GRAPH ATTENTION NETWORKS

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio
Department of Computer Science and Technology, University of Cambridge
Centre de Visio per Computador, UAB
Montreal Institute for Learning Algorithms

ICLR 2018

BrAIn Lab. Brain and Artificial Intelligence Lab.

SEOULTECH

Author

Petar Veličković

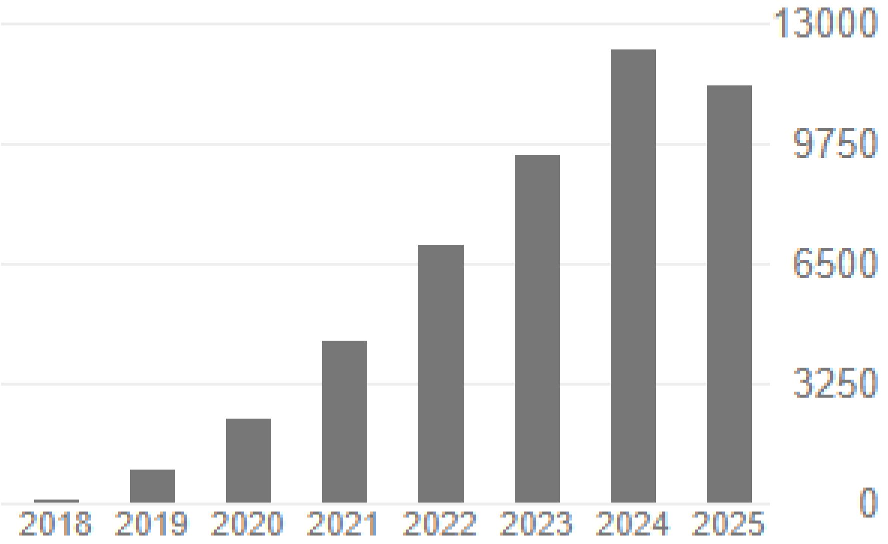
FOLLOW

Senior Staff Research Scientist, Google DeepMind | Affiliated Lecturer, University of Cambridge
Verified email at google.com - Homepage

Geometric Deep Learning   Graph Neural Networks   Categorical Deep Learning
Algorithmic Reasoning

Cited by

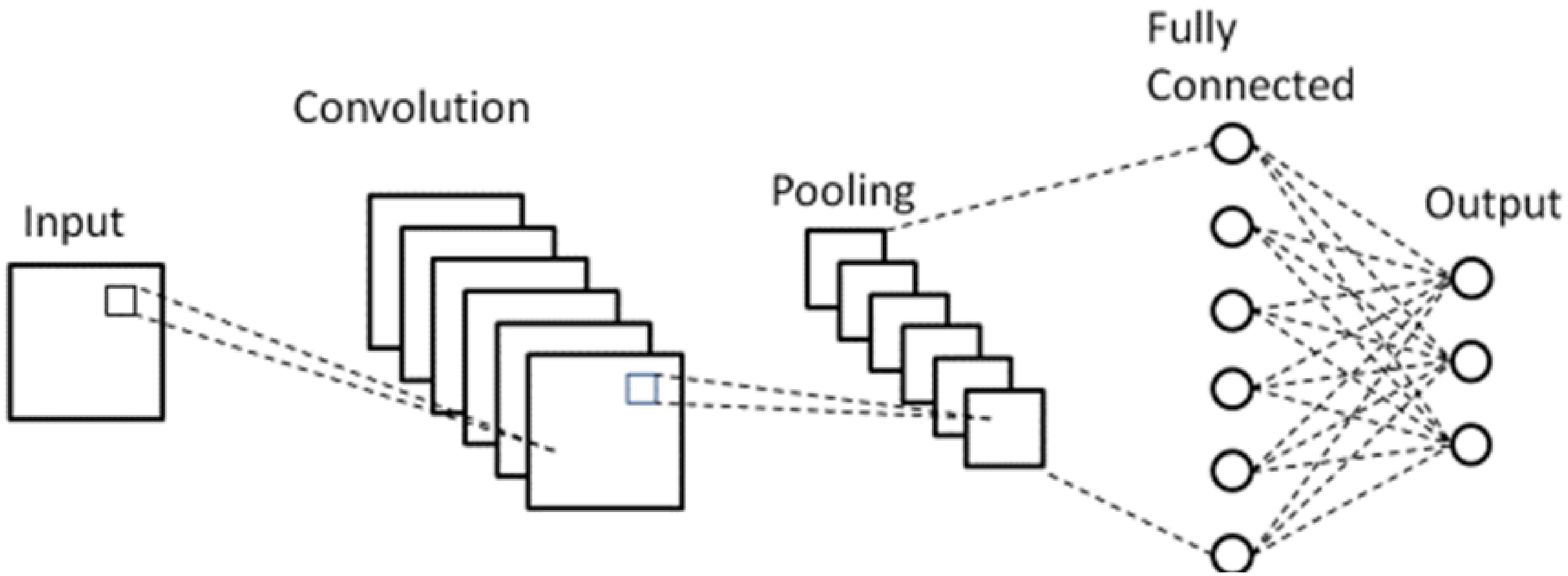
	All	Since 2020
Citations	48060	46837
h-index	40	40
i10-index	73	72



