Graph Attention Networks

lecture-8

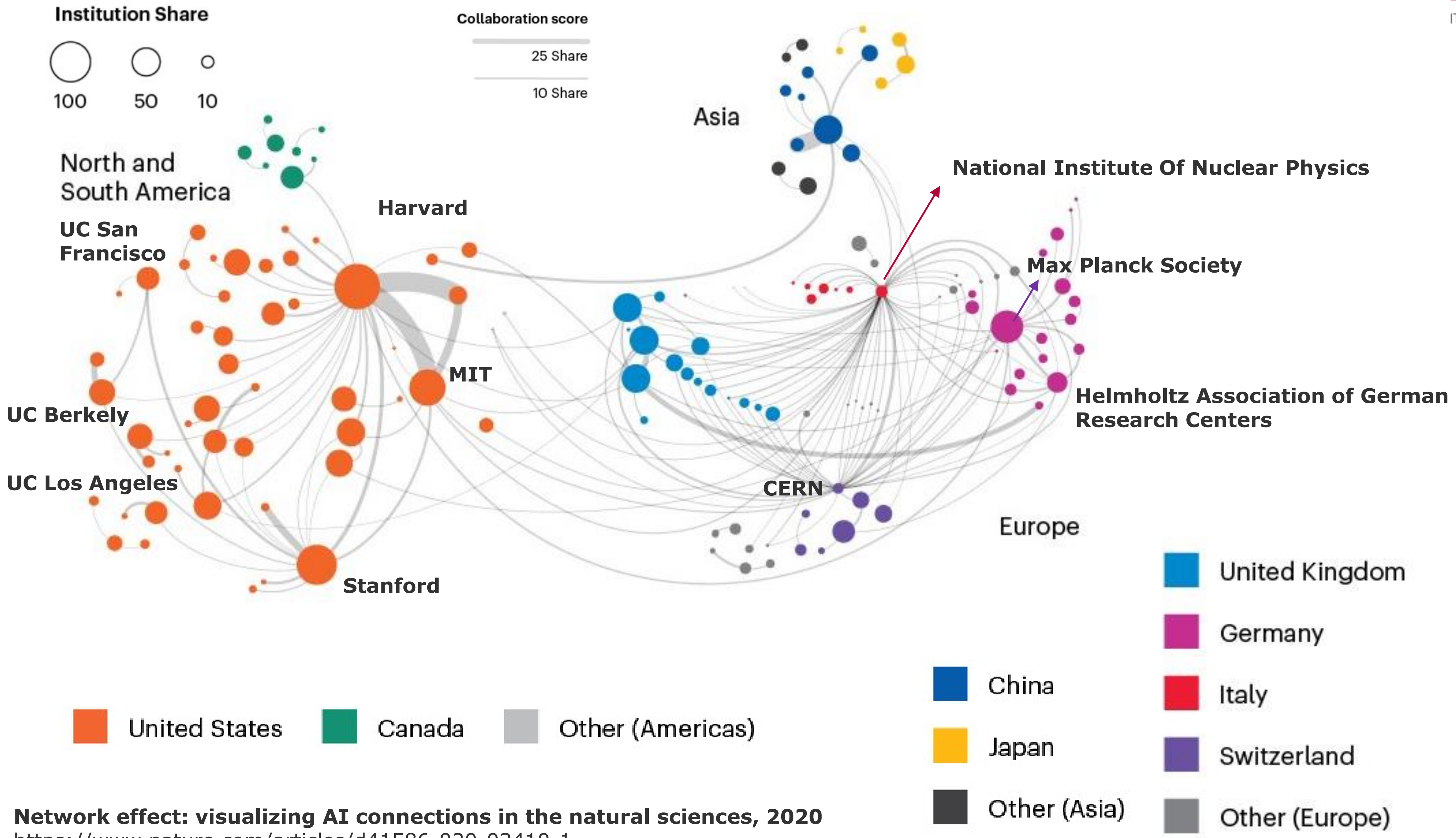
Course on Graph Neural Networks (Winter Term 21/22)

Prof. Dr. Holger Giese (holger.giese@hpi.de)

Christian Medeiros Adriano (christian.adriano@hpi.de) - **"Chris"**

Matthias Barkowski (matthias.barkowski@hpi.de)

