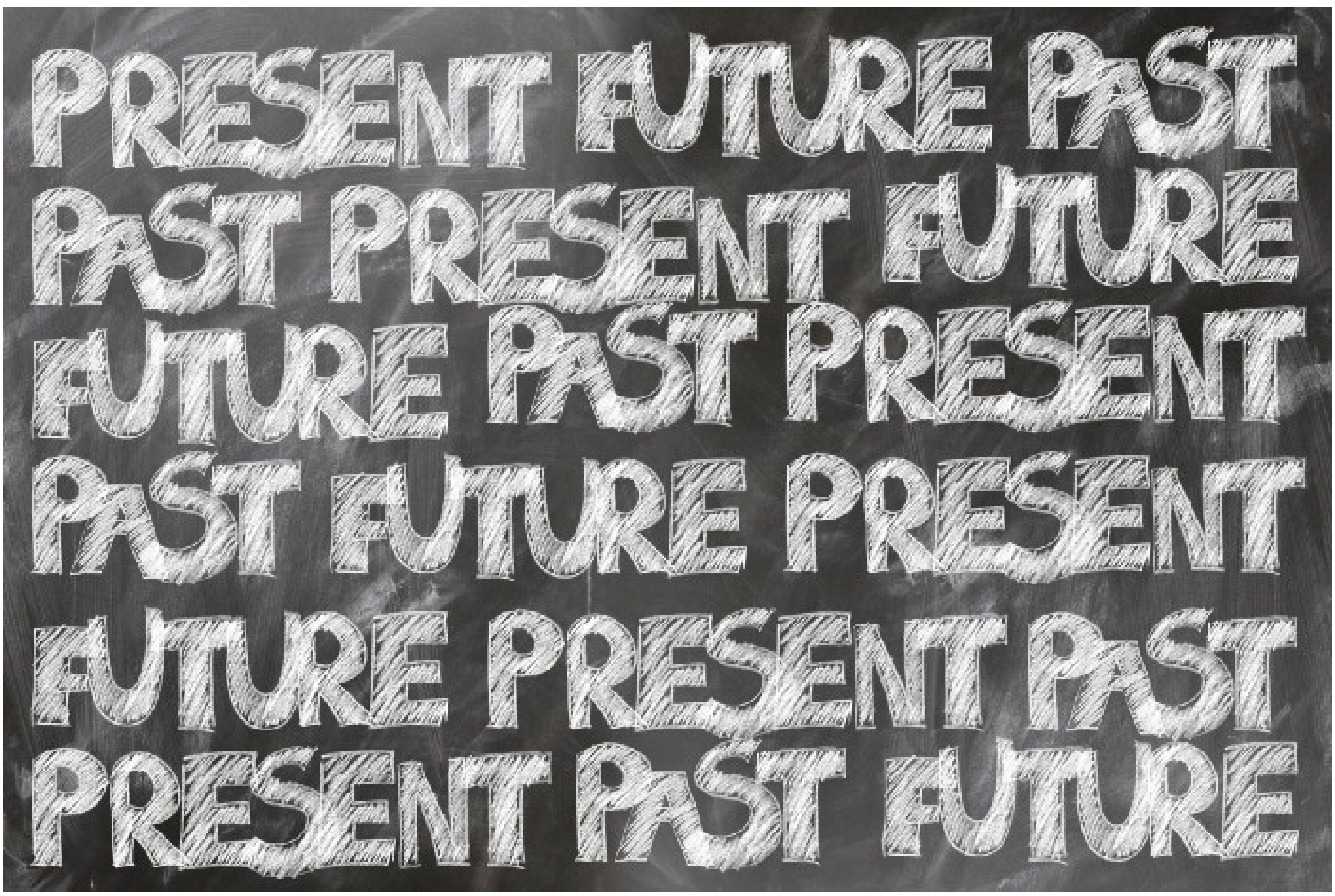
Character level CNN with Keras

Xu LIANG Follow
Jul 8, 2018 · 4 min read ★



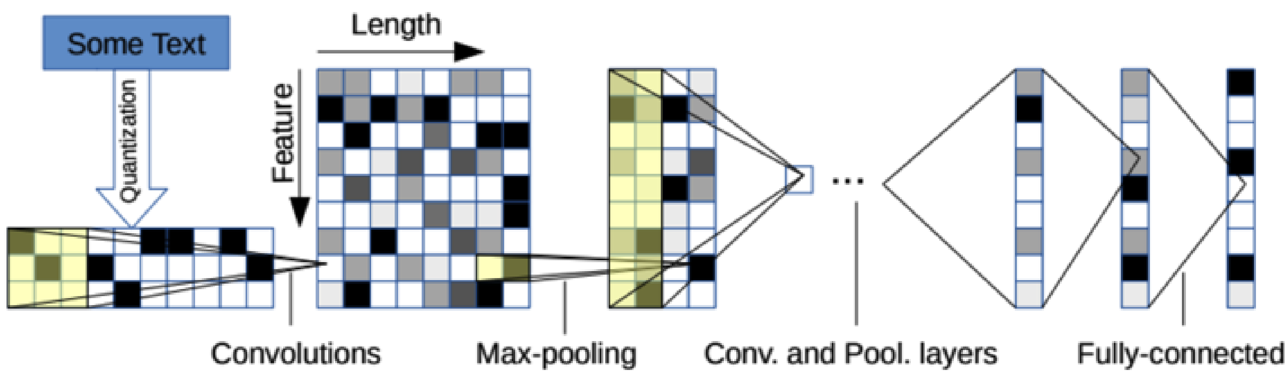
In this notebook, we will build a character level CNN model with Keras. You can find the model detail in this paper: Character-level Convolutional Networks for Text Classification.

The rest of the article is organized as follows.

- Model Introduction
- Why this model?
- Preprocessing
- Load Embedding Weights
- Model Construction
- Training

Model Introduction

The model structure:



This graph may look difficult to understand. Here is the model setup.

Table 1. Convolutional layers used in our experiments. The convolutional layers do not use stride and pooling layers are all non-overlapping ones, so we omit the description of their strides.

Layer	Large Frame	Small Frame	Kernel	Pool
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	N/A
4	1024	256	3	N/A
5	1024	256	3	N/A
6	1024	256	3	3

Table 2. Fully-connected layers used in our experiments. The number of output units for the last layer is determined by the prob-

lem. For example, for a 10-class classification problem it will be 10.

Layer	Output Units Large	Output Units Small
7	2048	1024
8	2048	1024
9	Depends on the problem	

If you want to see the detail for this model, please move to this notebook

We choose the small frame, 256 filters in convolutional layer and 1024 output units in dense layer.

- Embedding Layer
- Six convolutional layers, and 3 convolutional layers followed by a max pooling layer
- Two fully connected layer(dense layer in keras), neuron units are 1024.
- Output layer(dense layer), neuron units depends on classes. In this task, we set it 4.

Why this model?

After Kim proposed Convolutional Neural Networks for Sentence Classification, we knew CNN can have a good performance for the NLP tasks. I also implement this model, if you have some interests, you can find detail here: cnn-text-classification. But in this model, it takes sentence features from the word level, which will cause the **out-of-vocabulary** (**OOV**) problem.

In order to deal with the OOV problem, there are lots of approaches have been proposed. This character level CNN model is one of them. As the title implies that this model treat sentences in a character level. By this way, it can decrease the unknown words to a great extent so the CNN can extract mode feature to improve the text classification performance.

