CRIPAC
智能感知与计算研究中心
Center for Research on Intelligent Perception and Computing

中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2022

"Deep Learning Lecture"

# Lecture 11: Graph Neural Networks

Yan Huang

Center for Research on Intelligent Perception and Computing (CRIPAC)

National Laboratory of Pattern Recognition (NLPR)

Institute of Automation, Chinese Academy of Science (CASIA)

# 通知

- **大作业提交：**
  - 提交电子版：截止日期为5月22日（第13次课前的星期日）23:59，届时也将会在sep系统上布置并通知大家
  - 将所有文件（报告、代码、说明文档等）打包为一个压缩文件（rar/zip），命名为“深度学习大作业2022+组长姓名+组长学号”
- **小组汇报：**5月24日（第13次课）上课时间
  - 想要申请汇报的小组在5月15日（周日）23:59 前在课程群内填写链接报名，提前准备slides
  - 我们会根据报名情况协调安排顺序，如报名不足，将会随机抽取，具体需要汇报的小组名单将在大家报名结束后尽快整理并公布
- **期末考评：**5月31日 18:00-20:00 闭卷

# 课程大作业要求

课程设计作业评价规则:

1. 研究意义及动机是否明确?
2. 方案是否可行有效?
3. 整个解决方案是否完整，是否存在缺陷?
4. 是否具备创新性?
5. 技术报告是否逻辑清晰，叙述准确?

# 课程大作业要求

课程设计作业技术报告要求4-8页，提交电子版；原则上要求按照如下结构完成，英文中文均可，鼓励使用英文。

- Introduction - Motivation
- Problem definition
- Proposed method
  - Intuition - why should it be better than other methods?
  - Description of its algorithms
- Experiments
  - Description of your testbed; list of questions your experiments are designed to answer
  - Details of the experiments; observations
- Conclusions

# 课程大作业要求

- 技术报告推荐格式：CVPR的论文格式，地址如下:

https://cvpr2022.thecvf.com/author-guidelines

- 鼓励提交课程设计代码，要求给出详细的readme.txt代码说明文档，给出其参数配置及运行方法；

# Outline

1. Course Review
2. Graph Neural Networks
3. GNNs for Classical Problems
4. Applications

# Course Review: Attention

Everyone knows what attention is. It is the taking possession by the mind, in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought. Focalization, concentration, of consciousness are of its essence. It implies withdrawal from some things in order to deal effectively with others.

--- Williams James

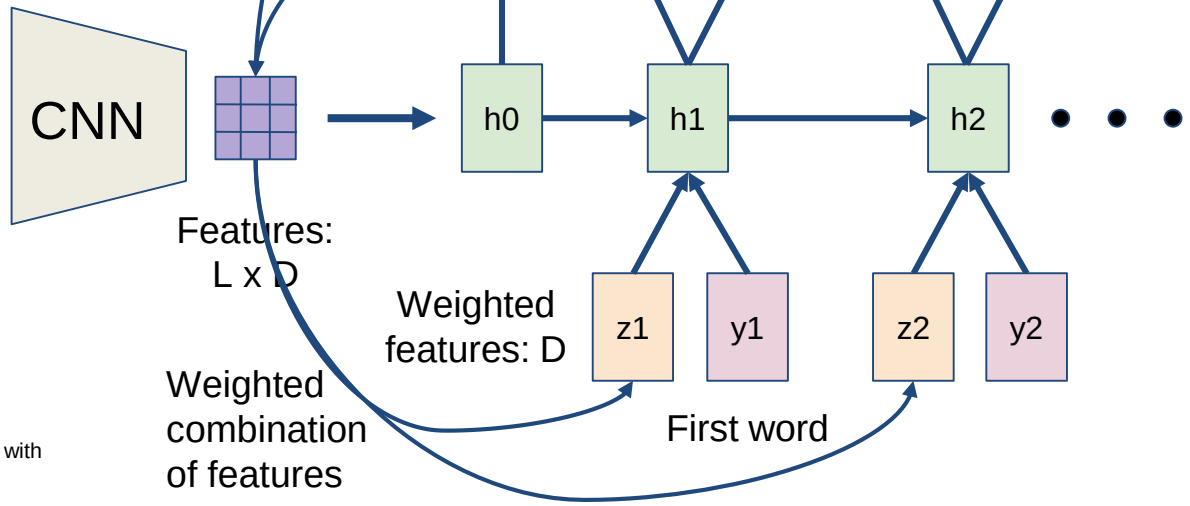
注意在人眼视觉信息处理过程中体现为信息的选择与过滤，从而减少向高级皮层传递的信息，避免冗余或者噪声信息对高级视觉任务的干扰，这种选择过程发生在视觉通道的多个阶段，而且这种选择过程既有各阶段视觉特征的刺激，也有高级皮层向下的反馈调节，对选择性注意的研究不仅有理论上的意义，而且有现实的迫切需求

# Course Review: Soft Attention for Captioning

Guess which framework was used to implement?

Crazy RNN = **Theano**

Image:
H x W x 3




Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Course Review: Soft vs Hard Attention

Image:
H x W x 3

CNN

| | |
|---|---|
| a | b |
| c | d |

Grid of features
(Each D-dimensional)

Context vector z
(D-dimensional)

From
RNN:

| | |
|---|---|
| $p_a$ | $p_b$ |
| $p_c$ | $p_d$ |

Distribution over grid locations
$p_a + p_b + p_c + p_c = 1$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

**Soft attention:**
Summarize ALL locations
$z = p_a a + p_b b + p_c c + p_d d$

Derivative dz/dp is nice!
Train with gradient descent

**Hard attention**:
Sample ONE location according to p, z = that vector

With argmax, dz/dp is zero almost everywhere …
Can't use gradient descent; need reinforcement learning

# Course Review: Attention Recap

- Soft attention:
  - Easy to implement: produce distribution over input locations, reweight features and feed as input
  - Attend to arbitrary input locations using spatial transformer networks

- Hard attention:
  - Attend to a single input location
  - Can’t use gradient descent!
  - Need **reinforcement learning!**

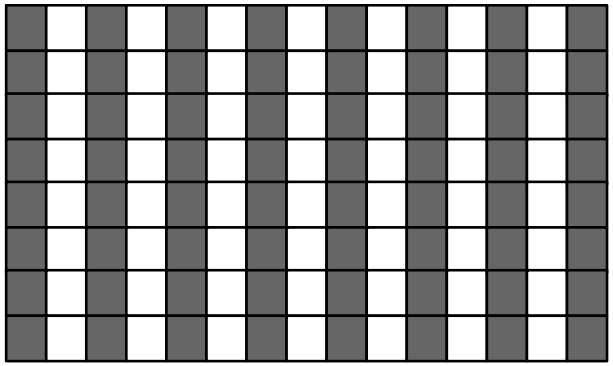
# Course Review: ML Models Need Memory

Deeper AI tasks require explicit memory and multi-hop reasoning over it

- RNNs have short memory
- Cannot increase memory without increasing number of parameters
- Need for compartmentalized memory
- Read/Write should be asynchronous

# Course Review: Memory Networks

- Class of Models with memory $m$ - Array of objects $m_i$



$m_i$

Each memory here is a dense vector

Four Components :

**I - Input Feature Map :** Input manipulation
**G - Generalization :** Memory Manipulation
**O - Output Feature Map :** Output representation generator
**R - Response :** Response Generator

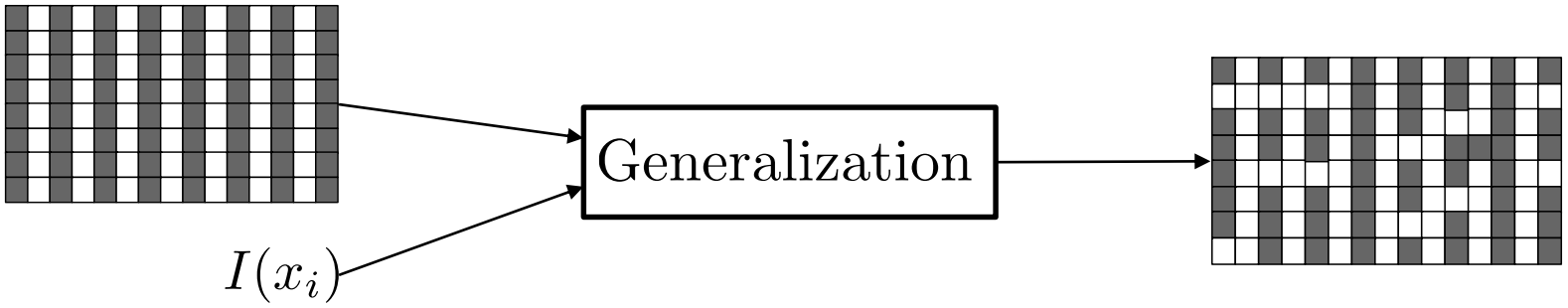
Memory Networks, Weston et. al., ICLR 2015

# Course Review: Memory Networks

## 1. Input Feature Map

- Imagine input as a sequence of sentences $x_i$


## 2. Update Memories

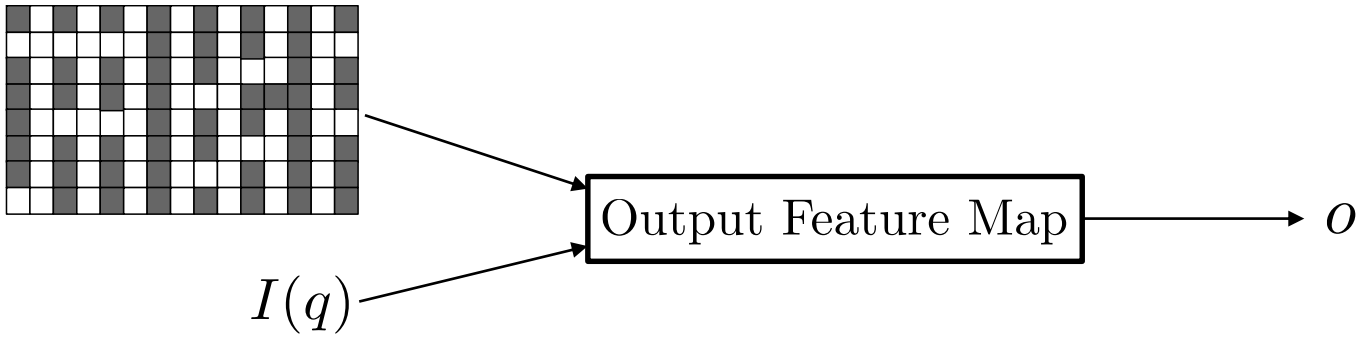



Memory Networks, Weston et. al., ICLR 2015

# Course Review: Memory Networks

## 3. Output Representation

- Say if $q$ is a question, compute output representation




## 4. Generate Answer Response


Memory Networks, Weston et. al., ICLR 2015

# Course Review: Neural Turing Machine

## Copy Task: Implement the Algorithm

Given a list of numbers at input, reproduce the list at output

Neural Turing Machine Learns:

1. What to write to memory
2. When to write to memory
3. When to stop writing
4. Which memory cell to read from
5. How to convert result of read into final output

# Course Review: Neural Turing Machine

Unrolled Feed-forward Controller

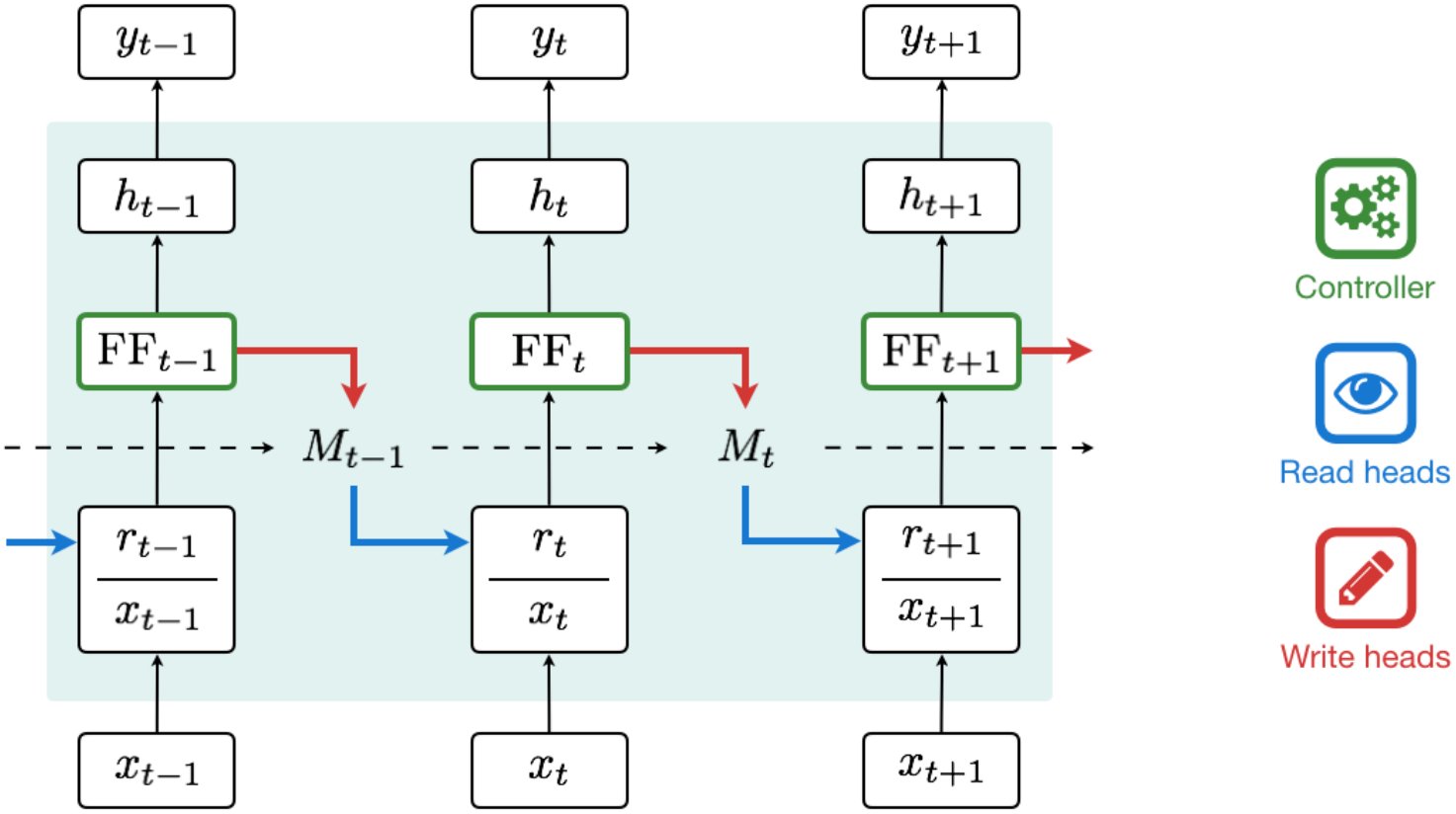



Figure from [Snips AI's Medium Post]

Neural Turing Machines, Graves et. al., arXiv:1410.5401

# Course Review: Summary

- Machine learning models require memory and multi-hop reasoning to perform AI tasks better
- Generic architectures with memory, such as Neural Turing Machine, limited applications shown
- Future directions should be focusing on applying generic neural models with memory to more AI tasks

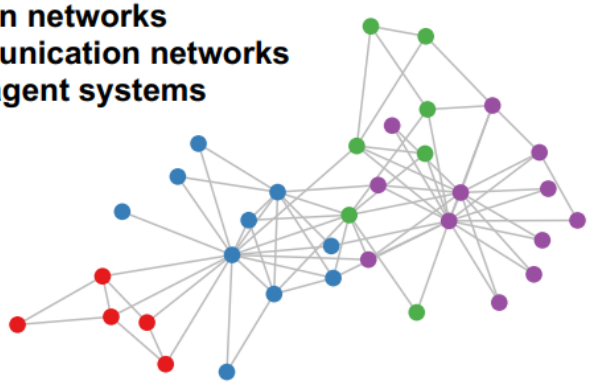
Hybrid computing using a neural network with dynamic external memory, Graves et. al., Nature *vol.* 538

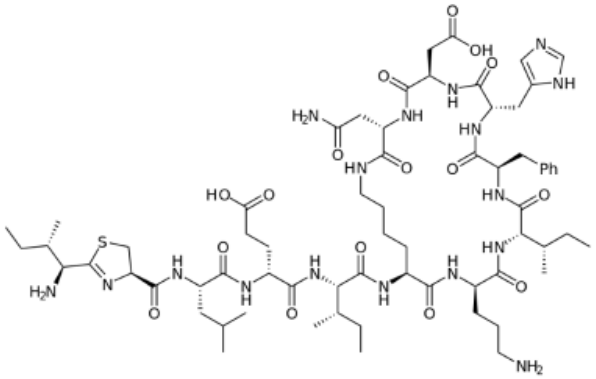
# Outline

1. Course Review
2. Graph Neural Networks
3. GNNs for Classical Problems
4. Applications

# Graph-structured Data

**Social networks**
**Citation networks**
**Communication networks**
**Multi-agent systems**





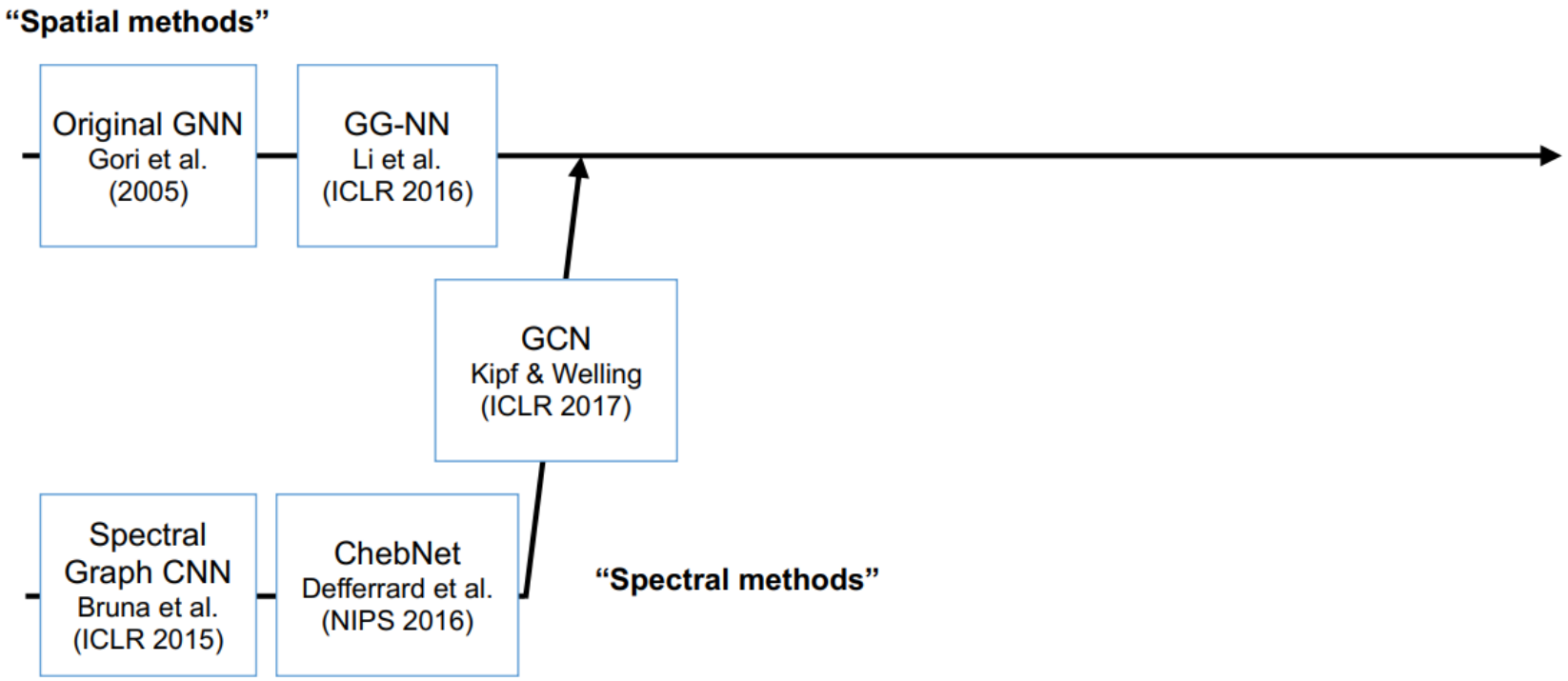
**Molecules**

**Maps**

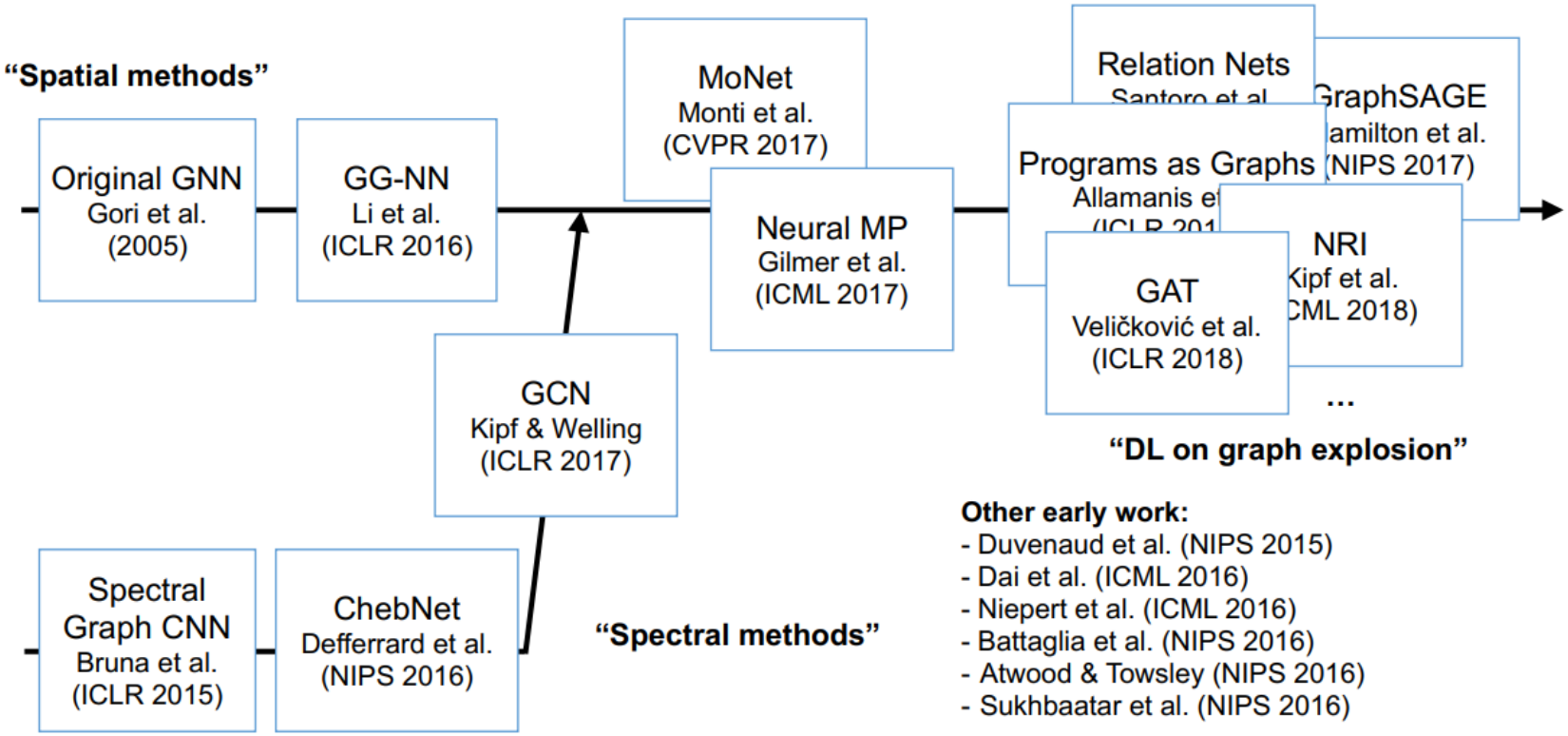

A lot of real-world data does not live on grids

Standard DL architectures like CNNs and RNNs **DO NOT** work here!

