How to Develop a Multichannel CNN Model for Text Classification

by **Jason Brownlee** on January 12, 2018 in **Deep Learning for Natural Language Processing**

Last Updated on December 4, 2019

A standard deep learning model for text classification and sentiment analysis uses a word embedding layer and one-dimensional convolutional neural network.

The model can be expanded by using multiple parallel convolutional neural networks that read the source document using different kernel sizes. This, in effect, creates a multichannel convolutional neural network for text that reads text with different n-gram sizes (groups of words).

In this tutorial, you will discover how to develop a multichannel convolutional neural network for sentiment prediction on text movie review data.

After completing this tutorial, you will know:

- How to prepare movie review text data for modeling.
- How to develop a multichannel convolutional neural network for text in Keras.
- How to evaluate a fit model on unseen movie review data.

Discover how to develop deep learning models for text classification, translation, photo captioning and more in my new book, with 30 step-by-step tutorials and full source code.

Let’s get started.

- **Update Feb/2018**: Small code change to reflect changes in Keras 2.1.3 API.

How to Develop an N-gram Multichannel Convolutional Neural Network for Sentiment Analysis
Photo by Ed Dunens, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. Movie Review Dataset
2. Data Preparation
3. Develop Multichannel Model
4. Evaluate Model

Python Environment

This tutorial assumes you have a Python 3 SciPy environment installed.

You must have Keras (2.0 or higher) installed with either the TensorFlow or Theano backend.

The tutorial also assumes you have scikit-learn, Pandas, NumPy, and Matplotlib installed.

If you need help with your environment, see this post:

- How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda

Movie Review Dataset

The Movie Review Data is a collection of movie reviews retrieved from the imdb.com website in the early 2000s by Bo Pang and Lillian Lee. The reviews were collected and made available as part of their research on natural language processing.

The reviews were originally released in 2002, but an updated and cleaned up version was released in 2004, referred to as “v2.0”.

The dataset is comprised of 1,000 positive and 1,000 negative movie reviews drawn from an archive of the rec.arts.movies.reviews newsgroup hosted at imdb.com. The authors refer to this dataset as the “polarity dataset.”

> *Our data contains 1000 positive and 1000 negative reviews all written before 2002, with a cap of 20 reviews per author (312 authors total) per category. We refer to this corpus as the polarity dataset.*

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

The data has been cleaned up somewhat; for example:

- The dataset is comprised of only English reviews.
- All text has been converted to lowercase.
- There is white space around punctuation like periods, commas, and brackets.

- Text has been split into one sentence per line.

The data has been used for a few related natural language processing tasks. For classification, the performance of machine learning models (such as Support Vector Machines) on the data is in the range of high 70% to low 80% (e.g. 78%-82%).

More sophisticated data preparation may see results as high as 86% with 10-fold cross-validation. This gives us a ballpark of low-to-mid 80s if we were looking to use this dataset in experiments of modern methods.

> *… depending on choice of downstream polarity classifier, we can achieve highly statistically significant improvement (from 82.8% to 86.4%)*

— A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, 2004.

You can download the dataset from here:

- Movie Review Polarity Dataset (review_polarity.tar.gz, 3MB)

After unzipping the file, you will have a directory ca
containing the text “*neg*” and “*pos*” for negative ar le
with a naming convention *cv000* to *cv999* for each neg and pos.

Start Machine Learning

Next, let’s look at loading and preparing the text data.

Data Preparation

In this section, we will look at 3 things:

1. Separation of data into training and test sets.
2. Loading and cleaning the data to remove punctuation and numbers.
3. Prepare all reviews and save to file.

Split into Train and Test Sets

We are pretending that we are developing a system that can predict the sentiment of a textual movie review as either positive or negative.

This means that after the model is developed, we will need to make predictions on new textual reviews.

This will require all of the same data preparation to be performed on those new reviews as is performed on the training data for the model.

We will ensure that this constraint is built into the evaluation of our models by splitting the training and test datasets prior to any data preparation. This means that any knowledge in the data in the test set that could help us better prepare the data (e.g. the words used) is unavailable in the preparation of data used for training the model.

That being said, we will use the last 100 positive reviews and the last 100 negative reviews as a test set (100 reviews) and the remaining 1,800 reviews as the training dataset.

This is a 90% train, 10% split of the data.

The split can be imposed easily by using the filenames of the reviews where reviews named 000 to 899 are for training data and reviews named 900 onwards are for test.

