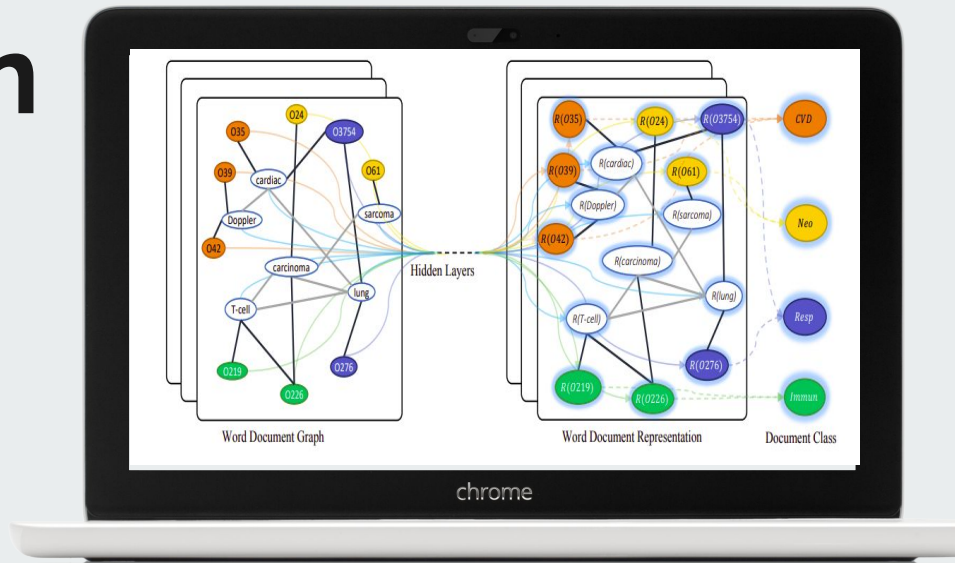


Text Classification by GCN

Graph Convolutional Network



Outline

The Problem

Abstract

State-of-the-art

Introduction

Solution Proposal

Methodology

Testing

Results

References



The Problem

Text Classification

- 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.



Abstract

- Text classification is a fundamental problem in natural language processing (NLP).
- There are numerous applications of text classification such as document organization, news filtering, spam detection, opinion mining, and computational phenotyping.

In this work, we used graph convolutional networks for text classification. We try to 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

—

State-of-art



Traditional Text Classification

- Traditional text classification studies mainly focus on feature engineering and classification algorithms.
- For feature engineering, the most commonly used feature is the bag of-words feature.
- In addition, some more complex features have been designed, such as n-grams and entities in ontologies.
- There are also existing studies on converting texts to graphs and perform feature engineering on graphs and subgraphs

Unlike these methods, our method can learn text representations as node embeddings automatically



Deep Learning for Text Classification

- Deep learning text classification studies can be categorized into two groups.
- One group of studies focused on models based on word embeddings. Several recent studies showed that the success of deep learning on text classification largely depends on the effectiveness of the word embeddings.
- Another group of studies employed deep neural networks. Two representative deep networks are CNN and RNN.

Although these methods are effective and widely used, they mainly focus on local consecutive word sequences, but do not explicitly use global word co-occurrence information in a corpus.



Graph Neural Networks

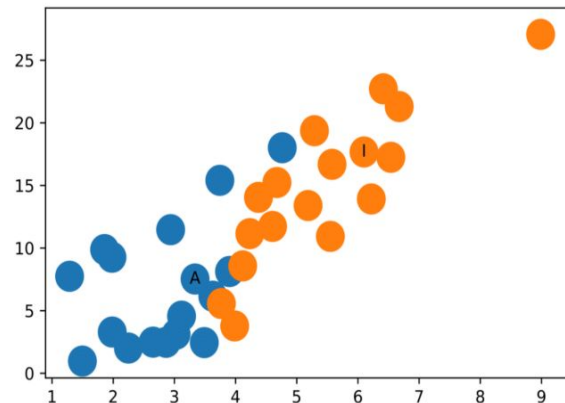
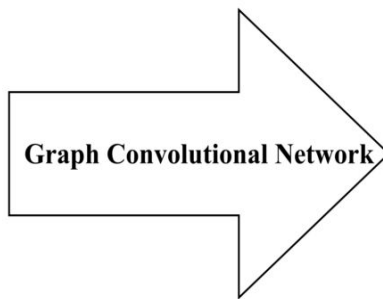
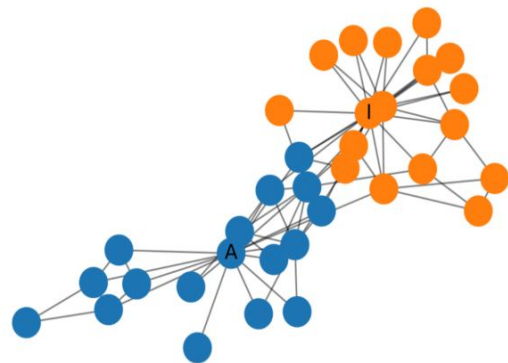
- The topic of Graph Neural Networks has received growing attentions recently. A number of authors generalized well-established neural network models like CNN that apply to regular grid structure (2-d mesh or 1-d sequence) to work on arbitrarily structured graphs.
- GCN was also explored in several NLP tasks such as semantic role labeling , relation classification and machine translation, where GCN is used to encode syntactic structure of sentences.

