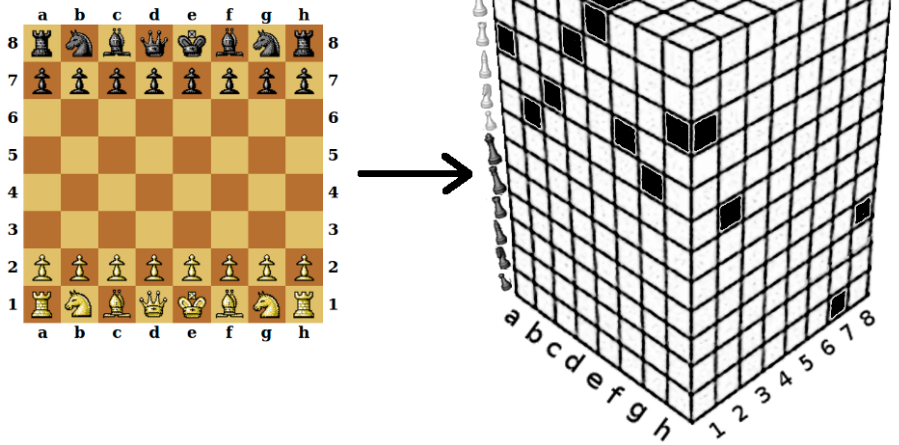
Graph Convolution Networks
(Deep Learning After You Drop The Camera)

Andrew Cullen

Convolutional Neural Networks

CNNs, as you've seen are

1. Flexible
2. Computationally Efficient
3. Expressive
4. Give great results.



The Dark Side of CNNs

But what is assumed when you use a CNN?

The data should be:

1. Regular
2. Dense (or non–sparse)
3. Have some local relational properties
4. Consistent
5. Suitable for taking layers of filtering

The Dark Side of CNNs

But what is assumed when you use a CNN?
The data should be:

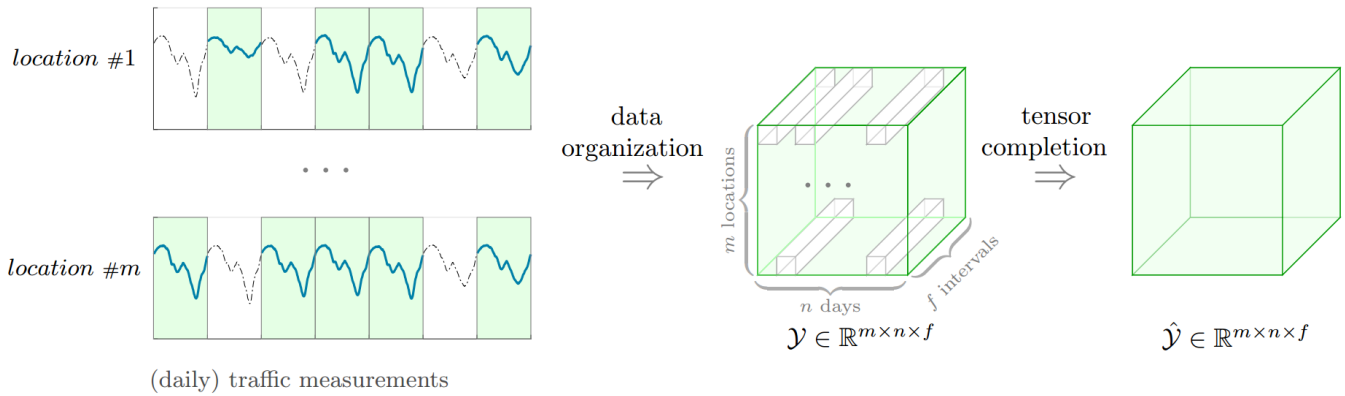
1. Regular
2. Dense (or non–sparse)
3. Have some local relational properties
4. Consistent
5. Suitable for taking layers of filtering

Breaking these assumptions?

What happens if these assumptions don't hold?

You can always try to map data into a more regular structure, or downsample your data—Uber does this for demand estimation.

Imputation is another option too, where we try and fill in missing data to make a regular structure



And even if the assumptions hold, it doesn't mean that there's not a better way.

Breaking these assumptions?

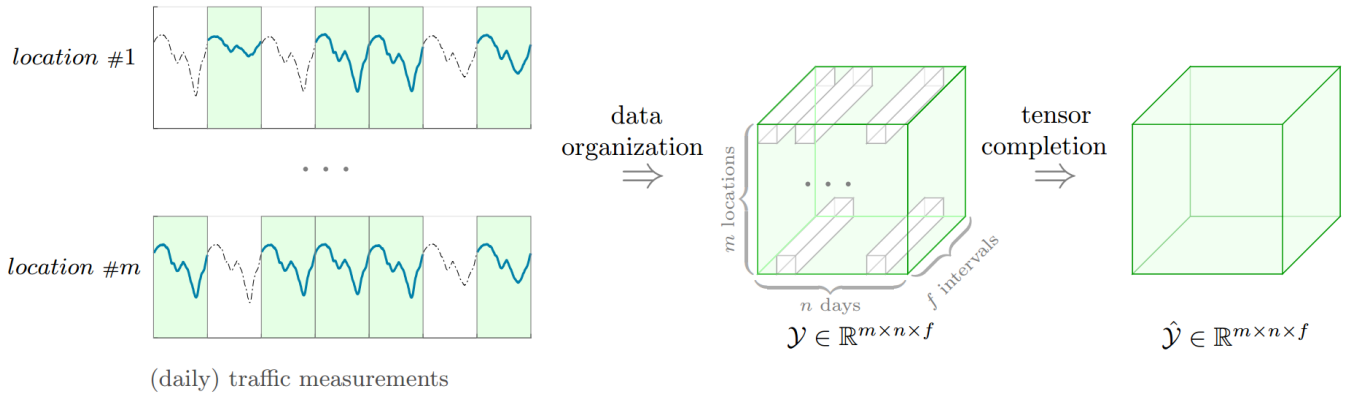
What happens if these assumptions don't hold?

You can always try to map data into a more regular structure, or downsample your data—Uber does this for demand estimation.

cnn doesn't know how to handle the empty space

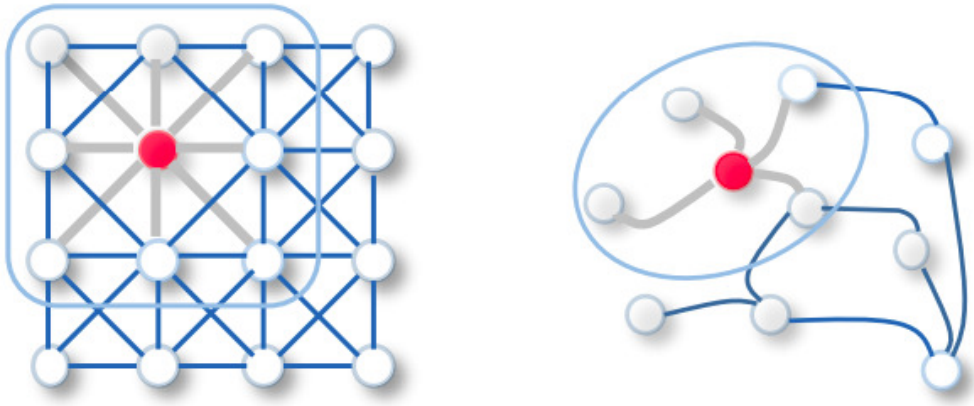
cnn is good with 'dense' data

Imputation is another option too, where we try and fill in missing data to make a regular structure



And even if the assumptions hold, it doesn't mean that there's not a better way.

Motivating Question



What happens if the data isn't nicely structured and suitable for a CNN? How do we manage real world data sets?

Real World Data

Real world data doesn't often exist on nice grids!

