Design and Implementation of a Mind Map Generation System Based on TextCNN

Ying Tian[1], Yang Zhao[1], Wanyue Liu[1,2,*]
[1]School of Artificial Intelligence and Big Data, Henan University of Technology, Zhengzhou, Henan, China
[2]iFLYTEK Co., Ltd., Hefei, Anhui, China
**Corresponding Author*

Abstract: In the process of organizing course materials, it is necessary to extract operation steps, core concepts, review points and other contents manually, which requires a lot of time and may lead to problems such as repetition and omission. In this study, in order to solve this problem, the TextCNN system is used to create a mind map. First, the system processes the CSV data and the course text, and inputs the processed data into the model in the form of tensors, so as to obtain a series of numerical information. Secondly, the system uses one-dimensional convolution kernel to extract local semantic features and complete text classification. Finally, the system generates a variety of forms of maps, including timelines and tree maps, and can also generate mind maps. This system has a simple structure, fast training speed, and does not require high-performance hardware, which can meet the needs of visual presentation of course texts.

Keywords*:* TextCNN; Text Classification; Knowledge Visualization; Mapdocument; Classification Mind Map Generation System.

1. Introduction

In the process of organizing knowledge and reviewing, students need to deal with a large number of unstructured texts. The learning materials are relatively long, the knowledge points are scattered, the chapters are not clear, and the hierarchy is not prominent. If students only rely on manual sorting, they can not only get key knowledge, but also deepen their understanding of the operation process. This is a time-consuming task, and it is easy to omit some key content and repeatedly arrange the same content, so we need to use the system to analyze the text and form a mind map, which has high application value.

The system is designed for course material organization [1]. Its goal is to automatically transform course texts, teaching handouts, or uploaded files into visual mind maps. The backend parses, cleans, extracts keywords, and organizes the text, while the frontend renders the result as a mind map. The system also supports node editing, theme switching, template switching, history saving, favorites, search, and recycle-bin recovery. With these functions, users can understand course knowledge structures more intuitively and improve review efficiency and knowledge summarization ability. As shown in Figure 1, users can enter text or upload course files.

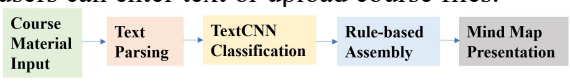

Figure 1. Application Scenario Diagram.

TextCNN is a classic convolutional neural network model for text classification and is suitable for short-text and paragraph-level classification tasks. It extracts local semantic features through multi-scale convolution kernels and uses classification results to judge whether a text belongs to course objectives, core concepts, mechanisms, operation steps, code examples, review highlights, and other knowledge-node types [2].

2. Overall Design

This project uses the TNEWS Chinese short-text classification dataset from CLUE Benchmark as the training and validation data for the TextCNN model [3]. The original TNEWS files are in JSON format, and each sample mainly contains sentence, label, and label_desc fields. The sentence field stores the news short text, label stores the original category ID, and label_desc stores the category name. Although the original task of the dataset is news classification, its texts are short and its categories are clear, making it suitable for

verifying the ability of TextCNN to classify Chinese short texts.

The original data are converted into CSV format so that the model training script can read them conveniently. The processed CSV fields include text, label, raw_label, label_name, split, and source. The specific field design is shown in Table 1.

Table 1. Processed Data Fields

Field	Example	Description
text	Where can I watch The Long Road Home?	Sentence read bythe TextCNNmodel forclassification
label	9	Re-encodednumeric label
label raw	110	Original label IDin the TNEWSdatasete
label name	news military	Original label IDin the TNEWSdatasete
split	train	Dataset splite
source	CLUE TNEWS	Dataset source

After preprocessing, 28,928 valid samples are obtained, including 24,768 training samples and 4,160 validation samples. The original TNEWS dataset contains 15 news categories, such as news_story, news_culture, news_sports, news_finance, news_house, news_car, news_edu, news_tech, and so on. The training set is used for model parameter learning, and the validation set is used to evaluate classification performance. The data preparation process is shown in Figure 2.

