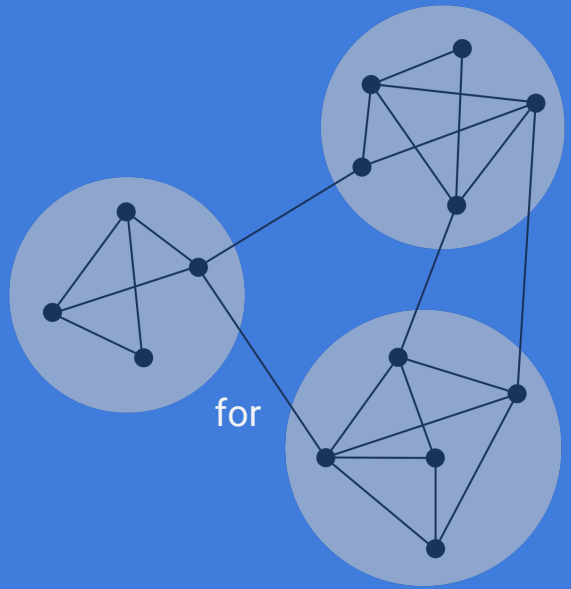
Graph NLP for ConvNets



By Ankit Pal
SHALA2020
https://shala2020.github.io/

Who Am I?

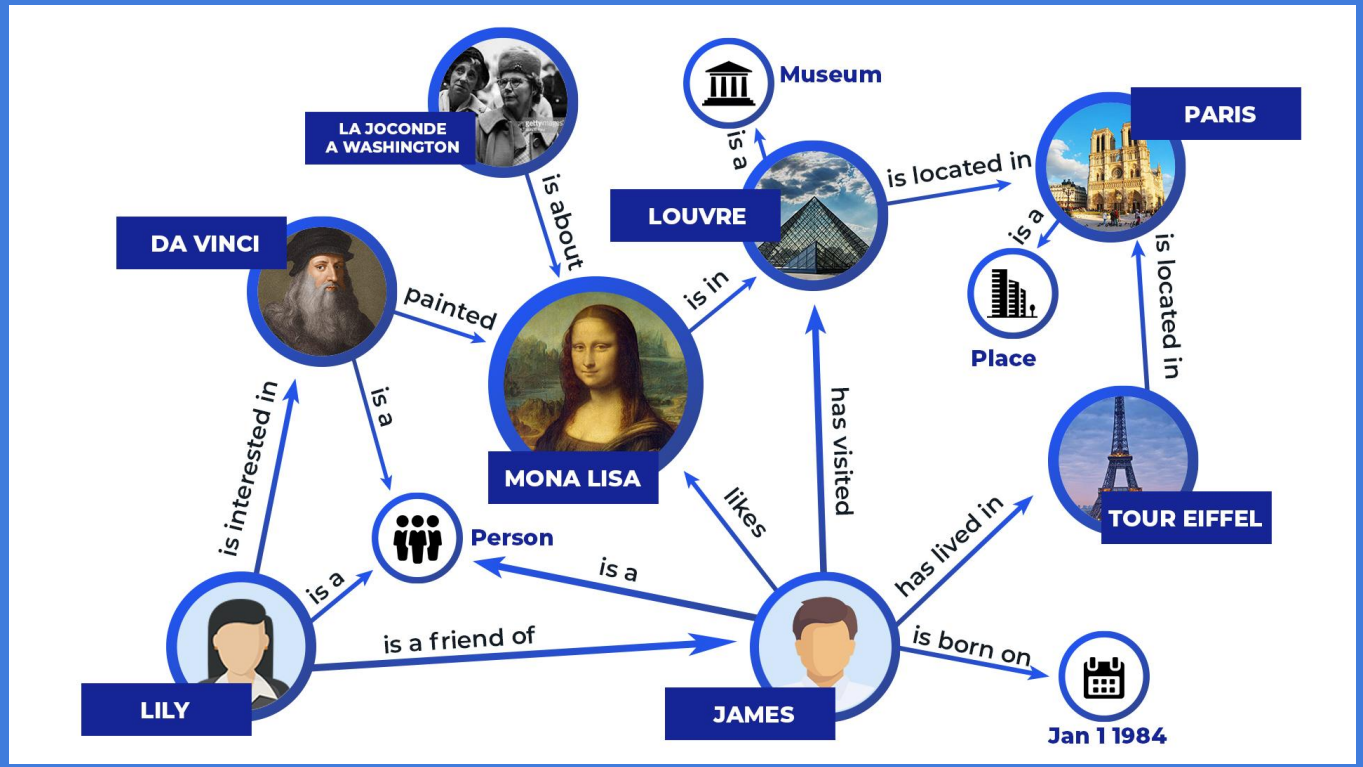
AI Research Engineer @ Saama AI Lab

- I work in Graph and NLP field
- I am interested in
  - Representation Learning on Graphs and Manifolds
  - Unsupervised & Meta learning
  - Neural Architecture Search
  - Spiking Neural networks
  - Biomedical signal processing

Github

Connect the Nodes..

Google Knowledge Graph



Graphs in NLP..

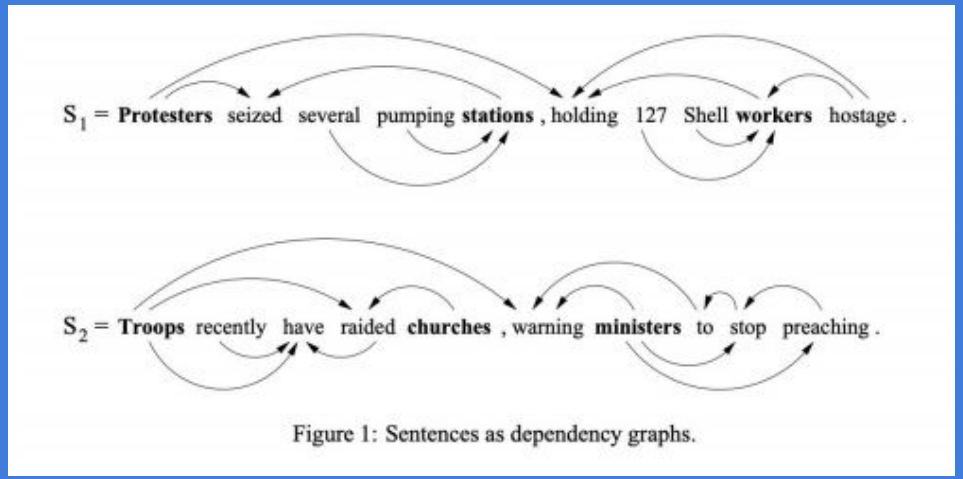
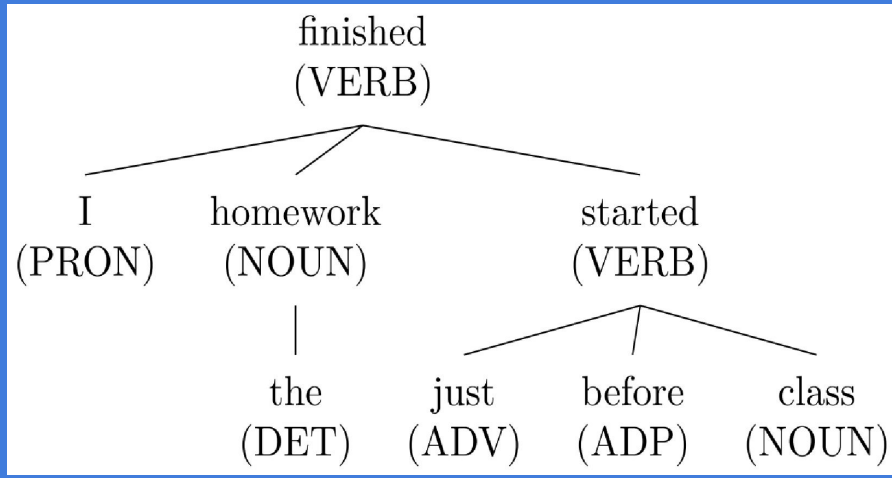


Figure 1: Sentences as dependency graphs.



Why NLP on Graphs?

Rcv1-v2	
Method	**Accuracy**
LR	0.692
SVM	0.691
HSVM	0.693
HLSTM	0.673
RCNN	0.686
XML-CNN	0.695
HAN	0.696
Bi-BloSAN	0.72
DCNN	0.732
SGM+GE	0.719
CAPSULE-B	0.739
CDN-SVM	0.738
HR-DGCNN	0.761
TEXTCNN	0.766
HE-AGCRCNN	0.778
BP-MLL$_{RAD}$	0.780
HTrans	0.805
BOW-CNN	0.827
HilAP	0.833
BERT	0.864
BERT + SGM	0.846
FMP + LaMP$_{pr}$	0.877
MAGNET	**0.885**

Table 4: Comparisons of Micro F1-score for various state-of-the-art models on Rcv1-v2 dataset.

[Pal, Ankit et al. “Multi-Label Text Classification using Attention-based Graph Neural Network.” ICAART (2020)]

Model	20NG
TF-IDF + LR	0.8319 ± 0.0000
CNN-rand	0.7693 ± 0.0061
CNN-non-static	0.8215 ± 0.0052
LSTM	0.6571 ± 0.0152
LSTM (pretrain)	0.7543 ± 0.0172
Bi-LSTM	0.7318 ± 0.0185
PV-DBOW	0.7436 ± 0.0018
PV-DM	0.5114 ± 0.0022
PTE	0.7674 ± 0.0029
fastText	0.7938 ± 0.0030
fastText (bigrams)	0.7967 ± 0.0029
SWEM	0.8516 ± 0.0029
LEAM	0.8191 ± 0.0024
Graph-CNN-C	0.8142 ± 0.0032
Graph-CNN-S	–
Graph-CNN-F	–
Text GCN	**0.8634 ± 0.0009**

[Text-GCN, Yao et al., 2019]

Outline of Today's Lecture

- **Graph Convolutional Networks**
  - Overview of GCN
  - Formulation
- **Hello World in Deep learning on Graph**
  - Implementation of GCN
- **Applications**
  - Text Classification
    - Text-GCN Paper
  - Feature extractor
    - Magnet Paper
  - Knowledge Graph
    - KG Paper

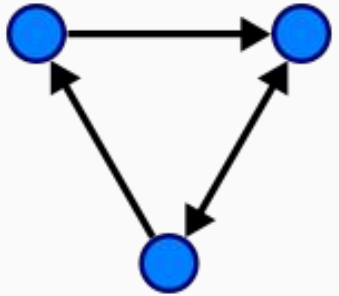
Graph Convolutional Networks

Graph, who?

In Computer Science, a graph is a data structure consisting of two components, vertices and edges. A graph G can be well described by the set of vertices V and edges E it contains.

