ME-GCN: Multi-dimensional Edge-embedded Graph Convolutional Networks for Semi-supervised Text Classification

Kunze Wang,1* Soyeon Caren Han,2† Siqu Long 1 and Josiah Poon2
School of Computer Science
The University of Sydney
NSW, Australia
1{kwan4418, slon6753}@uni.sydney.edu.au
2{caren.han, josiah.poon}@sydney.edu.au

ABSTRACT

Compared to sequential learning models, graph-based neural networks exhibit excellent ability in capturing global information and have been used for semi-supervised learning tasks. Most Graph Convolutional Networks are designed with the single-dimensional edge feature and failed to utilise the rich edge information about graphs. This paper introduces the ME-GCN (Multi-dimensional Edge-enhanced Graph Convolutional Networks) for semi-supervised text classification. A text graph for an entire corpus is firstly constructed to describe the undirected and multi-dimensional relationship of word-to-word, document-document, and word-to-document. The graph is initialised with corpus-trained multi-dimensional word and document node representation, and the relations are represented according to the distance of those words/documents nodes. Then, the generated graph is trained with ME-GCN, which considers the edge features as multi-stream signals, and each stream performs a separate graph convolutional operation. Our ME-GCN can integrate a rich source of graph edge information of the entire text corpus. The results have demonstrated that our proposed model has significantly outperformed the state-of-the-art methods across eight benchmark datasets. The code is available on: https://github.com/usydnlp/MEGCN

1 INTRODUCTION

Deep Learning models have performed well and have been widely used for text classification; however, the performance is not always satisfactory when utilising small labelled datasets. In many practical scenarios, the labelled dataset is very scarce as human labelling is time-consuming and may require domain knowledge. There is a pressing need for studying semi-supervised text classification with a relatively small number of labelled training data in deep learning paradigm. For the successful semi-supervised text classification, it is crucial to maximize effective utilization of structural and feature information of unlabelled data.

Graph Neural Networks have recently received lots of attention as it can analyse rich relational structure, prioritize global features exploitation, and preserve global structure of a graph in embeddings. Due to these benefit, there have been successful attempts to revisit semi-supervised learning with Graph Convolutional Networks (GCN) (Kipf & Welling, 2017). TextGCN (Yao et al., 2019) initialise the whole text corpus as a document-word graph and applies GCN. It shows potential of GCN-based semi-supervised text classification. Hu et al., (2019) worked on semi-supervised short text classification using GCN with topic-entity, and Liu et al., (2020) proposed tensorGCN with semantic, syntactic, and sequential information. One major problem in those existing GCN-based text classification models is that edge features are restricted to be one-dimensional, which are the indication about whether there is edge or not (e.g. binary connectedness) or often one-dimensional

* Equal contribution
† Corresponding author (Caren.Han@sydney.edu.au)
real-value representing similarities (e.g. pmi, tf-idf). Instead of being a binary indicator variable or a single-dimensional value, edge features can possess rich information and fully incorporated by using multi-dimensional vectors. Addressing this problem is likely to benefit several graph-based classification problems but is particularly important for the text classification task. This is because the relationship between words and documents can be better represented in a multi-dimensional vector space rather than a single value. For example, word-based vector space models embed the words in a vector space where similarly defined words are mapped near to each other. Rather than using the lexical-based syntactic parsers or additional resources, words that share semantic or syntactic relationships will be represented by vectors of similar magnitude and be mapped in close proximity to each other in the word embedding. Using this multi-dimensional word embedding as node and edge features, it will be more effective to analyze rich relational information and explore the global structure of a graph. Then, what would be the best way to exploit edge features in a text graph convolutional network? According to the recently reported articles (Gong & Cheng, 2019; Khan & Blumenstock, 2019; Huang et al., 2020; Liu et al., 2020; Schlichtkrull et al., 2018), more rich information should be considered in the relations in the graph neural networks.

In this paper, we propose a new multi-dimensional edge enhanced text graph convolutional networks (ME-GCN), which is suitable for the semi-supervised text classification. Note that the focus of our semi-supervised text classification task is on small proportion of labelled text documents with no other resource, i.e. no pre-trained word embedding or language model, syntactic tagger or parser. We construct a single large textual graph from an entire corpus, which contains words and documents as nodes. The graph describes the undirected and multi-dimensional relationship of word-to-word, document-document, and word-to-document. Each word and document are initialised with corpus-trained multi-dimensional word and document embedding, and the relations are represented based on the semantic distance of those representations. Then, the generated graph is trained with ME-GCN, which considers edge features as multi-stream signals, and each stream performs a separate graph convolutional operation. We conduct experiments on several semi-supervised text classification datasets. Our model can achieve strong text classification performance with a small proportion of labelled documents with no additional resources. The main contributions: (1) To the best of our knowledge, this is the first attempt to apply multi-dimensional edge features on GNN for text classification. (2) ME-GCN is proposed to use corpus-trained multi-dimensional word and document-based edge features for the semi-supervised text classification. (3) Experiments are conducted on several semi-supervised text classification datasets to illustrate the effectiveness of ME-GCN.