Preprocessing

Here just for simplicity, I write all preprocess code together. If you are interested in what happened in the preprocessing step, please move to here: How to preprocess character level text with Keras

```
# =======================Load data=========================
import numpy as np
import pandas as pd
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

from keras.layers import Input, Embedding, Activation, Flatten, Dense
from keras.layers import Conv1D, MaxPooling1D, Dropout
from keras.models import Model

train_data_source = './data/ag_news_csv/train.csv'
test_data_source = './data/ag_news_csv/test.csv'

train_df = pd.read_csv(train_data_source, header=None)
test_df = pd.read_csv(test_data_source, header=None)

# concatenate column 1 and column 2 as one text
for df in [train_df, test_df]:
    df[1] = df[1] + df[2]
    df = df.drop([2], axis=1)

# convert string to lower case
train_texts = train_df[1].values
train_texts = [s.lower() for s in train_texts]

test_texts = test_df[1].values
test_texts = [s.lower() for s in test_texts]

# ======================Convert string to index================
# Tokenizer
tk = Tokenizer(num_words=None, char_level=True, oov_token='UNK')
tk.fit_on_texts(train_texts)
# If we already have a character list, then replace the tk.word_index
# If not, just skip below part

# -----------------------Skip part start--------------------------
# construct a new vocabulary
alphabet = "abcdefghijklmnopqrstuvwxyz0123456789,;.!?:'\"/\\|_@#$%^&*~`+-=<>()[]{}"
char_dict = {}
for i, char in enumerate(alphabet):
    char_dict[char] = i + 1

```

```python
# Use char_dict to replace the tk.word_index
tk.word_index = char_dict.copy()
# Add 'UNK' to the vocabulary
tk.word_index[tk.oov_token] = max(char_dict.values()) + 1
# -----------------------Skip part end---------------------------

# Convert string to index
train_sequences = tk.texts_to_sequences(train_texts)
test_texts = tk.texts_to_sequences(test_texts)

# Padding
train_data = pad_sequences(train_sequences, maxlen=1014, padding='post')
test_data = pad_sequences(test_texts, maxlen=1014, padding='post')

# Convert to numpy array
train_data = np.array(train_data, dtype='float32')
test_data = np.array(test_data, dtype='float32')

# ======================Get classes================
train_classes = train_df[0].values
train_class_list = [x - 1 for x in train_classes]

test_classes = test_df[0].values
test_class_list = [x - 1 for x in test_classes]

from keras.utils import to_categorical

train_classes = to_categorical(train_class_list)
test_classes = to_categorical(test_class_list)
```

block1.py hosted with ♡ by GitHub view raw

Load Embedding Weights

In order to understand how to assign embedding weights to the embedding layer, here we initialize the embedding weights manually instead of initializing it randomly.

First, we have to confirm how many words in our vocabulary.

```
In [3]: print(tk.word_index)

{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 1
4, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26, '0':
27, '1': 28, '2': 29, '3': 30, '4': 31, '5': 32, '6': 33, '7': 34, '8': 35, '9': 36, ',': 37, ';': 38, '.': 39, '!':
40, '?': 41, ':': 42, "'": 43, '"': 44, '/': 45, '\\': 46, '|': 47, '_': 48, '@': 49, '#': 50, '$': 51, '%': 52,
'^': 53, '&': 54, '*': 55, '~': 56, '`': 57, '+': 58, '-': 59, '=': 60, '<': 61, '>': 62, '(': 63, ')': 64, '[': 65,
']': 66, '{': 67, '}': 68, 'UNK': 69}
```

```
In [4]: vocab_size = len(tk.word_index)
        vocab_size
Out[4]: 69
```

We can see, besides the 68 character, we also have a `UNK` (unknown token) to represent the rare characters in vocabulary.

