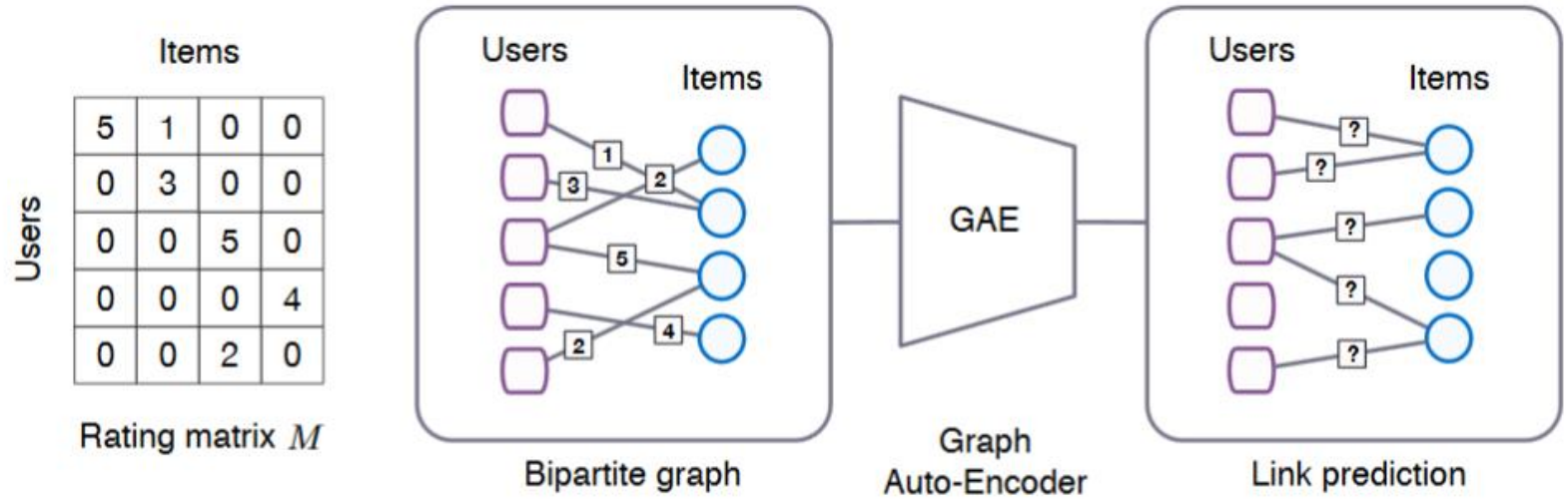


Figure 1: *Left*: Rating matrix M with entries that correspond to user-item interactions (ratings between 1-5) or missing observations (0). *Right*: User-item interaction graph with bipartite structure. Edges correspond to interaction events, numbers on edges denote the rating a user has given to a particular item. The matrix completion task (i.e. predictions for unobserved interactions) can be cast as a link prediction problem and modeled using an end-to-end trainable graph auto-encoder.

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

Items

Users

5	1	0	0
0	3	0	0
0	0	5	0
0	0	0	4
0	0	2	0

Rating matrix M

- A rating matrix M of shape $N_u \times N_v$

  N_u : the number of users, N_v : the number of items
- User i rated item i, or the rating is unobserved ($M_{ij} = 0$).
- Matrix completion problem or recommendation

  → a link prediction problem on a bipartite user-item interaction graph.

Input graph : $\boldsymbol{G(\mathcal{W}, \mathcal{E}, \mathcal{R})}$

- $\boldsymbol{\mathcal{W} = \mathcal{U} \cup \mathcal{V}}$: user nodes $\boldsymbol{u_i} \in \boldsymbol{\mathcal{U}}$, with $i \in \{1, \dots, N_u\}$ and item nodes $\boldsymbol{v_j} \in \boldsymbol{\mathcal{V}}$, with $j \in \{1, \dots, N_v\}$
- $\boldsymbol{(u_i, r, v_j) \in \mathcal{E}}$ represent rating levels, such as $r \in \{1, \dots, R\} = \boldsymbol{\mathcal{R}}$.

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

- GAE for the link prediction task

Take as input an $N \times D$ feature matrix, $\boldsymbol{X}$

$N \times E$ node embedding matrix, $\boldsymbol{Z} = \left[\boldsymbol{z_1^T}, \ldots, \boldsymbol{z_N^T}\right]^{\boldsymbol{T}}$ $\quad \boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{A})$

A graph adjacency matrix, $\boldsymbol{A}$

$$\breve{\boldsymbol{A}} = g(\boldsymbol{Z})$$

which takes pairs of node embeddings $(\boldsymbol{z_i}, \boldsymbol{z_j})$ and predicts respective entries $\breve{\boldsymbol{A}}_{\boldsymbol{ij}}$

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

- GAE for the bipartite recommender graphs, $\boldsymbol{G(\mathcal{W}, \mathcal{E}, \mathcal{R})}$

Encoder

$[U, V] = f(X, M_1, \ldots, M_R)$, where $M_r \in \{0,1\}^{N_u \times N_v}$ is the adjacency matrix associated with rating type $r \in \mathcal{R}$

U, V : matrices of user and item embeddings with shape $N_u \times E$ and $N_v \times E$, respectively.

Decoder

$\breve{M} = g(U, V)$, rating matrix $\breve{M}$ of shape $N_u \times N_v$

$G(\boldsymbol{X}, \boldsymbol{A}) \longrightarrow \boldsymbol{Z} = f(\boldsymbol{X}, \boldsymbol{A}) \longrightarrow \breve{\boldsymbol{A}} = g(\boldsymbol{Z})$ GAE for the link prediction task

$G(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{R}}) \longrightarrow [\boldsymbol{U}, \boldsymbol{V}] = f(\boldsymbol{X}, \boldsymbol{M_1}, \ldots, \boldsymbol{M_R}) \longrightarrow \breve{\boldsymbol{M}} = g(\boldsymbol{U}, \boldsymbol{V})$ GAE for the bipartite recommender

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

- GAE for the bipartite recommender graphs, $\boldsymbol{G(\mathcal{W}, \mathcal{E}, \mathcal{R})}$

$$G(\mathcal{W}, \mathcal{E}, \mathcal{R}) \longrightarrow [U, V] = f(X, M_1, \ldots, M_R) \longrightarrow \check{M} = g(U, V)$$

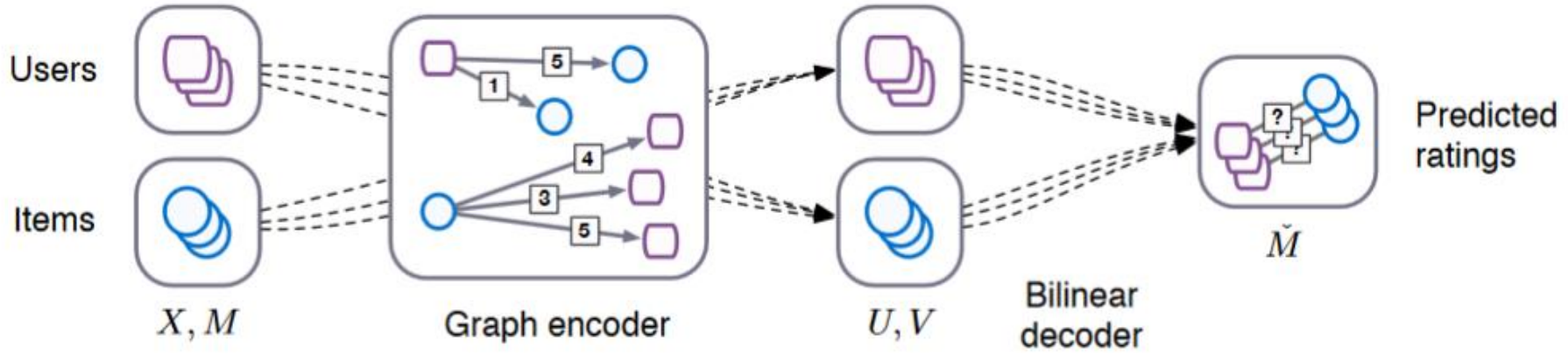


Figure 2: Schematic of a forward-pass through the GC-MC model, which is comprised of a graph convolutional encoder $[U, V] = f(X, M_1, \ldots, M_R)$ that passes and transforms messages from user to item nodes, and vice versa, followed by a bilinear decoder model that predicts entries of the (reconstructed) rating matrix $\check{M} = g(U, V)$, based on pairs of user and item embeddings.

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

Encoder

$u_i = \sigma(Wh_i)$: the final user embedding

$h_i = \sigma\left[accum\left(\sum_{j \in \mathcal{N}_i,1} \mu_{j \to i,1}, \ldots, \sum_{j \in \mathcal{N}_i,R} \mu_{j \to i,R}\right)\right]$: intermediate node state

$\mu_{j \to i,r} = \frac{1}{c_{ij}} W_r x_j$: Message function from item j to user i $\quad c_{ij} = \sqrt{|\mathcal{N}_i||\mathcal{N}_j|}$

Berg, Rianne van den, Thomas N. Kipf, and Max Welling.
"Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

Decoder

$$p(\breve{M}_{ij} = r) = \frac{e^{u_i^T Q_r v_j}}{\sum_{s \in R} e^{u_i^T Q_s v_j}}$$: probability that rating $\breve{M}_{ij}$ is r

$$\breve{M}_{ij} = g(u_i, v_j) = \mathbb{E}_{p(\breve{M}_{ij}=r)}[r] = \sum_{r \in R} r \cdot p(\breve{M}_{ij} = r)$$: expected rating

Loss function

$$\mathcal{L} = -\sum_{i,j}\sum_{r=1}^{R} I[r = M_{ij}] \log p(\breve{M}_{ij} = r)$$

$I[k = l] = 1$, when $k = l$ and 0 otherwise

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

Network analysis
3. Matrix completion

Dataset	Users	Items	Features	Ratings	Density	Rating levels
Flixster	3,000	3,000	Users/Items	26,173	0.0029	0.5, 1, . . . , 5
Douban	3,000	3,000	Users	136,891	0.0152	1, 2, . . . , 5
YahooMusic	3,000	3,000	Items	5,335	0.0006	1, 2, . . . , 100
MovieLens 100K (ML-100K)	943	1,682	Users/Items	100,000	0.0630	1, 2, . . . , 5
MovieLens 1M (ML-1M)	6,040	3,706	—	1,000,209	0.0447	1, 2, . . . , 5
MovieLens 10M (ML-10M)	69,878	10,677	—	10,000,054	0.0134	0.5, 1, . . . , 5

Table 1: Number of users, items and ratings for each of the MovieLens datasets used in our experiments. We further indicate rating density and rating levels.

Model	ML-100K + Feat
MC [3]	0.973
IMC [11, 31]	1.653
GMC [12]	0.996
GRALS [25]	0.945
sRGCNN [22]	0.929
GC-MC (Ours)	0.910
GC-MC+Feat	**0.905**

Model	Flixster	Douban	YahooMusic
GRALS	1.313/1.245	0.833	38.0
sRGCNN	1.179/0.926	0.801	22.4
GC-MC	**0.941/0.917**	**0.734**	**20.5**

Berg, Rianne van den, Thomas N. Kipf, and Max Welling. "Graph convolutional matrix completion." *arXiv preprint arXiv:1706.02263* (2017).

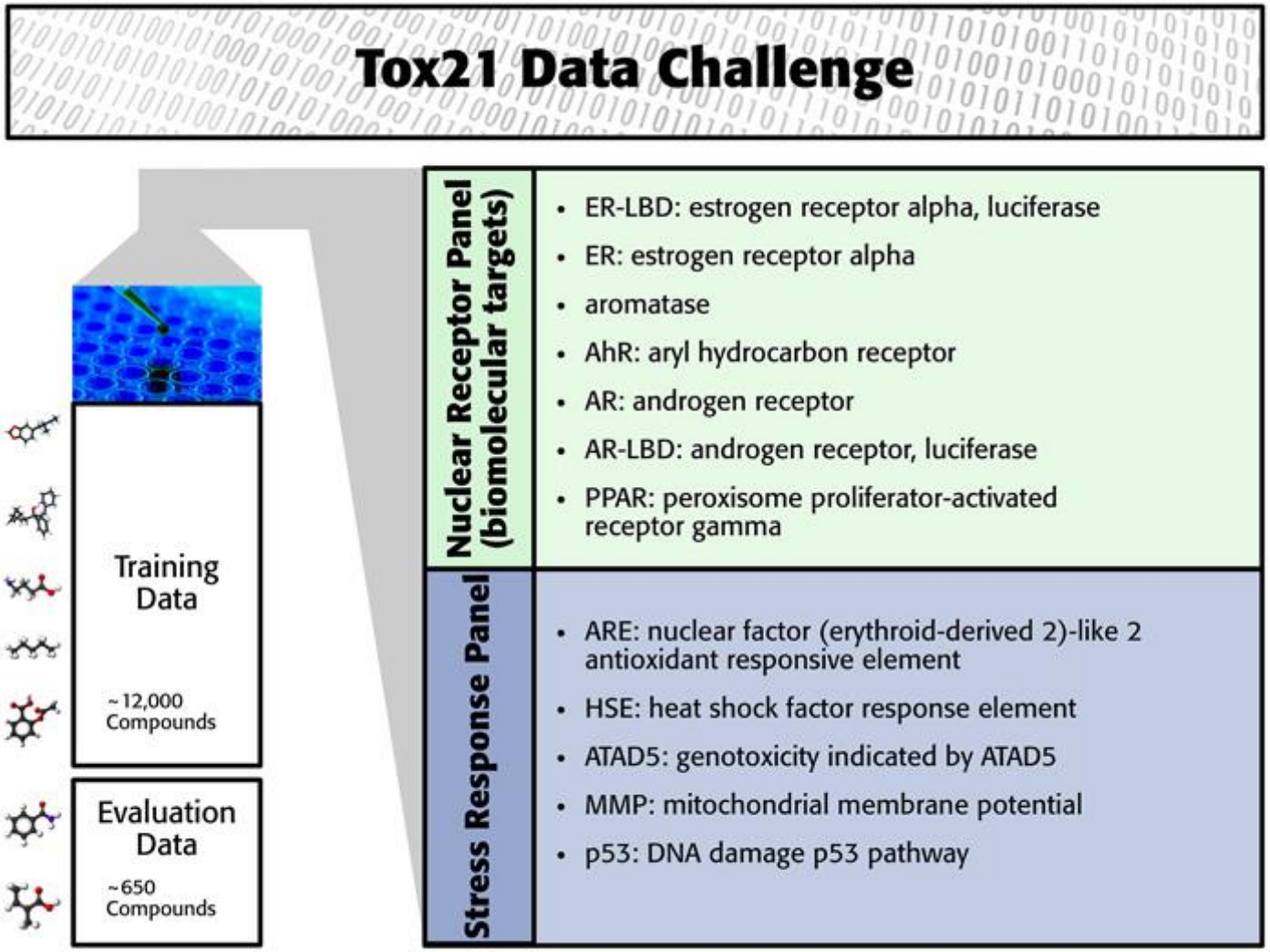
Molecular applications
: Which kinds of datasets exist?