Network effect – AI connections in Natural Sciences



Network effect: visualizing AI connections in the natural sciences, 2020
https://www.nature.com/articles/d41586-020-03410-1

Quick recap

1. Metrics
2. Classification
3. Random Walks
4. Node and Graph Embeddings
5. PageRank
6. Graph Structure Learning
7. Graph Convolutional Networks

2014 - Computer Vision allow to highlight important parts of the image that contribute most to the desired output [Bahdanau et al. 2014] [Wang & Tax 2016]

2015 – Aligned Machine Translation (allow decoder to peek at the input of the decoder) [Luong et al. 2015][Xu et a. 2015]

2017 – Language Modeling with Transformer Networks "Attention is all you need" [Vaswani et al. 2017]

Why do we need Graph Attention Networks?
We need a way to scale without having to make large random walks in the graph

The "Attention" approach allows us to:
- Decide which parts of the input we should focus most on
- Better allocate the limited processing resources to the most important parts

Attention history

[Bahdanau et al 2014] allowed their RNN model give particular attention to certain hidden states when decoding each word. This way they allowed the model to learn which words to give attention to and which ones to ignore while translating of each word at the decoder.

[Bahdanau et al 2014] Bahdanau, D., et al. 2014, "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473.

Topics

1. Introduction to Attention Models

2. Graph Attention Models (GAT)

3. Attention Models in Transformer Networks (Self-Attention)

Overview of Attention Models

Intuition for the need of attention

Take these two sentences from the **Winograd Schema Challenge** set:

1- The city councilmen refused the demonstrators a permit because **they** <u>feared</u> violence.

2- The city councilmen refused the demonstrators a permit because **they** <u>advocated</u> violence.

in 1, "they" is councilmen, while in 2, "they" is demonstrators.

Attention allows to determine if "they" is **councilmen** or **demonstrators**.

In the machine translation case, we call it self-attention.

[1] https://en.wikipedia.org/wiki/Winograd_Schema_Challenge

Motivation for Transformer Networks

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Challenges with Recurrent Neural Networks

