Keeping our graphs attentive

Petar Veličković

Artificial Intelligence Group
Department of Computer Science and Technology, University of Cambridge, UK
In this talk, I will present a survey of recent developments in applying **attentive mechanisms** to improving the exploitation of nontrivial graph structure in data.

This will involve a discussion of:

- *Graph Attention Networks.*
- Subsequently released generalisations and improvements (EAGCN, GaAN, DeepInf, *Attention Solves your TSP* and hyperbolic attention networks).
- Applications to relational reasoning, multi-agent interaction and adversarial defences.
- Subsequent related work on cortical mesh segmentation and paratope prediction I’ve been involved in.
Graphs are everywhere!
Mathematical formulation

We will focus on the **node classification** problem:

- **Input**: a matrix of *node features*, \( F \in \mathbb{R}^{N \times F} \), with \( F \) features in each of the \( N \) nodes, and an *adjacency matrix*, \( A \in \mathbb{R}^{N \times N} \).

- **Output**: a matrix of *node class probabilities*, \( Y \in \mathbb{R}^{N \times C} \), such that \( Y_{ij} = \mathbb{P}(\text{Node } i \in \text{Class } j) \).

We also assume, for simplicity, that the edges are **unweighted** and **undirected**:

- That is, \( A_{ij} = A_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & \text{otherwise} \end{cases} \)

but many algorithms we will cover are capable of generalising to weighted and directed edges.

There are **two** main kinds of learning tasks in this space...
Transductive learning

Training algorithm sees *all features* (*including test nodes*)!
Inductive learning

- Now, the algorithm does not have access to all nodes upfront!

- This often implies that either:
  - Test nodes are (incrementally) inserted into training graphs;
  - Test graphs are disjoint and completely unseen!

- A much harder learning problem (requires generalising across arbitrary graph structures), and many transductive methods will be inappropriate for inductive problems!
Explicit graph neural network methodologies

- We will restrict our attention solely to methods that directly leverage the graph structure when extracting features.

- **Main idea**: Compute node representations $\vec{h}_i$ based on the initial features $\vec{f}_i$ and the graph structure, and then use $\vec{h}_i$ to classify each node independently.
The silver bullet—a convolutional layer

- It would be, in particular, highly appropriate if we could somehow generalise the convolutional operator (as used in CNNs) to operate on arbitrary graphs!

- A “common framework” for many of the approaches to be listed now has been presented in “Neural Message Passing for Quantum Chemistry”, by Gilmer et al. (ICML 2017).
Convolution on images

\[
I \ast K = \begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \times
\begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
\end{bmatrix} =
\begin{bmatrix}
1 & 4 & 3 & 4 & 1 \\
1 & 2 & 4 & 3 & 3 \\
1 & 2 & 3 & 4 & 1 \\
1 & 3 & 3 & 1 & 1 \\
3 & 3 & 1 & 1 & 0 \\
\end{bmatrix}
\]
Convolution on images

\[
I \ast K = \begin{bmatrix}
1 & 4 & 3 & 4 & 1 \\
1 & 2 & 4 & 3 & 3 \\
1 & 2 & 3 & 4 & 1 \\
1 & 3 & 3 & 1 & 1 \\
3 & 3 & 1 & 1 & 0
\end{bmatrix}
\]
Convolution on images

\[ I \ast K = \]

\[ \begin{array}{cccc}
1 & 4 & 3 & 4 & 1 \\
1 & 2 & 4 & 3 & 3 \\
1 & 2 & 3 & 4 & 1 \\
1 & 3 & 3 & 1 & 1 \\
3 & 3 & 1 & 1 & 0 \\
\end{array} \]
Convolution on images

\[
\begin{align*}
I &= \begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \\
K &= \begin{bmatrix}
1 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 1
\end{bmatrix} \\
I \ast K &= \begin{bmatrix}
1 & 4 & 3 & 4 & 1 \\
1 & 2 & 4 & 3 & 3 \\
1 & 2 & 3 & 4 & 1 \\
1 & 3 & 3 & 1 & 1 \\
3 & 3 & 1 & 1 & 0
\end{bmatrix}
\end{align*}
\]
Graph convolutional network?