- Bioactive molecules with drug-like properties
- ~1,828,820 compounds
- https://www.ebi.ac.uk/chembldb/

- ✓ Drugs and targets
- ✓ FDA approved, investigational, experimental, …
- ✓ 7,713 (all drugs), 4,115 (targets), …
- ✓ https://drugbank.ca/

- ▪ Drugs and targets
- ▪ FDA approved, investigational, experimental, …
- ▪ 7,713 (all drugs), 4,115 (targets), …
- ▪ https://drugbank.ca/

Molecular applications
: Which kinds of datasets exist?

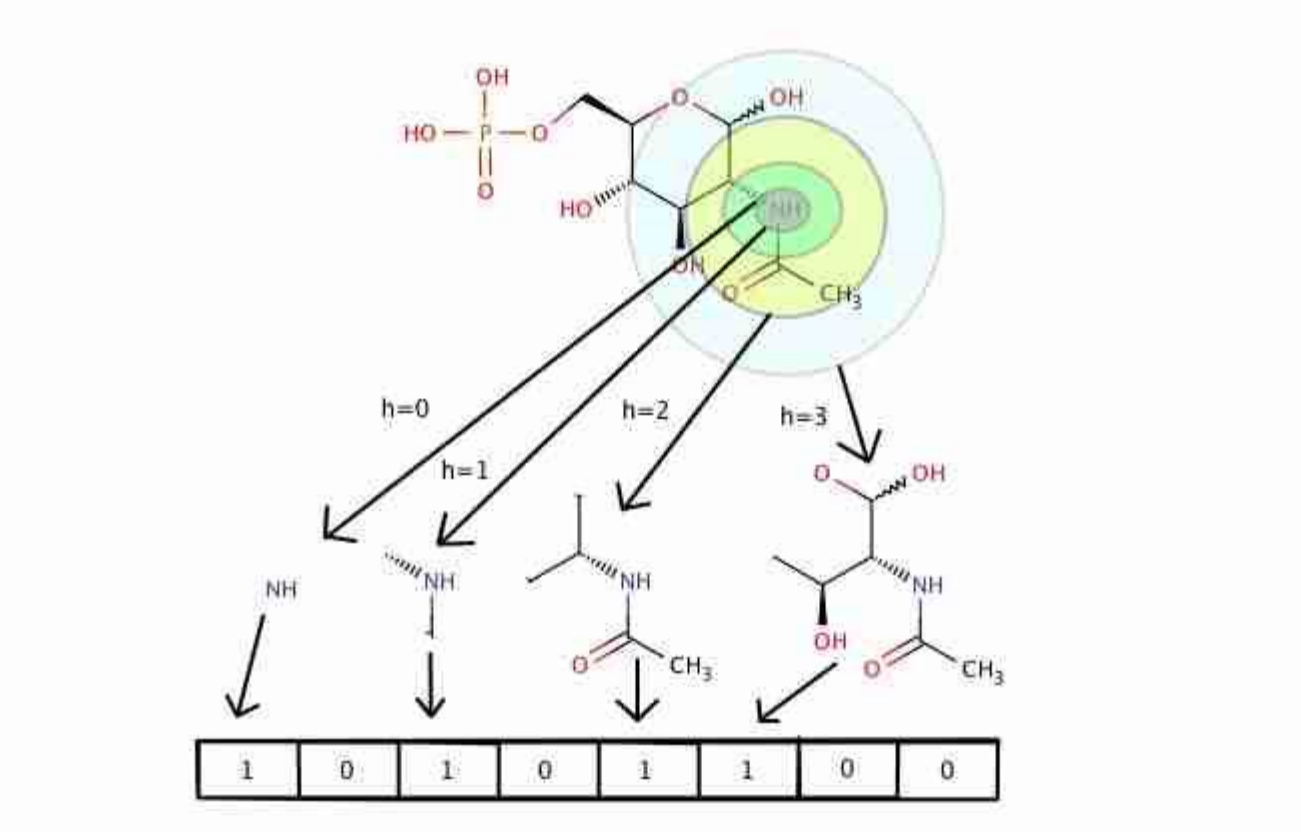


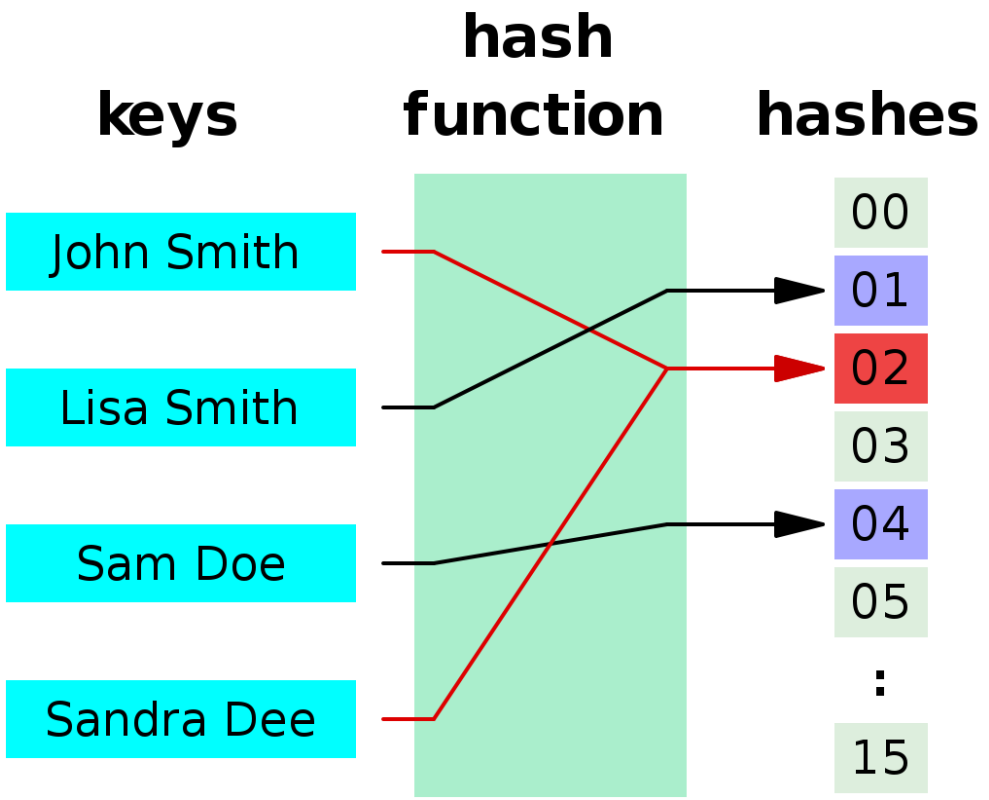
Tox21 Data Challenge (@ Kaggle)

- 12 types of toxicity
- Molecule species (represented with SMILES) and toxicity labels are given
- But too **small** to train a DL model

https://tripod.nih.gov/tox21/challenge/data.jsp

Molecular applications
1. Neural molecular fingerprint



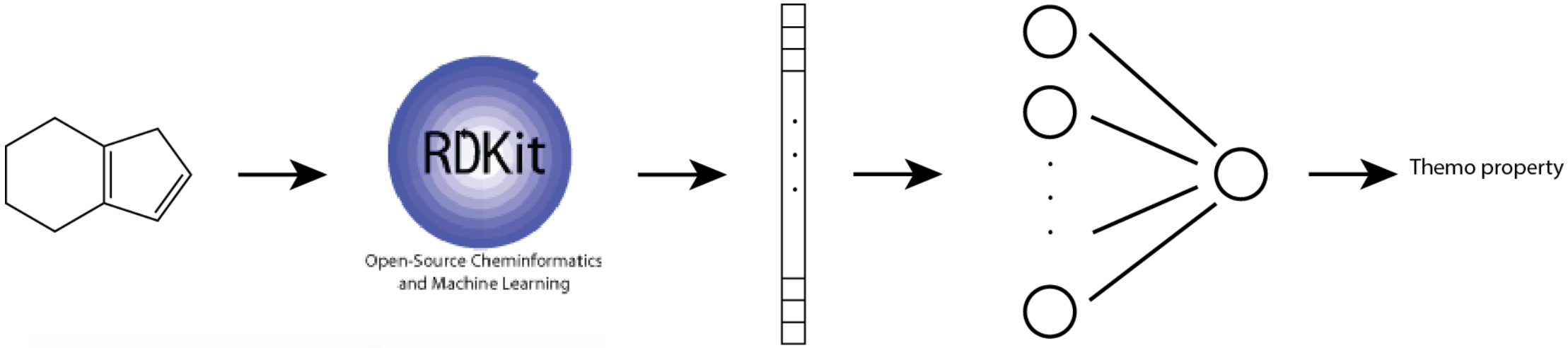


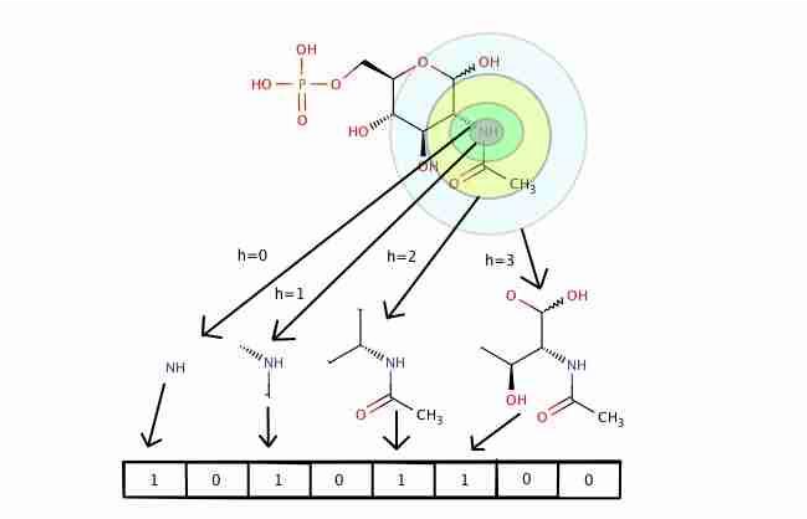
Hash function have been used to generate molecular fingerprints.
* Molecular fingerprint : a vector representation of molecular substructures

https://mc.ai/machine-learning-for-drug-discovery-in-a-nutshell%E2%80%8A-%E2%80%8Apart-ii/

Molecular applications
1. Neural molecular fingerprint

Such molecular fingerprints can be easily obtained by open source packages, e.g.) RDKit.

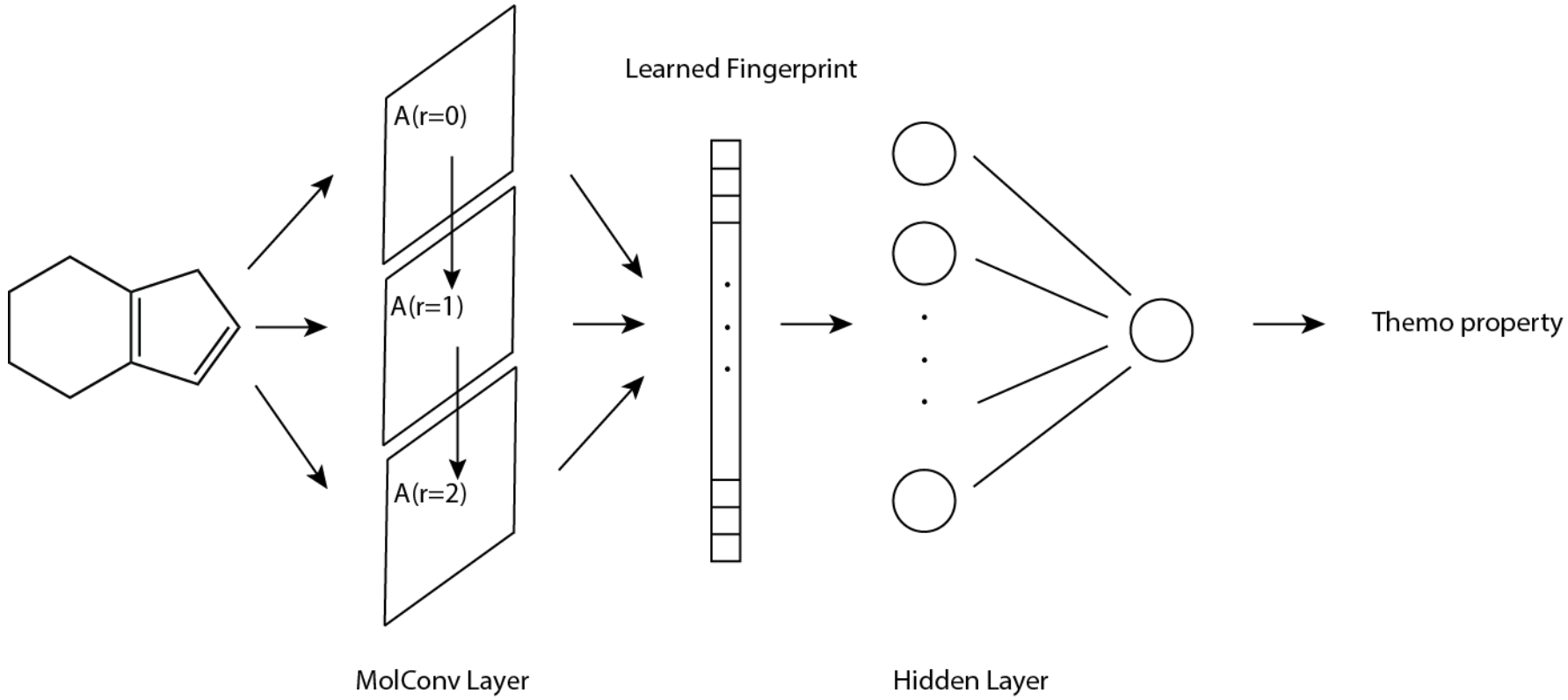




http://kehang.github.io/basic_project/2017/04/18/machine-learning-in-molecular-property-prediction/

Molecular applications
1. Neural molecular fingerprint

In recent days, neural fingerprints generated by graph convolutional network is widely used for more accurate molecular property predictions.



http://kehang.github.io/basic_project/2017/04/18/machine-learning-in-molecular-property-prediction/

Molecular applications
1. Neural molecular fingerprint

Algorithm 1 Circular fingerprints

Input: molecule, radius R, fingerprint length S
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ ▷ concatenate
    $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ ▷ hash function
    $i \leftarrow \text{mod}(r_a, S)$ ▷ convert to index
    $\mathbf{f}_i \leftarrow 1$ ▷ Write 1 at index
Return: binary vector $\mathbf{f}$

Algorithm 2 Neural graph fingerprints

Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

Figure 2: Pseudocode of circular fingerprints (*left*) and neural graph fingerprints (*right*). Differences are highlighted in blue. Every non-differentiable operation is replaced with a differentiable analog.

Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.