# Brief History

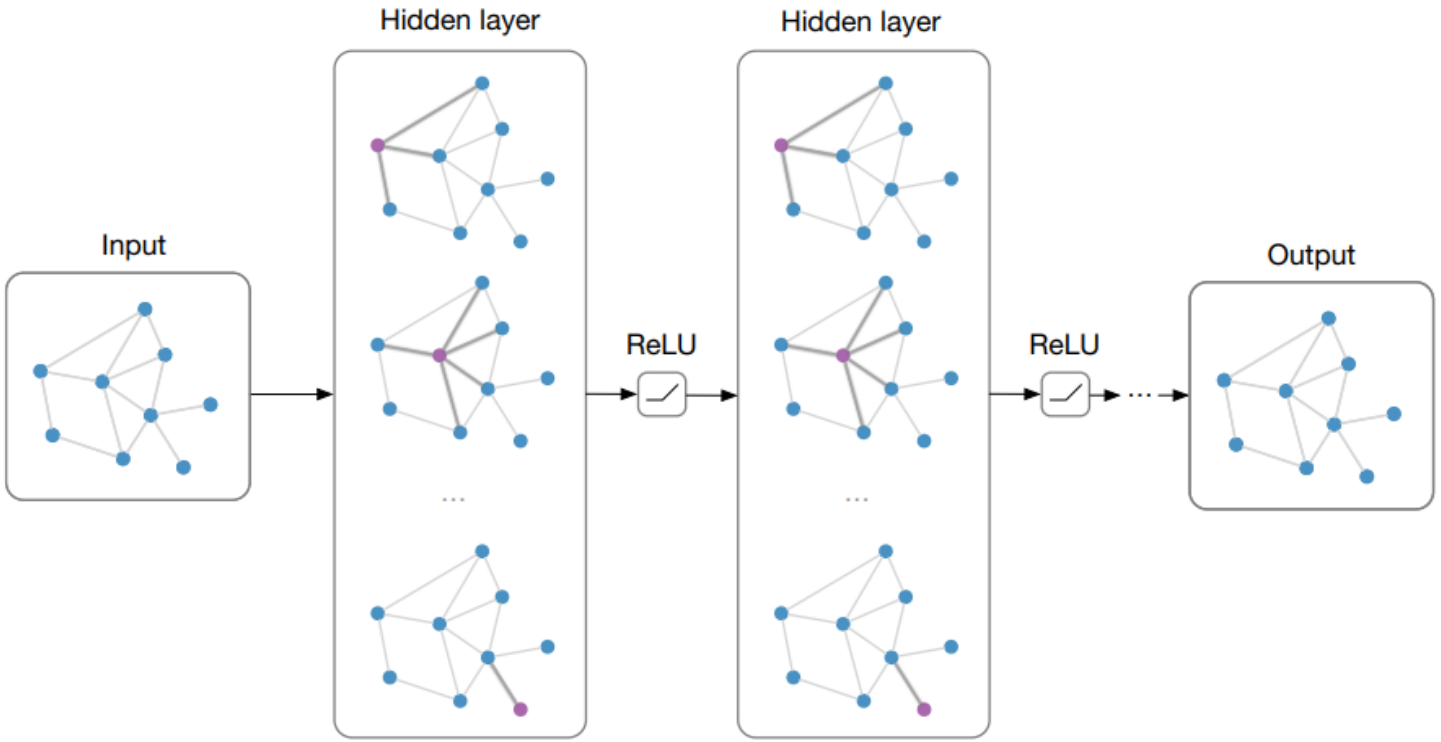


# Brief History



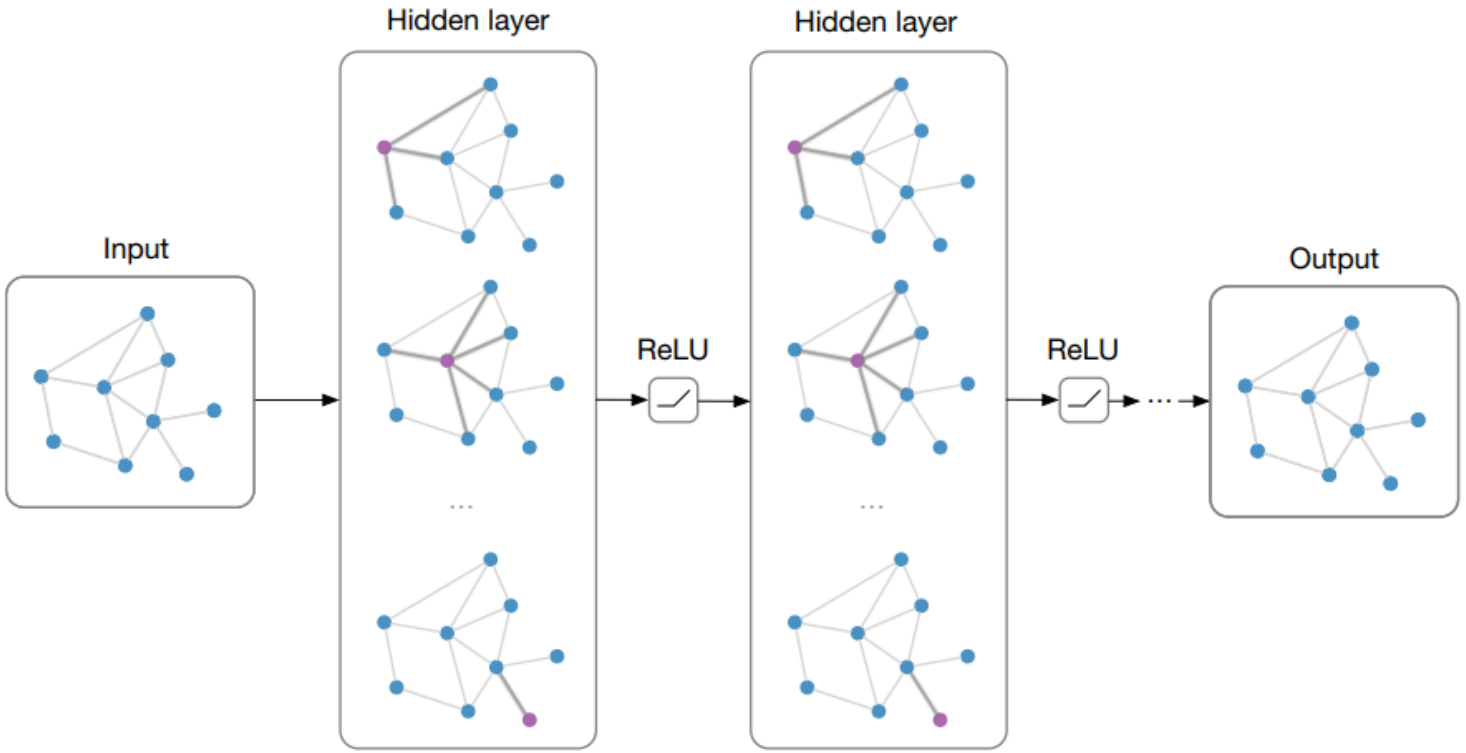
# Graph Neural Networks

## The bigger picture:




Pass messages between pairs of nodes

# Graph Neural Networks



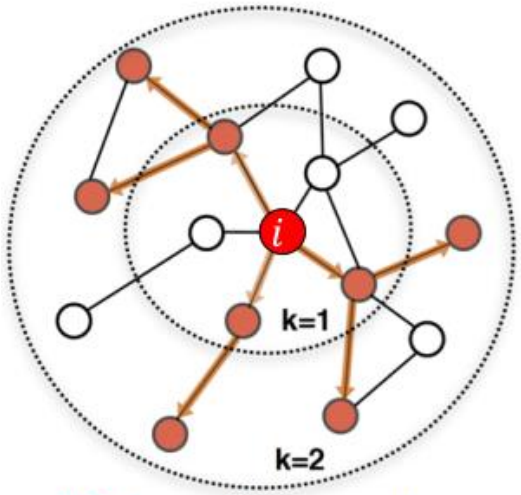

- Notation: $\mathcal{G} = (\mathbf{A}, \mathbf{X})$
- Adjacency matrix: $\mathbf{A} \in \mathbb{R}^{N \times N}$
- Feature matrix: $\mathbf{X} \in \mathbb{R}^{N \times F}$

$N$

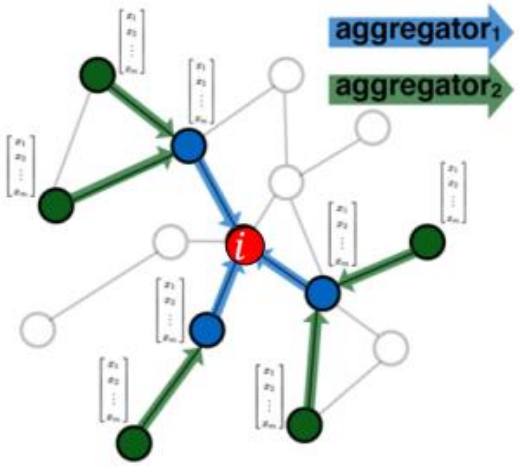
$D$

# Graph Neural Networks

1. Node's neighborhoods define a computation graph
2. Learn how to propagate information across the graph to compute node features
3. Generate node embeddings based on local network neighborhoods



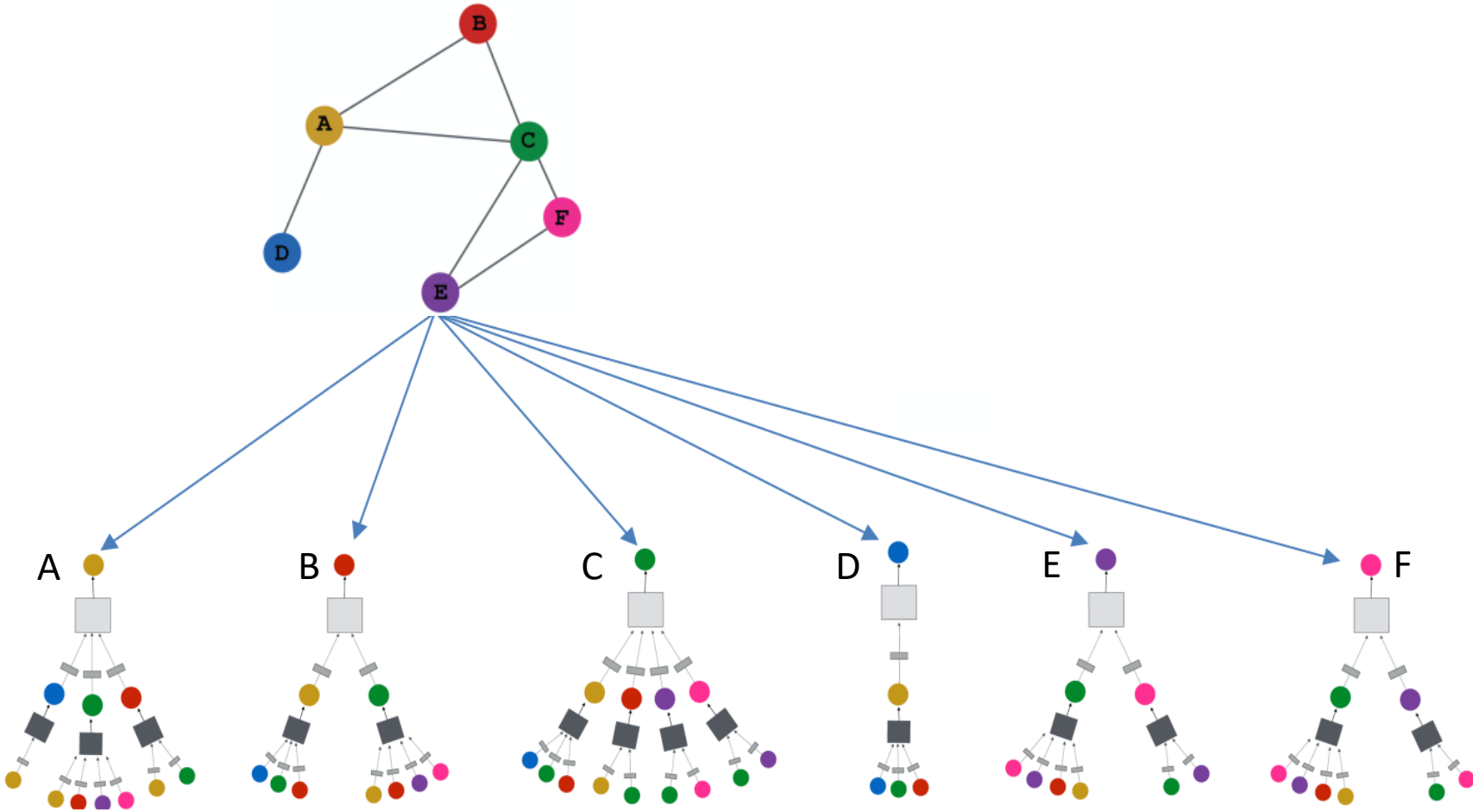

Determine node computation graph




Propagate and transform information

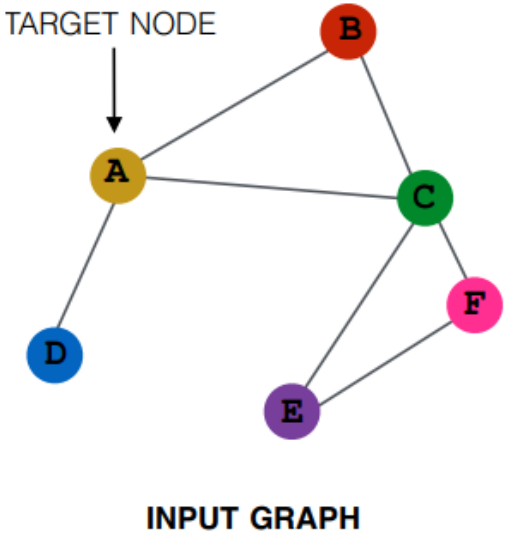
# Computation Graph

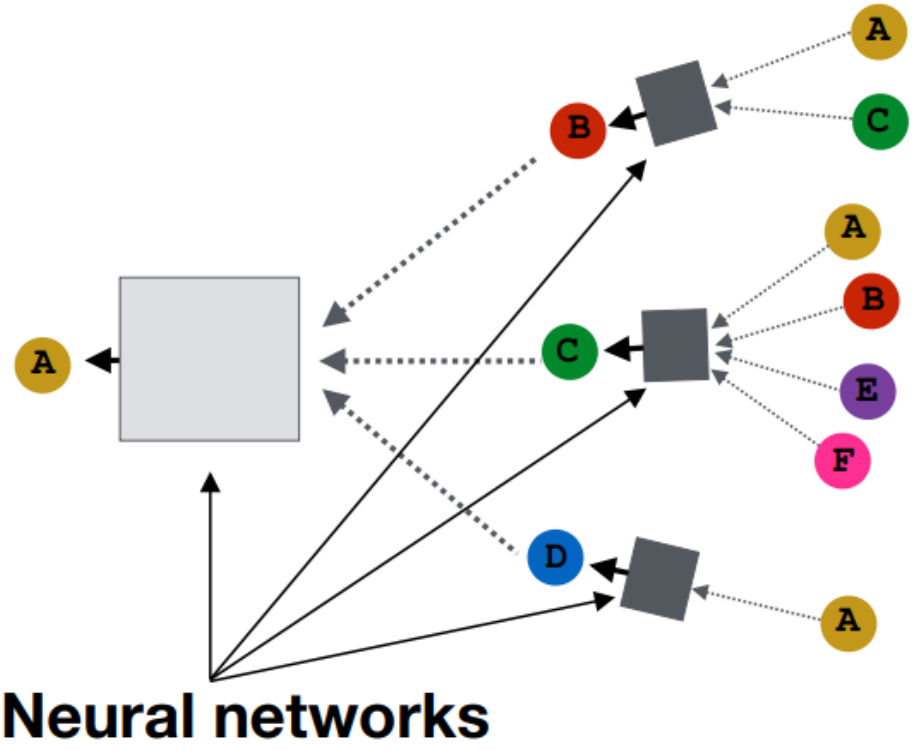
- Every node defines a computation graph based on its neighborhoods



# Propagate Information

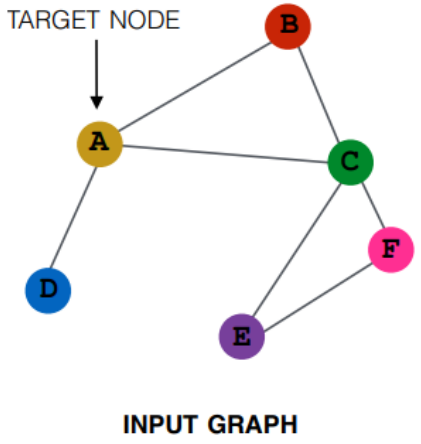
- Every node aggregates information from its neighbors using neural networks



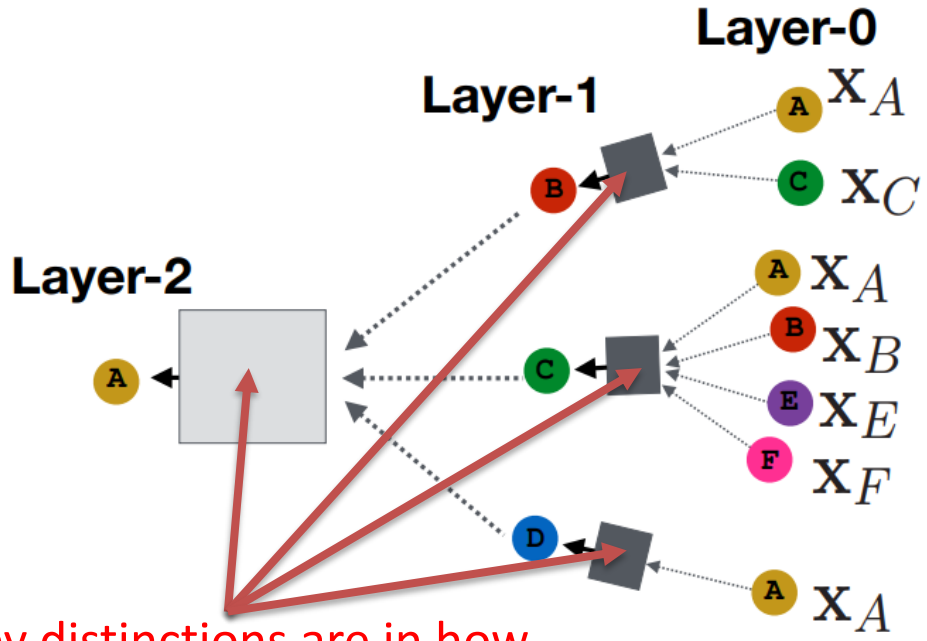



# Node Embedding

- Model can be of arbitrary depth
- Nodes have embeddings at each layer
- Layer-0 embedding of node $u$ is its input feature, $\mathrm{i.e.}, \mathrm{x_u}$