In a nutshell, obtain higher-level representations of a node $i$ by leveraging its neighbourhood $N_i$:

$$\vec{h}_{i}^{\ell+1} = g_\ell(\vec{h}_{a}, \vec{h}_{b}, \vec{h}_{c}, ... | a, b, c, ... \in N_i)$$

where $g_\ell$ is the $\ell$-th graph convolutional layer.
Graph convolutional network?

In a nutshell, obtain higher-level representations of a node \( i \) by leveraging its neighbourhood, \( \mathcal{N}_i \):

\[
\vec{h}_i^{\ell+1} = g^\ell(\vec{h}_a^\ell, \vec{h}_b^\ell, \vec{h}_c^\ell, \ldots) \quad (a, b, c, \ldots \in \mathcal{N}_i)
\]

where \( g^\ell \) is the \( \ell \)-th graph convolutional layer.
Challenges with graph convolutions

- Desirable properties for a graph convolutional layer:
  - **Computational and storage efficiency** \( \sim O(V + E) \);
  - **Fixed** number of parameters (independent of input size);
  - **Localisation** (acts on a *local neighbourhood* of a node);
  - Specifying **different importances** to different neighbours;
  - Applicability to **inductive problems**.

- Fortunately, images have a highly rigid and regular connectivity pattern (each pixel “connected” to its eight neighbouring pixels), making such an operator trivial to deploy (as a small kernel matrix which is slided across).

- Arbitrary graphs are a **much harder** challenge!
A large class of popular approaches attempts to define a convolutional operation by operating on the graph in the **spectral domain**, leveraging the *convolution theorem*.

These approaches utilise the **graph Laplacian matrix**, \( L \), defined as \( L = D - A \), where \( D \) is the degree matrix (diagonal matrix with \( D_{ii} = \text{deg}(i) \)) and \( A \) is the adjacency matrix.

Alternately, we may use the **normalised graph Laplacian**, \( \tilde{L} = I - D^{-1/2} A D^{-1/2} \).
Graph Laplacian example

\[ L = \begin{bmatrix}
2 & -1 & 0 & 0 & -1 & 0 \\
-1 & 3 & -1 & 0 & -1 & 0 \\
0 & -1 & 2 & -1 & 0 & 0 \\
0 & 0 & -1 & 3 & -1 & -1 \\
-1 & -1 & 0 & -1 & 3 & 0 \\
0 & 0 & 0 & -1 & 0 & 1 \\
\end{bmatrix} \]
The Laplacian is symmetric and positive semi-definite; we can therefore diagonalise it as \( L = U \Lambda U^T \), where \( \Lambda \) is a diagonal matrix of its eigenvalues.

This means that multiplying the feature matrix by \( U^T \) allows us to enter the \textit{spectral domain} for the graph! Therein, convolution just amounts to pointwise multiplication.

This “Graph Fourier Transform” is the essence of the work of Bruna \textit{et al.} (ICLR 2014).
To convolve two signals using the convolution theorem:

$$\text{conv}(\vec{x}, \vec{y}) = U \left( U^T \vec{x} \odot U^T \vec{y} \right)$$

Therefore, a learnable convolutional layer amounts to:

$$\vec{h}'_i = U \left( \vec{w} \odot U^T W \vec{h}_i \right)$$

where $\vec{w}$ is a learnable vector of weights, and $W \in \mathbb{R}^{F' \times F}$ is a shared, learnable, feature transformation.