Molecular applications
1. Neural molecular fingerprint

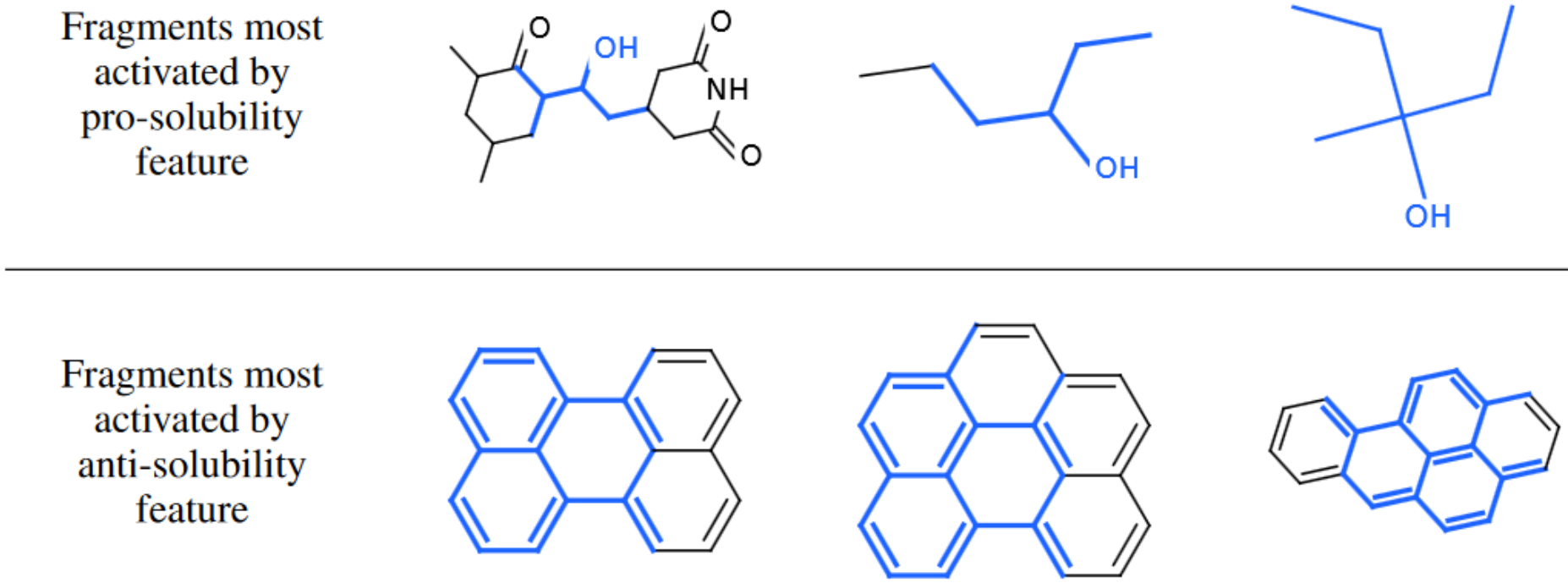
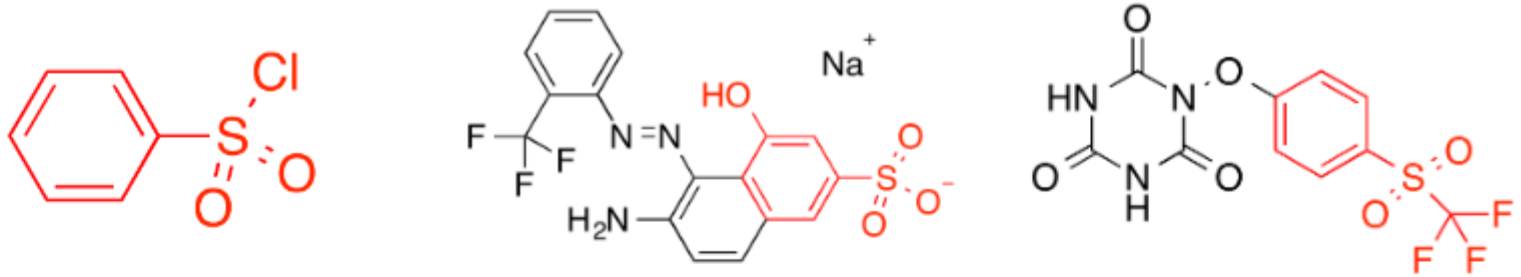


Figure 4: Examining fingerprints optimized for predicting solubility. Shown here are representative examples of molecular fragments (highlighted in blue) which most activate different features of the fingerprint. *Top row:* The feature most predictive of solubility. *Bottom row:* The feature most predictive of insolubility.

Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.

Molecular applications
1. Neural molecular fingerprint

Fragments most activated by toxicity feature on SR-MMP dataset



Fragments most activated by toxicity feature on NR-AHR dataset

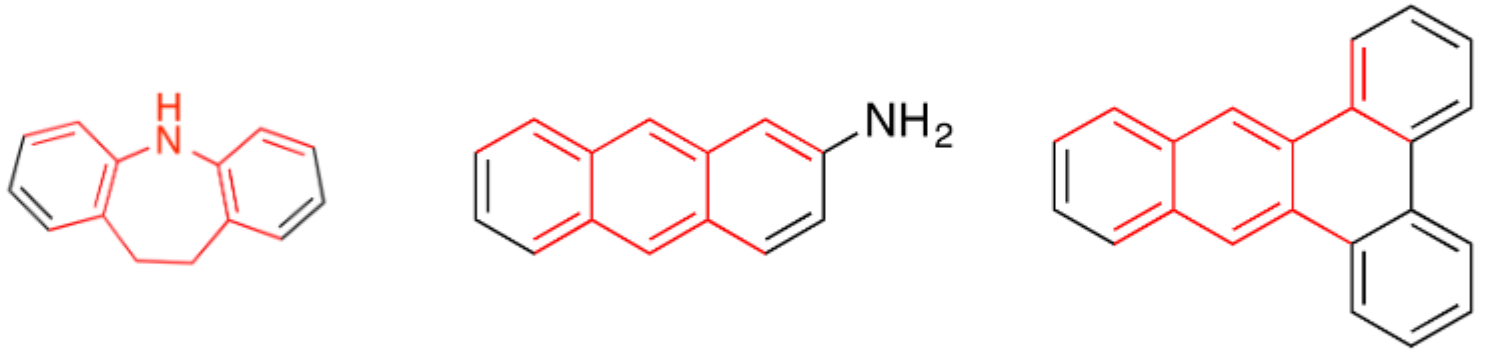


Figure 5: Visualizing fingerprints optimized for predicting toxicity. Shown here are representative samples of molecular fragments (highlighted in red) which most activate the feature most predictive of toxicity. *Top row:* the most predictive feature identifies groups containing a sulphur atom attached to an aromatic ring. *Bottom row:* the most predictive feature identifies fused aromatic rings, also known as polycyclic aromatic hydrocarbons, a well-known carcinogen.

Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.

Molecular applications
1. Neural molecular fingerprint

Dataset Units	Solubility [4] log Mol/L	Drug efficacy [5] EC_{50} in nM	Photovoltaic efficiency [8] percent
Predict mean	4.29 ± 0.40	1.47 ± 0.07	6.40 ± 0.09
Circular FPs + linear layer	1.71 ± 0.13	$\mathbf{1.13 \pm 0.03}$	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	$\mathbf{1.15 \pm 0.02}$	2.58 ± 0.18
Neural FPs + neural net	$\mathbf{0.52 \pm 0.07}$	$\mathbf{1.16 \pm 0.03}$	$\mathbf{1.43 \pm 0.09}$

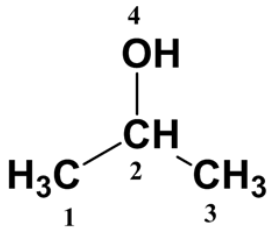
Table 1: Mean predictive accuracy of neural fingerprints compared to standard circular fingerprints.

Duvenaud, David K., et al.
"Convolutional networks on graphs for learning molecular fingerprints." *Advances in neural information processing systems*. 2015.

Molecular applications

2. Quantitative Structure-Property Relationships (QSPR)

(a)

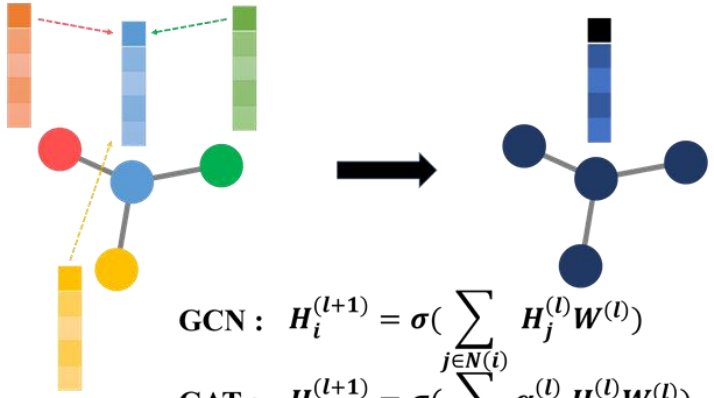


$$X_1 = \begin{bmatrix} 6 \\ 3 \\ 4 \\ 0 \end{bmatrix} \quad \cdots \quad X_4 = \begin{bmatrix} 8 \\ 1 \\ 4 \\ 0 \end{bmatrix}$$

Atom type
# of Hs.
Valence
Aromaticity

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

(b)

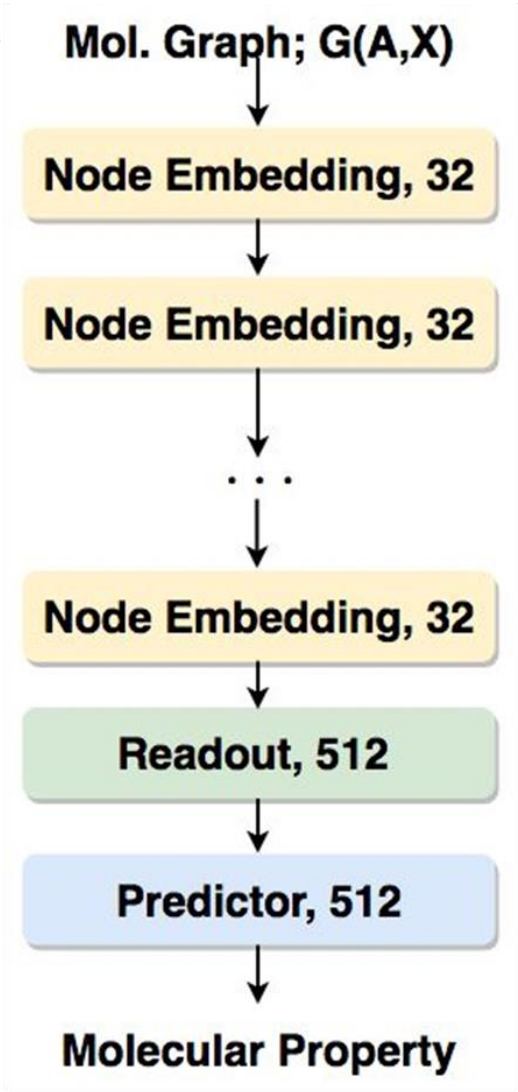


$$\text{GCN}: \quad H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} H_j^{(l)} W^{(l)}\right)$$

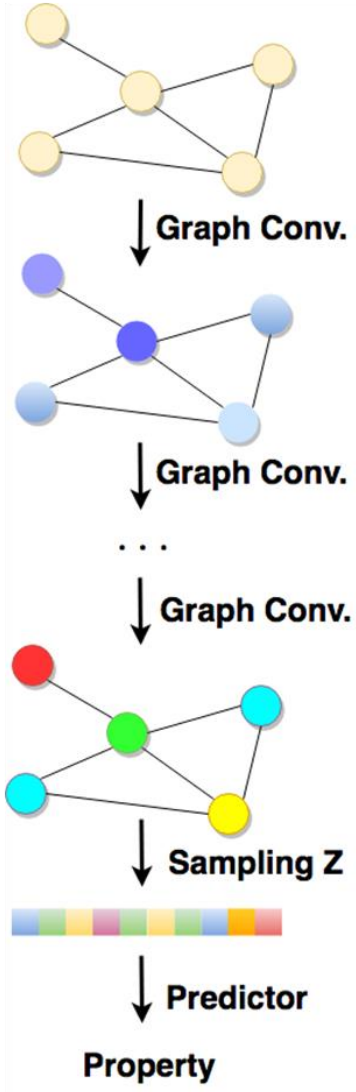
$$\text{GAT}: \quad H_i^{(l+1)} = \sigma\left(\sum_{j \in N(i)} \alpha_{ij}^{(l)} H_j^{(l)} W^{(l)}\right)$$

$$\text{GGNN}: \quad H_i^{(l+1)} = GRU\left(H_i^{(l)} W^{(l)}, \sum_{j \in N(i)} H_j^{(l)} W^{(l)}\right)$$

(c)



(d)



Ryu, Seongok, Jaechang Lim, and Woo Youn Kim. "Deeply learning molecular structure-property relationships using graph attention neural network." *arXiv preprint arXiv:1805.10988* (2018).

Molecular applications
2. Quantitative Structure-Property Relationships (QSPR)

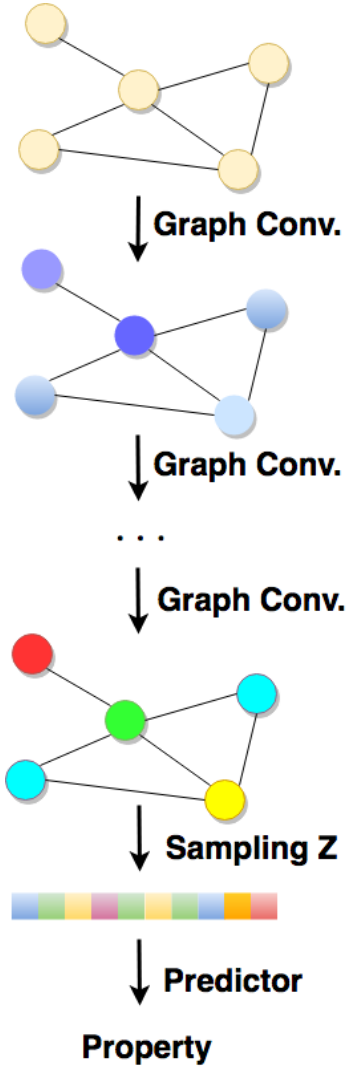
Prediction results of various molecular properties

Model	logP	TPSA	Atomization energy (kcal/mol)	PVE (%)
GAT	**0.019**	**0.088**	**4.12**	**0.63**
GCN	0.073	0.75	6.09	0.89
Previous works	Graph - 0.05[18] SMILES - 0.13[18]	-	-	Graph -1.43[11]

Table 1. Mean absolute error of the prediction results of molecular properties. The best ones are shown in bold.

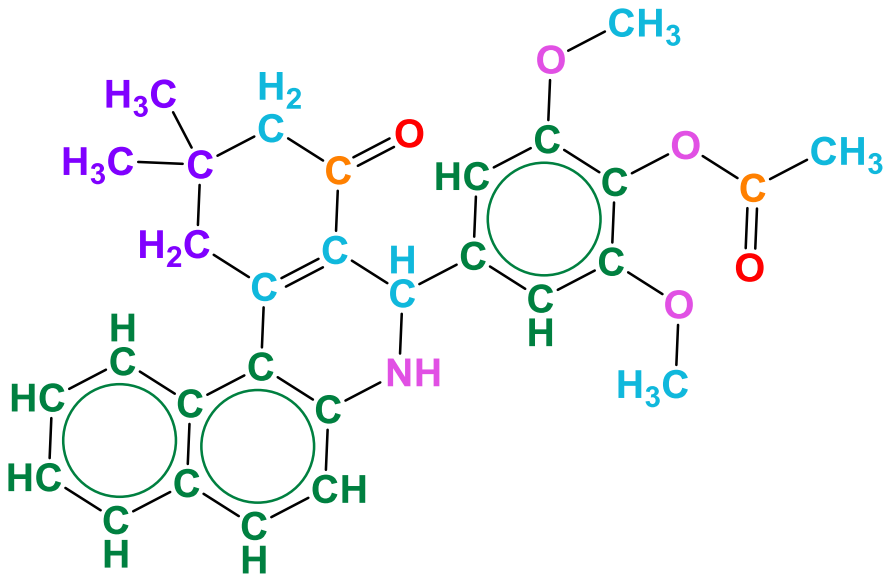
Ryu, Seongok, Jaechang Lim, and Woo Youn Kim. "Deeply learning molecular structure-property relationships using graph attention neural network." *arXiv preprint arXiv:1805.10988* (2018).

