applied sciences

MDPI

Article

A Tensor Space Model-Based Deep Neural Network for Text Classification

Han-joon Kim * and Pureum Lim

School of Electrical and Computer Engineering, University of Seoul, 163 Seoulsiripdaero, Seoul 02504, Korea; lim.pr0623@gmail.com

* Correspondence: khj@uos.ac.kr; Tel.: +82-2-6490-2339

Abstract: Most text classification systems use machine learning algorithms; among these, naïve Bayes and support vector machine algorithms adapted to handle text data afford reasonable performance. Recently, given developments in deep learning technology, several scholars have used deep neural networks (recurrent and convolutional neural networks) to improve text classification. However, deep learning-based text classification has not greatly improved performance compared to that of conventional algorithms. This is because a textual document is essentially expressed as a vector (only), albeit with word dimensions, which compromises the inherent semantic information, even if the vector is (appropriately) transformed to add conceptual information. To solve this 'loss of term senses' problem, we develop a concept-driven deep neural network based upon our semantic tensor space model. The semantic tensor used for text representation features a dependency between the term and the concept; we use this to develop three deep neural networks for text classification. We perform experiments using three standard document corpora, and we show that our proposed methods are superior to both traditional and more recent learning methods.

Keywords: text classification; machine learning; deep learning; tensor space model; neural networks; concept embedding

Citation: Kim, H.-j.; Lim, P. A Tensor Space Model-Based Deep Neural Network for Text Classification. *Appl. Sci.* **2021**, *11*, 9703. https://doi.org/10.3390/app11209703

Academic Editor: Giancarlo Mauri

Received: 21 September 2021
Accepted: 16 October 2021
Published: 18 October 2021

1. Introduction

Text classification is a text mining technology that automatically classifies a given textual document into appropriate categories; such classification is widely used to detect spam emails, to classify news, to answer questions, to perform emotional analyses, and to power chat bots. In general, text classification systems use machine learning algorithms, among which naïve Bayes and support vector machine (SVM) algorithms adapted to handle text data afford reasonable performance [1]. Recently, recurrent neural networks (RNNs) or convolutional neural networks (CNNs) have been used to improve text classification systems. For instance, Liu et al. developed a complex architecture for text classification using an RNN termed Long Short Term Memory (LSTM) [2]. Conneau et al. sought to improve classification performance using a CNN that employed character-level units as inputs [3]. Recently, as text classification has become increasingly important, many studies have focused on text embedding [4–8], which is closely related to text classification (because performance depends on how specifically such embedding represents words or documents).

Here, we develop three types of concept-driven deep neural networks (DNNs) featuring concept-embedding; we use our tensor space model (TSM) for text classification. In this model, the 'concept space' is an independent space corresponding to the 'term' and 'document' spaces used in the vector space model. Figure 1 shows a sample matrix representation of a document in the TSM, in which the term-by-concept matrix yields information on various concepts embedded in the document. Using this form of document representation, semantic features of concepts associated with the literal features of

terms significantly improve text classification. Here, we markedly improve classification performance using DNNs that consider the semantic information of the TSM.

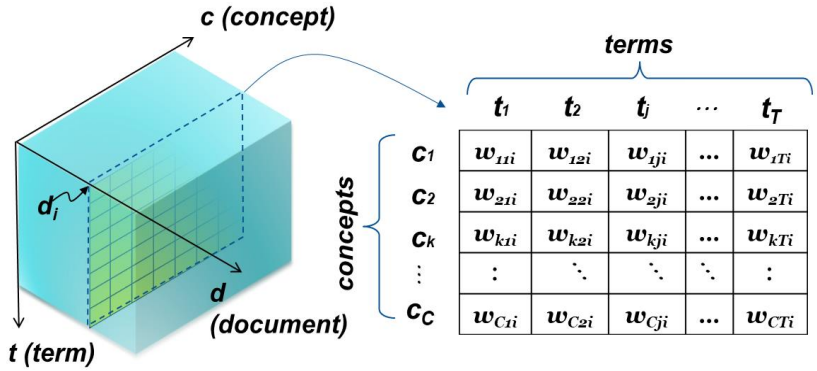

Figure 1. Our tensor space model for document representation.

2. Related Work

2.1. Deep Learning-Based Text Classification

Given recent developments in deep learning technology, various neural networks have been employed for text classification. Luong et al. [9] sought to improve classification performance using a recursive neural network that receives two pairs of words in vector form (in the order in which they appear in the document) and returns a word vector of the same size as the input vector, and a score. Then, after combining high-scoring words, the combined vector replaces the two words. Such scoring and replacement are repeated until there are no scores left to compare; this reveals the hierarchical structure of, and the semantic information in, the document.

Liu et al. [2] developed three complex LSTM architectures that considered a text as sequenced data. Lee et al. [10] sought to improve classification using text representations obtained employing an RNN and a CNN. Conneau et al. [3] sought to improve performance using a very deep CNN (VDCNN) to input data in character units (unlike the existing methods wherein data are pre-processed in units of words). Lai et al. [11] prepared a so-called RCNN (a combination of an RNN and a CNN) to improve text classification performance.

Recent text-embedding models are pre-trained using large text corpora and then fine-tuned employing target data; such training was first used when deriving the Word2Vec and ELMo models. Word2Vec, a two-layer neural network with autoencoder characteristics, was trained to learn the relationships among words in a sentence and then found semantic representations [6]. ELMo finds deep contextualized word representations using an RNN-based bi-directional language model [7]. Despite the use of text corpora that are unrelated to the target tasks, pre-training helps the models to understand the general features of natural language. During pre-training, the models predict the meanings of words based on the context. For instance, Word2Vec is trained to predict words that surround the central word in a sentence. The GPT and BERT models (stacked transformer blocks) were pre-trained using large text corpora and exhibited excellent performance when asked to perform various natural language processing tasks [4,8].

