RandolphVI / Multi-Label-Text-Classification

About Muti-Label Text Classification Based on Neural Network.

#text-classification #python3 #tensorflow #sentence-classification #multi-label-classification

23 commits | 1 branch | 0 packages | 0 releases | 1 contributor | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

RandolphVI Update param_parser.py		Latest commit a303af1 on 14 Apr
ANN	Update code to TensorFlow 1.14 version	2 months ago
CNN	Update code to TensorFlow 1.14 version	2 months ago
CRNN	Update code to TensorFlow 1.14 version	2 months ago

FastText	Update code to TensorFlow 1.14 version	2 months ago
HAN	Update code to TensorFlow 1.14 version	2 months ago
RCNN	Update code to TensorFlow 1.14 version	2 months ago
RNN	Update code to TensorFlow 1.14 version	2 months ago
SANN	Update code to TensorFlow 1.14 version	2 months ago
data	Update data sample.	2 months ago
utils	Update param_parser.py	2 months ago
.gitignore	Update .gitignore	2 months ago
.travis.yml	Update .travis.yml	2 months ago
LICENSE	Initial commit	14 months ago
README.md	Update README.md	2 months ago
Usage.md	Update Usage.md	2 months ago
requirements.txt	Update code to TensorFlow 1.14 version	2 months ago

README.md

Deep Learning for Multi-Label Text Classification

language python3.6 build passing code quality A license Apache-2.0 issues 7 open

This repository is my research project, and it is also a study of TensorFlow, Deep Learning (Fasttext, CNN, LSTM, etc.).

The main objective of the project is to solve the multi-label text classification problem based on Deep Neural Networks. Thus, the format of the data label is like [0, 1, 0, ..., 1, 1] according to the characteristics of such a problem.

Requirements

- Python 3.6
- Tensorflow 1.4
- Numpy
- Gensim

Project

The project structure is below:

```
.
├── Model
│   ├── test_model.py
│   ├── text_model.py
│   └── train_model.py
├── data
│   ├── word2vec_100.model.* [Need Download]
│   ├── Test_sample.json
│   ├── Train_sample.json
│   └── Validation_sample.json
└── utils
│   ├── checkmate.py
│   ├── data_helpers.py
│   └── param_parser.py
├── LICENSE
├── README.md
└── requirements.txt
```

Innovation

Data part

1. Make the data support **Chinese** and English (Can use `jieba` or `nltk`).
2. Can use **your pre-trained word vectors** (Can use `gensim`).
3. Add embedding visualization based on the **tensorboard** (Need to create `metadata.tsv` first).

Model part

1. Add the correct **L2 loss** calculation operation.
2. Add **gradients clip** operation to prevent gradient explosion.
3. Add **learning rate decay** with exponential decay.
4. Add a new **Highway Layer** (Which is useful according to the model performance).
5. Add **Batch Normalization Layer**.

Code part

1. Can choose to **train** the model directly or **restore** the model from the checkpoint in `train.py` .
2. Can predict the labels via **threshold** and **top-K** in `train.py` and `test.py` .
3. Can calculate the evaluation metrics --- **AUC** & **AUPRC**.
4. Can create the prediction file which including the predicted values and predicted labels of the Testset data in `test.py` .
5. Add other useful data preprocess functions in `data_helpers.py` .
6. Use `logging` for helping to record the whole info (including **parameters display**, **model training info**, etc.).
7. Provide the ability to save the best n checkpoints in `checkmate.py` , whereas the `tf.train.Saver` can only save the last n checkpoints.

Data

See data format in `/data` folder which including the data sample files. For example:

```
{"testid": "3935745", "features_content": ["pore", "water", "pressure", "metering", "device", "incorporating",
```

- **"testid"**: just the id.
- **"features_content"**: the word segment (after removing the stopwords)
- **"labels_index"**: The label index of the data records.
- **"labels_num"**: The number of labels.

Text Segment

1. You can use `nltk` package if you are going to deal with the English text data.

2. You can use `jieba` package if you are going to deal with the Chinese text data.

Data Format

This repository can be used in other datasets (text classification) in two ways:

1. Modify your datasets into the same format of the sample.
2. Modify the data preprocessing code in `data_helpers.py` .

Anyway, it should depend on what your data and task are.

