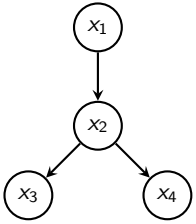
Aalto University

CS-E4890: Deep Learning
Graph neural networks

Alexander Ilin

Motivation

- Previously we processed the following types of inputs:
  - vectors whose elements do not have special order: multi-layer perceptron
  - inputs with 1d or 2d spatial structure: convolutional networks
  - sequences with varying lengths: recurrent neural networks, transformers
- In some applications, the input can be represented as a graph.
- A graph is defined as a 3-tuple $G = (u; V; E)$:
  - u is a global attribute
  - V is a set of nodes with attributes x_i
  - E is a set of edges with attributes e_{kj}



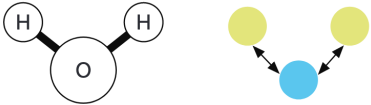
$V = \{x_1, x_2, x_3, x_4\}$

$E = \{e_{12}, e_{23}, e_{24}\}$

- In this lecture, we consider mainly undirected graphs: edges do not have directions.

Example of data that can be represented as graphs: Molecules

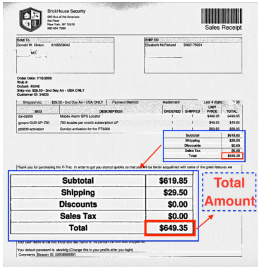
- The task is to predict chemical properties of molecules (Duvenaud et al., 2015; Gilmer et al., 2017):
  - toxicity
  - excitation spectra
  - the level of activity of a chemical compound against cancer cells

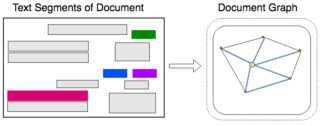


- A graph representation of a molecule:
  - global attribute u: some known property of a molecule (e.g., number of atoms)
  - nodes V: each node corresponds to an atom, a node's attribute x_i is the atom's identity
  - edges E correspond to bond (e.g., edges do not have properties)
- The task is similar to regression but the inputs are graphs: $\mathcal{G} \rightarrow \mathbb{R}^n$.

Example of data that can be represented as graphs: Documents

- The task is to extract information from documents, for example, extract line items from scanned receipts.
- OCR software can extract text segments from scanned documents. Then, we can build a graph representation of a scanned document:
  - nodes: each node corresponds to a text segment;
  - edges can have properties such as the distance between the segments, whether the segments are in the same row/column.
- The task: node classification (two classes: text segment represents a line item or not).

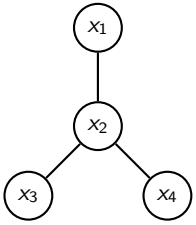




images from (Liu et al., 2019)

Graph neural networks

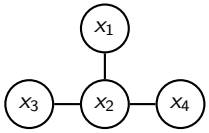
- Graph is an explicit representation of a set of entities (objects) and their relations.
- We need a learning algorithm which models objects and their interactions and grounds modeling in data.
- There is no "default" deep learning component which operates on an arbitrary relational structure. We will review several neural architectures proposed for this task. We will call all such architectures *graph neural networks*.

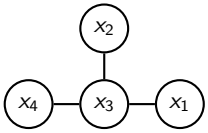


An example of an undirected graph

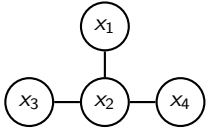
Graph neural networks: Requirements

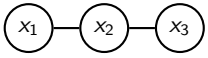
- Permutation invariance. Since nodes in a graph do not have an order (typically), the output of the network should be invariant to node permutations.



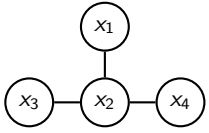


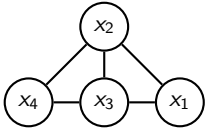
- The network should be able to process graphs with a varying number of nodes.





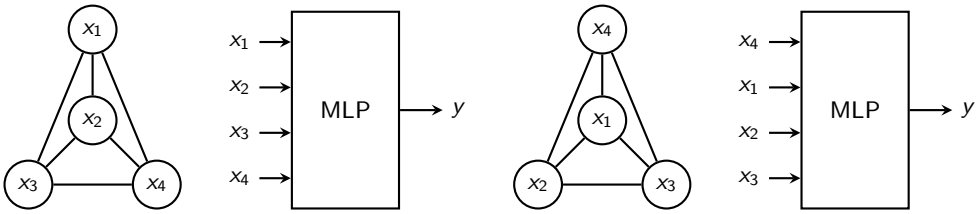
- The network should take into account the topology of the graph.





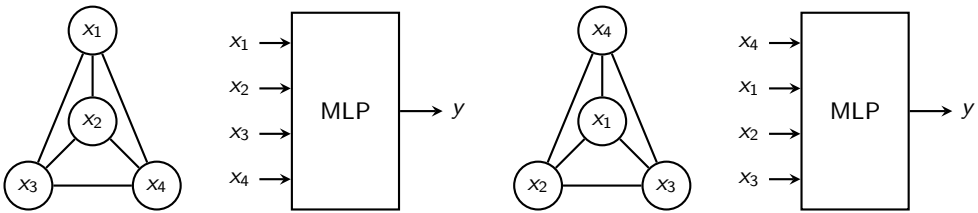
Permutation invariance

- Suppose we use an MLP to process fully-connected graphs. Is the output of an MLP invariant to input (node) permutations?