Long range dependencies

Vanishing gradient

Gradient explosion

Large number of training steps

Recurrence prevent parallel computation

Advantages of Transformer Networks

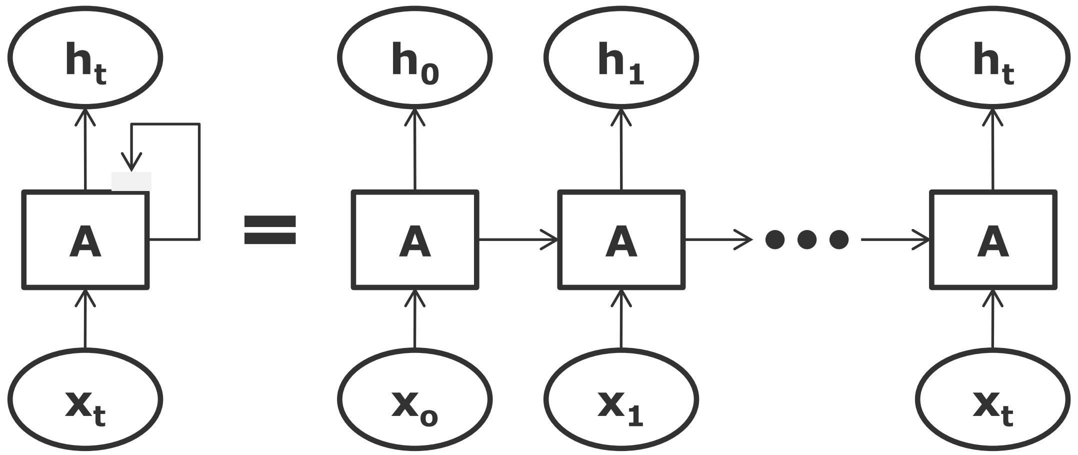
Facilitates long range dependencies

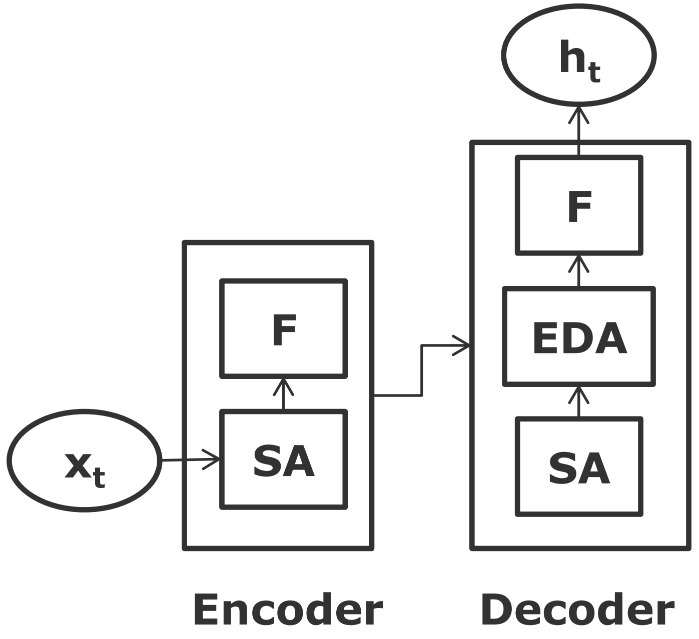
No Vanishing gradient

No Gradient explosion

Fewer number of training steps

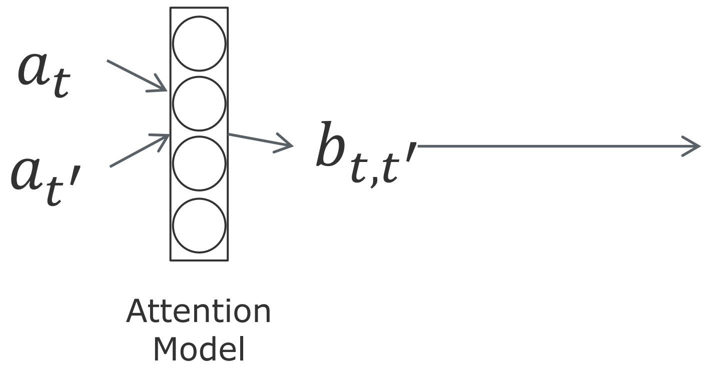
No Recurrence, so facilitates parallel computation





Attention

What is the amount of attention $a_{t,t'}$ that the learning should have on pairs (or groups) of the input data $a_{t'}$?



$$a_{t,t'} = \frac{\exp(b_{t,t'})}{\sum_{t'=1} \exp(b_{t,t'})}$$

The attention mechanism will allow me to decide which parts of the input the learning algorithm will should focus next.

Example

Take three vectors (Query, Key, and Value) that embeddings of sentences

Example:

Query: S_j (a hidden vector for ith output word)

Key= h_j (a hidden vector for jth input word)

Value: h_j (a hidden vector for the jth input word)

