UNIVERSITY OF CAMBRIDGE

Keeping our graphs attentive

Petar Veličković

Artificial Intelligence Group
Department of Computer Science and Technology, University of Cambridge, UK

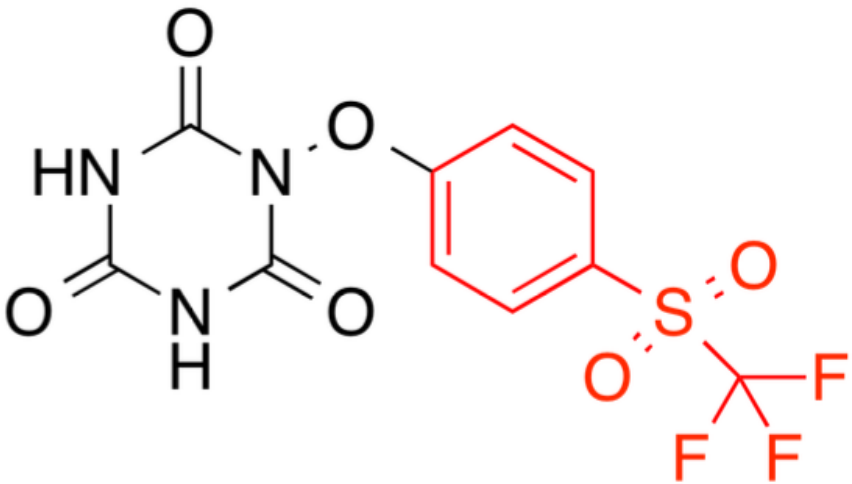
AMLab Seminar **10 April 2018**

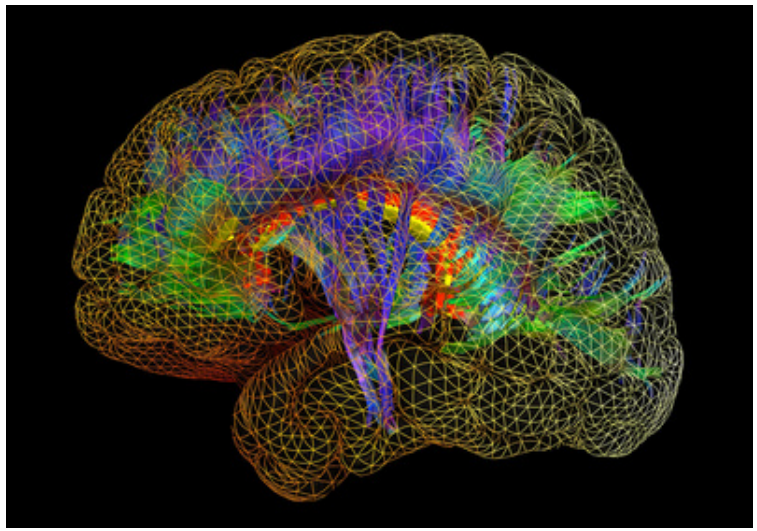
Introduction

- 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*).
  - Applications to relational reasoning, multi-agent interaction, cortical mesh segmentation and paratope prediction.
  - An ongoing project in graph classification (with Thomas Kipf).

Graphs are **everywhere**!



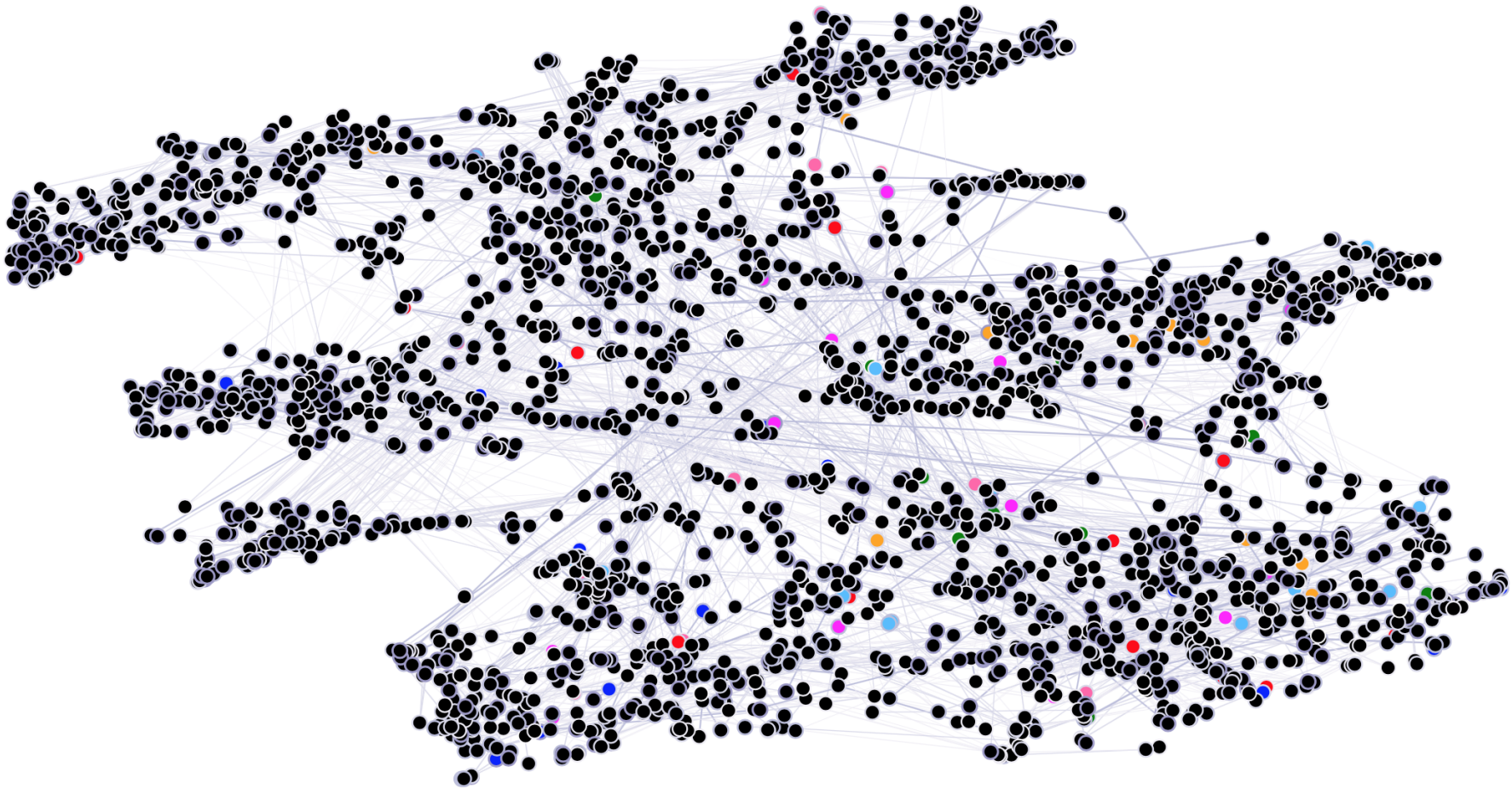


Mathematical formulation

- We will focus on the **node classification** problem:
  - **Input**: a matrix of *node features*, $\mathbf{F} \in \mathbb{R}^{N \times F}$, with F features in each of the N nodes, and an *adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{N \times N}$.
  - **Output**: a matrix of *node class probabilities*, $\mathbf{Y} \in \mathbb{R}^{N \times C}$, such that $Y_{ij} = \mathbb{P}(\textit{Node } i \in \textit{Class } j)$.
- We also assume, for simplicity, that the edges are **unweighted** and **undirected**:
  - That is, $A_{ij} = A_{ji} = \begin{cases} 1 & i \leftrightarrow j \\ 0 & \textit{otherwise} \end{cases}$

  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.

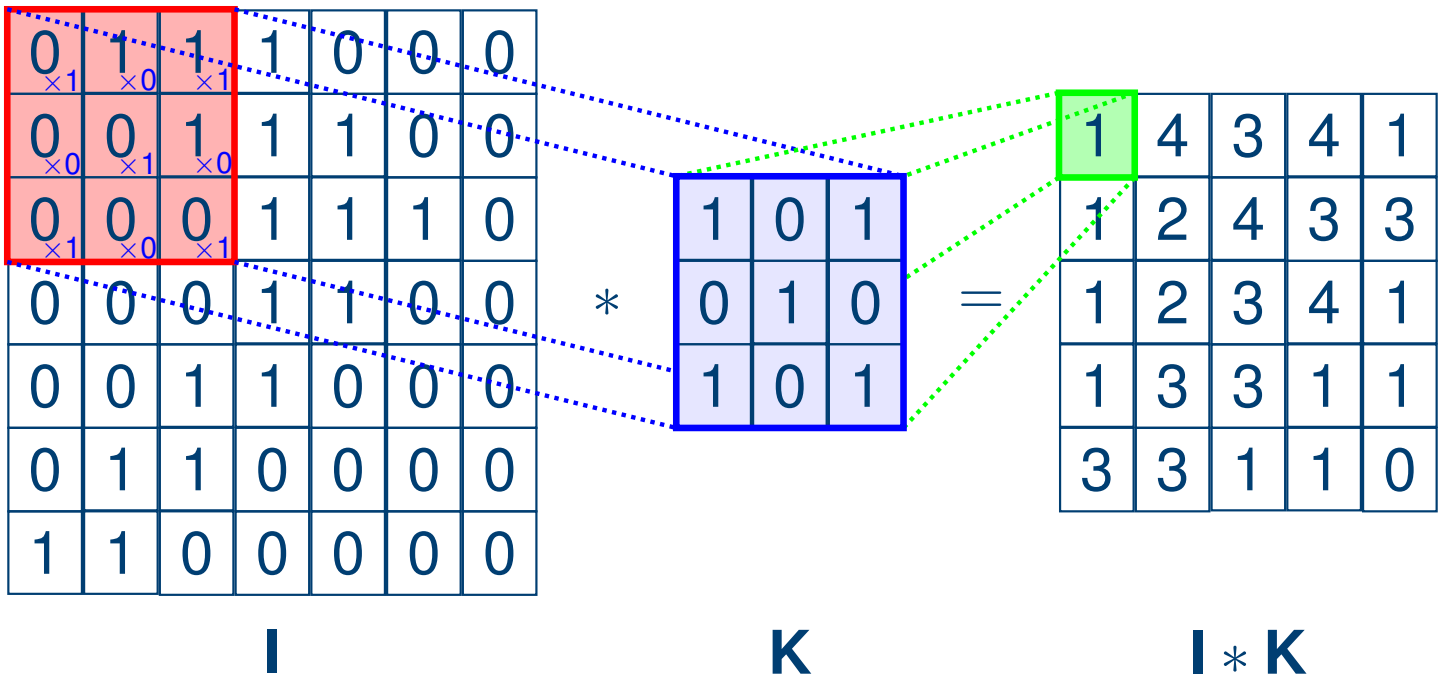
- **Main idea**: Compute node representations $\vec{h}_i$ based on the initial features $\vec{f}_i$ and the graph structure, and then use $\vec{h}_i$ to classify each node independently.