Permutation invariance

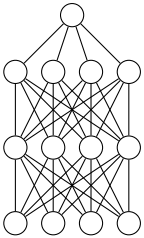
- Suppose we use an MLP to process fully-connected graphs. Is the output of an MLP invariant to input (node) permutations?



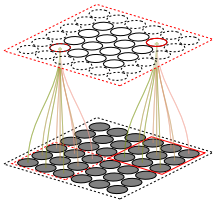
- No. Therefore, an MLP trained on a particular input (x_1, x_2, x_3, x_4) would not transfer to making a prediction for the same inputs under a different ordering, e.g., (x_4, x_1, x_2, x_3).
- Since there are $n!$ such possible permutations, an MLP would require a large number of input/output training examples.

Permutation invariance: Previous models

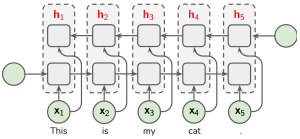
- Are some of the previously considered models invariant or equivariant to input permutations?



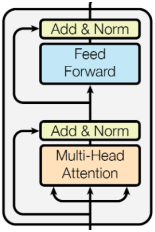
multi-layer perceptron



convolutional network



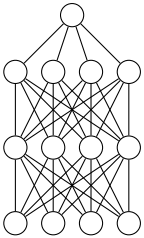
recurrent neural network



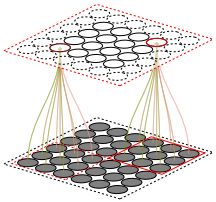
transformer encoder

Permutation invariance: Previous models

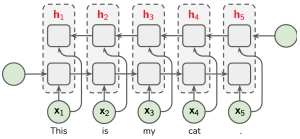
- Are some of the previously considered models invariant or equivariant to input permutations?



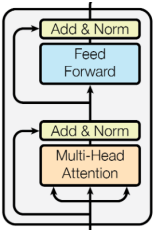
multi-layer perceptron



convolutional network



recurrent neural network

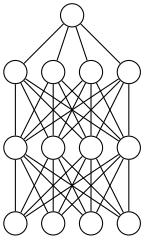


transformer encoder

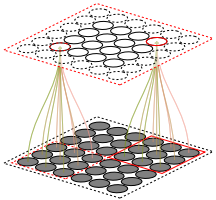
- Transformer encoder: Without positional encoding, the output is equivariant to input permutations.

Ability to process inputs with a varying number of elements

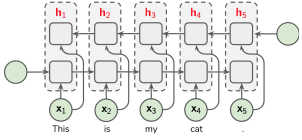
- Which of the previously considered neural networks can process inputs with a varying number of elements?



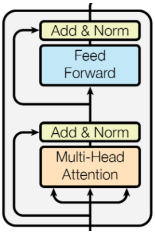
multi-layer perceptron



convolutional network



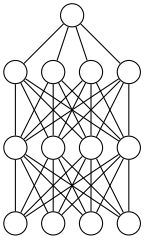
recurrent neural network



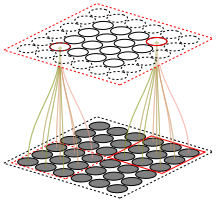
transformer encoder

Ability to process inputs with a varying number of elements

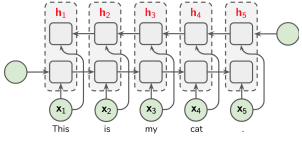
- Which of the previously considered neural networks can process inputs with a varying number of elements?



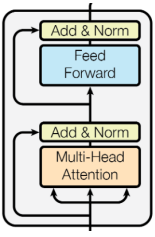
multi-layer perceptron



convolutional network



recurrent neural network

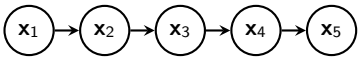


transformer encoder

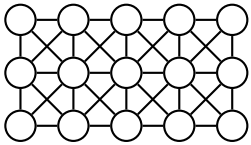
- CNN, RNN and transformer encoder can process sequences of varying lengths.

Previous models as “graph neural networks”

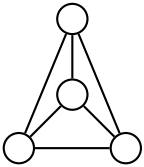
- RNN can be viewed as a neural network which can process graphs with the chain topology.



- CNN can be viewed as a network that can process graphs with the grid topology.



- Transformer encoder can be viewed as a neural network that processes fully connected graphs.



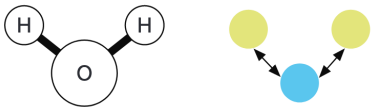
- We want to create a neural network that can process graphs with different topologies.