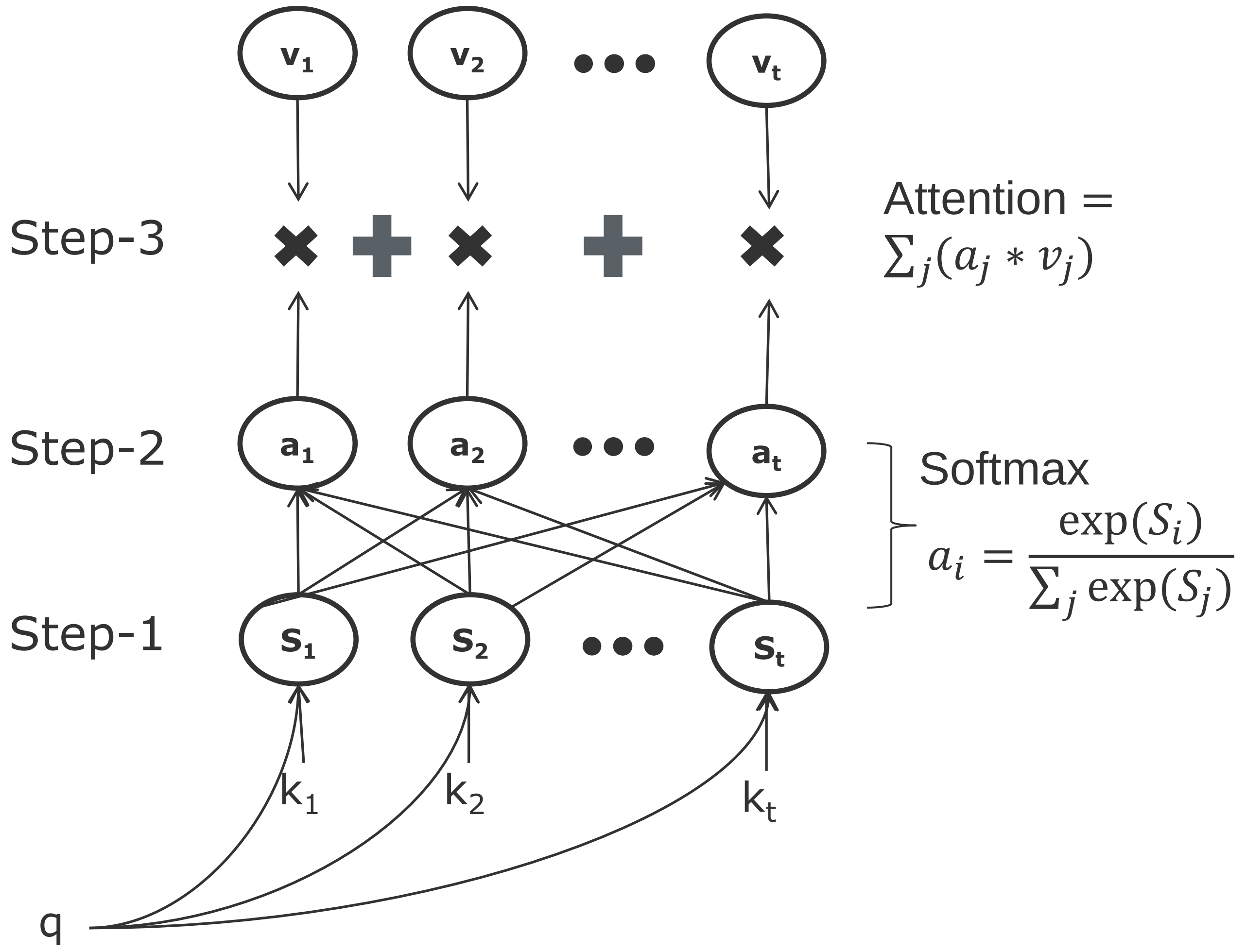
Translation

The attention mechanism will allow me to decide which are the words that I am interested in translating next.

Question&Answers

The attention mechanism allows to decide which answers should be match against a question

Attention Mechanism steps



Attention = $\sum_j (a_j * v_j)$

$$a_i = \frac{\exp(S_i)}{\sum_j \exp(S_j)}$$

The Query, Keys, and Values are vectors.

Example:

Query, keys, and values are embeddings

Query: S_j (a hidden vector for ith output word)

Key= k_j (a hidden vector for jth input word)

Value: h_j (a hidden vector for the jth input word)

The attention mechanism will allow me to decide which are the words that I am interested in translating next.

Attention

Attention is equivalent to the retrieval of a value v_i for a query q based on a key k_i in database.

$$attention\,(v_i, K_i, q) = \sum_i similarity(q, k_i) * v_i$$

Types of similarity functions

- Dot product: $q^T k_i$
- Scaled dot Product: $\frac{q^T k_i}{\sqrt{d}}$, where *d* is the dimensionality of each key
- General Dot Product: $q^T W\, k_i$, where *W* is a set of weights learned that allows to map query to a new space where similarity can be computed
- Additive Similarity: $W_q^T q + W_k^T\, k_i$

Attention in Graph Neural Networks

Recap GCN and GraphSage

Graph Convolutional Network (GCN) [Kipf & Welling 2017] combine local graph structure and node-level features to produce good performance on node classification tasks.

However, the way GCN aggregates information is structure-dependent, which can hurt its generalizability.

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ij}} W^{(l)} h_j^{(l)}\right)$$

where:

- $N(i)$ is the set of its one-hop neighbors,
- $c_{ij} = \sqrt{|N(i)|}\sqrt{|N(j)|}$ is normalization constant based on graph structure,
  - GraphSAGE [Hamilton, Ying & Leskovec 2017] employs the same update rule but sets $cij = |N(i)|$.
- σ is an activation function (GCN uses ReLU), and
- $W(l)$ is a shared weight matrix for node-wise feature transformation.