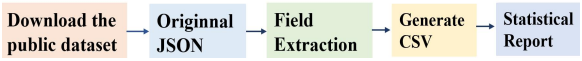

Figure 2. Flowchart of Data Preparation

The system rejiebads the data in JSON format, extracts the description, label and sentence from the data, generates a CSV file, and writes the sample into it. After this step, a unified data format can be provided for the next processing and training of the model.

This module is responsible for extracting data from TNEWS JSON files, which need to be read in a reasonable way. In the process of operation, the program can traverse JSON files, judge whether the file is a training set or a verification set according to the file name, and generally skip test files because there is no real label in the public test set, so as to avoid the problems caused by unlabeled samples.The text, labels and other contents of the sample are also excluded, and only valid data can be input into the model for training.

In order to meet the training requirements of the model, it is necessary to re-encode the original label.In this way, the original TNEWS label ID can be transformed into a digital label, and the starting number is 0, for example, the original label is 100, 101, 102, which can be transformed into 0, 1, 2 and so on.In the process of training, it is necessary to use cross-entropy loss function, which requires that the labels can be continuous integers. In addition, the raw_label field is also set to record the original label, and the data is saved in the form of CSV file after processing.

A unified CSV schema is designed to improve the coupling degree between the data preparation stage and the model training stage. In this way, the training script can read the structured CSV file and no longer needs to process complex JSON files.

3. Data Preprocessing

The goal of data preprocessing is to convert Chinese short texts in the CSV file into numerical tensors that can be accepted by TextCNN. Since TextCNN cannot directly process raw Chinese sentences, the texts must go through cleaning, word segmentation, vocabulary construction, text encoding, and fixed-length padding. The processing procedure is shown in Figure 3:

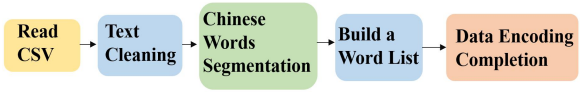

Figure 3. Data Preprocessing Process.

In this project, blank characters are first normalized and the text is cleaned. Jieba is then used for Chinese word segmentation [4]. If jieba is not installed in the current environment, the program falls back to character-level segmentation so that the code can still run. After segmentation, a vocabulary is built from the training set, and two special tokens, <PAD> and <UNK>, are added. <PAD> is used for sequence padding, and <UNK> represents words that do not appear in the vocabulary.

The text segmentation function compresses redundant blank characters through regular expressions and then preferentially uses jieba.lcut() to segment Chinese short texts. If the segmentation result is empty or jieba is unavailable, character-level splitting is used as a backup. This design improves the robustness of the program and prevents the training process from being interrupted by the absence

of a segmentation tool.

The vocabulary construction function builds a vocabulary according to words appearing in the training set. In the vocabulary, index 0 is fixed for <PAD>, and index 1 is fixed for <UNK>. This design ensures that padding tokens and unknown tokens keep consistent meanings during training, validation, and prediction.

Therefore, the input of TextCNN is not the raw Chinese sentence, but a fixed-length numerical sequence.

The system first builds a vocabulary from the words appearing in the training set and defines two special tokens, <PAD> and <UNK>, where <PAD> is used for padding short texts and <UNK> represents words not included in the vocabulary. After word segmentation, the original text is mapped to a sequence of word IDs, such as [35, 128, 76, 240, 0, 0, 0, ...] for "Artificial intelligence technology is developing rapidly". In this sequence, each number represents a word in the vocabulary, and 0 indicates the <PAD> token used for padding. After this processing, all text samples are unified to the same length, making them suitable for input into the TextCNN model for training and classification.

4. Model Construction

This project adopts TextCNN as the text classification model. TextCNN is a convolutional neural network model for text classification. Its core idea is to use one-dimensional convolution kernels of different sizes to extract local n-gram features [5]. For example, a convolution kernel of size 3 focuses on local semantic features formed by three consecutive words, while kernels of sizes 4 and 5 capture longer phrase features. The features extracted by different kernels are max-pooled and concatenated, and then fed into a fully connected layer for classification.

The TextCNN model in this project mainly consists of an Embedding layer, one-dimensional convolution layers, ReLU activation functions, max-pooling layers, a Dropout layer, and a fully connected classification layer [6,7]. The model input is the fixed-length word-ID sequence generated in the preprocessing stage, and the output is the logits scores corresponding to 15 classes. The structure is shown in Figure 4.