Neural fingerprint networks

(Duvenaud et al., 2015)

Prediction of chemical properties of molecules (Duvenaud et al., 2015)

- The task is to predict chemical properties of molecules, e.g.:
  - toxicity
  - excitation spectra
  - the level of activity of a chemical compound against cancer cells
- Neural fingerprint network: Convert a graph that represents a molecule into a real-valued vector **f** (fingerprint) which can be further processed to predict some property.



A molecule, in which each atom is represented as a node and edges correspond to bond.

Prior art: Circular fingerprints

- *Circular fingerprints* is an algorithm designed to encode which substructures are present in a molecule in a way that is invariant to atom-relabeling (permutation invariance).

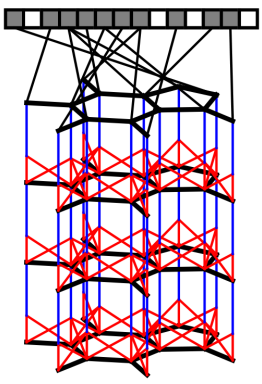
- Algorithm runs for a few iterations (layers).
- Generate each layer's features by applying a fixed hash function to the concatenated features of the neighborhood in the previous layer.
- The results of these hashes are treated as integer indices, where a 1 is written to the fingerprint vector at the index given by the feature vector at each node in the graph.

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 $\mathbf{f}$

Prior art: Circular fingerprints

- *Circular fingerprints* is an algorithm designed to encode which substructures are present in a molecule in a way that is invariant to atom-relabeling (permutation invariance).



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 $\mathbf{f}$

Neural fingerprint networks

- Duvenaud et al. (2015) "neuralized" the circular fingerprint algorithm:

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

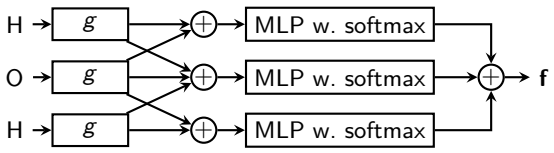
Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in 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 $\mathbf{f}$

Neural fingerprint networks

- Duvenaud et al. (2015) "neuralized" the circular fingerprint algorithm:

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 $\mathbf{f}$



Computational graph

Note: $\mathbf{H}_L^N$ is selected based on the number N of bonds of atom a (up to 5 in organic molecules).

- Mean predictive accuracy (I guess errors) of neural fingerprints compared to standard circular fingerprints:

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	$\mathbf{0.52 \pm 0.07}$	$\mathbf{1.16 \pm 0.03}$	$\mathbf{1.43 \pm 0.09}$

Interaction networks

(Battaglia et al., 2016)

Learning dynamics of physical systems

- The task is to predict the next state of a physical system. Examples:

A n-body system with gravitation.

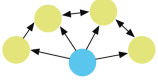
A rigid body system:
Balls moving inside a room.

A mass-spring system:
A rope and a fixed object.

- Modeling assumption: Each pair of objects are in a directed relationship (objects interact with one another). We can represent such physical systems using graphs:

The bodies are nodes and the underlying graph is fully connected.

The balls and walls are nodes, and the underlying graph defines interactions between the balls and between the balls and the walls.

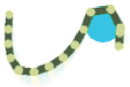


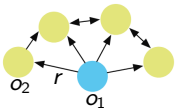
The rope is defined by a sequence of masses which are represented as nodes in the graph.

- The first object (the sender o_1) influences the second (the receiver o_2) via their interaction. The effect of this interaction is predicted by function f_R which takes as input o_1, o_2, as well as attributes of their relationship r:

$$e_{1\to 2,t+1} = f_R(o_{1,t}, o_{2,t}, r)$$

r can be for example, the spring constant if objects are attached by a spring.





- The future state $o_{2,t+1}$ of the receiver is predicted by an object-centric function f_O which takes as input both $e_{1\to 2,t+1}$ and the receiver's current state $o_{2,t}$:

$$o_{2,t+1} = f_O(o_{2,t}, e_{1\to 2,t+1})$$

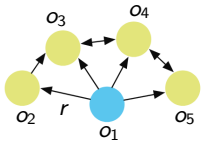
Interaction networks: The model

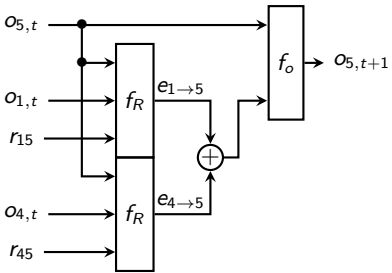
- If there are multiple objects interacting with a given object (e.g, o_3), their effects are aggregated:

$$o_{3,t+1} = f_O\left(o_{3,t}, \sum_{i=1,2,4} e_{i\to 3,t+1}\right)$$

  where summation is done over all objects interacting with o_3.