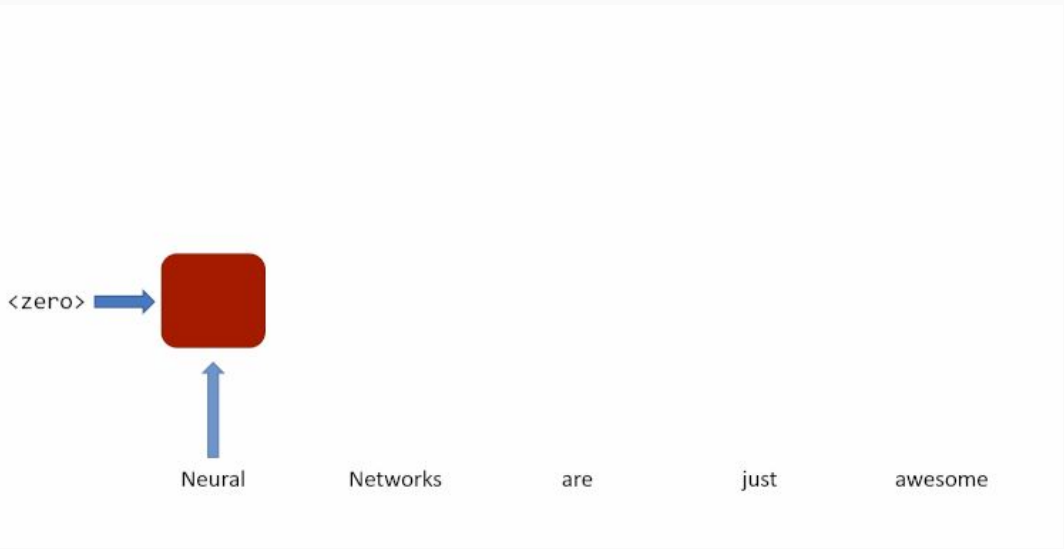
$$G = (V, E)$$

Edges can be either directed or undirected, depending on whether there exist directional dependencies between vertices.



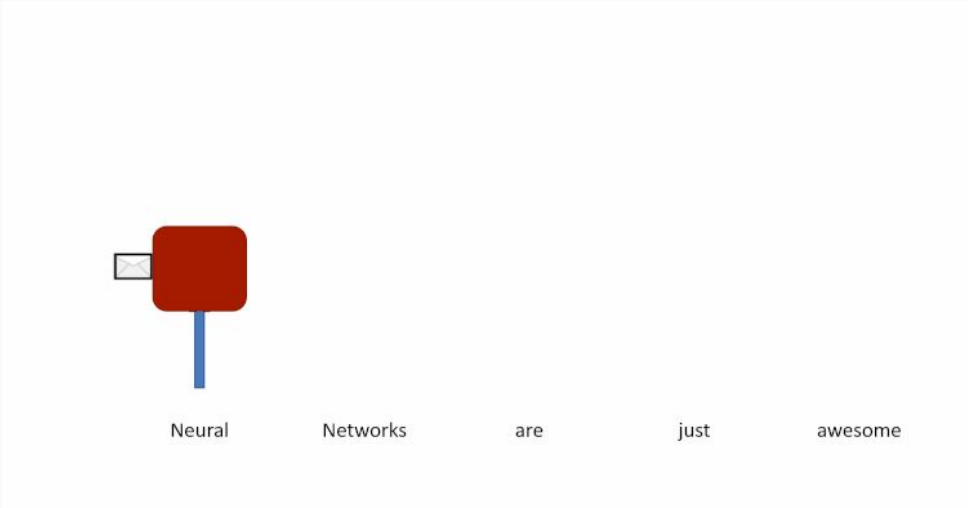
Graph Convolutional Networks

This recurrent simply combines the input — a word for instance — with the output of itself from the previous time-step and passes it through a tanh activation to get the output of the current time-step.



Graph Convolutional Networks

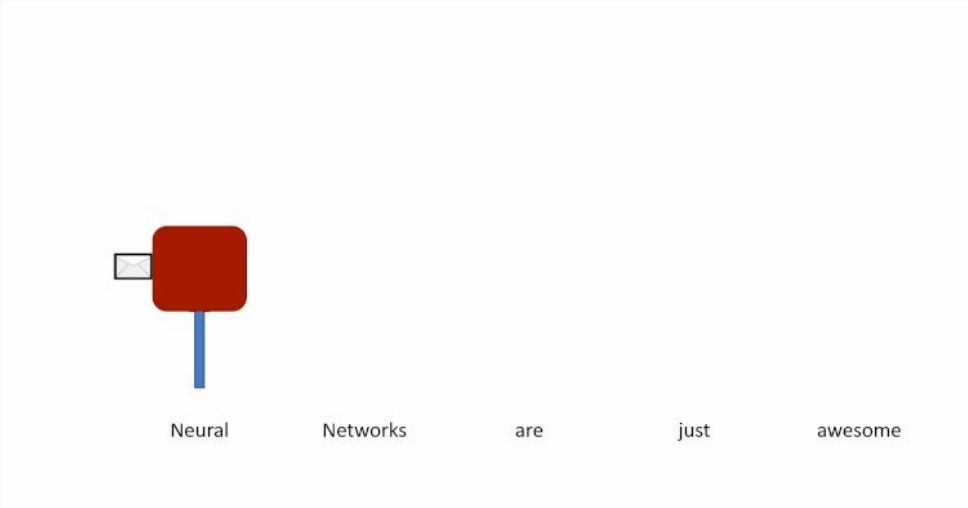
Replace the state with message, Each element in the sequence updates the message to incorporate information about itself so that the message reflects the information in the sequence as it is processed.



Carried state as a message that’s passed between the sequence elements.

Graph Convolutional Networks

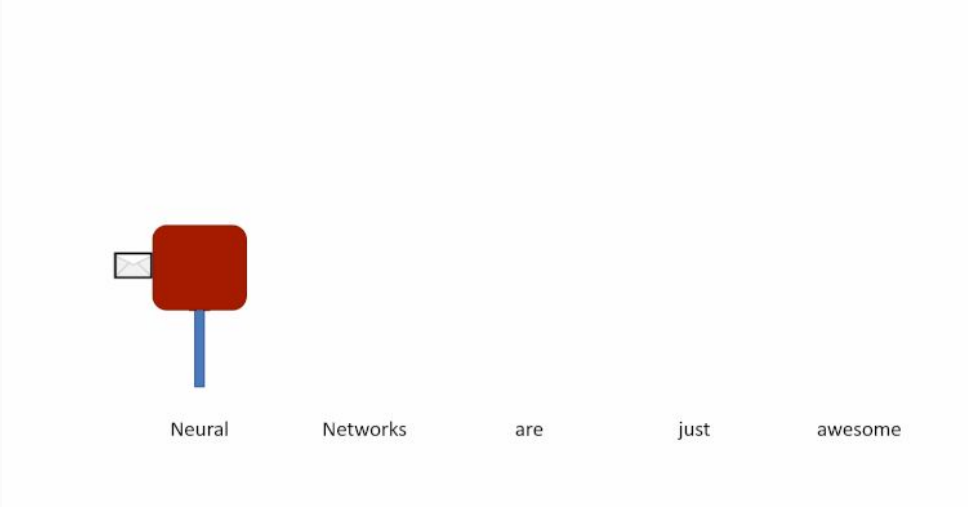
Replace the state with message, Each element in the sequence updates the message to incorporate information about itself so that the message reflects the information in the sequence as it is processed.



Carried state as a message that’s passed between the sequence elements.

Graph Convolutional Networks

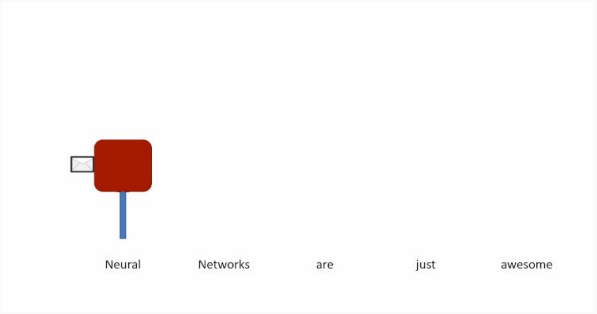
What if an element send a message to the next one on each timestep of the sequence,



Carried state as a message that’s passed between the sequence elements.

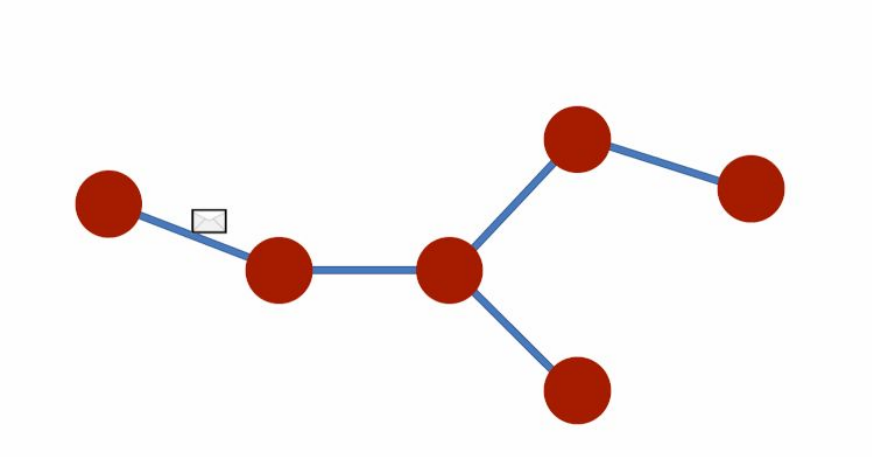
Graph Convolutional Networks

- length of the sequence = Number of time steps
- Essentially, copies of the RNN cell are made over time (unrolling/unfolding), with different inputs at different time steps.
- But graphs can be arbitrarily big and flexible.
- RNN unfold each time step for a message to collect information about it in its entirety. It might seem as though this is a lot of unnecessary work.



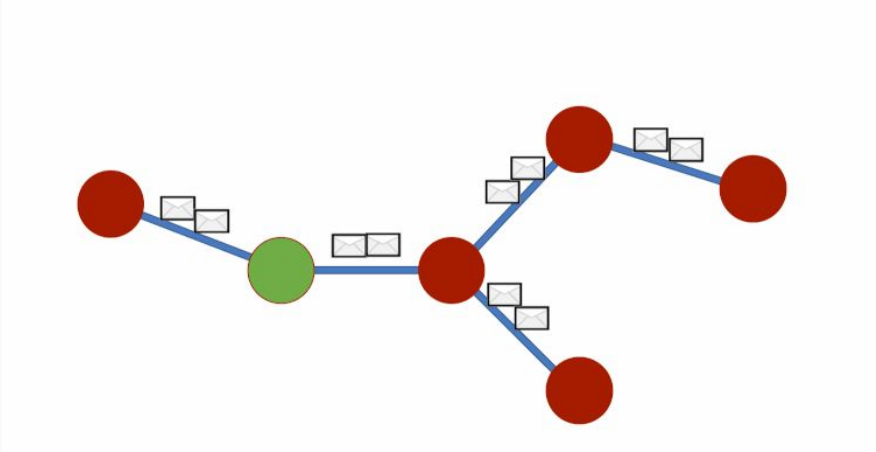
Graph Convolutional Networks

As we applied message passing on sequences, We can apply same idea of message-passing on Graphs