Downsides:

- Computing $U$ is $O(V^3)$—infeasible for large graphs!
- One independent weight per node—not fixed!
- Not localised!
Chebyshev networks

- These issues have been overcome by ChebyNets, the work of Defferrard et al. (NIPS 2016).
- Rather than computing the Fourier transform, use the related family of Chebyshev polynomials of order $k$, $T_k$:

$$\tilde{h}'_i = \sum_{k=0}^{K} w_k T_k(L) \mathbf{W} \tilde{h}_i$$

- These polynomials have a recursive definition, highly simplifying the computation:

$$T_0(x) = 1 \quad T_1(x) = x \quad T_k(x) = 2x T_{k-1}(x) - T_{k-2}(x)$$
Properties of Chebyshev networks

- Owing to its recursive definition, we can compute the output iteratively as \( \sum_{k=0}^{K} w_k \vec{t}_k \), where:

\[
\begin{align*}
\vec{t}_0 &= \mathbf{W} \vec{h}_i \\
\vec{t}_1 &= \mathbf{LW} \vec{h}_i \\
\vec{t}_k &= 2\mathbf{L} \vec{t}_{k-1} - \vec{t}_{k-2}
\end{align*}
\]

where each step constitutes a \textbf{sparse} multiplication with \( \mathbf{L} \).

- The number of parameters is \textbf{fixed} (equal to \( K \) weights).

- Note that \( T_k(\mathbf{L}) \) will be a (weighted) sum of all powers of \( \mathbf{L} \) up to \( \mathbf{L}^k \). This means that \( T_k(\mathbf{L})_{ij} = 0 \) if \( \text{dist}(i,j) > k \! \)!

\[ \Rightarrow \] The operator is \textbf{K-localised}!
To avoid issues with exploding or vanishing signals, typically a scaled version of $L$ is fed into the algorithm:

$$\tilde{L} = \frac{2L}{\lambda_{\text{max}}} - I$$

where $\lambda_{\text{max}}$ is the largest eigenvalue of $L$.

This constrains all eigenvalues to lie in the range $[-1, 1]$, therefore making the norm of all results controllable.

Major limitation: unable to specify different weights to different nodes in a neighbourhood! All $k$-hop neighbours will receive weight $w_k + w_{k+1} + \cdots + w_K$. 
Limited filters

Going back to the image scenario, under the assumption that each pixel of an image is connected to its immediate four neighbours, this would constrain our $3 \times 3$ convolutional kernel to be of the form:

$$
\begin{bmatrix}
  w_2 & w_1 + w_2 & w_2 \\
  w_1 + w_2 & w_0 + w_1 + w_2 & w_1 + w_2 \\
  w_2 & w_1 + w_2 & w_2 
\end{bmatrix}
$$

**severely limiting** the variety of patterns that can be usefully extracted from the image.
The **Graph Convolutional Network** (GCN) of Kipf & Welling (ICLR 2017) further fine-tunes the Chebyshev framework.

Setting $K = 1$ and assuming $\lambda_{\text{max}} \approx 2$ allows for redefining a single convolutional layer as simply:

$$\vec{h}'_i = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} W \vec{h}_i$$

which significantly improves computational performance on larger graphs and predictive power on small training sets.

However, the previous issue is *still there*...
Another fundamental constraint of all spectral-based methods is that the learnt filter weights are assuming a particular, fixed, graph Laplacian.

This makes them theoretically inadequate for arbitrary inductive problems!

We have to move on to non-spectral approaches...
An early notable approach towards such methods is the work of Duvenaud et al. (NIPS 2015).

Here, the method adapts to processing with various degrees by learning a separate weight matrix $H_d$ for each node degree $d$.

The authors dealt with an extremely specific domain problem (molecular fingerprinting), where node degrees could never exceed five; this does not scale to graphs with very wide degree distributions.
Conversely, the recently-published GraphSAGE model by Hamilton et al. (NIPS 2017) aims to restrict every degree to be the same (by sampling a fixed-size set of neighbours of every node, during both training and inference).

Inherently drops relevant data—limiting the set of neighbours visible to the algorithm.

Impressive performance was achieved across a variety of inductive graph problems. However, the best results were often achieved with an LSTM-based aggregator, which is unlikely to be optimal.
A recent development in attentional mechanisms concerns **self-attention**; a scenario where the input *attends over itself*:

\[
\alpha_{ij} = a(\vec{h}_i, \vec{h}_j)
\]

\[
\vec{h}'_i = \sum_j \text{softmax}_j(\alpha_{ij}) \vec{h}_j
\]

where \(a(\vec{x}, \vec{y})\) is a neural network (the *attention mechanism*).