Molecular applications
2. Quantitative Structure-Property Relationships (QSPR)



Final node states

Learning solubility of molecules

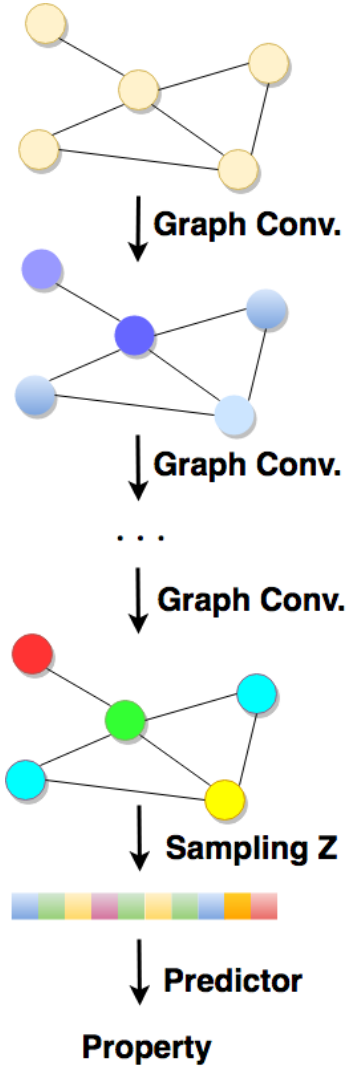


The neural network recognizes
several functional groups differently

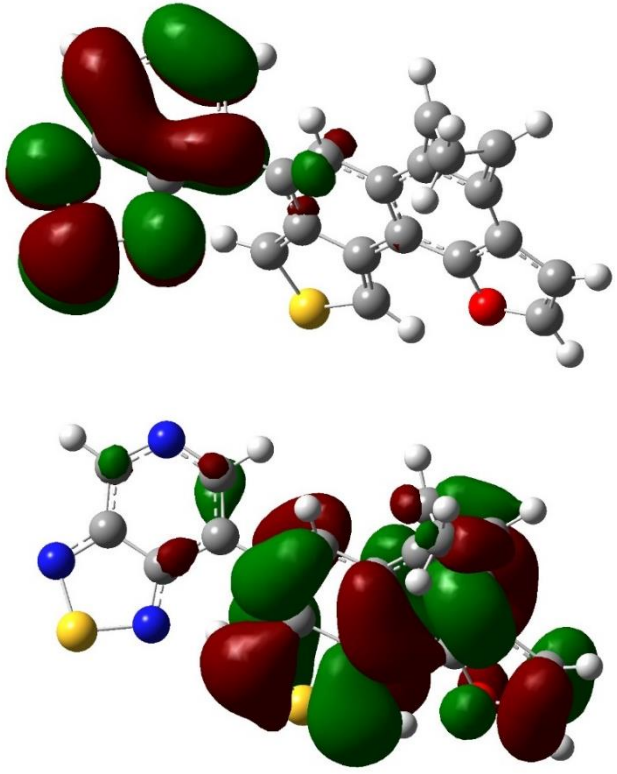
Ryu, Seongok, Jaechang Lim, and Woo Youn Kim. "Deeply learning molecular structure-property relationships using graph attention neural network." *arXiv preprint arXiv:1805.10988* (2018).

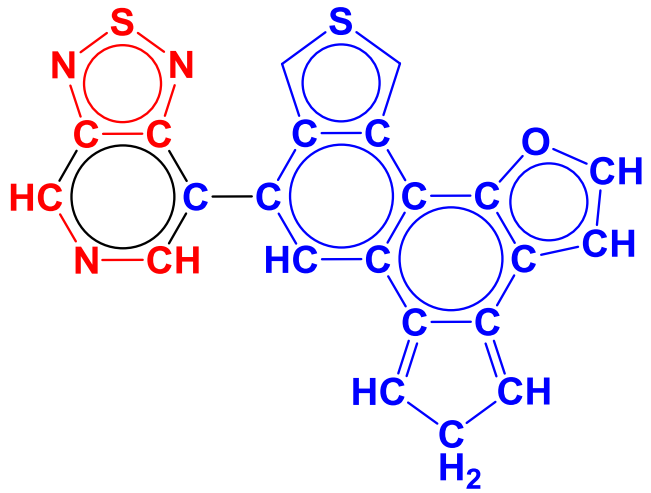
Molecular applications
2. Quantitative Structure-Property Relationships (QSPR)

Final node states



Learning photovoltaic efficiency (QM phenomena)



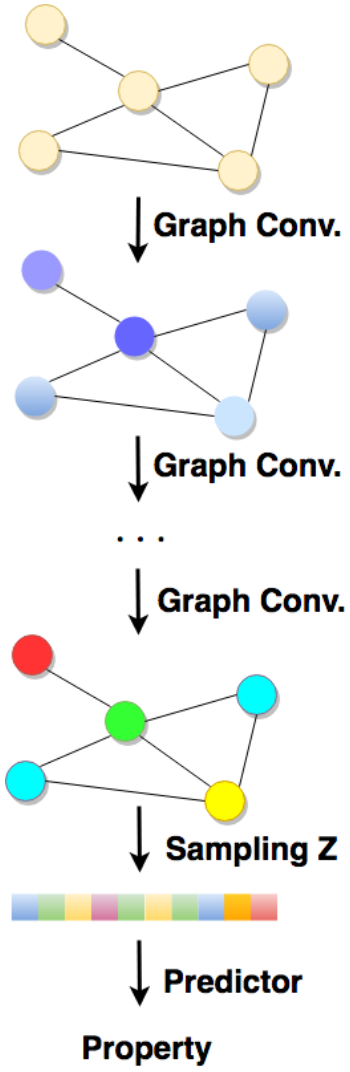


Interestingly, The NN also can differentiate nodes according to the **quantum mechanical characteristics**.

Ryu, Seongok, Jaechang Lim, and Woo Youn Kim. "Deeply learning molecular structure-property relationships using graph attention neural network." *arXiv preprint arXiv:1805.10988* (2018).

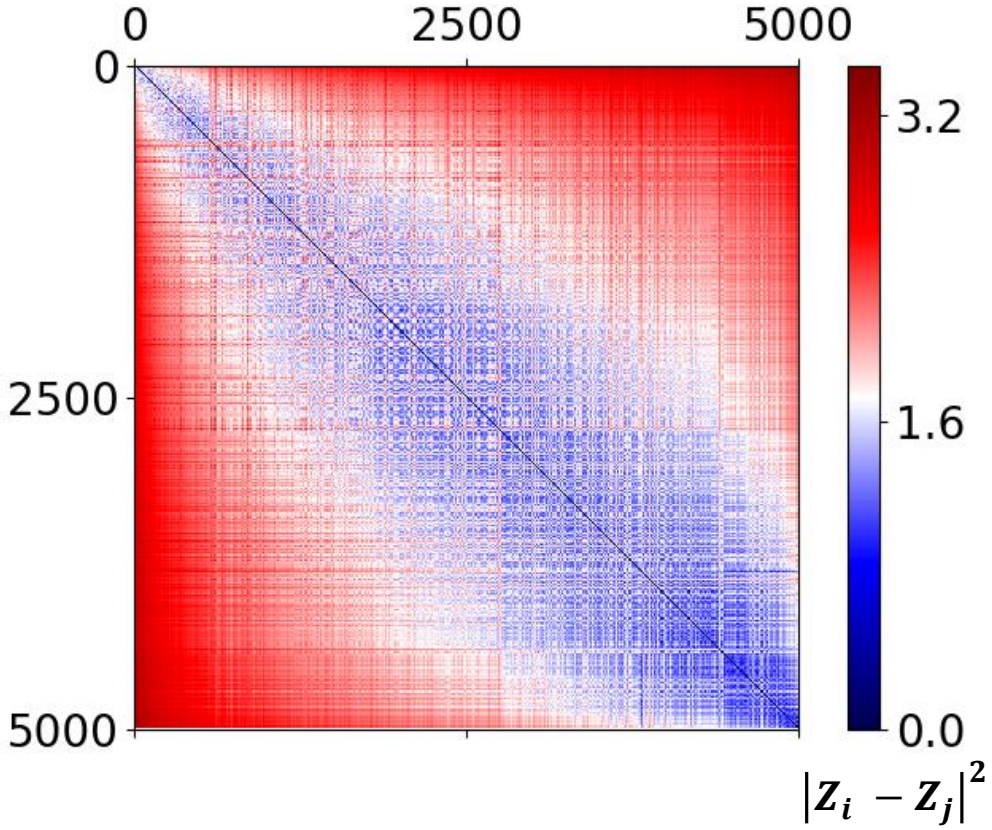
Molecular applications

2. Quantitative Structure-Property Relationships (QSPR)



Graph features

Learning solubility of molecules

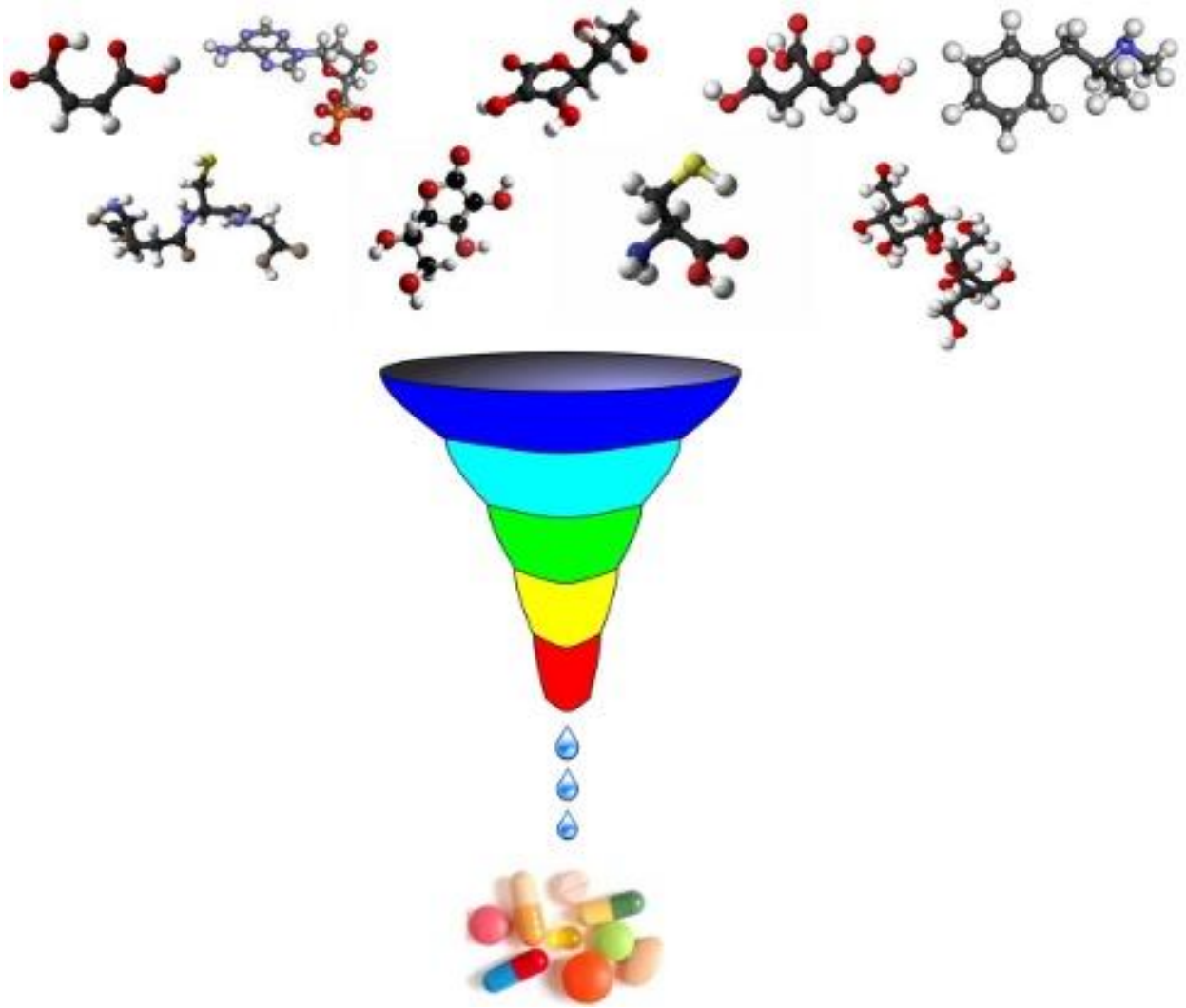


Ascending order

Similar molecules are located closely in the graph latent space

Molecular applications
3. Molecular generative model

Motivation : *de novo* molecular design

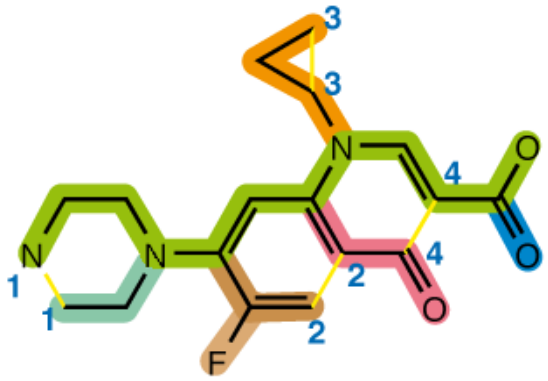


- Chemical space is too huge
  : only **10^8 molecules** have been synthesized as potential drug candidates, whereas it is estimated that there are **10^{23} to 10^{60} molecules**.