2.2. The Tensor Space Model for Text Representation

Our tensor space model (TSM) includes the dimension space 'concept', as well as the spaces 'document' and 'term'. Thus, a document is expressed as a matrix rather than a vector, and a set of documents becomes a 3rd-order tensor. The 3rd-order tensor text model for text mining proposed in Reference [12] is similar to our model in that a 'term' space is created by folding term vectors. However, it is not easy to determine the terms required for the new space, and the intrinsic limitations of using terms as semantic units remain. In addition, Reference [13] proposed a tensor space model in which 'structure' information

was a space that was independent of the document and term spaces; this method sought to represent structural information, thus not seeking word meanings, when clustering XML documents. Reference [14] created another tensor model (an extension of an n-gram model) in which each n-gram string is mapped to an nth-order tensor space, each dimension of which corresponds to one of the 26 English letters and a representative special letter. Unfortunately, most n-gram substrings are usually invalid words; thus, their semantic meanings will remain elusive. Moreover, the model requires a great deal of memory when handling only a small number of documents.

In fact, our TSM is more closely related to the means used to map documents or terms onto a concept space. In an earlier approach, latent semantic indexing (a variant of a classical vector space) was used in an attempt to produce a concept space for document indexing by capturing latent (hidden) concepts [15]. Several past studies on derivation of the correct word meanings in Wikipedia articles have markedly improved text mining algorithms [16–18]. Reference [17] sought to semantically represent documents with Wikipedia articles; significant terms in documents were identified and their meanings represented in terms of Wikipedia-based concepts. Boubacar and Niu [16] sought to improve document clustering by enriching document representation using concept-level Wikipedia articles. Similarly, Wang et al. [18] tried to improve text classification by expanding the vector space model to include semantic relationships, such as synonymy, hyponymy, and associative relationships derived from Wikipedia.

3. Concept-Driven Deep Neural Networks for Text Classification

We develop a concept-driven DNN optimized for our TSM to improve text classification. We first introduce our semantic TSM and then describe the concept-driven DNN that features the TSM and concept embedding.

3.1. The Semantic Tensor Space Model

Our semantic TSM is a human knowledge-based, text embedding model that represents a document as a two-dimensional matrix of terms and concepts (Figure 1). A term is a word in a document, and a concept is a newly defined feature, thus being semantic information that humans use when thinking about a word. We use Wikipedia articles as the semantic units of documents. Basically, a concept is defined by a combination of an intent and an extent in the theory of formal concept analysis (FCA) [19]; the 'extent' is the set of instances included in the concept, and the 'intent' is the set of common attributes of the instances included in the extent. In our work, the extent that represents a concept consists of a set of documents related to the concept; the intent is a set of keywords extracted from that set of documents. As shown in Figure 1, by combining the concepts of FCA theory and the conventional document representation term-by-document matrix, we build a concept-by-term-by-document 3rd-order tensor. In other words, the 'concept' space is regarded as an independent space with the same status as the 'term' and 'document' spaces.

As mentioned above, in our TSM model, a document is represented by a 2nd-order tensor of term and concept; the tensor is implemented by creating additional concept vectors. A concept vector expresses the relevance of each term with respect to all concepts inherent within a single document. Figures 2–4 show the detailed creation of a concept vector. The first step is to create a concept window based on the Lesk algorithm (or the 'gloss overlap') [20]. Basically, terms that occur in similar contexts may have similar meanings, and terms occurring with a target term can be considered to enrich the sense of that term. Hence, we first create a sliding window that is used to evaluate all concepts for a target (center) term, while considering the surrounding terms. Figure 2 shows the concept window for the target term 'java' with its surrounding words. The concept window is composed of the 'java' with surrounding terms, such as 'application', 'bytecode', 'virtual', and 'machine'. Figure 3 shows how we weight the concept vectors. Although the target term 'java' has several senses (i.e., a language, a coffee, an island), the concept window allows us to derive

its current context by examining all relevant concepts. To weight all concepts for a target term, the overlap between the concept window of a target term in context and the terms in a Wikipedia article corresponding to each concept is calculated. If each of the terms in the concept window occurs in a specific concept, a pre-calculated TF-IDF value for that term is added to its weight; for example, the weight of the concept 'java (programming language)' for the term 'java' becomes the average value (i.e., $\frac{\sum_{t=1}^{N} w_t}{N} = \frac{10.01}{5} \approx 2.0$, where N denotes the number of terms within the concept window, and w_t denotes the TF-IDF value of the tth term.) of the TF-IDF values for five words included in the concept window centered on the term 'java'. Then, as shown in Figure 4, the several senses of a target term can be quantitatively represented as a concept vector, and as a result, a document as a concept-by-term 2nd-order, and a set of documents as a concept-by-term-by-document 3rd-order tensor.

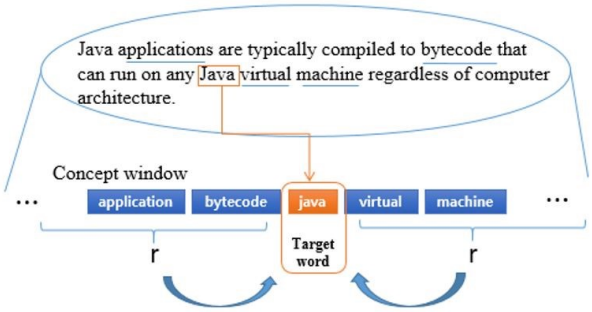

Figure 2. Concept window for generating concept vectors.