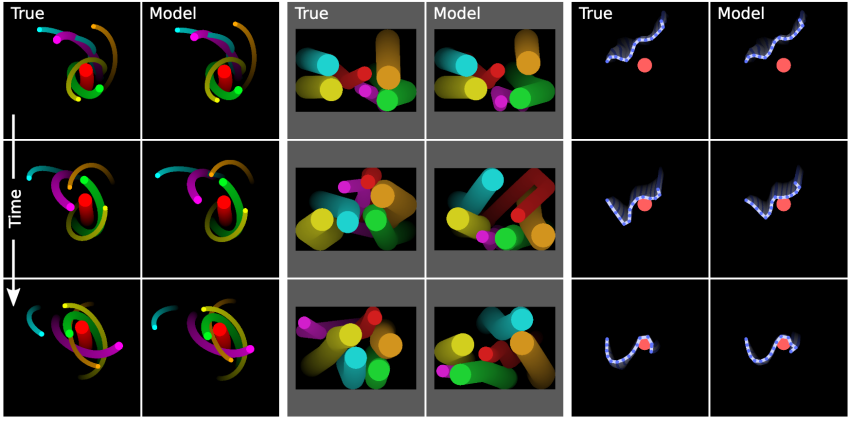
- The future states of all objects are computed in a similar way.





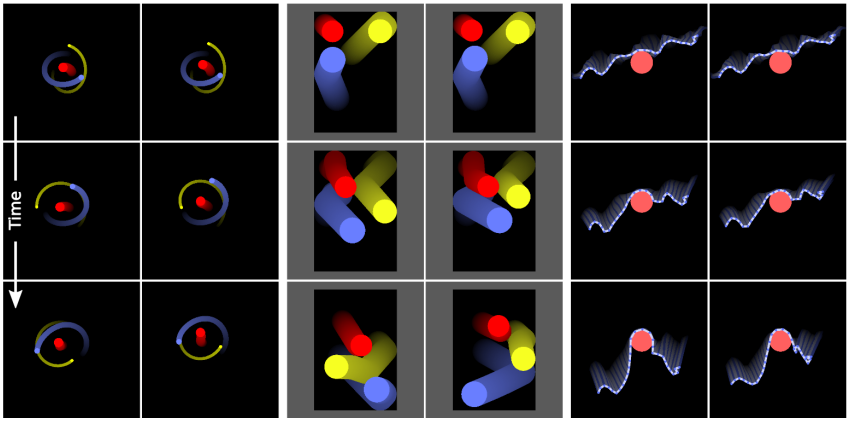
Computational graph

Interaction networks: Results



Prediction rollouts. Each column contains three panels of three video frames (with motion blur), each spanning 1000 rollout steps. Columns 1-2 are ground truth and model predictions for n-body systems, 3-4 are bouncing balls, and 5-6 are strings.

Interaction networks: Results



The model was able to generalize to systems of different sizes and structure. For n-body, the training was on 6 bodies, and generalization was to 3 bodies. For balls, the training was on 6 balls, and generalization was to 3 balls. For strings, thetraining was on 15 masses with 1 end pinned, and generalization was to 30 masses with 0 end pinned.

Interaction networks: Results

- Mean-squared prediction errors:

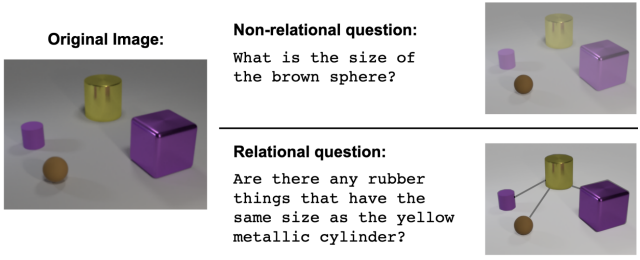
Domain	Constant velocity	Baseline	Dynamics-only IN	IN
n-body	82	79	76	**0.25**
Balls	0.074	0.072	0.074	**0.0020**
String	0.018	0.016	0.017	**0.0011**

  - Baseline: MLP with two 300-length hidden layers, which took as input a flattened vector of all of the input data
  - Dynamics-only IN: a variant of the IN with the interaction effects removed.

Relational network for
visual scene understanding
(Santoro et al., 2017)

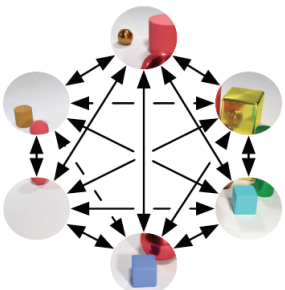
CLEVR dataset

- An example from CLEVR dataset of relational reasoning: An image containing four objects is shown alongside non-relational and relational questions. The relational question requires explicit reasoning about the relations between the four objects in the image, whereas the non-relational question requires reasoning about the attributes of a particular object.

