Graph Neural Networks

Shah Rukh Qasim
Physik Institute, University of Zurich
shah.rukh.qasim@cern.ch

29.10.2024

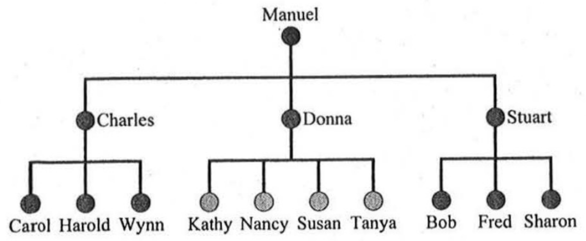
Maurizio Pierini (CERN)
Thea Aarrestad (ETH Zürich)

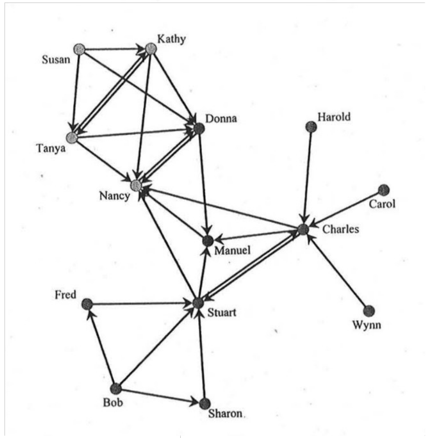
What are we doing today?

- Graph Neural Networks (GNNs)
- Part 1 (Lecture)
    - Why GNNs?
    - And then quickly go to a simple modern GNN / with message passing
        - So you can already get the understanding to be able to use GNNs
    - And then we’ll have a little bit of a history lesson
    - And possibly discuss some advanced topics as well
- Part 2 (Tutorial)
    - We’ll train a GNN for molecule classification on collab

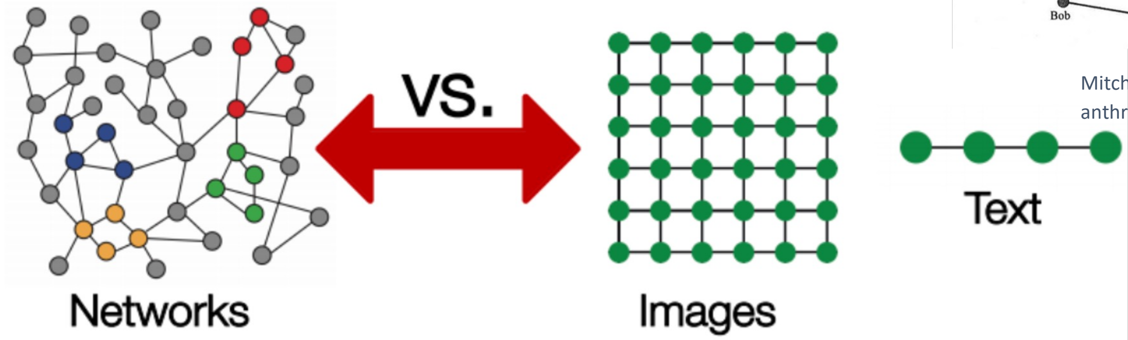
Why GNNs?

- A lot of data cannot be represented as fixed structure
  - That is sequences
  - or grids
- So we need a more generic data structure: graphs
- Graphs: $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$
  - Nodes / vertices ($\boldsymbol{V}$): relate to an "object" in your data
  - Edges ($\boldsymbol{E}$): captures the relationship between
  - Let's say node features are h_u (corresponding to the node u)
  - And the edge features are e_{vw} (corresponding to edge e)
- Graphs are very generic
  - Can represent sequences and images
  - And sets





Mitchell, J. Clyde. "Social networks." Annual review of anthropology 3 (1974): 279-299.