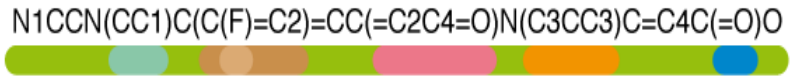
- Limitation of virtual screening

Molecular applications
3. Molecular generative model

Motivation : *de novo* molecular design



Molecule Graph

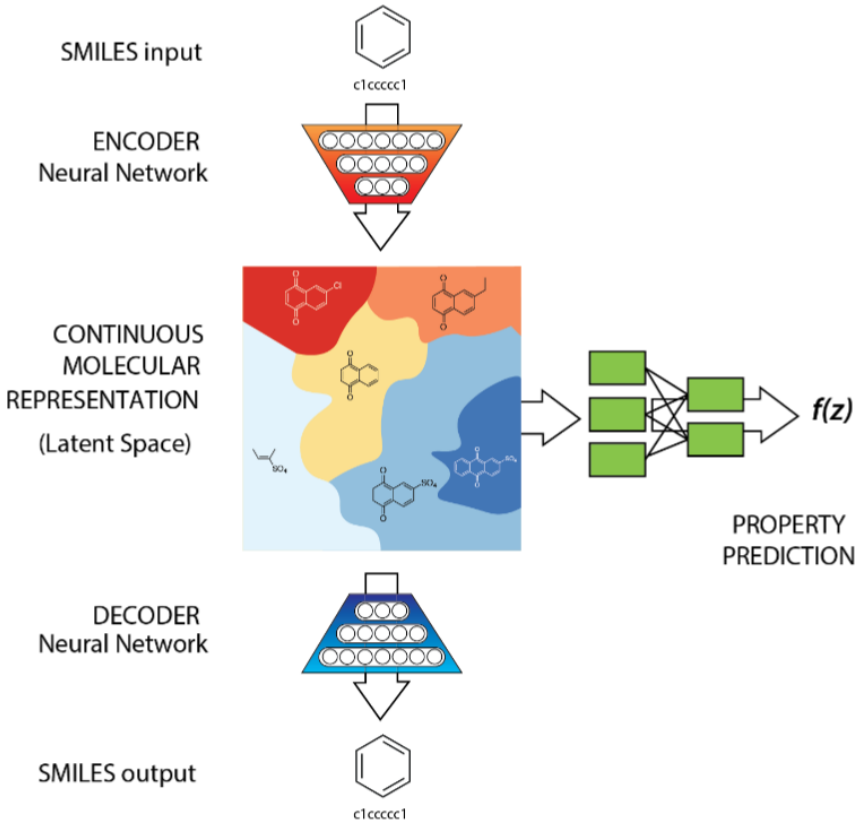


Simplified **M**olecule **L**ine-**E**ntry **S**ystem
(SMILES)

Molecules can be represented as strings according to defined rules

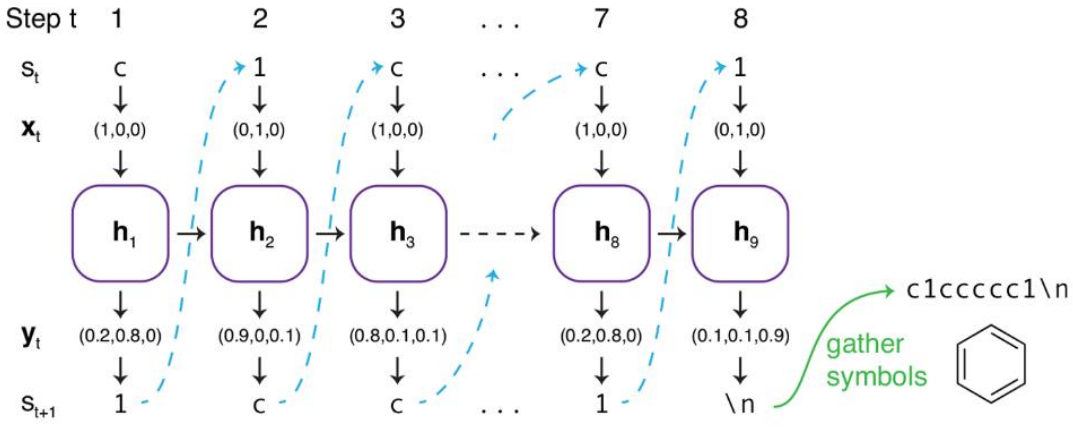
Molecular applications
3. Molecular generative model

Motivation : *de novo* molecular design



Gómez-Bombarelli, Rafael, et al. "Automatic chemical design using a data-driven continuous representation of molecules." *ACS central science* 4.2 (2018): 268-276.

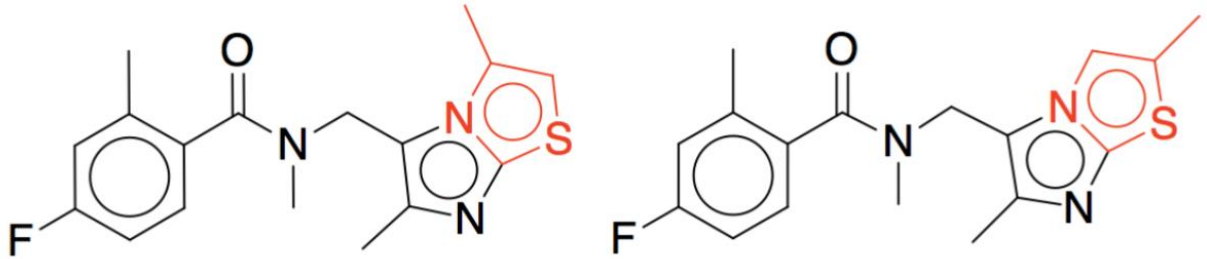
Many SMILES-based generative models exist



Segler, Marwin HS, et al. "Generating focused molecule libraries for drug discovery with recurrent neural networks." *ACS central science* 4.1 (2017): 120-131.

Molecular applications
3. Molecular generative model

Motivation : *de novo* molecular design



Cc1cn2c(CN(C)C(=O)c3ccc(F)cc3C)c(C)nc2s1
Cc1cc(F)ccc1C(=O)N(C)Cc1c(C)nc2scc(C)n12

Figure 1. Two almost identical molecules with markedly different canonical SMILES in RDKit. The edit distance between two strings is 22 (50.5% of the whole sequence).

SMILES representation also has a fatal problem that

small changes of structure can lead to quite different expressions.

→ Difficult to reflect topological information of molecules

Molecular applications
3. Molecular generative model

Literatures

- Li, Y., Vinyals, O., Dyer, C., Pascanu, R., & Battaglia, P. (2018). Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324.
- Jin, Wengong, Regina Barzilay, and Tommi Jaakkola. "Junction Tree Variational Autoencoder for Molecular Graph Generation." *arXiv preprint arXiv:1802.04364* (2018).
- Constrained Graph Variational Autoencoders for Molecule Design." arXiv preprint arXiv:1805.09076 (2018).
- "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders." arXiv preprint arXiv:1802.03480 (2018).
- De Cao, Nicola, and Thomas Kipf. "MolGAN: An implicit generative model for small molecular graphs." arXiv preprint arXiv:1805.11973 (2018).
- …

Molecular applications
3. Molecular generative model

Learning Deep Generative Models of Graphs

DeepMind

Yujia Li [1] **Oriol Vinyals** [1] **Chris Dyer** [1] **Razvan Pascanu** [1] **Peter Battaglia** [1]

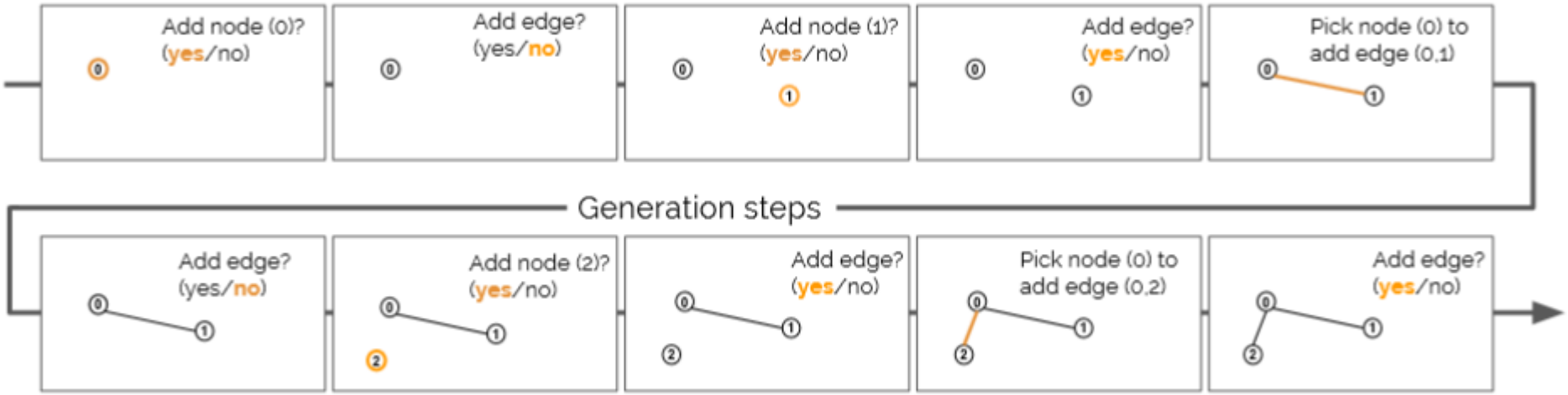


Figure 1. Depiction of the steps taken during the generation process.

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

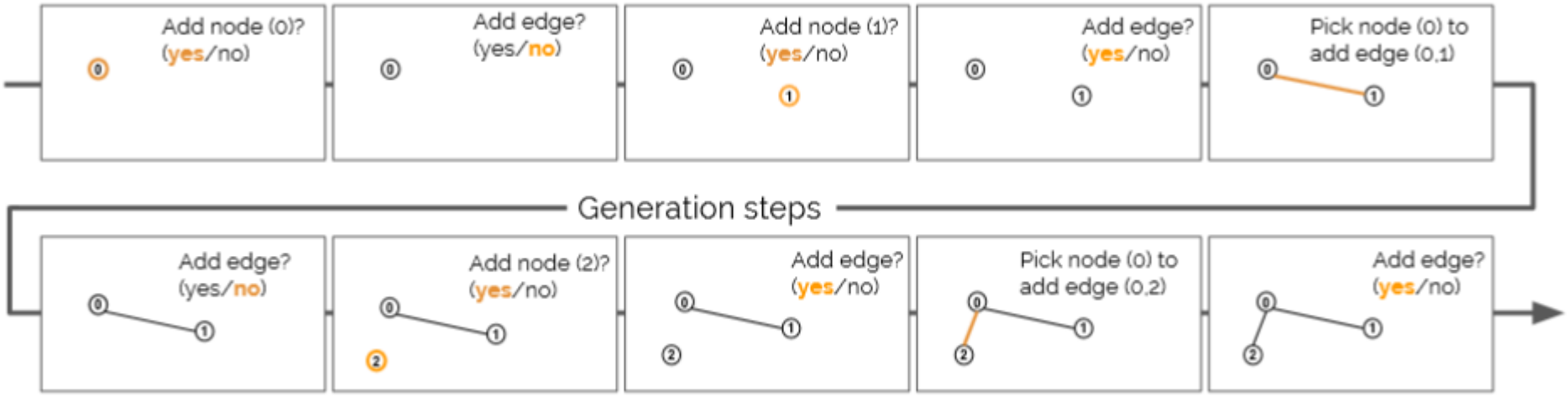


Figure 1. Depiction of the steps taken during the generation process.

1. Sample **whether to add a new node** of a particular type or terminate : if a node type is chosen
2. **Add a node** of this type to the graph
3. Check if **any further edges are needed to connect** the new node to the existing graph
4. If yes, select a node in the graph and **add an edge** connecting the new to the selected node.

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

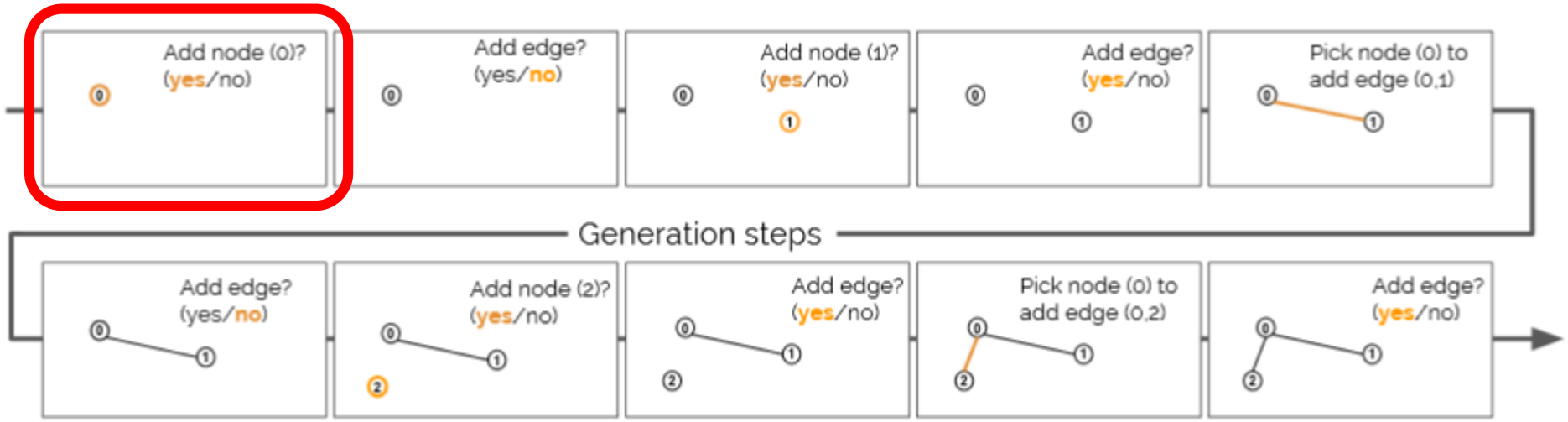


Figure 1. Depiction of the steps taken during the generation process.

- Determine that add a node or not

$$\mathbf{p}_t^{addnode} \leftarrow f_{addnode}(G_{t-1}) \quad \triangleright \text{ Probabilities of each node type and STOP for next node}$$
$$v_t \sim \text{Categorical}(\mathbf{p}_t^{addnode}) \quad \triangleright \text{ Sample next node type or STOP}$$

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

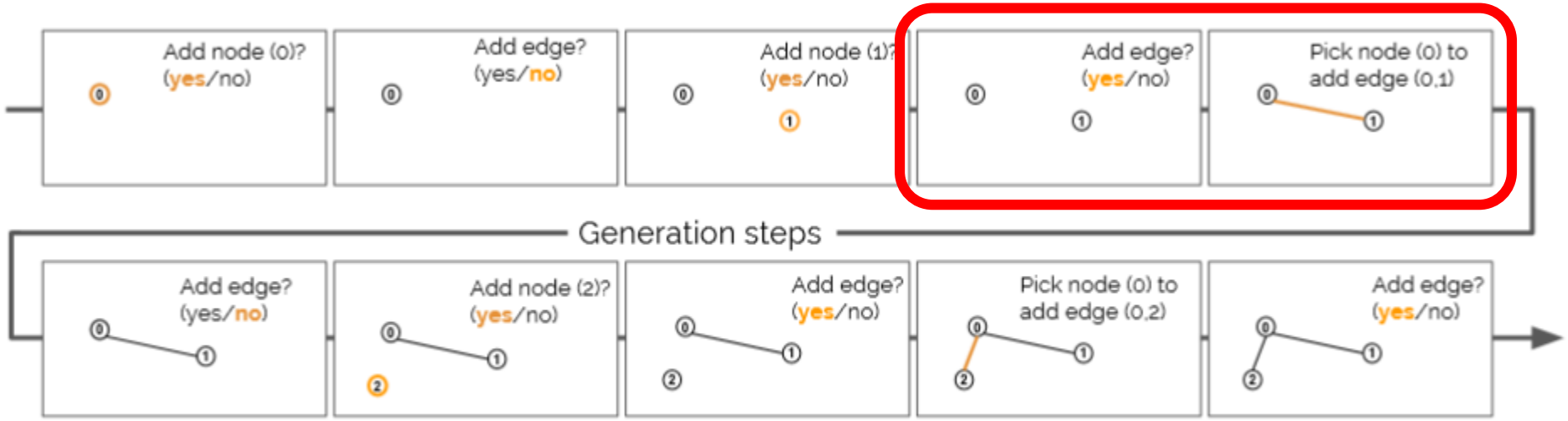


Figure 1. Depiction of the steps taken during the generation process.