In contrast, when constructing the corpus graph, we regard the documents and words as nodes (hence heterogeneous graph) and do not require inter-document relations.

Introduction - Graph Convolutional Network (GCN)

What is GCN?

GCNs are a very powerful neural network architecture for machine learning on graphs. In fact, they are so powerful that even a randomly initiated 2-layer GCN can produce useful feature representations of nodes in networks. The figure below illustrates a 2-dimensional representation of each node in a network produced by such a GCN.



Solution Proposed

We implemented a novel graph neural network method for text classification. This model uses a whole corpus as a heterogeneous graph and learn word and document embeddings with graph neural networks jointly.



Solution Proposed


Text Graph Convolutional Networks (Text GCN)



- We aim to build a large and heterogeneous text graph which contains word nodes and document nodes so that global word co-occurrence can be explicitly modeled and graph convolution can be easily adapted.
- The number of nodes in the text graph $|V|$ is the number of documents (corpus size) plus the number of unique words (vocabulary size) in a corpus.
- We set feature matrix $X = I$ as an identity matrix which means every word or document is represented as a one-hot vector as the input to Text GCN.

Solution Proposed

Text Graph Convolutional Networks (Text GCN)

- 
- We build edges among nodes based on word occurrence in documents (document-word edges) and word co-occurrence in the whole corpus (word-word edges).
 - The weight of the edge between a document node and a word node is the term frequency-inverse document frequency (TF-IDF) of the word in the document, where term frequency is the number of times the word appears in the document, inverse document frequency is the logarithmically scaled inverse fraction of the number of documents that contain the word.
 - To utilize global word co-occurrence information, we use a fixed size sliding window on all documents in the corpus to gather co-occurrence statistics.

Solution Proposed

Text Graph Convolutional Networks (Text GCN)



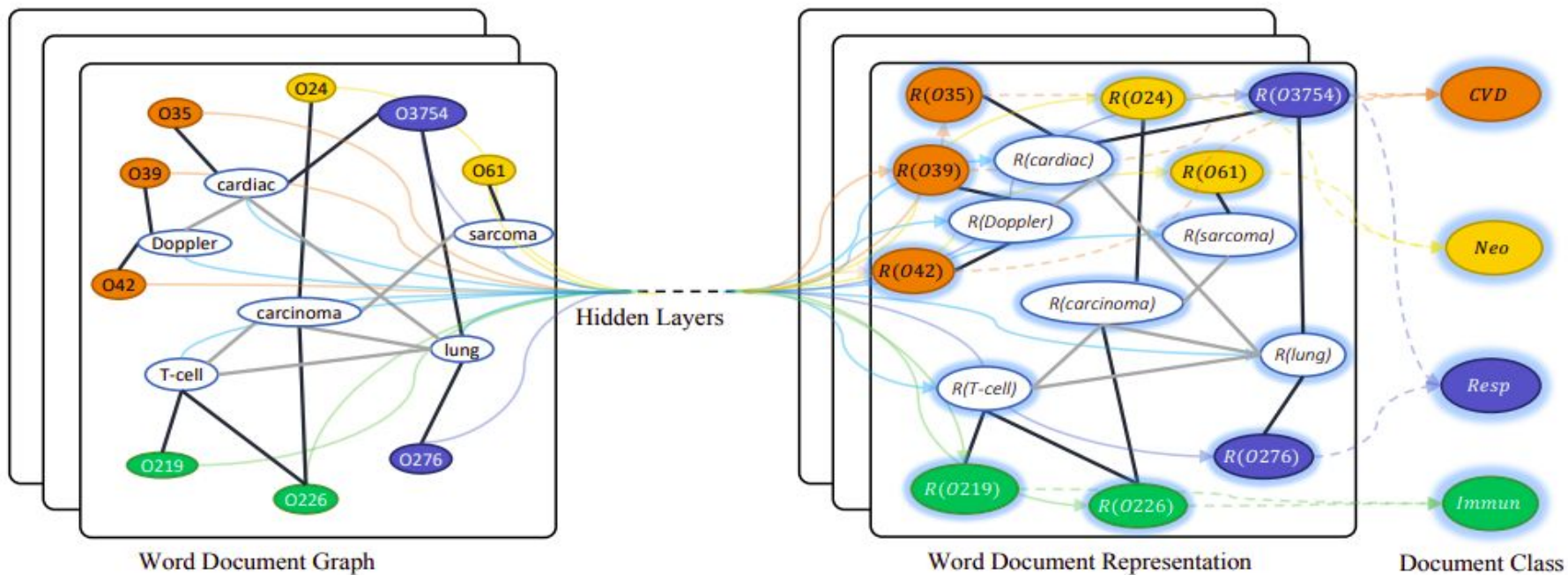
- We employ point-wise mutual information (PMI), a popular measure for word associations, to calculate weights between two word nodes.
- A positive PMI value implies a high semantic correlation of words in a corpus, while a negative PMI value indicates little or no semantic correlation in the corpus. Therefore, we only add edges between word pairs with positive PMI values.
- After building the text graph, we feed the graph into a simple two layer GCN, the second layer node (word/document) embeddings have the same size as the labels set and are fed into a softmax classifier.