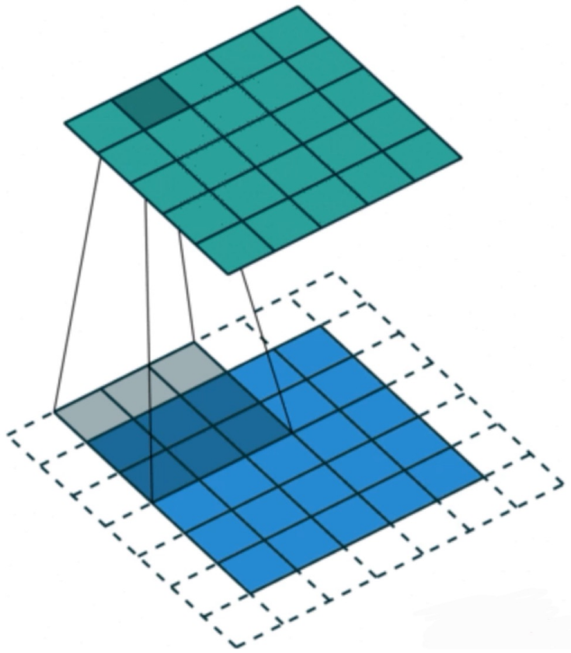


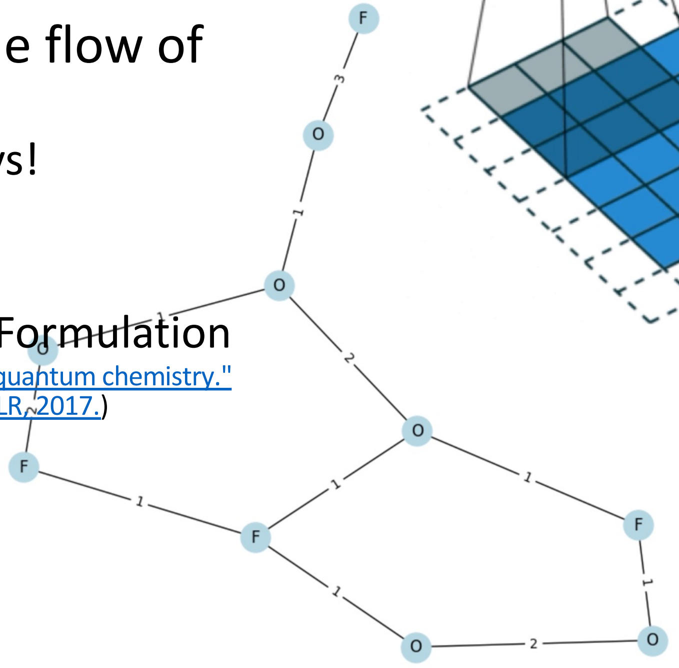
https://web.stanford.edu/class/cs224w/

GNNs

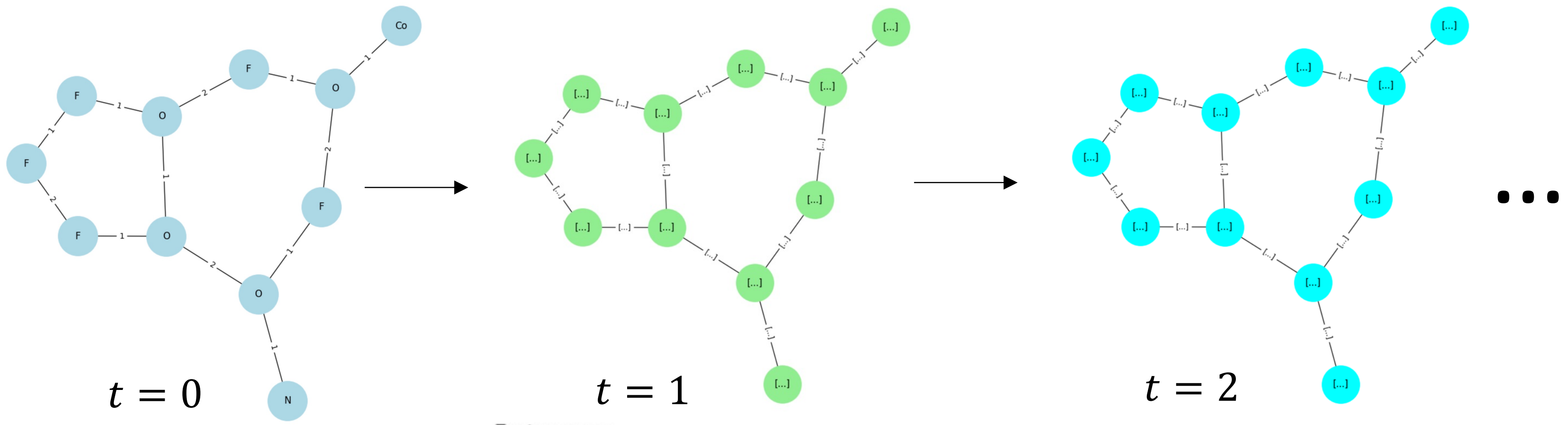
- GNNs: Pairing up the graph structure with neural networks
- Use the graph structure for the flow of information
  - Can also be used in other ways!

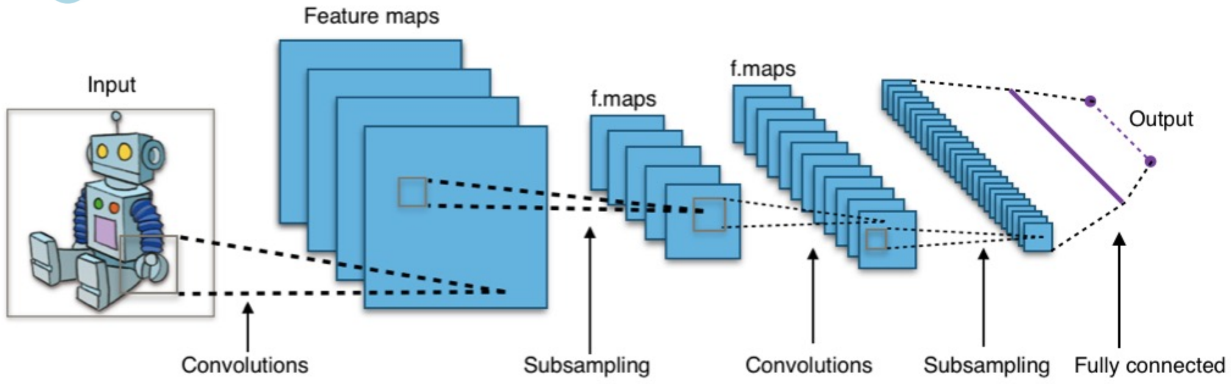
- In the next slides:
  - We'll adapt Message Passing Formulation
    ([Gilmer, Justin, et al. "Neural message passing for quantum chemistry." International conference on machine learning. PMLR, 2017.](#))