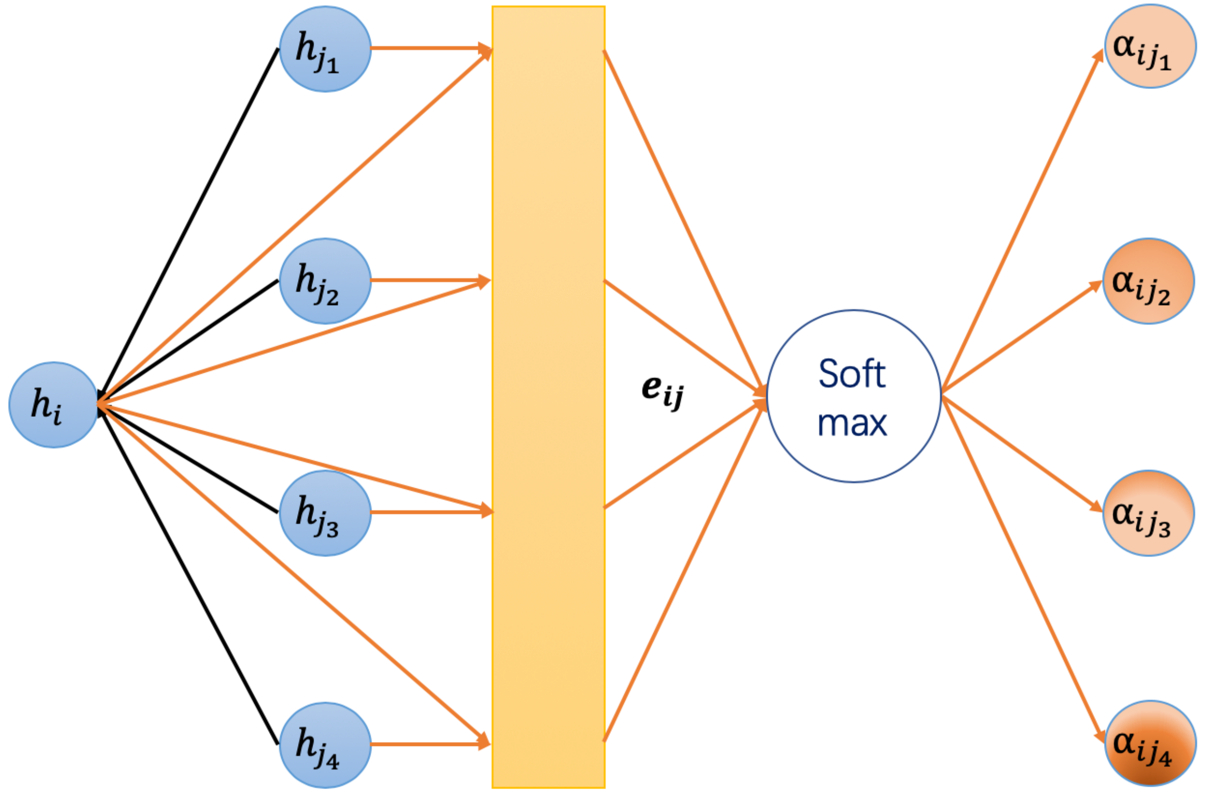
Graph Attention Network (GAT) [Veličković et al. 2018]

Graph Attention Network (GAT) proposes a different type of aggregation.

GAT uses the attention mechanism as a substitute for the statically normalized convolution operation.

weighting neighbor features + feature dependent and structure-free normalization

node embedding $h_i^{(l+1)}$ of layer $l+1$ from the embeddings of layer l.