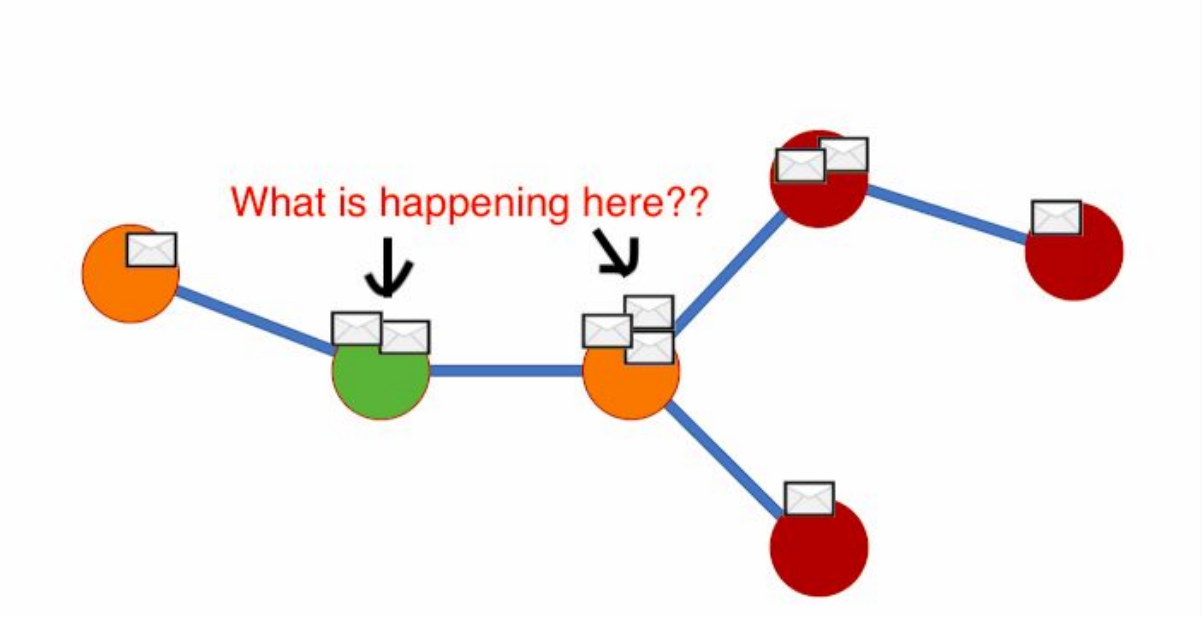


Graph Convolutional Networks

In each GCN layer, all nodes are supposed to receive information from neighboring nodes. Consider each element is passing messages to the elements it connects to — after a few timesteps with exchanges, each node will hold messages that contain information. After Aggregating this information, The representation of each node (each row) is now a sum of its neighbors features!

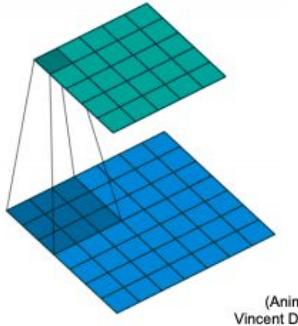


How this Aggregation is performed?

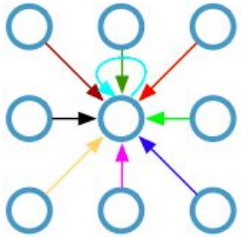


Recap: Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:

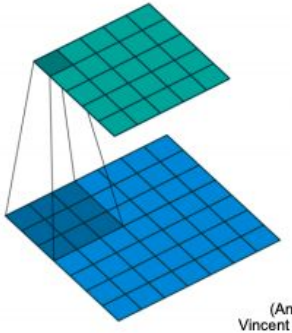


(Animation by Vincent Dumoulin)

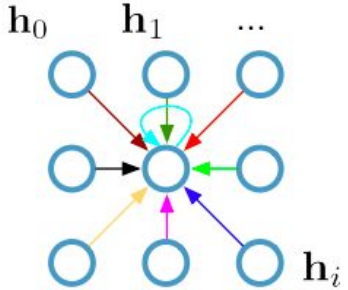


Recap: Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:

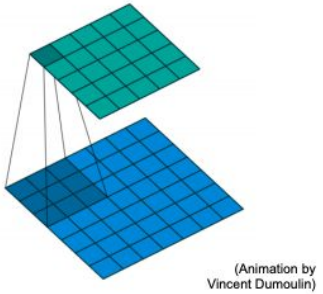


(Animation by Vincent Dumoulin)

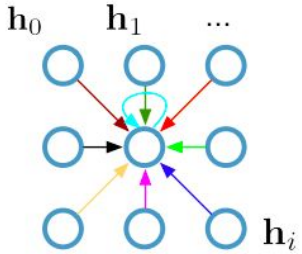


Recap: Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



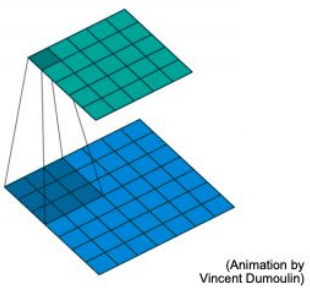
(Animation by Vincent Dumoulin)



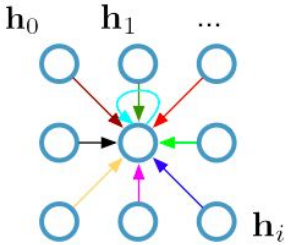
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Recap: Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



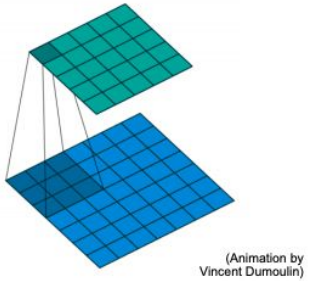
Update for a single pixel:

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

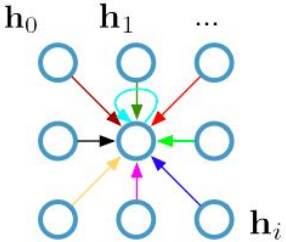
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Recap: Convolutional neural networks (on grids)

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

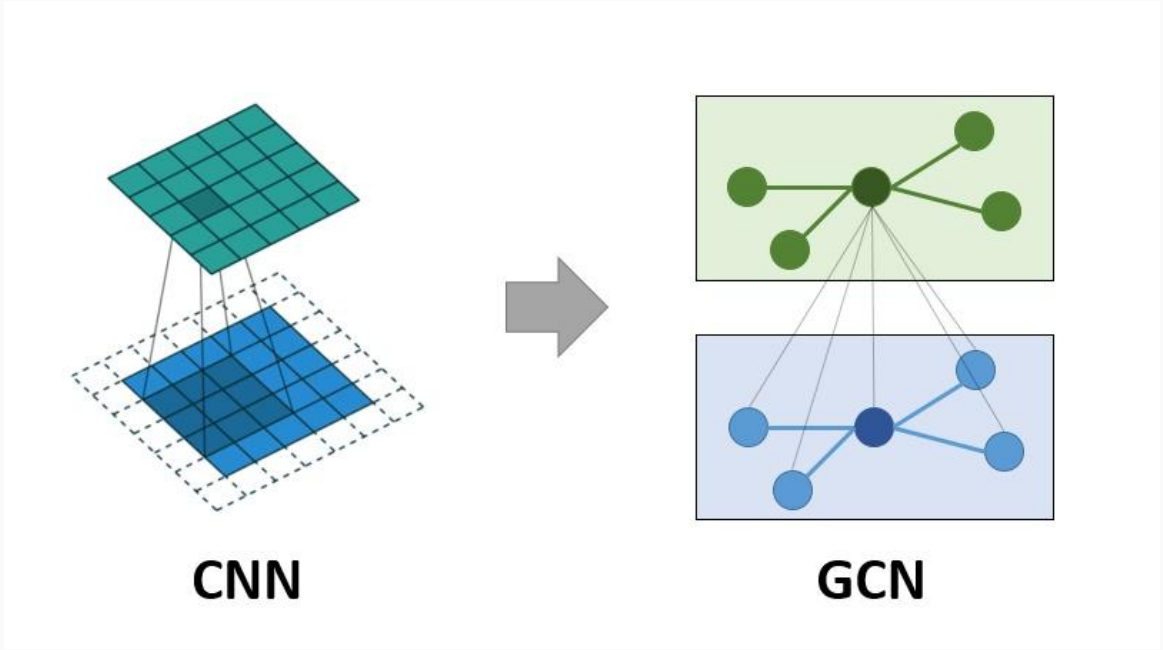
- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

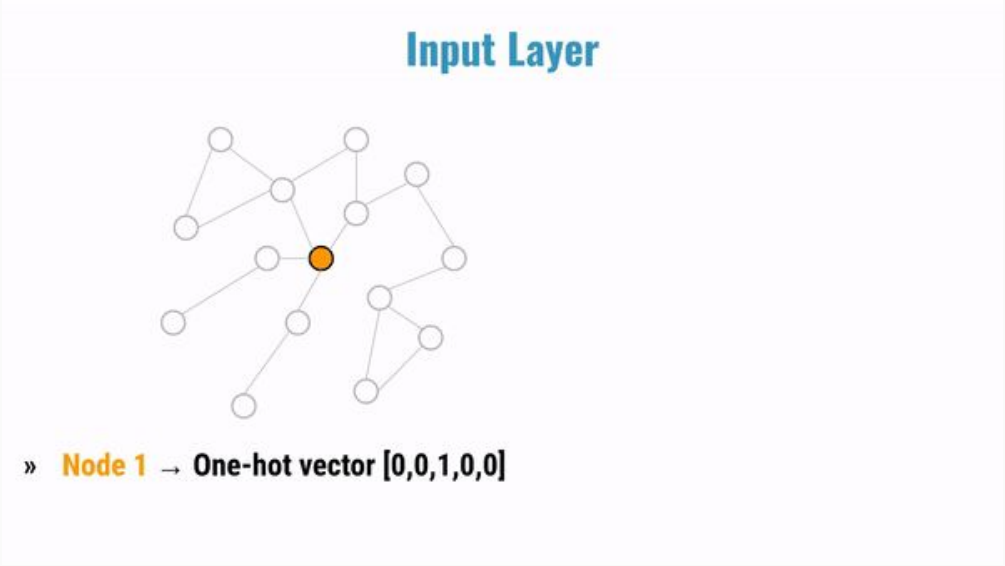
Full update:

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

Comparing



More Animation



Formulation

- GNN formulation by [Kipf et al., ICLR 2016]

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

Filter matrix (Model-parameter)

Initial Node Features

Nodes == Words → Word2vec Embeddings
Nodes == Authors → 0/1 value indicating frequently used keywords
No features → One-hot vector (length = #Nodes)

Formulation

- GNN formulation by [Kipf et al., ICLR 2016]

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

Normalization

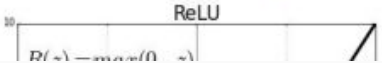
Neighborhood Aggregation

Bias (Model-parameter)

Non-Linearity

Filter matrix (Model-parameter)

Initial Node Features



Formulation

- GNN formulation by [Kipf et al., ICLR 2016]