GNNs and CNNs

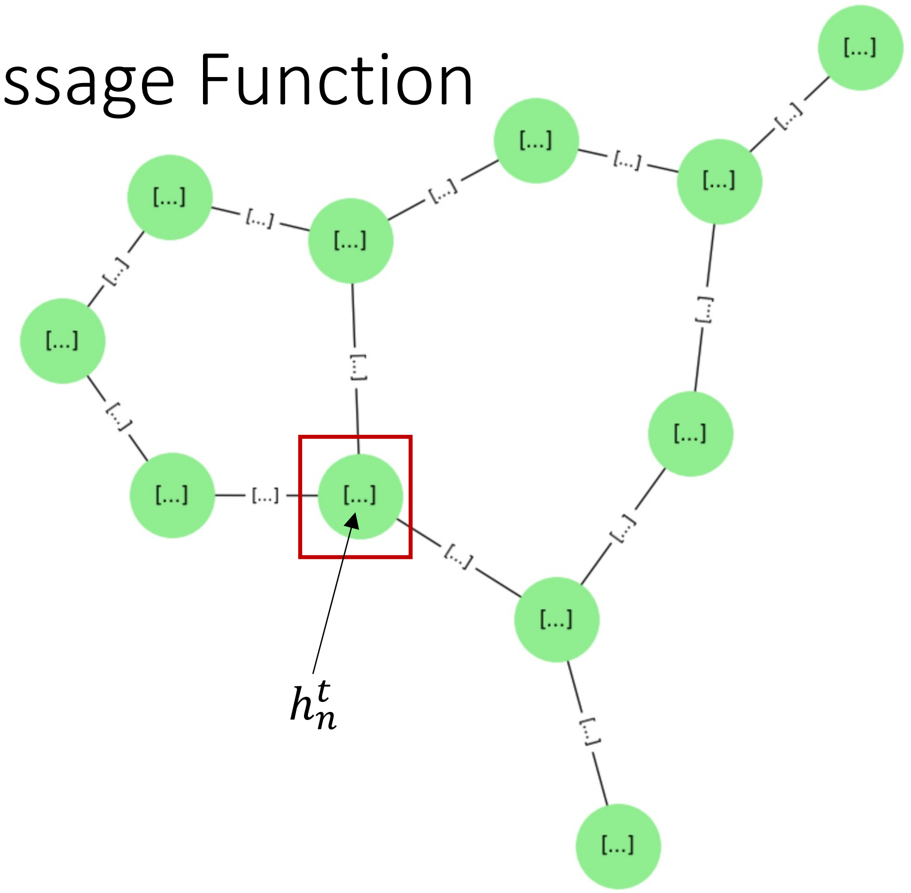




By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

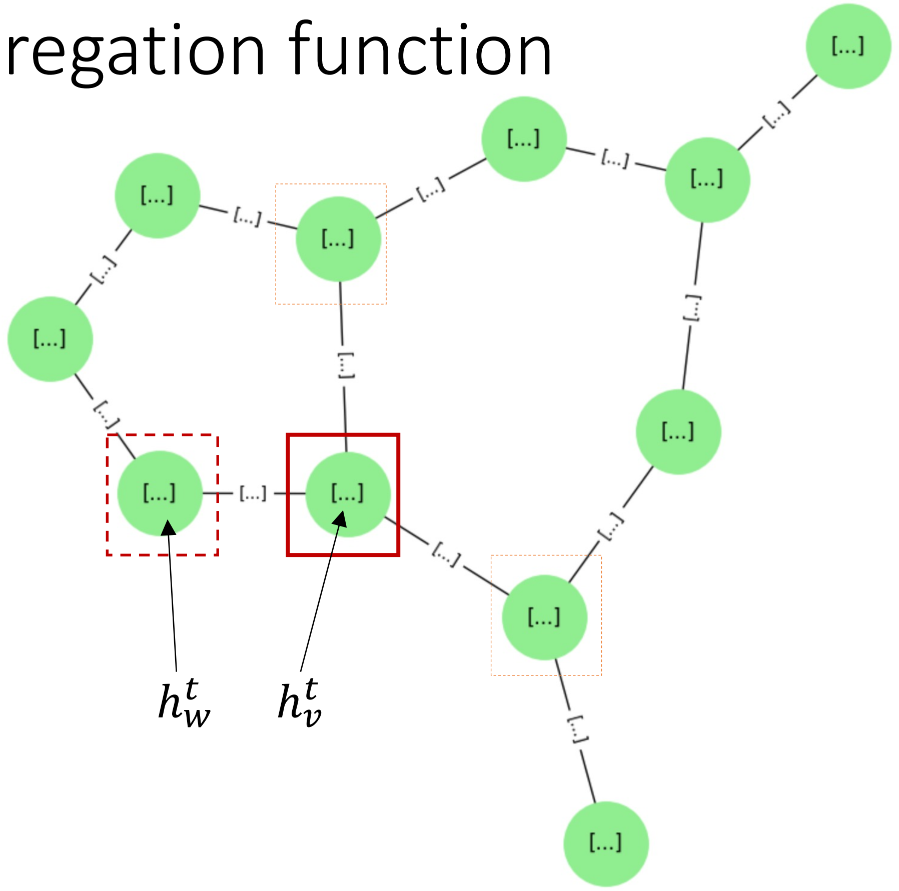
Graph Message Passing: Message Function

- Ignore edge features at the moment
- Every node has a feature vector h_n^t
- Need to transform it into a message to send to it's neighbours
- How?
- Apply a dense layer to it
- This is called a message function (M_t)
    - Message = $M_t(h_n^t)$
- Are the weights different for different nodes?
    - No



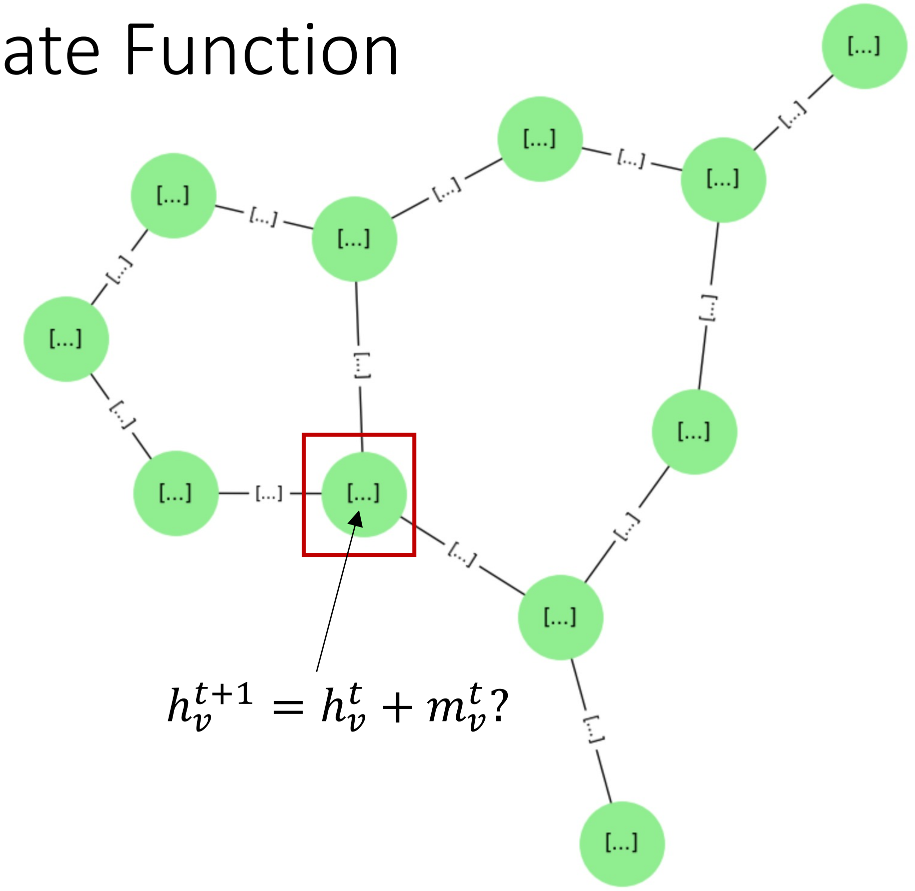
Graph Message Passing: Aggregation function