The silver bullet—a *convolutional* layer

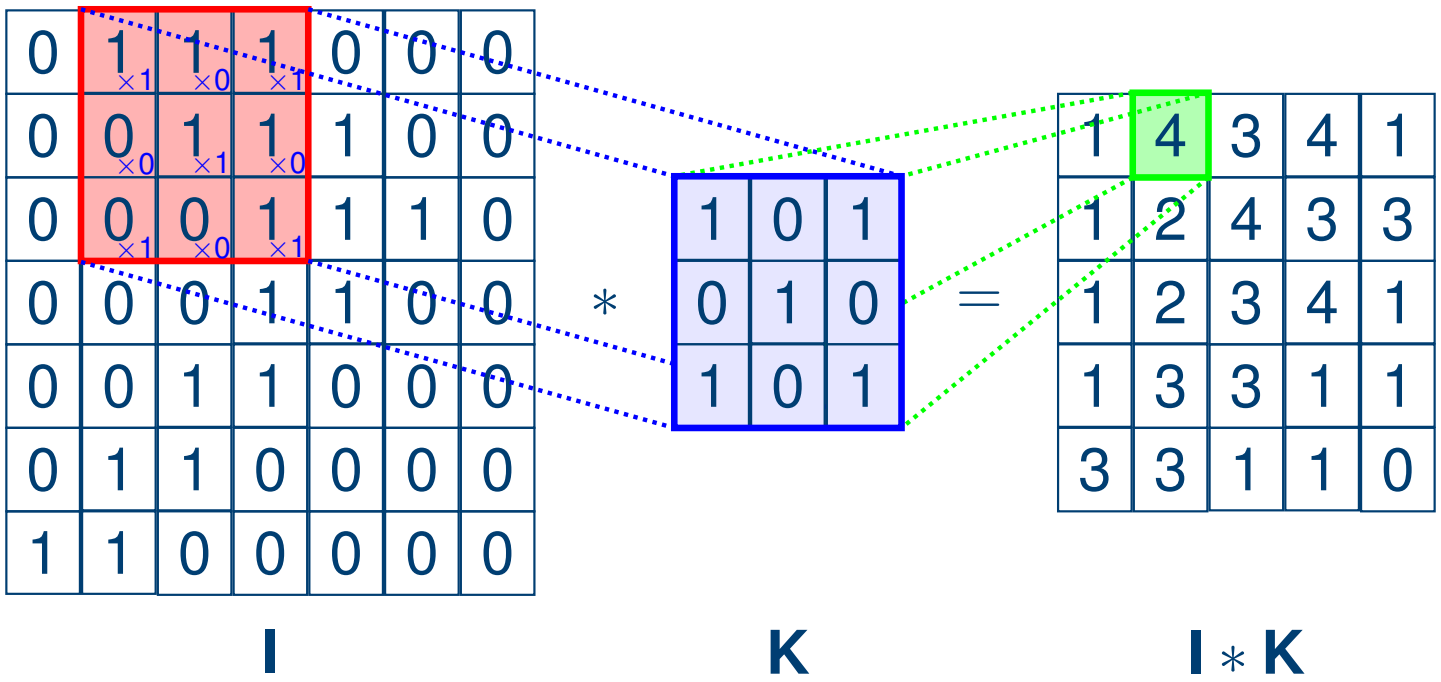
- It would be, in particular, highly appropriate if we could somehow generalise the *convolutional operator* (as used in CNNs) to operate on arbitrary graphs!

- 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).

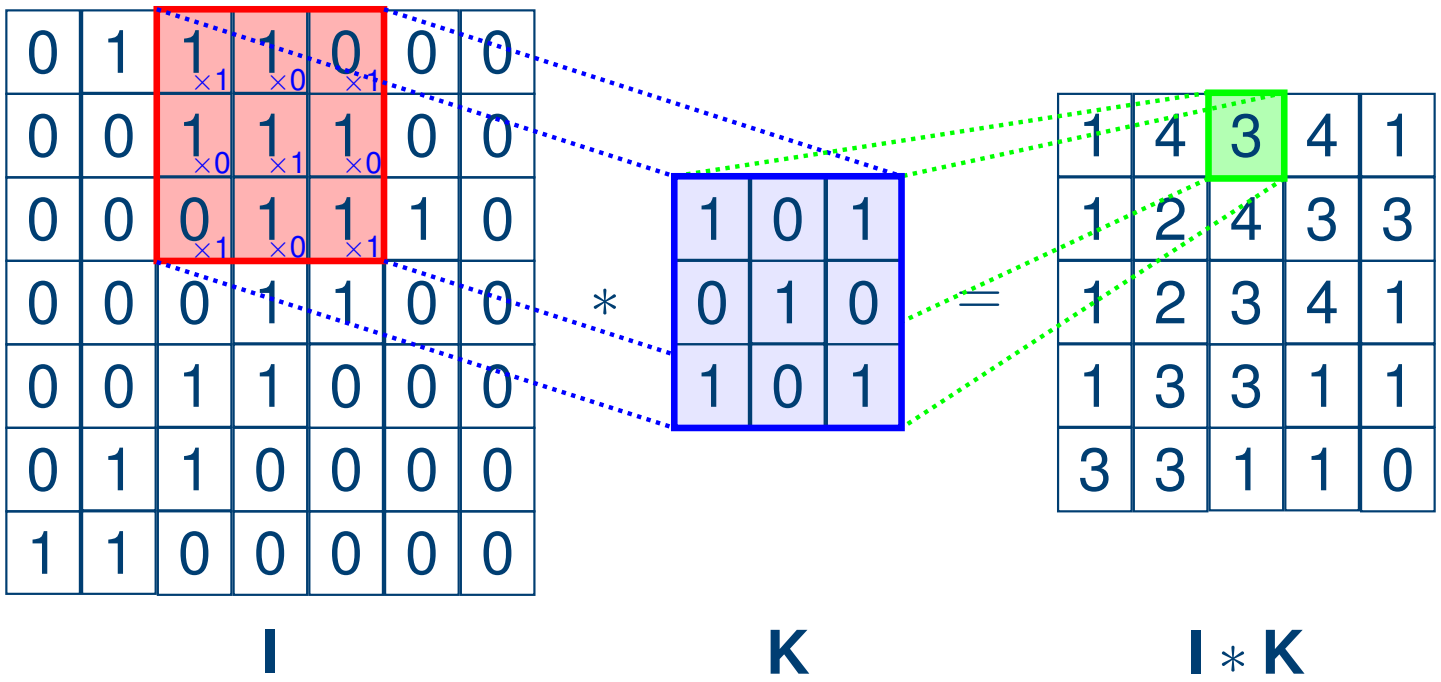
Convolution on images



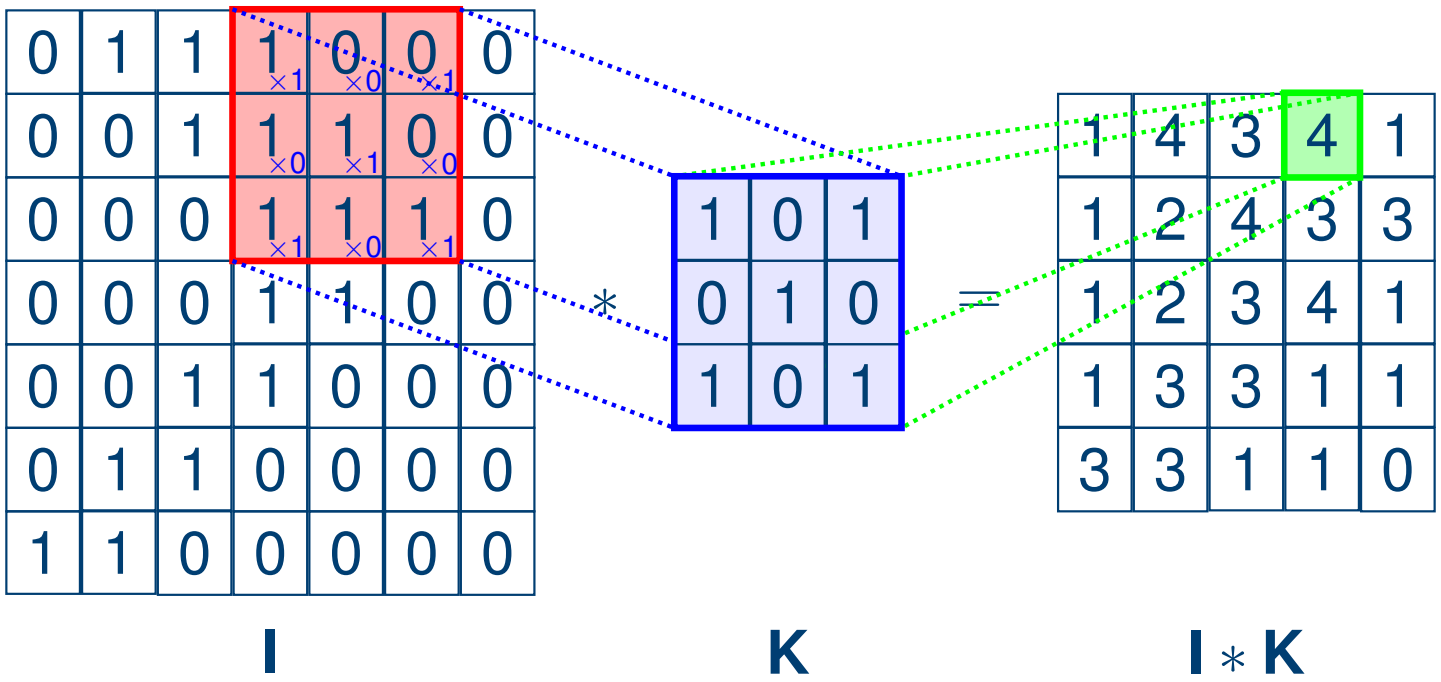
Convolution on images



Convolution on images



Convolution on images



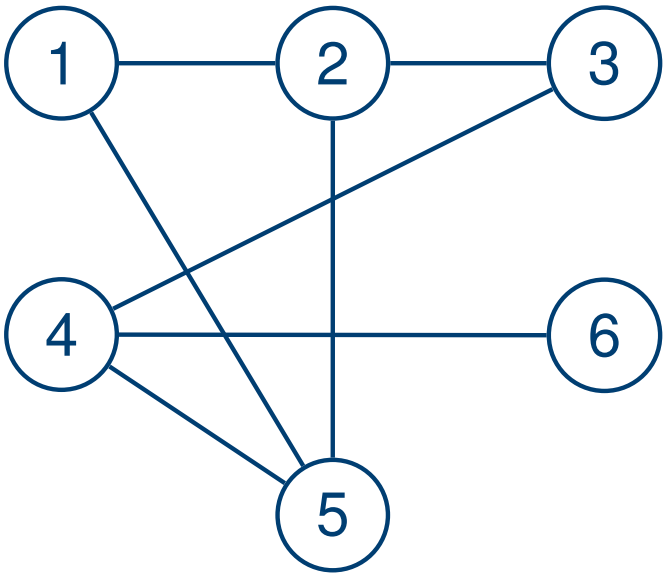
Challenges with graph convolutions

- 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!

Spectral graph convolution

- 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**, $\mathbf{L}$, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix (diagonal matrix with $D_{ii} = deg(i)$) and $\mathbf{A}$ is the adjacency matrix.
- Alternately, we may use the **normalised graph Laplacian**, $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$.

Graph Laplacian example



$$\mathbf{L} = \begin{bmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Graph Fourier Transform

- The Laplacian is symmetric and positive semi-definite; we can therefore diagonalise it as $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{\Lambda}$ is a diagonal matrix of its eigenvalues.
- This means that multiplying the feature matrix by $\mathbf{U}^T$ allows us to enter the *spectral domain* for the graph! Therein, convolution just amounts to pointwise multiplication.
- This "Graph Fourier Transform" is the essence of the work of Bruna *et al.* (ICLR 2014).

Graph Fourier Transform, *cont'd*

- To convolve two signals using the convolution theorem:

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

- Therefore, a *learnable convolutional layer* amounts to:

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

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

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

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

$$T_0(x) = 1 \qquad T_1(x) = x \qquad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$

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:

$$\vec{t}_0 = \mathbf{W}\vec{h}_i \qquad \vec{t}_1 = \mathbf{L}\mathbf{W}\vec{h}_i \qquad \vec{t}_k = 2\mathbf{L}\vec{t}_{k-1} - \vec{t}_{k-2}$$

  where each step constitutes a **sparse** multiplication with **L**.