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

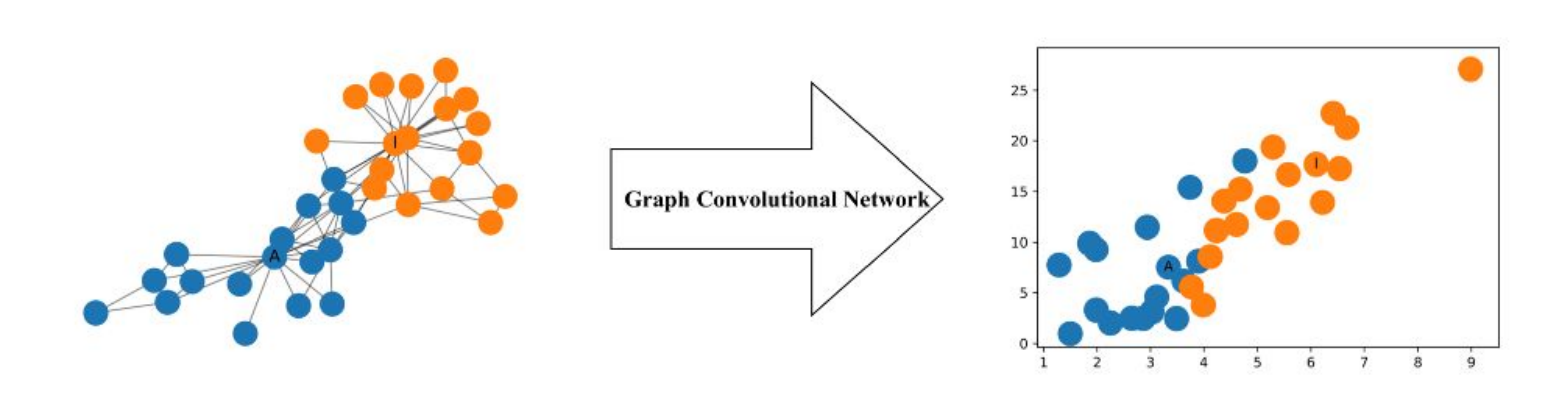
The above formulation is restricted to capturing just 1-hop of nodes

- Stacking K-GNN Layers for capturing K-hop nbd

$$h_v^{k+1} = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W^k h_u^k + b^k\right), \quad \forall v \in \mathcal{V}.$$

Hello World in Deep Learning on Graph

Let’s Understand by practical examples :



Hello World in Deep Learning on Graph

Given a graph G = (V, E), a GCN takes as input

- an input feature matrix $N \times F^0$ feature matrix, $\boldsymbol{X}$, where N is the number of nodes and F^0 is the number of input features for each node, and
- an $N \times N$ matrix representation of the graph structure such as the adjacency matrix $\boldsymbol{A}$ of G.

Hello World in Deep Learning on Graph

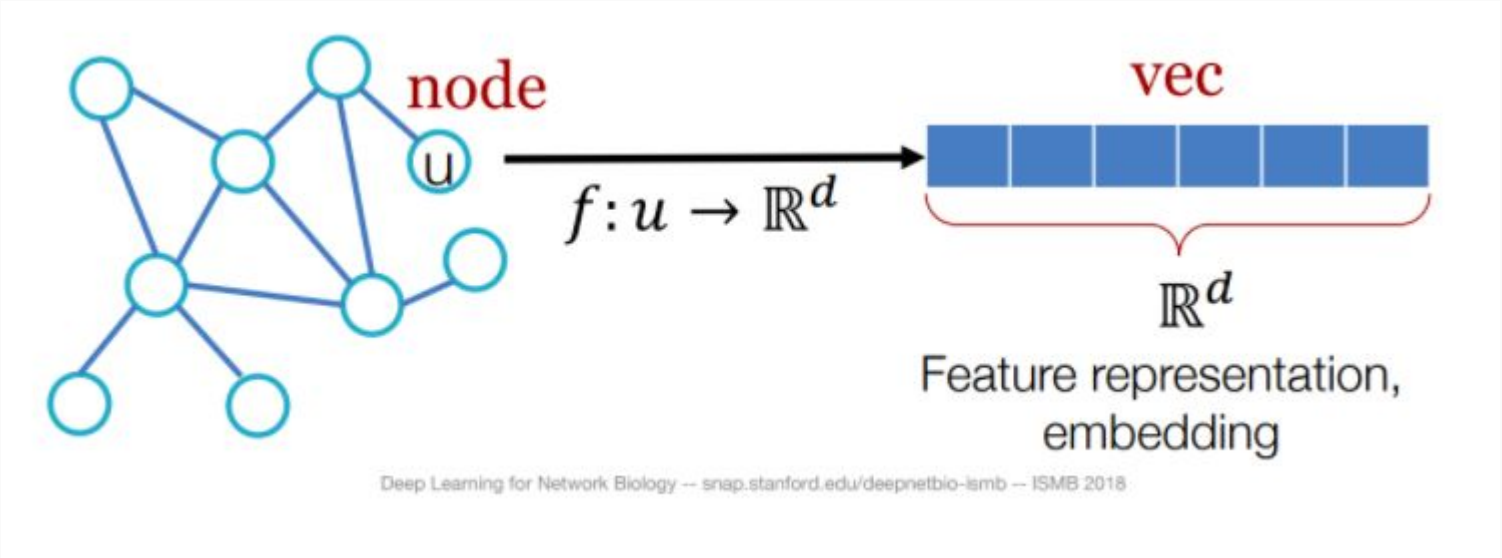
A Simple Propagation Rule

$$f(H^{i}, A) = \sigma(AH^{i}W^{i})$$

where W^{i} is the weight matrix for layer i and σ is a non-linear activation function such as the ReLU function. The weight matrix has dimensions $F^{i} \times F^{i+1}$; in other words the size of the second dimension of the weight matrix determines the number of features at the next layer. this operation is similar to a CNN's filtering operation since these weights are shared across nodes in the graph.

Hello World in Deep Learning on Graph

Every Node is a vector of d dimension



Hello World in Deep Learning on Graph

Jupyter Notebook explanation

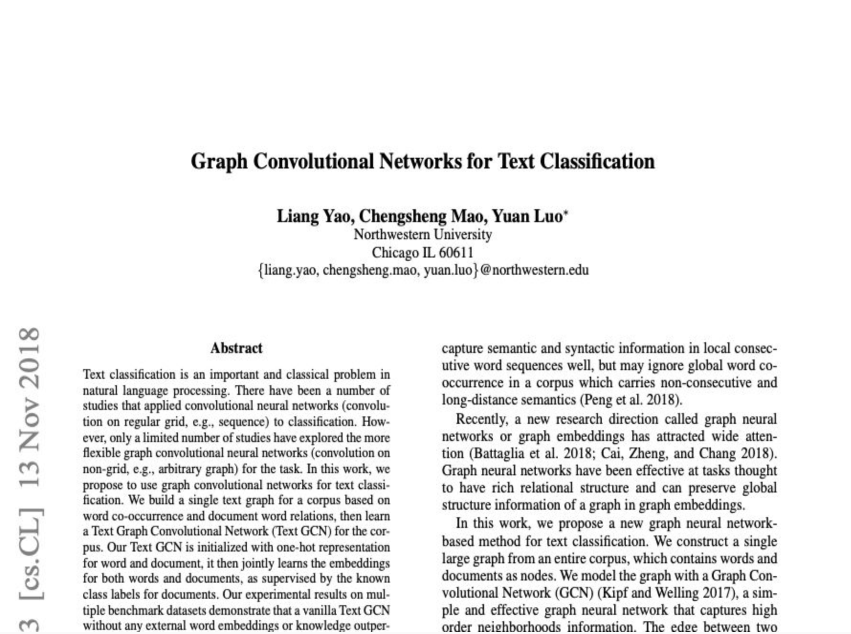
Key Points from Notebook

- In Graph based Models, using the Adjacency Matrix(A) enable the model to learn the features of neighboring nodes.
- Thomas Kipf and Max Welling (2017) Used the Renormalization trick to normalize the features in Fast Approximate Spectral-based Graph Convolutional Networks.
- GCNs can learn features representation to cluster even before training.

Applications
Text Classification using GCN

Key Points of Research Paper

- Embedding the entire corpus into a single graph with documents
- some labelled and some unlabelled
- words as nodes, with each document-word and word-word edges
- Assigning weights based on their relationships



Graph Convolutional Networks for Text Classification

Liang Yao, Chengsheng Mao, Yuan Luo*
Northwestern University
Chicago IL 60611
{liang.yao, chengsheng.mao, yuan.luo}@northwestern.edu

Abstract

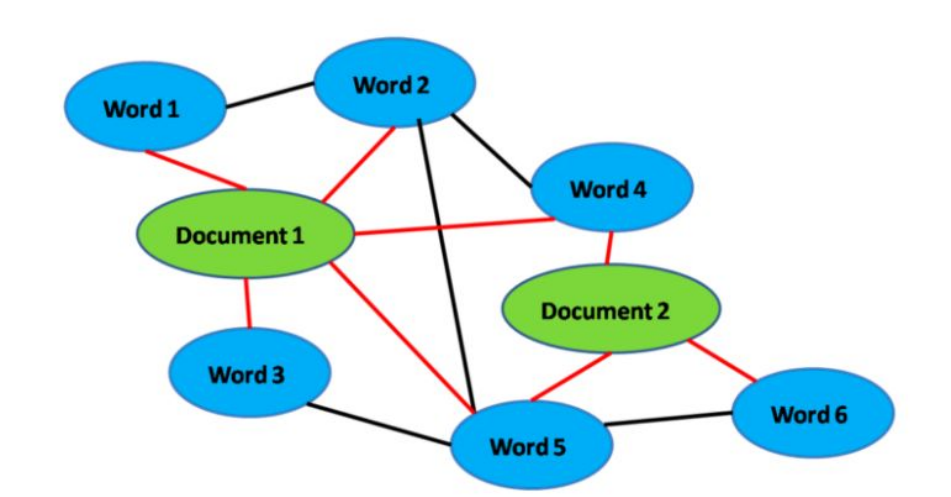
Text classification is an important and classical problem in natural language processing. There have been a number of studies that applied convolutional neural networks (convolution on regular grid, e.g., sequence) to classification. However, only a limited number of studies have explored the more flexible graph convolutional neural networks (convolution on non-grid, e.g., arbitrary graph) for the task. In this work, we propose to use graph convolutional networks for text classification. We build a single text graph for a corpus based on word co-occurrence and document word relations, then learn a Text Graph Convolutional Network (Text GCN) for the corpus. Our Text GCN is initialized with one-hot representation for word and document, it then jointly learns the embeddings for both words and documents, as supervised by the known class labels for documents. Our experimental results on multiple benchmark datasets demonstrate that a vanilla Text GCN without any external word embeddings or knowledge outper-

capture semantic and syntactic information in local consecutive word sequences well, but may ignore global word co-occurrence in a corpus which carries non-consecutive and long-distance semantics (Peng et al. 2018).

Recently, a new research direction called graph neural networks or graph embeddings has attracted wide attention (Battaglia et al. 2018; Cai, Zheng, and Chang 2018). Graph neural networks have been effective at tasks thought to have rich relational structure and can preserve global structure information of a graph in graph embeddings.

In this work, we propose a new graph neural network-based method for text classification. We construct a single large graph from an entire corpus, which contains words and documents as nodes. We model the graph with a Graph Convolutional Network (GCN) (Kipf and Welling 2017), a simple and effective graph neural network that captures high order neighborhoods information. The edge between two

Applications
Convert Text data into Graph



Corpus represented as a heterogeneous graph . Red lines represent document-word edges weighted by TF-IDF, Black lines represent word-word edges weighted by PMI.