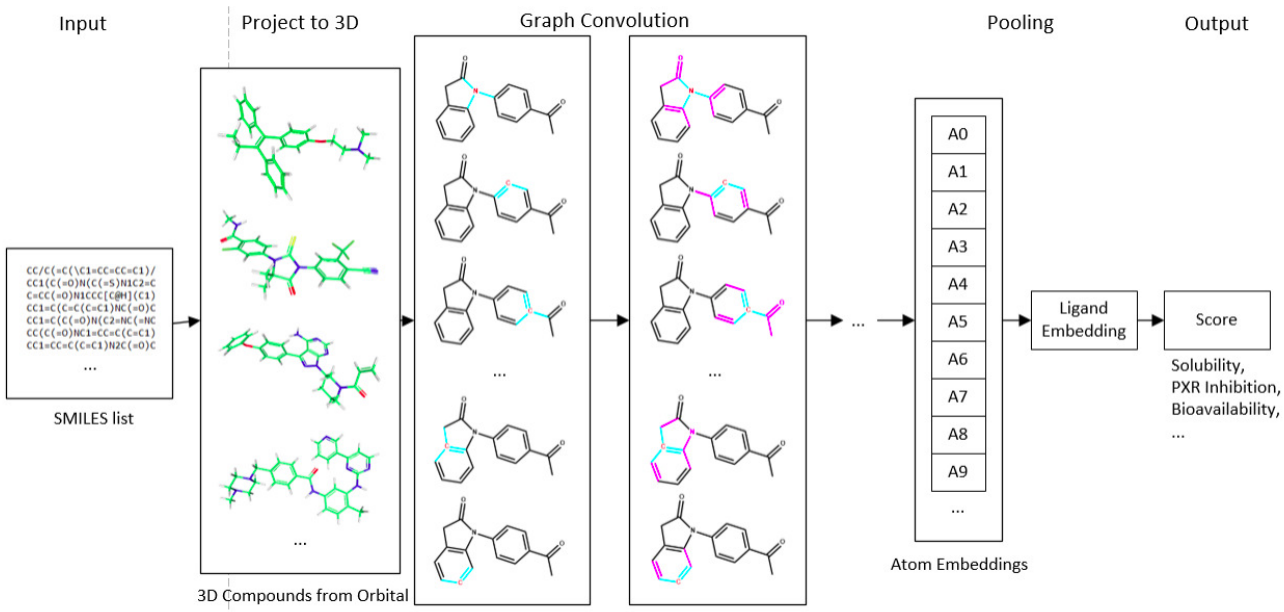
1. Traffic Graphs
2. Sensor network data
3. The relationship between your limbs as you move
4. Taxi demand
5. Social networks

We need a general language to describe the relationships between entities.

One potential solution is to use *graphs*.

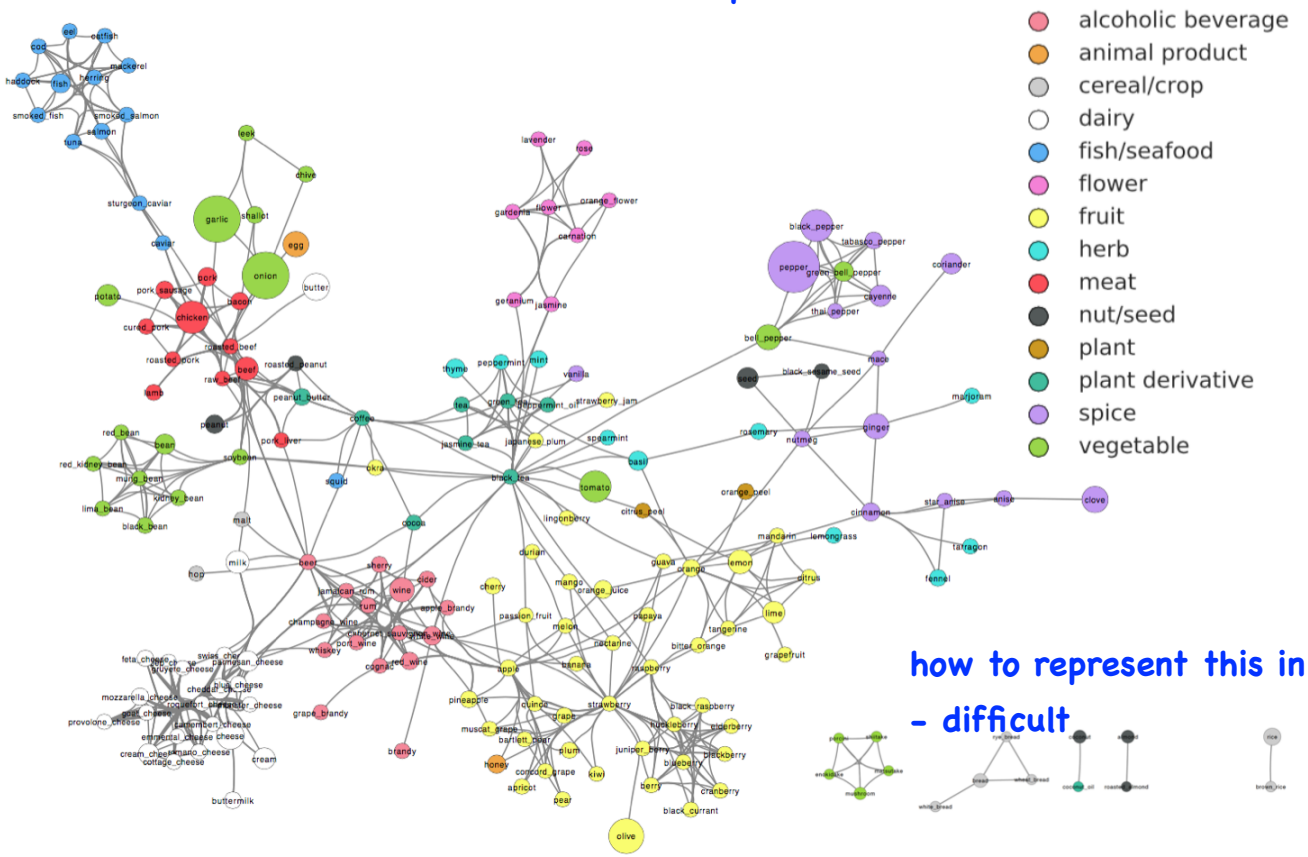
graph is the way to describe the relationship !

Real World Data



Real World Data

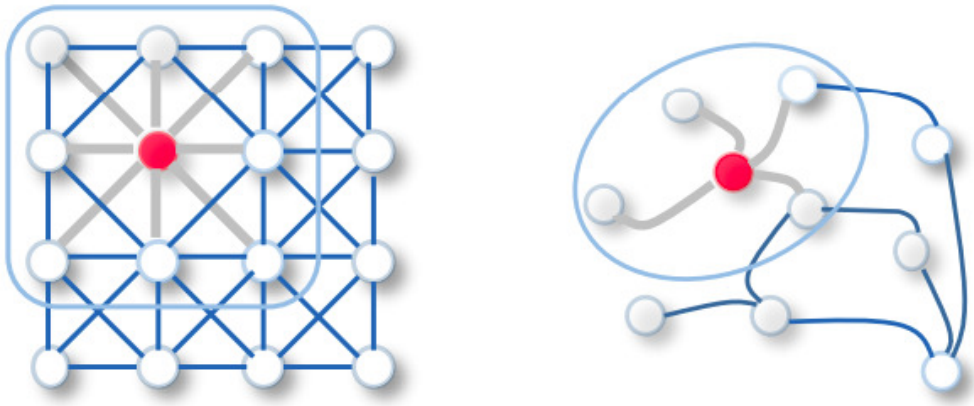
which ingredients are used with other ingredients
- a spase matrix



how to represent this in matrix?!
- difficult

It is possible to represent this as a matrix—specifically an adjacency matrix corresponding to all the links. But the matrix would inherently be incredibly sparse, which would make it inappropriate for CNNs.