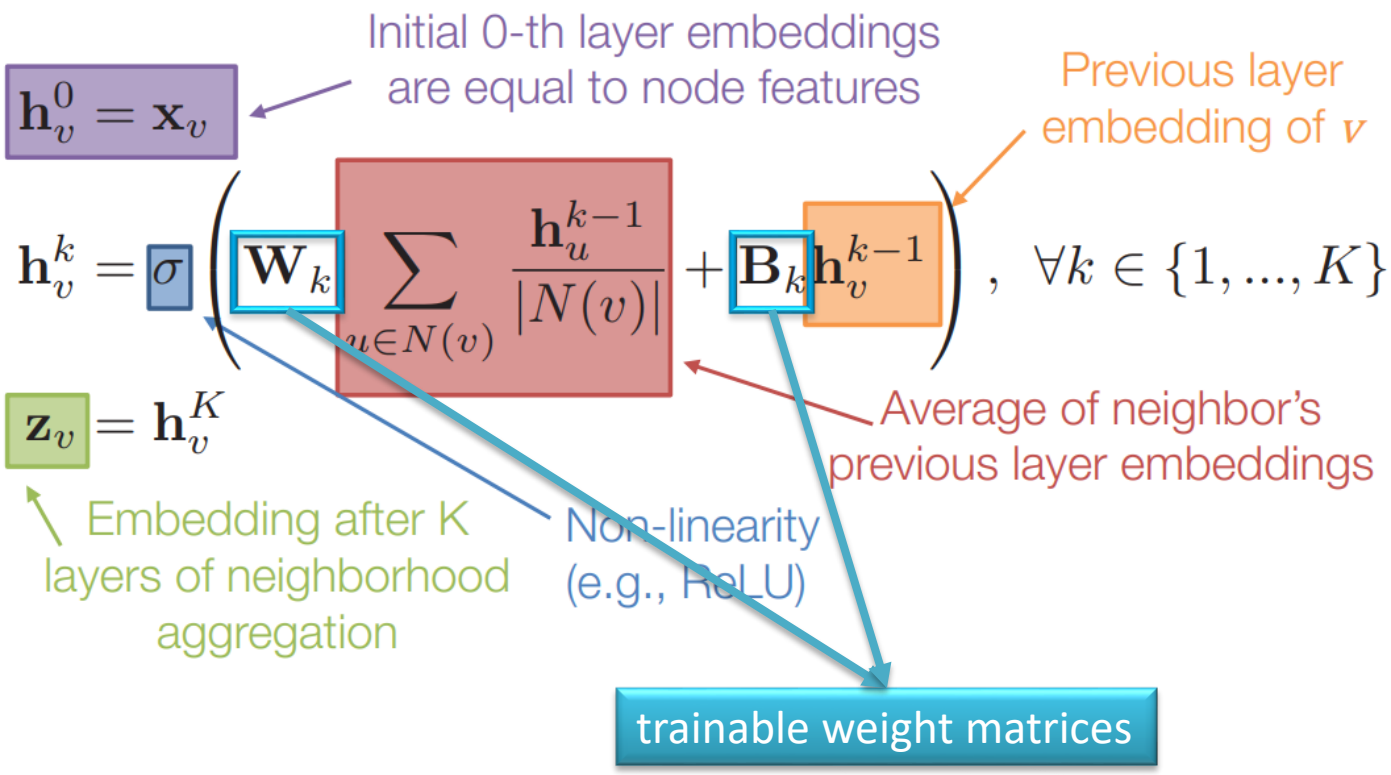

INPUT GRAPH




Key distinctions are in how different approaches aggregate information across the layers

# Node Embedding

**Basic approach:** Average information from neighbors and apply a neural network !



# GNN-Unsupervised Training

**Train in an unsupervised manner:**

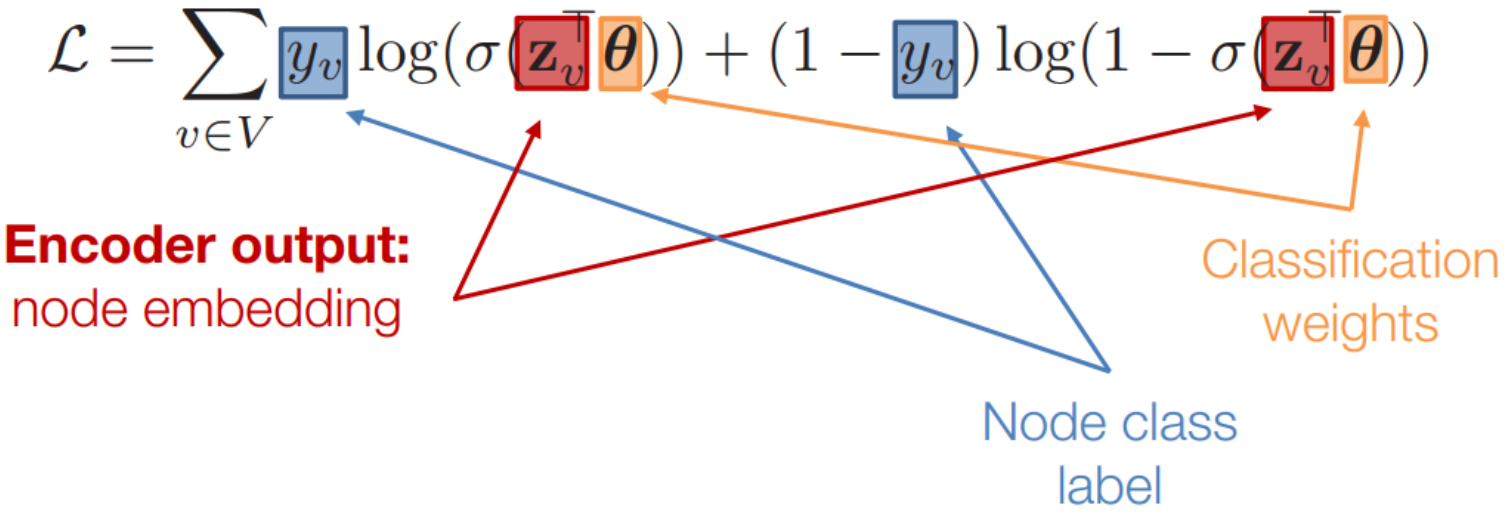
- Use only the graph structure, principle: "similar" nodes have similar embeddings
- Unsupervised loss function can be anything

**Loss based on:**

- Random walks (node2vec, DeepWalk, struc2vec)
- Graph factorization
- Node proximity in the graph

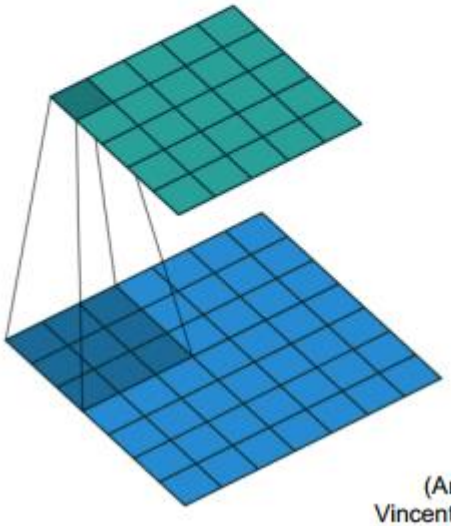
# GNN-Supervised Training

- Directly train the model for a supervised task (e.g., node classification)



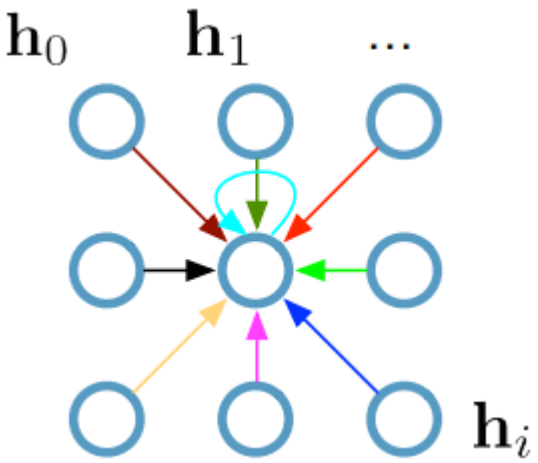
# Recap: CNN

**Single CNN layer with 3x3 filter:**



(Animation by Vincent Dumoulin)

# Recap: CNN




**Update for a single pixel:**

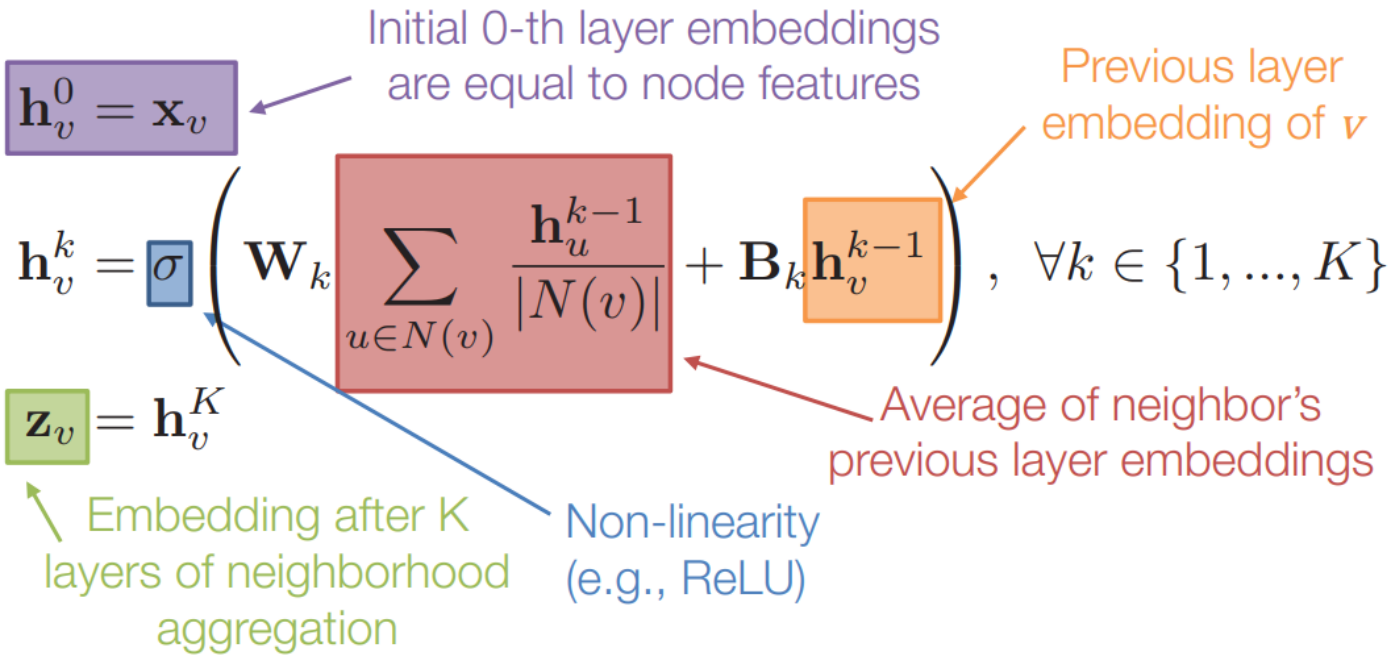
- Transform messages individually $\mathbf{W}_i \mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i \mathbf{h}_i$

**Full update:**

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

# Different Weights

- Average information from neighbors → most graph neural networks have a **somewhat universal architecture**
- How about nodes (like image pixels) have different weights?



# Same Input and Output

The goal is to **learn a function** of signals/features on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the **input**:

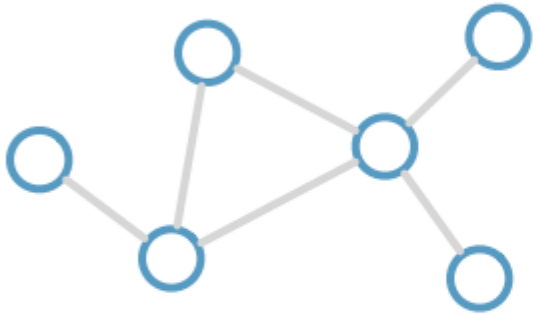
- A feature description $x_i$ for every node, summarized in a $N \times D$ feature matrix $X$
- A representative description of the graph structure in matrix form, typically in the form of an adjacency matrix $A$

**Output:**

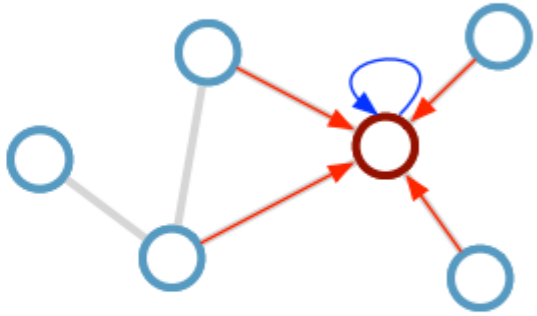
- **Node-level output** $Z$ : $N \times F$ feature matrix, where $F$ is the number of output features per node
- Graph-level outputs can be modeled by introducing some form of pooling operation

# General Framework

Consider this undirected graph:



Calculate update for node in red:



**Update rule:**

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

$\mathcal{N}_i$ : neighbor indices

$c_{ij}$ : norm. constant (fixed/trainable)

Different weights

# General Framework

- Every neural network layer can then be written as a non-linear function (**take *A* into consideration to obtain $c_{ij}$**):

$$H^{(l+1)} = f(H^{(l)}, A)$$

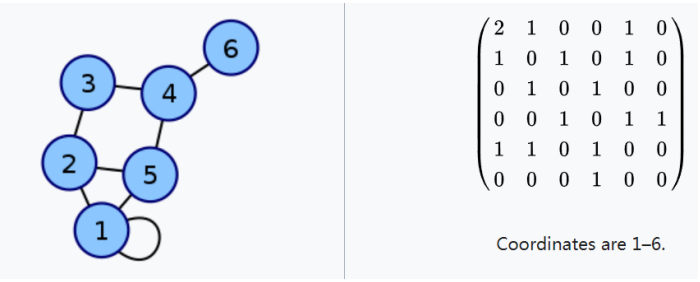
  - with $H^{(0)} = X$ and $H^{(L)} = Z$
  - $L$ being the number of layers

**The specific models differ in how $f(\cdot, \cdot)$ is chosen and parameterized**

# A Simple Case

Let's consider the following very simple form of a layer-wise propagation rule:

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right)$$




- $W^{(l)}$ is a weight matrix for the $l$-th neural network layer
- $\sigma(\cdot)$ is a non-linear activation function like the ReLU

- Multiplication with $A$ means that, for every node, we sum up (with coefficients) all the feature vectors of all neighboring nodes (but not the node itself unless there are self-loops in the graph)

# Limitations

- $A$ is typically not normalized, and therefore the multiplication with $A$ will completely change the scale of the feature vectors

# Solutions

- Normalizing $A$ such that all rows sum to one, $i.e.$, $D^{-1}A$, where $D$ is the diagonal node degree matrix.
- Multiplying with $D^{-1}A$ now corresponds to taking the weighted average of neighboring node features
- In practice, dynamics get more interesting when we use a symmetric normalization $D^{-(1/2)}AD^{-(1/2)}$

# Graph Convolutional Networks

Combining these two tricks, the propagation rule is:

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- With $\hat{A} = A + I$ where $I$ is the identity matrix
- $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$

# Graph Convolutional Networks

**Desirable properties:**

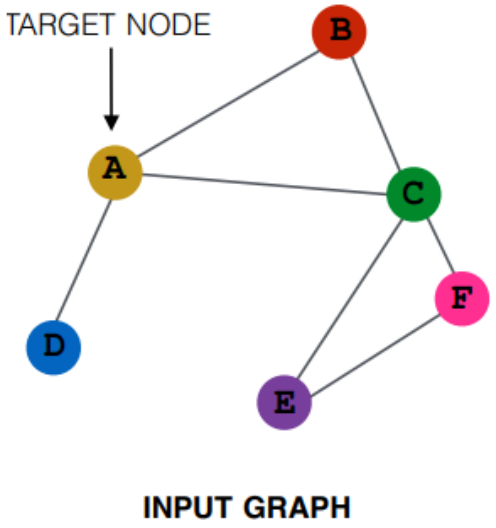
- Weight sharing over all locations
- Invariance to permutations
- Linear complexity $O(E)$

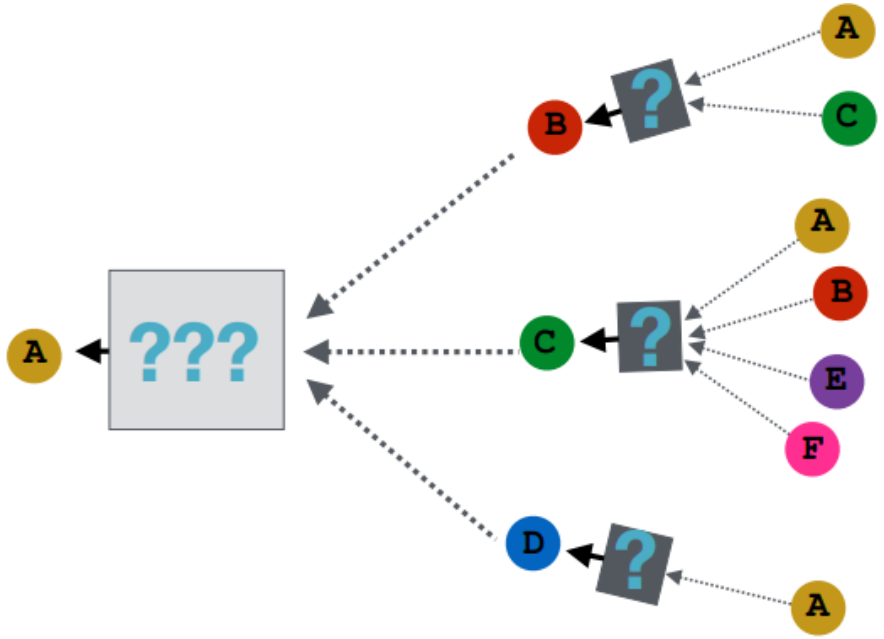
**Limitations:**

- Requires gating mechanism/residual connections for depth
- Only indirect support for edge features

# GraphSAGE (Graph SAmple and aggreGatE)

- We have aggregated the neighbor messages by taking their (weighted) average, **can we do better**?