- Critically, this is **parallelisable** across all input positions!
- *Vaswani et al.* (NIPS 2017) have successfully demonstrated that this operation is self-sufficient for achieving state-of-the-art on machine translation.
Graph Attention Networks

- My ICLR 2018 publication, proposing **Graph Attention Networks** (GATs), leverages exactly the self-attention operator!

- In its naïve form, the operator would compute attention coefficients *over all pairs of nodes*.

- To inject the graph structure into the model, we *restrict* the model to only attend over a node’s neighbourhood when computing its coefficient!
GAT equations

- To recap, a single attention head of a GAT model performs the following computation:

\[
e_{ij} = a(\tilde{h}_i, \tilde{h}_j)
\]

\[
\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}
\]

\[
\tilde{h}_i' = \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij} W \tilde{h}_j \right)
\]

- Some further optimisations (like multi-head attention and dropout on the \(\alpha_{ij}\) values) help further stabilise and regularise the model.
A single GAT step, visualised
GAT analysis

- **Computationally efficient**: attention computation can be parallelised across all edges of the graph, and aggregation across all nodes!

- **Storage efficient**—a sparse version does not require storing more than $O(V + E)$ entries anywhere;

- **Fixed** number of parameters (dependent only on the desirable feature count, not on the node count);

- Trivially **localised** (as we aggregate only over neighbourhoods);

- Allows for (implicitly) specifying **different importances** to **different neighbours**.

- Readily applicable to **inductive problems** (as it is a shared edge-wise mechanism)!
GAT performance

- It seems that we have finally satisfied all of the major requirements for our convolution!

- How well does it perform?
Datasets under study

Summary of the datasets used in our experiments.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Transductive</th>
<th>Inductive</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Citeseer</td>
<td>Pubmed</td>
</tr>
<tr>
<td># Nodes</td>
<td>2708</td>
<td>3327</td>
<td>19717</td>
</tr>
<tr>
<td># Edges</td>
<td>5429</td>
<td>4732</td>
<td>44338</td>
</tr>
<tr>
<td># Features/Node</td>
<td>1433</td>
<td>3703</td>
<td>500</td>
</tr>
<tr>
<td># Classes</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td># Training Nodes</td>
<td>140</td>
<td>120</td>
<td>60</td>
</tr>
<tr>
<td># Validation Nodes</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td># Test Nodes</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>
## Results on Cora/Citeseer/Pubmed

### Transductive

<table>
<thead>
<tr>
<th>Method</th>
<th>Cora</th>
<th>Citeseer</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>55.1%</td>
<td>46.5%</td>
<td>71.4%</td>
</tr>
<tr>
<td>ManiReg</td>
<td>59.5%</td>
<td>60.1%</td>
<td>70.7%</td>
</tr>
<tr>
<td>SemiEmb</td>
<td>59.0%</td>
<td>59.6%</td>
<td>71.7%</td>
</tr>
<tr>
<td>LP</td>
<td>68.0%</td>
<td>45.3%</td>
<td>63.0%</td>
</tr>
<tr>
<td>DeepWalk</td>
<td>67.2%</td>
<td>43.2%</td>
<td>65.3%</td>
</tr>
<tr>
<td>ICA</td>
<td>75.1%</td>
<td>69.1%</td>
<td>73.9%</td>
</tr>
<tr>
<td>Planetoid</td>
<td>75.7%</td>
<td>64.7%</td>
<td>77.2%</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>81.2%</td>
<td>69.8%</td>
<td>74.4%</td>
</tr>
<tr>
<td>GCN</td>
<td>81.5%</td>
<td>70.3%</td>
<td><strong>79.0%</strong></td>
</tr>
<tr>
<td>MoNet</td>
<td>81.7 ± 0.5%</td>
<td>—</td>
<td>78.8 ± 0.3%</td>
</tr>
</tbody>
</table>

| GCN-64*  | 81.4 ± 0.5% | 70.9 ± 0.5% | **79.0 ± 0.3%** |
| GAT (ours) | **83.0 ± 0.7%** | **72.5 ± 0.7%** | **79.0 ± 0.3%** |
### Results on PPI