- If a node is added, determine that add edges between current node and other nodes.

$$p_{t,i}^{addedge} \leftarrow f_{addedge}((V_t, E_{t,0}), v_t) \quad \triangleright \text{ Probability of adding an edge to } v_t$$
$$z_{t,i} \sim \text{Bernoulli}(p_{t,i}^{addedge}) \quad \triangleright \text{ Sample whether to add an edge to } v_t$$

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

Algorithm 1 Generative Process for Graphs

$E_0 = \phi, V_0 = \phi, G_0 = (V_0, E_0), t = 1$ ▷ Initial graph is empty
$\mathbf{p}_t^{addnode} \leftarrow f_{addnode}(G_{t-1})$ ▷ Probabilities of initial node type and STOP
$v_t \sim \text{Categorical}(\mathbf{p}_t^{addnode})$ ▷ Sample initial node type or STOP
while $v_t \neq$ STOP **do**
  $V_t \leftarrow V_{t-1} \cup \{v_t\}$ ▷ Incorporate node v_t
  $E_{t,0} \leftarrow E_{t-1}, i \leftarrow 1$
  $p_{t,i}^{addedge} \leftarrow f_{addedge}((V_t, E_{t,0}), v_t)$ ▷ Probability of adding an edge to v_t
  $z_{t,i} \sim \text{Bernoulli}(p_{t,i}^{addedge})$ ▷ Sample whether to add an edge to v_t
  while $z_{t,i} = 1$ **do** ▷ Add edges pointing to new node v_t
    $\mathbf{p}_{t,i}^{nodes} \leftarrow f_{nodes}((V_t, E_{t,i-1}), v_t)$ ▷ Probabilities of selecting each node in V_t
    $v_{t,i} \sim \text{Categorical}(\mathbf{p}_{t,i}^{nodes})$
    $E_{t,i} \leftarrow E_{t,i-1} \cup \{(v_{t,i}, v_t)\}$ ▷ Incorporate edge $v_t - v_{t,i}$
    $i \leftarrow i + 1$
    $p_{t,i}^{addedge} \leftarrow f_{addedge}((V_t, E_{t,i-1}), v_t)$ ▷ Probability of adding another edge
    $z_{t,i} \sim \text{Bernoulli}(p_{t,i}^{addedge})$ ▷ Sample whether to add another edge to v_t
  end while
  $E_t \leftarrow E_{t,i-1}$
  $G_t \leftarrow (V_t, E_t)$
  $t \leftarrow t + 1$
  $\mathbf{p}_t^{addnode} \leftarrow f_{addnode}(G_{t-1})$ ▷ Probabilities of each node type and STOP for next node
  $v_t \sim \text{Categorical}(\mathbf{p}_t^{addnode})$ ▷ Sample next node type or STOP
end while
return G_t

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

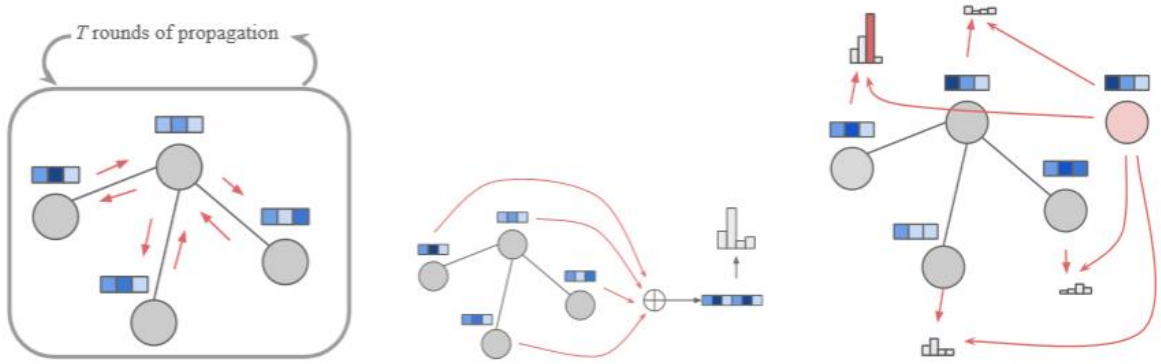


Figure 2. Illustration of the graph propagation process (left), graph level predictions using $f_{addnode}$ and $f_{addedge}$ (center), and node selection f_{nodes} modules (right).

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G) \quad (5)$$

$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G) \quad (6)$$

$$f_{addnode}(G) = \text{softmax}(f_{an}(\mathbf{h}_G)) \quad (7)$$

$$f_{addedge}(G, v) = \sigma(f_{ae}(\mathbf{h}_G, \mathbf{h}_v^{(T)})) \quad (8)$$

$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V \quad (9)$$

$$f_{nodes}(G, v) = \text{softmax}(\mathbf{s}) \quad (10)$$

Graph propagation process : readout all node states and generating a graph feature

$$\boldsymbol{h_G} = \sum_{v \in \mathcal{V}} \boldsymbol{h_V^G} \quad \text{or} \quad \boldsymbol{h_G} = \sum_{v \in \mathcal{V}} \boldsymbol{g_v^G} \odot \boldsymbol{h_v^G} \qquad \boldsymbol{g_v^G} = \sigma\big(g_m(\boldsymbol{h_v})\big)$$

$$\boldsymbol{h'_v} = f_n(\boldsymbol{a_v}, \boldsymbol{h_v}) \quad \forall v \in \mathcal{V} \qquad \boldsymbol{a_v} = \sum_{u,v \in \mathcal{E}} f_e\big(\boldsymbol{h_u}, \boldsymbol{h_v}, \boldsymbol{x_{u,v}}\big) \quad \forall v \in \mathcal{V}$$

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

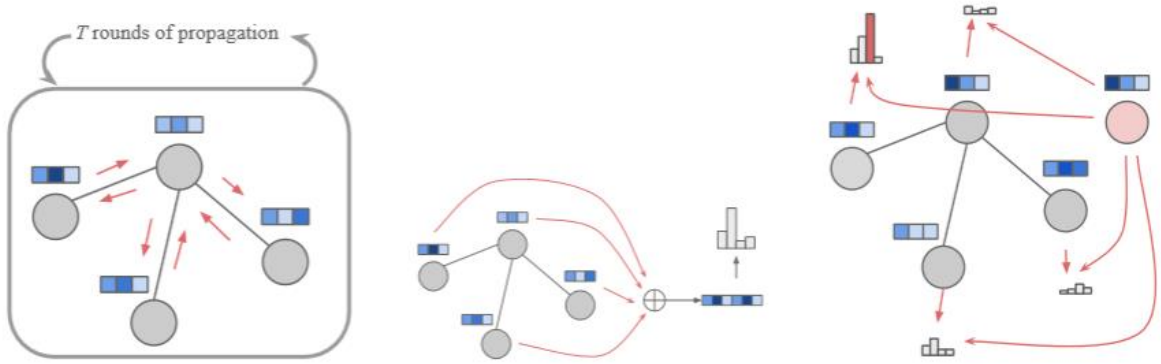


Figure 2. Illustration of the graph propagation process (left), graph level predictions using $f_{addnode}$ and $f_{addedge}$ (center), and node selection f_{nodes} modules (right).

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G) \quad (5)$$

$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G) \quad (6)$$

$$f_{addnode}(G) = \text{softmax}(f_{an}(\mathbf{h}_G)) \quad (7)$$

$$f_{addedge}(G, v) = \sigma(f_{ae}(\mathbf{h}_G, \mathbf{h}_v^{(T)})) \quad (8)$$

$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V \quad (9)$$

$$f_{nodes}(G, v) = \text{softmax}(\mathbf{s}) \quad (10)$$

Add node : a step to decide whether or not to add a node

$$\boldsymbol{f_{addnode}}(\boldsymbol{G}) = \text{softmax}\big(f_{an}(\boldsymbol{h_G})\big)$$

If "yes" → The new node vectors $\boldsymbol{h_V^{(T)}}$ are carried over to the next step

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

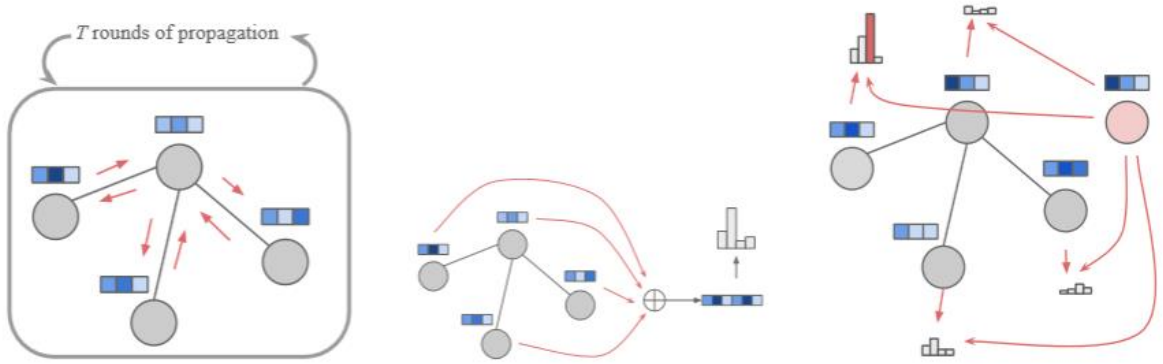


Figure 2. Illustration of the graph propagation process (left), graph level predictions using $f_{addnode}$ and $f_{addedge}$ (center), and node selection f_{nodes} modules (right).

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G) \quad (5)$$

$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G) \quad (6)$$

$$f_{addnode}(G) = \text{softmax}(f_{an}(\mathbf{h}_G)) \quad (7)$$

$$f_{addedge}(G, v) = \sigma(f_{ae}(\mathbf{h}_G, \mathbf{h}_v^{(T)})) \quad (8)$$

$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V \quad (9)$$

$$f_{nodes}(G, v) = \text{softmax}(\mathbf{s}) \quad (10)$$

Add edge : a step to add an edge to the new node

$f_{addedge}(G, v) = \sigma\left(f_{ae}\left(\boldsymbol{h_G}, \boldsymbol{h_v^{(T)}}\right)\right)$: the probability of adding an edge to the newly created node v.

If "yes" → $f_{nodes}(G, v) = \text{softmax}\left(f_s\left(\boldsymbol{h_u^{(T)}}, \boldsymbol{h_v^{(T)}}\right)\right)$: Score for each node to connect the edges

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

Objective function

$p(G)$: marginal likelihood $\xrightarrow{\text{permutation}}$ $p(G) = \sum_{\pi \in \mathcal{P}(G)} p(G, \pi) = \mathbb{E}_{q(\pi|G)}\left[\frac{p(G,\pi)}{q(\pi|G)}\right]$

Following all possible permutations is intractable
→ samples from data : $q(\pi|G) \approx p_{data}(\pi|G)$

$\mathbb{E}_{p_{data}(G,\pi)}[\log p(G,\pi)] = \mathbb{E}_{p_{data}(G)}\mathbb{E}_{p_{data}(\pi|G)}[\log p(G,\pi)]$

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

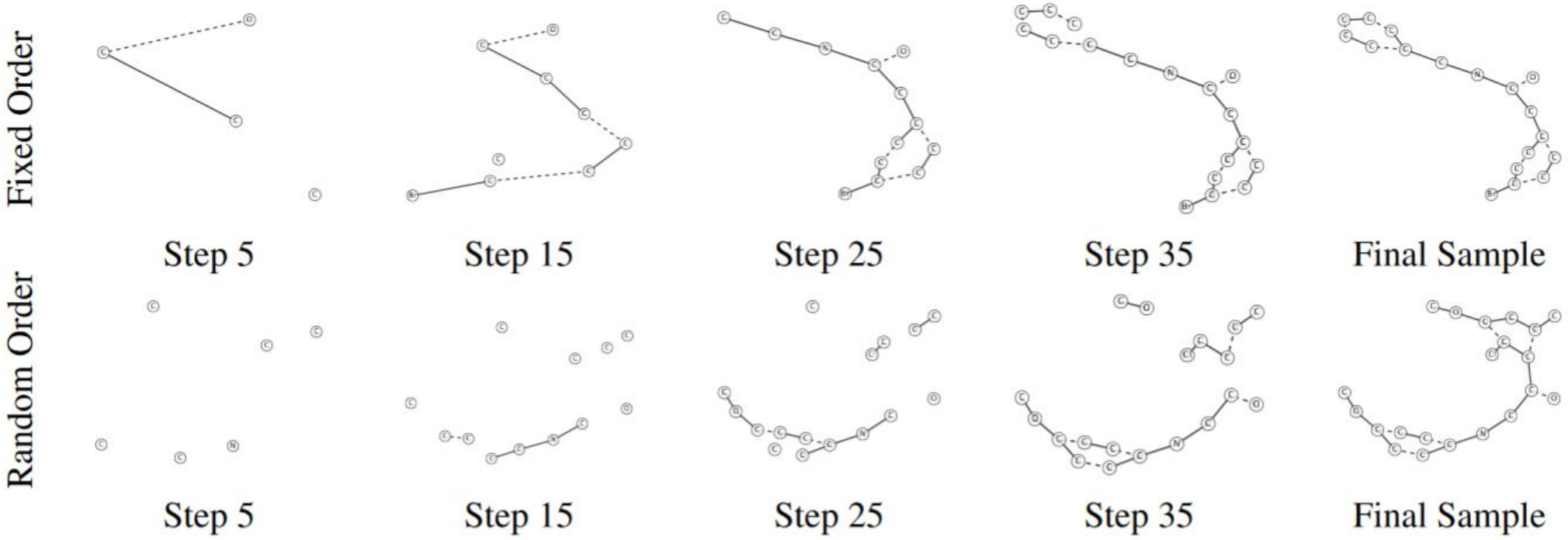


Figure 6. Visualization of the molecule generation processes for graph model trained with fixed and random ordering. Solid lines represent single bonds, and dashed lines represent double bounds.

Li, Yujia, et al.
"Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Molecular applications
3. Molecular generative model

Table 2. Molecule generation results. N is the number of permutations for each molecule the model is trained on. Typically the number of different SMILES strings for each molecule < 100.

Arch	Grammar	Ordering	N	NLL	%valid	%novel
LSTM	SMILES	Fixed	1	21.48	93.59	81.27
LSTM	SMILES	Random	< 100	**19.99**	93.48	83.95
LSTM	Graph	Fixed	1	22.06	85.16	80.14
LSTM	Graph	Random	$O(n!)$	63.25	91.44	91.26
Graph	Graph	Fixed	1	20.55	**97.52**	90.01
Graph	Graph	Random	$O(n!)$	58.36	95.98	**95.54**

Table 3. Negative log-likelihood evaluation on small molecules with no more than 6 nodes.