2 RELATED WORKS

Semi-supervised text classification: Due to the high cost of human labeling and the scarcity of fully-labeled data, semi-supervised models have received attention in text classification. Latent variable models (Chen et al., 2015) apply topic models by user-oriented seed information and infer the documents’ labels based on category-topic assignment. The embedding-based model (Tang et al., 2015; Meng et al., 2018) utilizes seed information to derive text (word or document) embeddings for documents and labels for text classification. Yang et al. (2017) leveraged sequence-to-sequence Variational AutoEncoders (VAEs), and Miyato et al. (2017) utilized adversarial training to the text domain by applying perturbations to the word embeddings. Graph convolutional networks (GCN) have been popular in semi-supervised learning as it shows superior global structure understanding ability (Kipf & Welling, 2017).

GNN for Text Classification: Graph Neural Networks have successfully used in various NLP tasks (Bastings et al., 2017; Yu et al., 2019; Cao et al., 2019; Xu et al., 2019) proposed the Text Graph Convolutional Networks by applying a basic GCN (Kipf & Welling, 2017) to the text classification task. In their work, a text graph for the whole corpus is constructed; word and document nodes are initialized with one-hot representation and edge features are represented as one-dimensional real values, such as PMI, TF-IDF. Several studies have attempted multiple different graph alignments using knowledge graph or semantic/syntactic graph. Vashishth et al. (2019) applied GCN to incorporate syntactic/semantic information for word embedding training. Cao et al. (2019) proposed an alignment-oriented knowledge graph embedding for entity alignment. TensorGCN (Liu et al., 2020) proposed semantic, syntactic, and sequential contextual information. In their framework, multiple aspect graphs are constructed from external resources, and those graph are jointly trained. There are several Multi-aspect, Multi-dimension edge research have been published but none of them are
working on the Natural Language Processing field (Schlichtkrull et al., 2018; Khan & Blumenstock, 2019; Ma et al., 2020; He et al., 2020). Recently, graph attention mechanism has been applied in text classification tasks (Mei et al., 2021; Liu et al., 2021; Yang et al., 2021). Others focus on using both local and global information (Jin et al., 2021), multi-modality with text and image information (Yang et al., 2021b), enhancing TextGCN with other models (Ragesh et al., 2021) and combining with external knowledge (Dai et al., 2022).

3 ME-GCN

We propose the Multi-dimensional Edge-enhanced Graph Convolutional Networks (ME-GCN) for semi-supervised text classification. Note that all graph components are only based on the given text corpus without using any external resources. We utilize the GCN as a base component, due to its simplicity and effectiveness. Figure 1 shows the overall architecture of ME-GCN.

3.1 Textual Graph Construction

We first describe how to construct a textual graph that contains word/document node representation and multi-dimensional edge features for a whole text corpus. We apply a straightforward textual construction approach that treats words and documents as nodes in the graph. Unlike (Yao et al., 2019), we have three types of edges, namely word-document edge, word-word edge, and document-document edge with the aim to investigate all possible relations between nodes. Formally, we define a ME-GCN graph $G = (V, E^t, A)$, where $|V| = N$ is the set of graph nodes, $E$ is the set of graph edges, and $A \in \mathbb{R}^{N \times N}$ is the graph adjacency matrix.

3.1.1 Textual Node Construction

From an entire textual corpus, we construct word and document nodes in a graph so that the global word and document distance can be explicitly modeled and graph convolution can be easily adapted. ME-GCN considers the word and document nodes as components for preserving rich information and representing the global structure of a whole corpus, which can fully support for the successful semi-supervised text classification. With this in mind, ME-GCN trains word/node feature by using a Word2Vec (Mikolov et al., 2013) for word nodes, and a Doc2Vec (Le & Mikolov, 2014) for document nodes. For instance, Word2Vec takes as its input a whole corpus of words, and the trained word vectors are positioned in a vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. This is well-aligned with the role of

Figure 1: ME-GCN model architecture
graph neural networks, representing the global structure of the corpus, and preserving rich semantic information of the corpus. Most importantly, those word/document embeddings are distributed representations of text in an $T$-dimensional space so the distance between words and documents can be represented as a multi-dimensional vector. Formally, the word/document node features in ME-GCN are initialised as follows. Note that the negative sampling is applied to reduce the training time.