Text dataset to Graph

Eight steps to convert Text dataset into Graph

1. Text Preprocessing on Raw Corpus
2. Building Vocabulary from Preprocessed dataset
3. Creating n-gram set from Preprocessed dataset
4. Count single word freq in N-gram
5. Count Bigram frequency in N-gram
6. Calculate tf-idf for single word frequency
7. Calculate PMI score for Bigram frequency
8. Embedding the entire corpus using Heterogeneous Graph

Text Preprocessing on Raw Corpus

We are using IMDB Review Dataset, Using only few rows (sentences) to understand the process better

Raw Dataset

	review	sentiment
0	How did Mike Hammer live - in a penthouse with...	negative
1	I had been looking forward to seeing Dreamgirl...	negative
2	Had I checked IMDb BEFORE renting this DVD fro...	negative
3	A very great movie. A big love stor...	positive
4	I found the documentary entitled Fast, Cheap, ...	negative

Preprocessed Dataset

	review	sentiment	encoded_labels
0	how did mike hammer live in a penthouse with a...	negative	0
1	i had been looking forward to seeing dreamgirl...	negative	0
2	had i checked imdb before renting this dvd fro...	negative	0
3	a very great moviea big love story lots of swo...	positive	1
4	i found the documentary entitled fast cheap an...	negative	0

Text Preprocessing : Remove HTML tags, remove special characters, make lowercase, etc...

Building Vocabulary from Corpus

Making a list of unique words from preprocessed dataset.

	review	sentiment	encoded_labels
0	how did mike hammer live in a penthouse with a...	negative	0
1	i had been looking forward to seeing dreamgirl...	negative	0
2	had i checked imdb before renting this dvd fro...	negative	0
3	a very great moviea big love story lots of swo...	positive	1
4	i found the documentary entitled fast cheap an...	negative	0

```
['years', 'then', 'couch', 'story', 'fighting', 'live', 'chinese', 'movie', 'unfiltered', 'imdb', 'interesting', 'cat
hode', 'whole', 'moviea', 'nah', 'bottle', 'downtown', 'care', 'must', 'credits', 'im', 'battle', 'tube', 'me', 'bi
g', 'say', 'tenutas', 'dynasty', 'was', 'sure', 'as', 'writter', 'desperate', '14', 'paperbacks', 'states', 'a', 'eve
r', 'reviews', 'named', 'wits', 'singing', 'mean', 'eg', 'china', 'the', 'han', 'thats', 'do', 'wrote', 'corner', 'tw
o', 'not', 'very', 'floor', 'overall', 'more', 'an', 'had', 'dvd', 'those', 'in', 'on', 'tv', 'same', 'did', 'la', 'b
etween', 'is', 'hours', 'him', 'thinks', 'it', 'love', 'led', 'see', 'of', 'army', 'scenes', 'twenty', 'trash', 'lous
y', 'all', 'music', 'were', 'historyfew', 'forward', 'judyism', 'off', 'even', 'golf', 'frankly', 'that', 'huge', 'se
riously', 'routines', 'quarter', 'checked', 'according', 'real']
```

Creating n-gram set from Vocabulary

Creating a set of N-gram from Preprocessed dataset
For demo purpose, let's use n = 4

	review	sentiment	encoded_labels
0	how did mike hammer live in a penthouse with a...	negative	0
1	i had been looking forward to seeing dreamgirl...	negative	0
2	had i checked imdb before renting this dvd fro...	negative	0
3	a very great moviea big love story lots of swo...	positive	1
4	i found the documentary entitled fast cheap an...	negative	0

```
[['how', 'did', 'mike', 'hammer'], ['did', 'mike', 'hammer', 'live'], ['mike', 'hammer', 'live', 'in'], ['hammer',
'live', 'in', 'a'], ['live', 'in', 'a', 'penthouse'], ['in', 'a', 'penthouse', 'with'], ['a', 'penthouse', 'with',
'a'], ['penthouse', 'with', 'a', 'golf'], ['with', 'a', 'golf', 'bag'], ['a', 'golf', 'bag', 'stashed'], ['golf', 'ba
g', 'stashed', 'in'], ['bag', 'stashed', 'in', 'the'], ['stashed', 'in', 'the', 'corner'], ['in', 'the', 'corner', 'n
ext'], ['the', 'corner', 'next', 'to'], ['corner', 'next', 'to', 'a'], ['next', 'to', 'a', 'big'], ['to', 'a', 'big',
'screen'], ['a', 'big', 'screen', 'cathode'], ['big', 'screen', 'cathode', 'ray']]
```

Count single word frequency in n-gram

How many times a word appears in n-gram where n = 4

```
[['how', 'did', 'mike', 'hammer'], ['did', 'mike', 'hammer', 'live'], ['mike', 'hammer', 'live', 'in'], ['hammer',
'live', 'in', 'a'], ['live', 'in', 'a', 'penthouse'], ['in', 'a', 'penthouse', 'with'], ['a', 'penthouse', 'with',
'a'], ['penthouse', 'with', 'a', 'golf'], ['with', 'a', 'golf', 'bag'], ['a', 'golf', 'bag', 'stashed'], ['golf', 'ba
g', 'stashed', 'in'], ['bag', 'stashed', 'in', 'the'], ['stashed', 'in', 'the', 'corner'], ['in', 'the', 'corner', 'n
ext'], ['the', 'corner', 'next', 'to'], ['corner', 'next', 'to', 'a'], ['next', 'to', 'a', 'big'], ['to', 'a', 'big',
'screen'], ['a', 'big', 'screen', 'cathode'], ['big', 'screen', 'cathode', 'ray']]
```

```
{'how': 1, 'did': 2, 'mike': 3, 'hammer': 4, 'live': 4, 'in': 4, 'a': 4, 'penthouse': 4, 'with': 4, 'golf': 4, 'bag':
4, 'stashed': 4, 'the': 4, 'corner': 4, 'next': 4, 'to': 4, 'big': 4, 'screen': 3, 'cathode': 2, 'ray': 1}
```

Count bigram frequency in n-gram

How many times a bigram appears in n-gram where n = 4

```
[['how', 'did', 'mike', 'hammer'], ['did', 'mike', 'hammer', 'live'], ['mike', 'hammer', 'live', 'in'], ['hammer',
'live', 'in', 'a'], ['live', 'in', 'a', 'penthouse'], ['in', 'a', 'penthouse', 'with'], ['a', 'penthouse', 'with',
'a'], ['penthouse', 'with', 'a', 'golf'], ['with', 'a', 'golf', 'bag'], ['a', 'golf', 'bag', 'stashed'], ['golf', 'ba
g', 'stashed', 'in'], ['bag', 'stashed', 'in', 'the'], ['stashed', 'in', 'the', 'corner'], ['in', 'the', 'corner', 'n
ext'], ['the', 'corner', 'next', 'to'], ['corner', 'next', 'to', 'a'], ['next', 'to', 'a', 'big'], ['to', 'a', 'big',
'screen'], ['a', 'big', 'screen', 'cathode'], ['big', 'screen', 'cathode', 'ray']]
```

```
{'did,how': 1, 'how,did': 1, 'mike,how': 1, 'how,mike': 1, 'mike,did': 2, 'did,mike': 2, 'hammer,how': 1, 'how,hamme
r': 1, 'hammer,did': 2, 'did,hammer': 2, 'hammer,mike': 3, 'mike,hammer': 3, 'live,how': 1, 'how,live': 1, 'live,di
d': 2, 'did,live': 2, 'live,mike': 3, 'mike,live': 3, 'live,hammer': 4, 'hammer,live': 4, 'in,how': 2, 'how,in': 2,
'in,did': 4, 'did,in': 4, 'in,mike': 6, 'mike,in': 6, 'in,hammer': 8, 'hammer,in': 8, 'in,live': 8, 'live,in': 8, 'a,
how': 3, 'how,a': 3, 'a,did': 6, 'did,a': 6, 'a,mike': 9, 'mike,a': 9, 'a,hammer': 12, 'hammer,a': 12, 'a,live': 12,
'live,a': 12, 'a,in': 24, 'in,a': 24, 'penthouse,how': 1, 'how,penthouse': 1, 'penthouse,did': 2, 'did,penthouse': 2,
'penthouse,mike': 3, 'mike,penthouse': 3, 'penthouse,hammer': 4, 'hammer,penthouse': 4}
```

Tf-idf : Term frequency–inverse document frequency

Tf-idf is a statistical measure used to evaluate how important a word is to a document in a collection or corpus.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

PMI : Pointwise mutual Information

Mutual Information has to do with the amount of information word W_1 adds to the word W_2

$$I(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)} = \log_2 \frac{C(w_1, w_2)N}{C(w_1)C(w_2)}$$

A high positive PMI between pairs of words means that they have a high semantic correlation, conversely we do not build edges between words with negative PMI.

Overall, TF-IDF-weighted document-word edges capture within-document context, while PMI-weighted word-word edges (which can span across documents) capture across-document contexts.

PMI : Pointwise mutual Information

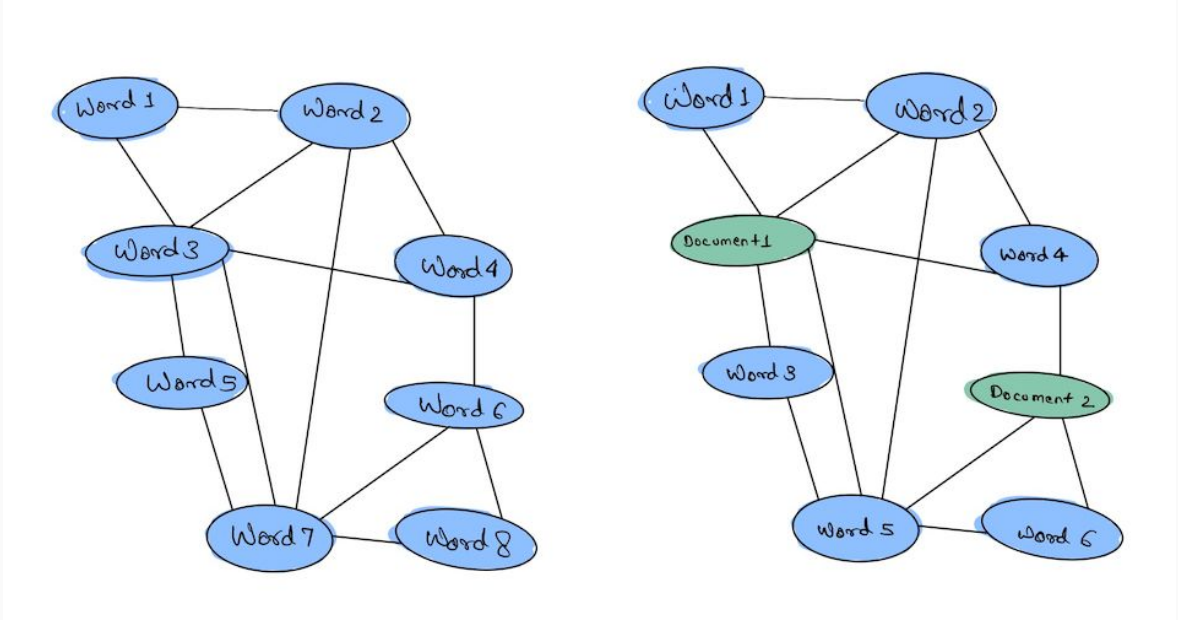

```
{'did,how': 1, 'how,did': 1, 'mike,how': 1, 'how,mike': 1, 'mike,did': 2, 'did,mike': 2, 'hammer,how': 1, 'how,hamme
r': 1, 'hammer,did': 2, 'did,hammer': 2, 'hammer,mike': 3, 'mike,hammer': 3, 'live,how': 1, 'how,live': 1, 'live,di
d': 2, 'did,live': 2, 'live,mike': 3, 'mike,live': 3, 'live,hammer': 4, 'hammer,live': 4, 'in,how': 2, 'how,in': 2,
'in,did': 4, 'did,in': 4, 'in,mike': 6, 'mike,in': 6, 'in,hammer': 8, 'hammer,in': 8, 'in,live': 8, 'live,in': 8, 'a,
how': 3, 'how,a': 3, 'a,did': 6, 'did,a': 6, 'a,mike': 9, 'mike,a': 9, 'a,hammer': 12, 'hammer,a': 12, 'a,live': 12,
'live,a': 12, 'a,in': 24, 'in,a': 24, 'penthouse,how': 1, 'how,penthouse': 1, 'penthouse,did': 2, 'did,penthouse': 2,
'penthouse,mike': 3, 'mike,penthouse': 3, 'penthouse,hammer': 4, 'hammer,penthouse': 4}
```