Arch	Grammar	Ordering	N	Fixed	Best	Marginal
LSTM	SMILES	Fixed	1	17.28	15.98	15.90
LSTM	SMILES	Random	< 100	**15.95**	15.76	15.67
LSTM	Graph	Fixed	1	16.79	16.35	16.26
LSTM	Graph	Random	$O(n!)$	20.57	18.90	15.96
Graph	Graph	Fixed	1	16.19	**15.75**	15.64
Graph	Graph	Random	$O(n!)$	20.18	18.56	**15.32**

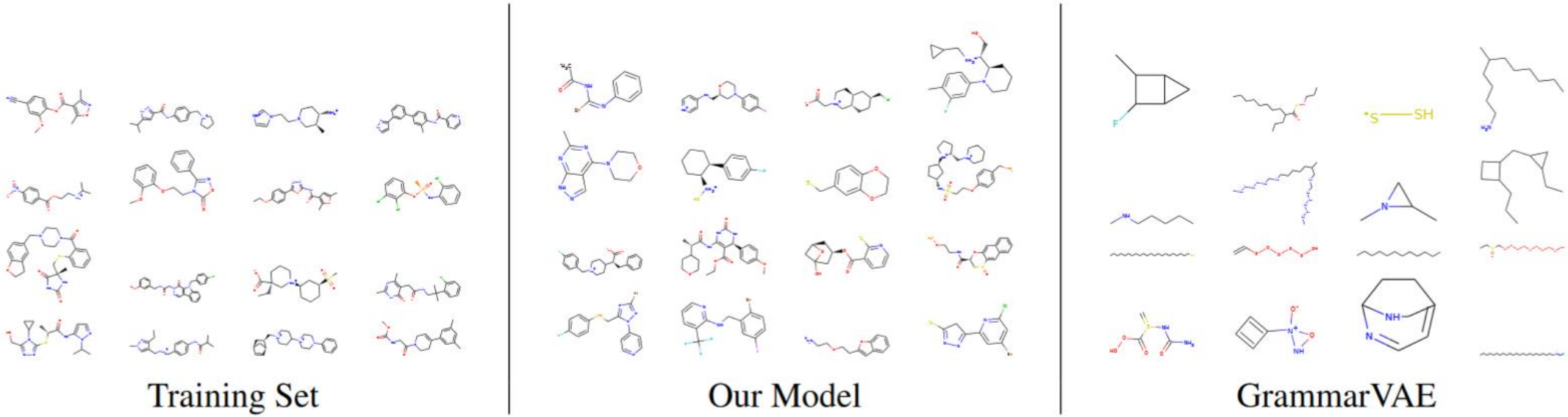


Figure 7. Random samples from the training set, our model and GrammarVAEs. Invalid samples are filtered out. (Kusner et al., 2017) showed better samples for GrammarVAEs in a neighborhood around a data example, while here we are showing samples from the prior.

Li, Yujia, et al. "Learning deep generative models of graphs." *arXiv preprint arXiv:1803.03324* (2018).

Interacting physical system

- Interacting systems
- Nodes : particles, Edges : (physical) interaction between particle pairs
- Latent code : the underlying interaction graph

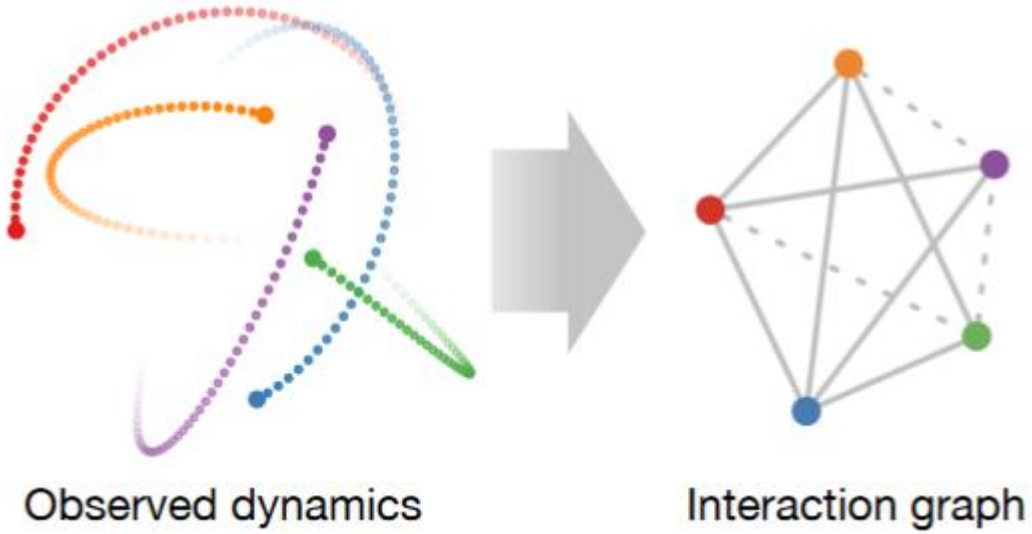


Figure 1. Physical simulation of 2D particles coupled by invisible springs (*left*) according to a latent interaction graph (*right*). In this example, solid lines between two particle nodes denote connections via springs whereas dashed lines denote the absence of a coupling. In general, multiple, directed edge types – each with a different associated relation – are possible.

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system

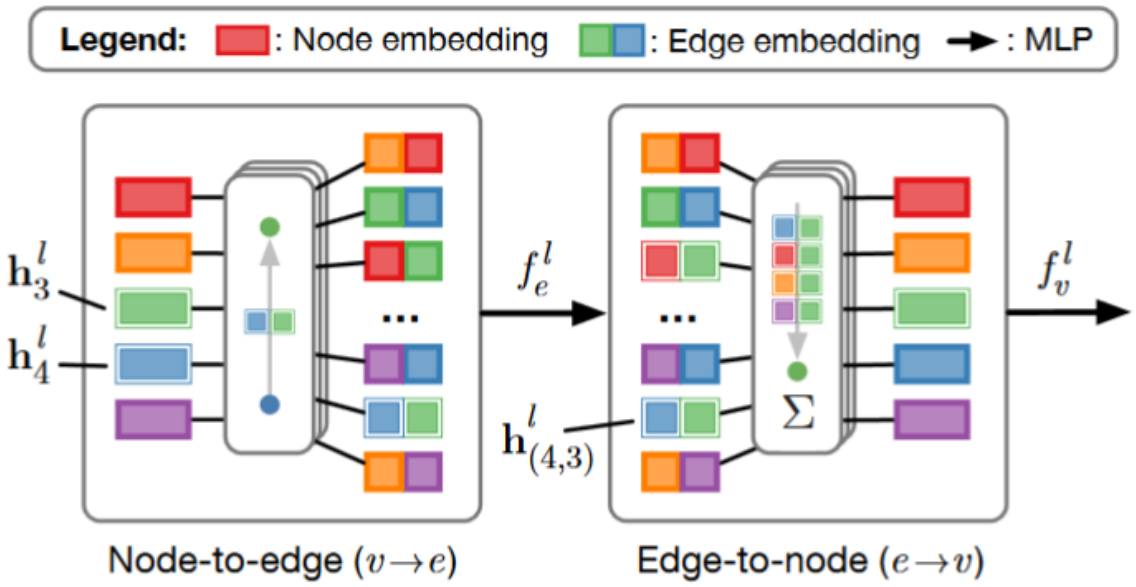


Figure 2. Node-to-edge ($v \to e$) and edge-to-node ($e \to v$) operations for moving between node and edge representations in a GNN. $v \to e$ represents concatenation of node embeddings connected by an edge, whereas $e \to v$ denotes the aggregation of edge embeddings from all incoming edges. In our notation in Eqs. (1)–(2), every such operation is followed by a small neural network (e.g. a 2-layer MLP), here denoted by a black arrow. For clarity, we highlight which node embeddings are combined to form a specific edge embedding ($v \to e$) and which edge embeddings are aggregated to a specific node embedding ($e \to v$).

Input graph : $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}})$ with vertices $v \in \boldsymbol{\mathcal{V}}$ and edges $e = (v, v') \in \mathcal{E}$

$$v \to e : \boldsymbol{h}_{(i,j)}^{\boldsymbol{l}} = f_e^l\left(\left[\boldsymbol{h}_{\boldsymbol{i}}^{\boldsymbol{l}}, \boldsymbol{h}_{\boldsymbol{j}}^{\boldsymbol{l}}, \boldsymbol{x}_{(i,j)}\right]\right)$$

$$e \to v : \boldsymbol{h}_{\boldsymbol{j}}^{\boldsymbol{l+1}} = f_v^l\left(\left[\sum_{i \in \mathcal{N}_j} \boldsymbol{h}_{(i,j)}^{\boldsymbol{l}}, \boldsymbol{x}_{\boldsymbol{j}}\right]\right)$$

$\boldsymbol{x}_{\boldsymbol{i}}$: initial node features, $\boldsymbol{x}_{(i,j)}$: initial edge features

Kipf, Thomas, et al. "Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system

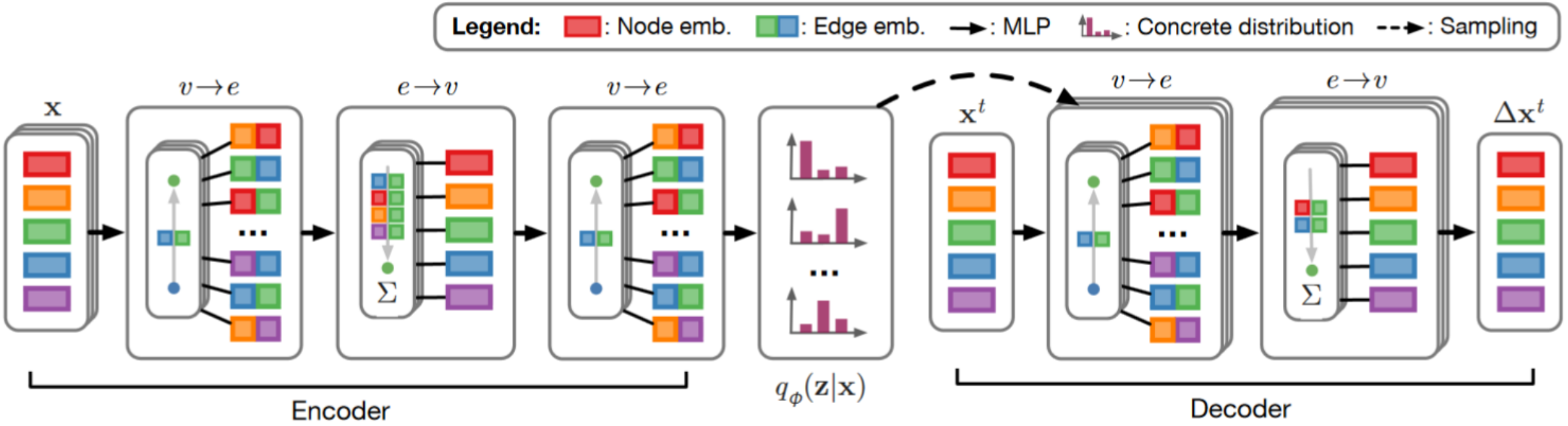


Figure 3. The NRI model consists of two jointly trained parts: An encoder that predicts a probability distribution $q_\phi(\mathbf{z}|\mathbf{x})$ over the latent interactions given input trajectories; and a decoder that generates trajectory predictions conditioned on both the latent code of the encoder and the previous time step of the trajectory. The encoder takes the form of a GNN with multiple rounds of node-to-edge ($v \to e$) and edge-to-node ($e \to v$) message passing, whereas the decoder runs multiple GNNs in parallel, one for each edge type supplied by the latent code of the encoder $q_\phi(\mathbf{z}|\mathbf{x})$.

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

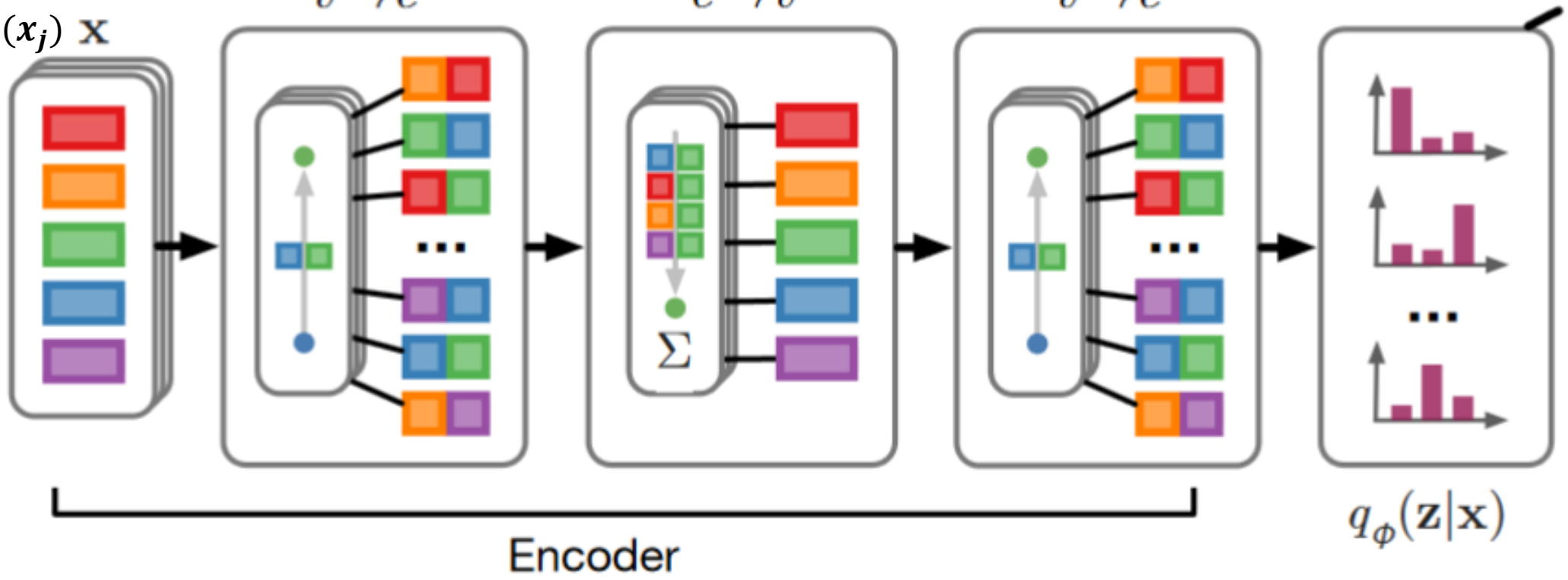
Interacting physical system

Encoder

$$h_{(i,j)}^1 = f_e^1([h_i^1, h_j^1]) \qquad h_j^2 = f_v^1\left(\sum_{i \neq j} h_{(i,j)}^1\right) \qquad h_{(i,j)}^2 = f_e^2([h_i^2, h_j^2])$$

$v \to e$ $\quad e \to v$ $\quad v \to e$

$h_j^1 = f_{emb}(x_j)$ $\mathbf{x}$



$q_\phi(\mathbf{z}|\mathbf{x})$

$$q_\phi(z_{ij}|x) = \text{softmax}(f_{enc,\phi}(x)_{ij,1:K}) = \text{softmax}(h_{(i,j)}^2)$$

