Previous Weeks

* CNN basics
* Popular CNN Architectures
* Object detection with YOLO
  - - - -> Fuel atomizer characteristics

Goal: (✓) know how convolution op. works
(✓) Aware: Advantages & issues of CNN
(✓) Able to import & use CNN models if need arises

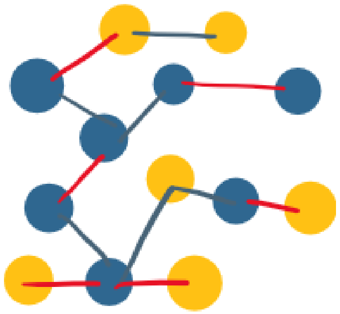
Graph Neural Networks:

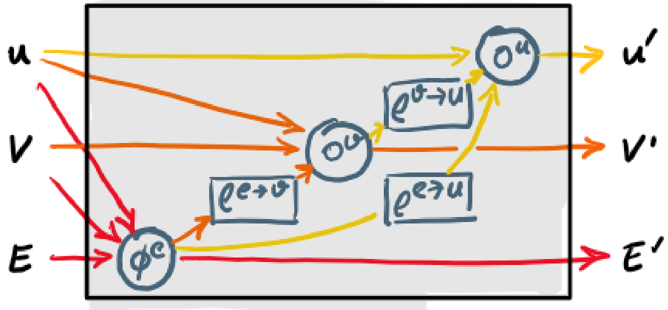
1) GNN Basics
2) How GNN works
3) Basic architectures
4) Coding: ~~Graph Nets library~~
   "PyTorch Geometric"
5) Graph Autoencoders; modelling transport phenomena } Next week

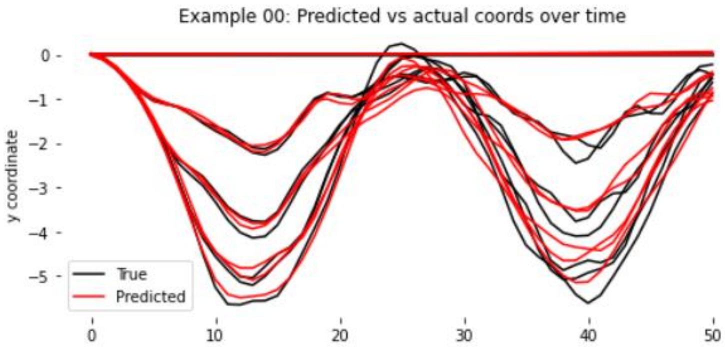
Data Driven Engineering II: Advaced Topics

Graph Neural Networks I

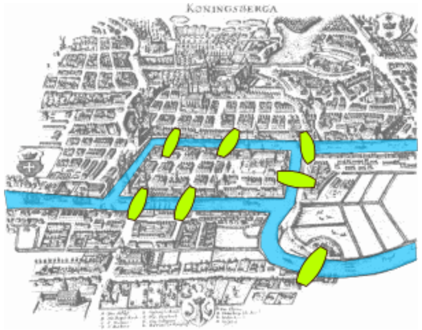
Institute of Thermal Turbomachinery
Prof. Dr.-Ing. Hans-Jörg Bauer







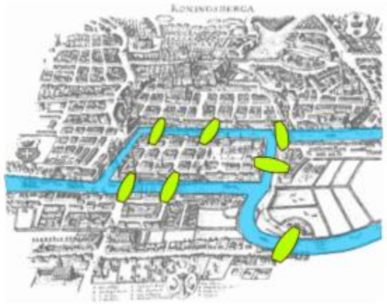
Seven Bridges of Königsberg

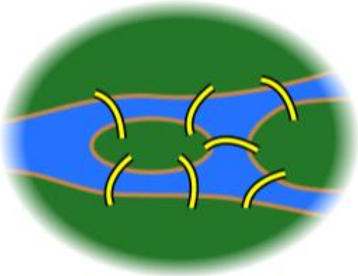


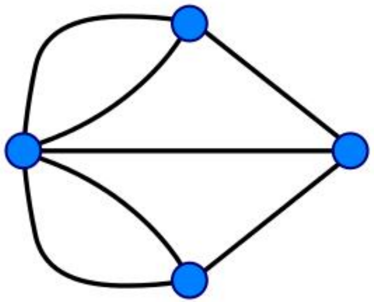
"Walk through the city that would cross each of those bridges once and only once"

Well, you can't.