**Inductive**

<table>
<thead>
<tr>
<th>Method</th>
<th>PPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.396</td>
</tr>
<tr>
<td>MLP</td>
<td>0.422</td>
</tr>
<tr>
<td>GraphSAGE-GCN</td>
<td>0.500</td>
</tr>
<tr>
<td>GraphSAGE-mean</td>
<td>0.598</td>
</tr>
<tr>
<td>GraphSAGE-LSTM</td>
<td>0.612</td>
</tr>
<tr>
<td>GraphSAGE-pool</td>
<td>0.600</td>
</tr>
<tr>
<td><strong>GraphSAGE</strong>(^\ast)</td>
<td>0.768</td>
</tr>
<tr>
<td>Const-GAT (ours)</td>
<td>0.934 ± 0.006</td>
</tr>
<tr>
<td><strong>GAT</strong> (ours)</td>
<td><strong>0.973 ± 0.002</strong></td>
</tr>
</tbody>
</table>

Here, *Const-GAT* is a GCN-like inductive model.
t-SNE + attention coefficients on Cora
Incorporating edge context

- The attentional setup of GAT treats each edge equally.
- This will not be appropriate for inputs such as chemical compounds, wherein the same atom can possess identical neighbourhoods but with different bonds!
EAGCN (Shang et al., 2018)

- The edge context was incorporated for the first time in the *edge attention-based multi-relational GCN* (EAGCN) model.

- Assume that there are $K$ different *edge attributes* (e.g. atom pair type, bond order . . . ) and that the $i$-th attribute has $d_i$ possible values.

- A separate attention coefficient $\alpha_{ij}$ is learned for every value of every attribute ($i \in \{1, \ldots, K\}, j \in \{1, \ldots d_i\}$), as a simple scalar embedding.
EAGCN attention mechanism

- These embeddings then form the (unnormalised) attention coefficient matrices $A^i$ for each edge attribute $i$:

$$A^i_{st} = \begin{cases} 
\alpha_{ij} & s \rightarrow t \text{ of type } j \text{ in attr. } i \\
-\infty & s \not\rightarrow t 
\end{cases}$$

which are then softmax-normalised:

$$\tilde{A}^i_{st} = \frac{\exp(A^i_{st})}{\sum_k \exp(A^i_{kt})}$$

- We can then use each of these as a separate attention head, and e.g. concatenate their outputs (for node features $H$):

$$H' = \sigma \left( \sum_{i=1}^{K} \tilde{A}^i HW \right)$$
EAGCN in action: computing $A^i$
EAGCN in action: single layer

Evaluated on molecular property classification and regression, outperforming several standard graph-based baselines.
GaAN (Zhang et al., 2018)

- The multi-head attention of GAT treats each attention head *equally*. However, not all heads necessarily convey equally important or meaningful feature spaces.

- The *Gated Attention Network (GaAN)* architecture introduces a *gating mechanism* on top of a key-value attention (as in Vaswani et al.), to control the impact of each output of each attention head.

- Evaluated on inductive node classification (Reddit/PPI) and traffic speed forecasting (METR-LA), outperforming many challenging baselines.
GaAN dataflow

- Assume we have node features $\boldsymbol{h}_i$ and node reference vectors $\boldsymbol{z}_j$ (useful to decouple when working on temporal graphs).