Graphs



- Graphs are a collection of Vertices (or Nodes) V and Edges E. For deep learning, we presume that the Graph Nodes have attributes $\mathbf{X} \in \mathcal{R}^{n \times d}$.
- It is also possible that the edges have attributes $\mathbf{X}^e \in \mathcal{R}^{m \times c}$. These properties can be constant or vary with time.
- With a Graph Convolution Network, we want to aggregate information from the neighbours of a node.
- A neighbourhood can be more than just the nodes immediate neighbours.

Predictions on Graphs

The big question is: How do we take advantage of this relational structure to make better predictions?

Rather than abstracting them to make the data fit modelling frameworks (like CNNs), we instead want to explicitly model these relationships to improve predictive performance.

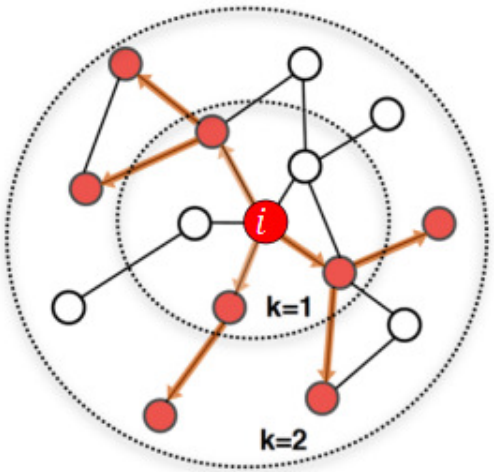
Predictions on Graphs

The big question is: How do we take advantage of this relational structure to make better predictions?

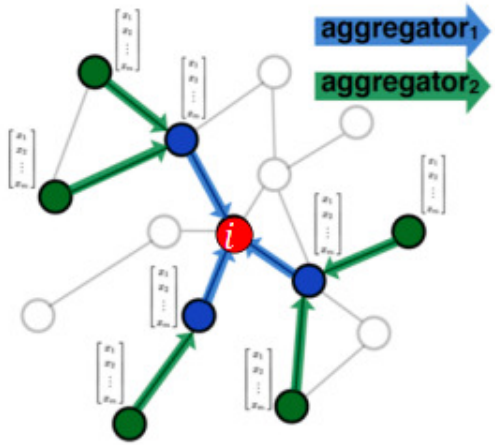
Rather than abstracting them to make the data fit modelling frameworks (like CNNs), we instead want to explicitly model these relationships to improve predictive performance.

Graph Neural Networks

With a graph neural network, we want to learn how to aggregate and propagate information across the graph, in a way that helps us extra **local** (node specific) or **global** (graph specific) features.



Determine node computation graph



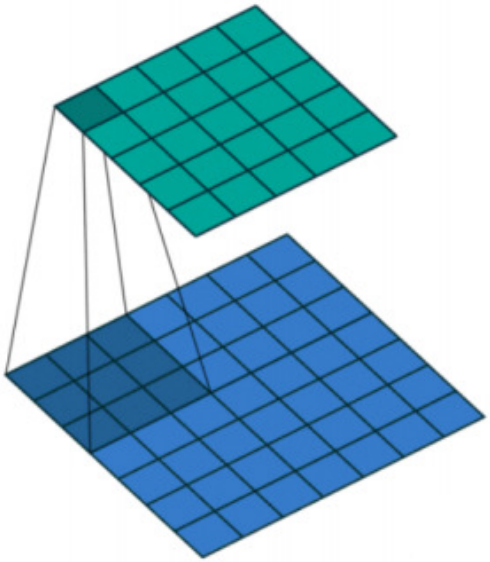
Propagate and transform information

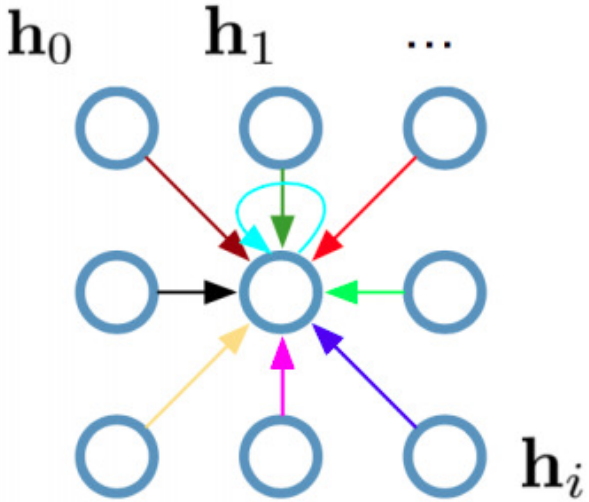
What exactly is Convolution again?

$$(f * g)(x) = \int_{\mathcal{R}^d} f(y)g(x-y)dy = \int_{\mathcal{R}^d} f(x-y)g(y)dy.$$

- In general, a convolution is the distortion of one function by another, so one takes the properties of the other.
- In a CNN, we project the data onto the convolution kernel, and extract properties about the local neighbourhood within the matrix representation.

CNN Networks



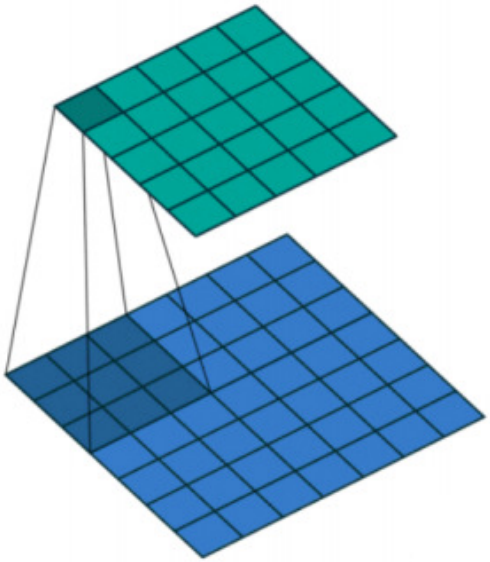


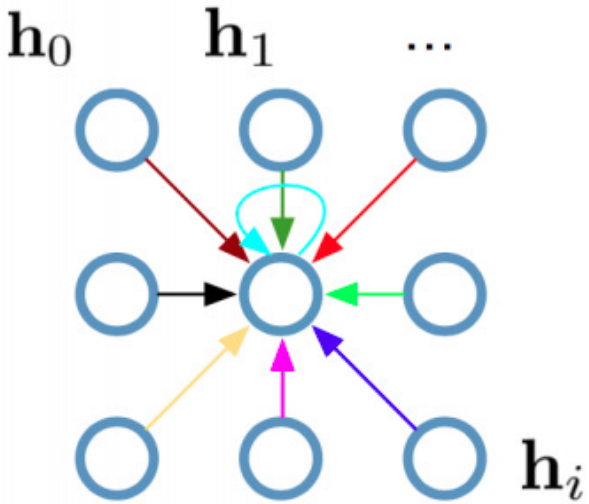
$\mathbf{h}_i \in \mathcal{R}^F$ are the hidden layer activations of each pixel.