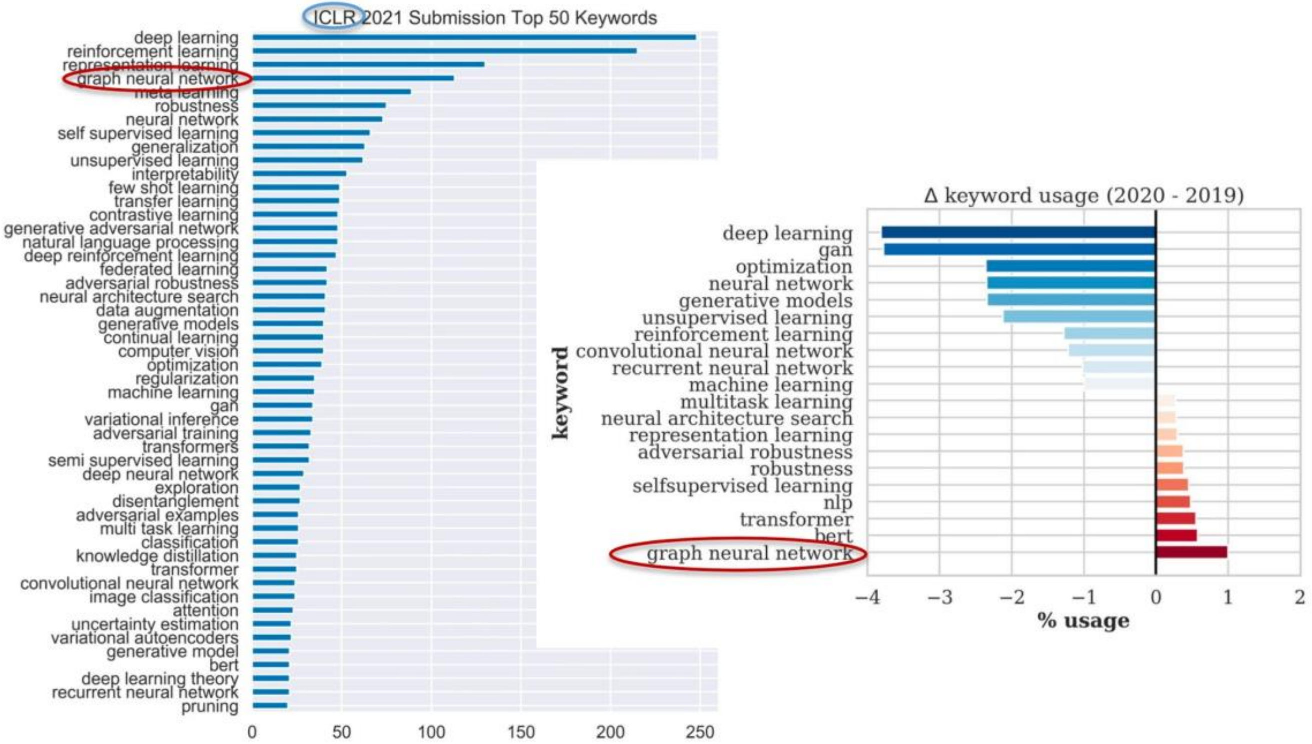
L. Euler



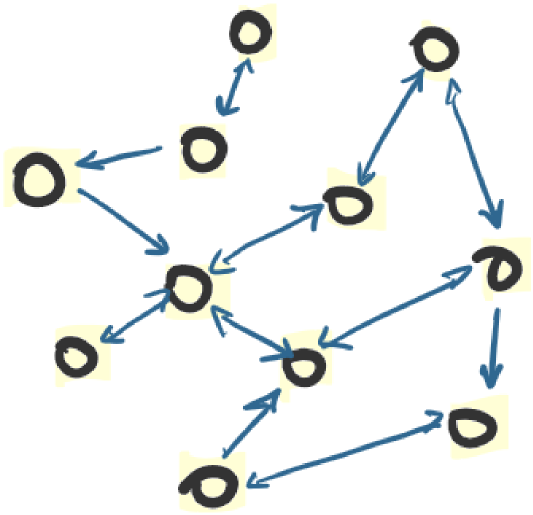




Graph Neural Networks: one of the hottest fields of AI



Graph Neural Networks :



Graphs : a way to represent what we know about the system including the relationships btw. entities.

Q How can we exploit relational structure for a better prediction ?

Graph Neural Networks :

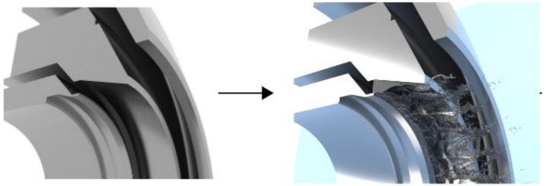
Graphs

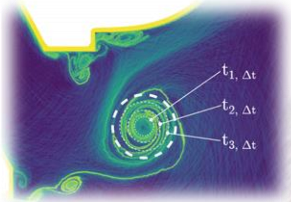
* Many data systems are "graphs"

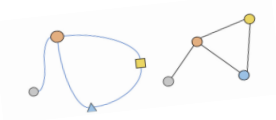
✓ Particle networks	✓ Social networks	✓ Mobility
✓ Disease modeling	✓ Recommend. systems	✓ IoT
✓ Multiphase flows	✓ Graph mining	✓ Codes
✓ Particle Physics	✓ Chemistry → Protein Folding	
✓ Robotics	→ Fingerprint	
✓ Image & text analysis	→ Rxn Models	
	→ Biomedical eng.	