🤔**Before you open the new issue about the data format, please check the `data_sample.json` and read the other open issues first, because someone maybe ask me the same question already. For example:**

- 输入文件的格式是什么样子的?
- Where is the dataset for training?

- 在 data_helpers.py 中的 content.txt 与 metadata.tsv 是什么，具体格式是什么，能否提供一个样例?

Pre-trained Word Vectors

You can download the Word2vec model file (dim=100). Make sure they are unzipped and under the `/data` folder.

You can pre-training your word vectors (based on your corpus) in many ways:

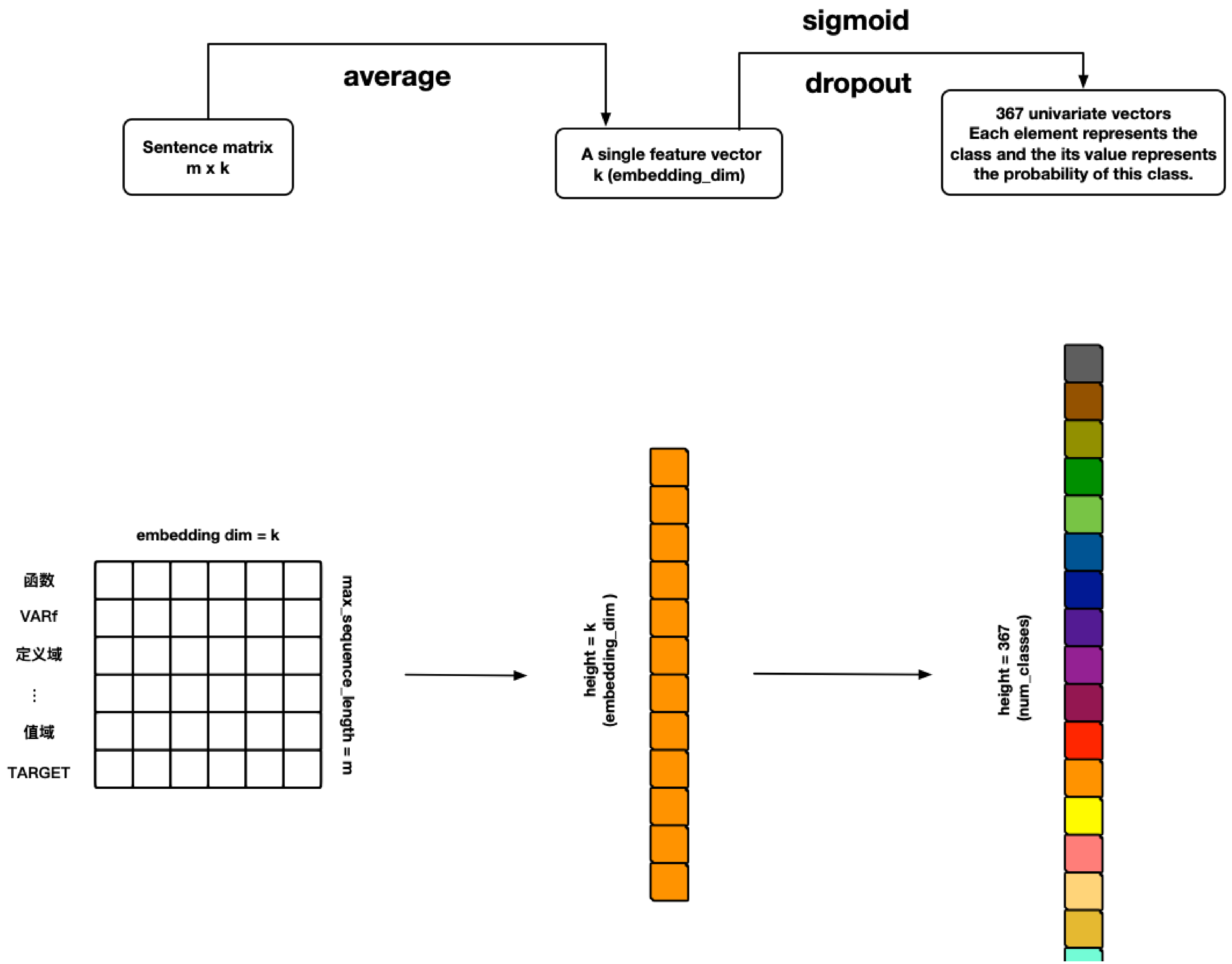
- Use `gensim` package to pre-train data.
- Use `glove` tools to pre-train data.
- Even can use a **fasttext** network to pre-train data.