- Every node (v) collects messages from its neighbours (w)
  - $m_v^t = \prod_{w \in \mathcal{N}(v)} M_t(h_w^t)$
  - Π is not muliplication
- What is Π ?
  - This is called **aggregation function**
  - We need it to be permutation invariant
  - Easiest and most common: sum!
  - $m_v^t = \sum_{w \in \mathcal{N}(v)} M_t(h_w^t)$
- Other aggregation functions:
  - Mean, min, max



Graph Message Pasing: Update Function

- Let's look at node v that has collected messages from its neighbours as m_v^t
- The nodes previous features were: h_v^t
- How do we merge them together for the next layer $(t+1)$?
    - $h_v^{t+1} = U_t(h_v^t, m_v^t)$
- What is U_t?
    - Again, a dense layer!
    - It is called an update function
- Before you apply a dense layer, how do you join the two vectors together (h_v^t, m_v^t)?
    - Concatenation
- You can also use only concatenation as the update function
    - But it will blow up the number of features as you add more layers (not ideal)



Graph Message Passing

- Generalization: A message passing function (M_t) operates on the hidden features of the source node, **the destination node and the features of the edges between them**
- The M_t is generally a dense layer
- Mean / min / max / sum are aggregation functions (permutation invariant)

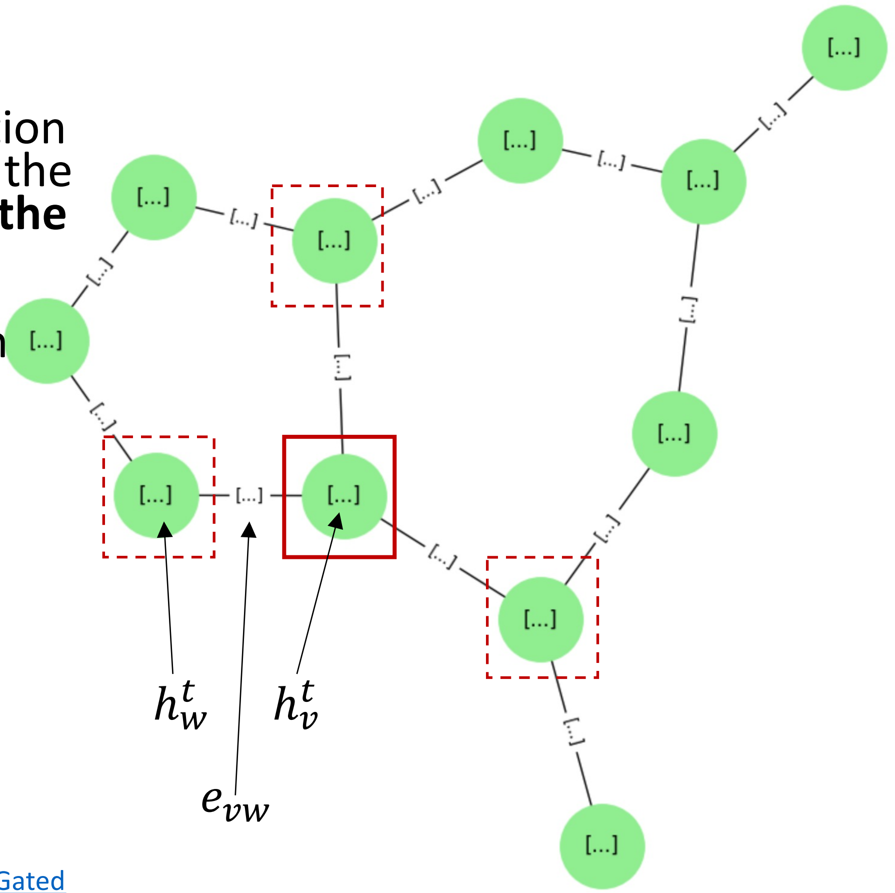
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

- And then the vertex features are updated
  - simplest: also, a dense layer

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

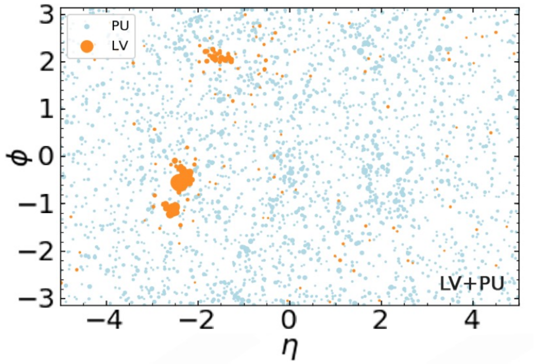
- GRUs have been tried too

  [Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. ICLR, 2016.]

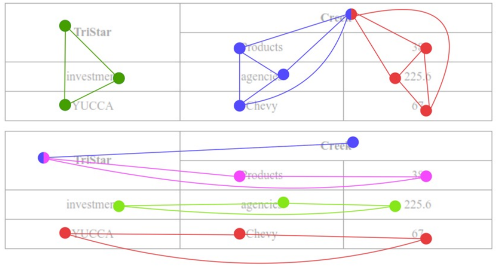


GNN tasks

- Graph level
    - Example: Molecular graph classification
    - Pool information from all the nodes
        - Mean / Min / Max
    - Output: The pooled vector
        - Also the loss function here
        - Generally after a couple of dense layers from the pooled information
- Node level
    - Example: segmentation
    - Output: the feature vector at the last layer of your GNN
- Edge level
    - Example: In a social network, if two people are friends or not
    - Don’t pool information from the graph
    - Get the output at every edge
        - If the edge features don’t exist, do subtraction or concatenation