- The number of parameters is **fixed** (equal to K weights).

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

  $\Longrightarrow$ The operator is **K-localised**!

Properties of Chebyshev networks, *cont'd*

- To avoid issues with exploding or vanishing signals, typically a scaled version of $\mathbf{L}$ is fed into the algorithm:

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

  where λ_{max} is the largest eigenvalue of $\mathbf{L}$.

- 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 3×3 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.

GCNs

- The **Graph Convolutional Network** (GCN) of Kipf & Welling (ICLR 2017) further fine-tunes the Chebyshev framework.
- Setting $K = 1$ and assuming $\lambda_{max} \approx 2$ allows for redefining a single convolutional layer as simply:

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

  which significantly improves computational performance on larger graphs and predictive power on small training sets.
- However, the previous issue is *still there...*

Applicability to inductive problems

- 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. . .

Molecular fingerprinting networks

- 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 $\mathbf{H}_d$ 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*.

GraphSAGE

- 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.

Reminder: Self-attention

- A recent development in attentional mechanisms concerns **self-attention**; a scenario where the input *attends over itself*:

$$\alpha_{ij} = a(\vec{h}_i, \vec{h}_j)$$
$$\vec{h}'_i = \sum_j softmax_j(\alpha_{ij})\vec{h}_j$$

  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:

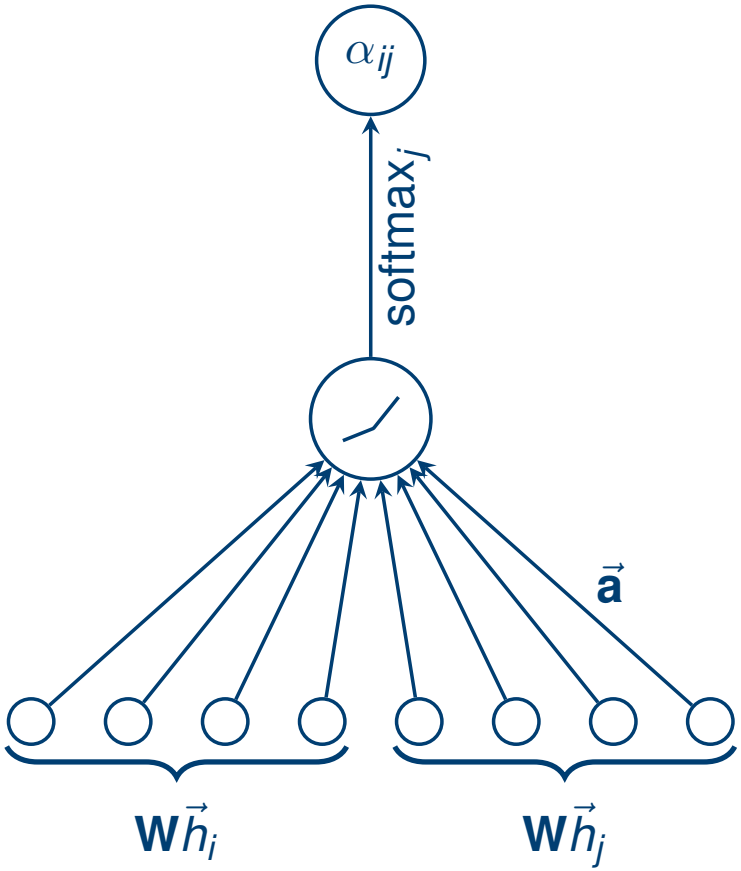
$$e_{ij} = a(\vec{h}_i, \vec{h}_j)$$

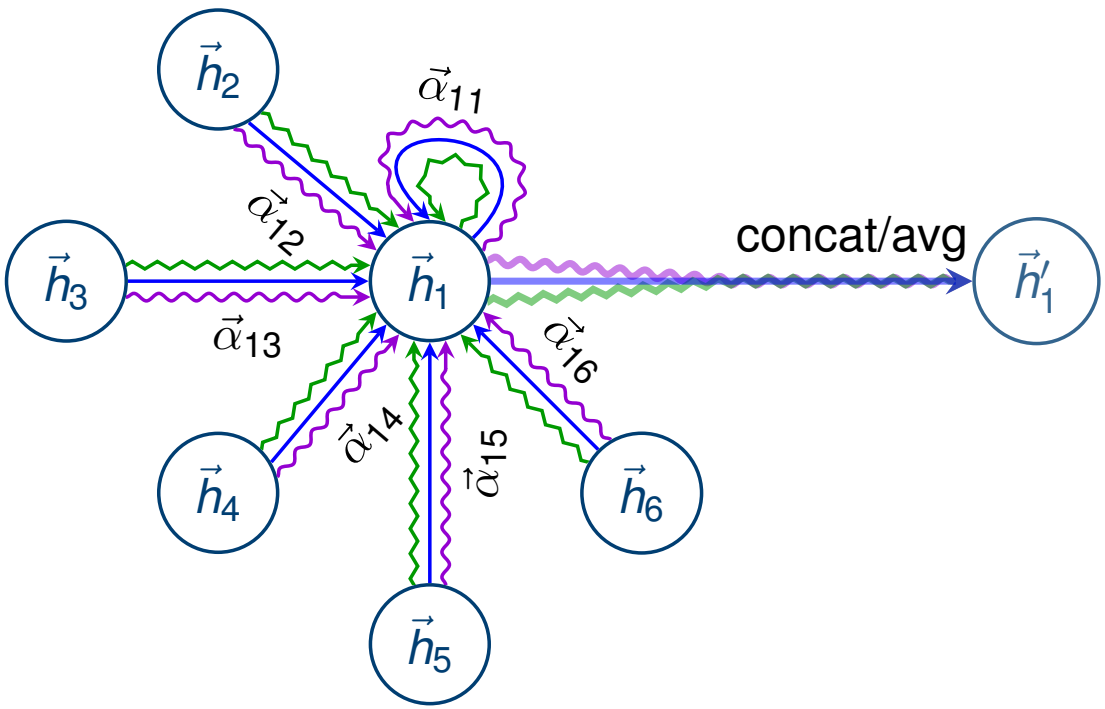
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$$

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

- Some further optimisations (like *multi-head attention* and *dropout* on the α_{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.

	Transductive			*Inductive*
	Cora	**Citeseer**	**Pubmed**	**PPI**
# Nodes	2708	3327	19717	56944 (24 graphs)
# Edges	5429	4732	44338	818716
# Features/Node	1433	3703	500	50
# Classes	7	6	3	121 (multilabel)
# Training Nodes	140	120	60	44906 (20 graphs)
# Validation Nodes	500	500	500	6514 (2 graphs)
# Test Nodes	1000	1000	1000	5524 (2 graphs)

Results on Cora/Citeseer/Pubmed

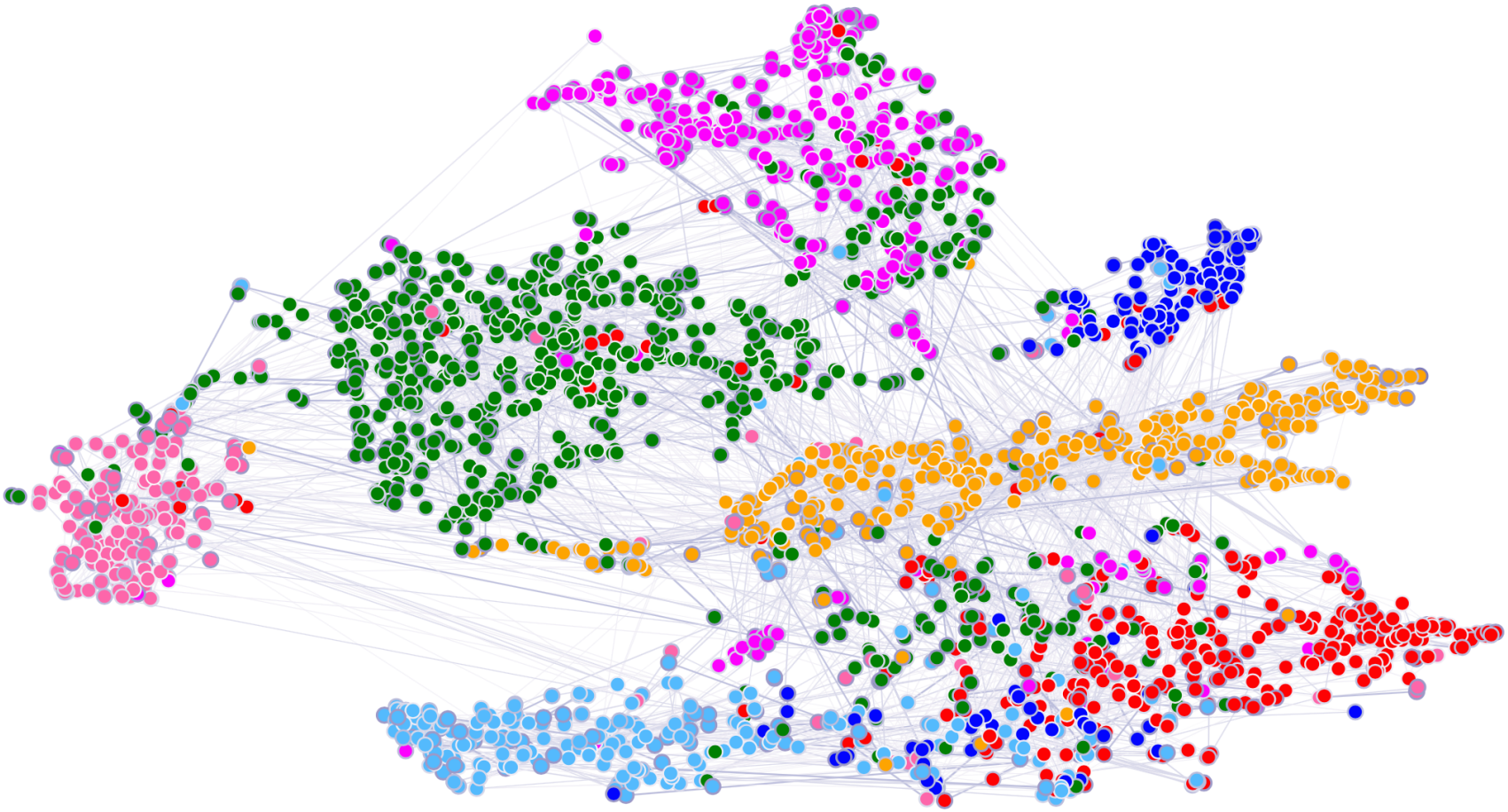
	Transductive		
Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg	59.5%	60.1%	70.7%
SemiEmb	59.0%	59.6%	71.7%
LP	68.0%	45.3%	63.0%
DeepWalk	67.2%	43.2%	65.3%
ICA	75.1%	69.1%	73.9%
Planetoid	75.7%	64.7%	77.2%
Chebyshev	81.2%	69.8%	74.4%
GCN	81.5%	70.3%	**79.0%**
MoNet	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	**79.0** $\pm 0.3\%$
GAT (ours)	**83.0** $\pm 0.7\%$	**72.5** $\pm 0.7\%$	**79.0** $\pm 0.3\%$

Results on PPI

	Inductive
Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN	0.500
GraphSAGE-mean	0.598
GraphSAGE-LSTM	0.612
GraphSAGE-pool	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

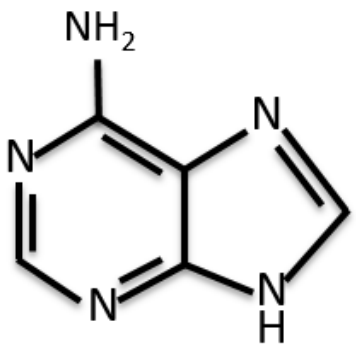
Here, *Const-GAT* is a GCN-like inductive model.

t-SNE + attention coefficients on Cora

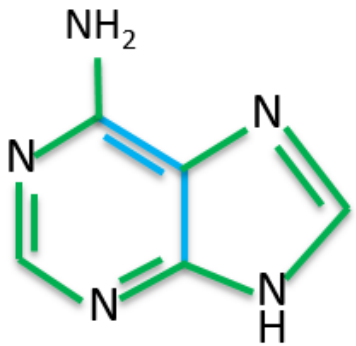


Incorporating edge context

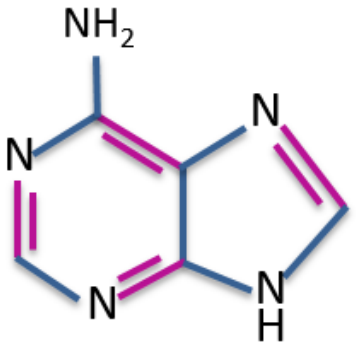
- The attentional setup of GAT treats each edge *equally*.
- This will not be appropriate for inputs such as **chemical compounds**, wherein the same atom can possess identical *neighbourhoods* but with different *bonds*!