**Word Node Construction** We train the Word2Vec CBOW (Mikolov et al., 2013) using context words to predict the centre word. Assume we have a given text corpus consisting of $K$ documents and $U$ unique words. The input is a set of context words $X_{ik}$ in document $k \in K$ encoded as one-hot vector of size $U$. Then the hidden layer $H$ and output layer $Output$ are formulated in equation (1) and (2), in which $W_{U \times T}$ and $W'_{T \times U}$ are two projection matrix. After training, we extract the $U$ vectors of dimension $T$ from the updated matrix $W_{U \times T}$ representing the corresponding $U$ unique words in the whole corpus.

$$H = \sum_{i=1}^{C} X_{ik} W_{U \times T}$$  
(1)

$$Output = HW'_{T \times U}$$
(2)

**Document Node Construction** Doc2Vec CBOW (Le & Mikolov, 2014) is essentially the same as Word2Vec. In Doc2Vec, we feed the context words $X_{ik}$ together with the current document $k$ to the model, which is also encoded as one-hot vector based on the document id, and the vector size becomes $\hat{U} = U + K$. We have the projection matrix $W_{\hat{U} \times T}$ containing $U + K$ vectors. After training, those $K$ vectors in the updated $W_{\hat{U} \times T}$ are used for representing the corresponding $K$ document.

$$H = D_k W_{\hat{U} \times T} + \sum_{i=1}^{C} X_{ik} W'_{\hat{U} \times T}$$
(3)

$$Output = HW'_{T \times \hat{U}}$$
(4)

3.1.2 Multi-dimensional Edge Construction

In this section, we describe how to construct a multi-dimensional edge feature in a graph. A traditional textual graph edge (Yao et al., 2019) was based on word occurrence in documents (document-word edges), and word co-occurrence in the whole corpus (word-word edges), however, the occurrence information is not enough to extract how close two pieces of text are in both surface proximity and meaning. According to Mikolov et al. (2013), Kusner et al. (2015), the distance between word/document embeddings learn semantically meaningful representations for words from local co-occurrence in sentences and each dimension of word2vec and doc2vec represents the same aspect of word/document representations. Inspired by this, we utilise the distance between each dimension of word/document embeddings to preserve the rich semantic information captured by edges, which are also presented as multi-dimensional vectors. To represent all possible edge types, we propose three types of edges: word-word edges, document-document edges, and word-document edges. Our goal is to incorporate the semantic similarity between individual node pairs (each unique word and document) into multi-dimensional edge features. One such measure for word/document node similarity is provided by their Euclidean distance in the Word2Vec or Doc2Vec embedding space. We separately use each dimension space in the node feature (Word2Vec/Doc2Vec) for representing each of the dimension in the multi-dimensional node edge. Thus, we have $T$ dimensional edges between nodes of $T$ dimensional features and each $t \in \{1, 2, ..., T\}$ is represented by one dimensional Euclidean distance calculation in the $t^{th}$ dimensional space. This edge calculation method is applied to word-word and doc-doc edges.

**Word-Word Edge Feature** We draw on the learned semantics in each feature dimension of the word embedding of size $T$ to calculate the edge weight for each dimension. Concretely, the $T$-dimensional word-word edge $E_{w_i, w_j}^{(t)}, t \in \{1, 2, ..., T\}$ between word $i$ and word $j$ is formulated as in equation (5), in which $W_i^{(t)}$ and $W_j^{(t)}$ represents the feature value at the dimension $t$ of the word embedding $W_i$ for word $i$ and $W_j$ for word $j$ respectively. The denominator calculates the distance of the two words regarding dimension $t$ and $tanh(-1)$ is used for normalization.

$$E_{w_i, w_j}^{(t)} = tanh\left(\frac{1}{|W_i^{(t)} - W_j^{(t)}|}\right)$$
(5)
**Doc-Doc Edge Feature** The document-document edge is constructed in a way similar to the word-word edge. As is shown in equation (6), the $T$-dimensional document-document edge $E_{d_i,d_j}^{(t)}$ is calculated based on the normalized Euclidean distance between the values $D_i^{(t)}$ and $D_j^{(t)}$ at each dimension $t$ of the features for document $i$ and $j$. To relieve over-smoothing issue, we only consider edges between two documents having over $u$ overlapping words.

$$E_{d_i,d_j}^{(t)} = \tanh \left( \frac{1}{|D_i^{(t)} - D_j^{(t)}|} \right) \text{if } W_{d_i \cap d_j} \geq u$$

(6)

**Word-Doc Edge Feature** We use the same calculation method for a single-dimension word-document edge as in TextGCN while repeating it for each dimension $t$. Thus, the $T$-dimensional word-document edge $E_{w_i,d_j}^{(t)}$ is simply represented as the TF-IDF value of word $i$ and document $j$. This is repeated for each dimension $t$, as is formulated in equation (7).

$$E_{w_i,d_j}^{(t)} = \text{TF-IDF}_{w_i,d_j}$$

(7)

### 3.2 ME-GCN Learning

After constructing the multi-dimensional edge enhanced text graph, we focus on applying effective learning framework to perform GCN on the textual graph with multi-dimensional edge features.