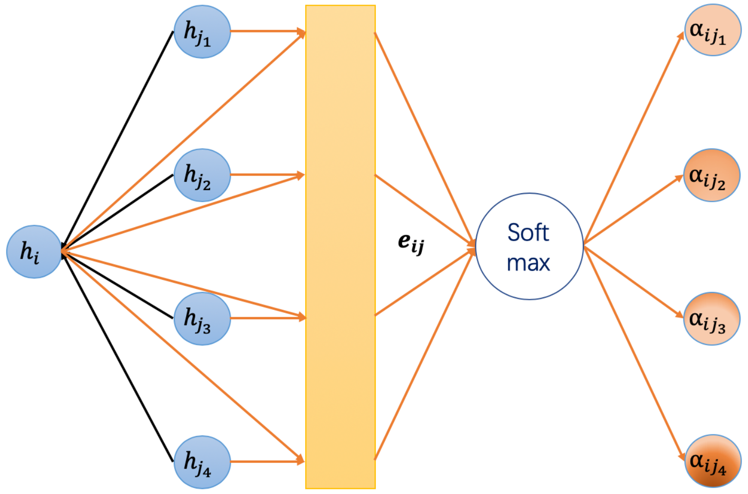


Attention weights

Source: https://docs.dgl.ai/en/0.4.x/tutorials/models/1_gnn/9_gat.html

Equations in GAT [Veličković et al. 2018]

Node embedding $h_i^{(l+1)}$ of layer $l+1$ from the embeddings of layer l.



$$z_i^{(l)} = W^{(l)} h_i^{(l)}$$

linear transformation of the lower layer embedding $h_i^{(l)}$ and $W^{(l)}$ is its learnable weight matrix.

$$\underbrace{e_{ij}^{(l)}}_{} = \text{LeakyReLU}\left(\vec{a}^{(l)^T}\left(z_i^{(l)} || z_j^{(l)}\right)\right)$$

"Additive attention" computes a pair-wise un-normalized attention score between two neighbors i,j.
1- concatenates the z embeddings of the two nodes (|| denotes concatenation),
2- takes a dot product of it and a learnable weight vector a⃗ (l).
3- applies LeakyReLU

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

Softmax normalizes the attention scores of each node incoming edge

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right)$$

Like GCN, the embeddings from neighbors are aggregated and scaled by the attention scores.

Source: https://docs.dgl.ai/en/0.4.x/tutorials/models/1_gnn/9_gat.html

GAT [Veličković et al. 2018]

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)	$\mathbf{83.0} \pm 0.7\%$	$\mathbf{72.5} \pm 0.7\%$	$\mathbf{79.0} \pm 0.3\%$

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)	$\mathbf{0.973} \pm 0.002$

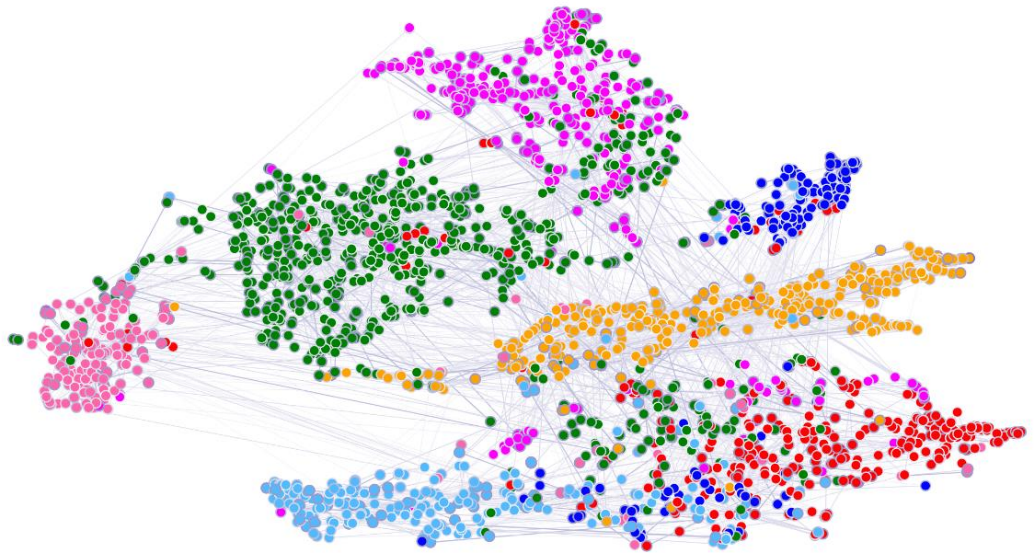


Figure 2: A t-SNE plot of the computed feature representations of a pre-trained GAT model's first hidden layer on the Cora dataset. Node colors denote classes. Edge thickness indicates aggregated normalized attention coefficients between nodes i and j, across all eight attention heads ($\sum_{k=1}^{K} \alpha_{ij}^k + \alpha_{ji}^k$).