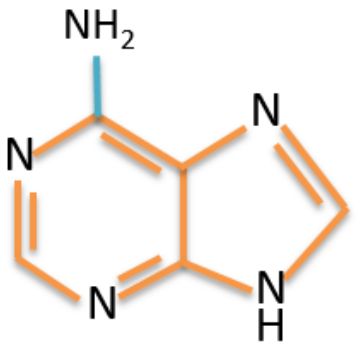
Adenine



Atom Pair Type



Bond Order



Ring Status

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 α_{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 $\mathbf{A}^i$ for each edge attribute i:

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

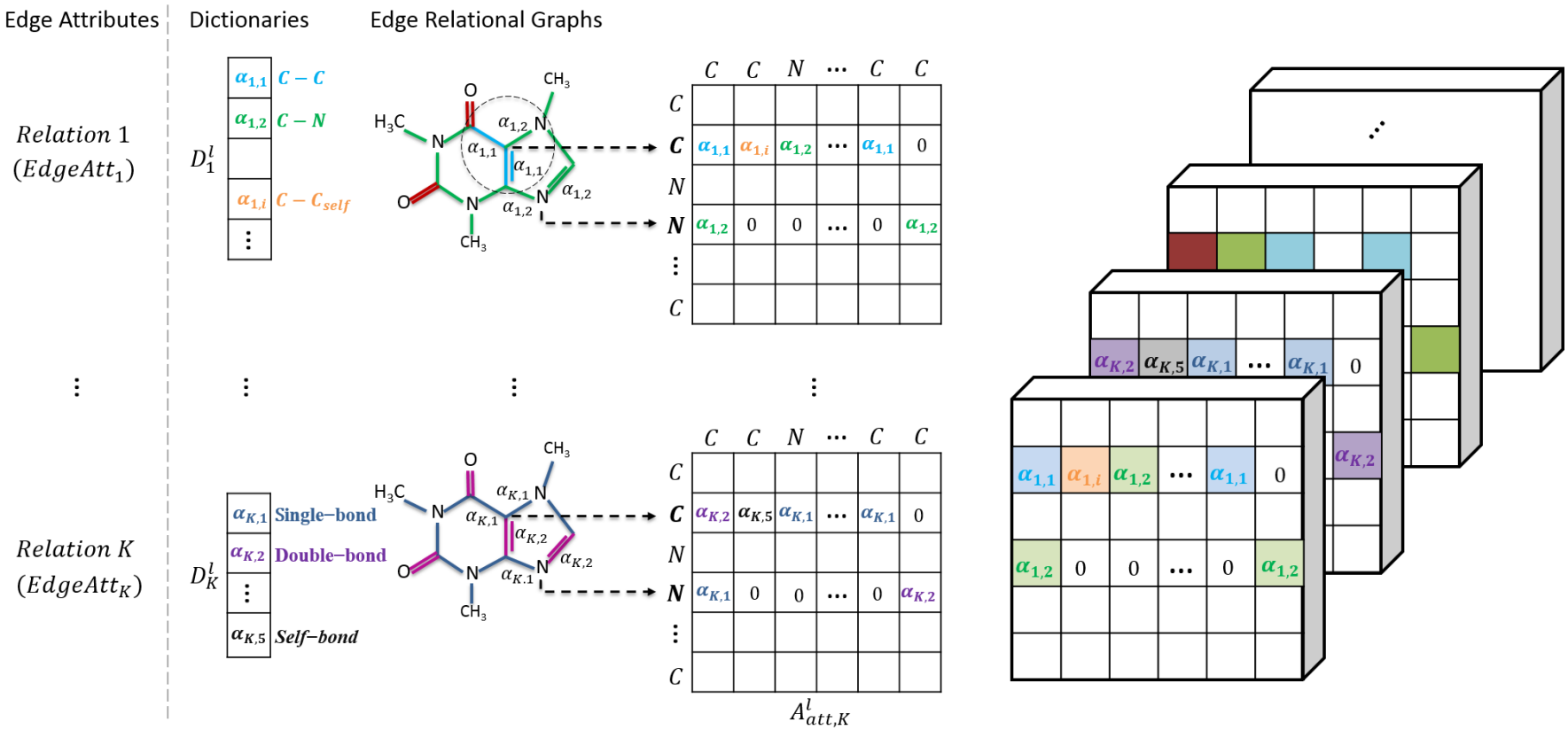
  which are then softmax-normalised:

$$\widetilde{\mathbf{A}}^i_{st} = \frac{\exp\left(\mathbf{A}^i_{st}\right)}{\sum_k \exp\left(\mathbf{A}^i_{kt}\right)}$$

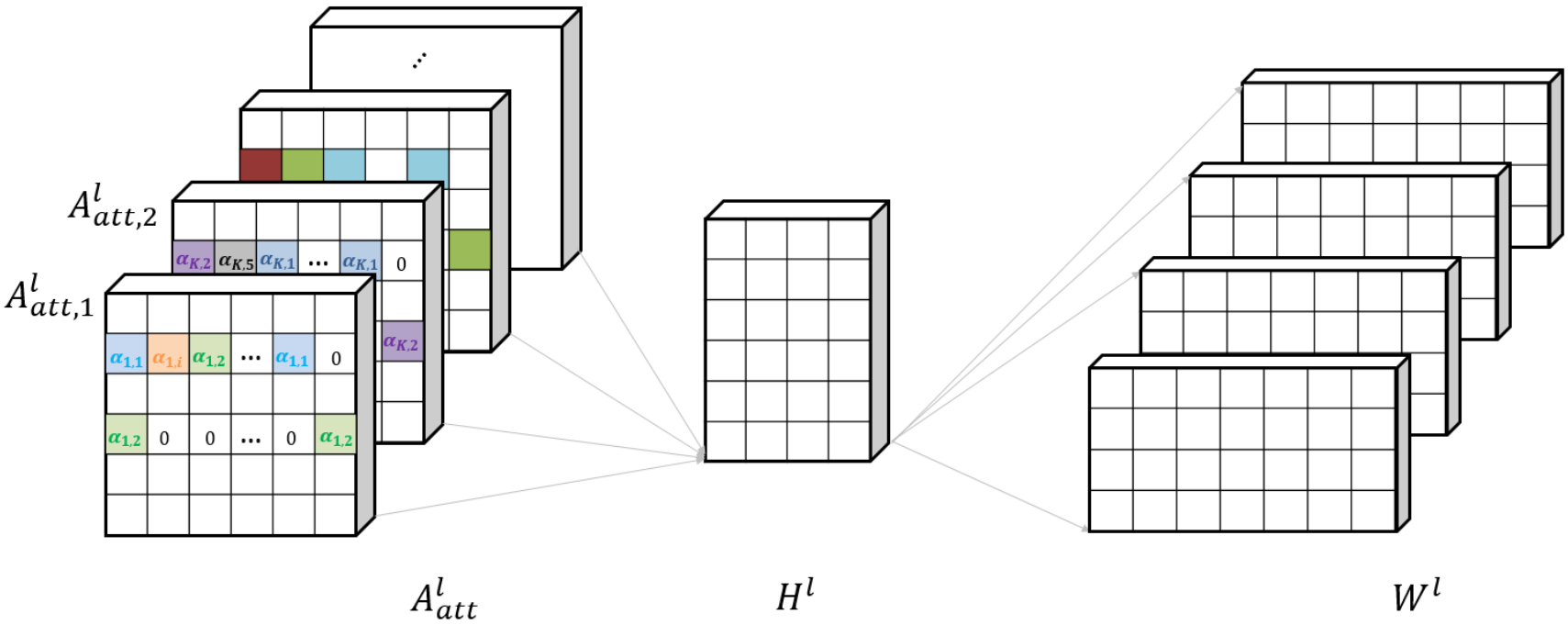
- We can then use each of these as a separate attention head, and e.g. concatenate their outputs (for node features $\mathbf{H}$):

$$\mathbf{H}' = \Big\Vert_{i=1}^{K} \sigma\left(\widetilde{\mathbf{A}}^i \mathbf{H}\mathbf{W}\right)$$

EAGCN in action: computing $\mathbf{A}^i$



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* $\vec{h}_i$ and node *reference vectors* $\vec{z}_j$ (useful to decouple when working on **temporal graphs**).
- First, derive *queries, keys and values* for the attention:

$$\vec{q}_i = \mathbf{W}_q \vec{h}_i \qquad \vec{k}_i = \mathbf{W}_k \vec{z}_i \qquad \vec{v}_i = \mathbf{W}_v \vec{z}_i$$

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

$$\alpha_{ij} = \frac{\exp\left(\langle \vec{q}_i, \vec{k}_j \rangle\right)}{\sum_{m \in \mathcal{N}_i} \exp\left(\langle \vec{q}_i, \vec{k}_m \rangle\right)}$$