The traditional GCN learning takes into the initial input matrix $H^{(0)} \in \mathbb{R}^{N \times d_0}$ containing $N$ node features of size $d_0$. Then the propagation through layers is made based on the rule in equation (9), which takes into consideration both node features and the graph structure in terms of connected edges.

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma(\hat{A}H^{(l)}W^{(l)})$$

(8)

The $l$ and $(l+1)$ represents the two subsequent layers, $\hat{A} = \hat{D}^{-\frac{1}{2}} \hat{A} \hat{D}^{-\frac{1}{2}}$ is the normalized symmetric adjacency matrix $\hat{A} = A + I$ ($I$ is an identity matrix for including self-connection), $\hat{D}$ is the diagonal node degree matrix with $\hat{D}(i,i) = \Sigma_j \hat{A}(i,j)$, and $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a layer-specific trainable weight matrix for $l$th layer. $d_l$ and $d_{l+1}$ indicates the node feature dimension for $l$th layer and $(l + 1)$th respectively. $\sigma$ denotes a non-linear activation function for each layer such as Leaky ReLu/ReLU except for the output layer where softmax is normally used for the classification.

Our goal is to represent the node representation by aggregating neighbour information with each edge features in a multi-stream manner. Hence, we generalize the traditional GCN learning approach to perform multi-stream(MS) learning for the multi-dimensional edge enhanced graph. The overall MS learning procedure is in equation (10), for each node feature in $H^{(l)} \in \mathbb{R}^{N \times d_l}$, we apply the multi-stream GCN learning $f_{MS}$ that formulates $t$ streams of traditional GCN learning in equation (9) through the $t$ dimensions of the connected edge, resulting in the multi-stream hidden feature $H^{(l+1)} \in \mathbb{R}^{N \times d_{ms}^{(l+1)}}$ at $(l+1)$th layer. Here $t \in \{1, 2, ..., T\}$ and $d_{ms}^{(l+1)}$ is the multi-stream feature size for each edge dimension at this layer. Then a multi-stream aggregation function $\phi_{MS}$ is applied over the $t$ streams, producing the feature matrix $H^{(l+1)} \in \mathbb{R}^{N \times d_{ms}^{(l+1)}}$ that contains the aggregated feature for each node in $N$. Here we use concatenation function as $\phi_{MS}$ for the hidden layer in the multi-stream aggregation, leading us to have $d_{l+1} = t \ast d_{ms}^{(l+1)}$. Specifically, for the output layer, pooling method is used instead and the details are provided in later paragraph. Accordingly, the updated propagation rule is provided in equation (11). Unlike the original GCN propagation in equation (9), we have $T$ streams of GCN learning in each layer, sharing the same input $H^{(l)}$ and propagating based on the $T$ adjacency matrices $ME^{(l)}$, which involves a set of layer and stream specific trainable weight matrices denoted as $W^{(l)}$. We also tried the shared-stream learning that shares the trainable weight matrices across each stream but found that separate stream-specific trainable weight matrices have better performance. The comparison of the two learning mechanisms is provided in Appendix Section[8]

$$H^{(l)} \xrightarrow{f_{MS}} H^{(l+1)} \xrightarrow{\phi_{MS}} H^{(l+1)}$$

(9)
### Evaluation Setup

We evaluate our ME-GCN on text classification in semi-supervised settings, and examine the effectiveness of corpus-based multi-dimensional edge features.

**Baselines:** We compare ME-GCN with state-of-the-art semi-supervised text classification models, which do not use any external resources. Additionally, we also include four baseline models, which use pretrained embedding or language model: CNN-Pretrained, LSTM-Pretrained, BERT, and TMix. 1)TF-IDF+LR, 2)TF-IDF+SVM: Term frequency inverse document frequency for feature engineering with Logistic Regression or SVM with rbf kernel. 3)CNN-Rand, 4)-Pretrained: Text-CNN \(^{(1)}\) is used as the classifier. Both CNN-Rand using random initialized word embedding and CNN-Pretrained using pretrained word embedding are evaluated. We used English Glove-pretrained \(^{[14]}\) and Chinese Word Vectors \(^{[18]}\) for Chinese dataset-zh. 5)LSTM-Rand, 6)-Pretrained: We apply the same set-up as the CNN, but with Long Short-Term Memory (LSTM). 7)TextGCN: We follow the same hyperparameters of the TextGCN \(^{(19)}\). \(^{8}\)BERT: We use huggingface \(^{(20)}\) BERT\(^{\text{BASE}}\) \(^{(19)}\) in our experiments ('bert-base-chinese' model is used for Chinese). 9)TMix: TMix \(^{(21)}\) generates new training text data by interpolating over labelled text encoded using BERT hidden representation and train on the generated text data for text classification. We use the default setting provided.