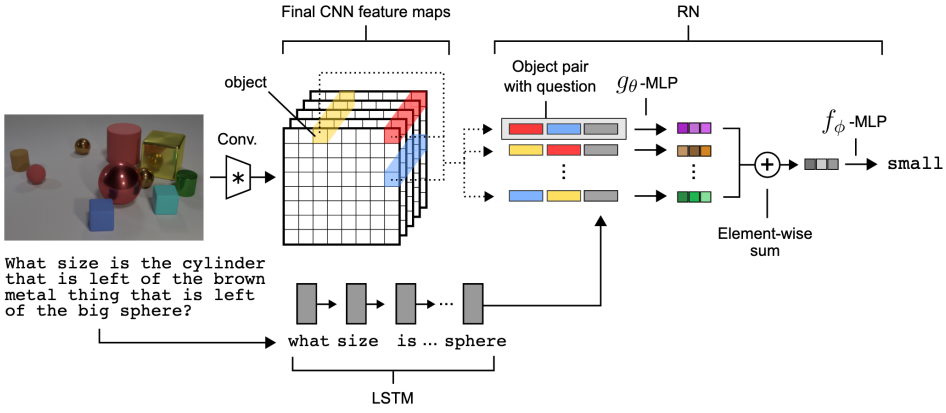


Visual scene understanding with a relational network (Santoro et al., 2017)

- An image is decomposed into patches. Each patch is treated as an object (a node in a fully connected graph).
- A graph is processed with a relational network (RN) which models relations between each pair of objects to produce the correct answer to a given question.
- The question (its embedding) is used as a global context for modeling relations.



Visual scene understanding with a relational network (Santoro et al., 2017)



Questions are processed with an LSTM to produce a question embedding. Images are processed with a CNN to produce a set of objects for the RN. Objects (three examples illustrated here in yellow, red, and blue) are constructed using feature-map vectors from the convolved image. The RN considers relations across all pairs of objects, conditioned on the question embedding, and integrates all these relations to answer the question.

Visual scene understanding with a relational network: Results

- Results on CLEVR from pixels. Accuracy on the test set broken down by question category:

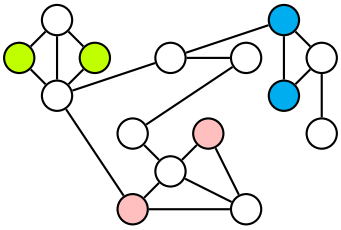
Model	**Overall**	Count	Exist	Compare Numbers	Query Attribute	Compare Attribute
Human	92.6	86.7	96.6	86.5	95.0	96.0
Q-type baseline	41.8	34.6	50.2	51.0	36.0	51.3
LSTM	46.8	41.7	61.1	69.8	36.8	51.8
CNN+LSTM	52.3	43.7	65.2	67.1	49.3	53.0
CNN+LSTM+SA	68.5	52.2	71.1	73.5	85.3	52.3
CNN+LSTM+SA*	76.6	64.4	82.7	77.4	82.6	75.4
CNN+LSTM+RN	**95.5**	**90.1**	**97.8**	**93.6**	**97.9**	**97.1**

Graph Convolutional Networks

(Kipf and Welling, 2017)

Graph Convolutional Networks (Kipf and Welling, 2017)

- Graph convolutional networks (GCNs) are perhaps the most well known "graph neural networks".
- Motivation of GCNs: semi-supervised classification of nodes in a graph.

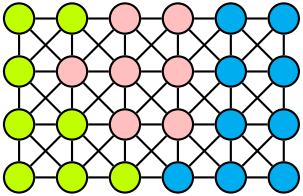


Example:

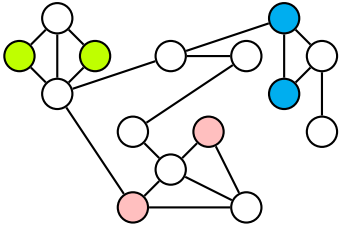
- nodes are documents
- edges are citation links
- node attributes x_i are bag-of-words features of documents
- some documents have class labels

- Assumption: when predicting the class of a node, the attributes and connectivity of nearby nodes provide useful side information or additional context.

Generalization of convolution to graphs



- Consider an image segmentation problem: classify each pixel of an image.
- Image segmentation is usually done with convolutional neural networks (U-net).
- We can view an image as a graph where each pixel is connected to all its neighbors.



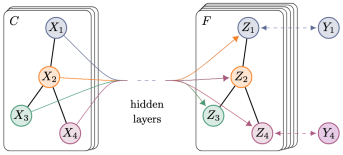
- Kipf and Welling (2017) generalize the concept of convolution to graphs with arbitrary structure.
- They adopt a spectral view on convolutions: convolutions in Fourier-domain are simple pointwise multiplication of the Fourier-transform of a signal.

Graph "convolutional" layer

- The input of a layer is a graph with N nodes with C-dimensional attributes $X \in \mathbb{R}^{N \times C}$.
- The output is a graph with the same structure and a new set of features $Z \in \mathbb{R}^{N \times F}$.
- The output is computed (in authors' notation) as:

$$Z = \hat{A}XW$$

where $W \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters.



- $\hat{A}$ is a matrix that describes the structure of the graph: it has non-zero diagonal elements and elements $a_{ij} \neq 0$ if node i is connected to node j.
- For each node i, we combine signals $\hat{a}_{ij} W^\top x_{j:}$ coming from all its neighbors $j \in \mathcal{N}(i)$:

$$z_{i:} = (\hat{a}_{i:} XW)^\top = \sum_{j \in \mathcal{N}(i)} \hat{a}_{ij} W^\top x_{j:}$$

Results for semi-supervised classification with GCNs

- Nodes are documents and edges are citation links.
- Node attributes x_i are bag-of-words features of documents.
- Some documents have class labels (the label rate is 0.036 for Citeseer, 0.052 for Cora, 0.003 for Pubmed and 0.001 for NELL).
- The task is to classify all the documents.