Examples of source code

DGL library based on PyTorch

- https://docs.dgl.ai/en/0.4.x/tutorials/models/1_gnn/9_gat.html

Implementations with TensorFlow

- https://github.com/PetarV-/GAT/blob/master/models/base_gattn.py
- https://github.com/Diego999/pyGAT/blob/master/layers.py

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Attention in Transformer Networks

Motivation for Transformer Networks

Performance: Faster, More Scalable, more Interpretable

- Unlike RNN, training can be completely parallelized across sequence timesteps
- Easier to train than LSTMs (RNN) does not suffer from Gradient Vanishing or Explosion
- Transfer Learning works (pre-trained models can be fine-tune for new tasks)
- Can be trained on unsupervised text

How it achieves it? It is Attention to the extreme

Examples of SotA in sequence-related tasks:

- BERT
- GPT, GPT2, GPT3
- TranformerTransfo
- Transformer-XL

Why is it important? Foundation for many pioneering work:

- Image Transformer: increase the size of images that can be processed (https://arxiv.org/abs/1802.05751)
- Self-Attention CycleGAN: Unpaired Image-to-Image Translation (https://junyanz.github.io/CycleGAN/)
- AlphaStar: Grandmaster level in StarCraft II using multi-agent reinforcement learning (https://deepmind.com/blog/article/AlphaStar-Grandmaster-level-in-StarCraft-II-using-multi-agent-reinforcement-learning)

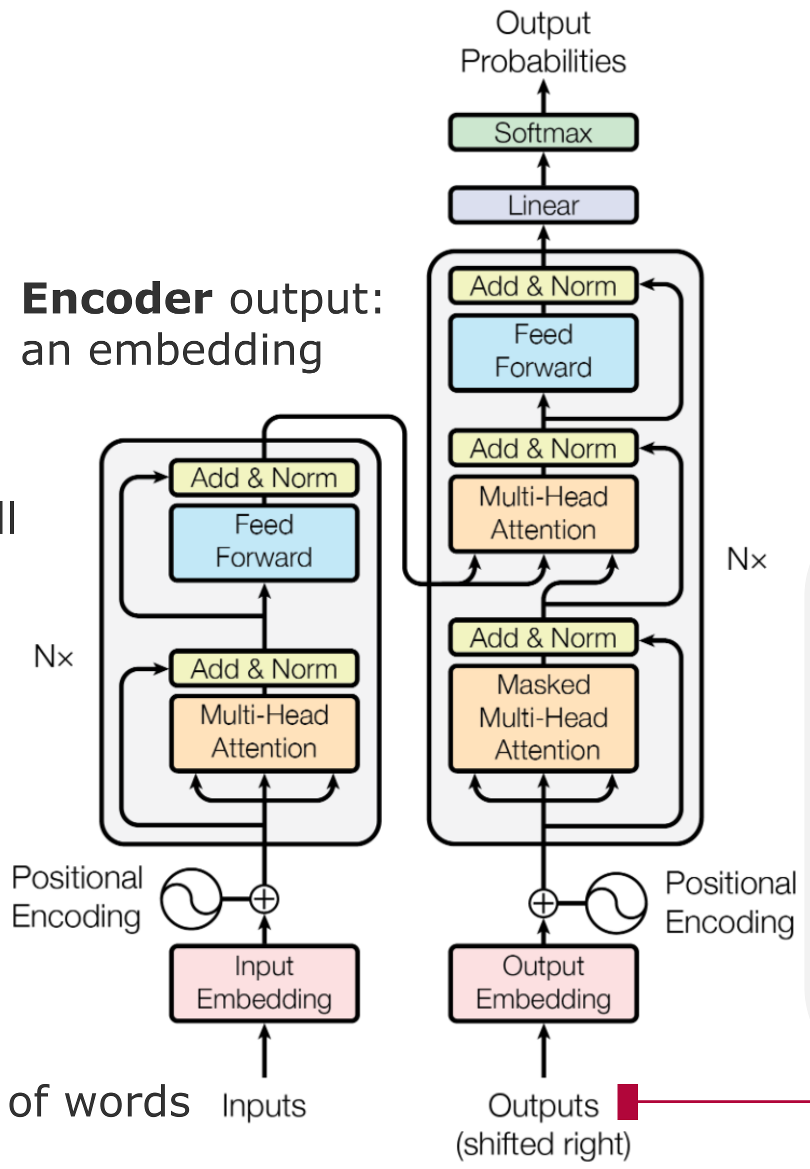
Transformer network – “Attention is all you Need” [Vaswani et al. 2017]

Nx This is repeated N times to combine larger groups: pairs, pairs of pairs, and so forth.