Update the hidden layer by

$\mathbf{h}_j^{(l+1)} = \sigma\left(\sum_{\forall i} \mathbf{W}_i^{(l)} h_i^{(l)}\right)$

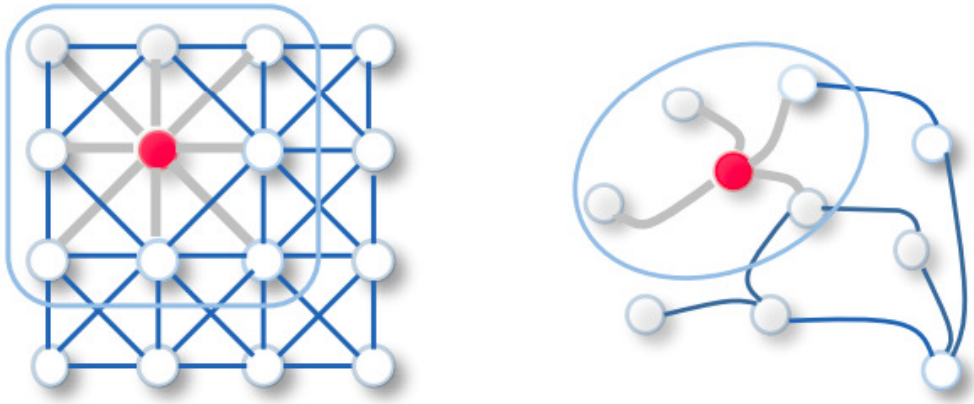
CNN Networks





$\mathbf{h}_i \in \mathcal{R}^F$ are the hidden layer activations of each pixel.
Update the hidden layer by
$\mathbf{h}_j^{(l+1)} = \sigma\left(\sum_{\forall i} \mathbf{W}_i^{(l)} h_i^{(l)}\right)$

Graph Convolutional Networks

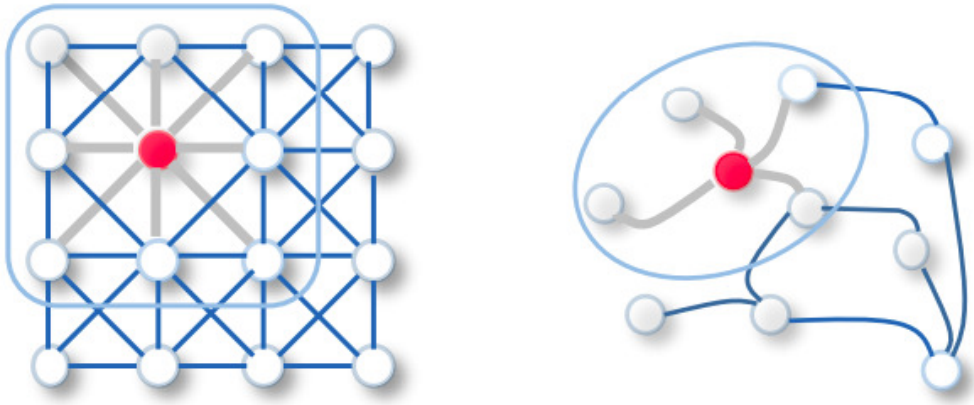


To update:

$$\mathbf{h}_0^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \mathbf{W}_2^{(l)}\mathbf{h}_2^{(l)} + \mathbf{W}_3^{(l)}\mathbf{h}_3^{(l)} + \mathbf{W}_4^{(l)}\mathbf{h}_4^{(l)}\right)$$

Note how **each node** has a **single weight** $\mathbf{W}_i$, rather than a unique weight for each link.

Graph Convolutional Networks



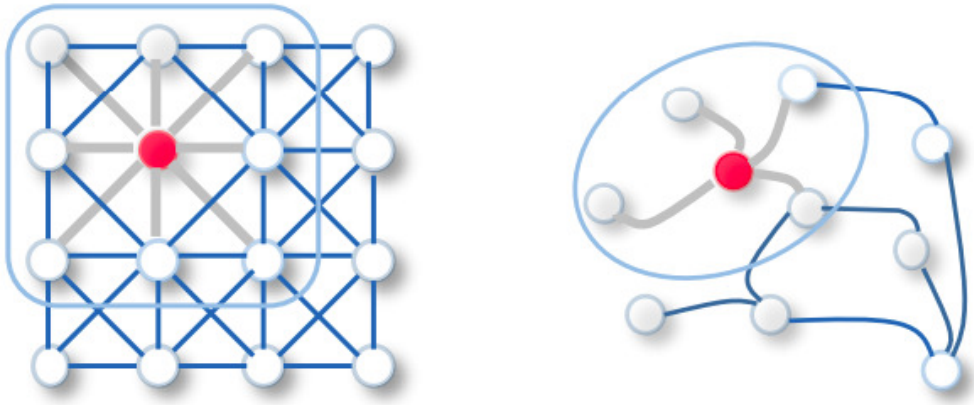
But how do we account for the fact that some nodes have fewer connections?

Weight the update, so that:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{W}_i \mathbf{h}_0^{(l)} + \sum_{\forall j \in \mathcal{N}_i} f(i, |\mathcal{N}_j|) \mathbf{h}_j^{(l)} W_j^{(l)}\right)$$

One possible weighting is $f(i, |\mathcal{N}_j|) = \frac{1}{|\mathcal{N}_i|}$

Graph Convolutional Networks



But how do we account for the fact that some nodes have fewer connections?

Weight the update, so that:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{W}_i \mathbf{h}_0^{(l)} + \sum_{\forall j \in \mathcal{N}_i} f(i, |\mathcal{N}_j|)\mathbf{h}_j^{(l)} W_j^{(l)}\right)$$

One possible weighting is $f(i, |\mathcal{N}_j|) = \frac{1}{|\mathcal{N}_i|}$

how much weighting do i want to give in the hidden states node

divide by the node connected to it - Equal Contribution from Neighbors

Graph Network Extensions

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

Can be generalised as

$$H^{(l+1)} = \sigma\left(H^{(l)}W_0^{(l)} + Agg\left(\{h_j^{(l)}, \forall j \in \mathcal{N}_i\}\right)\right)$$

Where the Agg function can be nearly anything. Examples include max–pooling

aggregation function

$$Agg = \gamma\left(\{\mathbf{Qh}\}\right)$$

where γ is a mean, max, or min function. We can also apply LSTMs to the output (or even, in some cases, on the hidden layers too)

$$Agg = \text{LSTM}(\mathbf{h})$$

Efficient Graph Updates

Just summing over all the connecting nodes is neither efficient, nor tensor–like. The update procedure can instead by framed as

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{H}^{(l)}\mathbf{W}_0^{(l)} + \tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}_1^{(l)}\right)$$

where $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{1/2}$ is the Laplacian operator.
$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)}, \mathbf{h}_1^{(l)}, \ldots, \mathbf{h}_N^{(l)}]$. In this $\mathbf{A}$ is the graph adjacency matrix, for which $\mathbf{A}_{i,j} = 1$ if there's a link from node i to j, $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A}$, where

$$\mathbf{D}_{ii} = \sum_{\forall i} \mathbf{A}_{ij}$$

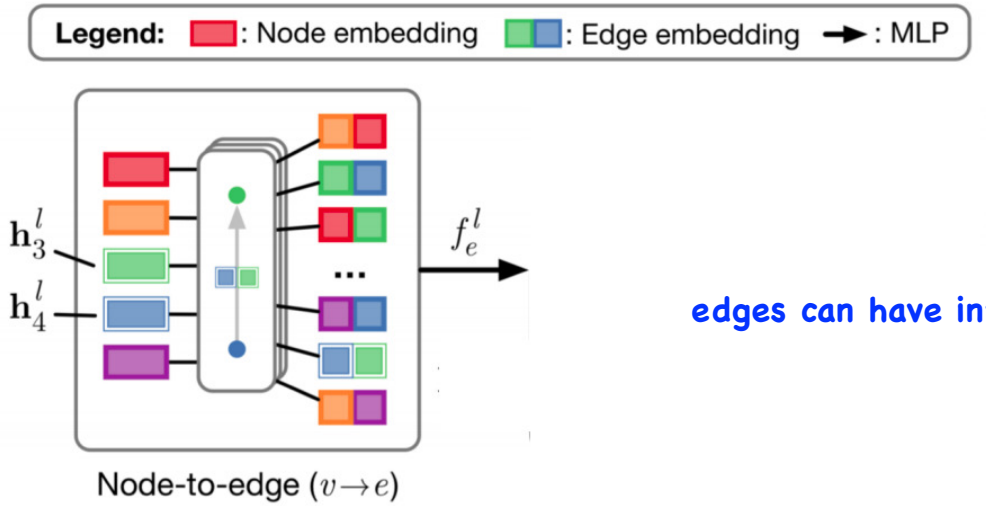
L is a modified Laplacian matrix used to incorporate both the graph's structure and self-loops for stable and effective feature propagation across nodes.