Pre-Processing based on CAD-Model

SPH-Simulation of Primary Breakup







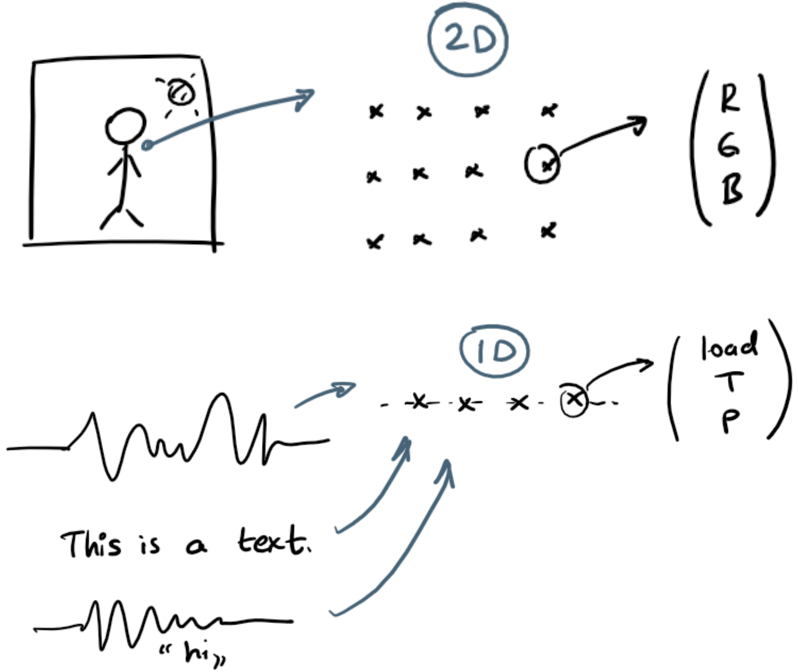
What we know already

* { CNN & RNN }

"regular & structured graphs"

~ deep learning ~

e.g.



Generalizing what we did...

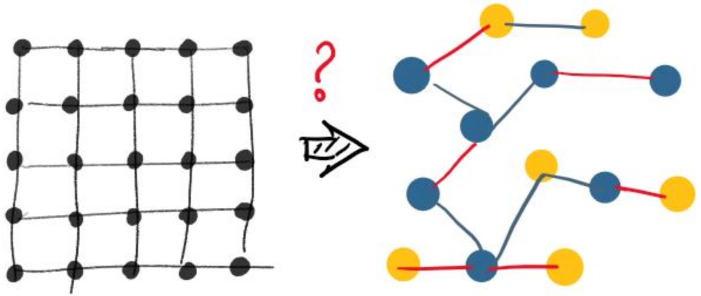
* GNN ← motivated by CNN

local connections

shared weights

hierarchical patterns captured

*



How to transform?

Graph Neural Networks :

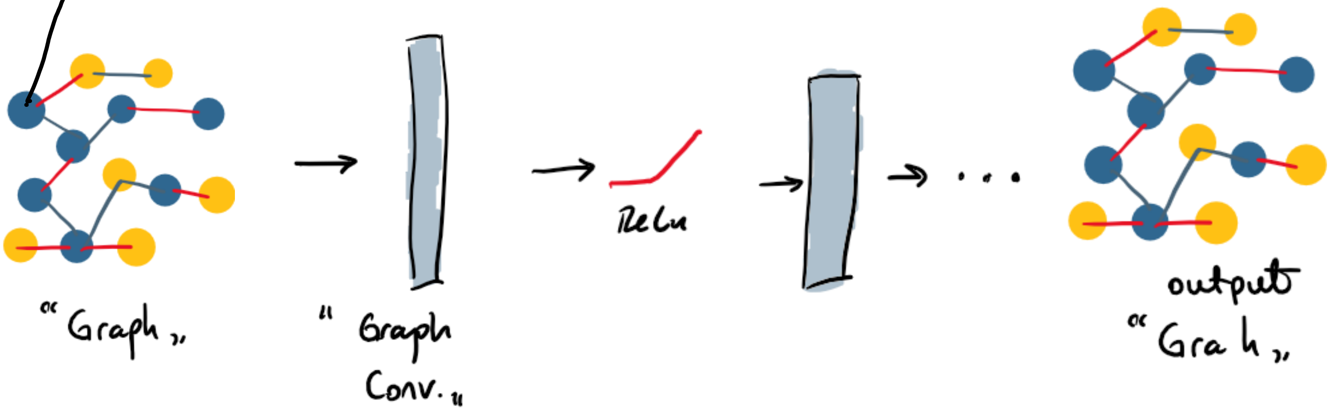
Every dot here is an instance

We are seeing all the training data at once.