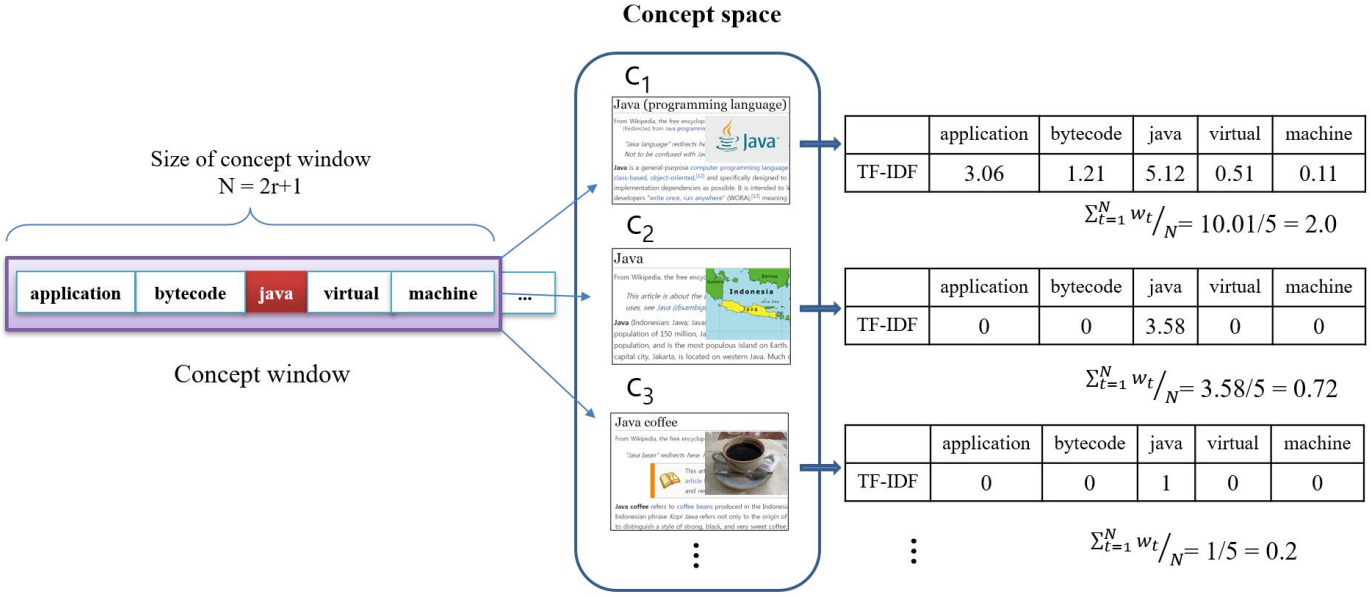

Figure 3. Generating a concept vector with weighting.

Furthermore, to represent a particular set of documents as the proposed 3rd-order tensor, an appropriate concept space should be automatically constructed. Firstly, we select only about 250,000 pages with more than 100 incoming links to eliminate the low quality of Wikipedia articles. The selected articles are then indexed by a search engine, such as ElasticSearch, which calculates the TF-IDF weight of each term occurring in the selected articles. Next, we generate a search query that consists of only highly weighted terms within each of the articles. For each article, its corresponding query is then submitted to the search engine, and the results of that query are returned from the search engine to serve as candidates to configure a concept space. Lastly, to construct a final concept space that accommodates the given set of documents, significant Wikipedia articles with a high

document concept frequency (that refers to the frequency of a particular concept in the entire set of documents) are selected from among the candidates, and each of the Wikipedia articles is defined as a dimension of the concept space.

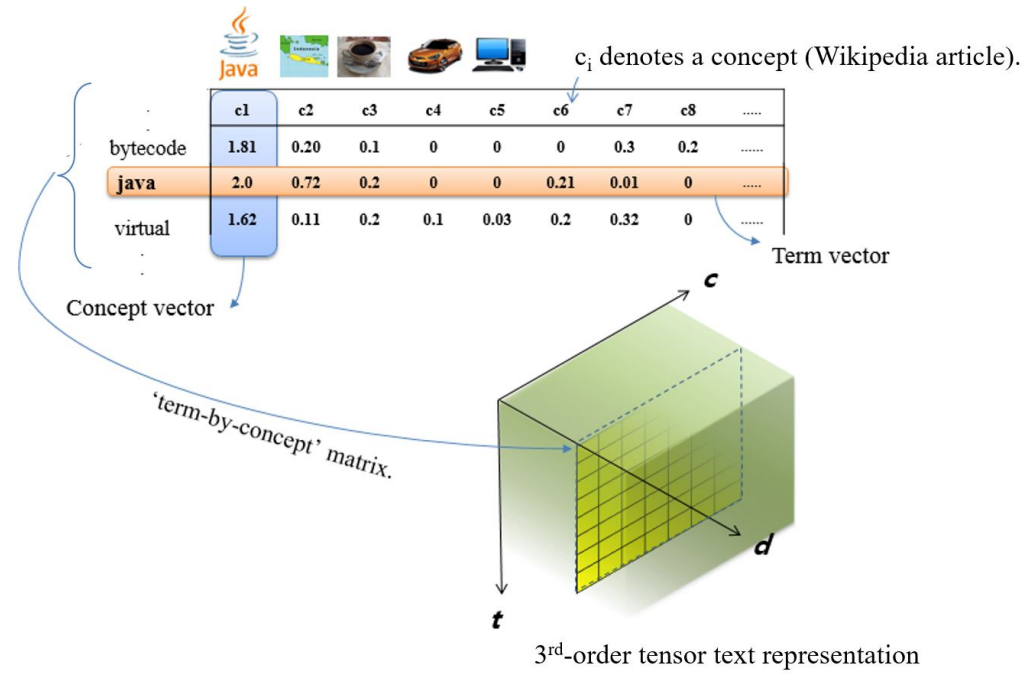

Figure 4. Generating a concept-by-term-by-document tensor.