Then we use the one-hot vector to represent these 69 words, which means each character has 69 dimensions. Because Keras use 0 for PAD, we add a zero vector to represent PAD.

```
In [5]: embedding_weights = [] #(70, 69)
        embedding_weights.append(np.zeros(vocab_size)) # zero vector to represent the PAD

        for char, i in tk.word_index.items(): # from index 1 to 69
            onehot = np.zeros(vocab_size)
            onehot[i-1] = 1
            embedding_weights.append(onehot)
        embedding_weights = np.array(embedding_weights)

In [6]: print(embedding_weights.shape) # first row all 0 for PAD, 68 char, last row for UNK
        embedding_weights

        (70, 69)

Out[6]: array([[0., 0., 0., ..., 0., 0., 0.],
               [1., 0., 0., ..., 0., 0., 0.],
               [0., 1., 0., ..., 0., 0., 0.],
               ...,
               [0., 0., 0., ..., 1., 0., 0.],
               [0., 0., 0., ..., 0., 1., 0.],
               [0., 0., 0., ..., 0., 0., 1.]])
```

Right now, the sentence is represented by the index. For example, `I love NLP` is represent as `[9, 12, 15, 22, 5, 14, 12, 16]`. First index `9` is corresponding to the `embedding_weights[9]`, which is the vector of character `I`.

After we get this embedding weights, we should pass it to initialize the embedding layer.

```
In [9]: # Embedding layer Initialization
        embedding_layer = Embedding(vocab_size+1,
                                    embedding_size,
                                    input_length=input_size,
                                    weights=[embedding_weights])
```

Model Construction

First, we give out the parameter setup.

```
In [8]: # parameter
        input_size = 1014
        # vocab_size = 69
        embedding_size = 69
        conv_layers = [[256, 7, 3],
                       [256, 7, 3],
                       [256, 3, -1],
                       [256, 3, -1],
                       [256, 3, -1],
                       [256, 3, 3]]

        fully_connected_layers = [1024, 1024]
        num_of_classes = 4
        dropout_p = 0.5
        optimizer = 'adam'
        loss = 'categorical_crossentropy'
```

Then we construction the model as the setup said.

```
In [9]: # Embedding layer Initialization
        embedding_layer = Embedding(vocab_size+1,
                                    embedding_size,
                                    input_length=input_size,
                                    weights=[embedding_weights])
```

```
In [19]: # Model Defination

         # Input
         inputs = Input(shape=(input_size,), name='input', dtype='int64')  # shape=(?, 1014)
         # Embedding
         x = embedding_layer(inputs)
         # Conv
         for filter_num, filter_size, pooling_size in conv_layers:
             x = Conv1D(filter_num, filter_size)(x)
             x = Activation('relu')(x)
             if pooling_size != -1:
                 x = MaxPooling1D(pool_size=pooling_size)(x) # Final shape=(None, 34, 256)
         x = Flatten()(x) # (None, 8704)
         # Fully connected layers
         for dense_size in fully_connected_layers:
             x = Dense(dense_size, activation='relu')(x) # dense_size == 1024
             x = Dropout(dropout_p)(x)
         # Output Layer
         predictions = Dense(num_of_classes, activation='softmax')(x)
         # Build model
         model = Model(inputs=inputs, outputs=predictions)
         model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy']) # Adam, categorical_crossentropy
         model.summary()
```

The output of `model.summary()`

```
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input (InputLayer)           (None, 1014)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1014, 69)          4830
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 1008, 256)         123904
_________________________________________________________________
activation_13 (Activation)   (None, 1008, 256)         0
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 336, 256)          0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 330, 256)          459008
_________________________________________________________________
activation_14 (Activation)   (None, 330, 256)          0
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 110, 256)          0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 108, 256)          196864
_________________________________________________________________
activation_15 (Activation)   (None, 108, 256)          0
_________________________________________________________________
conv1d_16 (Conv1D)           (None, 106, 256)          196864
_________________________________________________________________
activation_16 (Activation)   (None, 106, 256)          0
_________________________________________________________________
conv1d_17 (Conv1D)           (None, 104, 256)          196864
_________________________________________________________________
activation_17 (Activation)   (None, 104, 256)          0
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 102, 256)          196864
_________________________________________________________________
activation_18 (Activation)   (None, 102, 256)          0
_________________________________________________________________
max_pooling1d_9 (MaxPooling1 (None, 34, 256)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 8704)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              8913920
_________________________________________________________________
dropout_5 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_8 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_6 (Dropout)          (None, 1024)              0
```

```
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 4100
=================================================================
Total params: 11,342,818
Trainable params: 11,342,818
Non-trainable params: 0
_________________________________________________________________
```

block2 hosted with ♡ by GitHub view raw

Training

Our goal is to learn how to construct the model, so here I just use CPU to run the model, and only use 1000 samples for training and 100 samples for testing. The model is easy to overfit due to the small dataset.