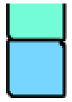
Usage

See Usage.

Network Structure

FastText

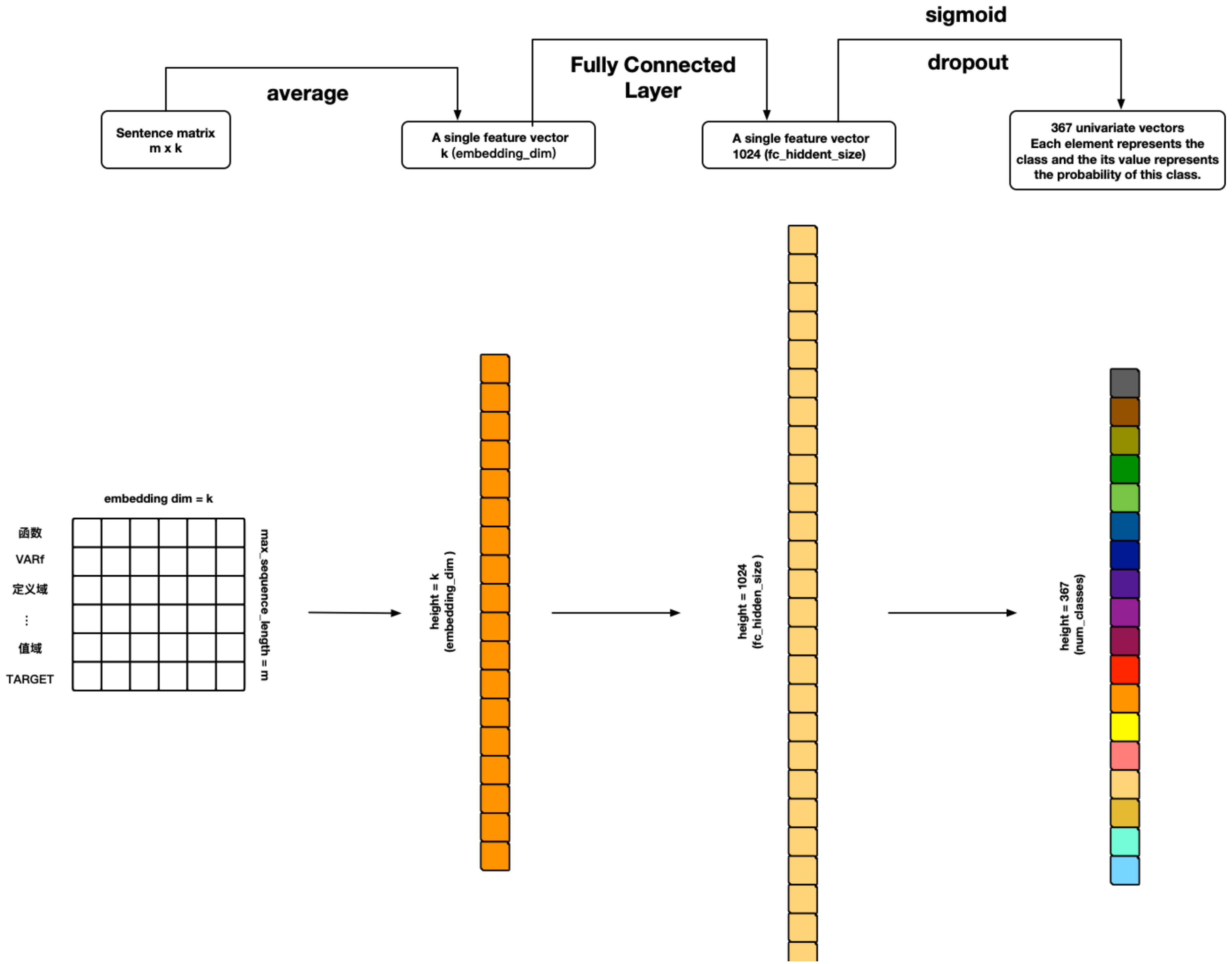

sigmoid
average
dropout
Sentence matrix m x k
A single feature vector k (embedding_dim)
367 univariate vectors Each element represents the class and the its value represents the probability of this class.
embedding dim = k
函数
VARf
定义域
⋮
值域
TARGET
max_sequence_length = m
height = k (embedding_dim)
height = 367 (num_classes)



References:

- Bag of Tricks for Efficient Text Classification

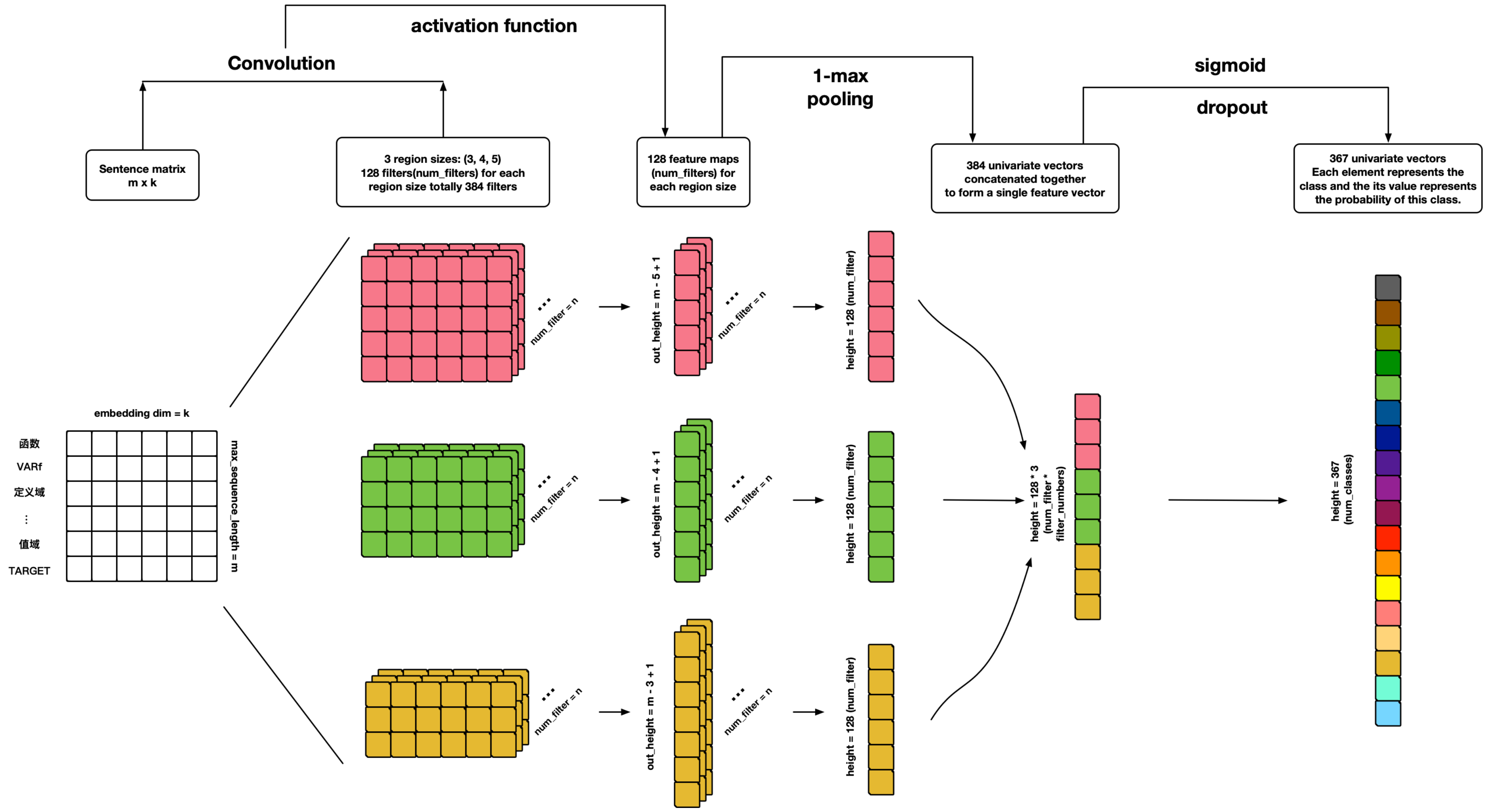
TextANN


sigmoid
dropout
Fully Connected Layer
average
Sentence matrix m x k
A single feature vector k (embedding_dim)
A single feature vector 1024 (fc_hiddent_size)
367 univariate vectors Each element represents the class and the its value represents the probability of this class.
embedding dim = k
函数
VARf
定义域
⋮
值域
TARGET
max_sequence_length = m
height = k (embedding_dim)
height = 1024 (fc_hidden_size)
height = 367 (num_classes)

References:

- Personal ideas 🙃

TextCNN

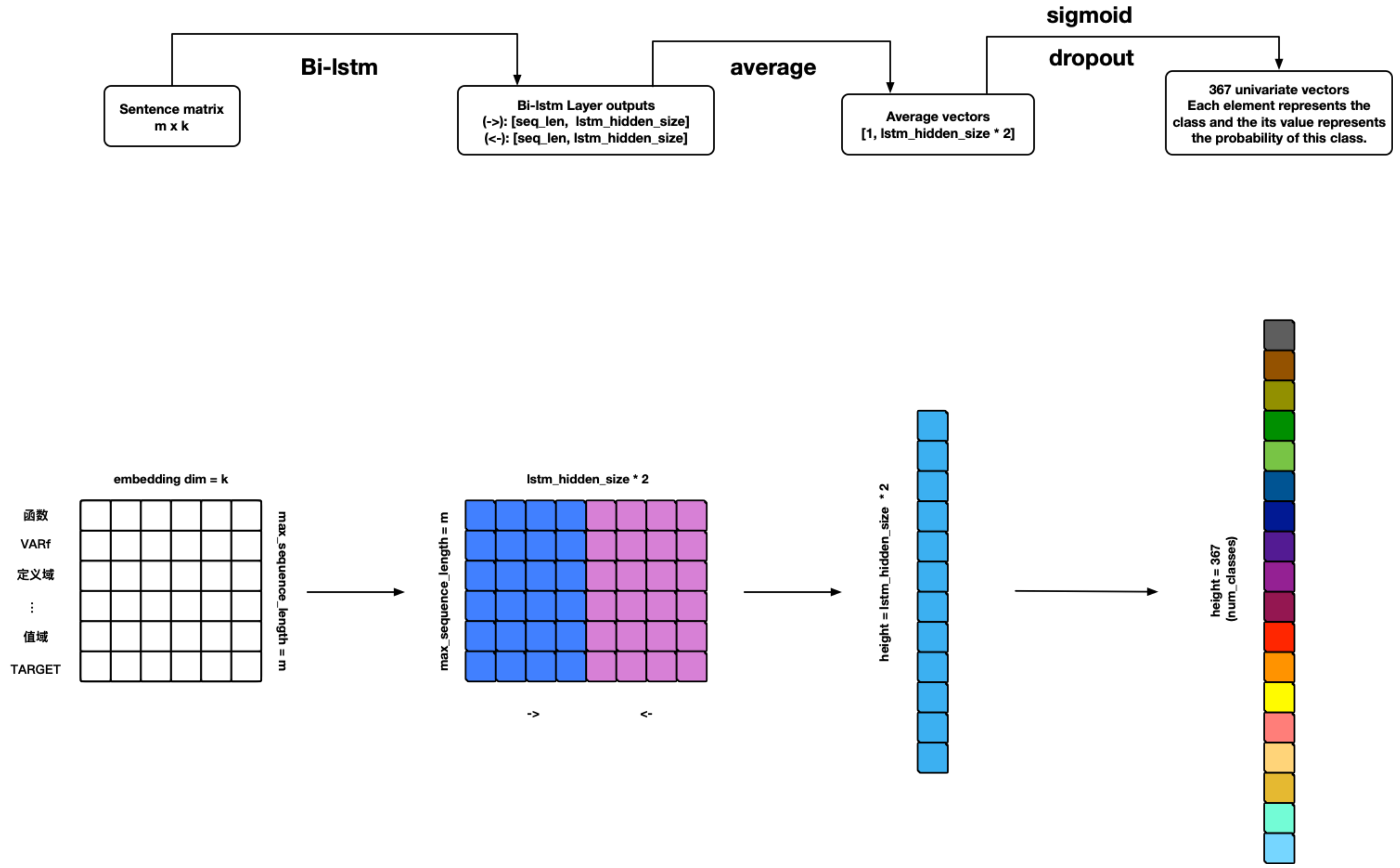

activation function
Convolution
1-max pooling
sigmoid
dropout
Sentence matrix m x k
3 region sizes: (3, 4, 5) 128 filters(num_filters) for each region size totally 384 filters
128 feature maps (num_filters) for each region size
384 univariate vectors concatenated together to form a single feature vector
367 univariate vectors Each element represents the class and the its value represents the probability of this class.
embedding dim = k
函数
VARf
定义域
⋮
值域
TARGET
max_sequence_length = m
num_filter = n
out_height = m - 5 + 1
out_height = m - 4 + 1
out_height = m - 3 + 1
height = 128 (num_filter)
height = 128 * 3 (num_filter * filter_numbers)
height = 367 (num_classes)

References:

- Convolutional Neural Networks for Sentence Classification
- A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

TextRNN

Warning: Model can use but not finished yet 🤪!