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system

Decoder

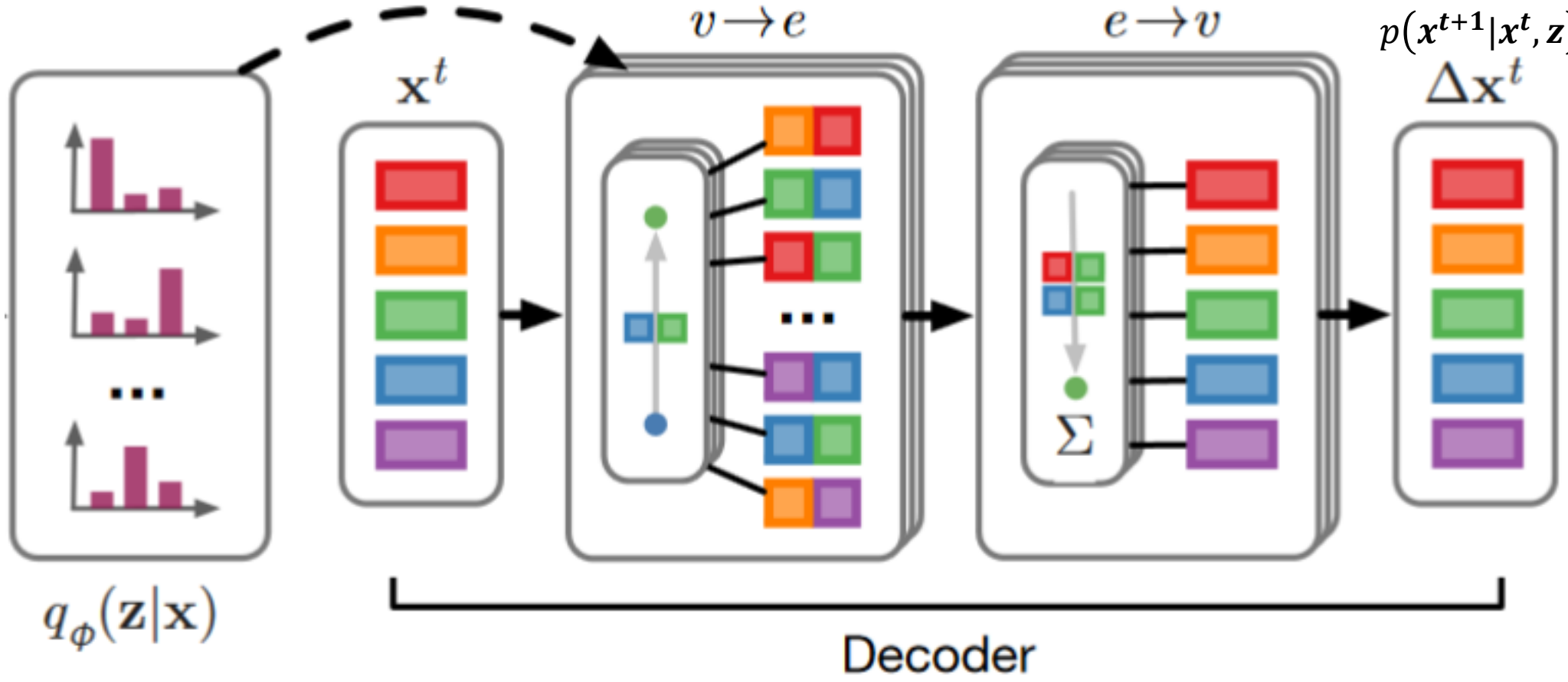
$$MSG_j^t = \sum_{i \neq j} \tilde{\boldsymbol{h}}_{(i,j)}{}^{t}$$

$$\boldsymbol{h}_j^{t+1} = GRU\left(\left[MSG_j^t, \boldsymbol{x}_j^t\right], \tilde{\boldsymbol{h}}_j{}^{t}\right)$$

$$\tilde{\boldsymbol{h}}_{(i,j)}^t = \sum_k z_{ij,k}\, \tilde{f}_e^k\left(\left[\tilde{\boldsymbol{h}}_i^t, \tilde{\boldsymbol{h}}_j^t\right]\right)$$

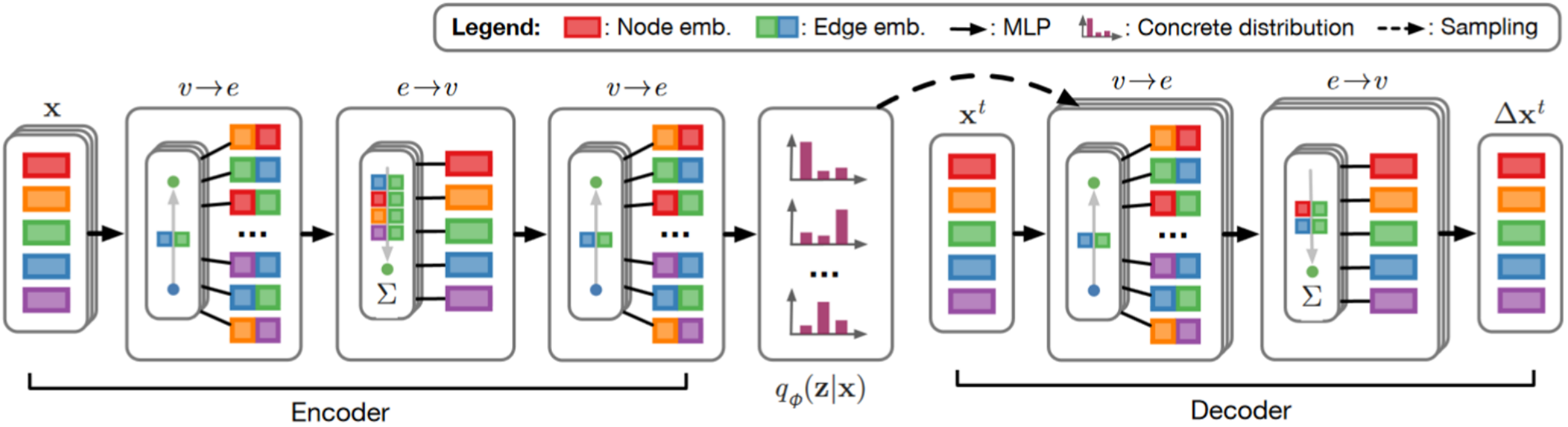
$$\boldsymbol{\mu}_j^{t+1} = \boldsymbol{x}_j^t + f_{out}\left(\tilde{\boldsymbol{h}}_j{}^{t+1}\right)$$

$$p\left(\boldsymbol{x}^{t+1} | \boldsymbol{x}^t, \boldsymbol{z}\right) = \mathcal{N}\left(\boldsymbol{\mu}^{t+1}, \sigma^2 \mathbf{I}\right)$$



Kipf, Thomas, et al. "Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system



$$\mathcal{L} = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]$$

$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \prod_{t=1}^{T} p_\theta(\boldsymbol{x^{t+1}}|\boldsymbol{x^t}, \ldots, \boldsymbol{x^1}|\boldsymbol{z}) = \prod_{t=1}^{T} p_\theta(\boldsymbol{x^{t+1}}|\boldsymbol{x^t}|\boldsymbol{z})$, since the dynamics is Markovian.

$p(\boldsymbol{z}) = \prod_{i \neq j} p_\theta(\boldsymbol{z_{ij}})$, the prior is a factorized uniform distribution over edge types

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system

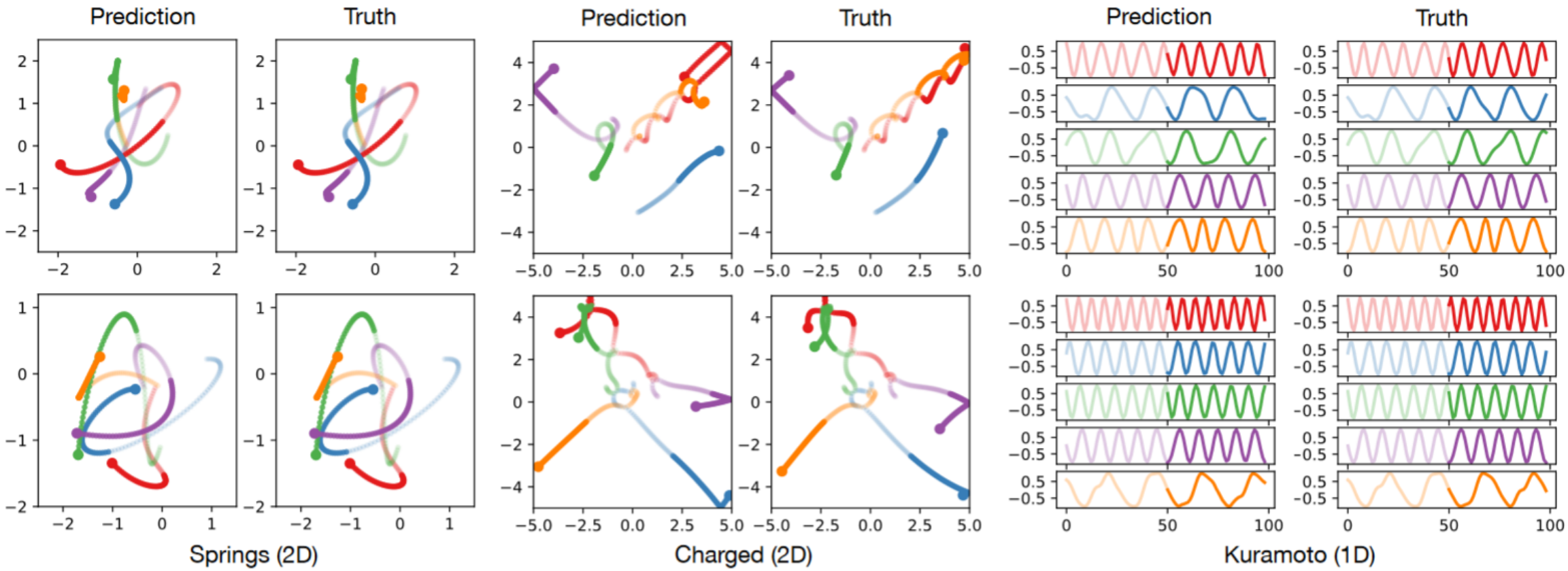


Figure 5. Trajectory predictions from a trained NRI model (unsupervised). Semi-transparent paths denote the first 49 time steps of ground-truth input to the model, from which the interaction graph is estimated. Solid paths denote self-conditioned model predictions.

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

Interacting physical system

Model	Springs	Charged	Kuramoto
	5 objects		
Corr. (path)	52.4±0.0	55.8±0.0	62.8±0.0
Corr. (LSTM)	52.7±0.9	54.2±2.0	54.4±0.5
NRI (sim.)	**99.8**±0.0	59.6±0.8	–
NRI (learned)	**99.9**±0.0	**82.1**±0.6	**96.0**±0.1
Supervised	99.9±0.0	95.0±0.3	99.7±0.0
	10 objects		
Corr. (path)	50.4±0.0	51.4±0.0	59.3±0.0
Corr. (LSTM)	54.9±1.0	52.7±0.2	56.2±0.7
NRI (sim.)	**98.2**±0.0	53.7±0.8	–
NRI (learned)	**98.4**±0.0	**70.8**±0.4	**75.7**±0.3
Supervised	98.8±0.0	94.6±0.2	97.1±0.1

Kipf, Thomas, et al.
"Neural relational inference for interacting systems." *arXiv preprint arXiv:1802.04687* (2018).

References

Geometric Deep Learning and Surveys on Graph Neural Networks

- Bronstein, Michael M., et al. "Geometric deep learning: going beyond euclidean data." IEEE Signal Processing Magazine 34.4 (2017): 18-42.
- [NIPS 2017] Tutorial - Geometric deep learning on graphs and manifolds, https://nips.cc/Conferences/2017/Schedule?showEvent=8735
- Goyal, Palash, and Emilio Ferrara. "Graph embedding techniques, applications, and performance: A survey." Knowledge-Based Systems 151 (2018): 78-94., https://github.com/palash1992/GEM
- Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." arXiv preprint arXiv:1806.01261 (2018).
- Awesome Graph Embedding And Representation Learning Papers, https://github.com/benedekrozemberczki/awesome-graph-embedding

References

Graph Convolutional Network (GCN)

- Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." Advances in Neural Information Processing Systems. 2016.
- Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).
- van den Berg, Rianne, Thomas N. Kipf, and Max Welling. "Graph Convolutional Matrix Completion." stat 1050 (2017): 7.
- Schlichtkrull, Michael, et al. "Modeling relational data with graph convolutional networks." European Semantic Web Conference. Springer, Cham, 2018.
- Levie, Ron, et al. "Cayleynets: Graph convolutional neural networks with complex rational spectral filters." arXiv preprint arXiv:1705.07664 (2017).

https://github.com/SeongokRyu/Graph-neural-networks

References

Attention mechanism in GNN

- Velickovic, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).
- GRAM: Graph-based Attention Model for Healthcare Representation Learning
- Shang, C., Liu, Q., Chen, K. S., Sun, J., Lu, J., Yi, J., & Bi, J. (2018). Edge Attention-based Multi-Relational Graph Convolutional Networks. arXiv preprint arXiv:1802.04944.
- Lee, John Boaz, et al. "Attention Models in Graphs: A Survey." arXiv preprint arXiv:1807.07984 (2018).

Message Passing Neural Network (MPNN)

- Li, Yujia, et al. "Gated graph sequence neural networks." arXiv preprint arXiv:1511.05493 (2015).
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212.

https://github.com/SeongokRyu/Graph-neural-networks

References

Graph AutoEncoder and Graph Generative Model

- Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." arXiv preprint arXiv:1611.07308 (2016).
- Simonovsky, Martin, and Nikos Komodakis. "GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders." arXiv preprint arXiv:1802.03480 (2018).
- Liu, Qi, et al. "Constrained Graph Variational Autoencoders for Molecule Design." arXiv preprint arXiv:1805.09076 (2018).
- Pan, Shirui, et al. "Adversarially Regularized Graph Autoencoder." arXiv preprint arXiv:1802.04407 (2018).
- Li, Y., Vinyals, O., Dyer, C., Pascanu, R., & Battaglia, P. (2018). Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324.
- Jin, Wengong, Regina Barzilay, and Tommi Jaakkola. "Junction Tree Variational Autoencoder for Molecular Graph Generation." arXiv preprint arXiv:1802.04364 (2018).
- Liu, Qi, et al. "Constrained Graph Variational Autoencoders for Molecule Design." arXiv preprint arXiv:1805.09076 (2018).

https://github.com/SeongokRyu/Graph-neural-networks

References

Applications of GNN

- Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." Advances in neural information processing systems. 2015.
- Kearnes, Steven, et al. "Molecular graph convolutions: moving beyond fingerprints." Journal of computer-aided molecular design 30.8 (2016): 595-608.
- Schütt, K. T., Arbabzadah, F., Chmiela, S., Müller, K. R., & Tkatchenko, A. (2017). Quantum-chemical insights from deep tensor neural networks. Nature communications, 8, 13890.
- Wu, Z., Ramsundar, B., Feinberg, E. N., Gomes, J., Geniesse, C., Pappu, A. S., ... & Pande, V. (2018). MoleculeNet: a benchmark for molecular machine learning. Chemical Science, 9(2), 513-530.
- Feinberg, Evan N., et al. "Spatial Graph Convolutions for Drug Discovery." arXiv preprint arXiv:1803.04465 (2018).

https://github.com/SeongokRyu/Graph-neural-networks

References

Applications of GNN

- Jin, Wengong, Regina Barzilay, and Tommi Jaakkola. "Junction Tree Variational Autoencoder for Molecular Graph Generation." arXiv preprint arXiv:1802.04364 (2018).
- Liu, Qi, et al. "Constrained Graph Variational Autoencoders for Molecule Design." arXiv preprint arXiv:1805.09076 (2018).
- De Cao, Nicola, and Thomas Kipf. "MolGAN: An implicit generative model for small molecular graphs." arXiv preprint arXiv:1805.11973 (2018).
- Selvan, Raghavendra, et al. "Extraction of Airways using Graph Neural Networks." arXiv preprint arXiv:1804.04436 (2018).
- Kipf, Thomas, et al. "Neural relational inference for interacting systems." arXiv preprint arXiv:1802.04687 (2018).

- **And many other interesting works!**

https://github.com/SeongokRyu/Graph-neural-networks

THANK YOU!