**Dataset:** We evaluated our experiments on five widely used text classification benchmark datasets \(^{(19)}\), 20NG, R8, R52, MR and Ohsumed, and three additional semi-supervised text classification datasets \(^{(19)}\), Agnews, Twitter nltk and Waimai. All the data is split based on the extreme low resource text classification environment- 1% training and 99% test set. The summary statistics of the datasets can be found in Appendix Table 3. For the data sample selection, we randomly select them but the class distribution is followed by the original datasets. 1)20NG is a 20-class news classification dataset and we select 3,000 samples from the original dataset. 2)R8, 3)R52 are from Reuters which is a topic classification dataset with 8 classes and 52 classes. 3,000 samples from each dataset are selected. 4)MR \(^{(22)}\) is a binary classification dataset about movie comments and we use all samples from the dataset. 5)Ohsumed is a medical dataset with 23 classes, and we select 3,000 samples from the original dataset. 6)Agnews \(^{(23)}\).

<table>
<thead>
<tr>
<th>Methods</th>
<th>Pretrained</th>
<th>20NG</th>
<th>R8</th>
<th>R52</th>
<th>Ohsumed</th>
<th>MR</th>
<th>Agnews</th>
<th>Twit nltk</th>
<th>Waimai (zh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF + SVM</td>
<td>✗</td>
<td>0.7246</td>
<td>0.6325</td>
<td>0.1889</td>
<td>0.5641</td>
<td>0.3825</td>
<td>0.6663</td>
<td>0.7261</td>
<td>0.8248</td>
</tr>
<tr>
<td>TF-IDF + LR</td>
<td>✗</td>
<td>0.7428</td>
<td>0.7249</td>
<td>0.6098</td>
<td>0.6636</td>
<td>0.6088</td>
<td>0.5737</td>
<td>0.7666</td>
<td>0.7261</td>
</tr>
<tr>
<td>CNN - Rand</td>
<td>✗</td>
<td>0.5791</td>
<td>0.6332</td>
<td>0.1798</td>
<td>0.5871</td>
<td>0.5730</td>
<td>0.5791</td>
<td>0.5458</td>
<td>0.6458</td>
</tr>
<tr>
<td>LSTM - Rand</td>
<td>✓</td>
<td>0.6811</td>
<td>0.7024</td>
<td>0.8248</td>
<td>0.7024</td>
<td>0.5737</td>
<td>0.5791</td>
<td>0.6458</td>
<td>0.7024</td>
</tr>
</tbody>
</table>

Table 1: Test accuracy comparison. The bottom row shows the best test accuracy from our proposed model using either max pooling or average pooling. The comparison of our model performance for each dataset using the three pooling methods is provided in Table 5. The second best is underlined.

The formula for combining node features over the streams for each node in layer \(l\) equals to the classification label number \(C\). Through max pooling, we select the best valued features over the \(T\) streams for each node in \(N\) before the final classification. We also tried other pooling and provide the comparison in Appendix Section E.

\[
H^{(l+1)} = \phi_{MS}(f_{MS}(H^{(l)}, ME^{(l)})) = \phi_{MS}(\sigma(\hat{M}E^{(l)}H^{(l)}W^{(l)}))
\]

(10)

Unlike the hidden layers where we use concatenation to aggregate the node features over each stream to continue propagation to next layer, we instead apply the pooling at the output layer to further synthesize the multi-stream features of each node to do the final classification.
Figure 2: Test accuracy by varying edge feature dimensions. The bottom right corner shows the average number of words per document in each corpus.

Table 1 presents a comprehensive performance experiment, conducted on the benchmark datasets. The most bottom row shows the accuracy from our best models using either max or average pooling.

Overall, our proposed model significantly outperforms the baseline models on all eight datasets, demonstrating the effectiveness of our ME-GCN on semi-supervised text classification for various length of text. With in-depth analysis, CNN/LSTM-Rand is quite low in performance on several datasets but increases significantly when using pretrained embeddings. While TextGCN achieves better accuracy than above baselines on most datasets, the performance is all lower than ME-GCN. This shows the efficiency of preserving rich information using multi-dimensional edge features. The merit of pre-training stands out with BERT and TMix, producing better accuracy than the baseline TextGCN on most datasets. Especially, BERT achieves the best and second best performance on MR and Waimai, which are short-text sentiment analysis datasets. This would be because of the two aspects of sentiment classification: (1) compared to topic-specific text classification, sentiment analysis task may benefit from the pretrained general semantics learned from a large external text; (2) word order matters for sentiment analysis, which could be missing in GNNs. Nevertheless, our ME-GCN, with no external resources, still outperforms those pretrained models in seven datasets, illustrating the potential superiority of self-exploration on the corpus via multi-dimensional edge graph in comparison of pretraining on large external resource.