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

Classification accuracy (in percent)

Recurrent Relational Networks

(Palm et al., 2018)

Recurrent Relational Networks (Palm et al., 2018)

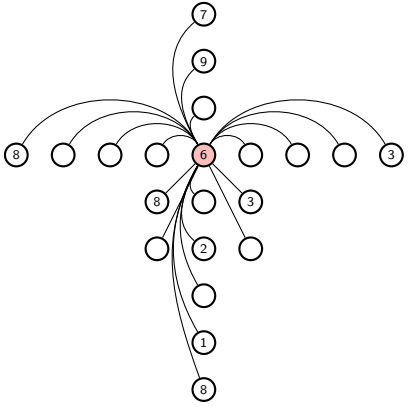
- We want to solve tasks that require a chain of interdependent steps of relational inference, like, for example, solving Sudoku.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A graph describing the Sudoku puzzle

- The puzzle is represented as a graph in which a cell is represented by a node. The nodes are connected to all nodes in the same row, in the same column and in the same 3×3 block.

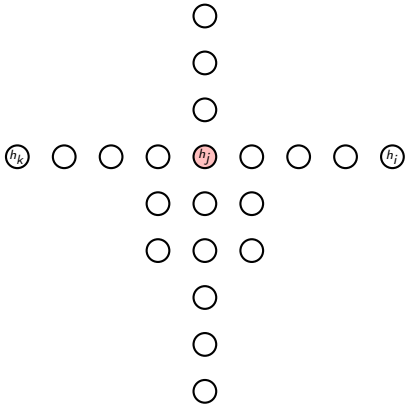
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



The subgraph that contains only the nodes connected to the pink node and the corresponding links.

Recurrent Relational Networks: The algorithm

- Initialize the states of the nodes to h_j^0.

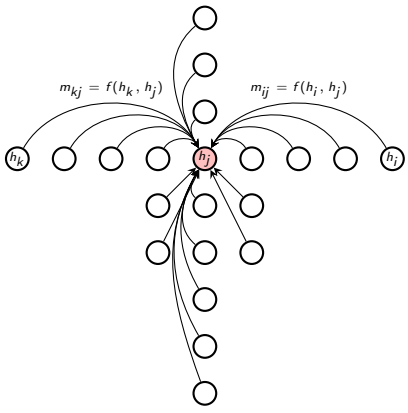


Recurrent Relational Networks: The algorithm

- Initialize the states of the nodes to h_j^0.
- Build a computational graph with T iterations. Each iteration consists of the following four steps.
  1. Compute messages for all edges connecting a pair of nodes i and j:

$$m_{ij}^t = f(h_i^{t-1}, h_j^{t-1})$$
$$m_{ji}^t = f(h_j^{t-1}, h_i^{t-1})$$

f can be modeled with an MLP. Note that for each edge we need to compute two messages.



Recurrent Relational Networks: The algorithm

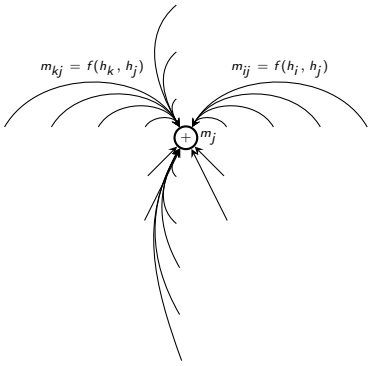
- Initialize the states of the nodes to h_j^0.
- Build a computational graph with T iterations. Each iteration consists of the following four steps.
  1. Compute messages for all edges connecting a pair of nodes i and j:

$$m_{ij}^t = f(h_i^{t-1}, h_j^{t-1})$$
$$m_{ji}^t = f(h_j^{t-1}, h_i^{t-1})$$

     f can be modeled with an MLP. Note that for each edge we need to compute two messages.

  2. In every node, aggregate all incoming messages by summation:

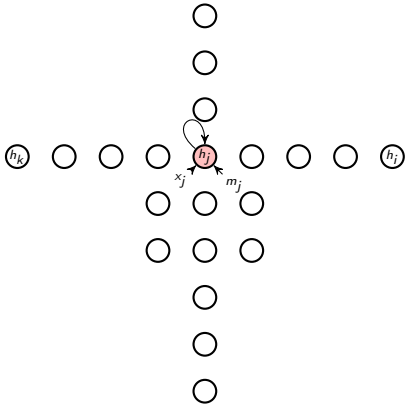
$$m_j^t = \sum_{i \in \mathcal{N}(j)} m_{ij}^t$$



3. Update the states of the nodes:

$$h_j^t = g(h_j^{t-1}, x_j, m_j^t)$$

Input x_j is either the given digit for cell j or a special token indicating a missing digit. g can be modeled by a recurrent unit, for example, GRU.