GaAN dataflow, *cont'd*

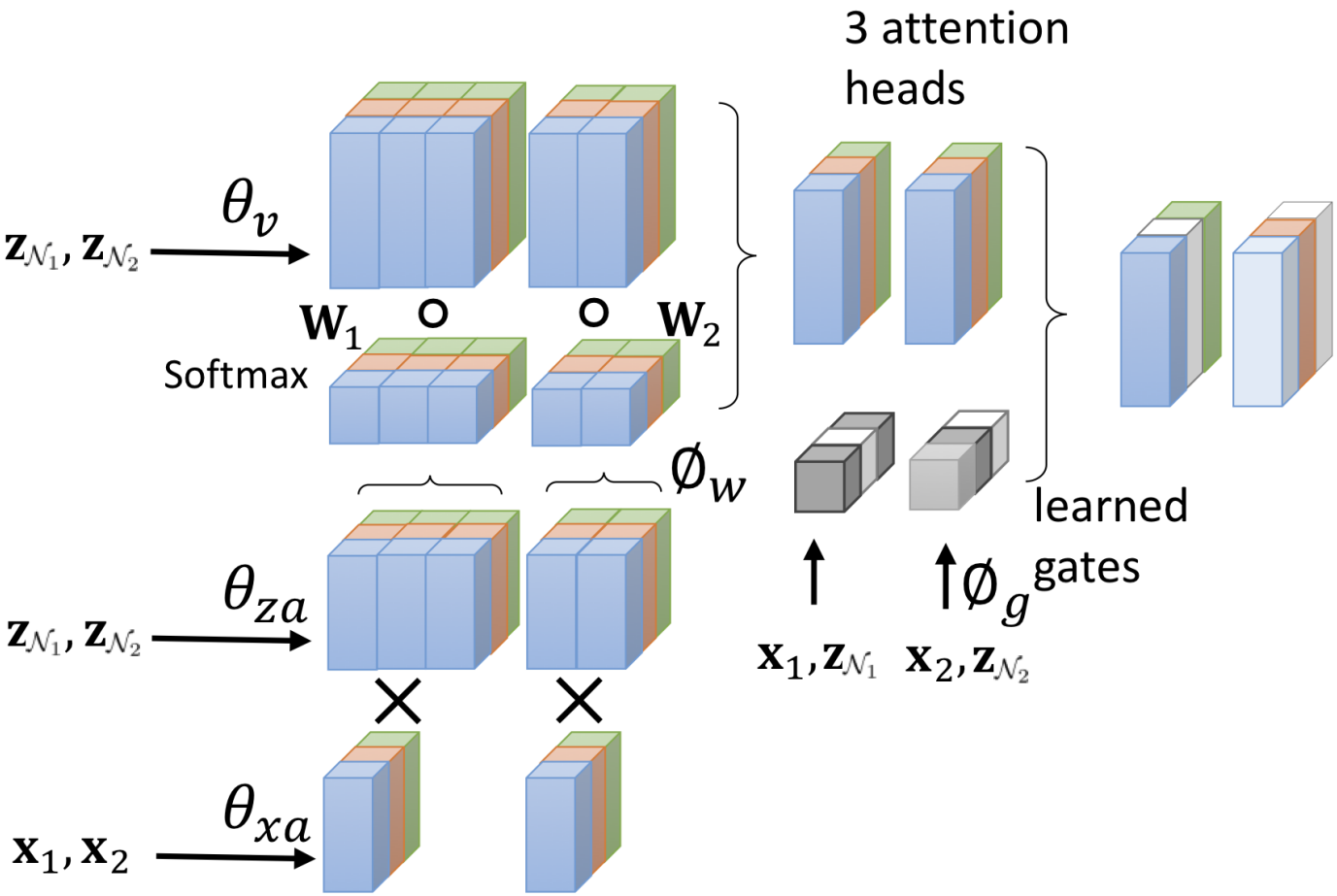
- At the same time, compute the gating for each node (using *max-pool* and *average-pool* information):

$$\vec{g}_i = \sigma\left(\mathbf{W}_g\left[\vec{h}_i \parallel \max_{j\in\mathcal{N}_i}\mathbf{W}_m\vec{z}_j \parallel \frac{\sum_{j\in\mathcal{N}_i}\vec{z}_j}{|\mathcal{N}_i|}\right]\right)$$

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

$$\vec{h}'_i = \sigma\left(\mathbf{W}_o\left[\vec{h}_i \parallel \Big\Vert_{k=1}^{K}\vec{g}_i^{(k)}\odot\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\vec{v}_j^{(k)}\right]\right)$$

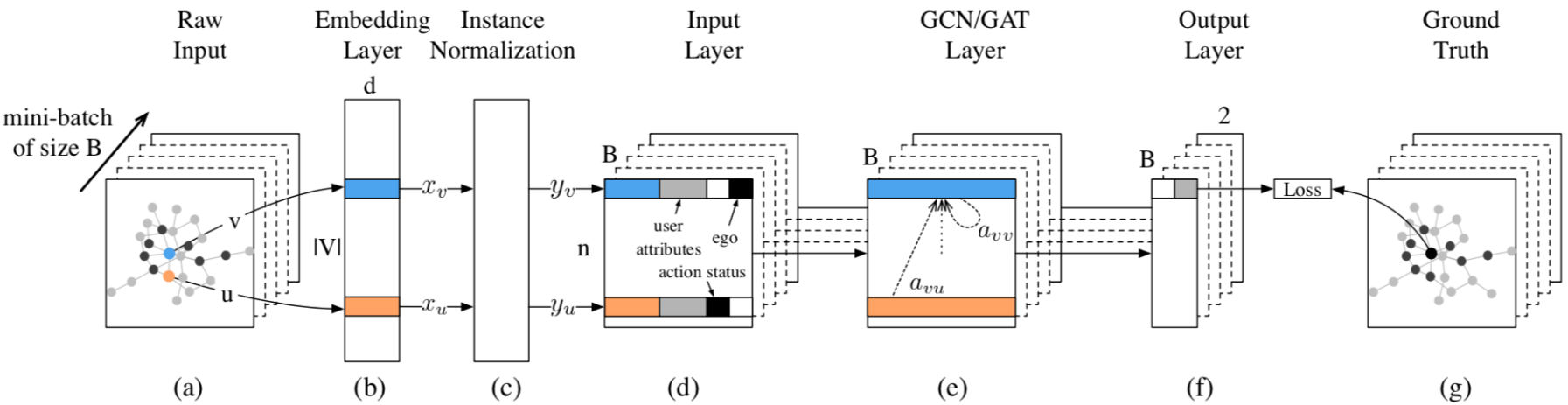
GaAN in action



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).

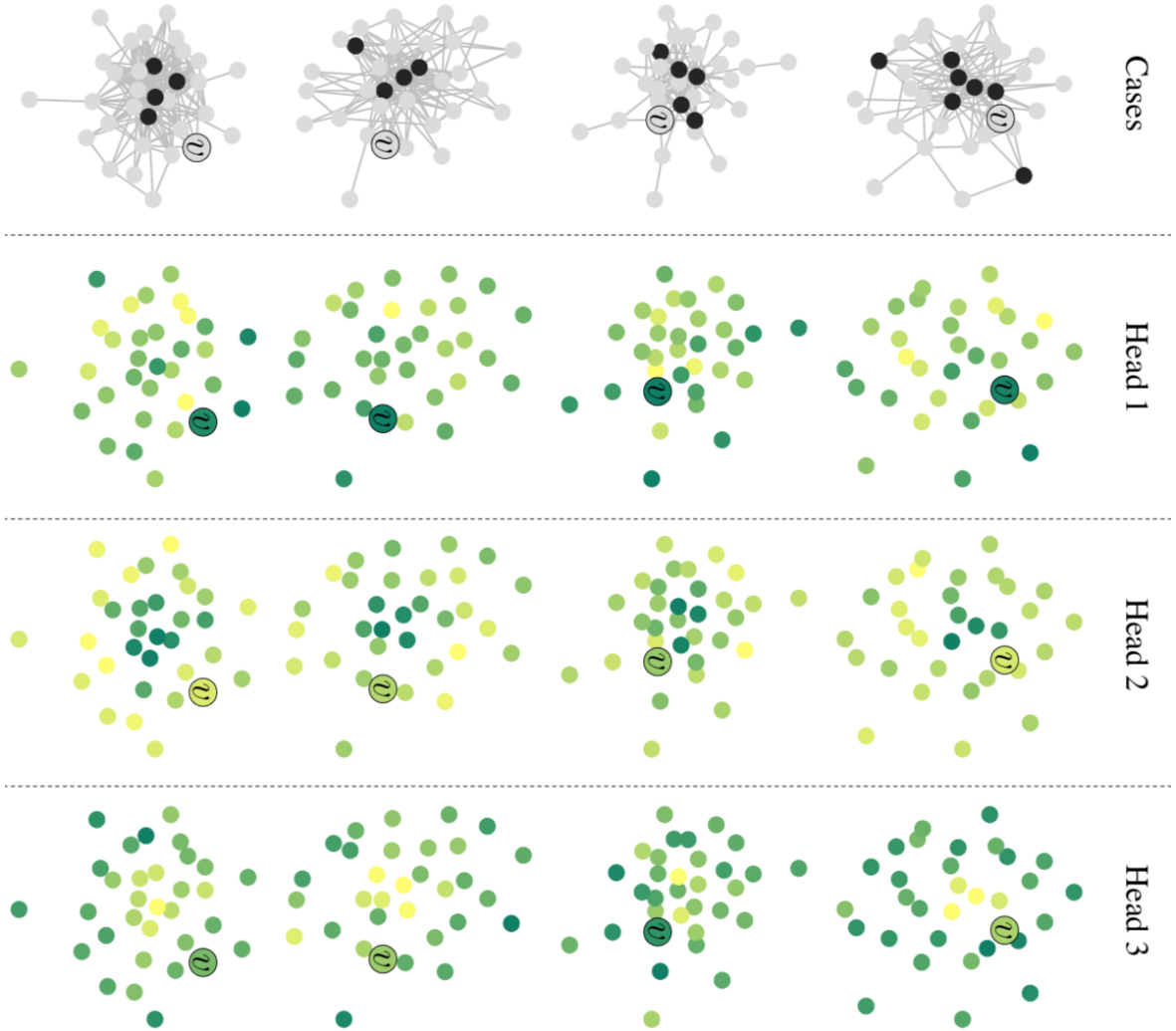
DeepInf pipeline



Datasets:

- **OAG** (*network*: coauthorship; *action*: citation)
- **Digg** (*network*: friendship; *action*: vote up)
- **Twitter** (*network*: follow; *action*: retweet “Higgs”)
- **Weibo** (*network*: follow; *action*: retweet)

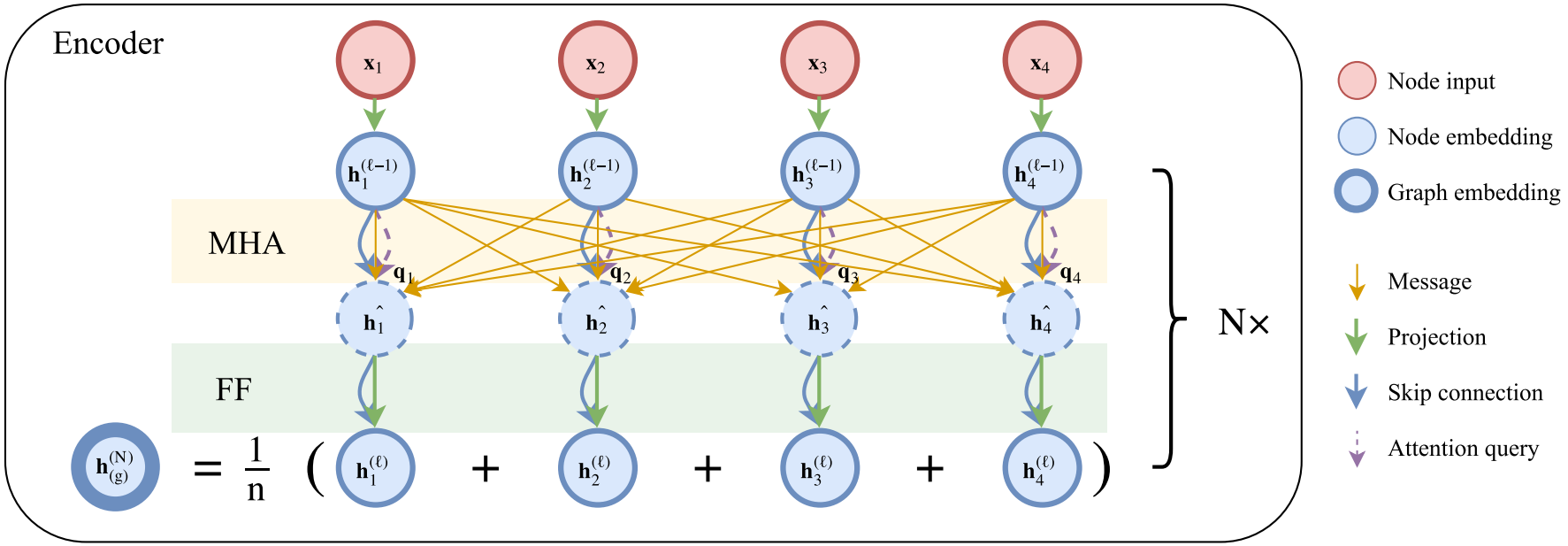
DeepInf: qualitative analysis of attention



Attention Solves Your TSP (Kool & Welling, 2018)

- 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, π_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).