IN CNN you know the position and location of each node, here not. Here we rely on the concept of representation learning.

Karlsruhe Institute of Technology

* What is the aim ?



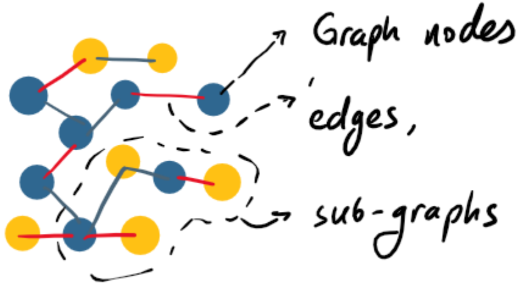
→ ...
- Nodes
- Connections
- Graph as a whole
- a decision

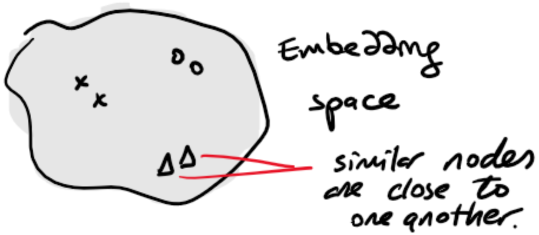
Each node can be a label

? how we can code it ?

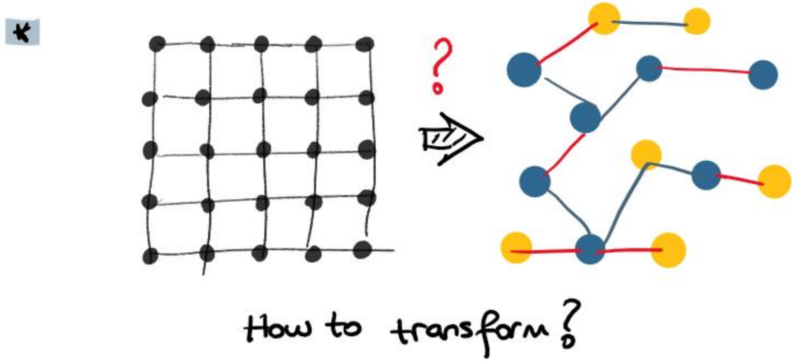
" Representation learning " } automate learning of features (embedding)

Embedding

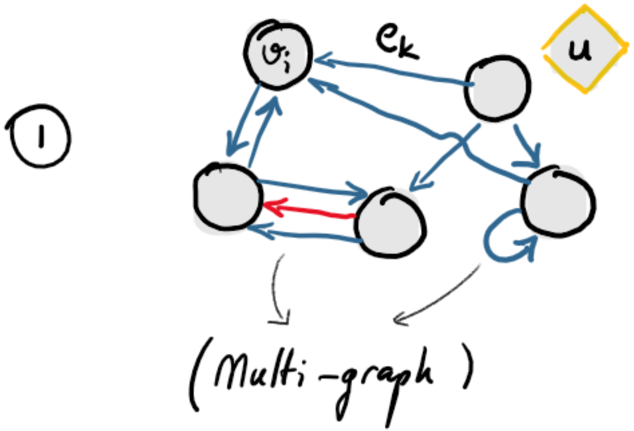




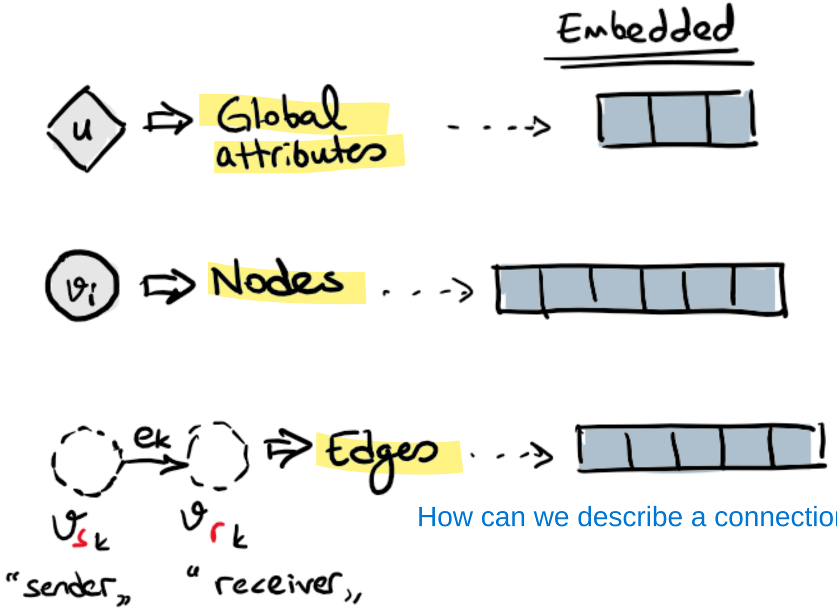
If they are not clustered, there is no chance to create a model and build up a function that can learn from the data.