$$I(w_1, w_2) = \log_2 \frac{P(w_1, w_2)}{P(w_1)P(w_2)} = \log_2 \frac{C(w_1, w_2)N}{C(w_1)C(w_2)}$$


```
{'did,how': 0.32115800164817027, 'how,did': 0.32115800164817027, 'mike,how': 0.831983625414161, 'how,mike': 0.8319836
25414161, 'mike,did': 3.7658404952500644, 'did,mike': 3.7658404952500644, 'hammer,how': 0.5443015529623801, 'how,hamm
er': 0.5443015529623801, 'hammer,did': 3.4781584227982836, 'did,hammer': 3.4781584227982836, 'hammer,mike': 4.3944491
54672439, 'mike,hammer': 4.394449154672439, 'live,how': 0.32115800164817027, 'how,live': 0.32115800164817027, 'live,d
id': 3.255014871484074, 'did,live': 3.255014871484074, 'live,mike': 4.1713056033582285, 'mike,live': 4.17130560335822
85, 'live,hammer': 4.1713056033582285, 'hammer,live': 4.1713056033582285, 'in,how': 0.9726324858075478, 'how,in': 0.9
726324858075478, 'in,did': 1.8889232176817026, 'did,in': 1.8889232176817026, 'in,mike': 1.8889232176817026, 'mike,i
n': 1.8889232176817026, 'in,hammer': 1.8889232176817028, 'hammer,in': 1.8889232176817028, 'in,live': 0.42334894254471
77, 'live,in': 0.4233489425447177, 'a,how': 0.9004700395593171, 'how,a': 0.9004700395593171, 'a,did': 1.8167607714334
721, 'did,a': 1.8167607714334721, 'a,mike': 1.8167607714334721, 'mike,a': 1.8167607714334721, 'a,hammer': 1.8167607714
4334721, 'hammer,a': 1.8167607714334721, 'a,live': 0.6549695811438309, 'live,a': 0.6549695811438309, 'a,in': 2.785011
242238338, 'in,a': 2.785011242238338, 'penthouse,how': 3.7013019741124933, 'how,penthouse': 3.7013019741124933, 'pent
house,did': 3.7013019741124933, 'did,penthouse': 3.7013019741124933, 'penthouse,mike': 3.7013019741124933, 'mike,pent
house': 3.7013019741124933, 'penthouse,hammer': 1.75539182505718, 'hammer,penthouse': 1.75539182505718}
```

Building Heterogeneous Graph

A graph with two or more types of node and/or two or more types of edge is called heterogeneous.



Homogeneous Graph

Heterogeneous Graph

Building Heterogeneous Graph

We have converted the Text Data into Graph, Now what?

Before feeding the data to GCN model, Let’s first understand the implementation of GCN..

Implementation of GCN

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features


        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)


        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' ('                + str(self.in_features) + ' -> '
     + str(self.out_features) + ')'
```

Implementation of GCN

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features

        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)

        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' ('                + str(self.in_features) + ' -> '
         + str(self.out_features) + ')'
```

Weight and Bias
(Parameters)

Implementation of GCN

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} Wx_u + b\right), \quad \forall v \in \mathcal{V}.$$

```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features


        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)


        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' ('                + str(self.in_features) + ' -> '
     + str(self.out_features) + ')'
```

Multiplying Node features x features

Implementation of GCN

$$h_v = f\left(\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} W x_u + b\right), \quad \forall v \in \mathcal{V}.$$

```python
class GraphConvolution(Module):
    """
    Simple GCN layer, similar to https://arxiv.org/abs/1609.02907
    """

    def __init__(self, in_features, out_features, bias=True):
        super(GraphConvolution, self).__init__()
        self.in_features = in_features
        self.out_features = out_features


        self.weight = Parameter(torch.FloatTensor(in_features, out_features))
        if bias:
            self.bias = Parameter(torch.FloatTensor(out_features))
        else:
            self.register_parameter('bias', None)


        self.reset_parameters()

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)

    def forward(self, input, adj):
        support = torch.mm(input, self.weight)
        output = torch.spmm(adj, support)
        if self.bias is not None:
            return output + self.bias
        else:
            return output

    def __repr__(self):
        return self.__class__.__name__ + ' ('                + str(self.in_features) + ' -> '
     + str(self.out_features) + ')'
```

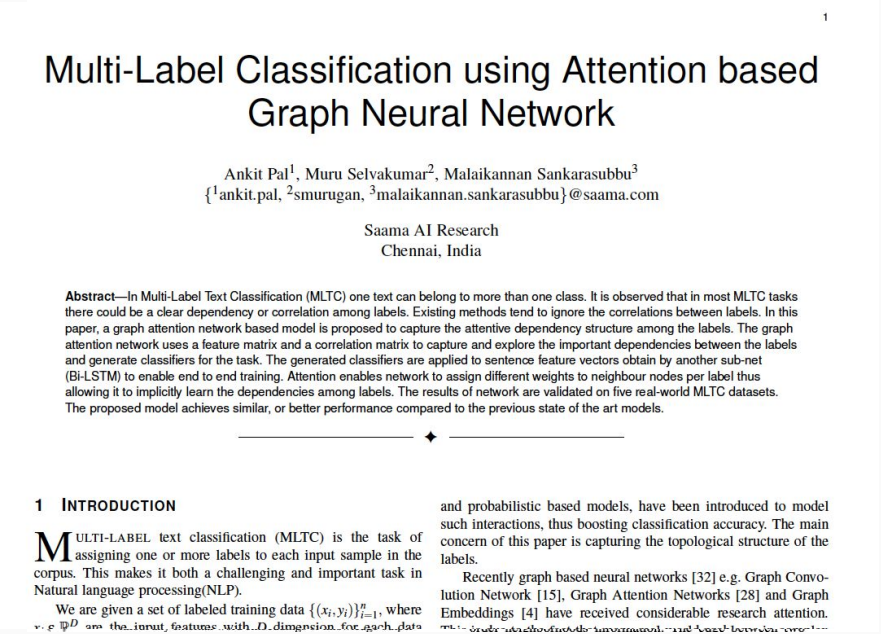
Implementation of GCN

Jupyter Notebook : 2 Demo of Sentiment Analysis with GCN

Applications
MAGNET: Multi-Label Text Classification using Attention-based Graph Neural Network

Key Points of Research Paper

- Using Graph Attention Network (GAT) as Feature extractor
- Constructing a graph between labels of dataset
- The graph attention network uses a feature matrix and a correlation matrix to capture and explore the crucial dependencies between the labels.
- The generated classifiers are applied to sentence feature vectors obtained from the text feature extraction network (BiLSTM) to enable end-to-end training.



1

Multi-Label Classification using Attention based Graph Neural Network

Ankit Pal[1], Muru Selvakumar[2], Malaikannan Sankarasubbu[3]
{[1]ankit.pal, [2]smurugan, [3]malaikannan.sankarasubbu}@saama.com

Saama AI Research
Chennai, India

Abstract—In Multi-Label Text Classification (MLTC) one text can belong to more than one class. It is observed that in most MLTC tasks there could be a clear dependency or correlation among labels. Existing methods tend to ignore the correlations between labels. In this paper, a graph attention network based model is proposed to capture the attentive dependency structure among the labels. The graph attention network uses a feature matrix and a correlation matrix to capture and explore the important dependencies between the labels and generate classifiers for the task. The generated classifiers are applied to sentence feature vectors obtain by another sub-net (Bi-LSTM) to enable end to end training. Attention enables network to assign different weights to neighbour nodes per label thus allowing it to implicitly learn the dependencies among labels. The results of network are validated on five real-world MLTC datasets. The proposed model achieves similar, or better performance compared to the previous state of the art models.

1 Introduction

Multi-label text classification (MLTC) is the task of assigning one or more labels to each input sample in the corpus. This makes it both a challenging and important task in Natural language processing(NLP).

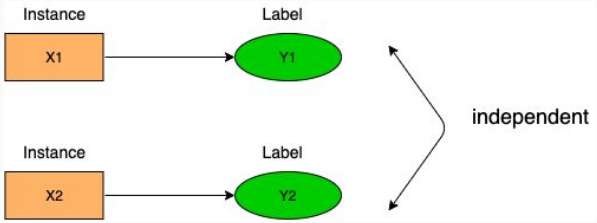
We are given a set of labeled training data $\{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathbb{R}^D$ are the input features with D dimension for each data

and probabilistic based models, have been introduced to model such interactions, thus boosting classification accuracy. The main concern of this paper is capturing the topological structure of the labels.

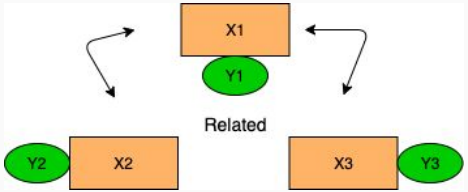
Recently graph based neural networks [32] e.g. Graph Convolution Network [15], Graph Attention Networks [28] and Graph Embeddings [4] have received considerable research attention.

Why Graph network for Multi-Label Classification ?

Conventional classification approaches assume that instances are independent identically distributed



In relational data and information networks, instances are correlated with each other.

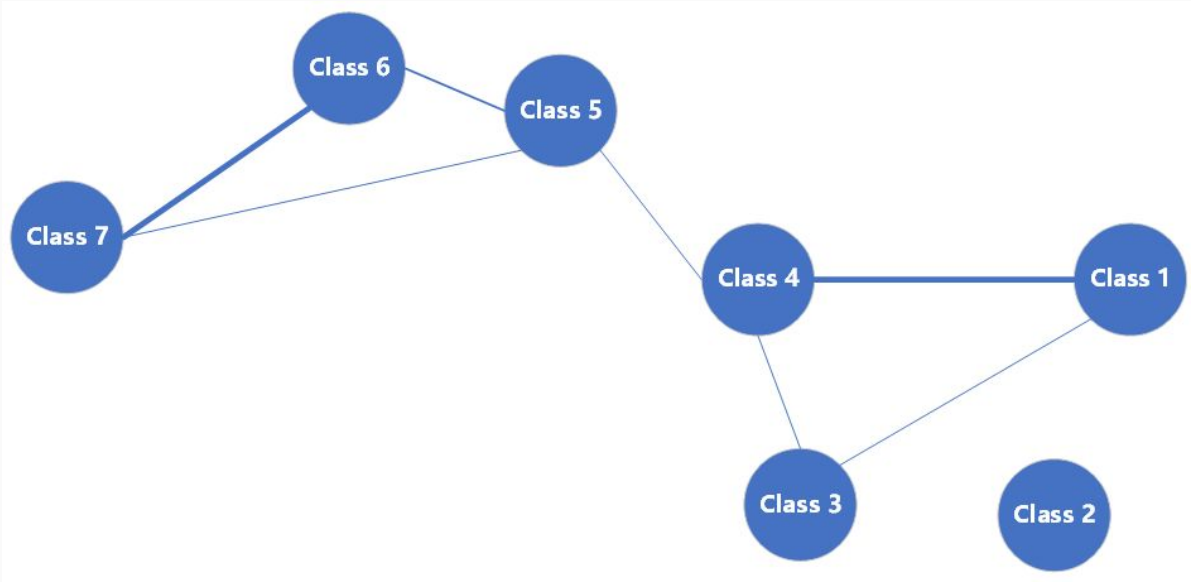


Why Graph network for Multi-Label Classification ?