Instead of using $\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{1/2}$, can use the modified Laplacian $\mathbf{L} = \mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{1/2}$, so that

$$\mathbf{H}^{(l+1)} = \sigma\left(\mathbf{L}\mathbf{H}^{(l)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\right)$$

Graph Edge Networks

But real data of interest might exist on the nodes and the edges.

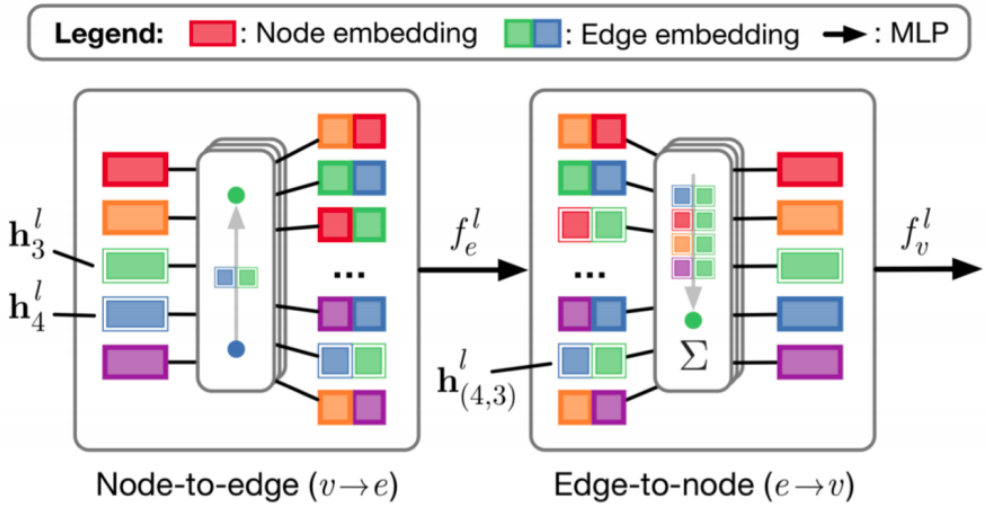


edges can have information as well!

Edge hidden weights can be assigned as

$$\mathbf{h}_{(i,j)}^{(l)} = f_e^{(l)}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}; \mathbf{x}_{(i,j)}\right)$$

Graph Edge Networks



Edge hidden weights can be assigned as

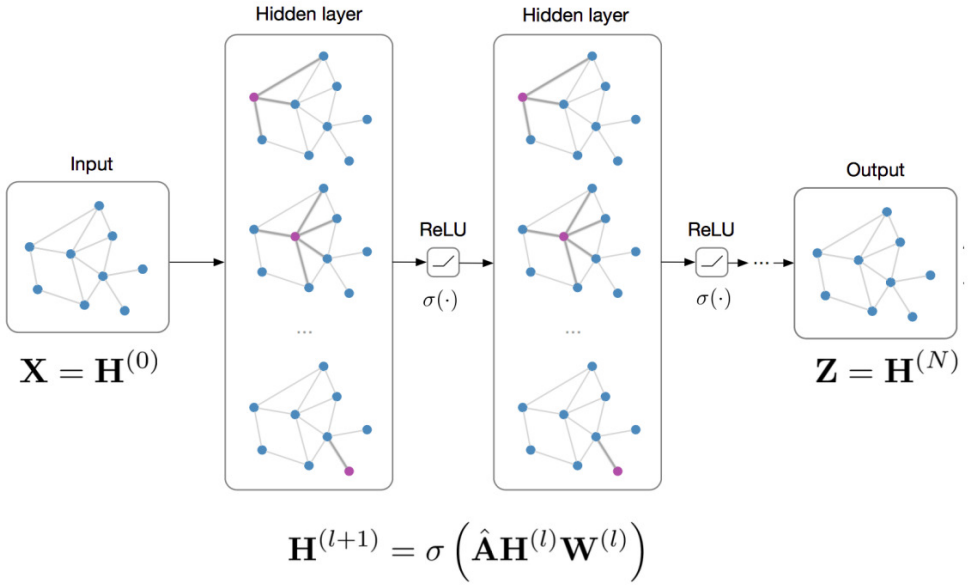
edge function first, and then the vertix function

$$\mathbf{h}_{(i,j)}^{(l)} = f_e^{(l)}\left(\mathbf{h}_i^{(l)}, \mathbf{h}_j^{(l)}; \mathbf{x}_{(i,j)}\right)$$

Vertex hidden weights are then

$$\mathbf{h}_j^{(l+1)} = f_v^{(l)}\left(\sum_{\forall i \in \mathcal{N}_j} \mathbf{h}_{(i,j)}^{(l)}; \mathbf{x}_i\right)$$

The Final Layer



the output may not necessarily be a graph, can be decoded as relationships etc.

The final layer can be processed in a number of ways. $\text{softmax}(\mathbf{z}_i)$ can be used for node classification, and $\text{softmax}(\sum \mathbf{z}_i)$ for graph classification. The importance of links can be predicted by $\sigma(\mathbf{z}_i^T \mathbf{z}_j)$. Other activation functions can be used for regression.

Summary so far

- Graph networks discard the structure of tensor/matrix blocks in exchange for flexible discretisations.
- Information can be encoded at the graph nodes and edges. These properties can even be time varying subject to local or global conditions.
- Care needs to be taken with how the networks weigh varying numbers of connections into a node.

how you define

Training

- With a CNN, all information is loaded into memory. If our data is an image, then we load the entire image at once.
- In a GCN, we can do the same thing. But a Graph $G = (E, V)$ has a lot more information to load in, and there's no implicit structure. the connection is informative -> more space
- But with a GCN, we don't need global information to train or run the model. So can randomly select a node, expand over its neighbours, and train over a subset of the graph. Changing this subset over training allows global behaviour to be learned.

