

Graph Convolution Networks

(Deep Learning After You Drop The Camera)

Andrew Cullen

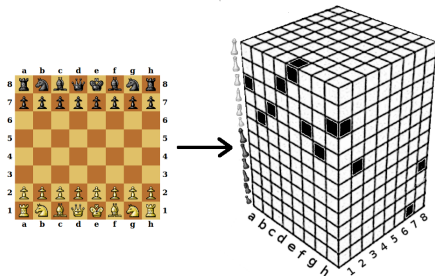
Copyright: University of Melbourne



CONVOLUTIONAL NEURAL NETWORKS

CNNs, as you've seen are

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



ASSUMPTIONS

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

ASSUMPTIONS

But what is assumed when you use a CNN?

The data should be:

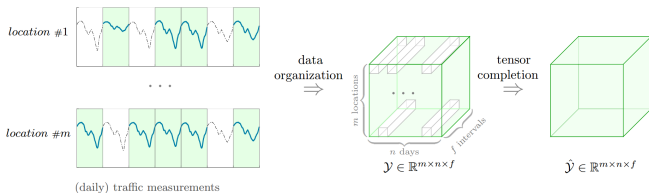
1. Regular
2. Dense (or non-sparse)
3. Have some local relational properties
4. Consistent
5. Suitable for taking layers of filtering

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



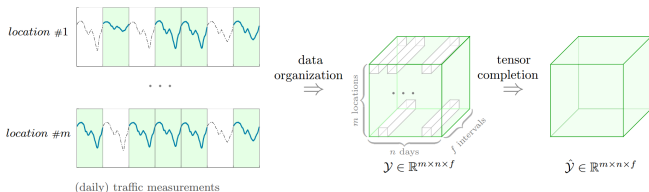
And even if the assumptions hold, it doesn't mean that there's not a better way.

BREAKING THESE ASSUMPTIONS?

What happens if these assumptions don't hold?

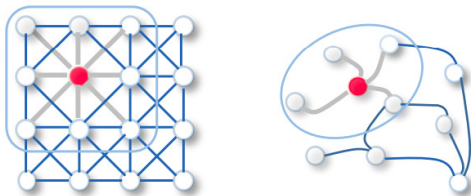
You can always try to map data into a more regular structure, or **downsample your data**—Uber does this for demand estimation.

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.

GRAPH CONVOLUTIONAL NETWORKS



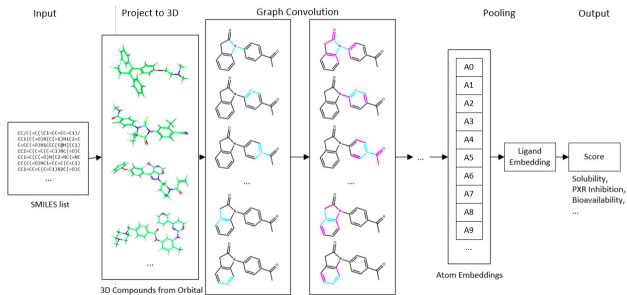
But what happens if the data isn't nicely structured, like you'd get with a CNN? How do we manage real world data sets?

REAL WORLD DATA

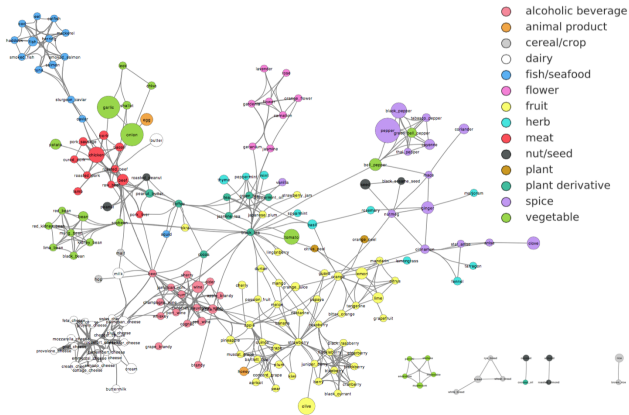
Real world data doesn't often exist on nice grids!

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

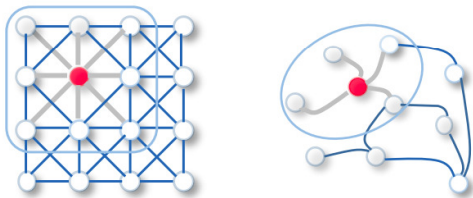
REAL WORLD DATA



REAL WORLD DATA



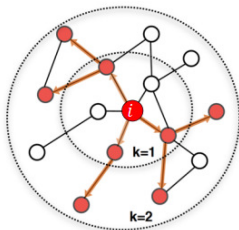
GRAPHS



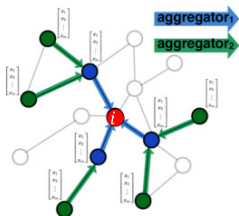
- ▶ Graphs are a collection of Vertices (or Nodes) V and Edges E . For deep learning, we presume that the Graph Nodes have attributes $\mathbf{X} \in \mathcal{R}^{n \times d}$.
- ▶ It is also possible that the edges have attributes $\mathbf{X}^e \in \mathcal{R}^{m \times c}$. These properties can be 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.

