CNN for Sentence Classification

11747 HW1 report

Andrew Id: xuelei

Github link: https://github.com/Xueelei/NN4NLP

Introduction

This is the implementation of CNN for sentence classification on Pytorch. The accuracy for validation dataset reaches **83.3%**. Development environment is shown as follows.

Name	Version
Python	3.5
Pytorch	1.0.1
Gensim	3.4.0

Specification:

- pretrain_embeddings.py: Transform pre-trained Glove to Word2vec, and create embeddings for this dataset.
- main.py: Define all parameters, and define main routine for calling all the functions.
- data_loader.py: Load train, validation and test dataset, create padded sentences, create mapping between index and words.
- model.py: Build a classification model use CNN.
- util.py: Define the training process and testing process, the prediction is printed as a text file.
- params.pkl: The trained parameters.

Usage:

```
python main.py
```

Output:

The prediction for validation dataset and test dataset is stored in "/result/xx_pred.txt"

Model & Dataset

Pre-trained Word vectors:

Initializing word vectors with a publicly available Glove vectors that were trained on common crawl, it contains 400K vocabs. The vectors have 300 dimensions, words not present in the set of pre-trained words are initialized to all zeros.

Before implementation the training, Glove embeddings are transformed to Word2vec.

Model Structure

Static CNN

To decrease computational cost, the CNN model is a static one. All vectors are pre-trained and kept static, only the other parameters of the model are learned.

Input

- batch_size x max_sentence_length

The sentence is already padded.

Embeddings

- batch_size x max_sentence_length x embedding_dimension

Let x_i be the k-dimensional word vector corresponding to the i-th word in the sentence. The embedding can be represented as:

$X_{i:n} = x_1 \bigoplus x_2 \bigoplus \ldots \bigoplus x_n$

where $\bigoplus$ is the concatenation operator

Convolution

Apply 3 windows (2, 3, 4) to the same embedding matrix, get 3 convolved matrix, and concatenate them.

For each of the window size (denote as h), a filter w is applied to produce a new feature. For example, a feature c_i is generated from a window of words $x_{i:i+h-1}$ by:

$$c_i = f(w \cdot x_{i:i+h-1} + b)$$

This filter is applied to each possible window of words in the sentence$\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$ to produce a feature map

$$c = [c_1, c_2, \ldots, c_{n-h+1}]$$

After convolution

- batch_size x filters x length

After apply 3 windows and concatenate them, we can get a concatenated feature map:

$$c = \begin{bmatrix} c_{11} & c_{12} & c_{13} \ldots \\ c_{21} & c_{22} & c_{23} \ldots \\ c_{31} & c_{32} & c_{33} \ldots \end{bmatrix}$$

Max pooling

- batch x length

Then apply a max pooling operation over the feature map and take the maximum value $\hat{c} = maxc$ as the feature corresponding to this particular filter. This idea can capture the highest value for each feature map.

Output

- batch_size x number_of_tags

Finally, use linear projection to get the corresponding tags for this sentence.

Result

The key parameters are set as follows:

parameters	value
window size	2, 3, 4
batch size	32
embedding dimension	300
dropout	0.3
filters	100

Accuracy for validation dataset is: 83.3%

Analysis

Training time reduction

1. Originally, the training time is about 120 min/epoch.
2. Then I tried to load data with batch size = 128 for speeding up. (15 min/epoch)
3. Instead of using randomly assigned word vectors, I used pre-trained vectors, and keep these vectors static, only the other parameters of the model are learned. (4-5 min/epoch)

Learning rate

It's often the case that after several epochs, there's overfitting and the validation accuracy doesn't increase anymore. In order to solve this automatically, I tried to compare validation accuracy with previous one, and if it still doesn't increase for 2 epochs, then divide the learning rate by two. The initially learning rate is 0.001.

Dropout

Initially, there's no dropout in my model, and the validation accuracy reaches highest within the first several epochs, and after which it decreases. So I add dropout in this model and change the value from 0.1 to 0.4. It seems that as dropout increases, validation accuracy can keep increasing for more epochs, and thus reach higher accuracy. As 0.3 is a safe one, I finally set dropout = 0.3.

Cite:

- Convolutional Neural Networks for Sentence Classification (Y.Kim, EMNLP 2014)
- [Dataset][http://www.phontron.com/class/nn4nlp2019/assignments.html]
- Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.