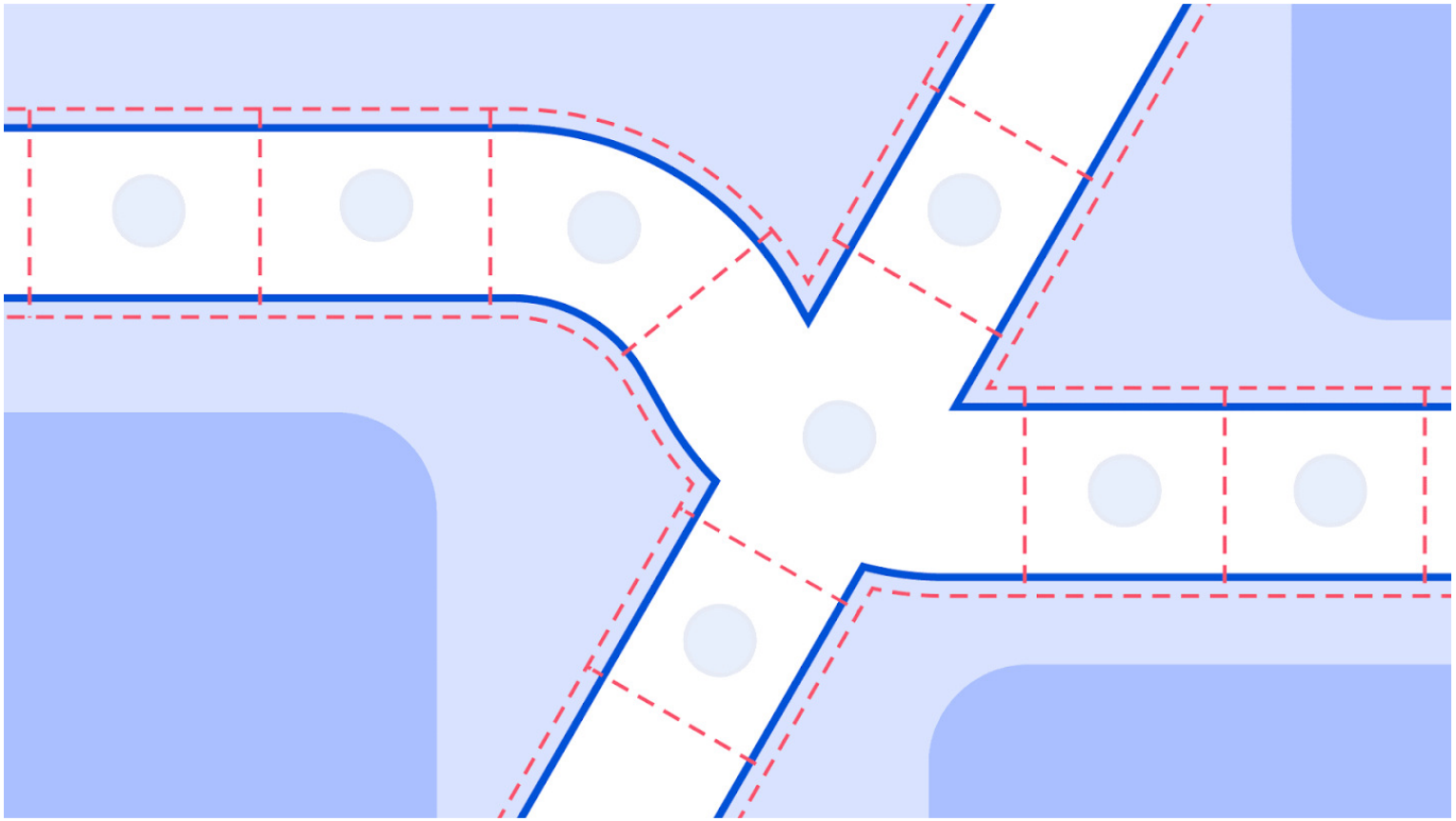
Training

GPU works well with well structured data

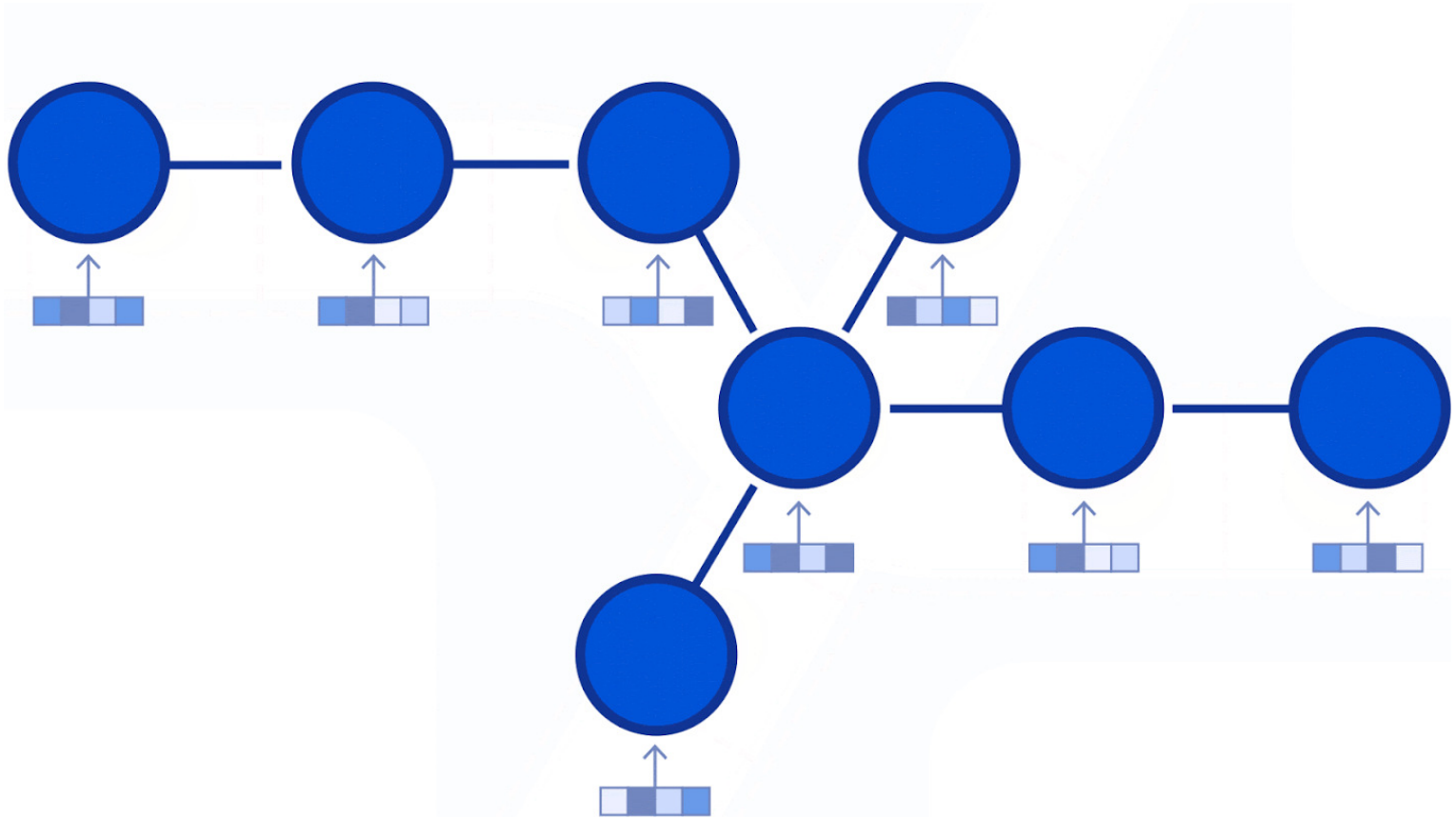
- GPUs (Graphics Processing Units) are designed to handle highly parallelizable tasks. Their origin in graphics rendering for video games required them to handle vast numbers of pixels and vertices simultaneously. As a result, their architecture is optimized for handling large arrays of data where the same operation is performed on each element in parallel.

- With a CNN, all information is loaded into memory. If our data is an image, then we load the entire image at once.
- In a GCN, we can do the same thing. But a Graph $G = (E, V)$ has a lot more information to load in, and there's no implicit structure.
- But with a GCN, we don't need global information to train or run the model. So can randomly select a node, expand over its neighbours, and train over a subset of the graph. Changing this subset over training allows global behaviour to be learned.