A configuration class is used to centrally manage the parameters of the TextCNN model, and the main model code defines the network layers of TextCNN.

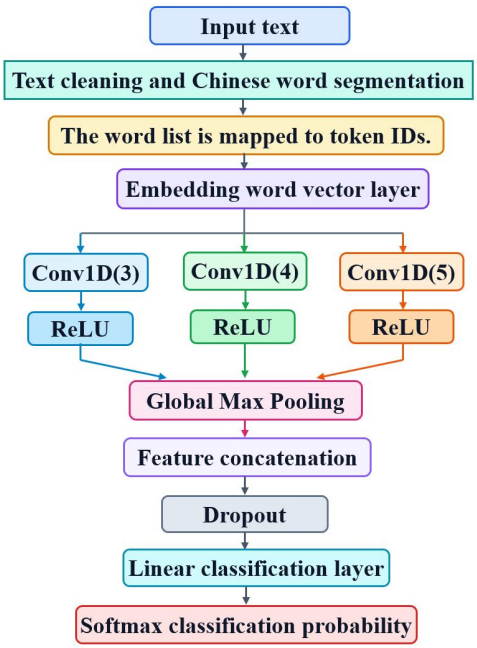

Figure 4. TextCNN Model Architecture

The forward propagation process can be divided into five steps. First, embedding(input_ids) converts the input word-ID sequence into a word-vector matrix. Second, transpose(1, 2) adjusts the tensor dimensions to meet the input requirements of Conv1d. Third, convolution kernels of different sizes extract local text features, and ReLU activation enhances nonlinear representation Fourth, max pooling selects the most important feature from each convolution channel. Fifth, the features obtained by different kernels are concatenated and then passed through Dropout and the fully connected layer to output the classification results.

The basic training process of TextCNN is as follows. The model is first created and moved to the computing device, and the cross-entropy loss function is used to calculate the error between predictions and true labels. AdamW is adopted as the optimizer to update model parameters. In each training round, the model obtains logits from the input text, calculates the loss, performs backpropagation through loss.backward(), and updates parameters with optimizer.step().

5. Mind Map Generation System

The proposed system is designed for course material organization and knowledge visualization. Its main objective is to help users automatically organize knowledge from course

texts, learning materials, or uploaded files into mind maps with clear hierarchy, thereby improving the efficiency of knowledge sorting, review, and presentation. Unlike a pure text classification model, this system focuses more on the application scenario, namely how to transform unstructured text into a visual and hierarchical knowledge structure.

From the functional perspective, the mind map generation system includes text input, file upload, content parsing, mind map generation, map display, node editing, and result saving. Users can directly enter course topics or course texts on the frontend page, or upload Word, PDF, PPT, and other course materials. After receiving the input, the backend extracts, cleans, and organizes the text, and then generates node data required by the mind map according to system rules or model interfaces. The frontend renders the central topic, first-level nodes, and second-level nodes according to the structured data returned by the backend.

The frontend interaction layer is responsible for user operations such as text input, file upload, mind map display, and node editing. User requests are then sent to the backend service layer, where the API interface handles communication, file parsing extracts the uploaded content, rule extraction organizes the information, and storage saves the related data. The system architecture is shown in Figure 5.

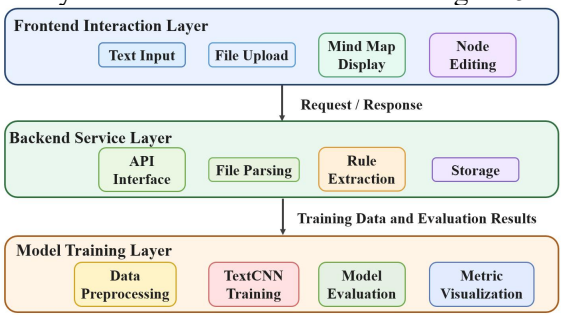

Figure 5. System Functional Architecture Diagram.