TITLE	CITED BY	YEAR
Graph Attention Networks P Veličković, G Cucurull, A Casanova, A Romero, P Liò, Y Bengio 6th International Conference on Learning Representations (ICLR 2018)	32588 *	2018
Deep Graph Infomax P Veličković, W Fedus, WL Hamilton, P Liò, Y Bengio, RD Hjelm 7th International Conference on Learning Representations (ICLR 2019)	3551	2019
Geometric deep learning: Grids, groups, graphs, geodesics, and gauges MM Bronstein, J Bruna, T Cohen, P Veličković arXiv preprint arXiv:2104.13478	2095	2021
Scientific discovery in the age of artificial intelligence H Wang*, T Fu*, Y Du*, W Gao, K Huang, Z Liu, P Chandak, S Liu, ... Nature 620 (7972), 47-60	1642	2023

Background

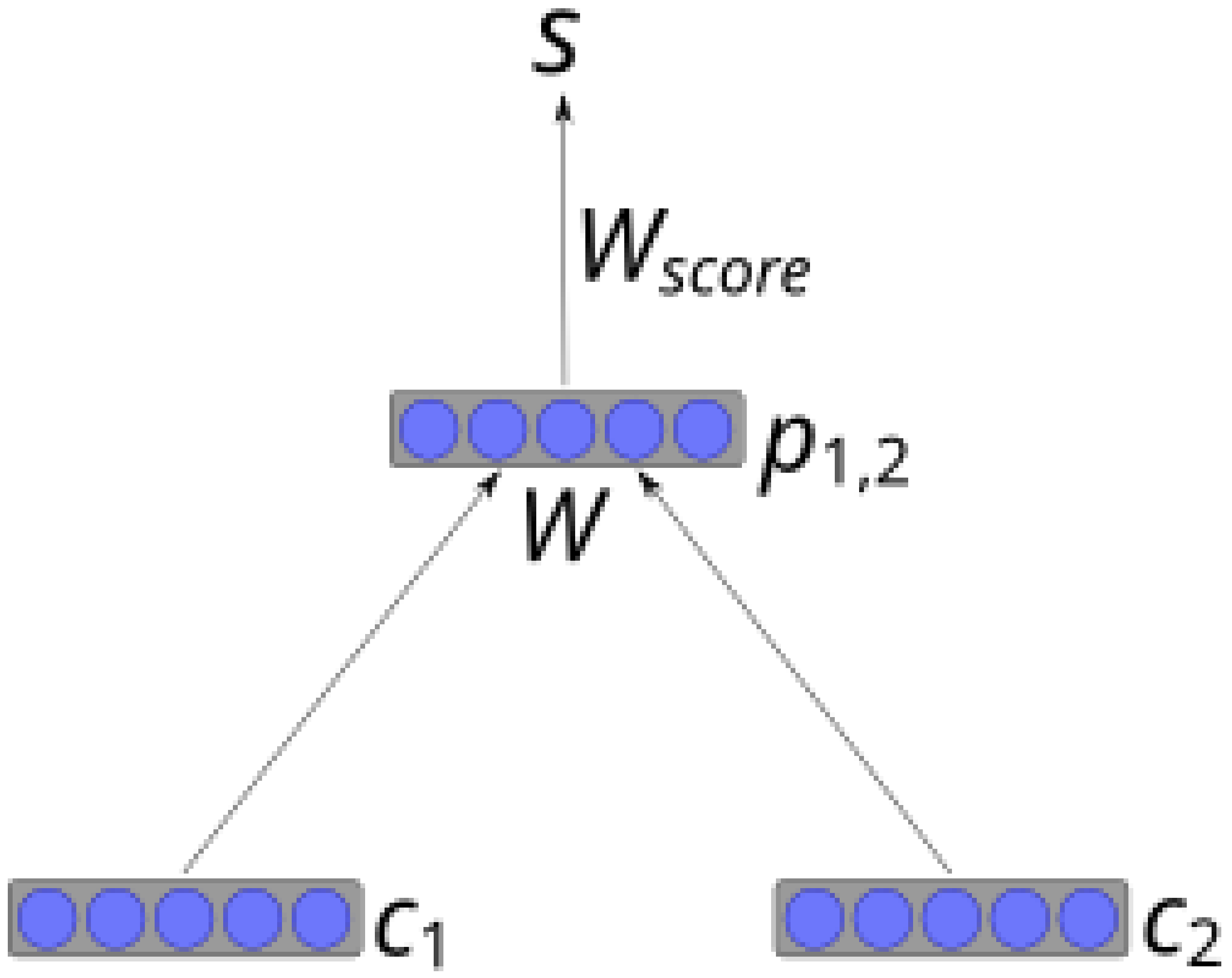
- **Convolutional Neural Networks (CNNs)** have been successfully applied to tackle problems such as image, semantic segmentation or machine translation, where the underlying data representation has a grid-like structure.



- However, many … grid-like structure
  and that instead

Background

- There have been several attempts in the literature to extend neural networks to deal with **arbitrarily structured graphs**.
- Early work used **recursive neural networks** to process data represented in graph domains as directed acyclic graphs.
- A recursive neural network is created by **applying the same set of weights recursively** over a structured input, to produce a structured prediction over variable-size input structures, by traversing a given structure **in topological order**.
- In the simplest architecture, nodes are combined into parents using a weight matrix and a non-linearity such as the tanh hyperbolic function.