5.2 Impact of Edge Feature Dimension

To evaluate the effect of the dimension size of the edge features, we tested ME-GCN with different dimensions. Figure 2 shows the test accuracy of our ME-GCN model on the four dataset, including R8, R52, MR, Waimai(zh). The bottom right corner for each subgraph includes the average number of the words per document. We noted that the test accuracy is related to the average number of words per document in the corpus. For instance, for ‘MR’ (avg #: 18.4), test accuracy first increases with the increase of the size of edge feature dimensions, reaching the highest value at 10; it falls when its dimension is higher than 15. However, for R8 and R52 (avg #: 84.2 and 104.5), got the highest value at 20 or 25. This is consistent with the intuition that the average number of words per document in the corpus should align with the dimension size of the edge features in ME-GCN. The trend is different in waimai dataset as it is Chinese, this is because different languages would have different nature of choosing the efficient edge feature dimension.

Moreover, in order to analyse the impact of the edge feature dimension, we present an illustrative visualisation of the document embeddings learned by ME-GCN. We use the t-SNE tool (Van der Maaten & Hinton, 2008) in order to visualise the learned document embeddings. Figure 3a and Figure 3b shows the visualisation of test set document embeddings in AgNews learned by ME-GCN
Figure 3: (a)(b)t-SNE visualisation of test set document embeddings in AgNews (4 classes). The (a) and (b) show second layer document embeddings learned by 5 and 25 dimensional node and edge features respectively. (c)(d) Test accuracy comparison with different number of labelled documents.

Table 2: Test accuracy comparison of our ME-GCN model with different word embedding techniques to train word node embeddings and word-word multi-dimensional edge features.

<table>
<thead>
<tr>
<th>Word Embedding</th>
<th>20NG</th>
<th>R8</th>
<th>R52</th>
<th>Ohsused</th>
<th>MR</th>
<th>Agnews</th>
<th>Twit nltk</th>
<th>Waimai(zh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec</td>
<td>0.2861</td>
<td>0.8679</td>
<td>0.7828</td>
<td>0.2740</td>
<td>0.6811</td>
<td>0.8043</td>
<td>0.8232</td>
<td>0.8393</td>
</tr>
<tr>
<td>fastText</td>
<td>0.2510</td>
<td>0.8394</td>
<td>0.7783</td>
<td>0.2550</td>
<td>0.6727</td>
<td>0.7812</td>
<td>0.8333</td>
<td>0.8191</td>
</tr>
<tr>
<td>GloVe</td>
<td>0.2526</td>
<td>0.8247</td>
<td><strong>0.7835</strong></td>
<td><strong>0.2832</strong></td>
<td><strong>0.6895</strong></td>
<td>0.7628</td>
<td><strong>0.8341</strong></td>
<td>0.8298</td>
</tr>
</tbody>
</table>

(5.3) IMPACT OF RATIO OF LABELLED DOCS

We choose 3 representative methods with the best performance from Table 1: CNN-Pretrained, TextGCN and our ME-GCN, in order to study the impact of the number of labelled documents. Particularly, we vary the ratio of labelled documents and compare their performance on the two datasets, Twitter nltk and R52, that have the smallest number and largest number of classes. Figure 3c and Figure 3d reports test accuracies with 1%, 10%, and 33% of the R52 and Twitter nltk training set. We note that our ME-GCN outperforms all other methods consistently. For instance, ME-GCN achieves a test accuracy of 0.8232 on Twitter nltk with only 1% training documents and a test accuracy of 0.8552 on R52 with only 10% training documents which are higher than other models with even the 33% training documents. It demonstrates that our method can more effectively take advantage of the limited labelled data for text classification.

5.4 COMPARISON OF EMBEDDING VARIANTS

ME-GCN apply a Word2Vec CBOW in order to train the word node embedding and the related multi-dimensional edge feature. We compare our model with three different word embedding techniques, Word2Vec, fastText, and Glove in Table 2. We noted that using Word2Vec and Glove, word-based models, is comparatively higher than applying the fastText, a character n-gram-based model. This would be affected because the node and edge of ME-GCN are based on words, not characters.

6 CONCLUSION

We introduced ME-GCN (Multi-dimensional Edge-enhanced Graph Convolutional Networks) for semi-supervised text classification, which takes full advantage of both limited labelled and large unlabelled data by rich node and edge information propagation. We propose corpus-trained multi-dimensional edge features to efficiently handle the distance/closeness between words and documents as multi-dimensional edge features, and all graph components are based on the given text corpus only. ME-GCN demonstrates promising results by outperforming numerous state-of-the-arts on eight semi-supervised text classification datasets consistently. In the future, it would be interesting to make this multi-aspect graph under inductive learning.
REFERENCES


Nuo Xu, Pinghui Wang, Long Chen, Jing Tao, and Junzhou Zhao. Mr-gnn: Multi-resolution and dual graph neural network for predicting structured entity interactions.