The model training layer focuses on the machine learning workflow. It begins with data preprocessing, where raw text is cleaned and transformed into model-ready input. The processed data are then used for TextCNN training, followed by model evaluation and metric visualization. In this way, the system connects user interaction, backend processing, and deep learning training into a complete pipeline for generating and analyzing mind maps.

6. Model Training

The training process of TextCNN includes several steps: data reading, text cleaning, word segmentation, vocabulary construction, text encoding, model training, result saving and so on. When training, the system reads the processed CSV file, converts the text into a sequence of fixed length, and uses PyTorch to build Dataset and DataLoader objects to provide unified input for evaluation and training. In the training stage, the system should be controlled from multiple aspects, including learning rate, batch size and other parameters, so as to obtain ideal training results. If the configuration is reasonable, the data required for training can be saved, and the performance of the validation set can be improved.Therefore, in the process of training, the parameters of TextCNN model should be adjusted to the appropriate level, which can not only ensure the convergence speed, but also bring good classification performance. See Table 2 for the training configuration.

Table 2. Training Configuration

Configuration-Iteme	Value
batch size	64
eval batch size	128
learning ratee	0.001
optimizer	Adam
weight_decay	0.0001
weight_decay	CrossEntropyLosse
max len	64
class weightse	balanced
class weightse	1.0

From the above table, we know that in order to improve the evaluation efficiency, the training batch size is 64, the validation batch size is 128, the learning rate is 0.001, and Adam is used as the optimizer. This optimizer can automatically adjust the update of parameters according to the gradient changes.The loss function is set to CrossEntropyLoss, which can deal with multi-class text classification problems and has strong applicability. The weight_decay value is 0.0001, which can effectively regularize the model parameters and avoid overfitting.64 is the length of each text, which is padded or truncated to 64 tokens. Class weights are balanced to reduce the impact of class imbalance, and 1.0 is the gradient clipping, which can improve the stability of training.

During the training process, the model first performs forward propagation and outputs probability scores for each class. These

predictions are then compared with the true labels, and the cross-entropy loss function is used to measure the difference between them. After backpropagation, the model updates its parameters to improve classification performance. Meanwhile, evaluation metrics such as accuracy, recall, and precision are calculated on the validation set to assess the model's generalization ability. The training and validation accuracy curves of the model over 74 epochs are shown in Figure 6.

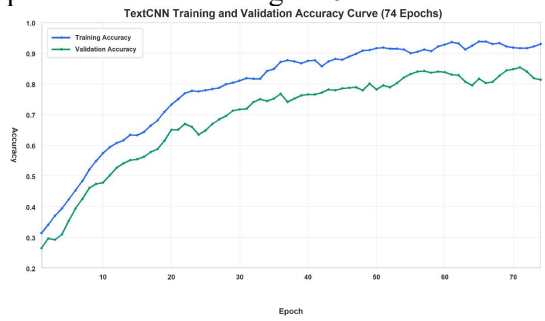

Figure 6. Accuracy Curves of the Training and Validation Sets

At the beginning of training, the training and validation accuracy values are both low, indicating that the model has not fully learned text features. As the number of epochs increases, the training accuracy continues to rise, and the validation accuracy also gradually improves. In the later stage, the training accuracy exceeds 0.9, while the validation accuracy stabilizes around 0.8, indicating that the model can distinguish different text categories effectively. The training and validation loss curves of the model over 74 epochs are shown in Figure 7.

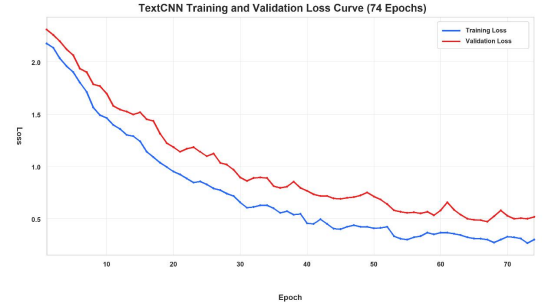

Figure 7. Loss Curves of the Training and Validation Sets

The training loss and validation loss both show a downward trend. The high loss at the beginning indicates a large prediction error. As training proceeds, the loss decreases significantly, showing that the model classification ability is gradually enhanced. In the later stage, the training loss is lower than the validation loss, which means that the model fits the training set more fully. However, the validation loss remains relatively low, indicating a certain degree of generalization ability.