3.2. Embedding of Concepts and Documents

Broadly, our neural network for text classification features two front layers for concept and document embedding; each concept is embedded by inputting the corresponding concept vector from the term-by-concept matrix into two fully connected layers, as shown in Figure 5; document embedding then proceeds by collecting the concept embeddings (as when creating a concept space).

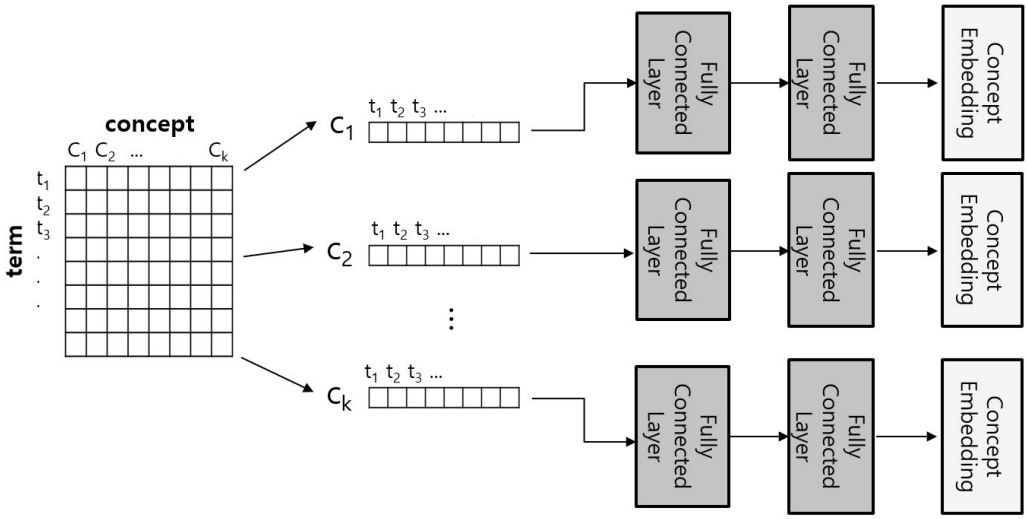

Figure 5. The concept embedding process.

Unlike the Bag-of-Words and Word2Vec models often used for machine learning-based text classification, our TSM contains an additional concept space, as well as a term space. Thus, it is inappropriate to deliver document data collected by the TSM to

conventional DNNs that handle only term-based data. Thus, we added concept embedding to our new concept-driven DNN. As shown in Figure 5, concept embedding proceeds by dividing the TSM text matrix by concept and then passing the split vectors through two fully-connected layers. The collection of all concept embeddings is termed 'document embedding', as shown in Figure 6. Below, we describe a new DNN featuring concept and document embedding.

Let d_i be the term-by-concept matrix of a document i. Then, a jth concept vector c_j within d_i is expressed as $c_j = d_i[:, j]$. Concept embedding E_{c_j} of the concept c_j proceeds as follows:

$$y_{c_j}^{(1)} = f^{(1)}(w_{c_j}^{(1)} c_j + b_{c_j}^{(1)}), \tag{1}$$

$$y_{c_j}^{(2)} = f^{(2)}(w_{c_j}^{(2)} y_{c_j}^{(1)} + b_{c_j}^{(2)}), \tag{2}$$

$$\cdots$$

$$E_{c_j} = f^{(n)}(w_{c_j}^{(n)} y_{c_j}^{(n-1)} + b_{c_j}^{(n)}), \tag{3}$$

where $y_{c_j}^{(n)}$, $w_{c_j}^{(n)}$, $b_{c_j}^{(n)}$, and $f^{(n)}$ denote the output value, the weight, the bias, and the activation function of the nth layer when performing the jth concept embedding, respectively. The above formula does not connect concepts when engaging in concept embedding. Thus, as each concept of the tensor space model is independent, concept embedding is performed individually.

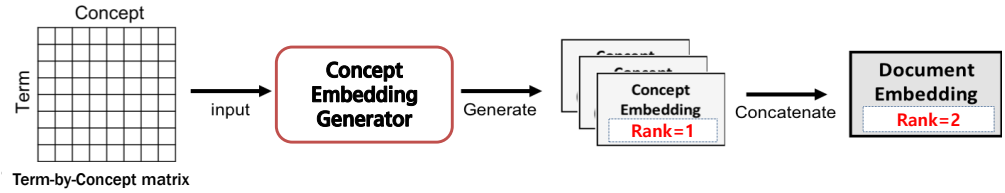

Figure 6. The document embedding process.

Next, in the document embedding step, the concept embedding vectors generated in the previous step are combined to create a two-dimensional document embedding matrix. The document embedding E_i for document i is expressed as follows:

$$E_i = [E_{c_1}; E_{c_2}; \cdots; E_{c_{n-1}}; E_{c_n}]. \tag{4}$$

Algorithm 1 demonstrates the pseudo code for the concept and document embeddings described above.

Algorithm 1 Document embedding

Input: A term-by-concept matrix d_i for a document i
Output: A document embedding E_i for a document i

$nc = d_i.numOfConcepts()$;
for $j \leftarrow 0$ *to* nc **do**
  $c_j \leftarrow d_i[:, j]$; // c_j denotes the jth concept (column) in the matrix d_i.
  $outL_1 \leftarrow L_1.output(c_j)$; //$L_k$ denotes the kth layer for concept embedding.
  $outL_2 \leftarrow L_2.output(outL_1)$;
  $\cdots$
  $E_{c_j} \leftarrow L_n.output(outL_{n-1})$;
  $E_i[j, :] \leftarrow E_i[j, :] + E_{c_j}$; //Each E_{c_j} vector is incrementally combined to create E_i.
end for
return E_i

3.3. The TSM-Based Deep Learning Architecture for Text Classification

Basically, our proposed neural network embeds text documents expressed in a term-by-concept matrix for each concept, and the combined document embedding is submitted to fully connected layers. In this section, we describe three types of autonomous neural networks appropriate for TSMs. Figures 7–9 show the new concept-driven DNNs that best accommodate our semantic TSM for text classification.