- First, derive queries, keys and values for the attention:

$$
\tilde{q}_i = \mathbf{W}_q \boldsymbol{h}_i \quad \tilde{k}_i = \mathbf{W}_k \boldsymbol{z}_i \quad \tilde{v}_i = \mathbf{W}_v \boldsymbol{z}_i
$$

- Now, use the queries and keys to derive coefficients:

$$
\alpha_{ij} = \frac{\exp \left( \langle \tilde{q}_i, \tilde{k}_j \rangle \right)}{\sum_{m \in \mathcal{N}_i} \exp \left( \langle \tilde{q}_i, \tilde{k}_m \rangle \right)}
$$
At the same time, compute the gating for each node (using max-pool and average-pool information):

\[ \vec{g}_i = \sigma \left( W_g \begin{bmatrix} \vec{h}_i \parallel \max_{j \in \mathcal{N}_i} W_m \vec{z}_j \parallel \sum_{j \in \mathcal{N}_i} \vec{z}_j / |\mathcal{N}_i| \end{bmatrix} \right) \]

Finally, attend over the values and apply the gating (distributed over \( K \) independent heads)—including a skip connection:

\[ \vec{h}_i' = \sigma \left( W_o \begin{bmatrix} \vec{h}_i \parallel K \parallel \vec{g}^{(k)}_i \odot \sum_{j \in \mathcal{N}_i} \alpha^{(k)}_{ij} \vec{v}^{(k)}_j \end{bmatrix} \right) \]
GaAN in action
DeepInf (Qiu et al., 2018)

- Modelling **influence locality** within large social networks.

- Let $s_u^t \in \{0, 1\}$ denote whether node $u$ has performed an action at any time $t' < t$.

- Aim to predict whether node $v$ ever performs the action ($s_v^{+\infty}$), given the action statuses of all of its $r$-hop neighbours at time $t$.

- First study where attentional mechanisms (such as GAT) appear to be *necessary* for surpassing baseline approaches (such as logistic regression or SVMs).
Datasets:

- **OAG** *(network: coauthorship; action: citation)*
- **Digg** *(network: friendship; action: vote up)*
- **Twitter** *(network: follow; action: retweet “Higgs”)*
- **Weibo** *(network: follow; action: retweet)*
DeepInf: qualitative analysis of attention
Successfully demonstrated the viability of attentional mechanisms on graphs to solving combinatorial problems (Euclidean TSP—each node is specified by \((x, y)\) coordinates).

A decoder computes the probability distribution for the next node to visit, \(\pi_t\), based on:

- a fixed-size encoding of the graph, \(\vec{h}_G\) (obtained by an encoder);
- the embeddings of the first and last visited node: \(\vec{h}_{\pi_1}, \vec{h}_{\pi_{t-1}}\).
- the embeddings \(\vec{h}_i\) of all nodes \(i\) still in the graph.

Then this probability distribution is optimised using REINFORCE (with a greedy rollout baseline).
Attention Solves Your TSP: Encoder

- Uses the *key-value* attention mechanism, as in GaAN.
- Every node attends over all others.
- We obtain node embeddings $\vec{h}_i$, as well as the graph embedding $\vec{h}_G$ (as their average).
Attention Solves Your TSP: Decoder

- First, create a **context node** containing $[\vec{h}_G, \vec{h}_{\pi_{t-1}}, \vec{h}_{\pi_1}]$.
- Then this node (multi-head) attends over all remaining nodes.
- Finally, the context node single-head attends over all remaining nodes, with the coefficients interpreted as probabilities.
Hyperbolic attention

Hyperbolic Attention Networks (Gulçehre et al., 2018)
Relational reasoning

Relation Networks (Santoro et al., 2017)
Modelling multi-agent interactions

The VAIN framework (Hoshen, 2017)
Neighbourhood attention

One-shot imitation learning (Duan et al., 2017)
Defence against adversarial examples

*PeerNets* (Svoboda *et al.*, 2018)
Mesh-based cortical parcellation

with Guillem Cucurull, Konrad Wagstyl et al. (MIDL 2018)
Antibody binding residue prediction

with Andreea Deac (ICML WCB 2018)
Thank you!

Questions?

petar.velickovic@cst.cam.ac.uk

http://www.cst.cam.ac.uk/~pv273/

http://petar-v.com/GAT

https://github.com/PetarV-/GAT