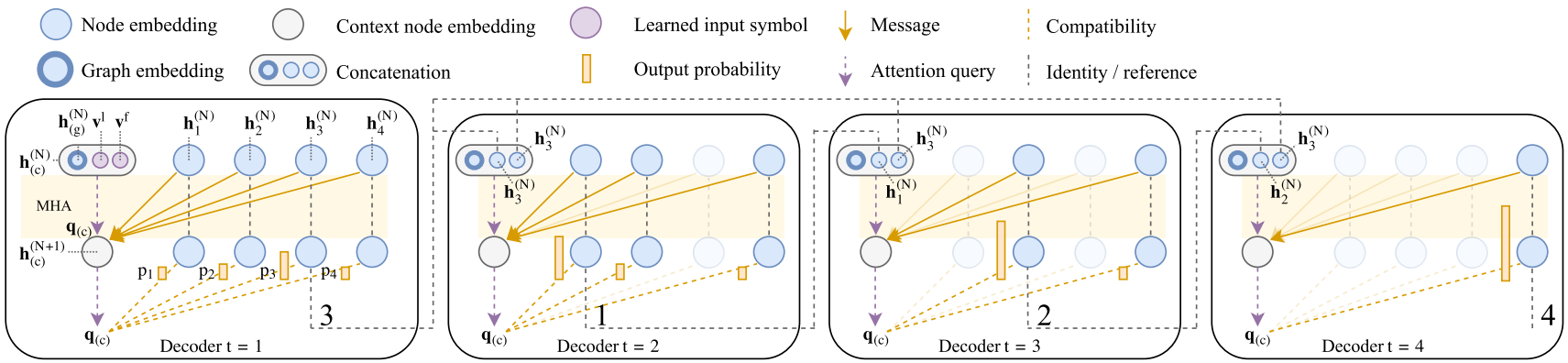
Attention Solves Your TSP: Encoder



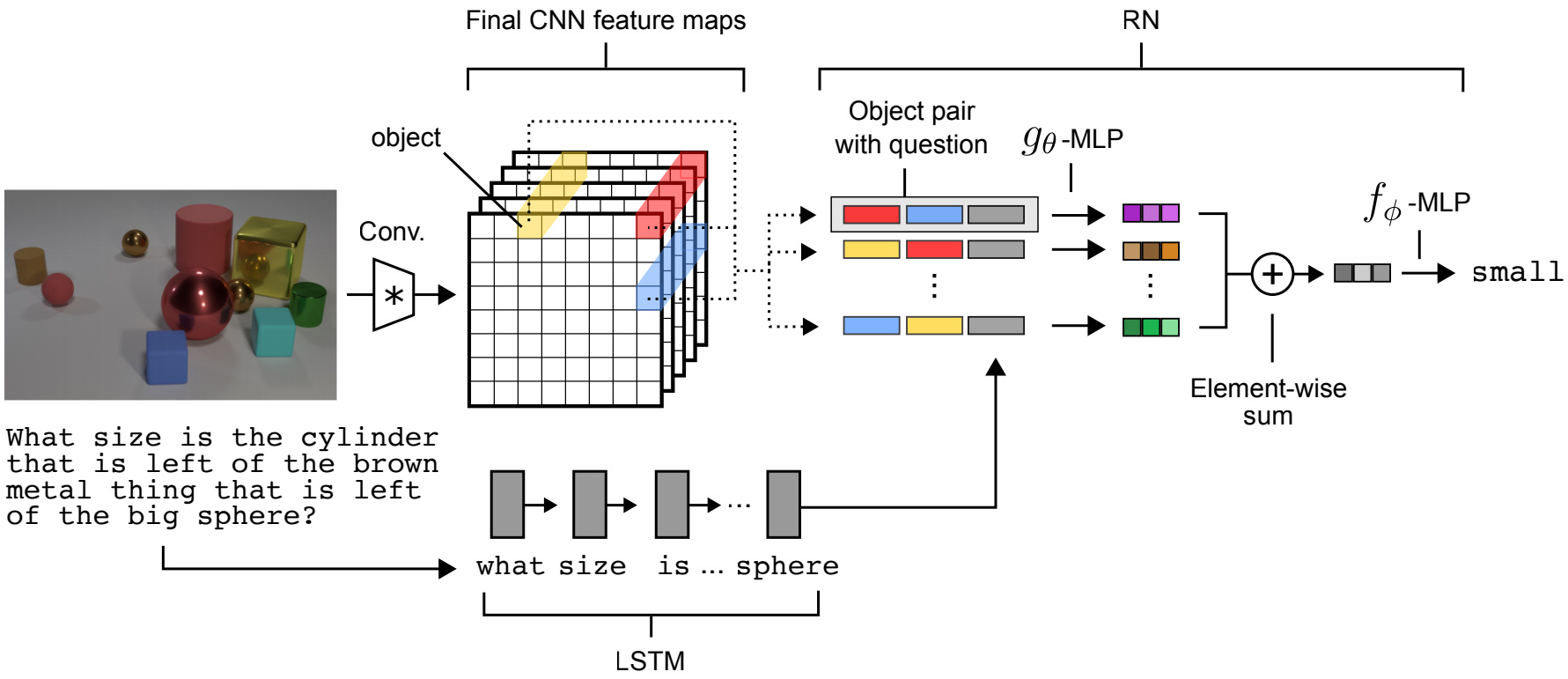
- Uses the *key-value* attention mechanism, as in GaAN.
- Every node attends over all others.
- We obtain node embeddings $\vec{h}_i$, as well as the graph embedding $\vec{h}_G$ (as their average).

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.



Relational reasoning

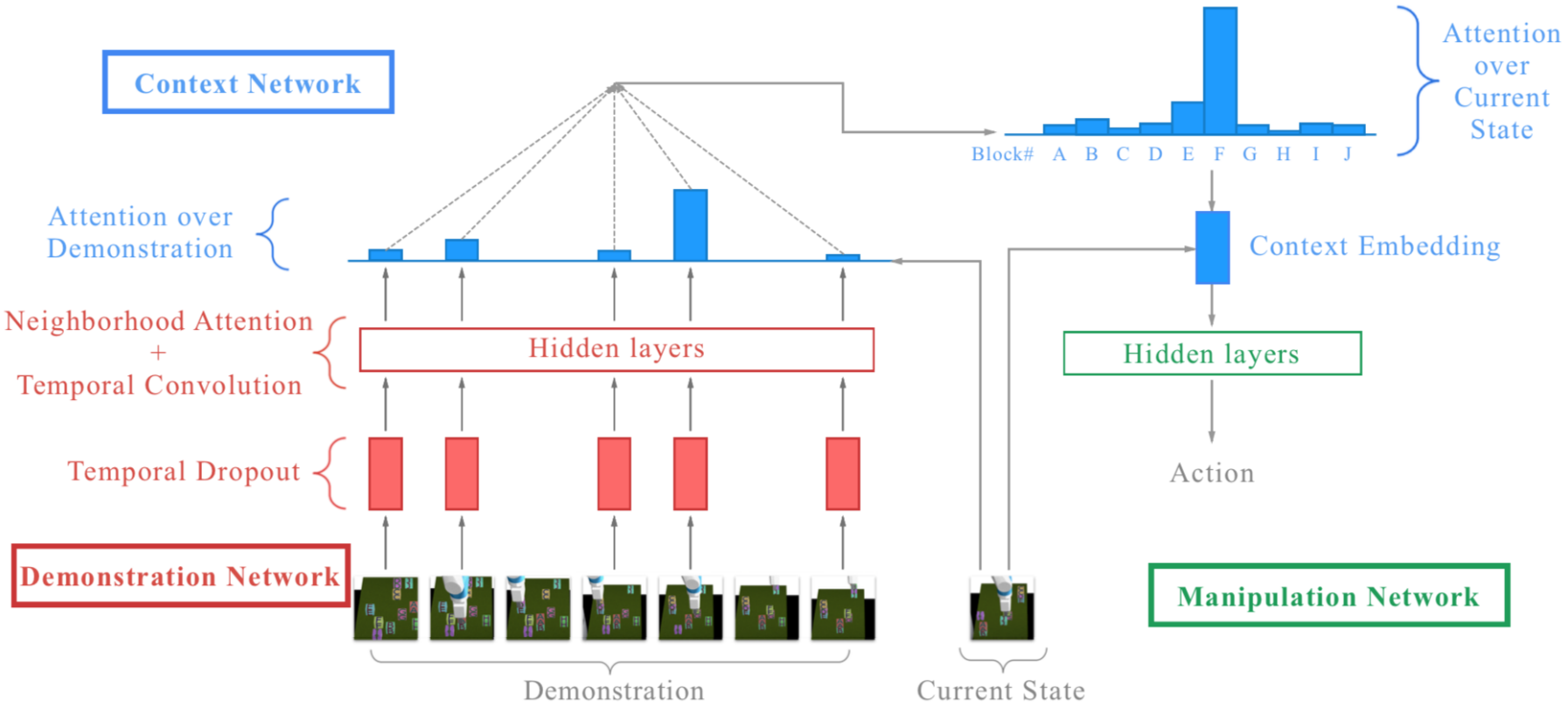


Relation Networks (Santoro *et al.*, 2017)

Modelling multi-agent interactions

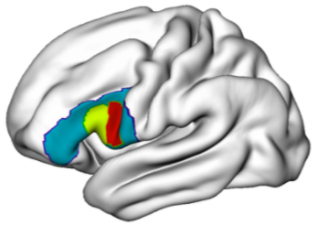
The *VAIN* framework (Hoshen, 2017)

Neighbourhood attention

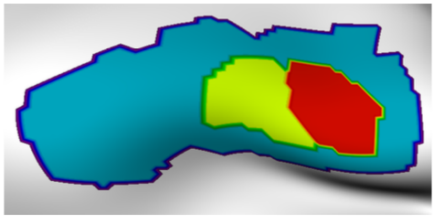


One-shot imitation slearning (Duan *et al.*, 2017)

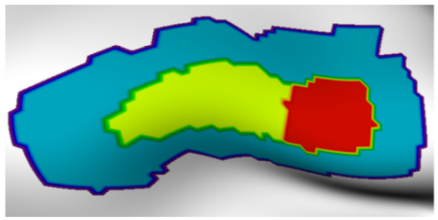
Mesh-based cortical parcellation



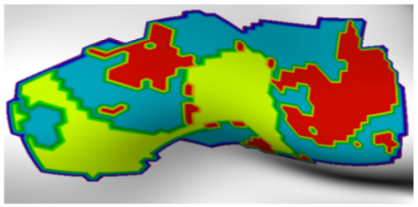
(a) Brain



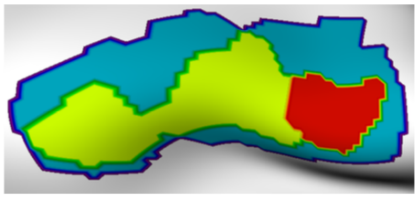
(b) Ground truth



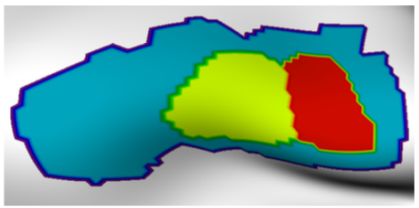
(c) NodeAVG



(d) NodeMLP



(e) Jakobsen et al. [22]