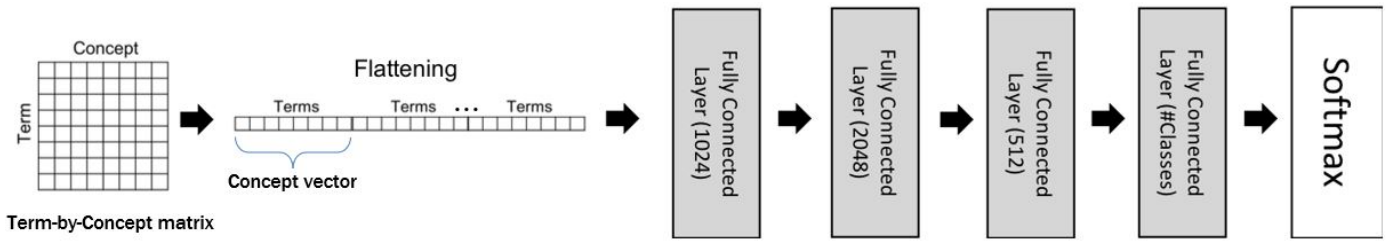

Figure 7. TSM-based deep neural network I: TSM-DNN.

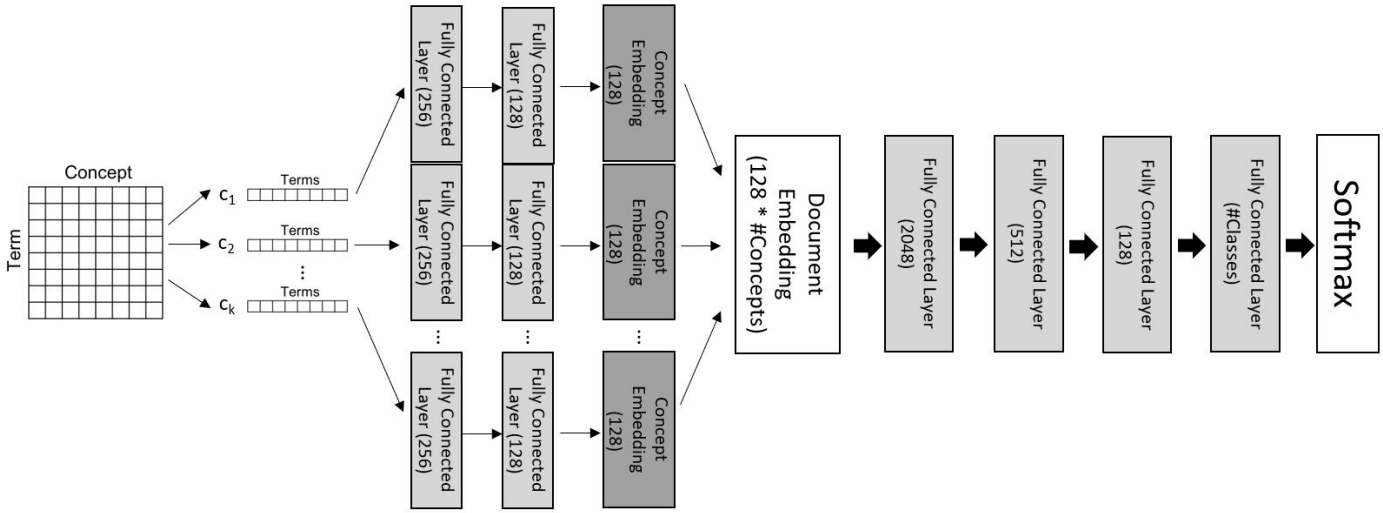

Figure 8. TSM-based deep neural network II: Concept-wise TSM-DNN.

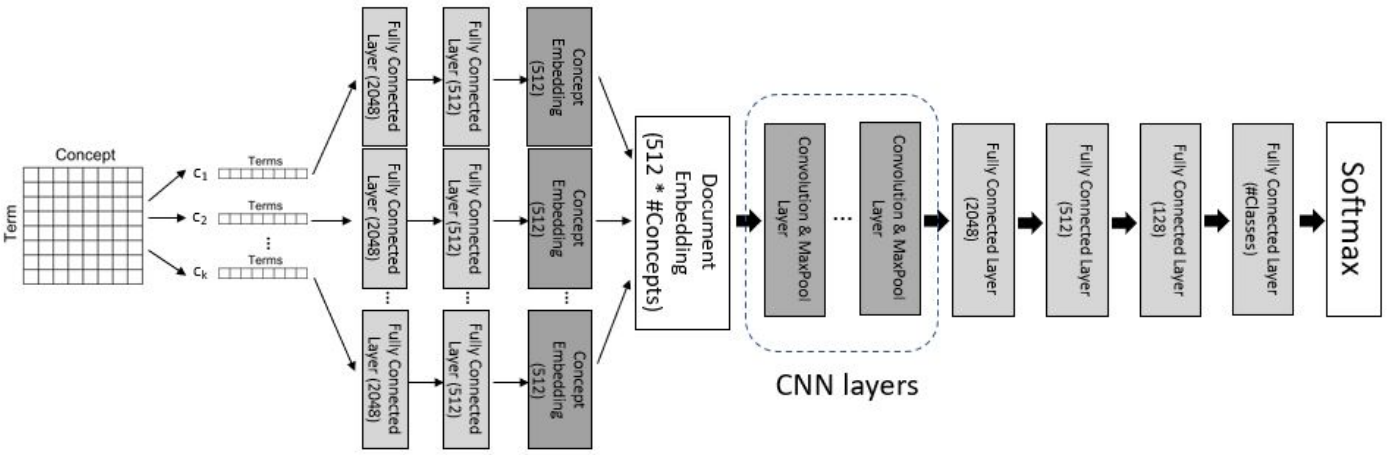

Figure 9. TSM-based deep neural network III: Concept-wise TSM-CNN.