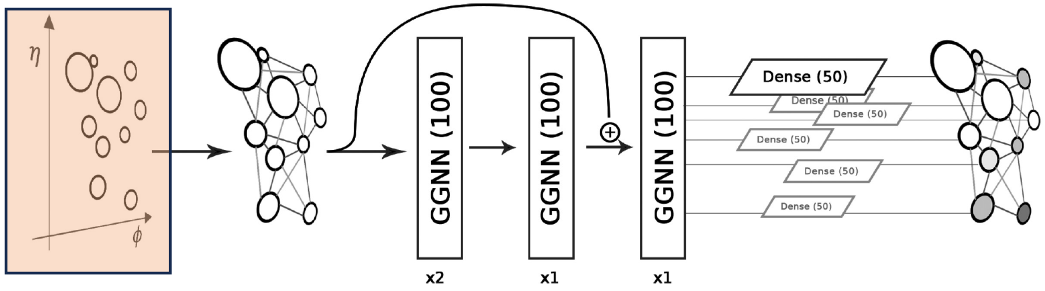
Martínez, J.A., Cerri, O., Spiropulu, M., Vlimant, J.R. and Pierini, M., 2019. Pileup mitigation at the Large Hadron Collider with graph neural networks. The European Physical Journal Plus, 134(7), p.333.



Qasim, Shah Rukh, Hassan Mahmood, and Faisal Shafait. "Rethinking table recognition using graph neural networks." 2019 International Conference on Document Analysis and Recognition (ICDAR). IEEE, 2019.

Graph Neural Networks: Practical Considerations

- Use one or more dense layers without message passing at the start
- Use one or more dense layers at the end after message passing
- Don't have to use a single dense layer as message passing function
  - Can use more for a more complex function
  - Balance between the complexity and the frequency of the messages
- Can play with concatenation
- What stays the same?
  - Almost everything apart from the computation graph
    - Mini batch training
    - Train/validation split
    - Batch norm/dropout
    - Loss function
    - Adam optimizer
    - etc

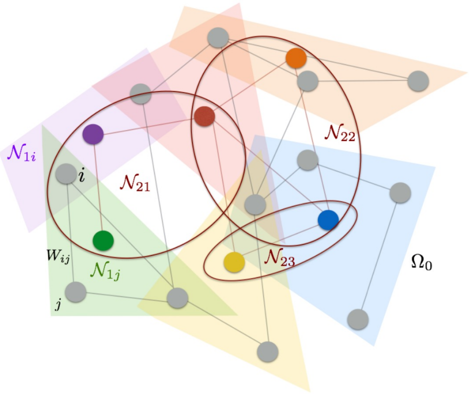


An example of a GNN architecture

Martínez, J.A., Cerri, O., Spiropulu, M., Vlimant, J.R. and Pierini, M., 2019. Pileup mitigation at the Large Hadron Collider with graph neural networks. The European Physical Journal Plus, 134(7), p.333.

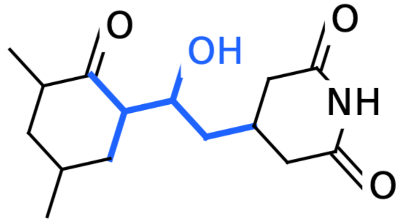
History: Spectral Networks

- Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral networks and locally connected networks on graphs." arXiv preprint arXiv:1312.6203 (2013).
- They tried to generalise CNNs beyond the regular array dataset paradigm
- They replaced the translation-invariant kernel structure of CNNs with hierarchical clustering
- Introduced spectral convolutions
  - Done using the laplacian matrices of the graph
  - Later extended by Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
- Today you won't use the spectral domain and perform message passing directly in the spatial domain



History: Message Passing

- Traced back to
  Duvenaud, David K., et al. "Convolutional networks on graphs for learning molecular fingerprints." Advances in neural information processing systems 28 (2015).
- Introduced "convolutional neural network that operates directly on graphs"
- Different language, similar idea



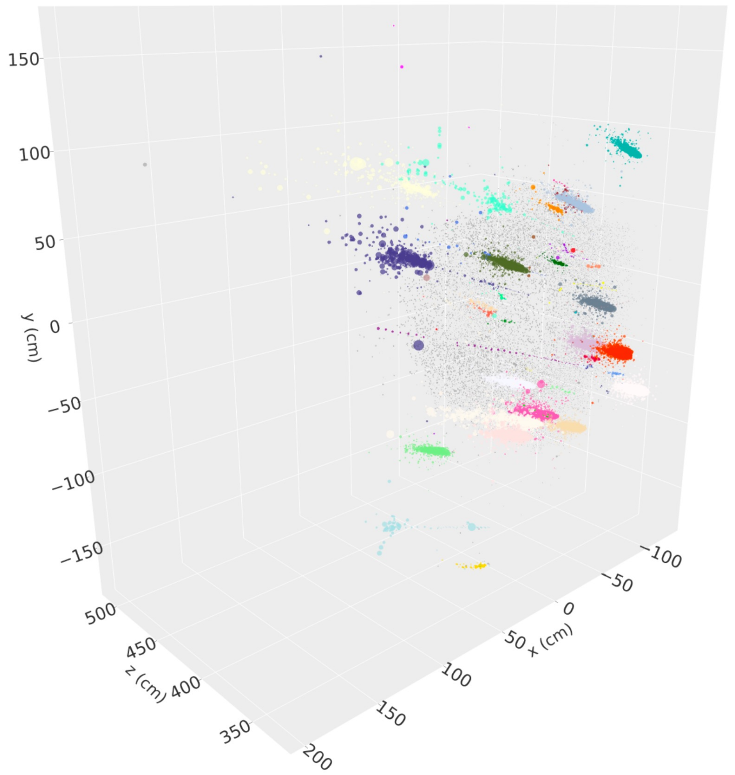
Algorithm 2 Neural graph fingerprints

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

What about point clouds?