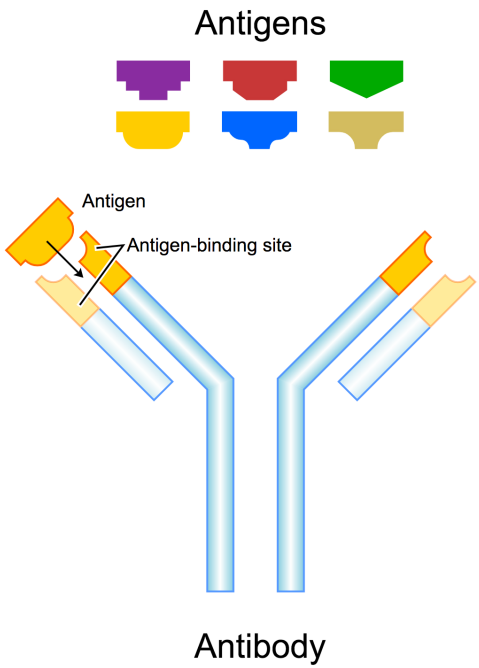
(f) GCN

(g) GAT

with *Guillem Cucurull, Konrad Wagstyl et al.* (NIPS BigNeuro 2017)

Motivation for antibody design

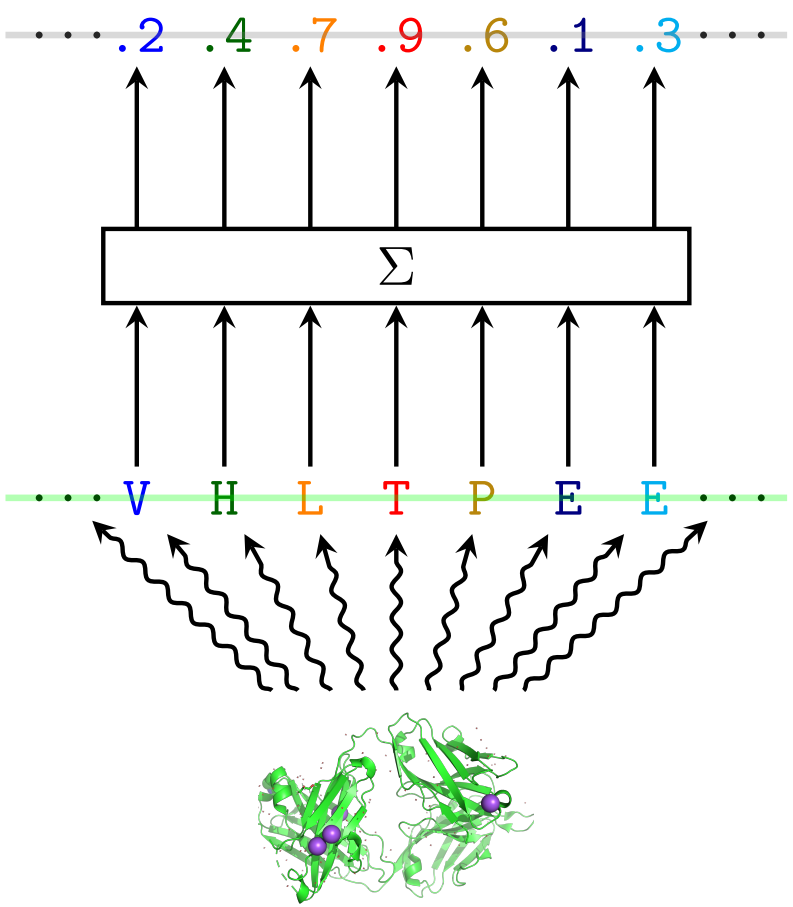
- Antibodies are
  - Y-shaped proteins
  - a critical part of our immune system
- They neutralise pathogenic bacteria and viruses by tagging the antigen in a ”lock and key” system.
- Designing our own arbitrary antibodies would be a big step towards personalised medicine.



Towards personalised medicine

- Generating an antibody requires first predicting the specific amino acids (the **paratope**) which participate in the neutralisation of the antigen.
- **Input**: a sequence of (one-hot encoded) antibody amino acids. *(+ a sequence of (one-hot encoded) antigen amino acids)*
- **Output**: probability for each amino acid to participate in the binding with the antigen.

Paratope prediction



$\mathbb{P}(\rho_i \text{ is binding})$

Paratope predictor

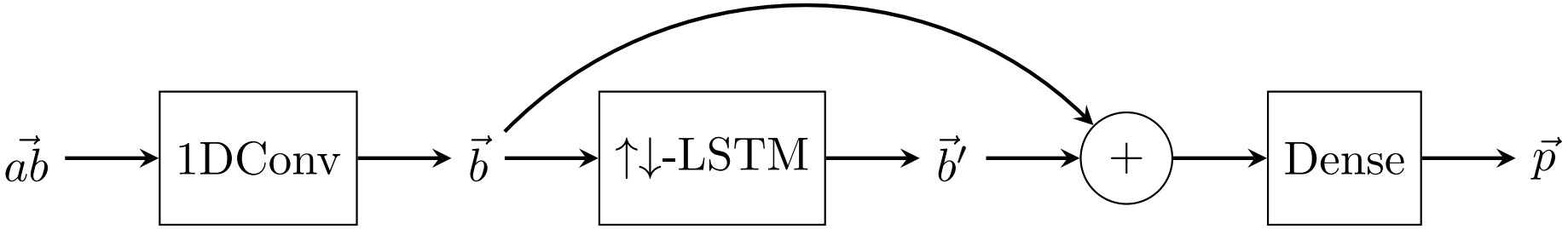
Amino acid sequence

Antibody

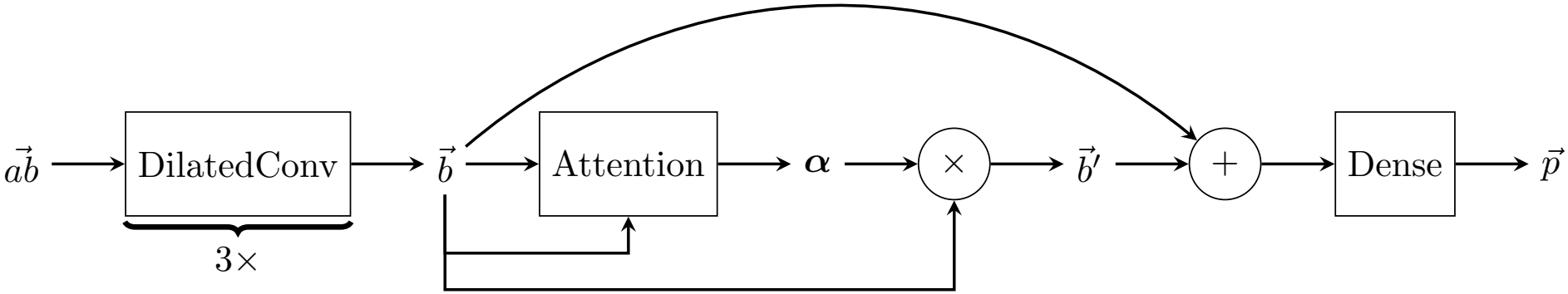
Related work

- i-Patch (Krawczyk *et al.* (2013)) is a hard-coded physical model which requires expensive data (e.g. positional information of each atom of both antibody and antigen).
- ProABC (Olimpieri *et al.* (2013)) uses a shallow classifier on antibody sequence data only.
- I have contributed to the two first viable deep learning architectures in this space, setting the new state-of-the-art without requiring positional information:
  - **Parapred** (Bioinformatics) (*with Edgar Liberis*), using a convolutional-recurrent neural network architecture.
  - **AG-Fast-Parapred** (*with Andreea Deac*), replacing these layers with dilated convolutions and (self-)attention, allowing for faster execution and integrating antigen sequence data.

The Parapred and Fast-Parapred architecture

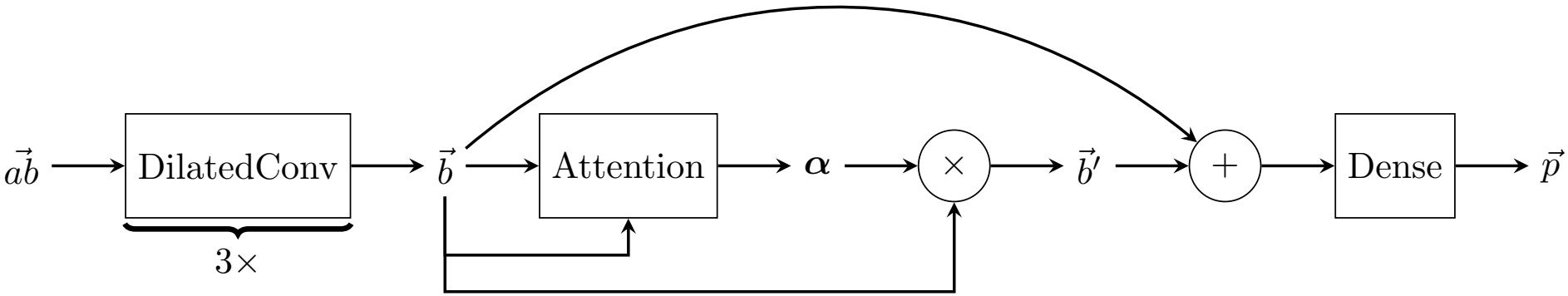


Parapred *(with Edgar Liberis)*



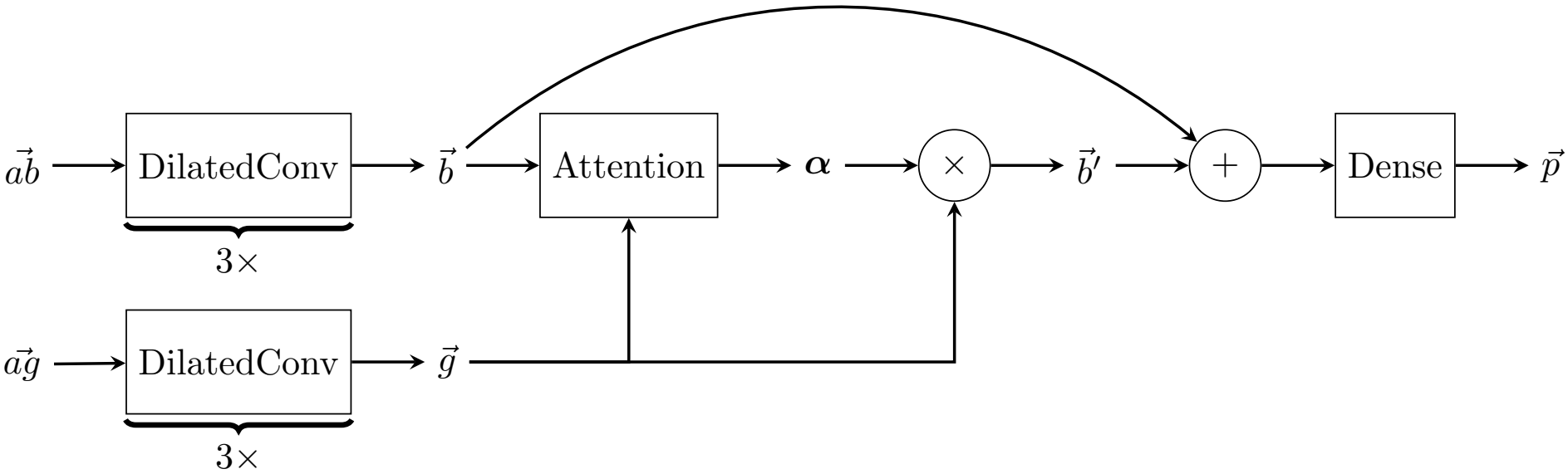
Fast-Parapred *(with Andreea Deac)*

Cross-modal attentive Parapred



Fast-Parapred *(with Andreea Deac)*

Cross-modal attentive Parapred



AG-Fast-Parapred *(with Andreea Deac)*

Attention!