During model training, the number of epochs, validation metrics, and overfitting control all affect the final model performance. To ensure stable and generalizable results, the accuracy and loss of the training and validation sets are recorded in each epoch, and the convergence state is judged according to the two curves. Training termination, early stopping, overfitting behavior, and improvement measures are summarized in Table 3.

Table 3. Training Termination and Overfitting Analysis

item	Descriptione
Training termination conditione	Training ends after the presetnumber of epochs,and-convergenceis judged according tovalidationAccuracyand-Loss.
Training termination conditione	Training ends after the presetnumber of epochs,and-convergenceis judged according tovalidationAccuracyand-Loss.
Over fitting criterion	Training ends after the presetnumber of epochs,and-convergenceis judged according tovalidationAccuracyand-Loss.
Current over fitting statuse	Training ends after the presetnumber of epochs,and-convergenceis judged according tovalidationAccuracyand-Loss.
Current over fitting statuse	Dropout=0.5, weight_decay=0.0001, and-balancedclass-weights are-used toreduce.overfitting and-class imbalance.

Although automatic early stopping is not used in this experiment, the validation accuracy and loss curves show that the model becomes stable in the later stage of training. There is a certain gap between training and validation metrics, but no serious overfitting appears. The current Dropout, weight decay, and balanced class-weight settings improve the generalization ability of the model to some extent.

7. Model Performance Analysis

To comprehensively evaluate the TextCNN model, this project uses Accuracy, Macro Precision, Macro Recall, Macro F1, and Weighted F1. Accuracy indicates the overall proportion of correct predictions. Precision measures the proportion of samples predicted as a certain class that truly belong to that class. Recall measures the proportion of real samples

in a certain class that are correctly identified. F1 score reflects the balance between precision and recall. The results are shown in Figure 8.

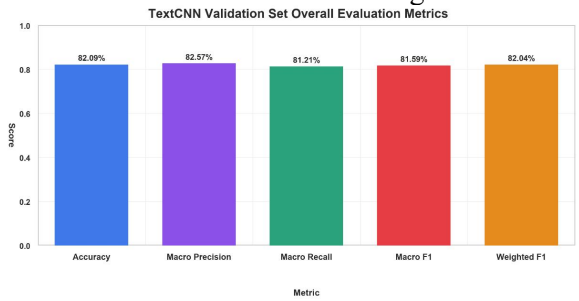

Figure 8. Overall Validation Metrics of TextCNN

Figure 8 shows that the TextCNN model achieves an accuracy of 82.09% on the validation set. Macro Precision, Macro Recall, Macro F1, and Weighted F1 are 82.57%, 81.21%, 81.59%, and 82.04%, respectively. All overall metrics exceed 80%, indicating that the model has good classification performance on the Chinese short-text classification task [8].

The model achieves generally strong and stable performance across the knowledge-node categories, with most Precision, Recall, and F1 scores clustered around 0.8 or higher. The close alignment among the three metrics indicates robust classification performance and suggests that most categories have relatively distinct textual patterns, as shown in Figure 9.

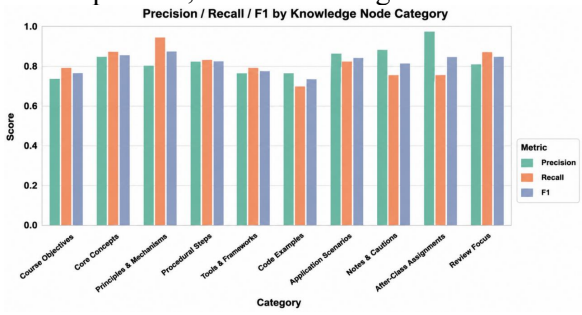

Figure 9. Scores for Different Knowledge Node Categories

Figure 9 presents the Precision, Recall, and F1 scores of different knowledge-node categories. Categories such as core concepts, mechanisms, application scenarios, after-class exercises, and review highlights perform well, showing that their textual features are relatively clear and easy for the model to identify. The scores of course objectives and code examples are slightly lower, possibly because their expressions are more flexible and may overlap with other categories.