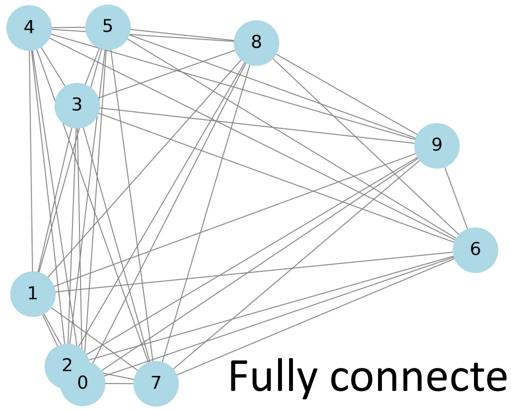
- Your non structured data is a bunch of points
  - aka a point cloud
- No graph structure defined
  - Very important for science, especially for physics
  - A data from a calorimeter or a tracker doesn't have a graph defined
  - A set of particles in a jet don't have a graph defined

- How do I apply machine learning here?

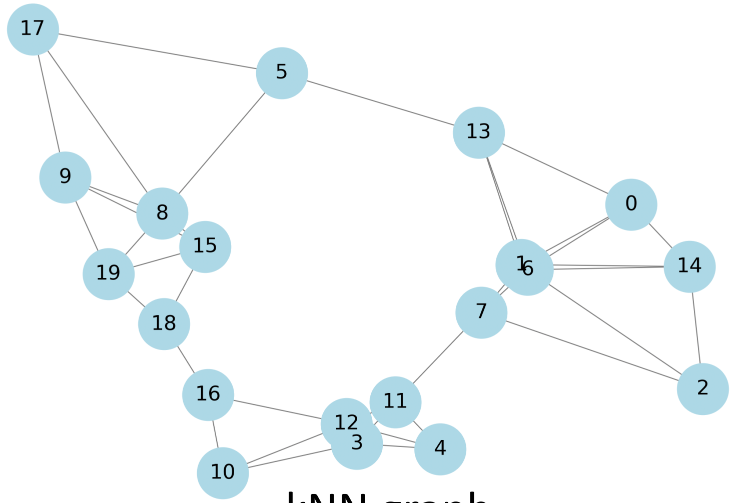


GNNs on Point Clouds

- Answer 1:
  - Connect everything to everything
  - Introduce virtual nodes
  - Will not scale!
- Answer 2:
  - Build a graph locally in
    - Radially in the input domain...
    - Or k Nearest Neighbour (kNN)



Fully connected graph



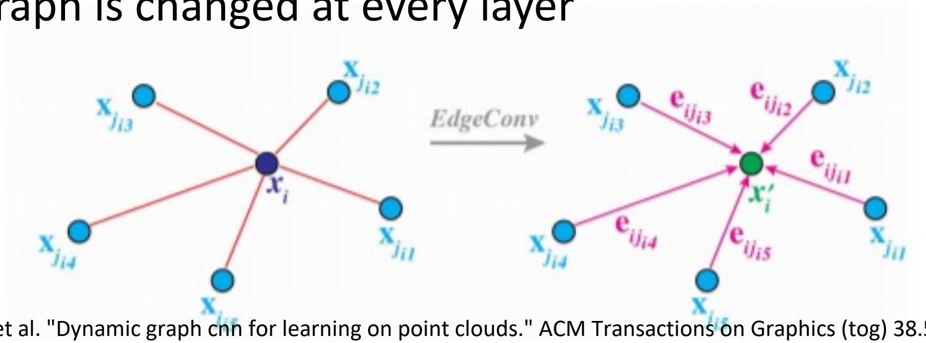
kNN graph

Better answer: Dynamic GNNs

- Build the graph in a dynamically learned space
- Feature vector for every point
- You apply a neural network to feature vectors of every point to get feature space F_S
- Build a graph as KNN of every node using euclidean distance in F_S
    - So you have N*K edges
    - Perform message passing with permutation invariant aggregation functions
    - Dynamic GNN:
        - The KNN graph isn't built in the original space but in a learned feature space

Dynamic GNNs: Edge Conv

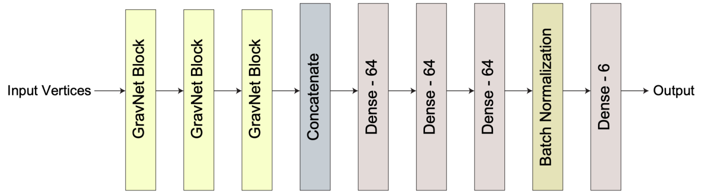
- kNN graph is built in h_t or x^t
- And then the next set of hidden features are constructed as:
  - $x_i^{t+1} = m_i^{t+1} = \sum_{j \in \mathcal{N}(i)} M_t(x_i^t, x_j^t - x_i^t)$
- Can also use min / max functions
- Same story: Do it a bunch of times to build your GNN
  - Different: The graph is changed at every layer

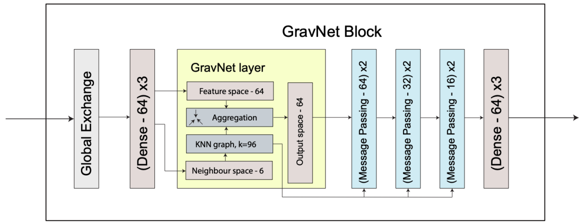


Wang, Yue, et al. "Dynamic graph cnn for learning on point clouds." ACM Transactions on Graphics (tog) 38.5 (2019): 1-12.