Understanding the Graph :



In a CNN we still have a graph, is structured. You assign a weight to each node, but there is no information about the edges.



How can we describe a connection?

Understanding the Graph :

You cannot break it them.

(2) Graph := 3-tuple ; $G = (u, V, \epsilon)$

- u is for the whole graph ⇨ label, oarameter ($\vec{g}$) ...

- $V = \{ v_i \}_{i=1,N^v}$ $v_i \Rightarrow$ "particle i" ⇨ [x, y, z, u, v, w, m]

- $\epsilon = \{ e_k, r_k, s_k \}_{k=1,N^e}$

$e_k \Rightarrow$ Edge attribute

$r_k \Rightarrow$ receiver index

$s_k \Rightarrow$ sender index

$e_k = [1, k]$

There is a spring

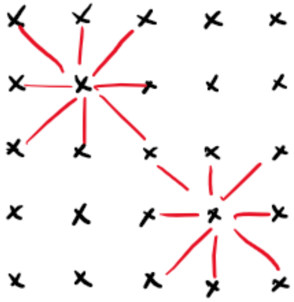
spring constant

Understanding Graph Network :

Convolution ⇒ $\underbrace{\text{"message passing" layers}}$

In CNN you know exactly the size of the kernel, you define the kernel 3x3 and apply a stride of 1.

node embeddings ⇒ info. about connections !
in a compressed format

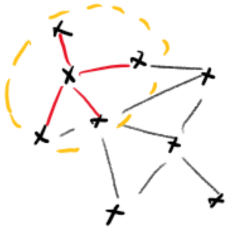


Conv.

- Kernel is moved.

↓

neigh. info.



graph conv.