- The first architecture ('TSM-DNN') is shown in Figure 7. In this architecture, the term-by-concept matrix for a given document is flattened and connected to form a single one-dimensional vector and then passed through multiple fully connected layers.

- The second architecture ('concept-wise TSM-DNN') is shown in Figure 8. This includes both the concept and document embedding layers introduced in Section 3.2. In other words, after creating a concept embedding vector for each concept, all concept embedding vectors are collected in two-dimensional form to create a document embedding.
- The third architecture ('concept-wise TSM-CNN') is shown in Figure 9. This is similar to the second architecture, except that multiple convolutional layers are added after the document embedding layer. In general, convolutional layers afford high performance when used to classify two- or three-dimensional image data. As document embedding is two-dimensional, we expected that classification would be improved on inclusion of convolutional layers. The width of the first such layer equals the length of document embedding, and the filter size is set to 3. When the number of convolutional layers is excessive, the number of hyper-parameters is too large to optimize. Hence, it is desirable to control the number of convolutional layers using one-dimensional Max-pooling with the stride set to 2.

In short, all three deep learning architectures perform text classification using fully connected layers that employ a document embedding matrix as an input. Unlike existing deep learning-based classification methods that employ units of terms, our proposed architectures fully utilize the conceptual information contained in the TSM. The TSM-DNN is connected to a basic neural network by concatenating the vectors of concept units, and the concept-wise TSM-DNN and concept-wise TSM-CNN create and combine new concept embedding vectors and then forward them to fully connected networks.

4. Experiments

4.1. Empirical Setup

For reliable evaluation of the proposed method, we used three controlled subsets: 20Newsgroups, Reuters-21578, and OHSUMED document corpora, which are known to be clean and are commonly used to evaluate various text classification methods. The 20Newsgroups corpus contains approximately 20,000 documents partitioned across 20 different newsgroups. We selected 10,000 of the top-ranked largest documents. Next, we generated a subset of the Reuters-21578 corpus so that documents can be not skewed over classes. For this, we first selected documents belonging to the the five most common classes and then chose the 10,000 largest documents with a single class (thus, we avoided the ambiguities of documents on multiple classes). Finally, we used the OHSUMED corpus that is a subset of clinically-oriented MEDLINE database and includes medical abstracts from the 23 Medical Subject Headings (MeSH) categories. Many studies have shown that this corpus is difficult to classify properly; thus, it is very challenging to achieve a high classification performance. From this corpus, we selected 3600 documents of the nine most common classes. To ensure that performance evaluation was reliable, we performed 5-fold cross-validation.

For comparison, we selected six conventional methods, of which two were traditional machine learning methods, including an SVM and naïve Bayes, and the others were recent deep learning techniques, including a DNN [21], a CNN [22], and a VDCNN [3]. The traditional machine learning methods use TF-IDF values as inputs after pre-processing textual documents. For the SVM, the Radial Basis Function (RBF) served as the kernel, and the penalty parameter C (for misclassification) was set to 1.0. We used a Multinomial naïve Bayes appropriate for textual data, and the parameter α for Laplace smoothing was set to 1.0. When using the DNN, textual documents were transformed into Bag-of-Words inputs. The DNN architecture featured seven fully connected layers, with as many neurons as input words, 1024, 2048, 4096, 1024, 256, and two neurons, respectively. ReLU served as the activation function, and the Softmax function was applied by the last layer for text classification. The DNN architecture was trained using the Adam optimization method with a batch size of 64 over 10 epochs. The CNN architecture featured three fully connected layers lying after two convolutional layers and two Max-pooling layers (in that order).

The kernel size of the first convolutional layer was set to 3, the number of filters to 8, and the zero-padding to 1. For the second convolutional layers, the figures were 3, 32, and 1. For the Max-pooling layer, the kernel size was 3, the zero-padding 1, and the stride 2. The outcomes of the layers were flattened into one-dimensional forms and then submitted to the fully connected layers. The three such layers featured 512, 64, and two neurons (in that order). Text classification proceeded using ReLU as the activation function and with application of the Softmax function to the output of the last layer. The CNN architecture was trained using the Adam optimization method with a batch size of 32 over 10 epochs.

The VDCNN [3] features four depths; we selected depth17 and depth29 because they afforded the best performance. The VDCNN classifies documents by inputting text data in character units; the data are not transformed into Bag-of-Words format (or the like). The maximum number of document characters is 1014; thus, the input data size is fixed. Momentum was employed for optimization; the learning rate was 0.01; and the batch size was 64 over 50 epochs.