In [11]:
```
# 1000 training samples and 100 testing samples
indices = np.arange(train_data.shape[0])
np.random.shuffle(indices)

x_train = train_data[indices][:1000]
y_train = train_classes[indices][:1000]

x_test = test_data[:100]
y_test = test_classes[:100]
```

In [13]:
```
# Training
model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          batch_size=128,
          epochs=10,
          verbose=2)
```

```
Train on 1000 samples, validate on 100 samples
Epoch 1/10
 - 72s - loss: 1.4544 - val_loss: 1.3411
Epoch 2/10
 - 68s - loss: 1.3877 - val_loss: 1.3666
Epoch 3/10
 - 61s - loss: 1.3798 - val_loss: 1.3100
```

Summarize all code together.

```python
# =====================Char CNN=======================
# parameter
input_size = 1014
vocab_size = len(tk.word_index)
embedding_size = 69
conv_layers = [[256, 7, 3],
               [256, 7, 3],
               [256, 3, -1],
               [256, 3, -1],
               [256, 3, -1],
               [256, 3, 3]]

fully_connected_layers = [1024, 1024]
num_of_classes = 4
dropout_p = 0.5
optimizer = 'adam'
loss = 'categorical_crossentropy'

# Embedding weights
embedding_weights = []  # (70, 69)
embedding_weights.append(np.zeros(vocab_size))  # (0, 69)

for char, i in tk.word_index.items():  # from index 1 to 69
    onehot = np.zeros(vocab_size)
    onehot[i - 1] = 1
    embedding_weights.append(onehot)

embedding_weights = np.array(embedding_weights)
print('Load')

# Embedding layer Initialization
embedding_layer = Embedding(vocab_size + 1,
                            embedding_size,
                            input_length=input_size,
                            weights=[embedding_weights])

# Model Construction
# Input
inputs = Input(shape=(input_size,), name='input', dtype='int64')  # shape=(?, 1014)
# Embedding
x = embedding_layer(inputs)
# Conv
```

```
for filter_num, filter_size, pooling_size in conv_layers:
    x = Conv1D(filter_num, filter_size)(x)
    x = Activation('relu')(x)
    if pooling_size != -1:
        x = MaxPooling1D(pool_size=pooling_size)(x)  # Final shape=(None, 34, 256)
x = Flatten()(x)  # (None, 8704)
# Fully connected layers
for dense_size in fully_connected_layers:
    x = Dense(dense_size, activation='relu')(x)  # dense_size == 1024
    x = Dropout(dropout_p)(x)
# Output Layer
predictions = Dense(num_of_classes, activation='softmax')(x)
# Build model
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])  # Adam, categorical_crossen
model.summary()

# Shuffle
indices = np.arange(train_data.shape[0])
np.random.shuffle(indices)

x_train = train_data[indices]
y_train = train_classes[indices]

x_test = test_data
y_test = test_classes

# Training
model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          batch_size=128,
          epochs=10,
          verbose=2)
```

block3.py hosted with ♡ by GitHub view raw

The notebook of this article is here, and the whole script is here. Preprocess article is here

I create a repository to contains my work while learning the NLP as a beginner. If you find it useful, please star the project. I am glad to hear feedback or advice.
nlp-beginner-guide-keras

> **Check out my other posts on Medium with a categorized view!**
> **GitHub: BrambleXu**
> **LinkedIn: Xu Liang**
> **Blog: BrambleXu**

Keras Machine Learning Data Science Naturallanguageprocessing Python