In Multi-Label Text Classification (MLTC), one sample can belong to more than one class. It is observed that most MLTC tasks, there are dependencies or correlations among labels. Existing methods tend to ignore the relationship among labels.

Our Proposed Model

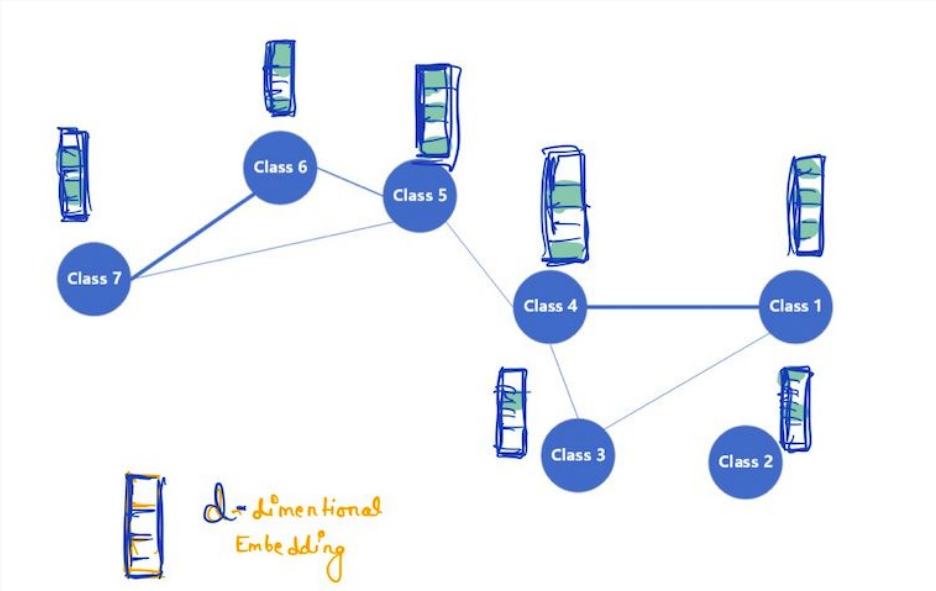
In this paper, a graph attention network-based model is proposed to capture the attentive dependency structure among the labels. GAT network takes the node features and adjacency matrix that represents the graph data as inputs. In the context of our model, the embedding vectors of the labels act as the node features.



Feature Matrix In Graph network

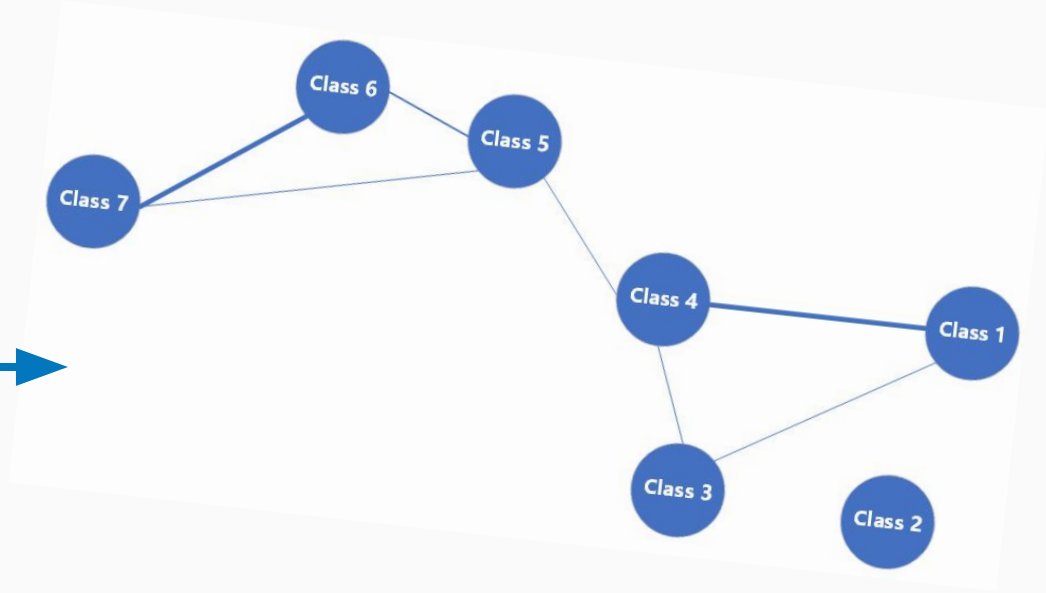
Feature Matrix

Assigning *n*-dim embedding vectors to each label, feature vector *N x F*



Adjacency Matrix

Adj matrix is initialized by counting the pairwise co-occurrence of labels in dataset. *N x N*



Graph Network as feature extractor

MAGNET network’s components :

- Feature Extraction
- Correlation between labels

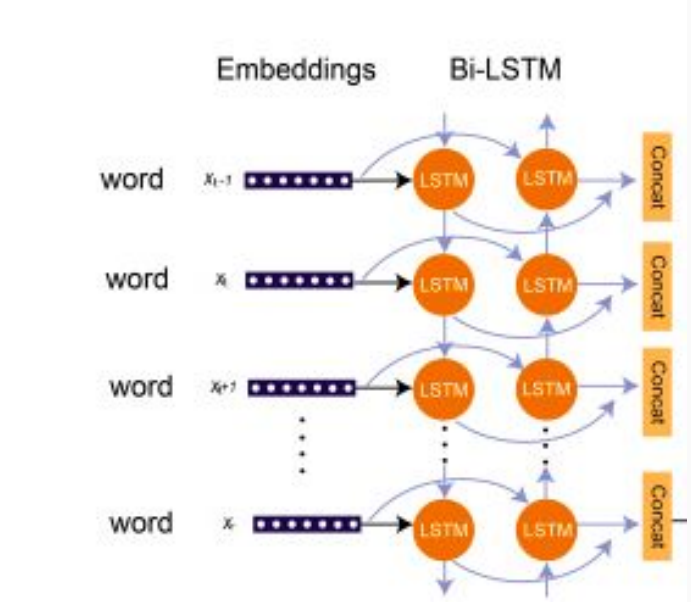
Feature Extraction

- Extracting Text features using LSTM network

$$\overrightarrow{h}_i = \overrightarrow{\text{LSTM}}\left(\overrightarrow{h}_{i-1}, x_i\right)$$
$$\overleftarrow{h}_i = \overleftarrow{\text{LSTM}}\left(\overleftarrow{h}_{i+1}, x_i\right) \quad (10)$$

We obtain the final hidden representation of the i-th word by concatenating the hidden states from both directions,

$$h_i = \left[\overrightarrow{h}_i ; \overleftarrow{h}_i\right] \quad (11)$$



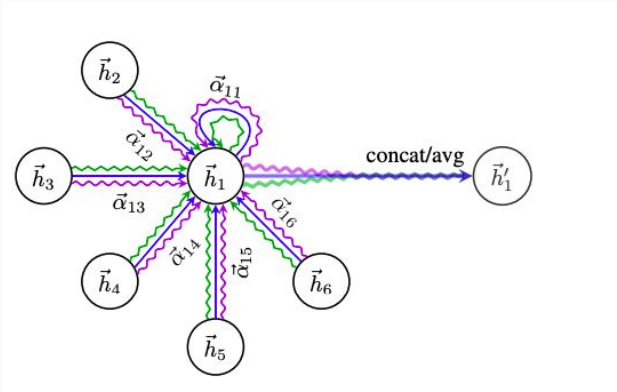
MAGNET Research Paper

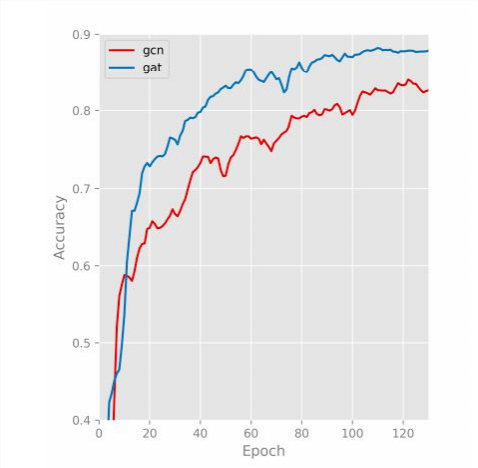
Attention in Graph?

Graph Attention Network over Graph Convolution Network

In GCNs, the neighborhoods of nodes combine with equal or pre-defined weights. However, the influence of neighbors can vary greatly, and the attention mechanism can identify label importance in correlation graph by considering the importance of their neighbor labels.

In our experiment, we are using multi-head attention (Vaswani et al., 2017) that utilizes K different heads to describe labels relationship.



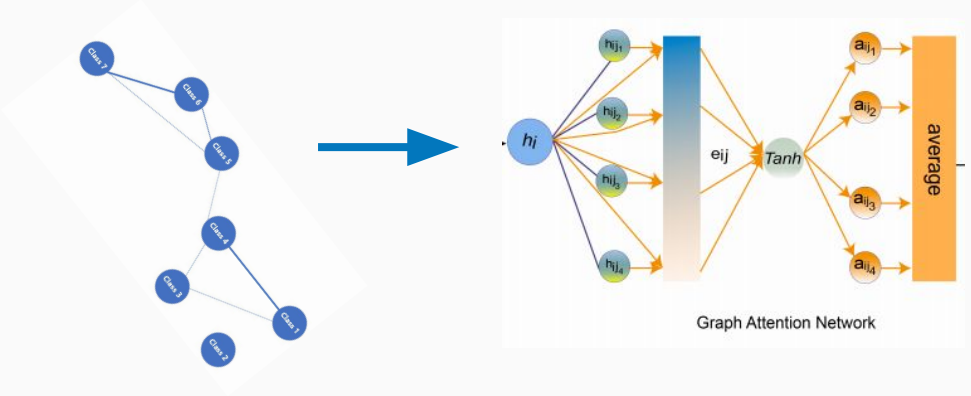


Graph Network as feature extractor

Using Graph attention network to find correlation between labels

$$\mathbf{H}_{\mathbf{i}}^{(\ell+1)} = \underbrace{Tanh\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in N(i)}\alpha_{ij,k}^{\ell}H_i^{\ell}W^{\ell}\right)}_{\text{attended label features}} \quad (9)$$

The output from the last layer is the attended label features $\mathbf{H_{gat}} \in \mathbb{R}^{\mathbf{c}\times\mathbf{d}}$ where c denotes the number of labels and d denotes the dimension of the attended label features. which is applied to the textual features from the BiLSTM.