Case Study: Google Maps

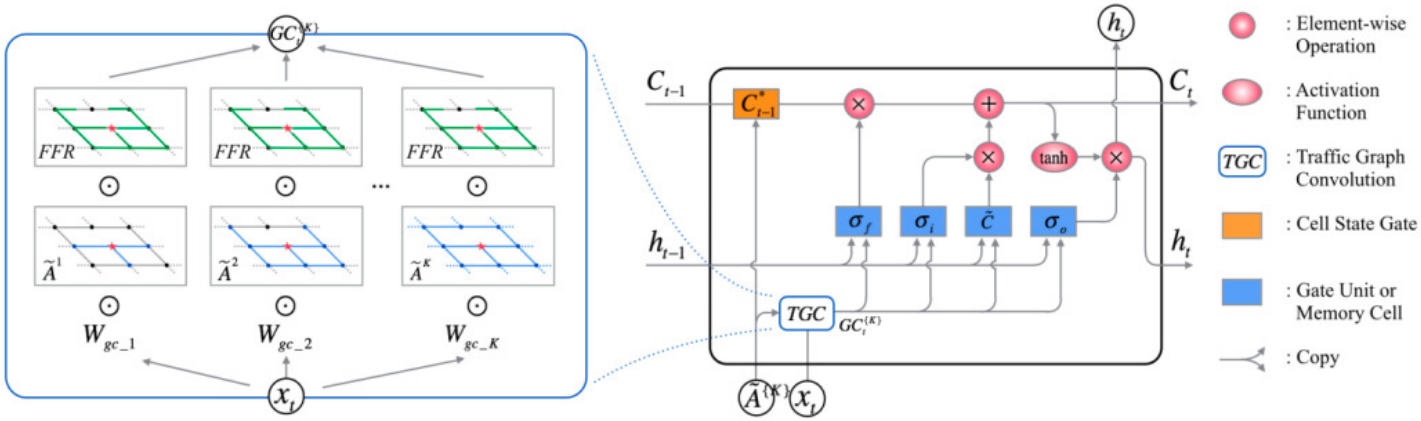


Case Study: Google Maps



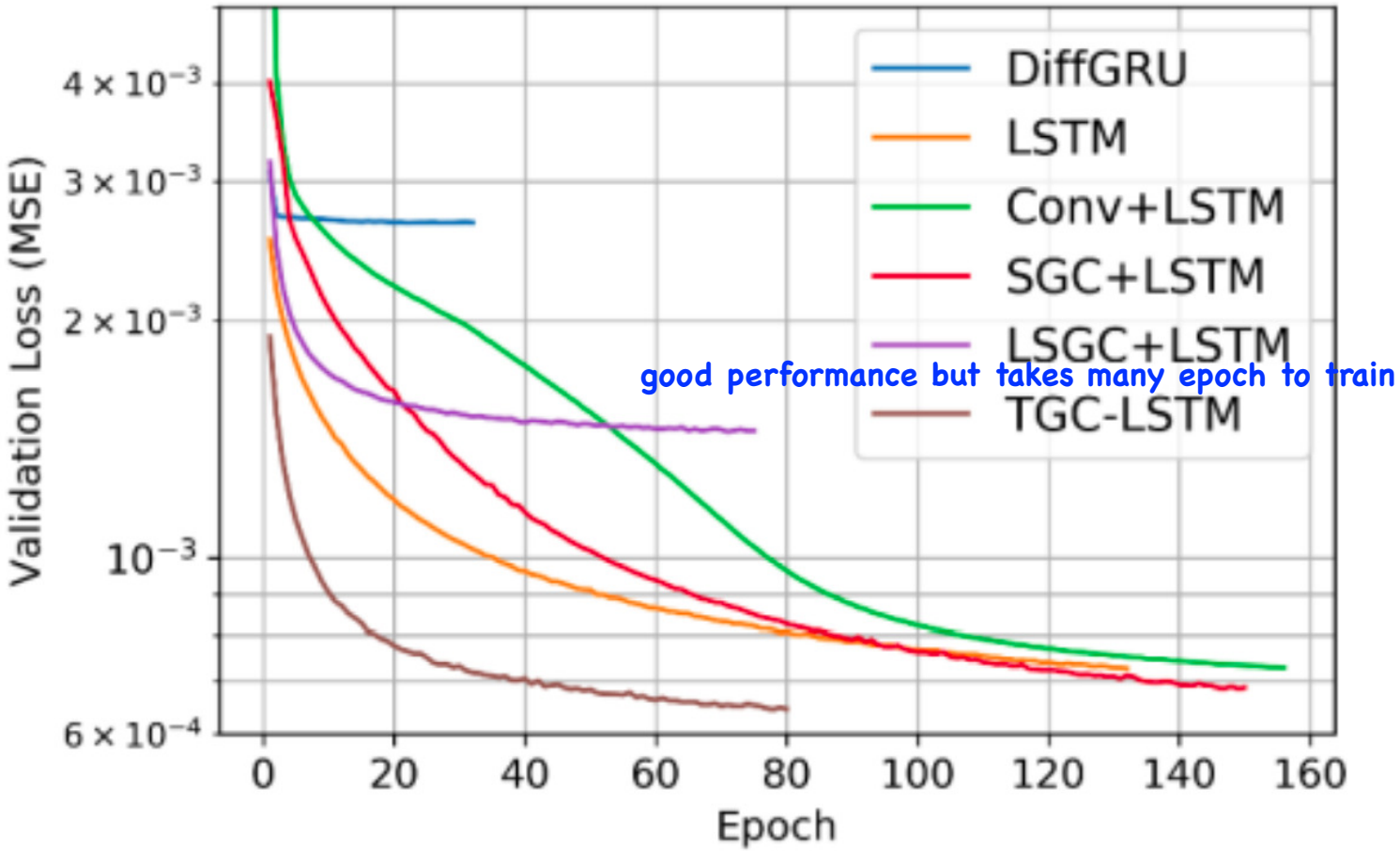
Create the graph network, and then at each node enter a sequence of time series data.

Case Study: Google Maps



Behaviour across the whole network can be described by a Graph Convolution Network, embedded within an LSTM–like structure.