$$p_{1,2} = \tanh(W[c_1; c_2])$$

Background

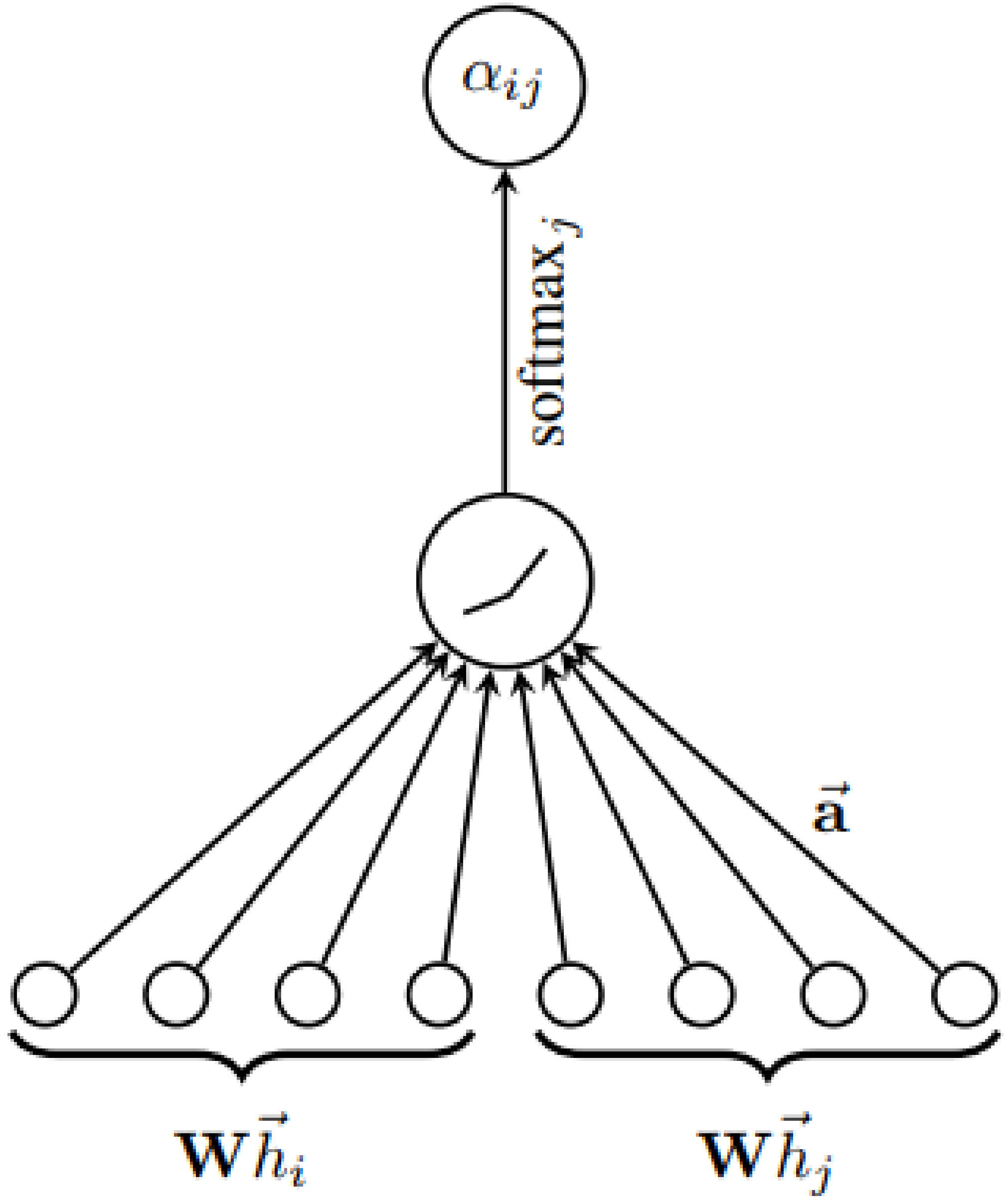
- A **Graph Convolutional Network (GCN)** extends the idea of convolution from regular grids (like images) to graph-structured data.
- Input Representation: Each node v has a feature vector x. The graph is defined by an adjacency matrix A.
- Neighborhood Aggregation (Message Passing): Each node updates its representation by aggregating features from its neighbors (and often itself).