A Settings

A.1 Hyperparameter Setting

All documents are tokenized using NLTK tokenizer [Bird et al., 2009], and words occurring no more than 5 times have been excluded. Both word2vec and Dec2vec are trained on the corpus we get using gensim package with window size = 5 and iter = 200. The initial feature dimension for node and document is set to $d_0 = 25$, which is same to the multi-dimension number for edge features and multi-stream number $T$ in ME-GCN learning. Different multi-stream numbers are tested and discussed in Section 5.2. The threshold $u = 5$ is used for document-document edge construction. We use two-layers of multi-stream GCN learning with $d_{i_{ns}}^{l_1} = 25$ (thus $d_{i_{ns}}^{l_1} = 625$) for the first multi-stream GCN layer and $d_{i_{ns}}^{l_0} = C$(no. of label in the datasets) for the output layer. In the training process, following Liu et al. (2020), we use dropout rate as 0.5 and learning rate as 0.002 with Adam optimizer. The number of epochs is 2000 and 10% of the training set is used as the validation set for early stopping when there is no decreasing in validation set’s loss for 100 consecutive epochs.
### Hyperparameter Search

For each dataset we use grid search to find the best set of hyperparameters and select the base model based on the average accuracy by running each model for 5 times. The number of stream: 5, 10, 20, 25, 30, 40, 50. The document edge threshold: 3, 5, 10, 15. The pooling method: max pooling, min pooling, average pooling. The number of hyperparameter search trials is \(72 \times 6 \times 4 \times 3\) for each dataset. The best hyperparameters for each dataset and their average accuracy on test set shows in Table 4. And the trend of validation performance is very similar to the testing performance trend.

<table>
<thead>
<tr>
<th>Datasets</th>
<th># Doc</th>
<th># Words</th>
<th># Node</th>
<th># Class</th>
<th>Avg. length</th>
</tr>
</thead>
<tbody>
<tr>
<td>20NG</td>
<td>3,000</td>
<td>6,095</td>
<td>9,095</td>
<td>20</td>
<td>249.4</td>
</tr>
<tr>
<td>R8</td>
<td>3,000</td>
<td>4,353</td>
<td>7,353</td>
<td>8</td>
<td>84.2</td>
</tr>
<tr>
<td>R52</td>
<td>3,000</td>
<td>4,619</td>
<td>7,619</td>
<td>52</td>
<td>104.5</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>3,000</td>
<td>8,659</td>
<td>11,659</td>
<td>23</td>
<td>132.6</td>
</tr>
<tr>
<td>MR</td>
<td>10,662</td>
<td>4,501</td>
<td>15,163</td>
<td>2</td>
<td>18.4</td>
</tr>
<tr>
<td>Agnews</td>
<td>6,000</td>
<td>5,360</td>
<td>11,360</td>
<td>4</td>
<td>35.2</td>
</tr>
<tr>
<td>Twit nltk</td>
<td>3,000</td>
<td>634</td>
<td>3,634</td>
<td>2</td>
<td>11.5</td>
</tr>
<tr>
<td>Waimai(zh)</td>
<td>11,987</td>
<td>10,979</td>
<td>22,966</td>
<td>2</td>
<td>15.5</td>
</tr>
</tbody>
</table>

Table 3: The summary statistics of datasets

<table>
<thead>
<tr>
<th># Stream</th>
<th>20NG</th>
<th>R8</th>
<th>R52</th>
<th>Ohsumed</th>
<th>MR</th>
<th>Agnews</th>
<th>Twit nltk</th>
<th>Waimai(zh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Threshold</td>
<td>30</td>
<td>20</td>
<td>15</td>
<td>30</td>
<td>10</td>
<td>20</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>Pooling Method</td>
<td>avg</td>
<td>avg</td>
<td>max</td>
<td>avg</td>
<td>max</td>
<td>avg</td>
<td>max</td>
<td>max</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.2861</td>
<td>0.8679</td>
<td>0.7828</td>
<td>0.2740</td>
<td>0.6811</td>
<td>0.8043</td>
<td>0.8232</td>
<td>0.8393</td>
</tr>
</tbody>
</table>

Table 4: Best hyperparameters for each dataset

### Running Details

All the models are trained by using 16 Intel(R) Core(TM) i9-9900X CPU @ 3.50GHz and NVIDIA Titan RTX 24GB using Pytorch [Paszke et al., 2019].

The number of parameters for each part of the model is: Word Node (Word2vec): \(2UT\), Document Node (Doc2vec): \(2(T + K)\), ME-GCN Learning: \(T^2d_{l,s}^1(1 + C)\). The default value of \(d_{l,s}^1\) is 25. The number of parameters of TextGCN is \((U + K) \times D + D \times C\) and the default value of \(D\) is 200.