Recurrent Relational Networks: The algorithm

3. Update the states of the nodes:

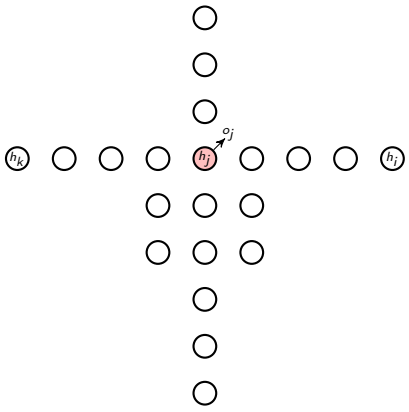
$$h_j^t = g(h_j^{t-1}, x_j, m_j^t)$$

Input x_j is either the given digit for cell j or a special token indicating a missing digit. g can be modeled by a recurrent unit, for example, GRU.

4. For each node, compute outputs o_j^t:

$$o_j^t = f_o(h_j^t)$$

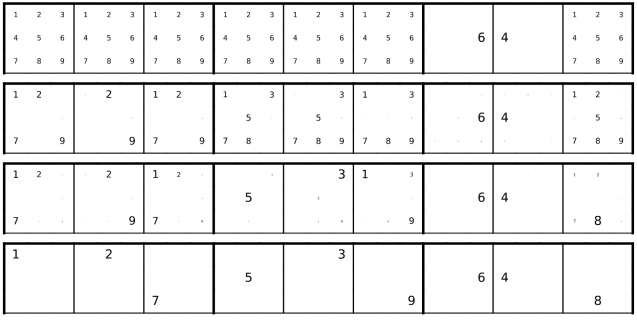
and compute the loss. The loss function relates the outputs with the correct digits in the solved Sudoku puzzle (we use `CrossEntropyLoss`). f_o can be modeled with a linear layer.



- Comparison of methods for solving Sudoku puzzles (only differentiable methods):

Method	Givens	Accuracy
*Recurrent Relational Network** (this work)	17	**96.6%**
Loopy BP, modified [Khan et al., 2014]	17	92.5%
Loopy BP, random [Bauke, 2008]	17	61.7%
Loopy BP, parallel [Bauke, 2008]	17	53.2%
Deeply Learned Messages* [Lin et al., 2015]	17	0%
Relational Network, node* [Santoro et al., 2017]	17	0%
Relational Network, graph* [Santoro et al., 2017]	17	0%
Deep Convolutional Network [Park, 2016]	24-36	70%

Recurrent Relational Networks: Solving Sudoku



Example of how the trained network solves part of a Sudoku. Only the top row of a full 9x9 Sudoku is shown for clarity. From top to bottom steps 0, 1, 8 and 24 are shown. Each cell displays the digits 1-9 with the font size scaled (non-linearly for legibility) to the probability the network assigns to each digit. Notice how the network eliminates the given digits 6 and 4 from the other cells in the first step.

General algorithms for graph neural networks (GNNs)

Similarities between different graph neural networks

- We have considered several algorithms for graph neural networks:
  - neural networks for learning molecular fingerprints (Duvenaud et al., 2015)
  - interaction networks (Battaglia et al., 2016)
  - simple relational network for visual scene understanding (Santoro et al., 2017)
  - graph convolutional networks (Kipf and Welling, 2017)
  - recurrent relational networks (Palm et al., 2018)
- They all have very similar structure: In every iteration, nodes send messages to their neighbors and node attributes are updated using the received messages. The differences are mainly in the parametric form of the messages and the way the messages are aggregated.
- Let us review the computational steps in such graph neural networks (GNNs).

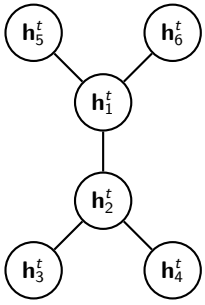
Computational steps in GNNs

- The input of a GNN is an undirected graph G with node features $\mathbf{x}_i$ and edge features $\mathbf{e}_{ij}$.
- Each node has hidden state $\mathbf{h}_i$ which is initialized to $\mathbf{h}_i^{t=0}$.
- There are T iterations which consist of several steps (see next slide) that update the states of the nodes:

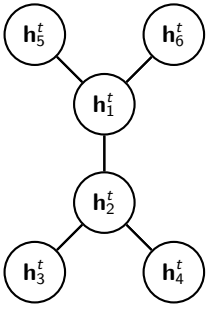
$$\mathbf{h}_i^0 \rightarrow \mathbf{h}_i^1 \rightarrow ... \rightarrow \mathbf{h}_i^T$$

- Finally, a readout function combines all node states to compute a single output:

$$\mathbf{y} = o(\{\mathbf{h}_i^T \mid i \in G\})$$



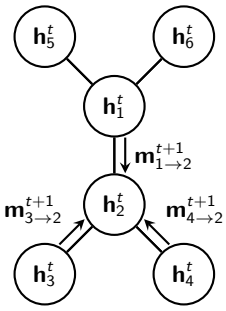
One iteration in a GNN



One iteration in a GNN

1. Each node receives messages from all its neighbors

$$\mathbf{m}_{j\to i}^{t+1} = g_t(\mathbf{h}_j^t, \mathbf{h}_i^t, \mathbf{e}_{ji})$$