# GraphSAGE

- Simple neighborhood aggregation:

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

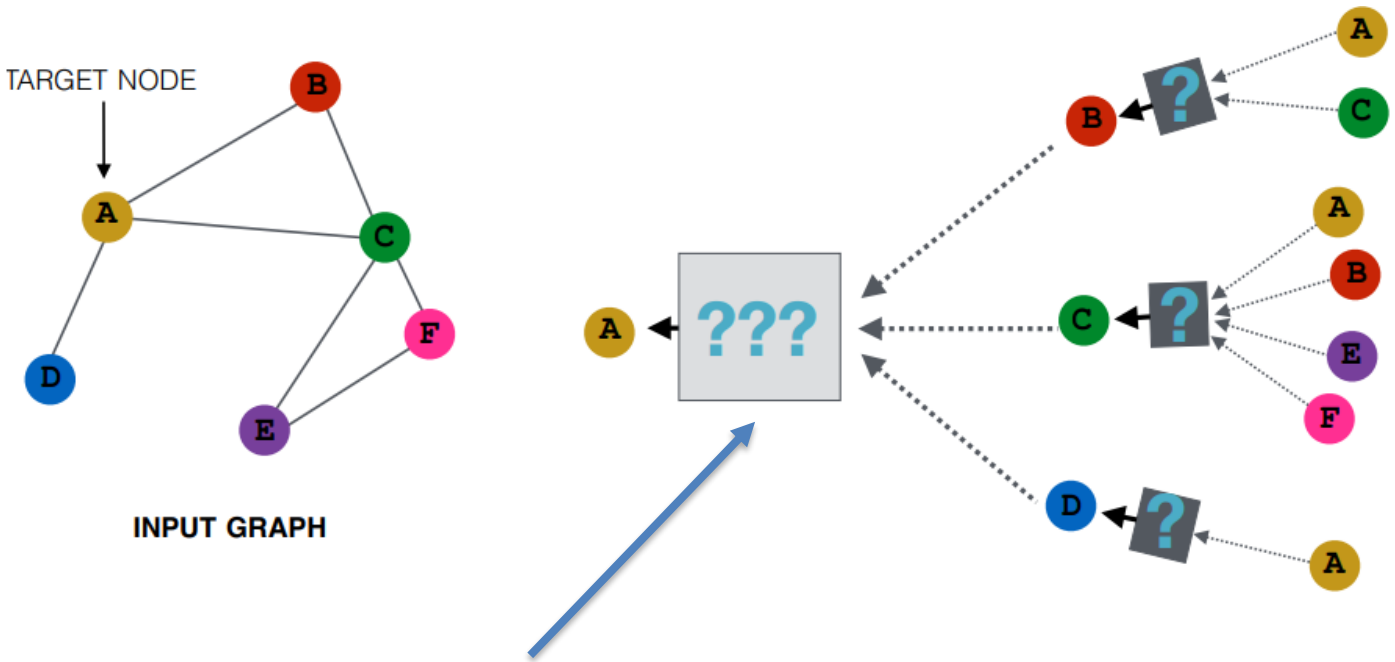
- GraphSAGE

Concatenate self embedding and neighbor embedding

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{W}_k \cdot \mathrm{AGG}\left(\left\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\right\}\right), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

generalized aggregation

# GraphSAGE




Any differentiable function
that maps set of vectors to a single vector

$$\mathbf{h}_v^k = \sigma\left(\left[\mathbf{A}_k \cdot \text{AGG}(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}), \mathbf{B}_k \mathbf{h}_v^{k-1}\right]\right)$$

# GraphSAGE Variants

- **Mean**:

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|}$$

- **Pool**:
  - Transform neighbor vectors by element-wise pooling

element-wise mean/max

$$\text{AGG} = \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right)$$

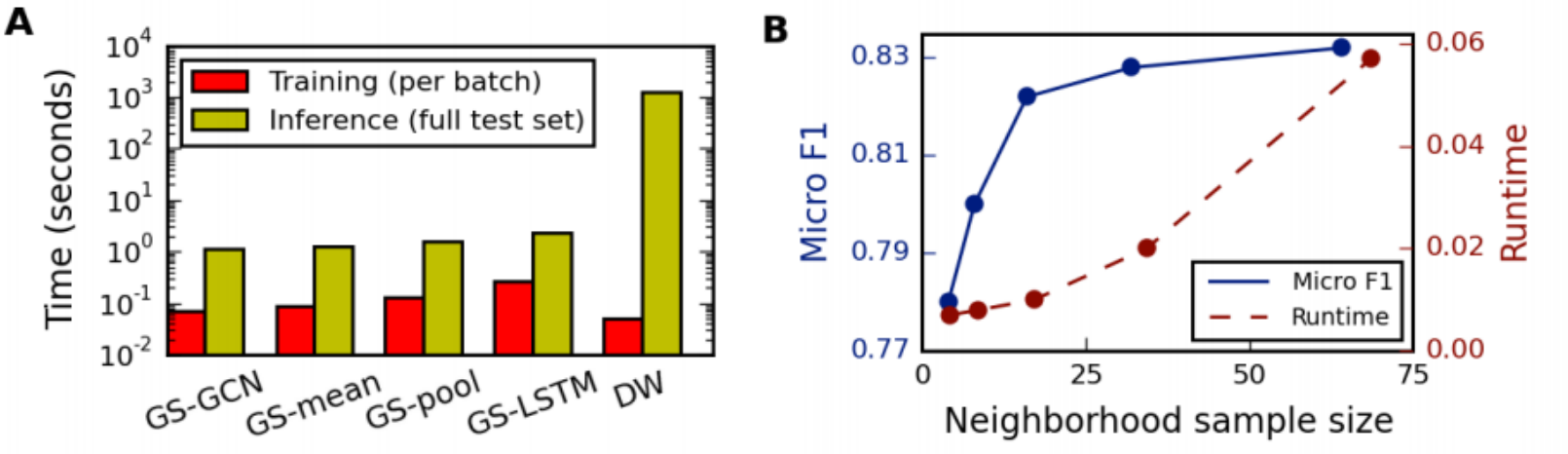
- **LSTM**:
  - Apply LSTM to random permutation of neighbors

$$\text{AGG} = \text{LSTM}\left([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]\right)$$

# Comparisons

Table 1: Prediction results for the three datasets (micro-averaged F1 scores). Results for unsupervised and fully supervised GraphSAGE are shown. Analogous trends hold for macro-averaged scores.

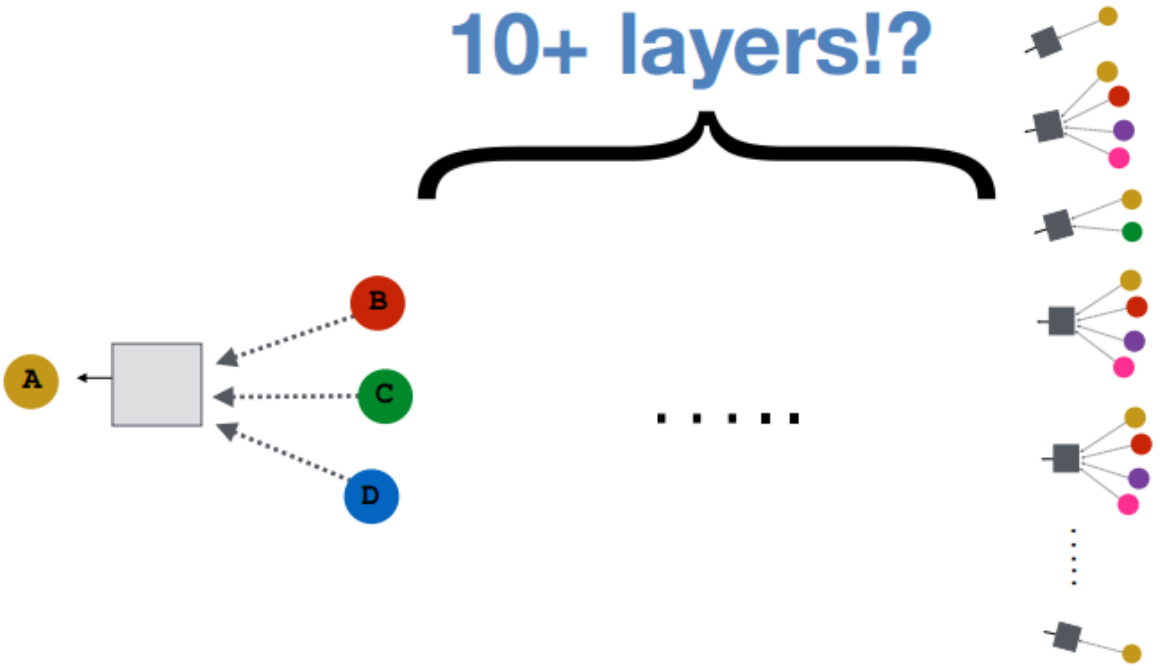
| | Citation | | Reddit | | PPI | |
|---|---|---|---|---|---|---|
| Name | Unsup. F1 | Sup. F1 | Unsup. F1 | Sup. F1 | Unsup. F1 | Sup. F1 |
| Random | 0.206 | 0.206 | 0.043 | 0.042 | 0.396 | 0.396 |
| Raw features | 0.575 | 0.575 | 0.585 | 0.585 | 0.422 | 0.422 |
| DeepWalk | 0.565 | 0.565 | 0.324 | 0.324 | — | — |
| DeepWalk + features | 0.701 | 0.701 | 0.691 | 0.691 | — | — |
| GraphSAGE-GCN | 0.742 | 0.772 | **0.908** | 0.930 | 0.465 | 0.500 |
| GraphSAGE-mean | 0.778 | 0.820 | 0.897 | 0.950 | 0.486 | 0.598 |
| GraphSAGE-LSTM | 0.788 | 0.832 | **0.907** | **0.954** | 0.482 | **0.612** |
| GraphSAGE-pool | **0.798** | **0.839** | 0.892 | 0.948 | **0.502** | 0.600 |
| % gain over feat. | 39% | 46% | 55% | 63% | 19% | 45% |



# Gated Graph Neural Networks

- Nodes aggregate “messages” from their neighbors using neural networks
- **GCNs generally only 2-3 layers deep**

**But what if we want to go deeper?**



# Gated Graph Neural Networks

## Challenges

- Overfitting from too many parameters
- Vanishing/exploding gradients during backpropagation

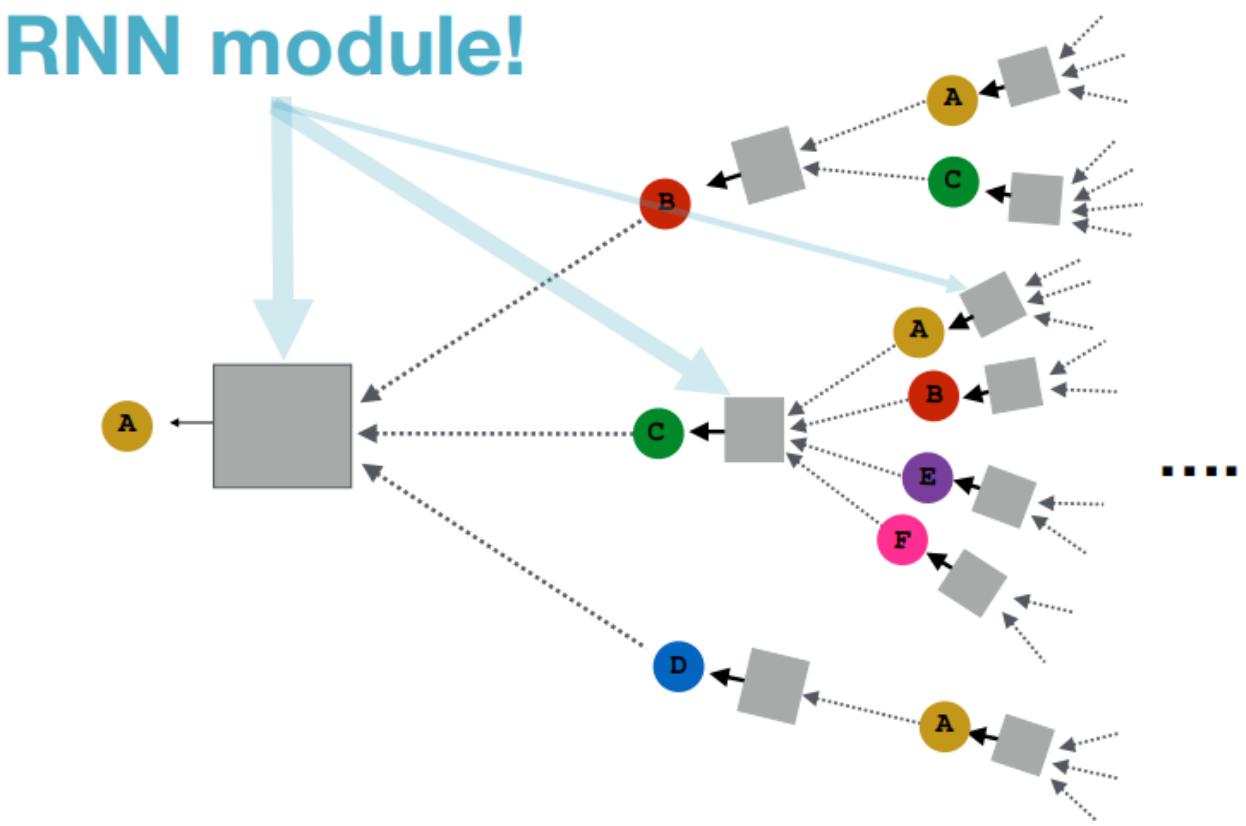
Use techniques from modern recurrent neural networks!

- Parameter sharing across layers
- Recurrent state update

Li et al, Gated Graph Sequence Neural Networks. ICLR, 2016.

# Gated Graph Neural Networks

- Recurrent state update




Li et al, Gated Graph Sequence Neural Networks. ICLR, 2016.

# Gated Graph Neural Networks

## Neighborhood aggregation with RNN state update

- Get "message" from neighbors at step k:

$$\mathbf{m}_v^k = \mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}$$

aggregation function does not depend on $\mathbf{k}$

- Update node "state" using Gated Recurrent Unit (GRU)
- New node state depends on the old state and the message from neighbors:

$$\mathbf{h}_v^k = \mathrm{GRU}(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k)$$

Li et al, Gated Graph Sequence Neural Networks. ICLR, 2016.

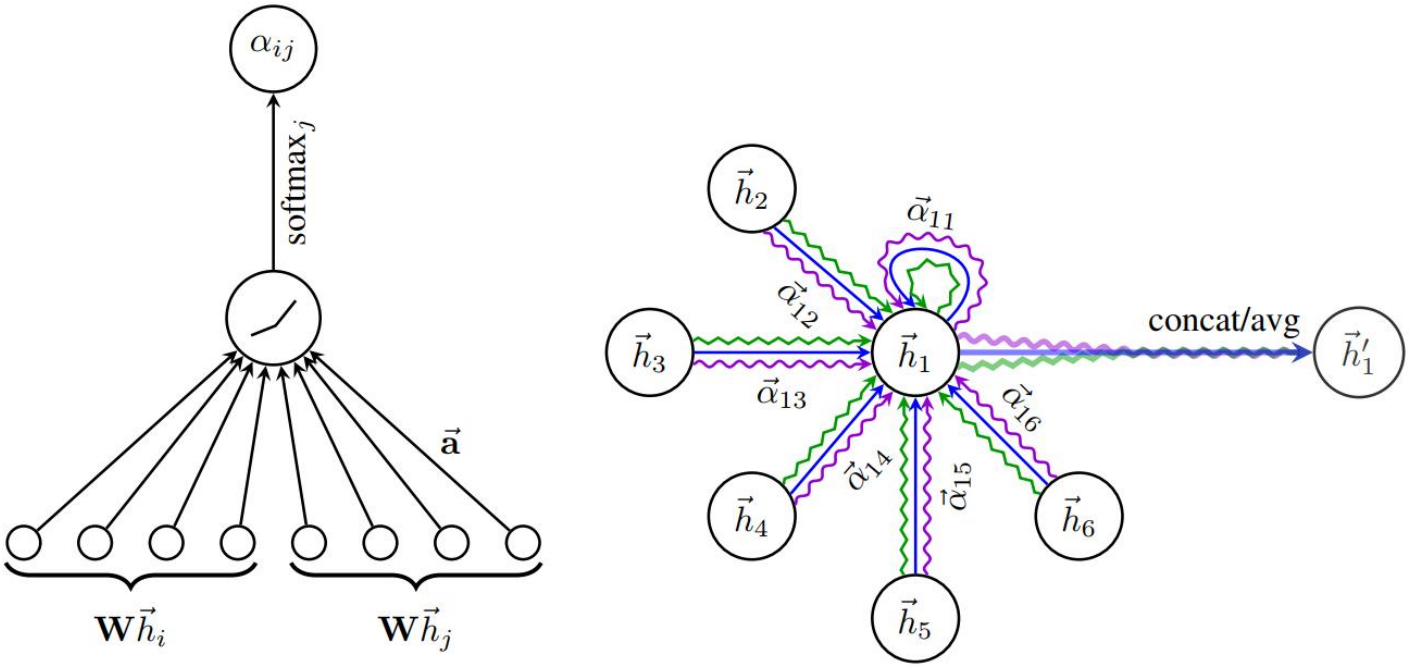
# Gated Graph Neural Networks

**Advantages:**

- Can handle models with **>20 layers**
- Allows for **complex information about global graph structure** to be propagated to all nodes
- Useful for complex networks representing: such as logical formulas and programs

Li et al, Gated Graph Sequence Neural Networks. ICLR, 2016.

# GNN with Attention




$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$

Similar to adjacent matrix

Average: $\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$ Concat: $\vec{h}'_i = \Big\|_{k=1}^{K}\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\vec{h}_j\right)$

Monti et al. (CVPR 2017), Hoshen (NIPS 2017), Veličkovićet al. (ICLR 2018)

# GNN with Attention

**Pros:**

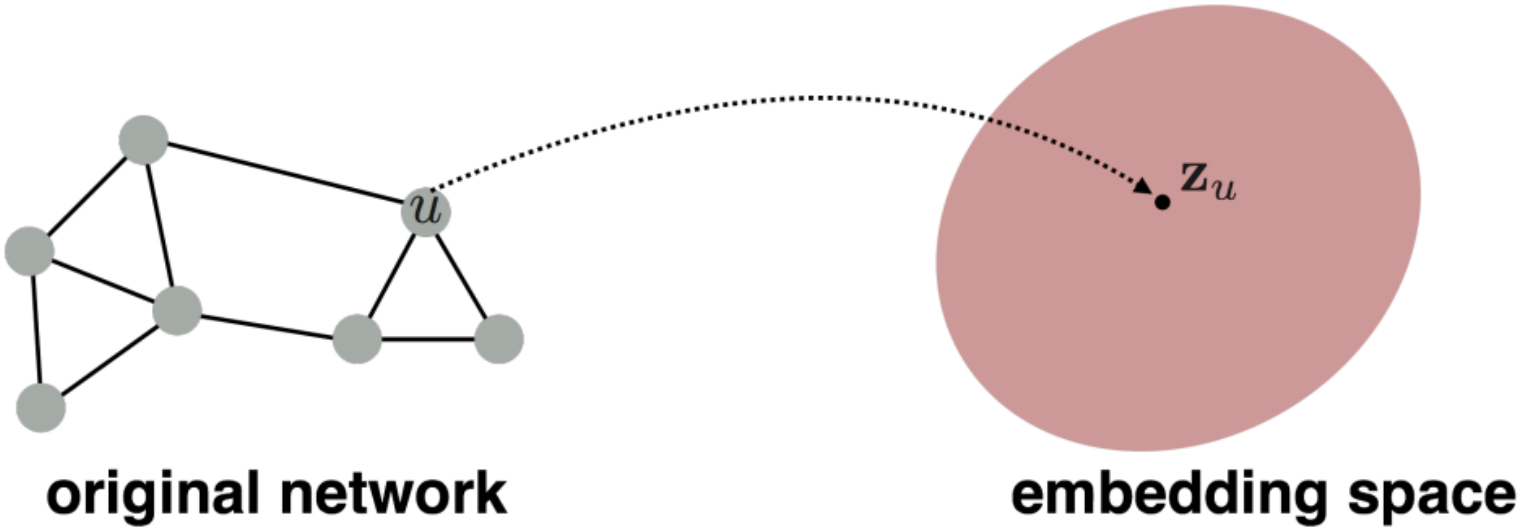
- **No need to store intermediate edge-based activation vectors** (when using dot-product attntion)
- Slower than GCNs but faster than GNNs with edge embeddings (will introduce in the subsequent slides)

**Cons:**

- (Most likely) less expressive than GNNs with edge embeddings
- Can be more difficult to optimize

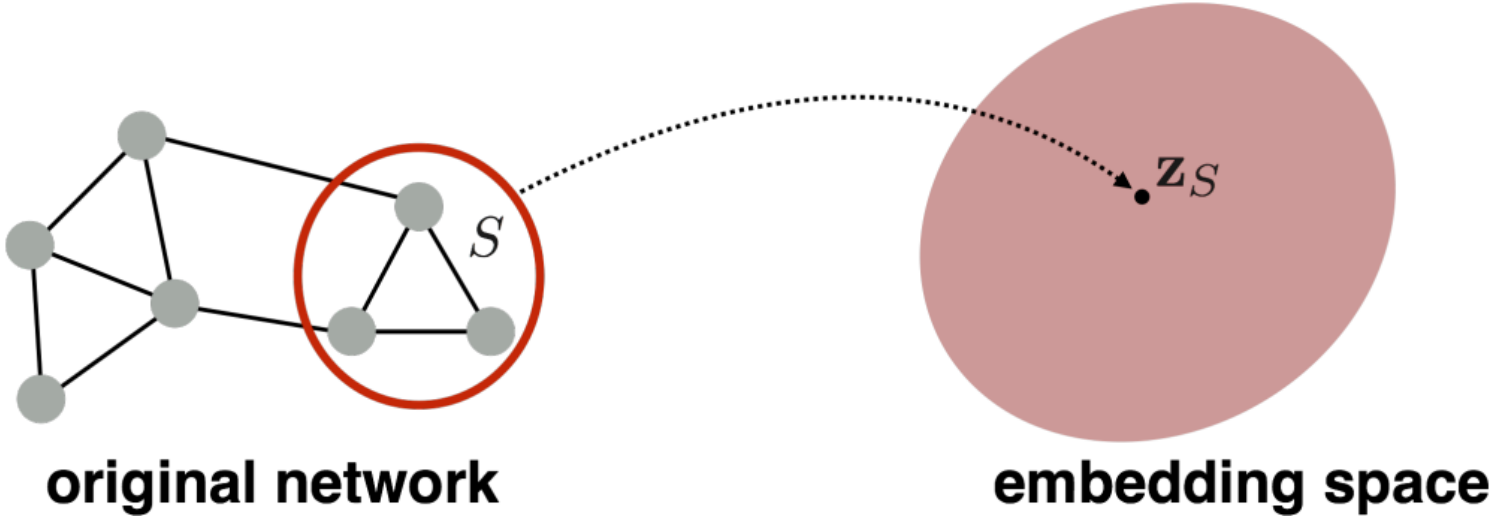
# Subgraph Embeddings

So far we have focused on **node-level embeddings**…



# Subgraph Embeddings

What about **sub-graph embeddings**?