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 extract **local** (node specific) or **global** (graph specific) features.



Determine node
computation graph



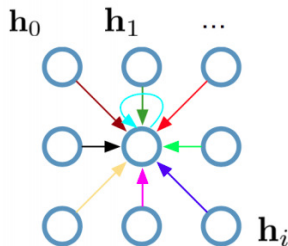
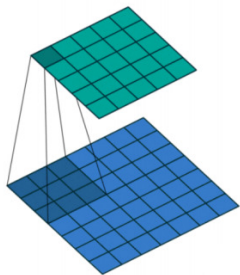
Propagate and
transform information

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 they take the properties.
- ▶ In a CNN, we project the data onto the convolution kernel, and extract properties.

CNN NETWORKS

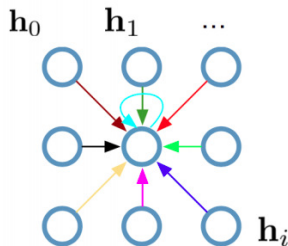
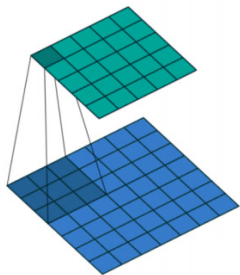


$\mathbf{h}_i \in \mathcal{R}^F$ are the hidden layer activations of each pixel.

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

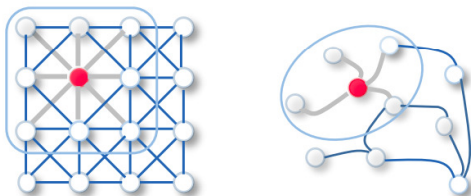


$\mathbf{h}_i \in \mathcal{R}^F$ are the hidden layer activations of each pixel.

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



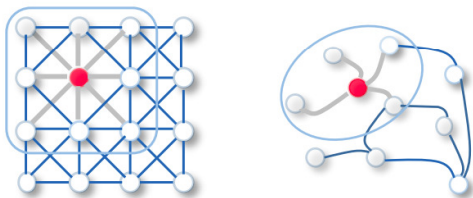
We are using the same weight for all edge that come from the same node

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



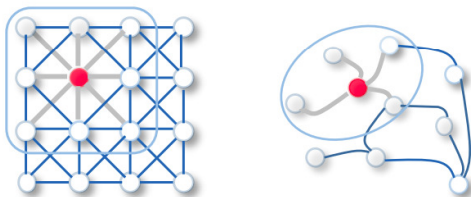
But how do we account for the fact that some nodes have fewer connections?

Weight the update, so that:

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

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:

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

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

normalise the sum to avoid imbalance

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

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

Where the *Agg* function can be. Examples include max-pooling

$$\text{Agg} = \gamma(\{\mathbf{Qh}\})$$

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)

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

EFFICIENT GRAPH UPDATES

Just summing over all the connecting nodes is neither efficient, nor tensor-like. The update procedure can instead be 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.

$\mathbf{H}^{(l)} = [\mathbf{h}_1^{(l)}, \mathbf{h}_1^{(l)}, \dots, \mathbf{h}_N^{(l)}]$. 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

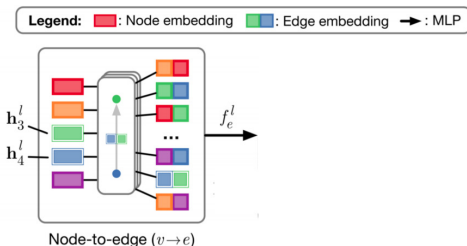
$$\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$$

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

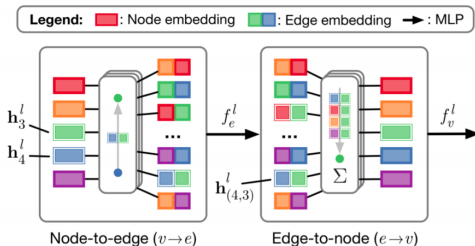
But real data of interest might exist on the nodes and the edges.



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



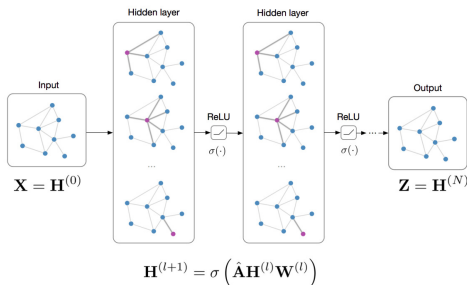
Edge hidden weights can be assigned as

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

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 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.