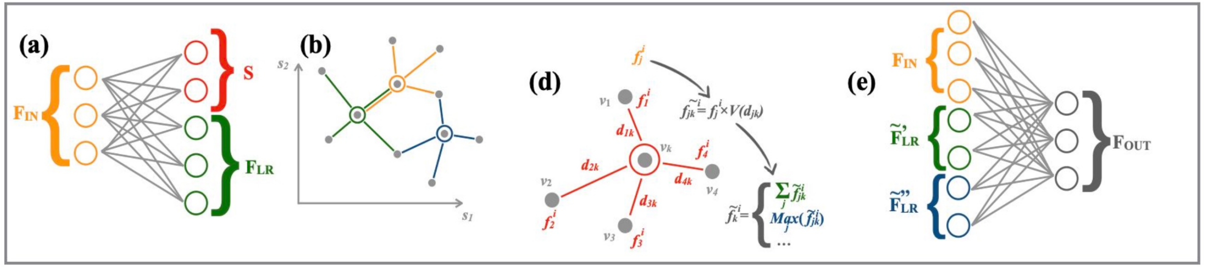
Dynamic GNNs: GravNet

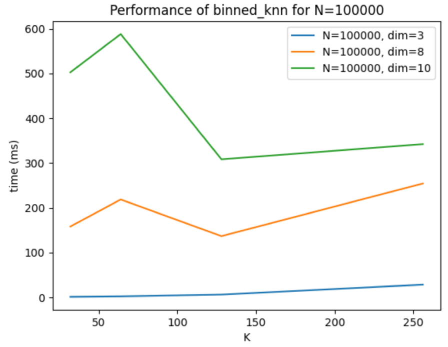
- Edge Conv:
  - You build the graph in the feature space at step t
  - Hope that the graph is good
    - Works very well in practice
- Can we do better?
  - At every layer, transform the features into two different spaces
    - Coordinate space (S) – Low dimensional
    - Feature space (F_{LR}) – Higher dimensional
  - F_{LR} is weighed by (neg exp. of) distance between nodes in S
  - In this way, the coordinate space is learned such that the nodes are brought closer if it helps the gradient
- Also much more performant:
  - kNN building is very expensive: dominates the time taken
  - Much faster in smaller dimensions





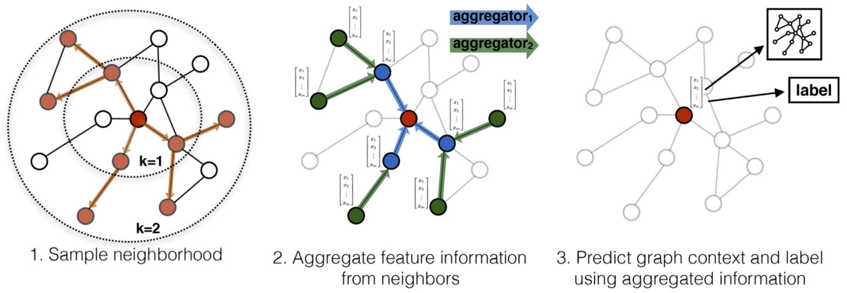
Qasim, Shah Rukh, et al. "Learning representations of irregular particle-detector geometry with distance-weighted graph networks." The European Physical Journal C 79.7 (2019): 1-11.





GNNs on large graphs

- GraphSAGE (SAmple and AGgregate)
  Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017).
- Sample in the neighbourhood
- And then do the aggregation
- Here, the authors also explored LSTMs
  - Not permutation invariant though
- Applied on large graphs: citation and reddit graphs


k=1
k=2
1. Sample neighborhood
aggregator$_1$
aggregator$_2$
2. Aggregate feature information from neighbors
label
3. Predict graph context and label using aggregated information

Graph Attention Networks

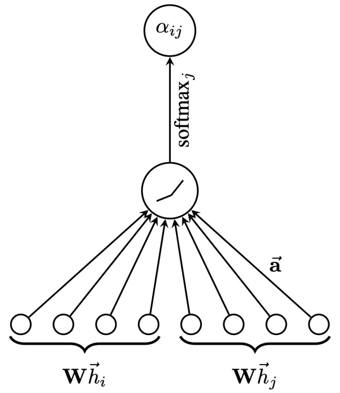
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y., 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

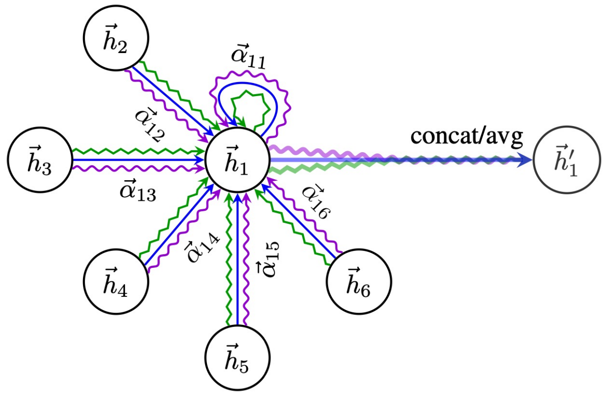
$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j)$$

$$\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$$

$$\vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

- ”Attention” on messages
- Maurizio will go into attention in more detail on November 19 when when discussing transformers
- The paper is more for reference here as an important GNN paper





Summary

- GNNs are powerful and work on very different types of datasets
- And now also very popular
- Transformers overshadow everything
  - But in a lot of cases, GNNs make more sense
- Have been applied to a large sets of problems across multiple domains
  - Traffic forecasting
  - Molecule property prediction
  - Recommendation systems
  - Reconstruction in particle physics
  - Simulation in particle physics
  - the list goes on…

GNN Hands-on Tutorial

Software Libraries for the use of GNNs

1. PyTorch: PyTorchGeometric / PyG: https://pyg.org/
    - Academia standard
2. **TensorFlow: TF-GNN**
    - We are going to use this to keep compatibility
    - Can use tools like keras here
3. Deep Graph Library (DGL): Framework agnostic

Copy the notebook

- https://colab.research.google.com/drive/1rmeb4qIIbZ6WHxeqBmVzahPwKyur7skR?usp=share_link
- Shorter link:
  - bit.ly/3YCFCtf
- Copy to save

- For slides: go to the very last of the notebook where there is a link to the slides

Quick primer to NetworkX

- Stop at **Checkpoint #1**
- NetworkX is a Python library for creating and analyzing graphs and networks
- Also provides tools for visualization with matplotlib
- Has a bunch of classical graph algorithms (matching, clustering etc)
- We are going to use it to plot our graphs

```
my_graph = train_graphs[5]
my_graph_nx = graph_tensor_to_networkx(my_graph)
nx.draw(my_graph_nx)
```

Quick primer to NetworkX

- Stop at **Checkpoint #2**
- We plot with the function we provided
    - with ids and without ids
- Add another node
    - And connect it to one of the existing ones
- Plot it again

```
my_graph = train_graphs[5]
my_graph_nx = graph_tensor_to_networkx(my_graph)
# nx.draw(my_graph_nx)
plot_networkx_graph(my_graph_nx, plot_with_ids=False)
```