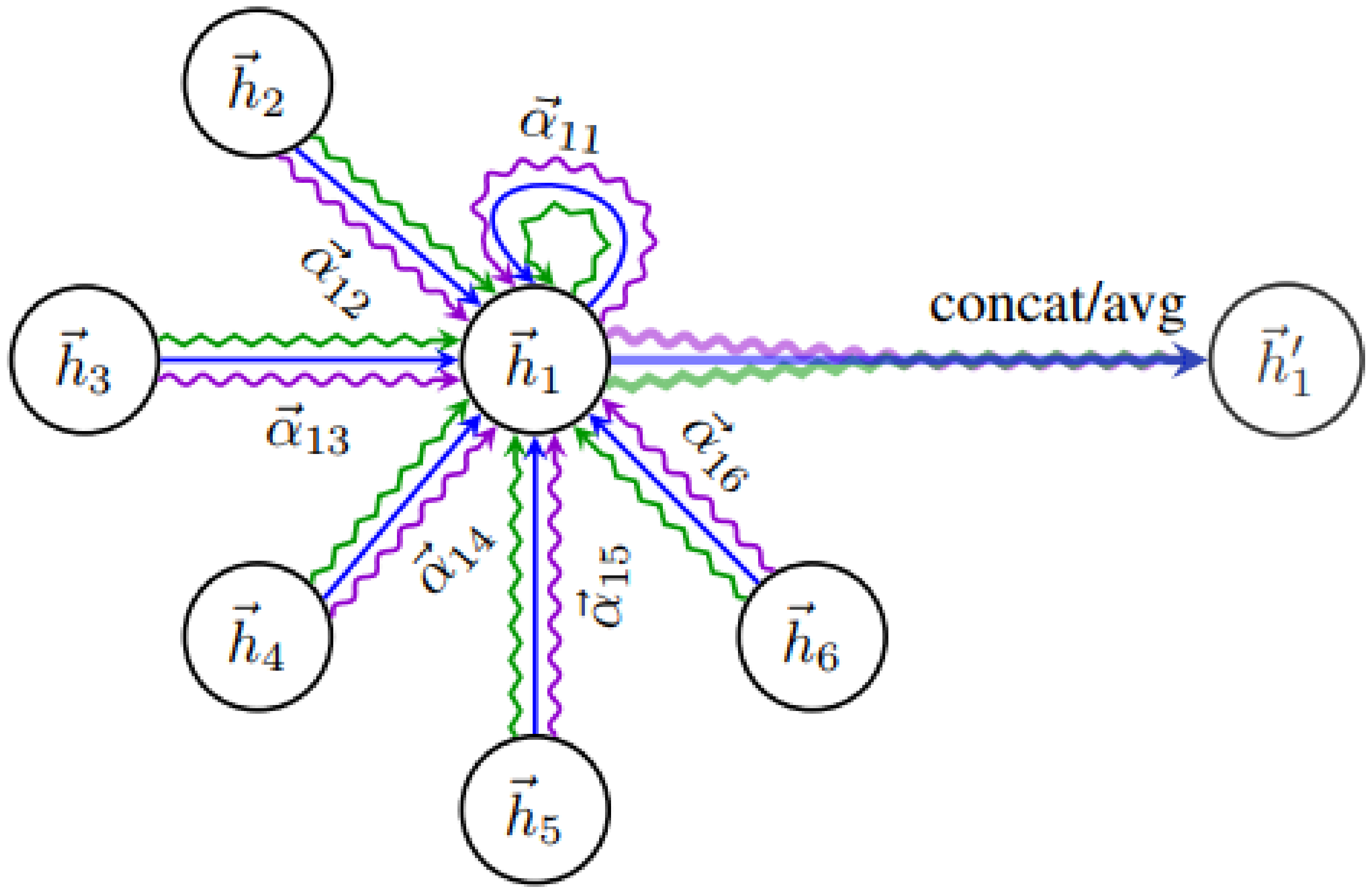
- There is an increasing interest in generalizing convolutions to the graph domain. **However, the learned filters depends** n a specific structure can not be directly applied t