Multi-head Attention
Feed sub-vectors of words compute the attention among all positions, i.e., each word is a query and each other word is a key.

Position encoding
captures the sequence of the input. It uses a sin and cosine to have a continuous encoding [2]

Encoder output:
an embedding



Decoder output:
For each word produces a distribution over the words in the dictionary

Second multi-head attention allows to map each position in the output to the position in the input

Masked multi-head(K) attention allows to focus only on the previous words (mask future words)

$$\text{concatenation} : h_i^{(l+1)} = ||_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k W^k h_j^{(l)}\right)$$

$$\text{average} : h_i^{(l+1)} = \sigma\left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k W^k h_j^{(l)}\right)$$

Feeds entire sequence of words

Partially translated sentence is fed to the bottom of decoder at each decoding step [1]

[1]http://jalammar.github.io/images/t/transformer_decoding_2.gif
[2]https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

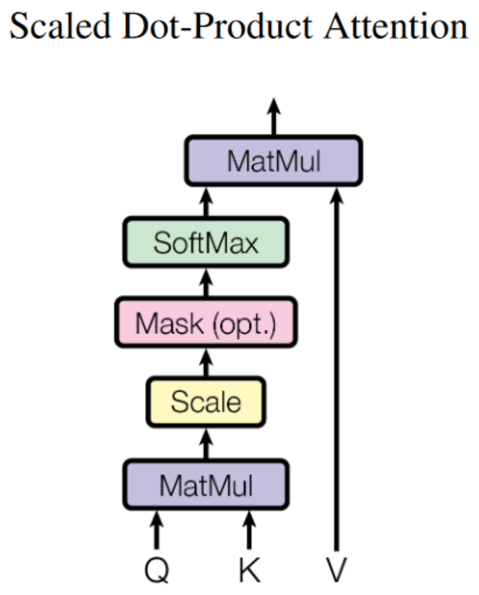
The Attention Function

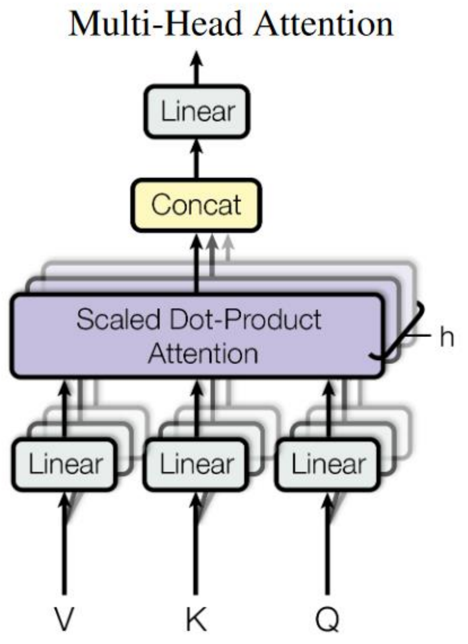
An attention function is a mapping of a query and a set of key-value pairs to an output.

The Query, Keys, and Values are vectors.

The output is computed as a weighted sum of the value.

The Weight assigned to each value is computed by a compatibility function of the query with the corresponding key.





Next and Future Tasks

1. Compute and compare graph metrics (Wednesday, 2.12)
   - Any metrics and networks of your choice
2. First draft of **abstract** (Friday, 4.12)
3. Predictions using traditional method (Wednesday, 9.12)
   - Any two methods of your choice
4. Related work draft (Friday, 11.12)
5. Node and Graph Feature Learning (Wednesday, 16.12)
   - Any two methods of your choice
6. Design alternative pipelines for your GNN (Wednesday, 06.01)
   - Three alternative with different options for embedding, aggregation, and enconding
   - Test at least one.

END