Solution Proposed

Text Graph Convolutional Networks (Text GCN)

- The loss function is defined as the cross-entropy error over all labeled documents.
- The weight parameters can be trained via gradient descent.

Schematic of Text GCN



Schematic of Text GCN. Example taken from Ohsumed corpus. Nodes begin with “O” are document nodes, others are word nodes. Black bold edges are document-word edges and gray thin edges are word-word edges. $R(x)$ means the representation (embedding) of x . Different colors mean different document classes (only four example classes are shown to avoid clutter). CVD: Cardiovascular Diseases, Neo: Neoplasms, Resp: Respiratory Tract Diseases, Immun: Immunologic Diseases.

Methodology

The entire process is divided into 3 parts



- 1) We start off by creating a cleaned document from the existing data set and removing the stopwords from it.
- 2) We construct a single large graph from an entire corpus, which contains words and documents as nodes. The edge between two words is built by word co-occurrence information and the edge between a word node and a document node is built using the word frequency.
- 3) In the final stage the training part we convert the problem to a node classification problem and finally, have the classification result.

Stage 1

We start off by writing a script to clean the dataset and get it in the required format. We have done this with the help of Natural Language Toolkit(nltk). We use this library to get the English stop words so that the document can be cleaned and the processed version of the dataset we get.

Stage 1

We used the following lines of code to get stop words.

```
nltk.download('stopwords')  
stop_words = set(stopwords.words('english'))  
print(stop_words)
```

Following lines of code help us to get all the stop words that are there in english. Once we have all the stop words than we open our dataset using the following command

```
f = open('/kaggle/input/remove-words-20ng-txt/' + dataset + '.txt', 'rb')
```

We then create the copy for our dataset.

```
for line in f.readlines():  
    doc_content_list.append(line.strip().decode('latin1'))  
f.close()
```

Stage 1

To remove the rare words we have created a word_frequency dictionary which is being used for pre-processing of the document.

```
word_freq = {} # to remove rare words
for doc_content in doc_content_list:
    temp = clean_str(doc_content)
    words = temp.split()
    for word in words:
        if word in word_freq:
            word_freq[word] += 1
        else:
            word_freq[word] = 1
```

Stage 1

After this we create a new document file which contains words only with frequency greater than 5 and that are no stop words. This file at the latter stage is used for the purpose of constructing the heterogeneous graph.

```
for doc_content in doc_content_list:
    temp = clean_str(doc_content)
    words = temp.split()
    doc_words = []
    for word in words:
        # word not in stop_words and word_freq[word] >= 5
        if word not in stop_words and word_freq[word] >= 5:
            doc_words.append(word)

    doc_str = ' '.join(doc_words).strip()
    #if doc_str == '':
        #doc_str = temp
    clean_docs.append(doc_str)
```

Stage 2

We build a large and heterogeneous text graph which contains word nodes and document nodes so that global word co-occurrence can be explicitly modeled and graph convolution can be easily adapted.