$$H^{(l+1)} = \sigma(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)})$$

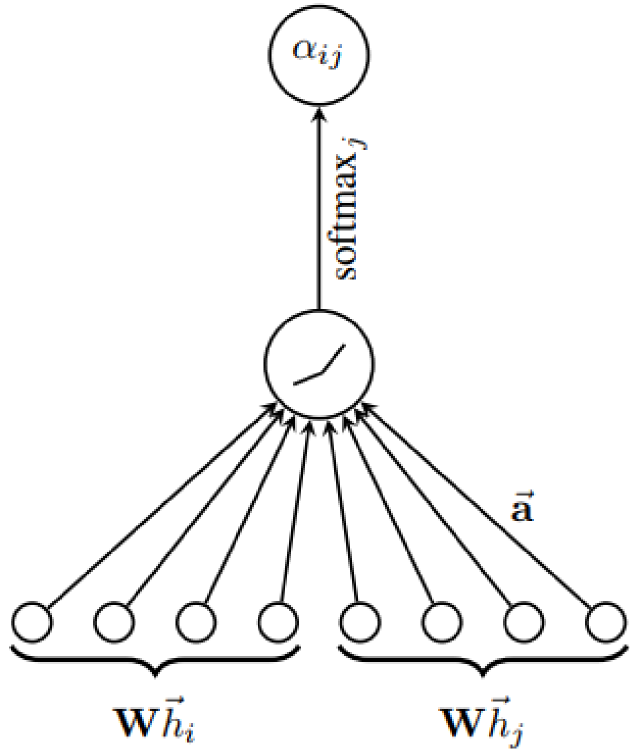
Graph Attention Network

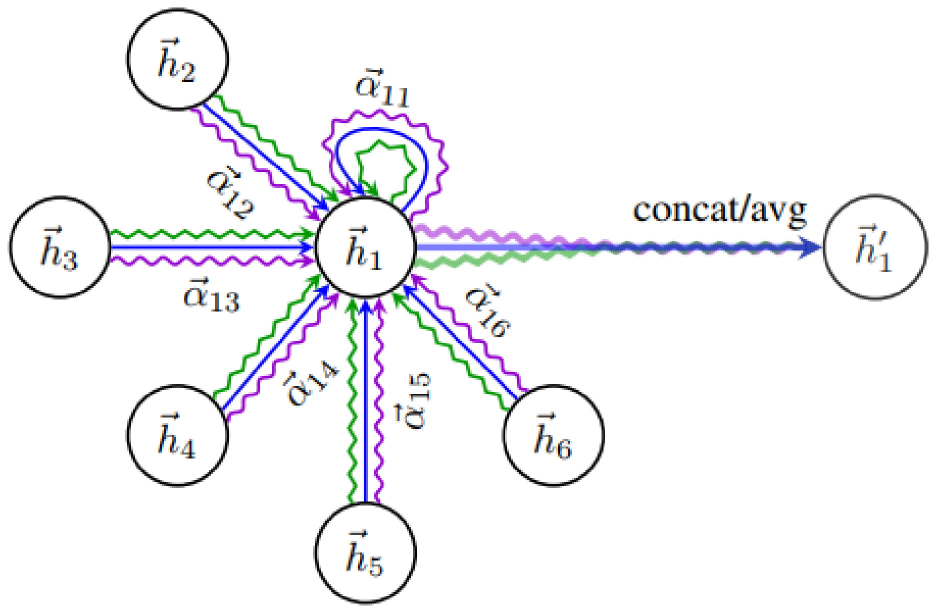
- The input to the layer is a set of **h** nodes with **F** features in each node. The layer produces a new set of node features (of potentially different cardinality **F'**), as its output.
- In order to obtain sufficient expressive power to transform the input features into higher-level features, a shared linear transformation, parametrized by a weight matrix, **W**, is applied to every node.





Graph Attention Network





- Self-attention is then performed on the nodes—a shared attentional mechanism $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ computes attention coefficients:

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j) \qquad \alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}_i}\exp(e_{ik})}$$

$$\alpha_{ij} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\mathrm{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i\|\mathbf{W}\vec{h}_k]\right)\right)} \qquad \vec{h}'_i = \sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\vec{h}_j\right)$$

Transductive Learning

Table 1: Summary of the datasets used in our experiments.

	Cora	Citeseer	Pubmed	PPI
Task	Transductive	Transductive	Transductive	Inductive
# Nodes	2708 (1 graph)	3327 (1 graph)	19717 (1 graph)	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)

- Utilize three standard citation network benchmark datasets—Cora, Citeseer and Pubmed. In all these datasets, nodes correspond to documents and edges to (undirected) citations. Node features correspond to elements of a bag-of-words representation of a document.
- Each node has a class label.
- Following the transductive setup, the training algorithm has access to all of the nodes' feature vectors.