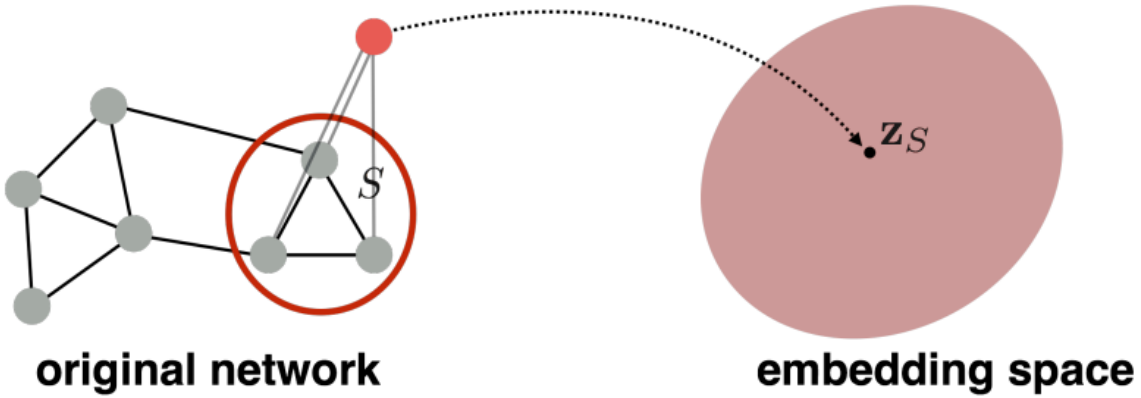


# Subgraph Embeddings

- **Approach 1**
  - Just **sum** (or average) the node embeddings in the (sub)graph:

$$\mathbf{z}_S = \sum_{v \in S} \mathbf{z}_v$$

- **Approach 2**
  - Introduce a "**virtual node**" to represent the subgraph and run a standard graph neural network



# Outline

1. Course Review
2. Graph Neural Networks
3. GNNs for Classical Problems
4. Applications

# GNNs for Classical Problems

- Node classification
  - Predict a type of a given node
- Graph classification
  - Predict a type of a given graph
- Link prediction
  - Predict whether two nodes are linked

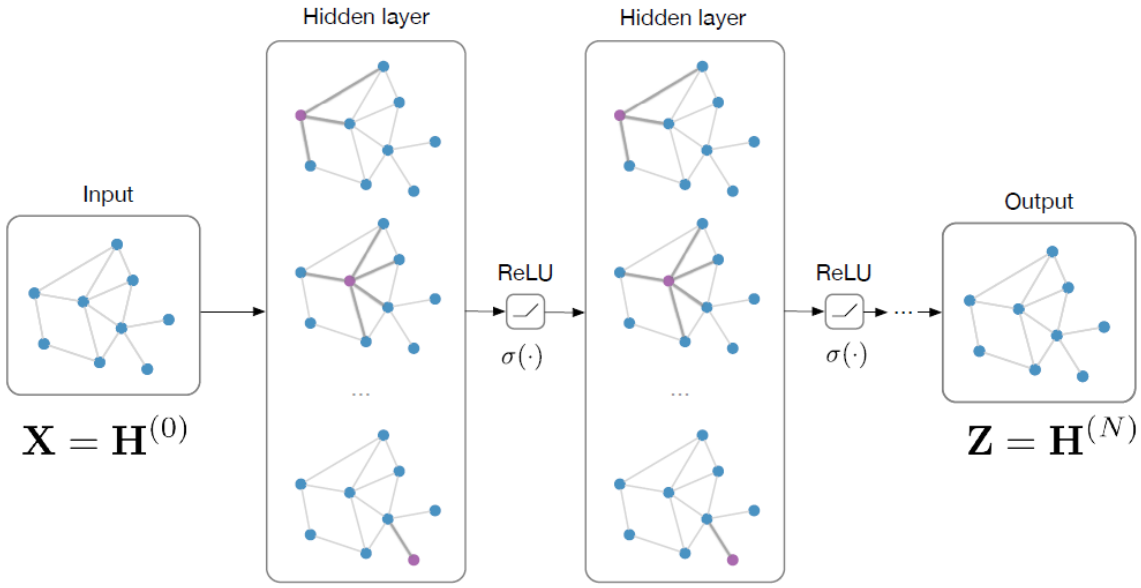
# Classification and Link Prediction

**Input:** feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\widehat{\mathbf{A}}$

**Node classification:**

$softmax(\mathbf{Z_n})$

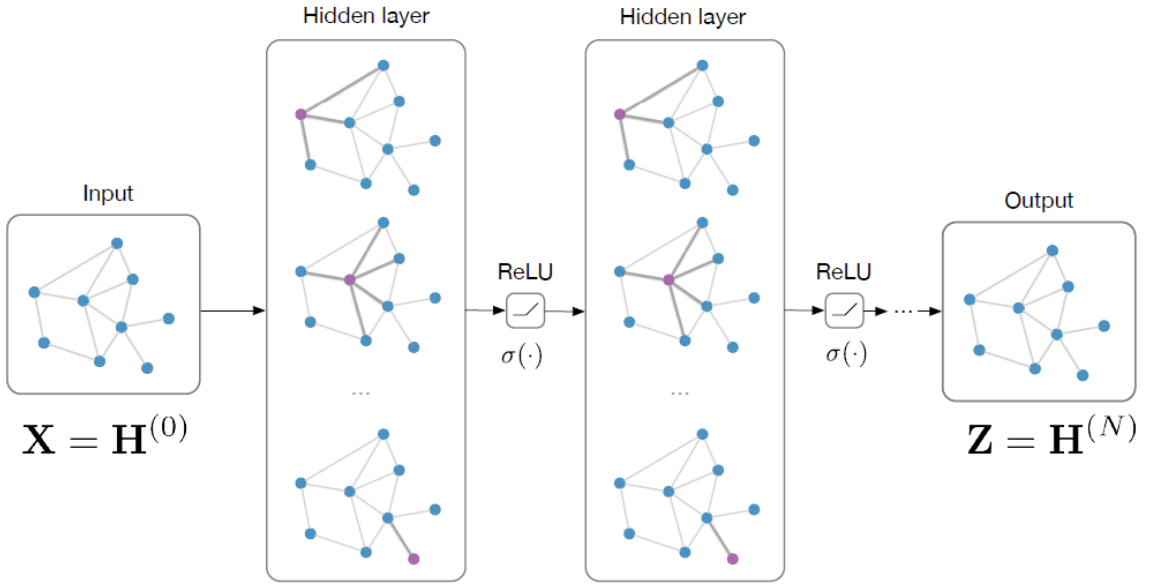
e.g. Kipf & Welling (ICLR 2017)




$$\mathbf{H}^{(l+1)} = \sigma(\widehat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$$

# Classification and Link Prediction

**Input:** Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\widehat{\mathbf{A}}$




**Node classification:**

$$softmax(\mathbf{Z_n})$$

e.g. Kipf & Welling (ICLR 2017)

**Graph classification:**

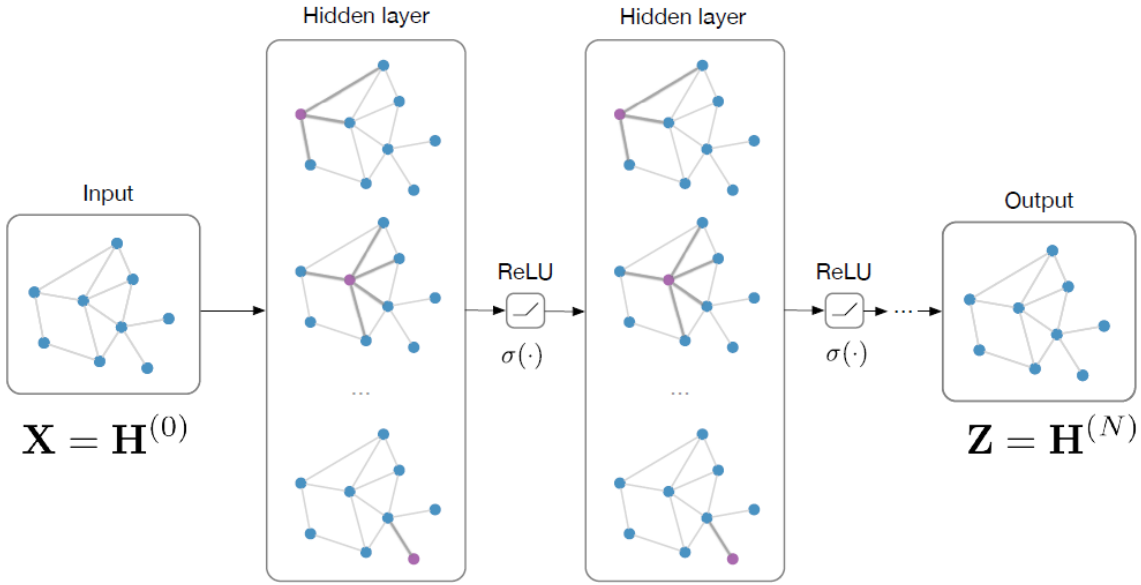
$$softmax(\sum_n \mathbf{Z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

$$\mathbf{H}^{(l+1)} = \sigma(\widehat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$$

# Classification and Link Prediction

**Input:** Feature matrix $\mathbf{X} \in \mathbb{R}^{N \times E}$, preprocessed adjacency matrix $\widehat{\mathbf{A}}$




$$\mathbf{H}^{(l+1)} = \sigma(\widehat{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)})$$

**Node classification:**

$$softmax(\mathbf{Z_n})$$

e.g. Kipf & Welling (ICLR 2017)

**Graph classification:**

$$softmax(\sum_n \mathbf{Z_n})$$

e.g. Duvenaud et al. (NIPS 2015)

**Link prediction:**

$$p(A_{ij}) = \sigma(\mathbf{z}_i^T \mathbf{z_j})$$

Kipf & Welling (NIPS BDL 2016)

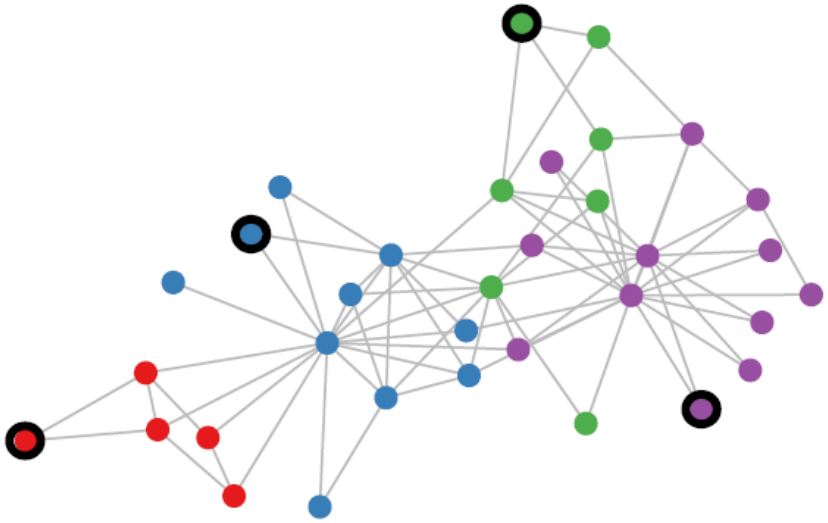
"Graph Auto-Encoders"

# Semi-supervised Classification

**Setting:**

Some nodes are labeled (black circle)

All other nodes are unlabeled



**Task:**

Predict node label of unlabeled nodes

Evaluate loss on labeled nodes only:

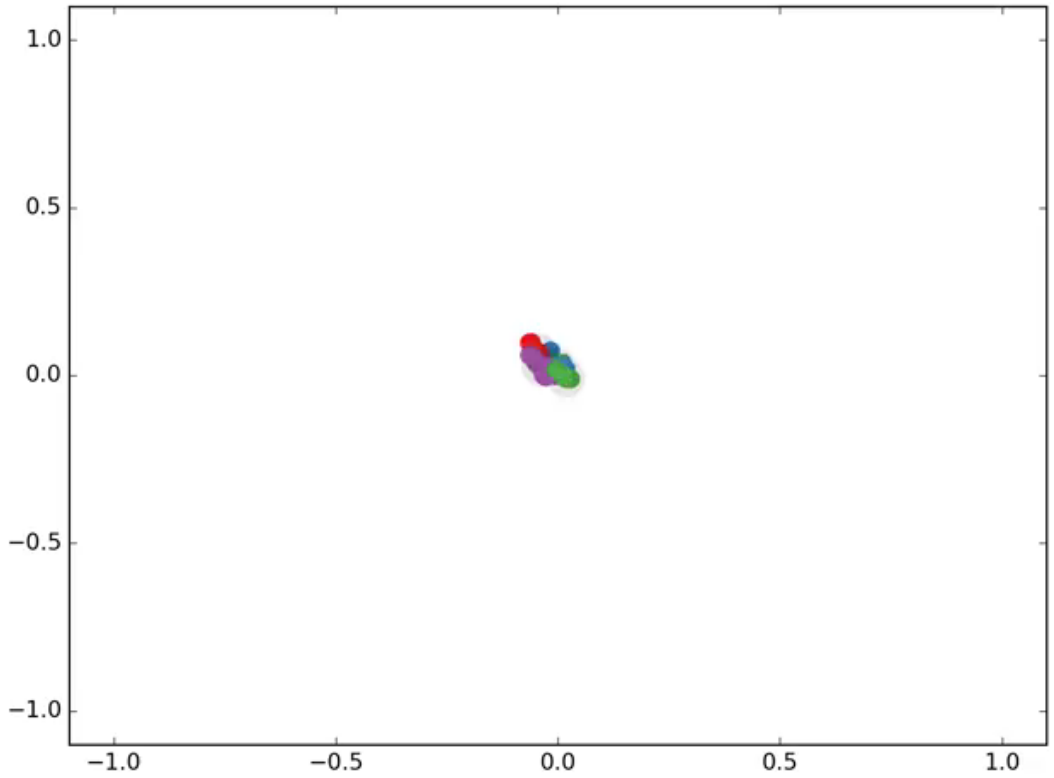
$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}$$

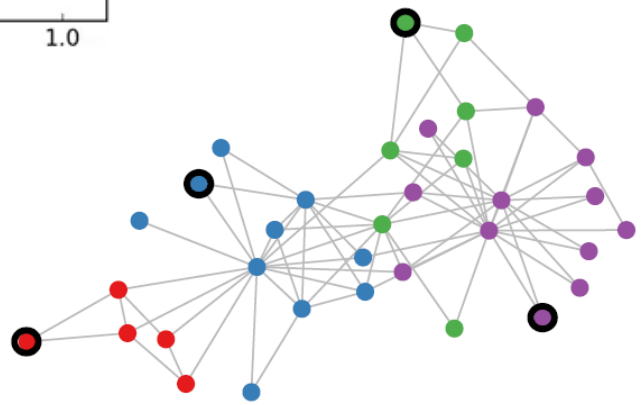
$\mathcal{Y}_L$ set of labeled node indices

$\mathbf{Y}$ label matrix

$\mathbf{Z}$ GCN output (after softmax)

# Toy Example



Video also available here:

http://tkipf.github.io/graph-convolutional-networks

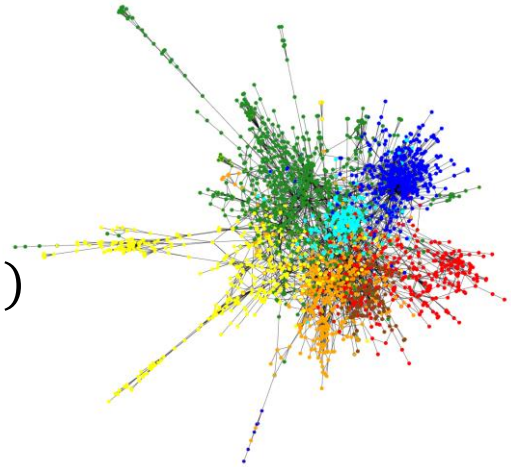
# Other Examples: Paper Classification

**Input:** Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

**Target:** Paper classification (e.g. stat.ML, cs.LG, …)

**Model:**

2-layer GCN $Z = f(X, A) = softmax(\hat{A}\, ReLU(\hat{A}XW^{(0)})W^{(1)})$



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

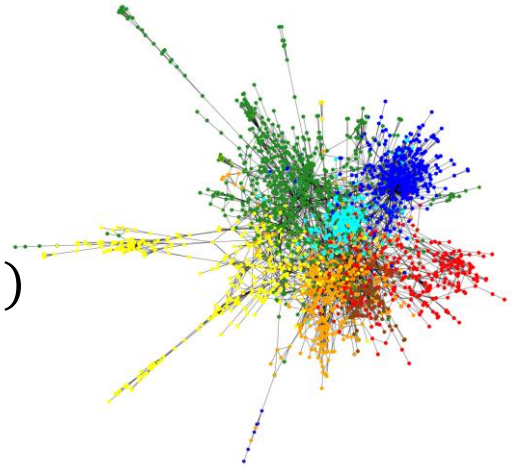
# Other Examples: Paper Classification

**Input:** Citation networks (nodes are papers, edges are citation links, optionally bag-of-words features on nodes)

**Target:** Paper classification (e.g. stat.ML, cs.LG, …)

**Model:**

2-layer GCN $Z = f(X, A) = softmax(\hat{A}\, ReLU(\hat{A}XW^{(0)})W^{(1)})$



(Figure from: Bronstein, Bruna, LeCun, Szlam, Vandergheynst, 2016)

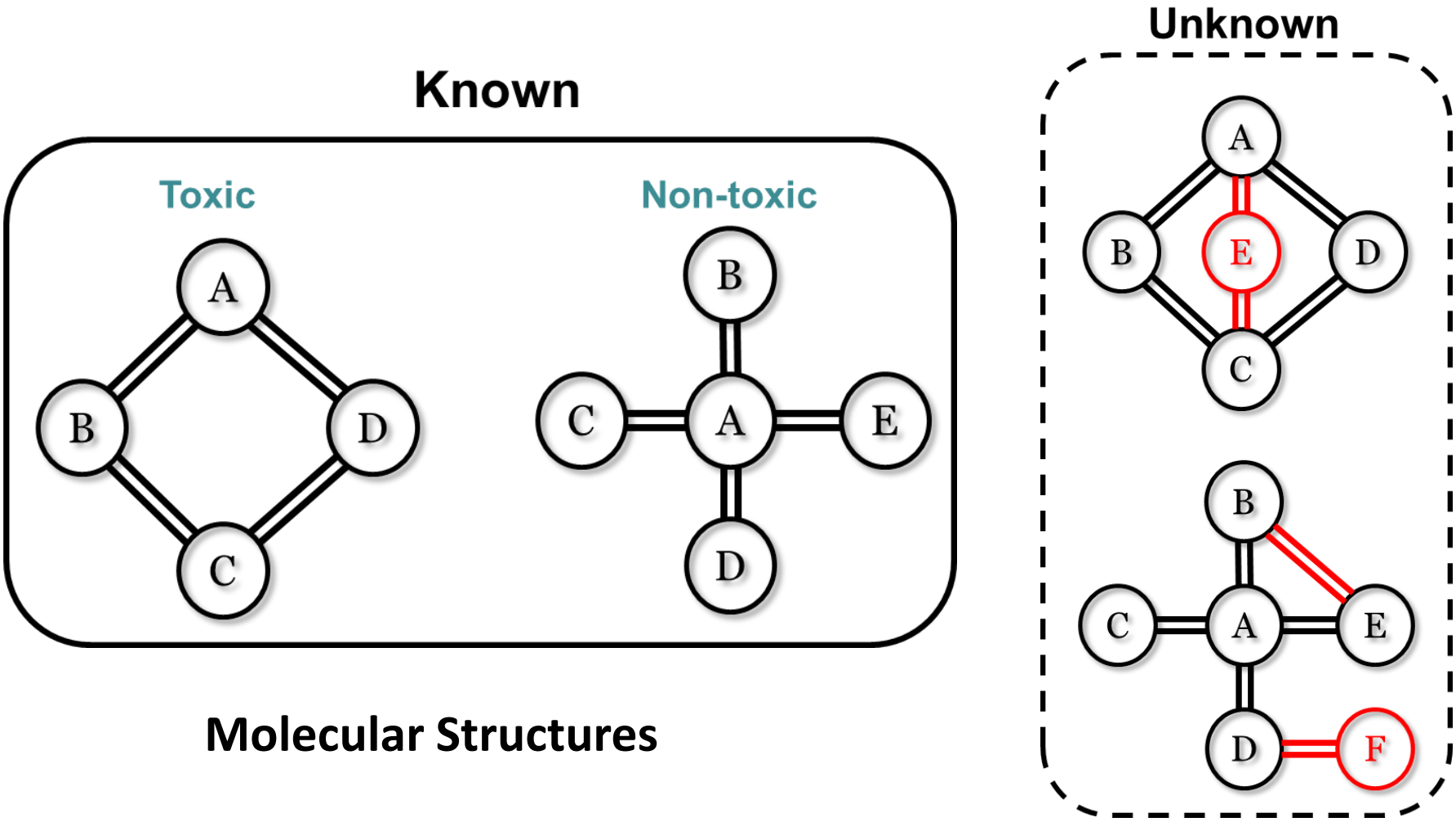
Classification results (accuracy)

| Method | Citeseer | Cora | Pubmed | NELL |
|---|---|---|---|---|
| ManiReg [3] | 60.1 | 59.5 | 70.7 | 21.8 |
| SemiEmb [24] | 59.6 | 59.0 | 71.1 | 26.7 |
| LP [27] | 45.3 | 68.0 | 63.0 | 26.5 |
| DeepWalk [18] | 43.2 | 67.2 | 65.3 | 58.1 |
| Planetoid* [25] | 64.7 (26s) | 75.7 (13s) | 77.2 (25s) | 61.9 (185s) |
| **GCN** (this paper) | **70.3** (7s) | **81.5** (4s) | **79.0** (38s) | **66.0** (48s) |
| GCN (rand. splits) | $67.9 \pm 0.5$ | $80.1 \pm 0.5$ | $78.9 \pm 0.7$ | $58.4 \pm 1.7$ |

no input features

Kipf & Welling, Semi-Supervised Classification with Graph Convolutional Networks, ICLR 2017