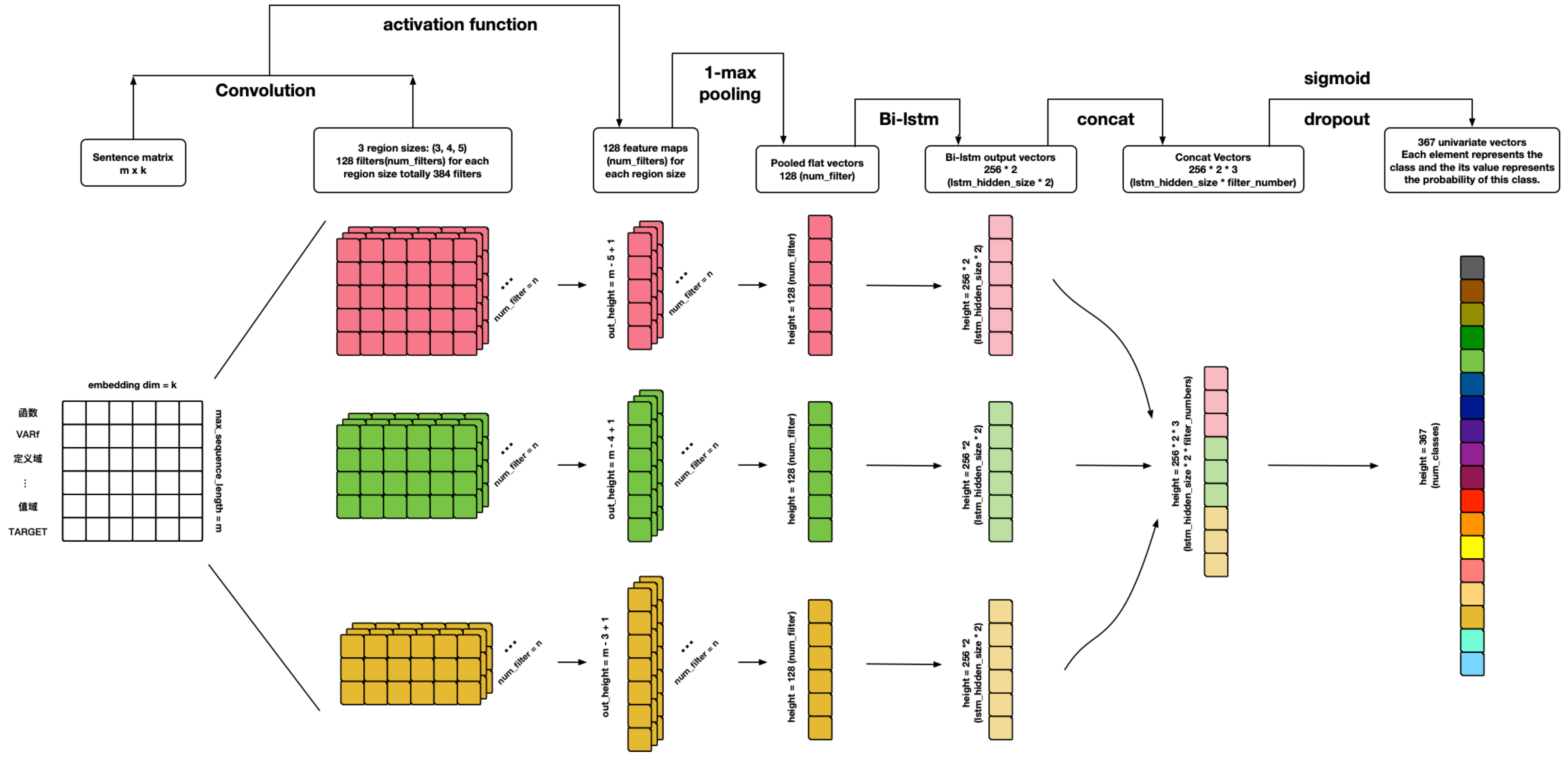
TODO

1. Add BN-LSTM cell unit.
2. Add attention.

References:

- Recurrent Neural Network for Text Classification with Multi-Task Learning

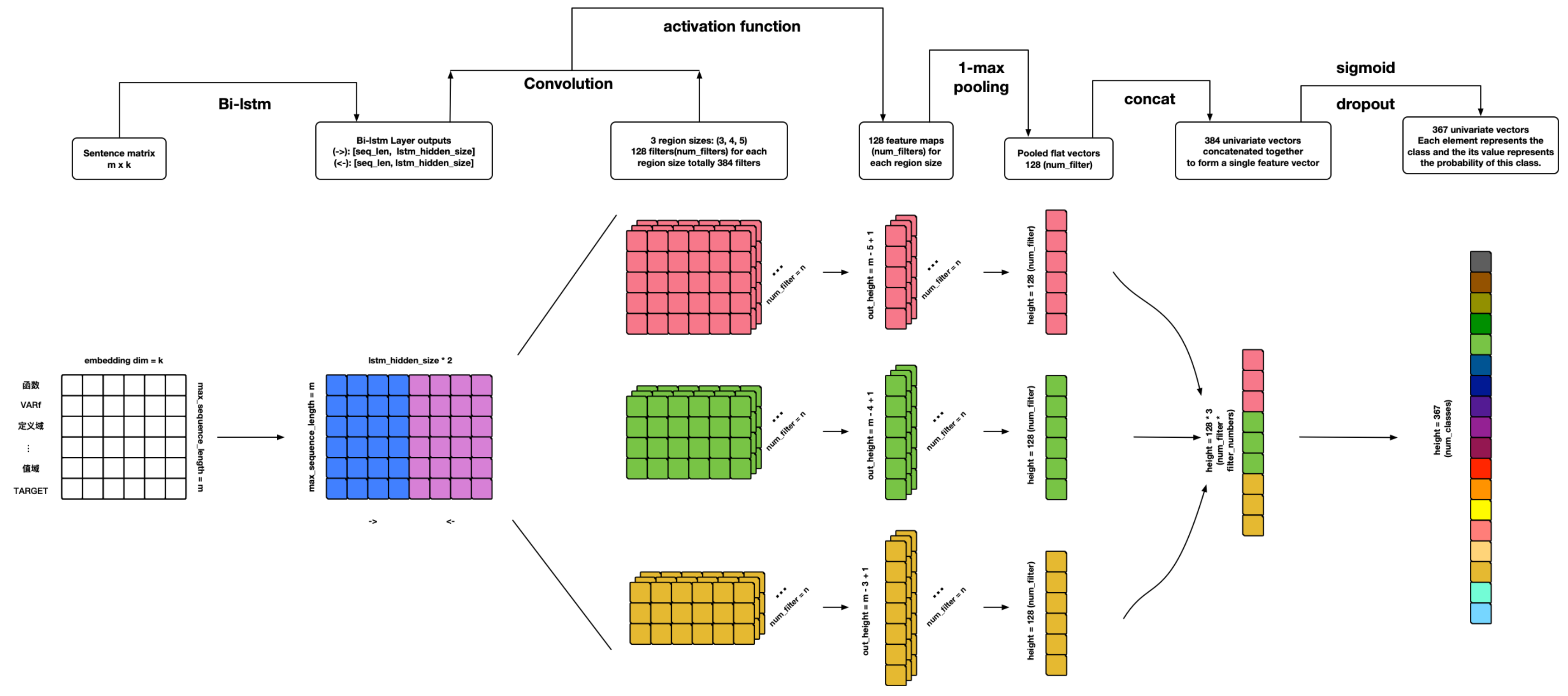
TextCRNN


activation function
Convolution
1-max pooling
Bi-lstm
concat
sigmoid
dropout
Sentence matrix m x k
3 region sizes: (3, 4, 5) 128 filters(num_filters) for each region size totally 384 filters
128 feature maps (num_filters) for each region size
Pooled flat vectors 128 (num_filter)
Bi-lstm output vectors 256 * 2 (lstm_hidden_size * 2)
Concat Vectors 256 * 2 * 3 (lstm_hidden_size * filter_number)
367 univariate vectors Each element represents the class and the its value represents the probability of this class.
embedding dim = k
函数
VARf
定义域
⋮
值域
TARGET
max_sequence_length = m
num_filter = n
out_height = m - 5 + 1
out_height = m - 4 + 1
out_height = m - 3 + 1
height = 128 (num_filter)
height = 256 * 2 (lstm_hidden_size * 2)
height = 256 * 2 * 3 (lstm_hidden_size * 2 * filter_numbers)
height = 367 (num_classes)

References:

- Personal ideas 🙃

TextRCNN


activation function
Bi-lstm
Convolution
1-max pooling
concat
sigmoid
dropout
Sentence matrix m x k
Bi-lstm Layer outputs (->): [seq_len, lstm_hidden_size] (<-): [seq_len, lstm_hidden_size]
3 region sizes: (3, 4, 5) 128 filters(num_filters) for each region size totally 384 filters
128 feature maps (num_filters) for each region size
Pooled flat vectors 128 (num_filter)
384 univariate vectors concatenated together to form a single feature vector
367 univariate vectors Each element represents the class and the its value represents the probability of this class.
embedding dim = k
函数
VARf
定义域
⋮
值域
TARGET
max_sequence_length = m
lstm_hidden_size * 2
max_sequence_length = m
->
<-
num_filter = n
out_height = m - 5 + 1
out_height = m - 4 + 1
out_height = m - 3 + 1
height = 128 (num_filter)
height = 128 * 3 (num_filter * filter_numbers)
height = 367 (num_classes)

References:

- Personal ideas 🙃

TextHAN

References:

- Hierarchical Attention Networks for Document Classification

TextSANN

Warning: Model can use but not finished yet 🤪!

TODO

1. Add attention penalization loss.
2. Add visualization.

References:

- A STRUCTURED SELF-ATTENTIVE SENTENCE EMBEDDING

About Me

黄威，Randolph

SCU SE Bachelor; USTC CS Master

Email: chinawolfman@hotmail.com

My Blog: randolph.pro

LinkedIn: randolph's linkedin