4.2. Tuning the Deep Learning Architectures

We now focus on various parameters of our new deep learning architectures for text classification. First, to create TSM tensors, we set the size of the concept space to 30, and extracted 1000∼5000 meaningful terms from each dataset. For example, if 1000 terms were selected, each document was represented as a 1000×30 matrix. In the TSM-DNN architecture, the fully connected layers featured 1024, 2048, 512, and two neurons (in that order). We employed the LeakyReLU activation function, and performed batch normalization (to avoid overfitting) after each fully connected layer generated an output. The output of the last fully connected layer was submitted to the Softmax function for text classification. In the concept-wise TSM-DNN architecture, the two fully connected layers for concept embedding featured 256 and 128 neurons, respectively. As a result, document embedding was in the form of a two-dimensional tensor of size 128×30. Then, the document embedding tensor was flattened and connected to four fully connected layers with 2048, 512, 128, and two neurons (in that order). Batch normalization was performed after each fully connected layer created an output, and LeakyReLU served as the activation function. In the last fully connected layer, text classification was performed using the Softmax function.

The concept-wise TSM-CNN employs convolutional layers to reduce the size of the feature map; thus, the concept embedding size was quadrupled to avoid information loss. Therefore, the numbers of neurons in the two fully connected layers for concept embedding were set to 2048 and 512, respectively, so that the size of document embedding became 512×30. The document embedding then passed through a one-dimensional convolutional layer of kernel size 3, zero-padding 1, and with 64 and 128 filters. Each convolutional layer was connected to a one-dimensional Max-pooling layer with a kernel size of 3, a stride of 2, and zero-padding of 1. The four fully connected layers featured 2048, 512, 128, and two neurons, respectively. The concept-wise TSM-CNN featured batch normalization to avoid overfitting; the activation function was ReLU. In the last fully connected layer, text classification was performed using the Softmax function. All of our architectures used Adam for optimization; the learning rate was 0.01, and the mini-batch size was 32.

4.3. Performance Evaluation

As shown in Table 1, the classification models trained using our architectures performed best; the classifications of the Reuters-21578 and 20Newsgroup data were near-perfect. The proposed neural network architectures, including concept embedding, contributed greatly to constructing a high-performance text classification model; the concept-wise TSM-DNN classification model and the TSM-DNN classification model were evaluated to have almost the same classification performance, and the concept-wise TSM-CNN classification model showed the best performance. This experimental result implies that, as the meaning of terms appearing in a given document became clearer through concept

embedding, our deep learning architecture dramatically improved the text classification performance. Moreover, it suggests that the convolutional layers within the concept-wise TSM-CNN can further enhance the effectiveness of concept embedding and document embedding. Similar to why CNN architecture is very effective for image classification, we analyze that the concept-wise TSM-CNN model showed the best performance by extracting meaningful local patterns that help text classification from a document embedding matrix with a structure similar to image data.

The accuracies of the comparative methods were over 90% for the Reuters-21578 data. In contrast, for the OHSUMED data, which is difficult to classify automatically, our methods were much better than the other methods. Furthermore, although we used fewer convolutional layers than the DNN, CNN, and VDCNN, our architectures exhibited high-level classification accuracy. The VDCNN classifications of the OHSUMED datasets were very poor; the classification accuracy severely decreased when the number of selected terms increased.

Table 1. Performance comparisons in terms of accuracy.

	SVM	Naïve Bayes	DNN	CNN	VDCNN Depth-17	VDCNN Depth-29	TSM DNN	Concept-Wise TSM-DNN	Concept-Wise TSM-CNN
Reuters-21578	0.9073	0.9104	0.9700	0.9823	0.9853	0.9853	0.9952	0.9956	0.9965
20Newsgroup	0.9245	0.8792	0.7220	0.8653	0.9216	0.9290	0.9597	0.9562	0.9667
OHSUMED	0.7665	0.7279	0.5546	0.6574	0.5954	0.4898	0.8400	0.8461	0.8522

When using the TSM-CNN architecture, it is necessary to define an optimal number of convolutional layers. Figure 10 shows the changes in classification accuracy when the number of convolutional layers for the OHSUMED dataset was varied in the concept-wise TSM-CNN. The classification accuracy for the OHSUMED data varied dramatically by the number of layers. A concept-wise TSM-CNN with eight convolutional layers afforded the best accuracy (85.2%); classification performance decreased with increasing numbers of layers. We conjecture that the classification model remains under-fitted because the training data do not increase when the classifier capacity increases to create a deeper network. A neural network with more than eight convolutional layers is of large capacity, associated with overfitting of the training data and, thus, poorer classification performance. The optimal layer number may differ for different sets of training data; however, we found that the optimal number was eight for all 3 datasets.

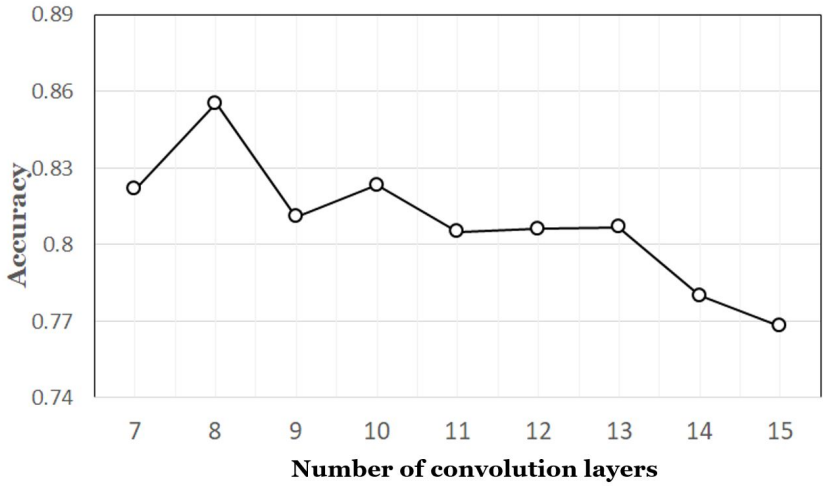

Figure 10. Changes in classification accuracy on variation of the number of convolutional layers in the concept-wise TSM-CNN.