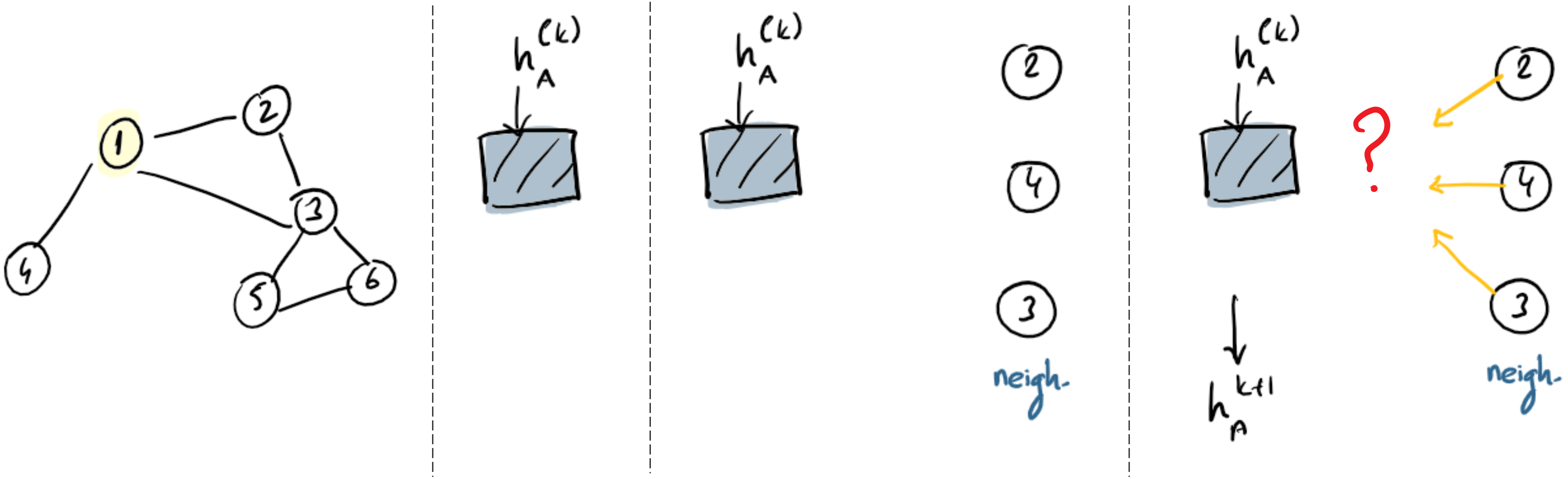
⇒ Connections are dynamic ↔ change

⇒ hidden state ↻ updated

$\underbrace{\text{hidden state updated}}$ hidden embedding

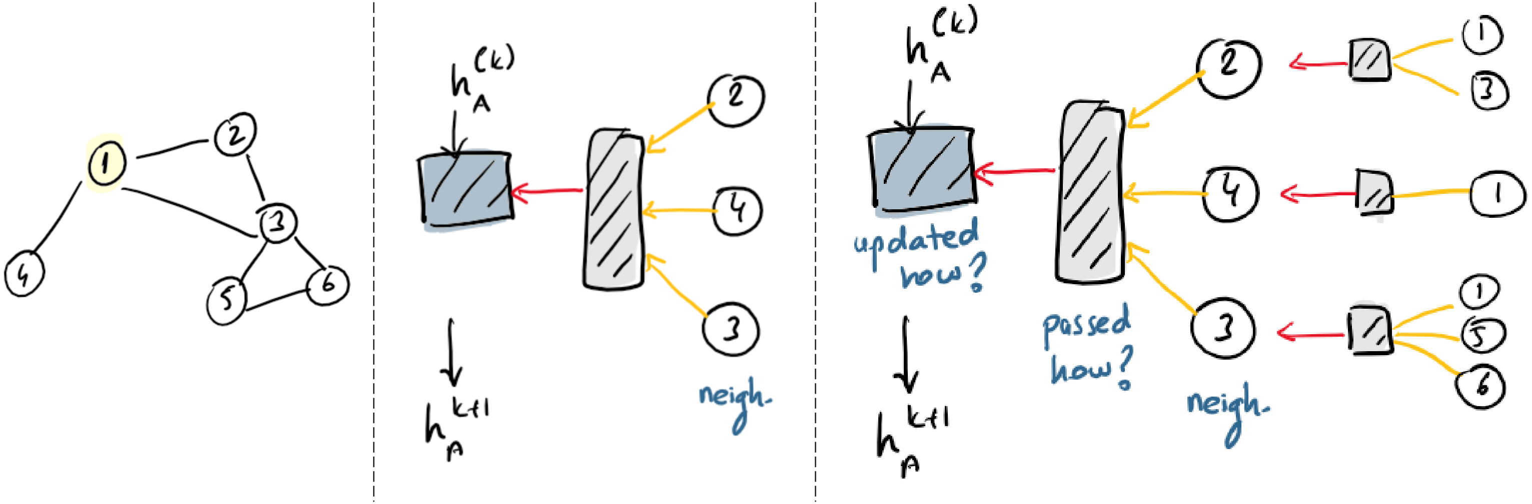
Understanding Graph Network :

idea ⇒ hidden states of nodes v_i updated (I) according to the info. passed (II) from neighbours (III)



Understanding Graph Network :

idea ⇒ hidden states of nodes v_i updated (I) according to the info. passed (II) from neighbours (III)



Understanding Graph Network :

idea ⇒ hidden states of nodes v_i updated (I) according to the info. passed (II) from neighbours (III)

$$* \quad h_i^{(k+1)} = \phi_{update}\left(h_i^{(k)}, \phi_{aggregate}^{(k)}\left(\{h_j^{(k)}, \forall j \in \mathcal{N}(i)\}\right)\right)$$

arbitrary differentiable functions

message from neighbour

One part of the hidden state is represented as for a RNN and another by a CNN.

Understanding Graph Network :

* Algorithm of a graph network

! edges, nodes, global $\Bigg\}$ (i) 3 update functions → edge, nodes, globals
(ii) 3 aggregate functions

1) Update edge attributes $e'_k = \phi^e(e_k, v_{r_k}, v_{s_k}, u)$

2) Aggregate edge att. per node $\bar{e}'_i = \rho^{e \to v}\{(e'_k, r_k, s_k)\}_{r_k = i,\, k = 1:N^e}$

Understanding Graph Network :

* Algorithm of a graph network

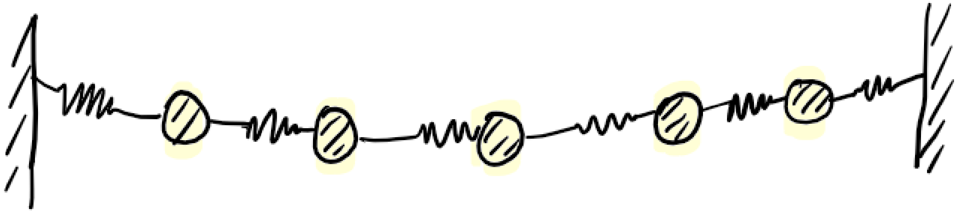
3) Update node attributes $v_i' = \phi(\bar{e}_i', v_i, u)$

4) Aggregate edge att. globally $\bar{e}' = \rho^{e \to u}\{(e_k', r_k, s_k)\}_{k=1:N^e}$

5) Aggregate node att. globally $\bar{v}' = \rho^{v \to u}\{(v_i')\}_{i=1,N^v}$

6) Update global attributes $u' = \phi^u(\bar{e}', \bar{v}', u)$

Eg



① Apply $\phi^e \rightarrow e'_k$ $\quad e_k \Rightarrow$ forces btw. two connected balls.