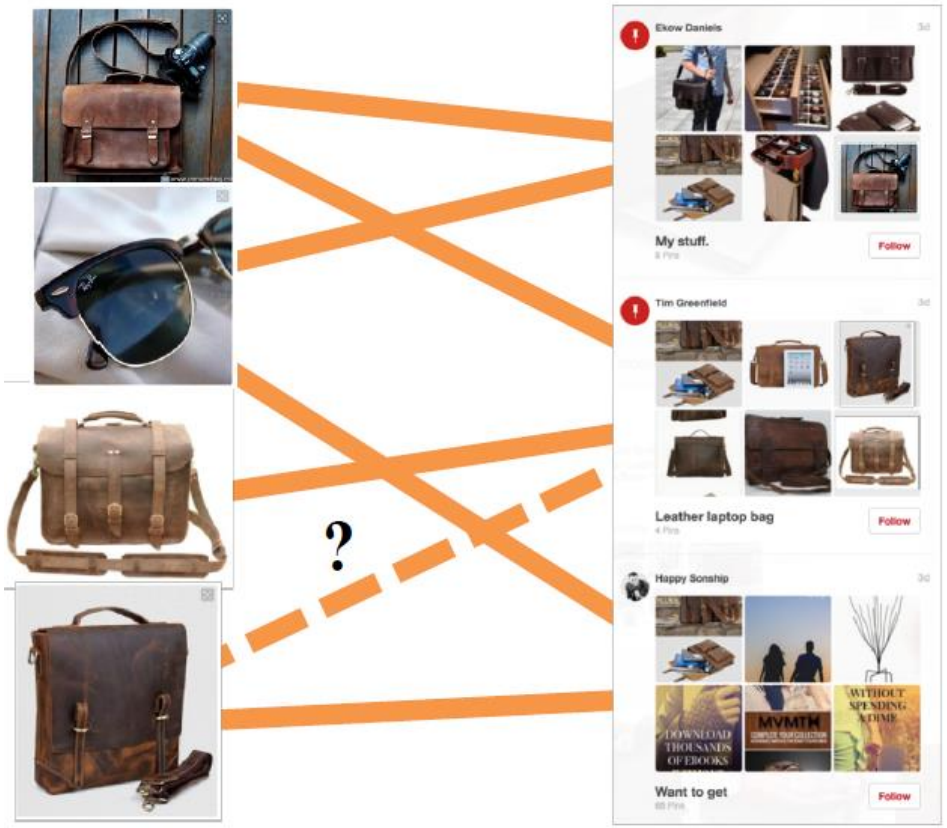
# Other Examples: Molecular Classification




**Molecular Structures**

**Task:** predict whether molecules are toxic, given set of known examples

# Other Examples: Link Prediction




Content recommendation is link prediction!

# Outline

1. Course Review
2. Graph Neural Networks
3. GNNs for Classical Problems
4. Applications

# Neural Relational Inference for Interacting Systems

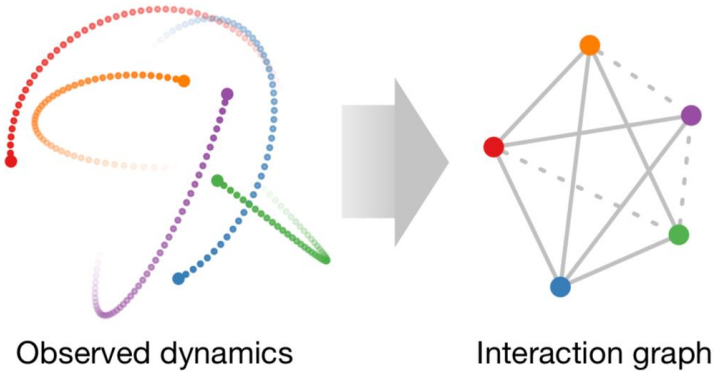
Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang,
Max Welling, Richard Zemel

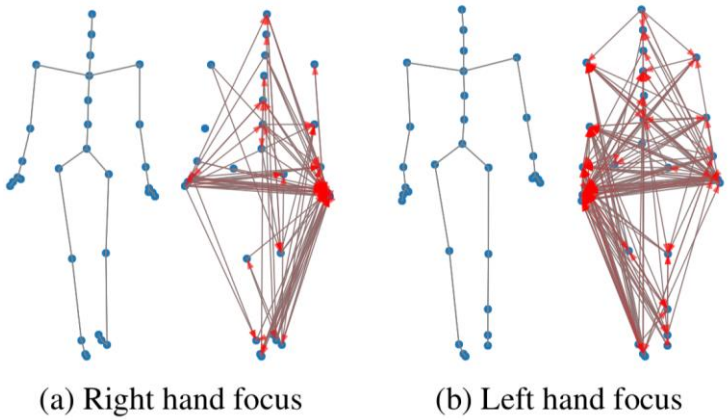
ICML, 2018

# Latent Graph Inference

## Neural Relational Inference for Interacting Systems

Thomas Kipf [*1] Ethan Fetaya [*2 3] Kuan-Chieh Wang [2 3] Max Welling [1 4] Richard Zemel [2 3 4]



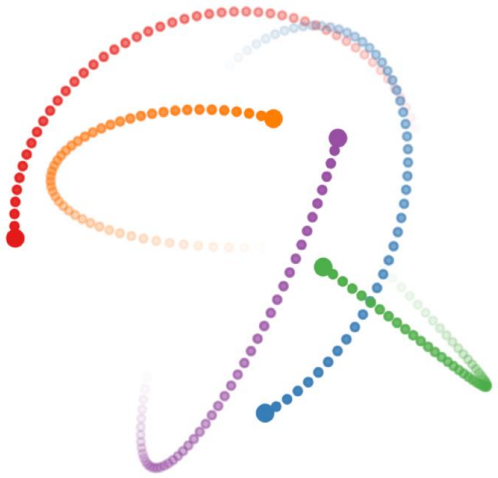




Kipf, Thomas, et al. "Neural relational inference for interacting systems." ICML(2018).

# Motivation: Learning Physical Dynamics

Need model **interactions** and their **effect on dynamics**

**Simple example:**



Observed dynamics

- 5 particles + their trajectories $\mathrm{X}_i^t$
- Concatenate feature vectors

$$\mathrm{X}^t = [\mathrm{X}_1^t, \mathrm{X}_2^t, \ldots, \mathrm{X}_5^t]$$

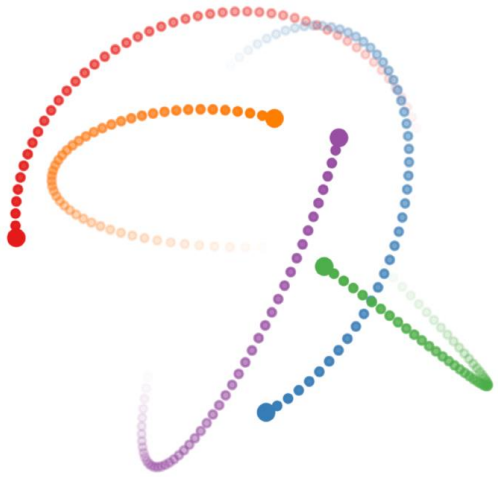
- Feed into neural net

$$\mathrm{X}^{t+1} = \mathrm{MLP}(\mathrm{X}^t) \text{ (or RNN)}$$

- Done?

**Works (somewhat), but we can do a lot better.**

# A Naïve Model of Intuitive Physics



Observed dynamics

**Problems:**

- Arbitrary ordering of nodes

  Need permutation equivariance

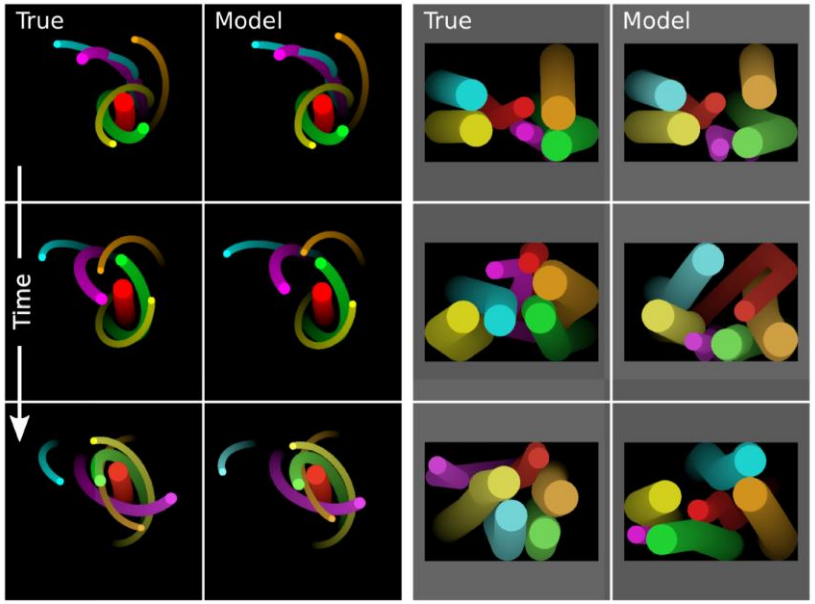
- Model doesn't know about structure of interactions

  For many fundamental physical systems, interactions are pairwise

**Using GNNs, we can learn to model physical dynamics of interacting systems with very high precision**

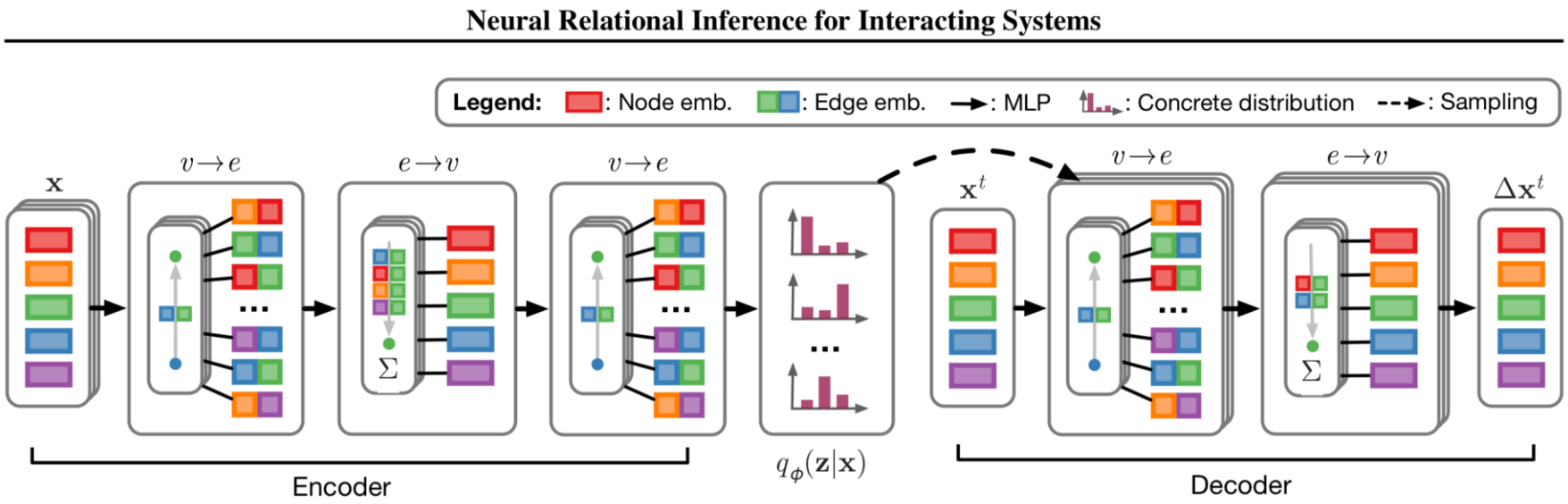
# GNNs for Interacting Systems

1. Learn dynamics of interacting system without knowing structure of interactions
2. Infer latent interaction graph (plus edge types) using a VAE
3. Applications for physical systems, motion capture data and multi-agent systems




Battaglia et al., (NIPS 2016)

# Model Overview




**Model:** Variational auto-encoder with (discrete) edge types as discrete latent variables

**Encoder and decoder are GNN-based!**

Main intuition:

- Encoder generates hypothesis on how the system interacts
- Decoder learns a dynamical model constrained by the encoder's "interaction hypothesis"

Kipf, Thomas, et al. "Neural relational inference for interacting systems." ICML(2018).

# Learning Latent Interaction Graphs

*Table 1.* Accuracy (in %) of unsupervised interaction recovery.

| Model | Springs | Charged | Kuramoto |
|---|---|---|---|
| 5 objects | | | |
| Corr. (path) | 52.4 | 55.8 | 62.8 |
| Corr. (LSTM) | 55.0 | 61.8 | 55.9 |
| NRI (sim.) | **99.9** | 59.1 | – |
| NRI (learned) | **99.8** | **82.6** | **96.0** |
| Supervised | 99.9 | 96.3 | 99.8 |
| 10 objects | | | |
| Corr. (path) | 50.4 | 51.4 | 59.3 |
| Corr. (LSTM) | 51.0 | 52.9 | 53.9 |
| NRI (sim.) | **98.2** | 53.9 | – |
| NRI (learned) | **98.4** | **71.3** | **76.2** |
| Supervised | 98.7 | 94.9 | 96.9 |

**NRI can learn to discover ground-truth relations with very high accuracy!**

**Potential applications:**

- Inference of causal relations
- Discover protein interaction networks

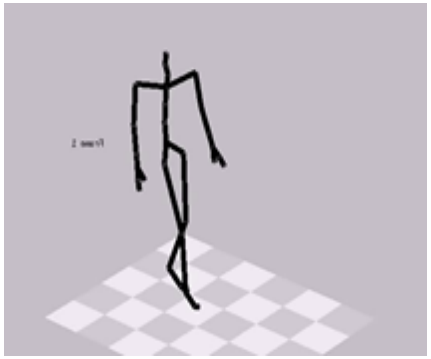
# Skeleton-Based Action Recognition with Spatial Reasoning and Temporal Stack Learning

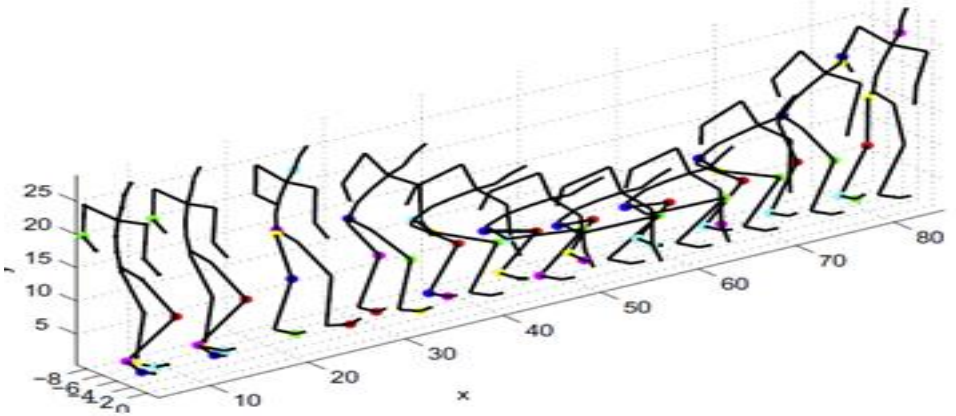
Chenyang Si, Ya Jing, Wei Wang, Liang Wang, Tieniu Tan