Graph Spec

1. Stop at CHECKPOINT #3
2. Not always needed
   a. keras needs it
3. Looks scary piece of code but it is not doing much
4. Simplified for you below
5. If you understand the basic concept
   a. Ask chatgpt to figure out the technical details for you

To start

1. Stop at **CHECKPOINT #4**
2. Define the input
3. The graph is then merged across batch elements
   a. Nothing is happening here in reality

```
# Model building with Keras's Functional API starts with an input object
# (a placeholder for the eventual inputs). Here is how it works for
# GraphTensors:
input_graph = tf.keras.layers.Input(type_spec=graph_tensor_spec)

# IMPORTANT: All TF-GNN modeling code assumes a GraphTensor of shape []
# in which the graphs of the input batch have been merged to components of
# one contiguously indexed graph. (There are no edges between components,
# so no information flows between them.)
graph = input_graph.merge_batch_to_components()
```

Simple transformations

1. Apply a simple linear transformations to the initial features
   - Both for the nodes and edges
   - Similar to applying 1x1 conv
2. No message passing here

```
# Nodes and edges have one-hot encoded input features. Sending them through
# a Dense layer effectively does a lookup in a trainable embedding table.
def set_initial_node_state(node_set, *, node_set_name):
    # Since we only have one node set, we can ignore node_set_name.
    return tf.keras.layers.Dense(node_dim)(node_set[tfgnn.HIDDEN_STATE])

def set_initial_edge_state(edge_set, *, edge_set_name):
    return tf.keras.layers.Dense(edge_dim)(edge_set[tfgnn.HIDDEN_STATE])

graph = tfgnn.keras.layers.MapFeatures(
    node_sets_fn=set_initial_node_state, edge_sets_fn=set_initial_edge_state)(
    graph)
```

Add another dense layer in between

- Before checkpoint 5, add two dense layers

Message Passing

- From the documentation
  - `This layer can compute a convolution over an edge set by applying the passed-in message_fn for all edges on the concatenated inputs from some or all of: the edge itself, the sender node, and the receiver node, followed by pooling to the receiver node.`

Message Passing: Tasks

- Add three message passing layers
- In the first one, we are only using the sender node features
- In the second one, we are using the sender node features and the edge features
- In the third one, we are using the sender node features, the edge features and the receiver node features

Graph Message Passing

- Generalization: A message passing function (M_t) operates on the hidden features of the source node, **the destination node and the features of the edges between them**
- The M_t is generally a dense layer
- Mean / min / max / sum are aggregation functions (permutation invariant)

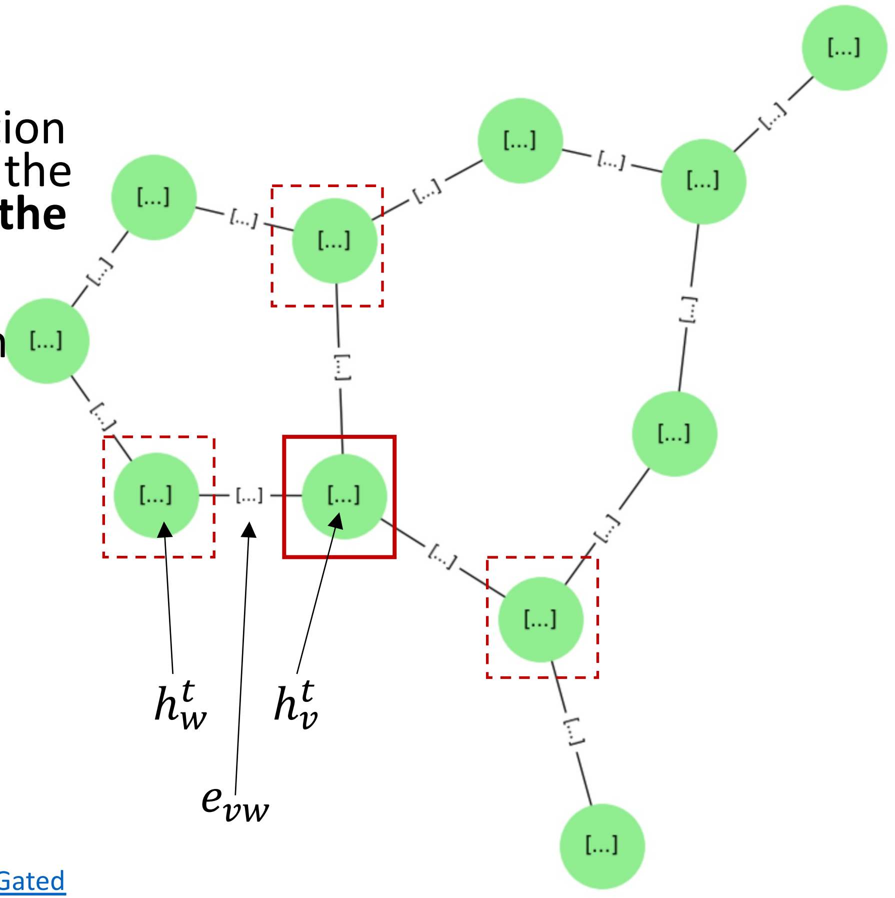
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

- And then the vertex features are updated
  - simplest: also, a dense layer

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

- GRUs have been tried too

  Li, Yujia, Tarlow, Daniel, Brockschmidt, Marc, and Zemel, Richard. Gated graph sequence neural networks. ICLR, 2016.



Final Pooling

- We are performing a graph level task
- We need to pool information from all the nodes
  - to the graph context
- **Possible task: Do max pooling and concat before getting the output**
- Easy
  - Just do mean | min | max of the features of the nodes
  - Ignore the edges
  - And then you can add one or more dense layers if you want

```
readout_features = tfgnn.keras.layers.Pool(
        tfgnn.CONTEXT, "mean", node_set_name="atoms")(graph)

    # Put a linear classifier on top (not followed by dropout).
    logits = tf.keras.layers.Dense(1)(readout_features)

    # Build a Keras Model for the transformation from input_graph to logits.
    return tf.keras.Model(inputs=[input_graph], outputs=[logits])
```

The rest is the same

- Build the model
  - And the cost function
  - Use the same Adam optimize
- And then train
- And print some results
- GNNs are only doing the message passing part