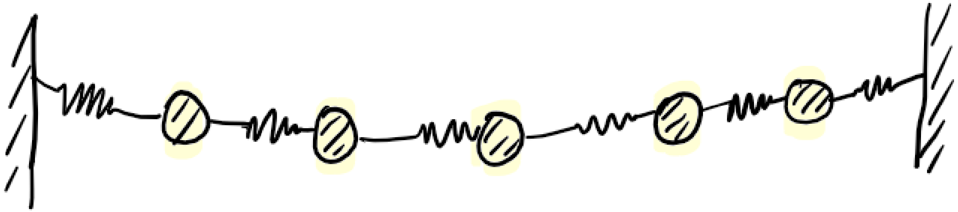
$\Downarrow$ get forces updated for each ball for each connection.

② $\rho^{e\rightarrow v} \Rightarrow \bar{e}'_i$ $\quad \bar{e}_i \Rightarrow \sum$ force acting on i^{th} ball.

③ $\phi^v \Rightarrow v'_i$ $\quad v_i \Rightarrow$ position, velocity, KE of ball i;

$\phi^v \hookrightarrow$ updates v_i as a func. $(\bar{e}'_i, v_i, u)$.

Eg

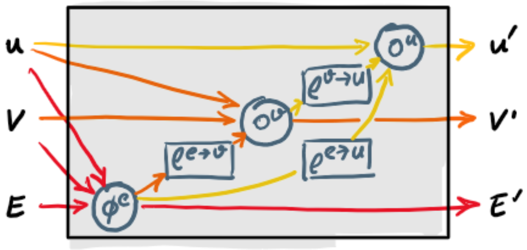


④ $\rho^{e \to u} \Rightarrow \bar{e}'$ $\qquad \bar{e} \Rightarrow \underbrace{\sum \text{forces}}_{=\emptyset}$ } Global info.

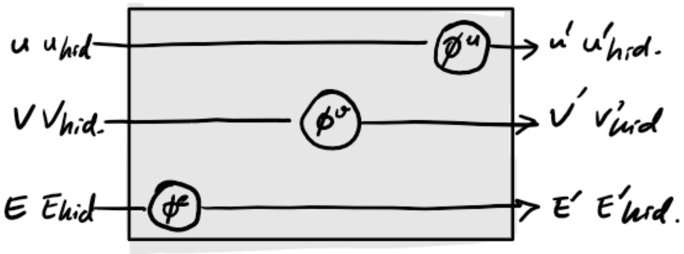
⑤ $\rho^{v \to u} \Rightarrow \bar{v}'$ $\qquad \bar{v} \Rightarrow \sum KE$ } Global info.

⑥ $\phi^{u} \Rightarrow u'$ $\qquad u' \Rightarrow \sum \text{energy}$ } Global info.

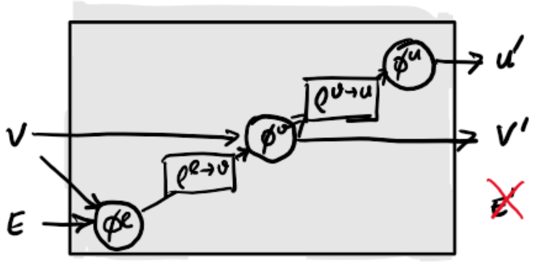
Graph Neural Networks



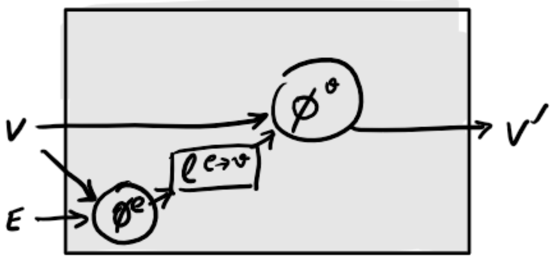
Full GN block



Independent recurrent block



Message-passing NN.



Non-local N.N.

GNN 101: $\phi = ?\ ;\ \ell = ?$

$$h_i^{(k)} = \underbrace{\sigma}_{\text{ReLu, tanh, ...}}\Big(\underbrace{W_{self}^{(k)}}_{\text{Trainable}} \underbrace{h_i^{(k-1)}}_{\text{self state}} + \underbrace{W_{neigh.}^{(k)}}_{\text{Trainable}} \sum_{j \in \mathcal{N}} \underbrace{h_j^{(k-1)}}_{\text{neigh. steps.}} + \underbrace{b^{(k)}}_{\text{bias term}}\Big)$$

Eg

$$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow G_3 \cdots \rightarrow G_T \Rightarrow h_i^T$$

The Graph can be updated.

Output; $x_i = \sigma(w^T h_i^T + b)$

Loss; $\mathcal{L} = \underbrace{y_i \cdot \log x_i + (1-y_i)\log(1-x_i)}_{\text{binary cross entropy.}}$

Improvements over basic GNN :

Neighbourhood normalization

* Aggregation ⇒ $\sum$ operation } not very stable & sensitive to node degrees.