Inductive Learning

- Make use of a protein-protein interaction (PPI) dataset that consists of graphs corresponding to different human tissues.
- The dataset contains 20 graphs for training, 2 for validation and 2 for testing. Critically, testing graphs remain completely unobserved during training.
- The average number of nodes per graph is 2372. Each node has 50 features that are composed of positional gene sets, motif gene sets and immunological signatures.

Experimental Setup

- For the **transductive learning** tasks, a **two-layer GAT** model was applied. The first layer consists of 8 attention heads computing, F', 8 features each (for a total of 64 features), followed by an exponential linear unit (ELU) nonlinearity.
- The second layer is used for classification: a single attention head that computes C features (where C is the number of classes), followed by a softmax activation.
- For the **inductive learning** tasks, a **three-layer GAT** model was applied. Both first 2 layers consist of 4 attention heads computing, F', 256 features each (for a total of 1024 features), followed by an ELU nonlinearity.
- The final layer is used for (multi-label) classification with 6 attention heads computing 121 features each, that are averaged and followed by a logistic sigmoid activation.

Results

Table 2: Summary of results in terms of classification accuracies, for Cora, Citeseer and Pubmed. GCN-64* corresponds to the best GCN result computing 64 hidden features (using ReLU or ELU).

Transductive

Method	**Cora**	**Citeseer**	**Pubmed**
MLP	55.1%	46.5%	71.4%
ManiReg (Belkin et al., 2006)	59.5%	60.1%	70.7%
SemiEmb (Weston et al., 2012)	59.0%	59.6%	71.7%
LP (Zhu et al., 2003)	68.0%	45.3%	63.0%
DeepWalk (Perozzi et al., 2014)	67.2%	43.2%	65.3%
ICA (Lu & Getoor, 2003)	75.1%	69.1%	73.9%
Planetoid (Yang et al., 2016)	75.7%	64.7%	77.2%
Chebyshev (Defferrard et al., 2016)	81.2%	69.8%	74.4%
GCN (Kipf & Welling, 2017)	81.5%	70.3%	**79.0%**
MoNet (Monti et al., 2016)	$81.7 \pm 0.5\%$	—	$78.8 \pm 0.3\%$
GCN-64*	$81.4 \pm 0.5\%$	$70.9 \pm 0.5\%$	$\mathbf{79.0} \pm 0.3\%$
GAT (ours)	$\mathbf{83.0} \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$\mathbf{79.0} \pm 0.3\%$

Results

Table 3: Summary of results in terms of micro-averaged F_1 scores, for the PPI dataset. GraphSAGE* corresponds to the best GraphSAGE result we were able to obtain by just modifying its architecture. Const-GAT corresponds to a model with the same architecture as GAT, but with a constant attention mechanism (assigning same importance to each neighbor; GCN-like inductive operator).

Inductive

Method	**PPI**
Random	0.396
MLP	0.422
GraphSAGE-GCN (Hamilton et al., 2017)	0.500
GraphSAGE-mean (Hamilton et al., 2017)	0.598
GraphSAGE-LSTM (Hamilton et al., 2017)	0.612
GraphSAGE-pool (Hamilton et al., 2017)	0.600
GraphSAGE*	0.768
Const-GAT (ours)	0.934 ± 0.006
GAT (ours)	$\mathbf{0.973} \pm 0.002$

Conclusion

- The graph attention networks (GATs) operate on graph-structured data.
- The graph attentional layer utilized throughout these networks is **computationally efficient**, **allows for assigning different importances to different nodes** within a neighborhood, and does not depend on knowing the entire graph structure upfront.
- Potential improvements: able to handle larger batch sizes, extending the method to perform graph classification instead of node classification, extending the model to incorporate edge features would allow us to tackle a larger variety of problems.