Comparison of the number of parameters between TextGCN and our ME-GCN shows in Figure 4.

Figure 4: Number of Parameters Comparison
## B Learning and Pooling Variant Testing

We compare ME-GCN with three different pooling approaches (max, average, and min pooling) and the result is shown in Table 5. Most datasets produce better results when using max pooling, and the result with max and average pooling outperforms that with min pooling. This is very obvious because the min pooling captures the minimum value of each graph component.

We also compare two multi-stream graph learning methods, including separated and shared stream learning to examine the effectiveness of ME-GCN learning with multi-dimensional edge features. Table 6 presents that the separated stream learners significantly outperforms the shared learners. This shows it is much efficient to learn each dimensional stream with an individual learning unit and initially understand the local structure, instead of learning all global structures at once.

### C Links Related to Datasets and Baseline Models

The links for Datasets:

- **R8, R52**: [https://www.cs.umb.edu/~smimarog/textmining/datasets/](https://www.cs.umb.edu/~smimarog/textmining/datasets/)
- **Ohsumed**: [http://disi.unitn.it/moschitti/corpora.htm](http://disi.unitn.it/moschitti/corpora.htm)
- **Twitter nltk**: [http://nltk.org/howto/twitter.html](http://nltk.org/howto/twitter.html)
- **Waimai**: [https://github.com/SophonPlus/ChineseNlpCorpus/](https://github.com/SophonPlus/ChineseNlpCorpus/)

The links for Baseline Models:

- **TextCNN**: [https://github.com/DongjunLee/text-cnn-tensorflow](https://github.com/DongjunLee/text-cnn-tensorflow)
- **TextGCN**: [https://github.com/yao8839836/text_gcn](https://github.com/yao8839836/text_gcn)
- **BERT BASE**: [https://huggingface.co/bert-base-uncased](https://huggingface.co/bert-base-uncased)
- **Tmix**: [https://github.com/GT-SALT/MixText](https://github.com/GT-SALT/MixText)
- **Chinese BERT**: [https://huggingface.co/bert-base-chinese](https://huggingface.co/bert-base-chinese)
- **GloVe-pretrained**: [https://nlp.stanford.edu/projects/glove/](https://nlp.stanford.edu/projects/glove/)

The tokenizer used:

- **English Tokenizer - NLTK**: [https://www.nltk.org/api/nltk.tokenize.html](https://www.nltk.org/api/nltk.tokenize.html)
- **Chinese Tokenizer - Jieba**: [https://github.com/fxsjy/jieba](https://github.com/fxsjy/jieba)

---

### Table 5: Test accuracy of ME-GCN with three different pooling methods, max, average, and min pooling

<table>
<thead>
<tr>
<th>Pooling Method</th>
<th>20NG</th>
<th>R8</th>
<th>R52</th>
<th>Ohsumed</th>
<th>MR</th>
<th>Agnews</th>
<th>Twit nltk</th>
<th>Waimai(zh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Pooling</td>
<td>0.2775</td>
<td>0.8473</td>
<td>0.7828</td>
<td>0.2475</td>
<td>0.6811</td>
<td>0.3043</td>
<td>0.8232</td>
<td>0.8393</td>
</tr>
<tr>
<td>Avg Pooling</td>
<td>0.2861</td>
<td>0.8679</td>
<td>0.7448</td>
<td>0.2740</td>
<td>0.6658</td>
<td>0.3043</td>
<td>0.8232</td>
<td>0.8393</td>
</tr>
<tr>
<td>Min Pooling</td>
<td>0.0424</td>
<td>0.2987</td>
<td>0.2550</td>
<td>0.0294</td>
<td>0.5000</td>
<td>0.2005</td>
<td>0.5000</td>
<td>0.6663</td>
</tr>
</tbody>
</table>

### Table 6: Test accuracy of ME-GCN with two multi-stream learning methods, shared and separated learners.

<table>
<thead>
<tr>
<th>Learning Methods</th>
<th>20NG</th>
<th>R8</th>
<th>R52</th>
<th>Ohsumed</th>
<th>MR</th>
<th>Agnews</th>
<th>Twit nltk</th>
<th>Waimai(zh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separated Learning</td>
<td>0.2861</td>
<td>0.8679</td>
<td>0.7828</td>
<td>0.2740</td>
<td>0.6811</td>
<td>0.3043</td>
<td>0.8232</td>
<td>0.8393</td>
</tr>
<tr>
<td>Shared Learning</td>
<td>0.1582</td>
<td>0.8016</td>
<td>0.6554</td>
<td>0.2635</td>
<td>0.6575</td>
<td>0.6993</td>
<td>0.7037</td>
<td>0.8137</td>
</tr>
</tbody>
</table>