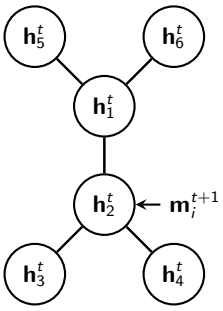
One iteration in a GNN

1. Each node receives messages from all its neighbors

$$\mathbf{m}_{j\to i}^{t+1} = g_t(\mathbf{h}_j^t, \mathbf{h}_i^t, \mathbf{e}_{ji})$$

2. Each node aggregates messages (for example, by summation):

$$\mathbf{m}_i^{t+1} = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{j\to i}^{t+1}$$



One iteration in a GNN

1. Each node receives messages from all its neighbors

$$\mathbf{m}_{j\to i}^{t+1} = g_t(\mathbf{h}_j^t, \mathbf{h}_i^t, \mathbf{e}_{ji})$$

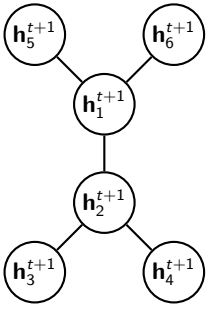
2. Each node aggregates messages (for example, by summation):

$$\mathbf{m}_i^{t+1} = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{j\to i}^{t+1}$$

3. The state of each node is updated using the aggregate message:

$$\mathbf{h}_i^{t+1} = f(\mathbf{h}_i^t, \mathbf{m}_i^{t+1}, \mathbf{x}_i)$$

f is often implemented using a recurrent unit such as GRU. We can use $\mathbf{x}_i$ as extra inputs.



Example: Message passing in graph convolutional networks (GCNs)

1. Each node receives messages from all its neighbors

$$\mathbf{m}_{j\to i}^{t+1} = g(\mathbf{h}_j) = \hat{a}_{ij}\mathbf{W}^\top \mathbf{h}_j$$

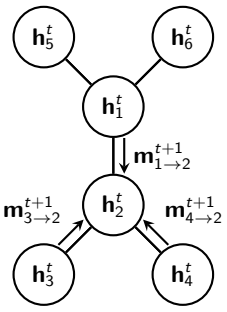
2. Each node aggregates messages (including a message from itself):

$$\mathbf{m}_i^{t+1} = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{j\to i}^{t+1}$$

3. The state of each node is updated using aggregate messages:

$$\mathbf{h}_i^{t+1} = f(\mathbf{m}_i^{t+1}) = \text{relu}(\mathbf{m}_i^{t+1})$$

- GCNs use very simple g and f, which limits their representation power.



Neural Message Passing

- Gilmer et al. (2017) proposed to unify several graph neural network algorithms in more general *message passing neural networks* (MPNN).
- Many previously considered graph neural networks can be viewed as an instance of MPNN.
- Later, Battaglia et al. (2018) defined a more general framework that also includes the update of the edge attributes in the first phase of the forward pass.
- Note: A message-passing algorithm is used for performing inference on probabilistic graphical models, such as Bayesian networks and Markov random fields (known as belief propagation).

Tips on designing a deep learning model

Tips on designing a deep learning model

- The main building blocks:
  - multi-layer perceptron
  - convolutional layers
  - recurrent layers
  - transformer blocks with multi-head attention
- The computational graph should reflect the important modeling assumptions (for example, sharing weights assumes that different parts of the model input should be processed in the same way).
- Think about the optimal inference procedure that a neural network needs to learn.
  - How would you design a hand-crafted algorithm to solve the problem?
  - If you would start with a probabilistic graphical model that describes your problem and derive the inference procedure, what would be the computational graph of the algorithm? What are the needed computations?

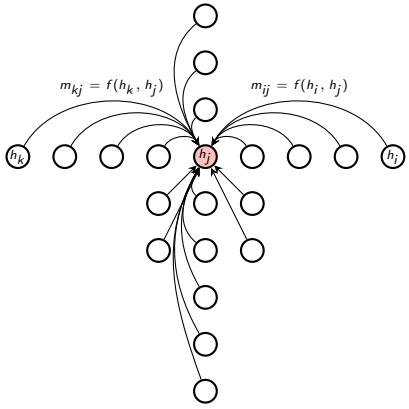
Tips on designing a deep learning model

- Debug the model:
  - Use units tests (similar to the ones used in the notebooks).
  - Do not start training on a big dataset, you may waste a lot of time.
  - Fit a tiny toy dataset first (even simpler than MNIST). The model should learn it. If you can run the experiments fast, you will save a lot of time when debugging your model.
- Tune the model on large data:
  - Think of data augmentation. Ideally you want to generate infinitely many data samples.
  - Do hyperparameter search (random search is a good option).
  - Analyze the mistakes that the model makes to figure out how to change the model architecture.

Home assignment

Assignment 07_gnn

- You need to implement a graph neural network which solves Sudoku puzzles, which is inspired by (Palm et al., 2018).



Recommended reading

- Battaglia et al., 2018. Relational inductive biases, deep learning, and graph networks.
- Other papers cited in the lecture slides.