5. Conclusions

We developed new, concept-driven deep learning architectures optimized for our semantic TSM of text representation, and we tried to improve text classification. We first introduce concept and document embedding using a Wikipedia-based representative TSM model. We integrate both embeddings with DNN and CNN architectures; we devise three types of deep learning architectures that enable very reliable text classification; and we have solved the 'loss of term senses' problem. Basically, our TSM for text representation features dependencies between terms and concepts in the form of two-dimensional matrices; such tensors greatly improve text classification. Our new deep learning architectures afforded near-perfect classification of the Reuters-21578 and 20Newsgroup datasets. In addition, we observed that simply increasing the number of convolutional layers in the CNN architecture did not improve the performance of deep learning-based text classifiers.

The disadvantage of our proposed method is that it requires an external knowledge base, such as Wikipedia, to construct a 3rd-order tensor. Therefore, as a future study, we intend to improve the proposed deep learning architecture for text classification so that it does not depend on an external knowledge base; for this, we can use pre-trained word embedding vectors, such as GloVe (Global Vectors for Word Representation) [23]. In addition, we will enhance the concept embedding process so that the proposed architecture can generate a high-performance classification model for textual documents of various domains, including SNS text data.

Author Contributions: All authors made contributions to this work. H.-j.K. contributed to the organization of the research as well as the final manuscript preparation. P.L. proposed the original idea, conducted data processing, and wrote the original draft of this work. Conceptualization, P.L. and H.-j.K.; methodology, P.L.; software, P.L.; validation, P.L. and H.-j.K.; formal analysis, P.L. and H.-j.K.; investigation, P.L.; resources, H.-j.K.; data curation, P.L.; writing—original draft preparation, P.L.; writing—review and editing, H.-j.K.; visualization, P.L.; supervision, H.-j.K.; project administration, H.-j.K.; funding acquisition, H.-j.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A1A02086148), and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00121, Development of data improvement and dataset correction technology based on data quality assessment). In addition, this research was also supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2018-08-01417) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Korde, V.; Mahender, C.N. Text classification and classifiers: A survey. *Int. J. Artif. Intell. Appl.* **2012**, *3*, 85–99.
2. Liu, P.; Qui, X.; Huang, X. Recurrent Neural Network for Text Classification with Multi-Task Learning. *arXiv* **2016**, arXiv:1605.05101.
3. Conneau, A.; Schwenk, H.; Barrault, L.; Lecun, Y. Very deep convolutional networks for text classification. *arXiv* **2016**, arXiv:1606.01781.
4. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
5. Fei-Fei, L.; Perona, P. A bayesian hierarchical model for learning natural scene categories. In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 20–25 June 2005; Volume 2, pp. 524–531.
6. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed representations of words and phrases and their compositionality. In Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013; pp. 3111–3119.
7. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.

8. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://blog.openai.com/language-unsupervised (accessed on 21 September 2021).
9. Luong, M.T.; Socher, R.; Manning, C.D. Better word representations with recursive neural networks for morphology. In Proceedings of the Seventeenth Conference on Computational Natural Language Learning, Sofia, Bulgaria, 8–9 August 2013; pp. 104–113.
10. Lee, J.Y.; Dernoncourt, F. Sequential short-text classification with recurrent and convolutional neural networks. *arXiv* **2016**, arXiv:1603.03827.
11. Lai, S.; Xu, L.; Liu, K.; Zhao, J. Recurrent convolutional neural networks for text classification. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015; pp. 2267–2273.
12. Cai, D.; He, X.; Han, J. Tensor space model for document analysis. In Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, WA, USA, 6–11 August 2006; pp. 625–626.
13. Kutty, S.; Nayak, R.; Li, Y. XML documents clustering using a tensor space model. In Proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Shenzhen, China, 24–27 May 2011; pp. 488–499.
14. Liu, N.; Zhang, B.; Yan, J.; Chen, Z.; Liu, W.; Bai, F.; Chien, L. Text representation: From vector to tensor. In Proceedings of the 5th IEEE International Conference on Data Mining, Houston, TX, USA, 27–30 November 2005; pp. 4–10.
15. Liu, T.; Chen, Z.; Zhang, B.; Ma, W.Y.; Wu, G. Improving text classification using local latent semantic indexing. In Proceedings of the 4th IEEE International Conference on Data Mining, Brighton, UK, 1–4 November 2004; pp. 162–169.
16. Boubacar, A.; Niu, Z. Conceptual Clustering. In *Future Information Technology*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 1–8.
17. Gabrilovich, E.; Markovitch, S. Wikipedia-based semantic interpretation for natural language processing. *J. Artif. Intell. Res.* **2009**, *34*, 443-498. [CrossRef]
18. Wang, P.; Hu, J.; Zeng, H.J.; Chen, Z. Using Wikipedia knowledge to improve text classification. *Knowl. Inf. Syst.* **2009**, *19*, 265–281. [CrossRef]
19. Wille, R. Formal concept analysis as mathematical theory of concepts and concept hierarchies. In *Formal Concept Analysis*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 1–33.
20. Lesk, M. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In Proceedings of the 5th ACM International Conference on Systems Documentation, Toronto, ON, Canada, 8–11 June 1986; pp. 24–26.
21. Liu, W.; Wang, Z.; Liu, X.; Zeng, N.; Liu, Y.; Alsaadi, F.E. A survey of deep neural network architectures and their applications. *Neurocomputing* **2017**, *234*, 11–26. [CrossRef]
22. Jacovi, A.; Shalom, O.S.; Goldberg, Y. Understanding convolutional neural networks for text classification. *arXiv* **2018**, arXiv:1809.08037.
23. Pennington, J.; Socher, R.; Manning, C.D. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.