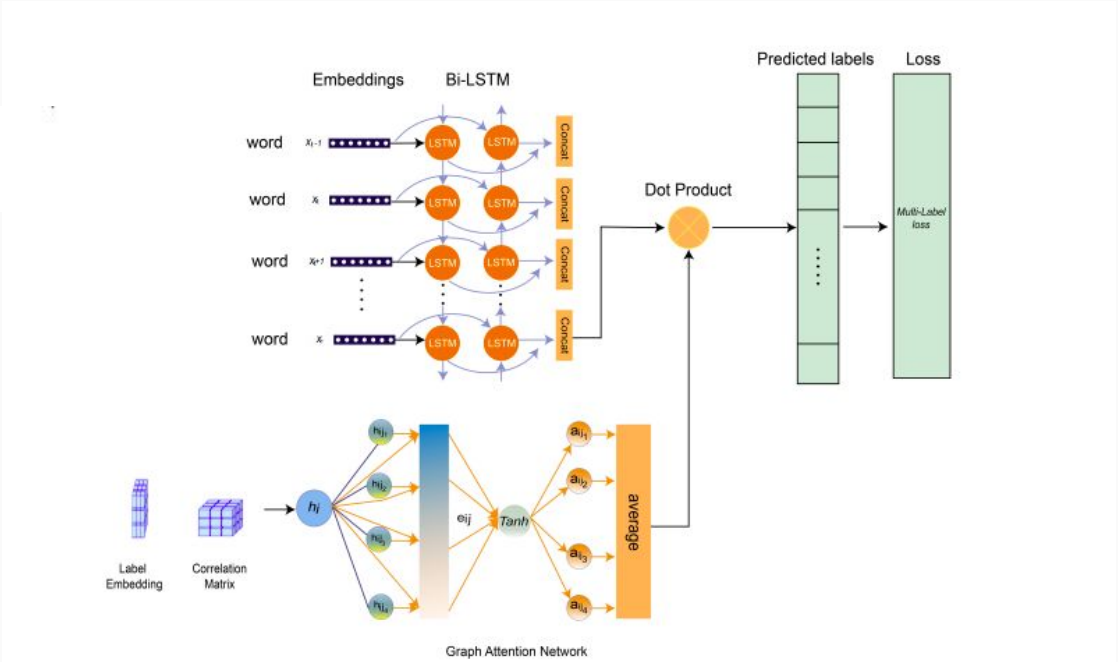


Graph Network as feature extractor

Using both features to classify the documents. The proposed model with graph attention allows us to capture the dependency structure among labels for MLTC tasks.

$$\hat{y} = F \odot H_{gat}$$

Where F are the features obtained from Bi-LSTM and H_{gat} are the graph features from Graph attention network.



Result

State-of-the-art accuracy compared to 27 different deep learning models including bert across five benchmark datasets

Table 3: Comparisons of Micro F1-score for various models on four benchmark datasets.

	F1-accuracy			
Methods	**Reuters-21578**	**AAPD**	**Slashdot**	**Toxic**
BR	0.878	0.648	0.486	0.853
BR-support	0.872	0.682	0.516	0.874
CC	0.879	0.654	0.480	0.893
CNN	0.863	0.664	0.512	0.775
CNN-RNN	0.855	0.669	0.530	0.904
MAGNET	**0.899**	**0.696**	**0.568**	**0.930**

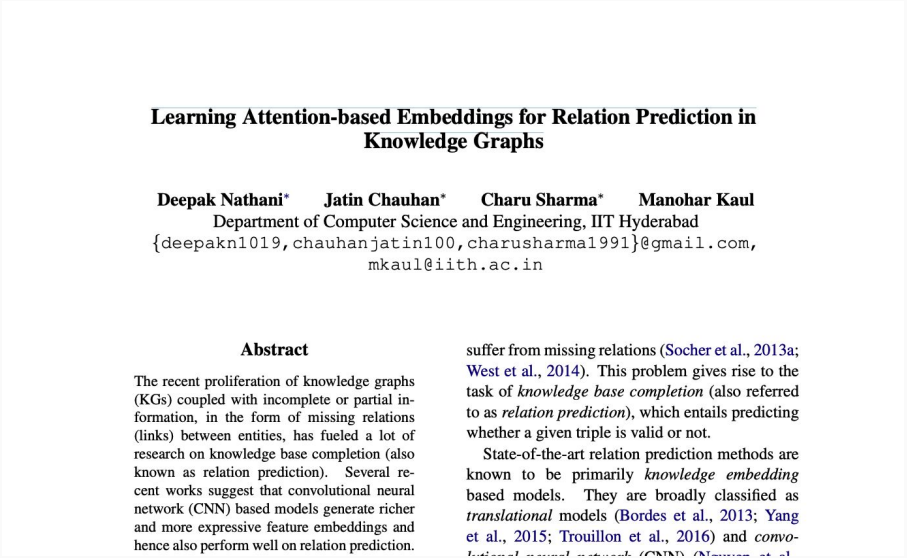
Table 4: Comparisons of Micro F1-score for various state-of-the-art models on Rcv1-v2 dataset.

Rcv1-v2	
Method	**Accuracy**
LR	0.692
SVM	0.691
HSVM	0.693
HLSTM	0.673
RCNN	0.686
XML-CNN	0.695
HAN	0.696
Bi-BloSAN	0.72
DCNN	0.732
SGM+GE	0.719
CAPSULE-B	0.739
CDN-SVM	0.738
HR-DGCNN	0.761
TEXTCNN	0.766
HE-AGCRCNN	0.778
BP-MLL$_{RAD}$	0.780
HTrans	0.805
BOW-CNN	0.827
HilAP	0.833
BERT	0.864
BERT + SGM	0.846
FMP + LaMP$_{pr}$	0.877
MAGNET	**0.885**

Applications
Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs

Key Points of Research Paper

- Using Graph based Attention Model to learn the relationships in Knowledge Graph
- Using Entity and Relation embeddings as input to Graph Model
- Model learn a new representation of every edge and aggregate the information by summing this over the neighborhood multiplying with the appropriate attention values.



Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs

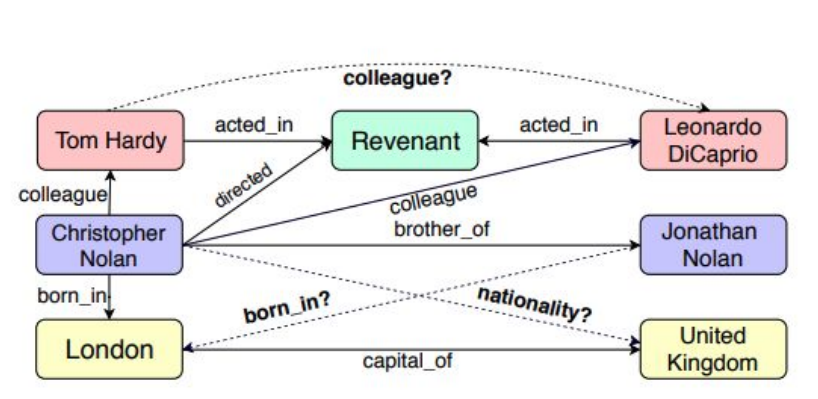
Deepak Nathani* **Jatin Chauhan*** **Charu Sharma*** **Manohar Kaul**
Department of Computer Science and Engineering, IIT Hyderabad
{deepakn1019,chauhanjatin100,charusharma1991}@gmail.com,
mkaul@iith.ac.in

Abstract

The recent proliferation of knowledge graphs (KGs) coupled with incomplete or partial information, in the form of missing relations (links) between entities, has fueled a lot of research on knowledge base completion (also known as relation prediction). Several recent works suggest that convolutional neural network (CNN) based models generate richer and more expressive feature embeddings and hence also perform well on relation prediction. suffer from missing relations (Socher et al., 2013a; West et al., 2014). This problem gives rise to the task of *knowledge base completion* (also referred to as *relation prediction*), which entails predicting whether a given triple is valid or not.

State-of-the-art relation prediction methods are known to be primarily *knowledge embedding* based models. They are broadly classified as *translational* models (Bordes et al., 2013; Yang et al., 2015; Trouillon et al., 2016) and *convo-*

Applications
Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs



A knowledge graph is denoted by $\mathcal{G} = (\mathcal{E}, R)$, where $\mathcal{E}$ and R represent the set of entities (nodes) and relations (edges), respectively. A triple (e_s, r, e_o) is represented as an edge r between nodes e_s and e_r in $\mathcal{G}$. A triple in the Knowledge Graph denotes a fact, for example in the image, the triple *(London, capital_of, United Kingdom)* represents the fact that *London* is the *capital of United Kingdon*, so *capital_of* is the relatoin between two specified entities.

Applications
Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs

Model

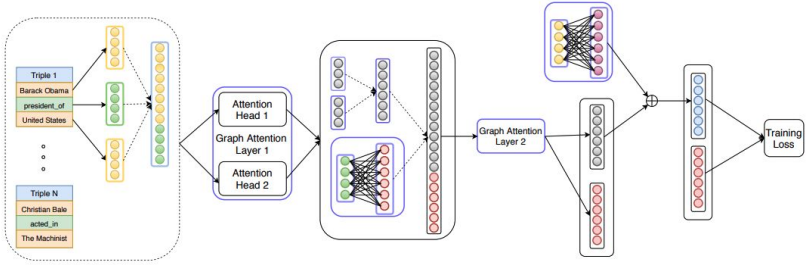


Figure 4: This figure shows end-to-end architecture of our model. Dashed arrows in the figure represent concatenation operation. Green circles represents initial entity embedding vectors and yellow circles represents initial relation embedding vectors.

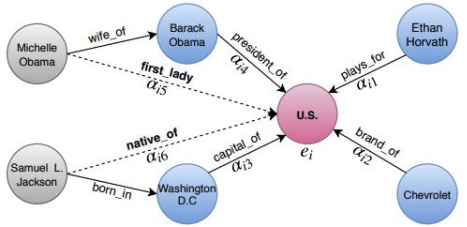


Figure 2: This figure shows the aggregation process of our graph attentional layer. α_{ij} represents relative attention values of the edge. The dashed lines represent an *auxiliary* edge from a n-hop neighbors, in this case $n = 2$.

Applications
Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs

Result

	WN18RR					FB15K-237				
			Hits@N					Hits@N		
	MR	MRR	@1	@3	@10	MR	MRR	@1	@3	@10
DistMult (Yang et al., 2015)	7000	0.444	41.2	47	50.4	512	0.281	19.9	30.1	44.6
ComplEx (Trouillon et al., 2016)	7882	0.449	40.9	46.9	53	546	0.278	19.4	29.7	45
ConvE (Dettmers et al., 2018)	4464	**0.456**	**41.9**	47	53.1	245	0.312	22.5	34.1	49.7
TransE (Bordes et al., 2013)	2300	0.243	4.27	44.1	53.2	323	0.279	19.8	37.6	44.1
ConvKB (Nguyen et al., 2018)	**1295**	0.265	5.82	44.5	55.8	216	0.289	19.8	32.4	47.1
R-GCN (Schlichtkrull et al., 2018)	6700	0.123	8	13.7	20.7	600	0.164	10	18.1	30
Our work	1940	0.440	36.1	**48.3**	**58.1**	**210**	**0.518**	**46**	**54**	**62.6**

Summary

- How Message Passing works in Graph networks
- How GCN perform aggregation
- Similarities between GCN and CNN
- GCN formulation
- How Renormalization works
- How to convert Text data into Graph for GCN
- Heterogeneous Graphs
- Implementation of GCN
- Sentiment Analysis using GCN
- Magnet Paper : Graph network as feature extractor
- Knowledge graph using GCN

Questions?