* Normalize the agg. operation by degree ;

divided by the number of neighbours

$$\text{message} = m_{\mathcal{N}(i)} = \sum_{j \in \mathcal{N}(i)} h_j \Big/ \underbrace{|\mathcal{N}(i)|}$$

* Symmetric normalization ;

$$m_{\mathcal{N}(i)} = \sum_{j \in \mathcal{N}(i)} \left(h_j \Big/ \sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|} \right)$$

Graph Convolutional Networks

* GCN ⇒ Popular Baseline model

* GCN := Symm.-normalized aggregation + Self-loop update

$$h_i = \sigma\Big(W \underbrace{\sum_{j \in \mathcal{N}(i) \cup \{i\}}} \frac{h_j}{\sqrt{|\mathcal{N}(i)|\,|\mathcal{N}(j)|}} \Big)$$

agg. is taken over the neigh. & the node itself.

Like in CNN

! No need to define update function

! Info. coming from the nodes // neighbors ?

Aggregation: A key step for success

Karlsruhe Institute of Technology

* $\sum$ maybe not the best option. → normalization
→ something better?

If assigning a weight is not the best idea, let's put a MLP. May be its a better idea.

idea 1

$$m_{\mathcal{N}(i)} = MLP_\theta \Big(\underbrace{\sum_{j \in \mathcal{N}(i)}}_{\text{min / max}} MLP_\phi (h_j) \Big)$$

⇒ permutation invariant

idea 2

$$m_{\mathcal{N}(i)} = MLP_\theta \Big(\frac{1}{|\Pi|} \sum_{\pi \in \Pi} LSTM \left(h_{j_1}, h_{j_2}, \dots h_{j_{\mathcal{N}(i)}} \right)_{\pi_i} \Big)$$

set of permutations

permutation sensitive

Aggregation: A key step for success

idea 3

Not all neigh. are equally important $\Rightarrow$ Attention

* GAT := Graph Attention Network $\Rightarrow$ $m_{\mathcal{N}(i)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij} h_j$

Here we are just putting a mask on it.

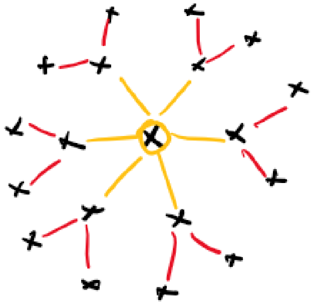
* Attention models $\Rightarrow$ $\alpha_{ij} = \dfrac{\exp(h_i^T W h_j)}{\sum_{j' \in \mathcal{N}(i)} \exp(h_i^T W h_{j'})}$

Update Methods

$$\ast \quad h_i^{(k+1)} = \phi_{update}\left(h_A^{(k)}, \; \phi_{aggregate}^{(k)}\left(\{ h_j^{(k)}, \; \forall k \in \mathcal{N}(i) \} \right) \right)$$

In the first Convolution, every node i

Issue: Over-smoothing ⇒ node info. "washed out"



@ each rolling; h_i will changed by neigh.
& their neigh;
& their neigh;
...

Update Methods

Solution $\Rightarrow$ CNN $\rightarrow$ vector concatenations

$\rightarrow$ skip connections

(i) $$\text{update}^{*}(h_i, m_{\mathcal{N}(i)}) = [\text{update}(h_i, m_{\mathcal{N}(i)}) \oplus h_i]$$

$\rightarrow$ message from neighbours

$\rightarrow$ current representation of the node

(ii) $$\text{update}^{**}(h_i \; m_{\cdot}, \alpha) = \alpha \; \text{update}(h_i, m_{\mathcal{N}(i)}) + (1-\alpha) h_i$$

$\Rightarrow$ linear interpolation

$\Rightarrow$ learnable

Update Methods

Solution ⇒ RNN → Gated information prop.

Agg. Function := Receive observation from neighbours & update hidden states

$$h_i^k = \begin{matrix} \text{GRU} \\ \text{LSTM} \end{matrix} \left(h_i^{k-1}, m_{\mathcal{N}(i)}^k \right)$$

hidden state ⇒ hidden embedding

observation x^t ⇒ $m_{\mathcal{N}(i)}$ (agg. info. from neigh.)

Graph Pooling :

* So far ⇒ Node embeddings } Graph level information

$$z_G = \frac{\sum_{i \in V} h_i^T}{fn(|V|)}$$

last hidden emb.

normalization

sum ⇓ mean

* Use LSTM update + Attention mechanisms
* Using clustering on graph (~CNN) ↘ fclustering must be differentible

We have to build the graph, we need to use different libraries and so on.

PyG. Py torch Geometric is like Scikitlearn. Dataset Sheet is great to understand which model is better to use for the specific problem at hand.

colab