ECCV, 2018

# Skeleton-Based Action Recognition

**Human action** – skeleton sequence



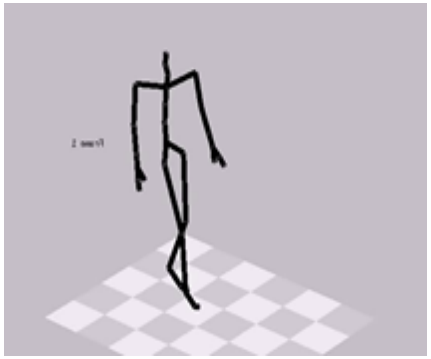


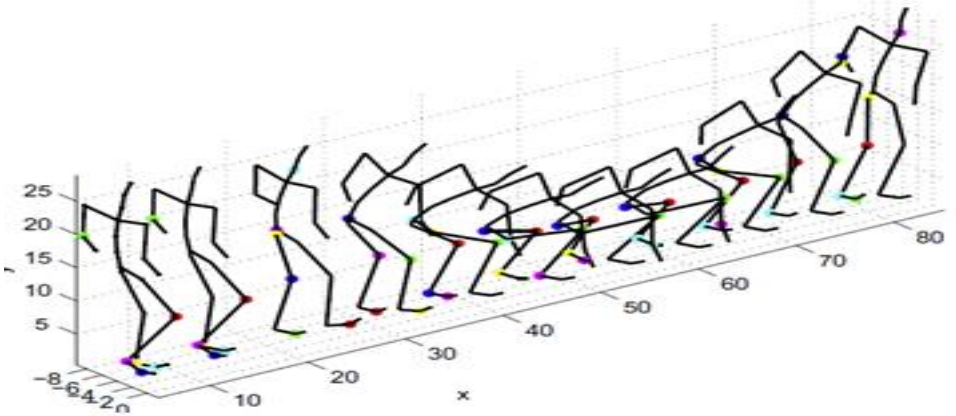

**Spatial**: joints coordinates

**Temporal**: sequences

# Skeleton-Based Action Recognition

**Human action** – skeleton sequence





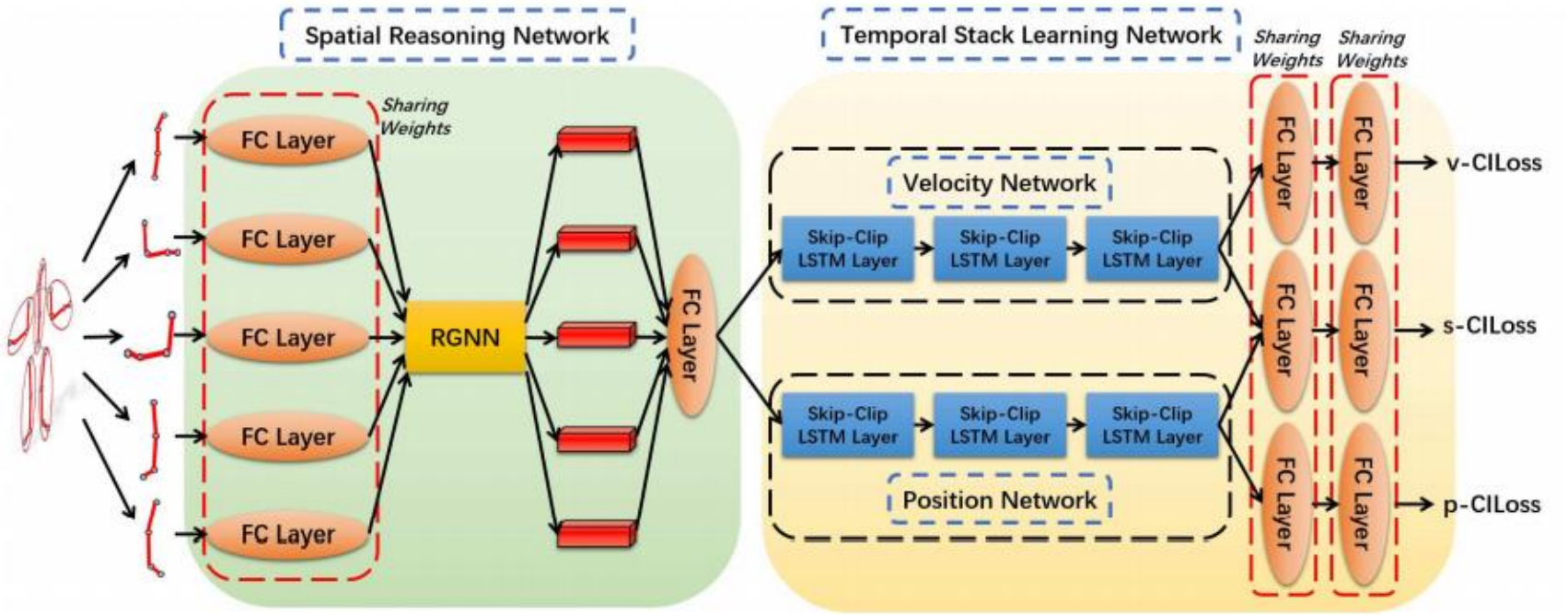

**Spatial**: joints coordinates

Spatial structural information

**Temporal**: sequences

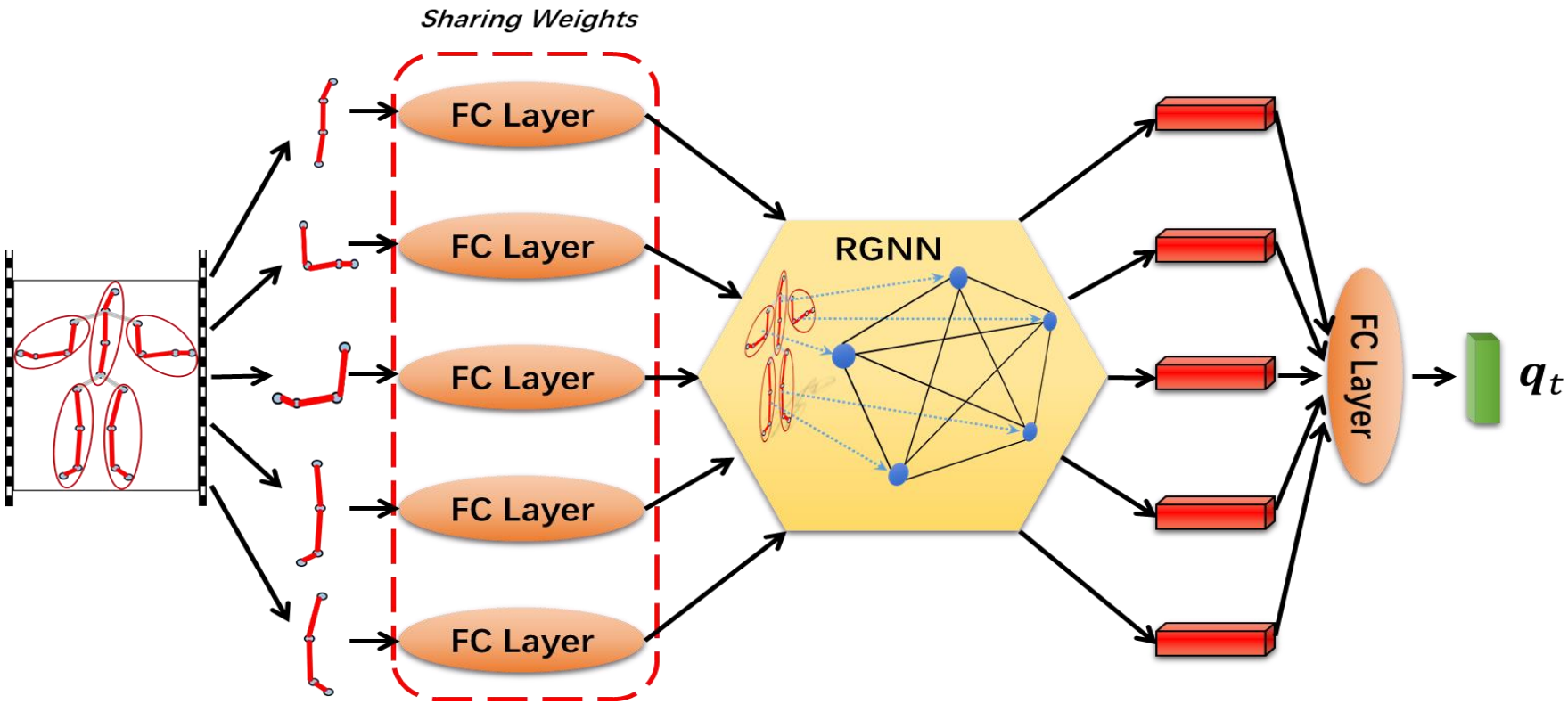
Detailed temporal dynamics

# Skeleton-Based Action Recognition

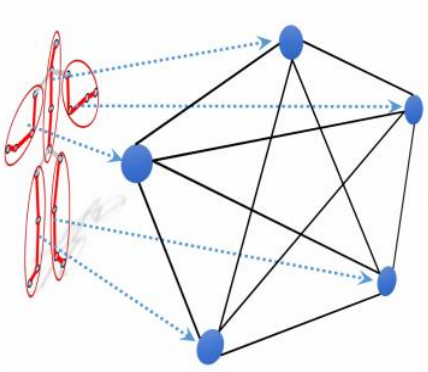


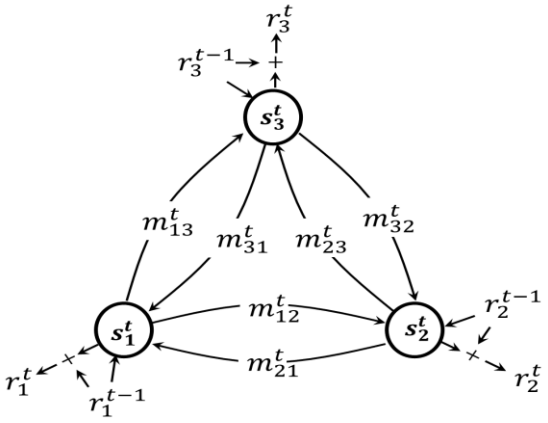

- Propose a spatial reasoning network for each skeleton frame, to capture the **spatial structural information**
- Propose a temporal stack learning network to model the detailed **temporal dynamics of skeleton sequences**

# Spatial Reasoning Network



# Spatial Reasoning Network






Residual graph neural network (RGNN) with three nodes

I. Aggregating the messages:

$$m_k^t = \sum_{i \in \Omega_{v_k}} m_{ik}^t$$

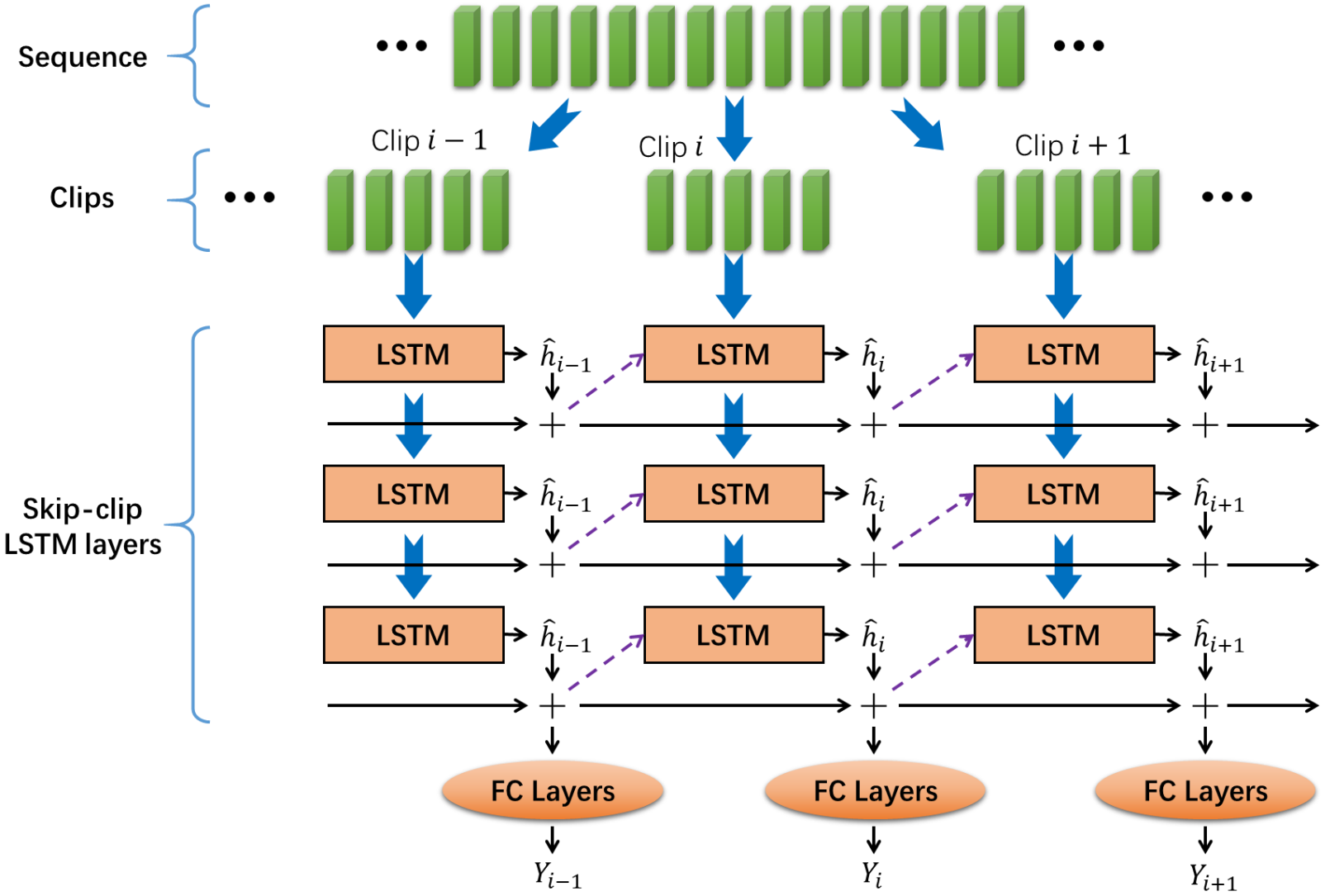
II. Updating hidden state:

$$s_k^t = f_{lstm}(r_k^{t-1}, m_k^t, s_k^{t-1})$$

III. The relation representation:

$$r_k^t = r_k^{t-1} + s_k^t$$

# Temporal Stack Learning Network



# Experiments

Experimental results on NTU RGB+D dataset

| Method | Cross-Subject(%) | Cross-View(%) |
|---|---|---|
| HBRNN-L (CVPR2015) | 59.1 | 64.0 |
| Part-aware LSTM (CVPR2016) | 62.9 | 70.3 |
| Trust Gate ST-LSTM (ECCV2016) | 69.2 | 77.7 |
| Two-stream RNN (CVPR2017) | 71.3 | 79.5 |
| STA-LSTM (AAAI2017) | 73.4 | 81.2 |
| Ensemble TS-LSTM (ICCV2017) | 74.6 | 81.3 |
| Visualization CNN (PR 2017) | 76.0 | 82.6 |
| VA-LSTM (ICCV2017) | 79.4 | 87.6 |
| ST-GCN (AAAI2018) | 81.5 | 88.3 |
| SR-TSL (Ours) | **84.8** | **92.4** |

NTU RGB+D Dataset:

- 60 actions, 40 subjects, 56880 sequences, 25 joints in the skeleton
- Evaluation:
  - Cross-Subject: train (20 subjects), test (20 subject)
  - Cross-View: train (camera 2 and 3), test (camera 1)

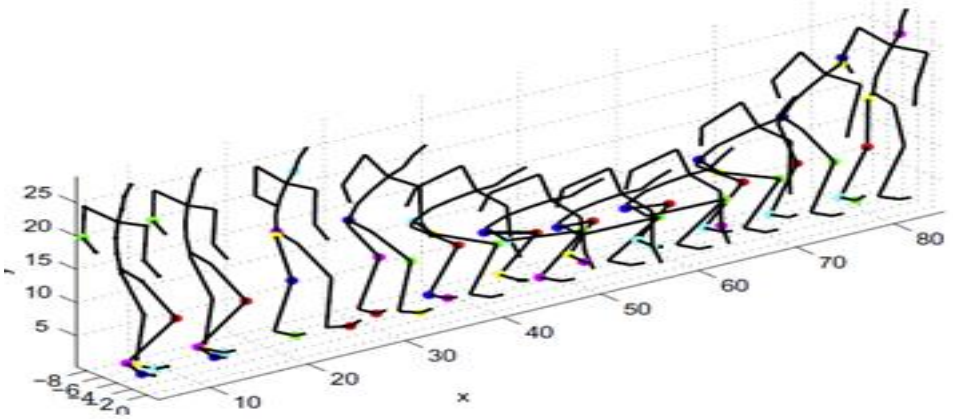
# An Attention Enhanced Graph Convolutional LSTM Network for Skeleton-Based Action Recognition

Chenyang Si, Wentao Chen, Wei Wang, Liang Wang, Tieniu Tan

CVPR, 2019

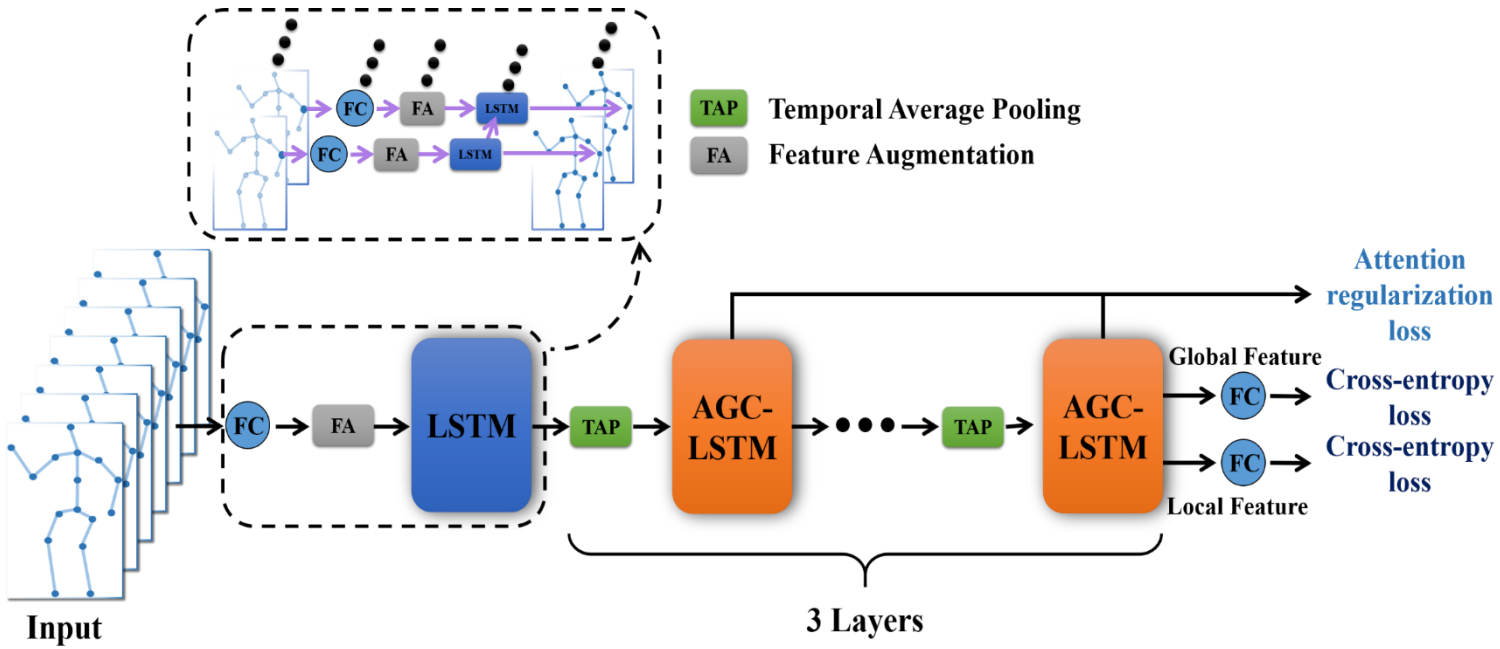
# Skeleton-Based Action Recognition

**Human action** – skeleton sequence



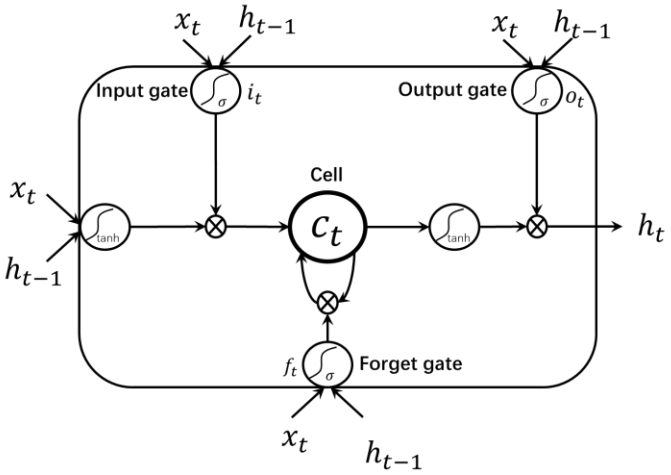

- The skeleton data contains abundant spatial structure information
- Temporal continuity exists not only in the same joints (e.g., hand, wrist and elbow), but also in the body structure
- There is a co-occurrence relationship between spatial and temporal domains

# An Attention Enhanced GC-LSTM



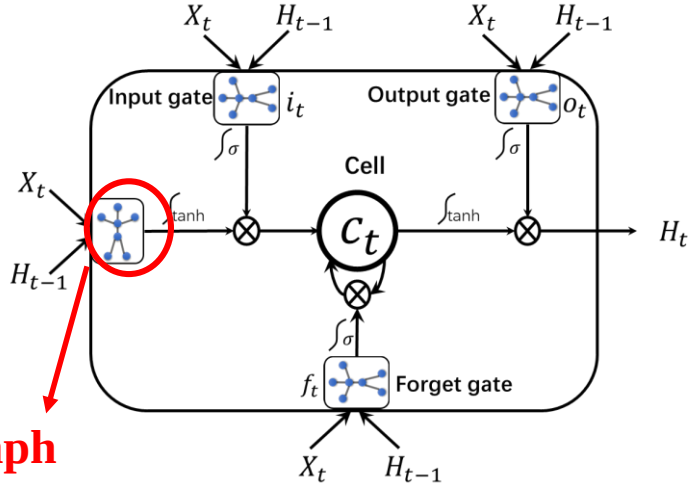

- FA computes feature differences with position features and concatenates both position features and feature differences
- TAP is the implementation of average pooling in the temporal domain
- Three AGC-LSTM layers can model discriminative spatial-temporal features

# Graph Convolutional LSTM




**LSTM**

$$\mathbf{i}_t = \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$$
$$\mathbf{f}_t = \sigma\left(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f\right)$$
$$\mathbf{o}_t = \sigma(\boldsymbol{W}_{xo}\boldsymbol{x}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$$
$$\mathbf{u}_t = tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c)$$
$$\boldsymbol{c}_t = \mathbf{f}_t \odot \boldsymbol{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$
$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tanh(\boldsymbol{c}_t)$$

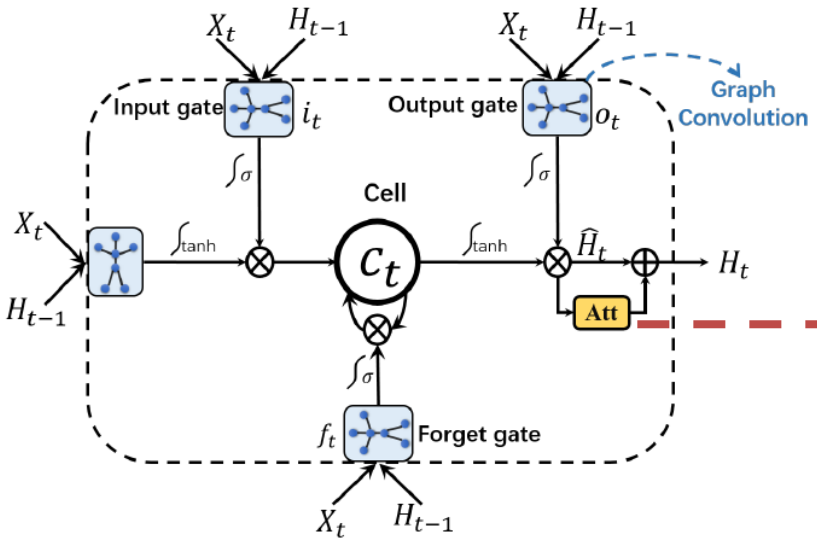



**GC-LSTM**

**Graph Convolution**

$$\mathbf{i}_t = \sigma\left(\boldsymbol{W}_{xi} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hi} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\boldsymbol{W}_{xf} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hf} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\boldsymbol{W}_{xo} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{ho} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_o\right)$$
$$\mathbf{u}_t = tanh\left(\boldsymbol{W}_{xc} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hc} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_c\right)$$
$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$
$$\mathbf{H}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$

# Attention Enhanced GC-LSTM



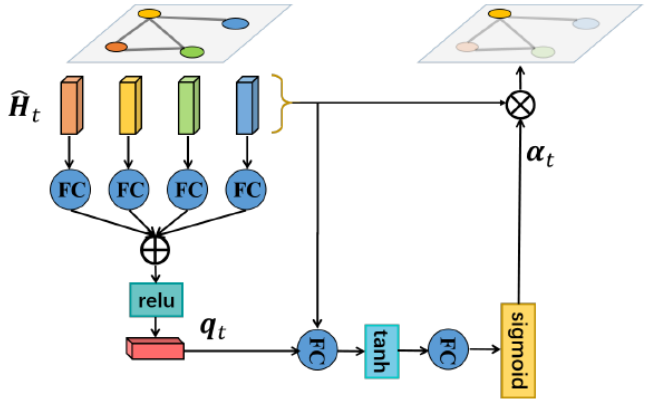




Figure 4. Illustration of the spatial attention network.