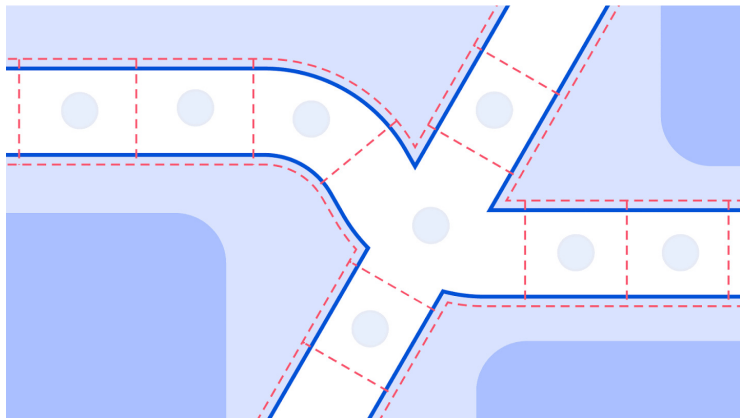
TRAINING

- ▶ 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.
- ▶ 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.

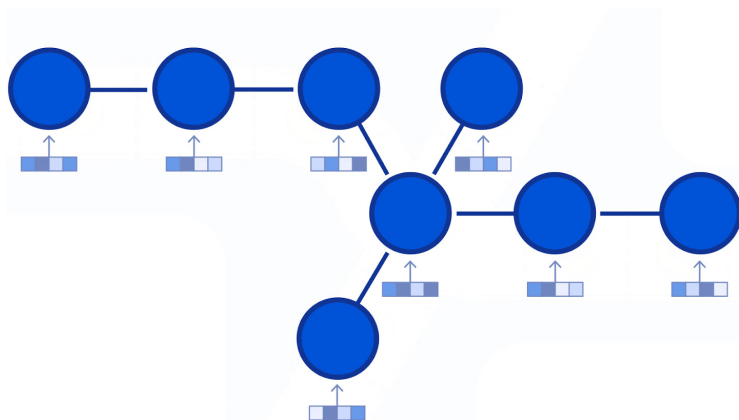
TRAINING

- ▶ 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

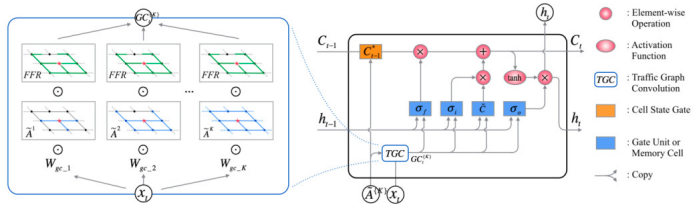


CASE STUDY: GOOGLE MAPS



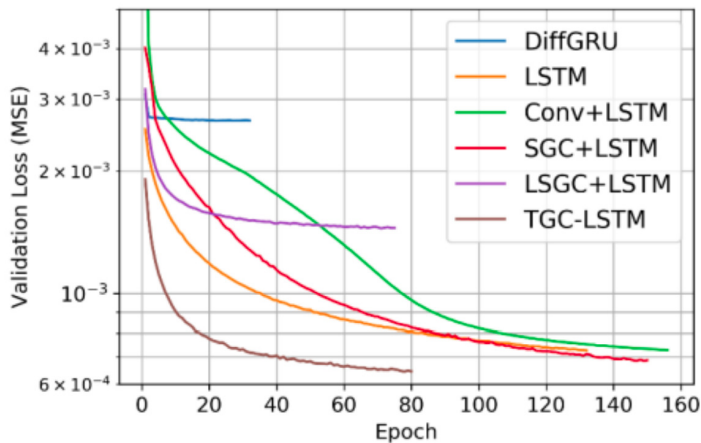
Create the graph network, and then at each node enter a sequence of time series data.

CASE STUDY: GOOGLE MAPS

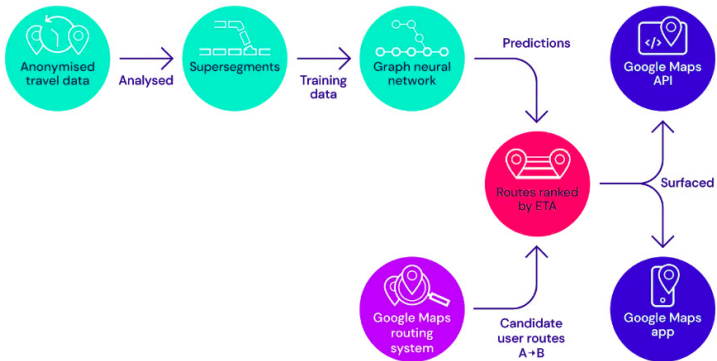


Behaviour across the whole network can be described by a Graph Convolution Network, embedded within an LSTM-like structure.

CASE STUDY: GOOGLE MAPS



CASE STUDY: GOOGLE MAPS



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.