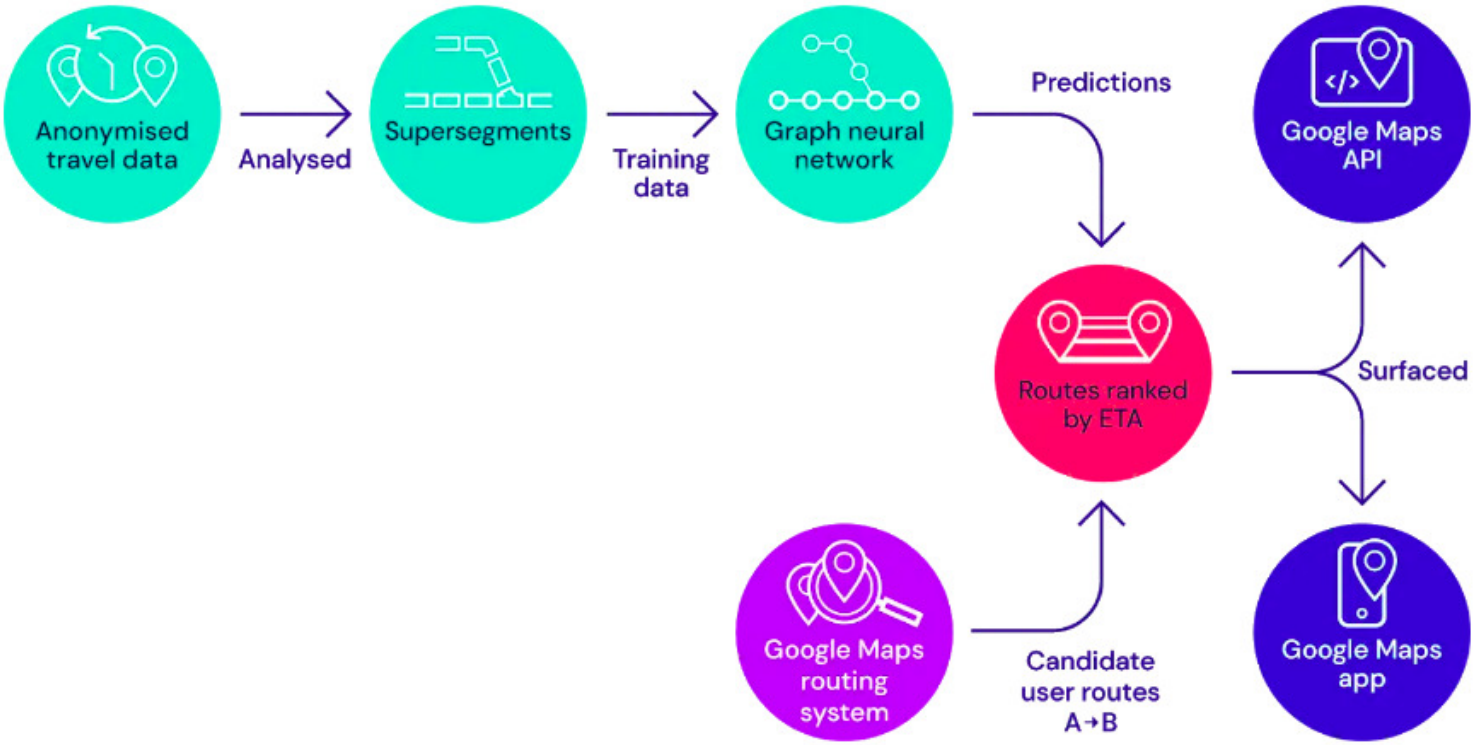
Case Study: Google Maps



good performance but takes many epoch to train

Case Study: Google Maps

what resolution is required (need many techniques to find



Case Study: Point Clouds

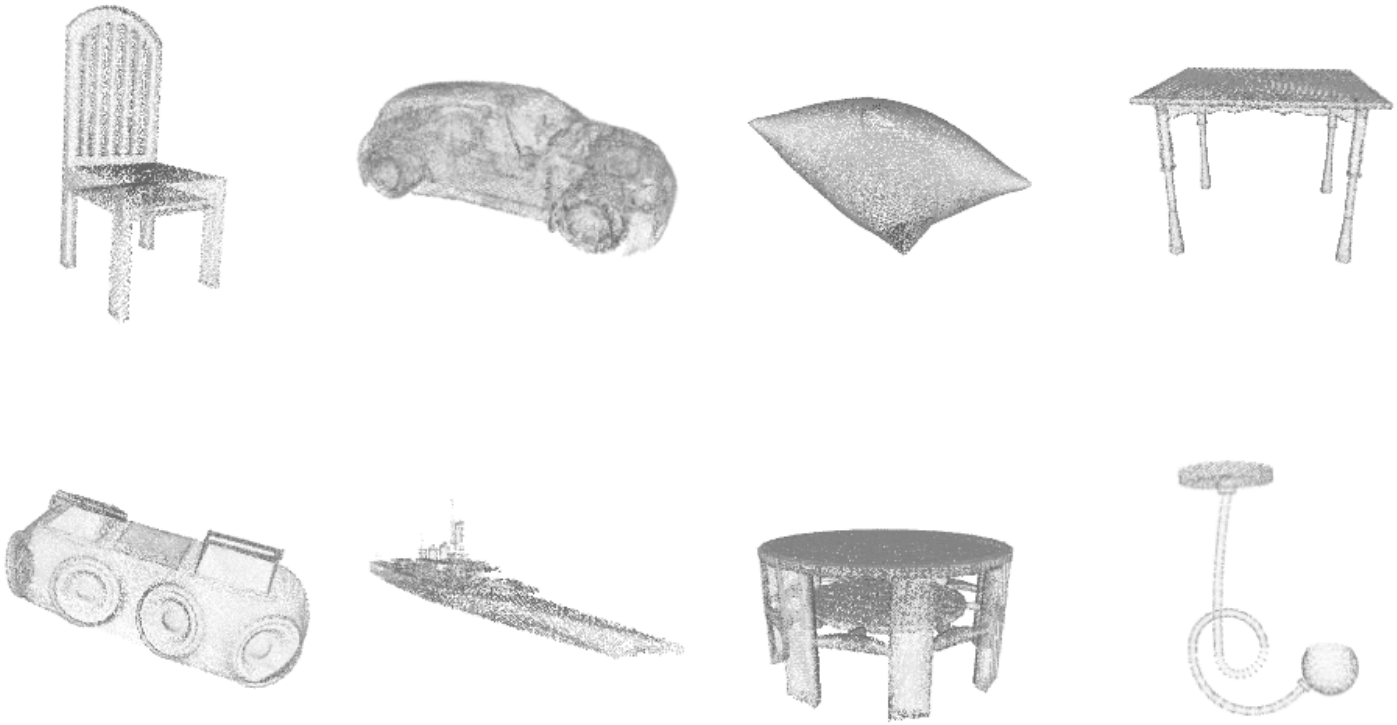


Figure: ShapeNet point clouds

These can be represented as a density matrix, but this approach may fail in the case of complex geometries, noisy data, or areas with holes. Would require significant preprocessing, as point clouds are irregular and unordered.

Case Study: Point Clouds

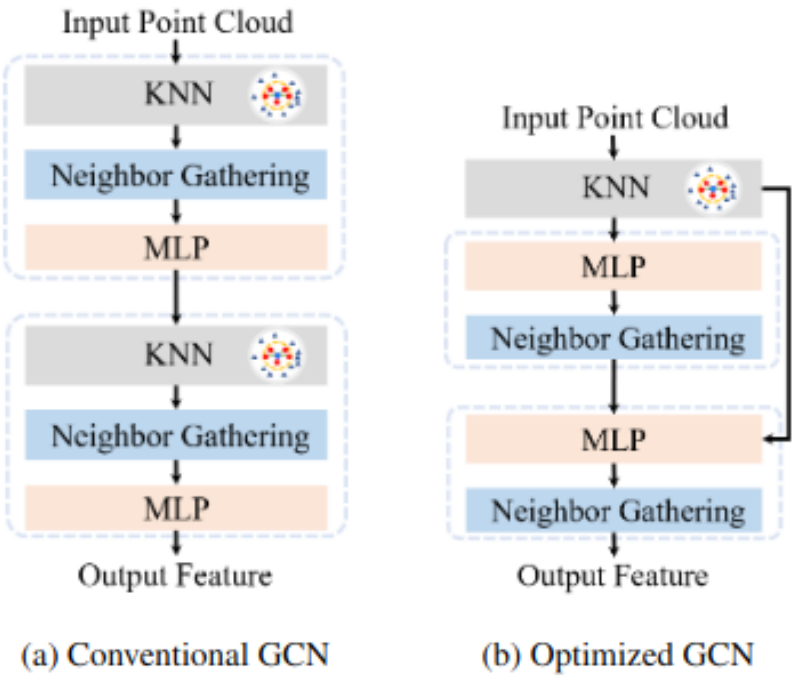


Figure: Architecture from "Towards Efficient Graph Convolutional Networks for Point Cloud Handling" Li et. al, 2021

Case Study: Point Clouds

Network	Method	Points	mIoU	Runtime [ms]	GPU mem. [GB]	FLOPs [G]	#GPU
PointCNN [21]	Baseline	2048	83.34	123.0 / 100.%	3.3 / 100.%	9.7 / 100.%	1
	Accel.	2048	83.21	111.9 / 91.0%	2.7 / 82.7%	7.6 / 78.8%	1
DGCNN [45]	Baseline	2048	84.95	116.1 / 100.%	17.2 / 100.%	158.8 / 100.%	2
	Accel.	2048	84.78	81.8 / 70.5%	4.1 / 23.8%	71.2 / 44.8%	1
[50]	Baseline	2048	84.13	365.3 / 100.%	9.7/ 100.%	202.9 / 100.%	2
	Accel.	2048	84.02	46.2 / 12.6 %	1.9 / 19.5%	52.7 / 26.0%	1

Figure: Architecture from "Towards Efficient Graph Convolutional Networks for Point Cloud Handling" Li et. al, 2021

While GNNs are less efficient than CNNs, as a relatively new and emerging architecture there's still plenty of scope for improvements.

trade off - more informative but computational expensive!

Overview

The biggest feature of Graph Neural Networks is their flexibility. A GNN can perform almost any operation that you could see with a traditional CNN, but with extra flexibility in how you design the network.
This flexibility in turn allows ML practitioners to both approach a greater range of problems, and to tackle traditional problems with the potential for greater accuracy.
With careful design, even with the additional overhead of managing the graph, they also hold the potential to be more scalable to large data sets than other NN approaches.