$$\mathbf{i}_t = \sigma\left(\boldsymbol{W}_{xi} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hi} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_i\right)$$
$$\mathbf{f}_t = \sigma\left(\boldsymbol{W}_{xf} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hf} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_f\right)$$
$$\mathbf{o}_t = \sigma\left(\boldsymbol{W}_{xo} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{ho} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_o\right)$$
$$\mathbf{u}_t = tanh\left(\boldsymbol{W}_{xc} *_{\mathcal{G}} \boldsymbol{X}_t + \boldsymbol{W}_{hc} *_{\mathcal{G}} \boldsymbol{H}_{t-1} + \boldsymbol{b}_c\right)$$
$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t$$
$$\hat{\mathbf{H}}_t = \mathbf{o}_t \odot \tanh(\mathbf{C}_t)$$
$$\mathbf{H}_t = f_{att}\left(\hat{\mathbf{H}}_t\right) + \hat{\mathbf{H}}_t$$

$$\boldsymbol{q}_t = ReLU\left(\sum_{i=1}^{N} \boldsymbol{W}\hat{\mathbf{H}}_{ti}\right)$$
$$\alpha_t = Sigmoid\left(\boldsymbol{U}_s tanh\left(\boldsymbol{W}_h\hat{\mathbf{H}}_t + \boldsymbol{W}_q\boldsymbol{q}_t + \boldsymbol{b}_s\right)\right)$$
$$\mathbf{H}_{ti} = (1 + \alpha_{ti}) \cdot \hat{\mathbf{H}}_{ti}$$

# Experiments

Experimental results on NTU RGB+D dataset

| Method | Cross-Subject(%) | Cross-View(%) |
|---|---|---|
| STA-LSTM (AAAI2017) | 73.4 | 81.2 |
| Ensemble TS-LSTM (ICCV2017) | 74.6 | 81.3 |
| Visualization CNN (PR 2017) | 76.0 | 82.6 |
| VA-LSTM (ICCV2017) | 79.4 | 87.6 |
| ST-GCN (AAAI2018) | 81.5 | 88.3 |
| SR-TSL(ECCV2018) | 84.8 | 92.4 |
| HCN(IJCAI2018) | 86.5 | 91.1 |
| PB-GCN(BMVC2018) | 87.5 | 93.2 |
| AGC-LSTM | **89.2** | **95.0** |

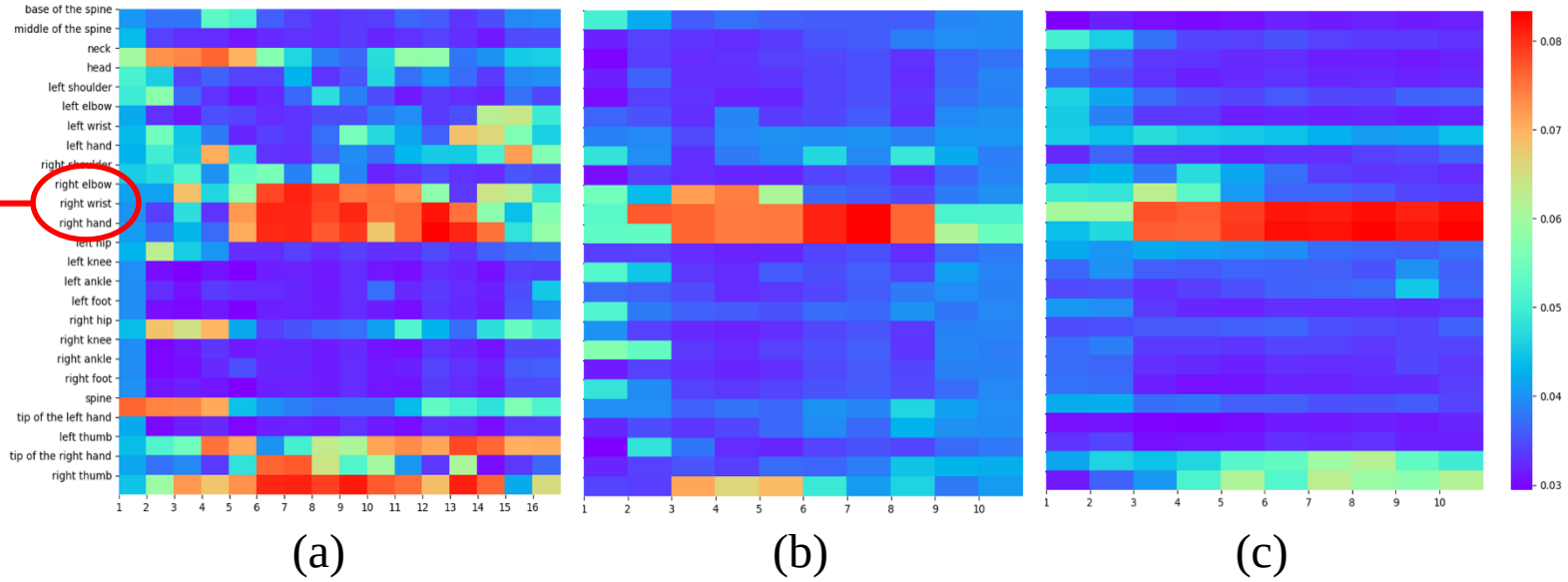
NTU RGB+D Dataset:

- 60 actions, 40 subjects, 56880 sequences, 25 joints in the skeleton
- Evaluation:
  - Cross-Subject: train (20 subjects), test (20 subject)
  - Cross-View: train (camera 2 and 3), test (camera 1)

# Experiments

Visualizations of attention weights of three layers on one actor of the action "handshaking"

Right elbow
Right wrist
Right hand




(a) (b) (c)

Vertical axis denotes the joints. Horizontal axis denotes the frames. (a), (b), (c) are the attention results of the first, second and third AGCLSTM layers, respectively.

# Relational Prototypical Network for Weakly Supervised Temporal Action Localization

Linjiang Huang, Yan Huang, Wanli Ouyang, Liang Wang

AAAI, 2020

# Action Localization

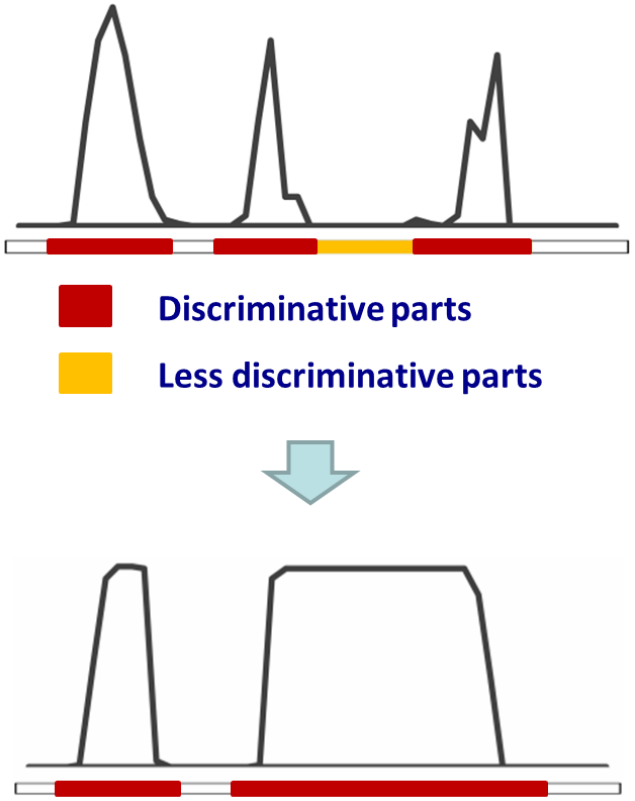
- Action Localization (temporal action localization)
  - untrimmed video
  - predict intervals and labels of actions

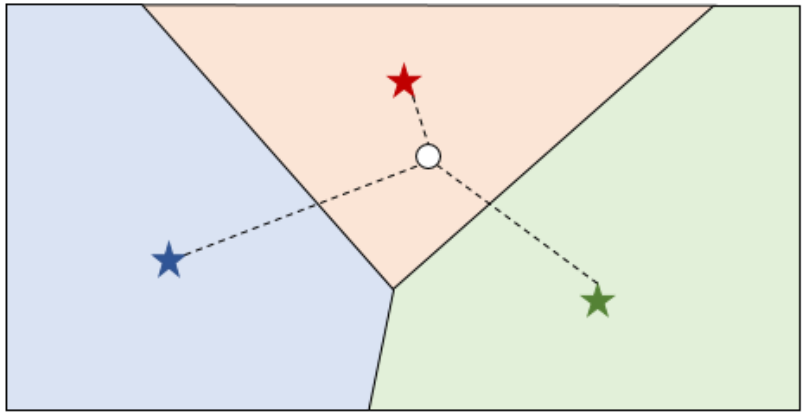
**Longboarding**

**Tumbling**

# Weak Supervisions

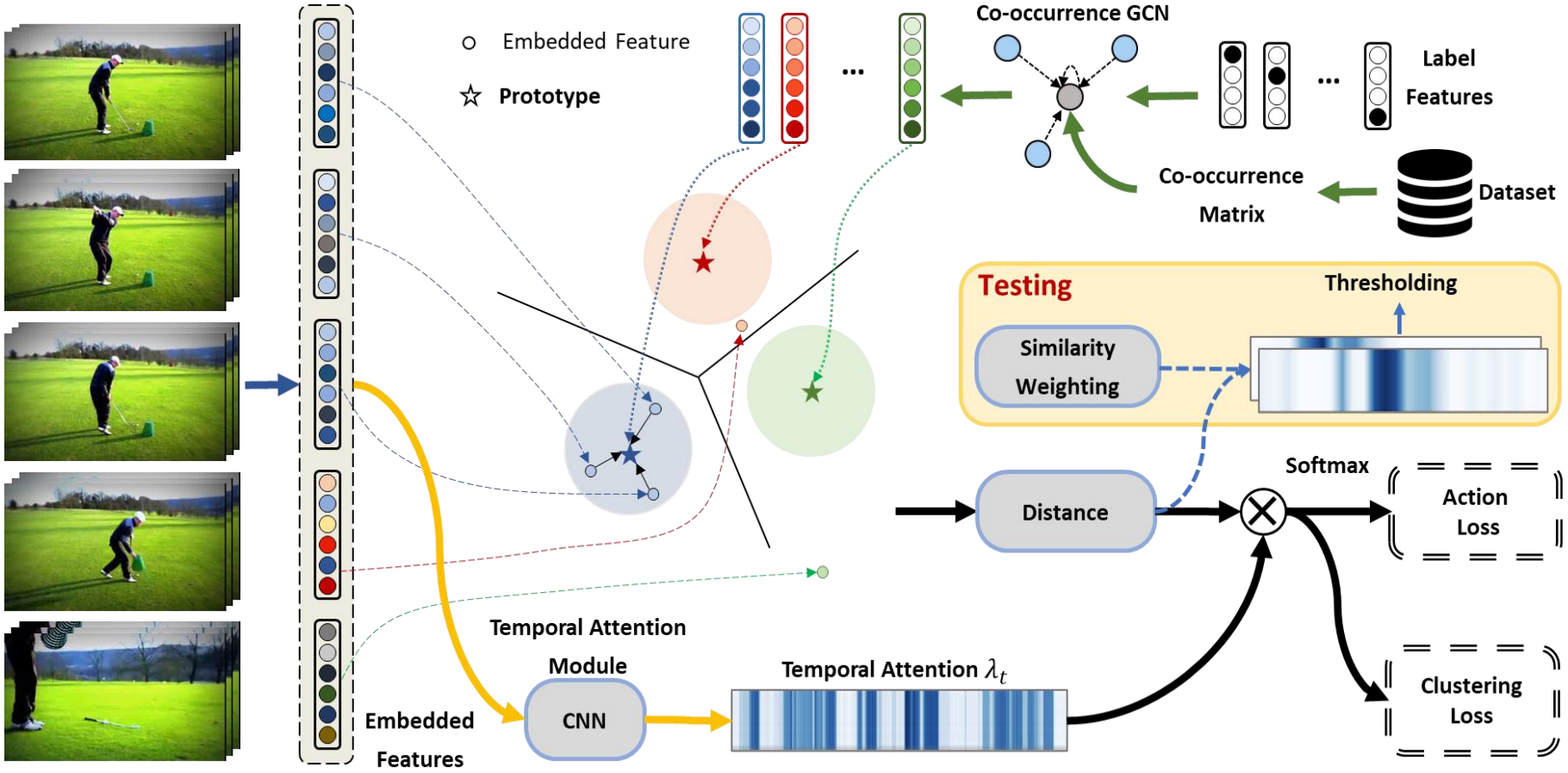
- Learning compact and discriminative features is difficult, due to the imbalance distribution of different actions
- Modeling relations among actions with prototypical network



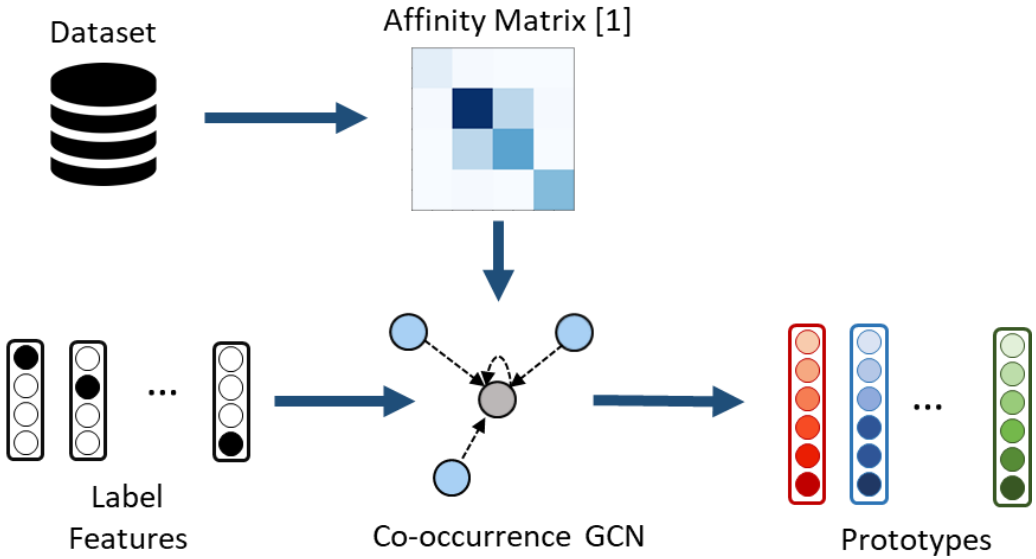



Prototypical network

# Model Architecture



# Prototype Embedding Module



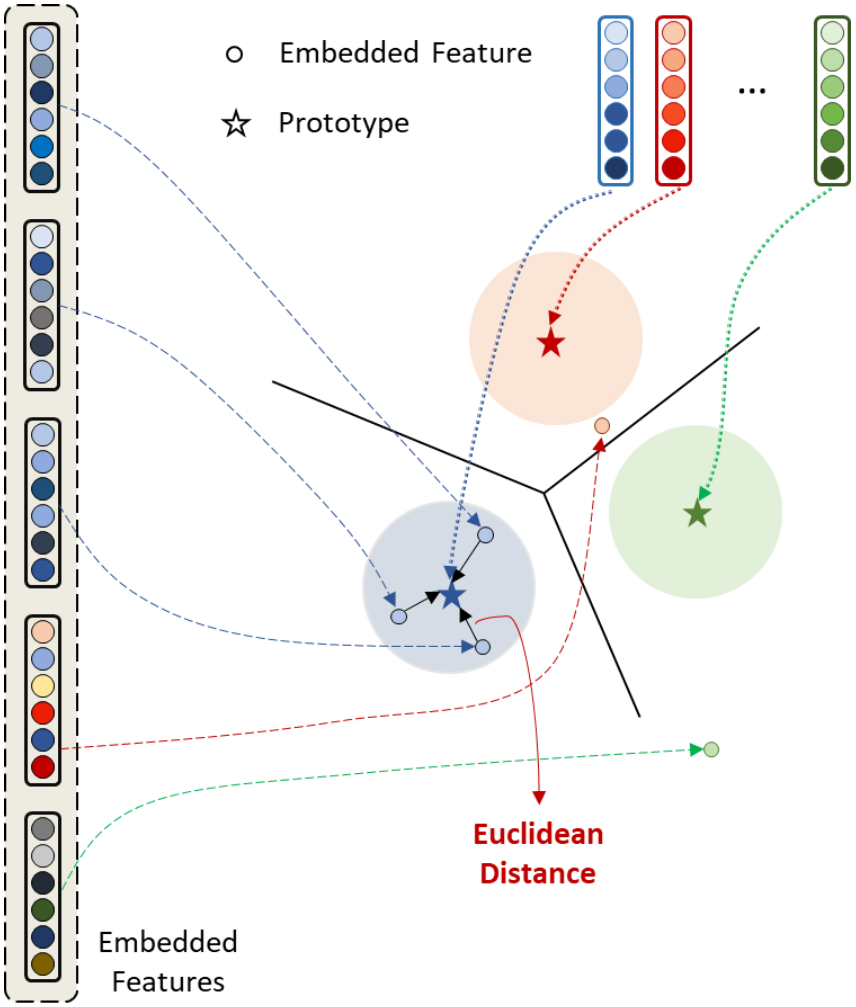

- Affinity matrix derived from statistics in the dataset
- Label features represent different actions
- Co-occurrence GCN captures relations and pulls related prototypes closer

Learning inter-dependent prototypes rather than independent prototypes

$$\text{Prototypes } \boldsymbol{P} = \{\boldsymbol{p}_i\}_{i=1}^{C}$$

[1] Chen, Z.-M.; Wei, X.-S.; Wang, P.; and Guo, Y. Multi-label image recognition with graph convolutional networks, CVPR 2019

# Prototype Matching Module




Negative Euclidean distance is employed as the similarity between feature and prototype

Matching score $s_{tj} = -\|\boldsymbol{x}_e^t - \boldsymbol{p}_j\|_2^2$

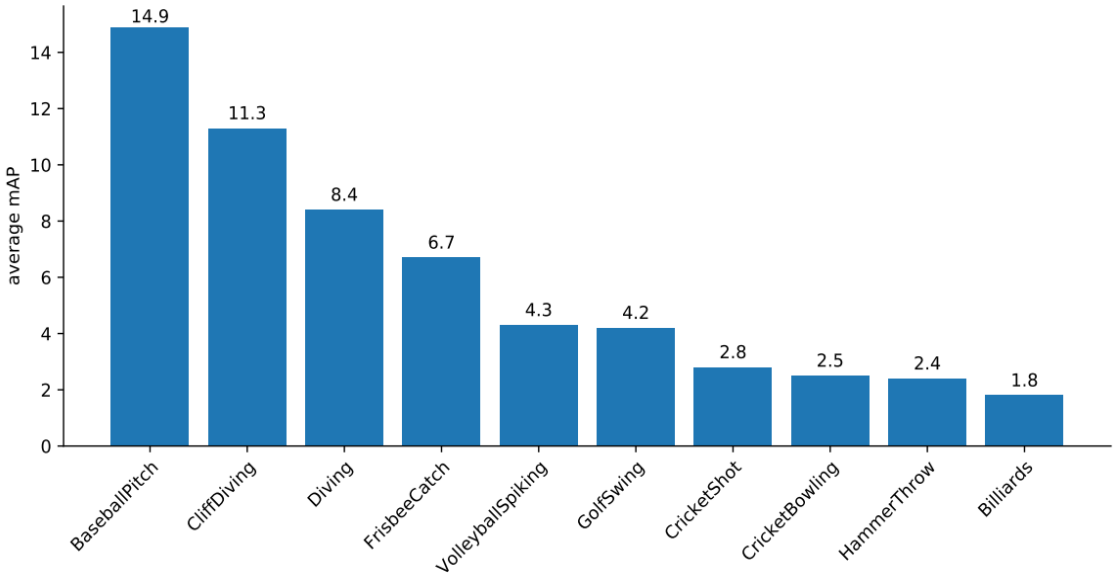
# Experimental Results

Table 2: Results on ActivityNet1.2 validation set. The AVG indicates the average mAP at IoU thresholds 0.5:0.05:0.95.

| Method | AP @ IoU | | | |
|---|---|---|---|---|
| | 0.5 | 0.75 | 0.95 | AVG |
| Step-by-Step Erasion | 27.3 | 14.7 | 2.9 | 15.6 |
| AutoLoc (U) | 27.3 | 15.1 | 3.3 | 16.0 |
| CMCS (U) | 33.9 | 19.9 | 5.1 | 20.5 |
| Ours (U) | **37.0** | **21.1** | **5.2** | **22.0** |
| W-TALC (I) | 37.0 | - | - | 18.0 |
| CMCS (I) | 36.8 | 22.0 | **5.6** | 22.4 |
| Ours (I) | **37.6** | **23.9** | 5.4 | **23.3** |

1.5% ↑

0.9% ↑




Class-specific gain resulting from building relations of actions

# Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course
- Thomas Kipf, University of Cambridge, CompBio Seminar
- Semi-Supervised Classification with Graph Convolutional Networks. T. N. Kipf, M. Welling, ICLR 2017
- Deep Learning for Network Biology : snap.stanford.edu/deepnetbio-ismb
- Representation Learning on Networks, snap.stanford.edu/proj/embeddings-www, WWW 2018

# Next time

- Applications in Deep Learning

# Questions?

# Thank You !