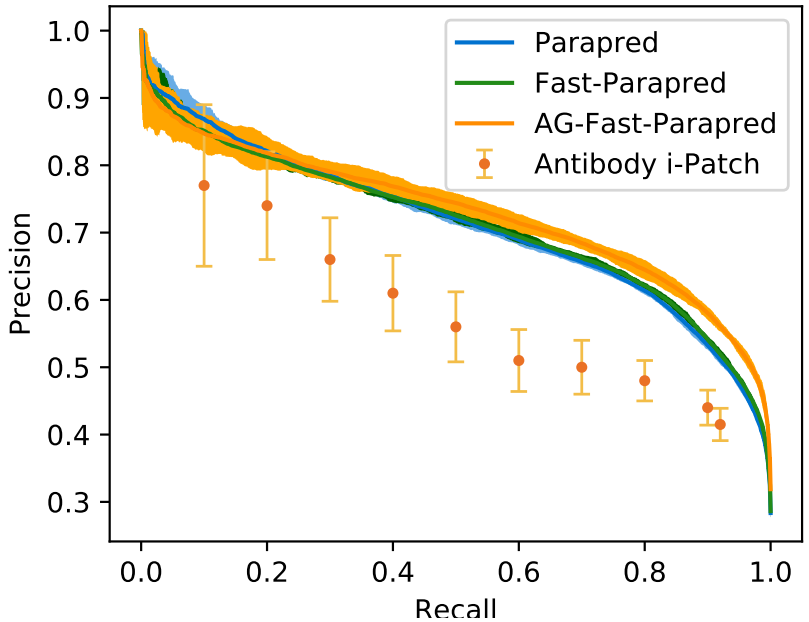
- **Input**:
  - antibody computed residue features $\mathbf{b} = \{\vec{b}_1, \vec{b}_2, ...\vec{b}_k\}$, $\vec{b}_i \in \mathbb{R}^N$
  - antigen computed residue features $\mathbf{g} = \{\vec{g}_1, \vec{g}_2, ...\vec{g}_l\}$, $\vec{g}_j \in \mathbb{R}^M$
  - for each $\vec{b}_i$ a set of neighbouring residues ν_i

- The attention coefficients are then computed using the shared attentional mechanism a: $\mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$ and a new feature vector is obtained:

$$\vec{b}' = \sigma\left(\sum_{j \in \nu_i} a(\vec{b}_i, \vec{g}_j)\vec{g}_j\right)$$

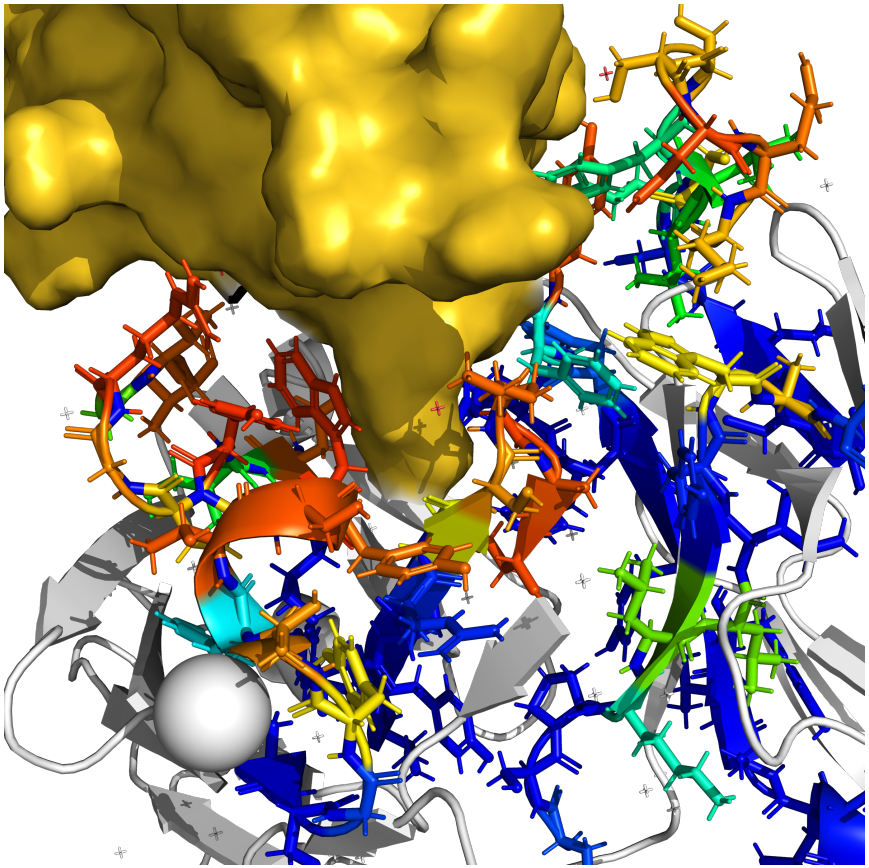
Quantitative results

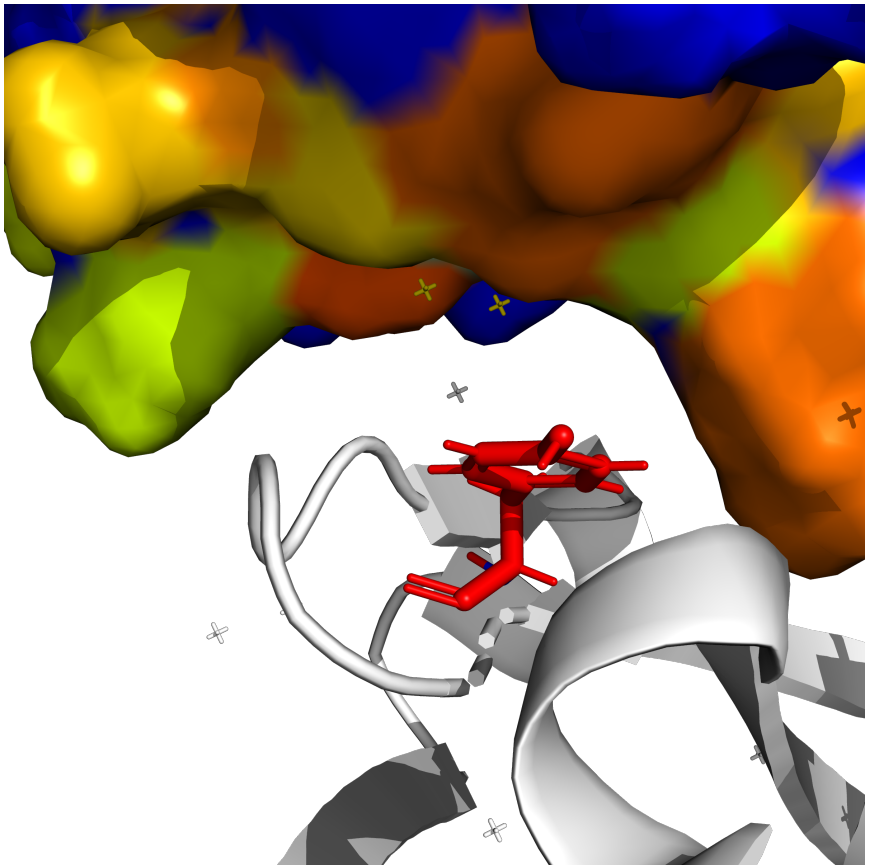
	ROC AUC
ProABC	0.851
Parapred	0.880 ± 0.002
Fast-Parapred	0.883 ± 0.001
AG-Fast-Parapred	$\mathbf{0.899} \pm 0.004$



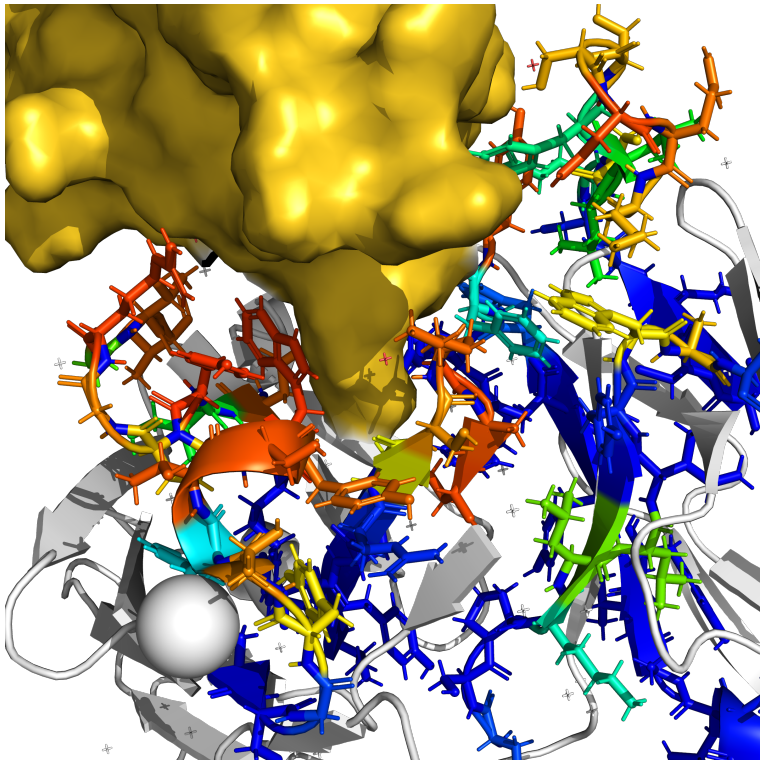
95% confidence intervals, after 10 runs of 10-fold crossvalidation.

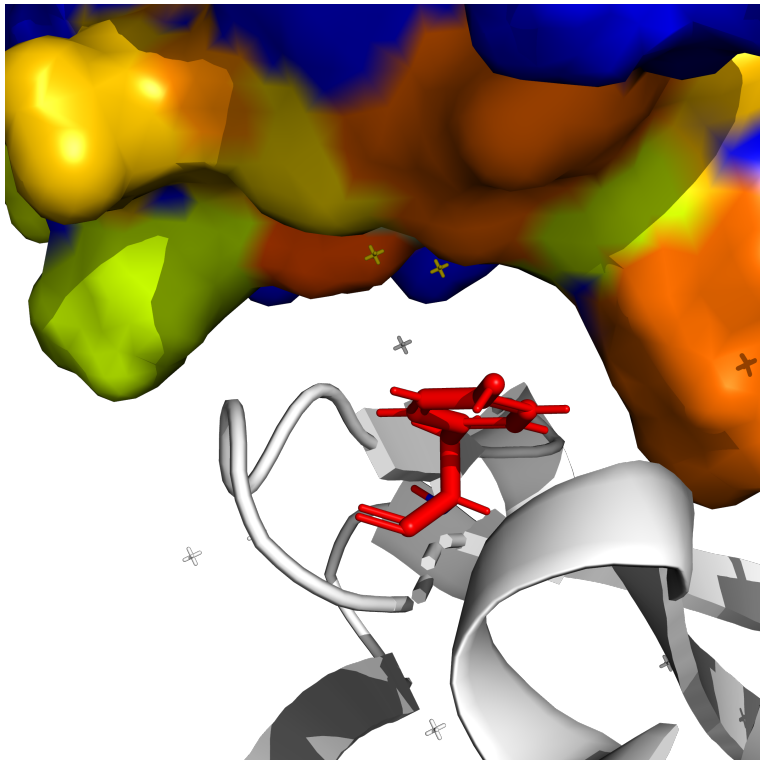
Qualitative results





Qualitative results





- The model learns the antibody/antigen geometry without being given any positional information.
- This could enable us to build an epitope predictor!

Thank you!

Questions?

petar.velickovic@cst.cam.ac.uk

http://www.cst.cam.ac.uk/∼pv273/

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