Overall, TextCNN can complete the classification of course text fragments in a stable way. Although the model does not directly generate a complete mind map, it provides classification evidence for the system and helps distinguish different types of knowledge content during mind map generation.

8. System Demonstration

The frontend of the system is implemented with React, TypeScript, and Vite. It is responsible for course text input, file upload, mind map rendering, node editing, template switching, theme switching, and history management [9]. The backend uses FastAPI to provide RESTful API interfaces, receive frontend requests, parse uploaded files, call the mind map generation logic, save history data, and return standardized MapDocument data.

The left side of the system page is the history management area, the middle is the mind map canvas, and the right side is the style editor. This layout reflects the overall system design: the left area manages maps, the middle area displays visualization results, and the right area supports editing operations. Users can view maps, select nodes, adjust styles, and switch history records on the same page, forming a complete operation process.

Users can create mind maps, view history, switch templates, edit nodes, modify styles, view node descriptions, and export results. The TextCNN model mainly serves as an auxiliary module for text-fragment classification and node categorization in the system, helping the system determine the knowledge-node type of course content [10]. The final output is a visual, editable, and storable course knowledge mind map, which can meet the basic needs of course material organization and review summarization.

9. Conclusion

This project focuses on the TextCNN text classification model and its application in a mind map generation system. It first selects the TNEWS Chinese short-text classification dataset from CLUE Benchmark as experimental data, extracts fields from the original JSON files, cleans the data, encodes labels, and divides the dataset to form standardized data for model training. Then a TextCNN model is constructed with word embedding, convolution, pooling, and fully connected classification structures to realize automatic Chinese short-text classification.

Future improvements include the following aspects. First, course-domain datasets should be expanded, and knowledge-point classification labels that better fit the system scenario should be established. Second, richer evaluation metrics should be introduced to evaluate both model classification performance and mind map generation quality. Third, file upload and parsing ability should be optimized to support complex PPT files, scanned PDFs, and documents with mixed text and images. Fourth, the frontend editing experience should be improved so that node layout, style adjustment, and export results become more stable. Fifth, TextCNN can be combined with other text representation methods to improve semantic representation of course texts.

In general, this project completes the design goal of combining deep learning model training with practical web system functions. The system not only demonstrates the application process of TextCNN in Chinese short-text classification, but also introduces classification ability into mind map generation, supporting course material organization, knowledge structure sorting, and learning review. Through this design, the project improves the implementation ability of deep learning models as well as comprehensive abilities in frontend and backend development, interface debugging, and integrated application design.

Acknowledgments

This research was supported by the Undergraduate Innovation Training Program (No. 202610463006).

References

[1] Reales D, Manrique R, Grévisse C. Core concept identification in educational resources via knowledge graphs and large language models. SN Computer Science, 2024, 5(8): 1029.

[2] Zhang Y, Wallace B C. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification//Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2017: 253-263.

[3] Xu L, Hu H, Zhang X, et al. CLUE: A Chinese language understanding evaluation benchmark//Proceedings of the 28th international conference on computational linguistics. 2020: 4762-4772.

[4] Lin C, Lin Y J, Yeh C J, et al. Improving multi-criteria Chinese word segmentation through learning sentence representation//Findings of the Association for Computational Linguistics: EMNLP 2023. 2023: 12756-12763.

[5] Kim Y. Convolutional neural networks for sentence classification//Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014: 1746-1751.

[6] Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines//Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 807-814.

[7] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 2014, 15(1): 1929-1958.

[8] Zeng G. Invariance properties and evaluation metrics derived from the confusion matrix in multiclass classification. Mathematics, 2025, 13(16): 2609.

[9] Lu J, Dong X, Yang C. Weakly supervised concept map generation through task-guided graph translation. IEEE transactions on knowledge and data engineering, 2023, 35(10): 10871-10883.

[10] Stamper J, Gaind B, Thankachan K, et al. Hierarchical concept map generation from course data//AAAI 2023 Workshop on Artificial Intelligence in Education (AI4Edu). 2023.