The number of nodes in the text graph $|V|$ is the number of documents (corpus size) plus the number of unique words (vocabulary size) in a corpus.

Stage 2

We simply set feature matrix $X = I$ as an identity matrix which means every word or document is represented as a one-hot vector as the input to Text GCN.

```
word_embeddings_dim = 300
```

We created a 300 dimension matrix for the purpose of word embedding. We build edges among nodes based on word occurrence in documents (document-word edges) and word co-occurrence in the whole corpus (word-word edges).

The weight of the edge between a document node and a word node is the term frequency-inverse document frequency (TF-IDF) of the word in the document, where term frequency is the number of times the word appears in the document, inverse document frequency is the logarithmically scaled inverse fraction of the number of documents that contain the word.

Stage 2

To utilize global word co-occurrence information, we use a fixed size sliding window on all documents in the corpus to gather co-occurrence statistics.

This co-occurrence is used to construct the graph between the words of the document.

```
# build vocab
word_freq = {}
word_set = set()
for doc_words in shuffle_doc_words_list:
    words = doc_words.split()
    for word in words:
        word_set.add(word)
        if word in word_freq:
            word_freq[word] += 1
        else:
            word_freq[word] = 1

vocab = list(word_set)
vocab_size = len(vocab)
```

Stage 2

The sliding window used for the purpose of generation of the word co occurrence matrix is of 20

```
# word co-occurrence with context windows  
window_size = 20
```

We tried to calculate a statistic to get the weights on the edges where we find the ratio of the number of sliding windows having the word i among all sliding windows , Similarly for 2 words namely i, j we have calculated the same ratio as windows that contain both word i and j , to the total number of sliding windows in the corpus

Stage 2

```
# pmi as weights
num_window = len(windows)

for key in word_pair_count:
    temp = key.split(',')
    i = int(temp[0])
    j = int(temp[1])
    count = word_pair_count[key]
    word_freq_i = word_window_freq[vocab[i]]
    word_freq_j = word_window_freq[vocab[j]]
    pmi = log((1.0 * count / num_window) /
              (1.0 * word_freq_i * word_freq_j / (num_window * num_window)))
    if pmi <= 0:
        continue
    row.append(train_size + i)
    col.append(train_size + j)
    weight.append(pmi)
```

Stage 3

After building the text graph, we feed the graph into a simple two layer GCN as in the second layer node (word/document) embeddings have the same size as the labels set .

Stage 3

A two-layer GCN can allow message passing among nodes that are at maximum two steps away. Although there are no direct document-document edges in the graph, the two-layer GCN allows the information exchange between pairs of documents.

```
class GCN(Model):
    def __init__(self, placeholders, input_dim, **kwargs):
        super(GCN, self).__init__(**kwargs)

        self.inputs = placeholders['features']
        self.input_dim = input_dim
        # self.input_dim = self.inputs.get_shape().as_list()[1] # To be supported in future
        # Tensorflow versions
        self.output_dim = placeholders['labels'].get_shape().as_list()[1]
        self.placeholders = placeholders

        self.optimizer = tf.train.AdamOptimizer(learning_rate=FLAGS.learning_rate)

        self.build()
```

The above code is for the GCN model which is taken from (Kipf and Welling 2017). The layers are also taken from there Paper.

Stage 3

```
def _build(self):

    self.layers.append(GraphConvolution(input_dim=self.input_dim,output_dim=FLAGS.hidden1,placeholders=self.placeholders,act=tf.nn.relu,dropout=True,featureless=True,sparse_inputs=True,logging=self.logging))

    self.layers.append(GraphConvolution(input_dim=FLAGS.hidden1,output_dim=self.output_dim,placeholders=self.placeholders,act=lambda x: x,dropout=True,logging=self.logging))

def predict(self):
    Return tf.nn.softmax(self.outputs)
```

The above code is for the GCN model which is taken from (Kipf and Welling 2017). The layers are also taken from there Paper.

Stage 3

After generating the graph (two consecutive layer of GCN) text classification problem is converted into node classification problem as nodes are used to represent documents.

Apart from training the classification model word embeddings and document embeddings are also generated which can be used for transfer learning and other purposes.

Masked softmax cross entropy and masked accuracy are being used for loss function and accuracy. Precision, recall and F1 Score are also being calculated for comparison purposes.

As can be seen in the above code snippet, two graph based layers are being stacked along with a softmax layer (for classification purpose). First layer does not contain features and ReLU activation is applied to the output while the second layer contains features with an identity output i.e, $f(x)=x$ or no activation.

Stage 3

Following hyper parameters are being used for training after suitable considerations:

- Learning rate = 0.02
- No. of epochs = 200
- No. of units in hidden layer 1 = 200
- Dropout parameter = 0.5

Stage 3
<p>Since early stopping was also employed, the model was optimally trained by the end of the 93rd epoch. Metrics in our training were obtained as follows:</p>

Test Precision, Recall and F1-Score...						
	precision	recall	f1-score	support		
0	0.9483	0.9673	0.9577	398		
1	0.7382	0.8046	0.7700	389		
2	0.8467	0.7962	0.8207	319		
3	0.8908	0.9112	0.9009	394		
4	0.9916	0.9415	0.9659	376		
5	0.9307	0.9495	0.9400	396		
6	0.8808	0.9472	0.9128	398		
7	0.8368	0.8390	0.8379	385		
8	0.7089	0.7704	0.7384	392		
9	0.7277	0.6813	0.7037	251		
10	0.7946	0.8929	0.8409	364		
11	0.8288	0.6871	0.7513	310		
12	0.8260	0.8092	0.8175	393		
13	0.9544	0.9496	0.9520	397		
14	0.8665	0.8051	0.8346	395		
15	0.7957	0.8590	0.8261	390		
16	0.9302	0.9091	0.9195	396		
17	0.9103	0.8712	0.8903	396		
18	0.7863	0.7284	0.7563	394		
19	0.9673	0.9649	0.9661	399		
accuracy			0.8599	7532		
macro avg			0.8580	0.8542	0.8551	7532
weighted avg			0.8613	0.8599	0.8597	7532

Results

Results

Using GPU available on kaggle, the model took ~ 30 minutes to train and generate word as well as document embeddings.

Our setup and implementation can be referenced [here](#).

Log:

Time	Line	#Log	Message
1.7s	1	[NbConvertApp]	Converting notebook __notebook__.ipynb to html
3.2s	2	[NbConvertApp]	Writing 550422 bytes to __results__.html
3.2s	3		
3.2s	5		Complete. Exited with code 0.

Results

Performance of Text GCN on various Dataset:

DATASET	MODEL	METRIC NAME	METRIC VALUE
20NEWS	Text GCN	Accuracy	86.34
MR	Text GCN	Accuracy	76.74
Ohsumed	Text GCN	Accuracy	68.36
R52	Text GCN	Accuracy	93.56
R8	Text GCN	Accuracy	97.07

References

References

1. [Cai, Zheng, and Chang 2018] Cai, H.; Zheng, V. W.; and Chang, K. 2018. A comprehensive survey of graph embedding: problems, techniques and applications.
2. [Chen, Ma, and Xiao 2018] Chen, J.; Ma, T.; and Xiao, C. 2018. Fastgcn: Fast learning with graph convolutional networks via importance sampling.
3. [Aggarwal and Zhai 2012] Aggarwal, C. C., and Zhai, C. 2012. A survey of text classification algorithms. In Mining text data. Springer. 163–222.
4. [Liang Yao, Chengsheng Mao, Yuan Luo 2018]Liang Yao; Chengsheng Mao;Yuan Lu. 2018. Graph Convolutional Networks for Text Classification
5. [Bastings et al. 2017] Bastings, J.; Titov, I.; Aziz, W.; Marcheggiani, D.; and Simaan, K. 2017. Graph convolutional encoders for syntax-aware neural machine translation.
6. [Battaglia et al. 2018] Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018.

Contributors

Amarjeet Sinha - 17135012

Karthik Gupta - 17135049

Naman Kaushik - 17134020



Contributions of Amarjeet Sinha (17135012)

- 1) Extensive Reading of the Material and Environment Setup for Training and Testing.
- 2) Implemented the Graph Build Algo. from the paper for the purpose of creation of graph between the words and the word document occurrence.
- 3) Implemented the `utils.py` module from the paper and essential part of the preprocessing of the Dataset.

Contributions of Kartik Gupta(17135049)

- 1) Extensive Reading of the Material and Training.
- 2) Implemented the GCN model from the paper for training purpose. For feeding the graph created in previous stage 2.
- 3) Implemented the models. Py and layers. Py and essential part for the training .

Contributions of Naman Kaushik(17134020)

- 1) Extensive Reading of the Material and Preprocessing of the data.
- 2) Implemented the pre processing module for Dataset and metric creation for results.
- 3) Implemented the remove_words. Py and metrics.Py and essential part for pre processing and Result generation .

Thank You