Loading and Cleaning Reviews

The text data is already pretty clean; not much preparation is required.

Without getting bogged down too much by the details, we will prepare the data in the following way:

- Split tokens on white space.
- Remove all punctuation from words.
- Remove all words that are not purely comprised of alphabetical characters.
- Remove all words that are known stop words.
- Remove all words that have a length <= 1 character.

We can put all of these steps into a function called *clean_doc()* that takes as an argument the raw text loaded from a file and returns a list of cleaned tokens. We can also define a function *load_doc()* that loads a document from file ready for use with the *clean_doc()* function. An example of cleaning the first positive review is listed below.

```
from nltk.corpus import stopwords
import string

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# turn a doc into clean tokens
def clean_doc(doc):
    # split into tokens by white space
    tokens = doc.split()
    # remove punctuation from each token
    table = str.maketrans('', '', string.punctuation)
    tokens = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 1]
    return tokens

# load the document
filename = 'txt_sentoken/pos/cv000_29590.txt'
text = load_doc(filename)
tokens = clean_doc(text)
print(tokens)
```

Running the example loads and cleans one movie review.

The tokens from the clean review are printed for review.

```
...
'creepy', 'place', 'even', 'acting', 'hell', 'solid', 'dreamy', 'depp', 'turning', 'typically
```

Clean All Reviews and Save

We can now use the function to clean reviews and apply it to all reviews.

To do this, we will develop a new function named *process_docs()* below that will walk through all reviews in a directory, clean them, and return them as a list.

We will also add an argument to the function to indicate whether the function is processing train or test reviews, that way the filenames can be filtered (as described above) and only those train or test reviews requested will be cleaned and returned.

The full function is listed below.

```
# load all docs in a directory
def process_docs(directory, is_trian):
    documents = list()
    # walk through all files in the folder
    for filename in listdir(directory):
        # skip any reviews in the test set
        if is_trian and filename.startswith('cv9'):
            continue
        if not is_trian and not filename.startswith('cv9'):
            continue
        # create the full path of the file to open
        path = directory + '/' + filename
        # load the doc
        doc = load_doc(path)
        # clean doc
        tokens = clean_doc(doc)
        # add to list
        documents.append(tokens)
    return documents
```

We can call this function with negative training reviews as follows:

```
negative_docs = process_docs('txt_sentoken/neg', True)
```

Next, we need labels for the train and test documents. We know that we have 900 training documents and 100 test documents. We can use a Python list comprehension to create the labels for the negative (0) and positive (1) reviews for both train and test sets.

```
trainy = [0 for _ in range(900)] + [1 for _ in range(900)]
testY = [0 for _ in range(100)] + [1 for _ in range(100)]
```

Finally, we want to save the prepared train and test sets to file so that we can load them later for modeling and model evaluation.

The function below-named *save_dataset()* will save a given prepared dataset (X and y elements) to a file using the pickle API.

```
# save a dataset to file
def save_dataset(dataset, filename):
    dump(dataset, open(filename, 'wb'))
    print('Saved: %s' % filename)
```

Complete Example

We can tie all of these data preparation steps together.

The complete example is listed below.

```
from string import punctuation
from os import listdir
from nltk.corpus import stopwords
from pickle import dump

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# turn a doc into clean tokens
def clean_doc(doc):
    # split into tokens by white space
    tokens = doc.split()
    # remove punctuation from each token
    table = str.maketrans('', '', punctuation)
    tokens = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 1]
    tokens = ' '.join(tokens)
    return tokens

# load all docs in a directory
def process_docs(directory, is_trian):
    documents = list()
    # walk through all files in the folder
    for filename in listdir(directory):
        # skip any reviews in the test set
        if is_trian and filename.startswith('cv9'):
            continue
        if not is_trian and not filename.startswith('cv9'):
            continue
        # create the full path of the file to open
        path = directory + '/' + filename
        # load the doc
        doc = load_doc(path)
        # clean doc
        tokens = clean_doc(doc)
        # add to list
        documents.append(tokens)
    return documents

# save a dataset to file
def save_dataset(dataset, filename):
    dump(dataset, open(filename, 'wb'))
    print('Saved: %s' % filename)

# load all training reviews
negative_docs = process_docs('txt_sentoken/neg', True)
positive_docs = process_docs('txt_sentoken/pos', True)
trainX = negative_docs + positive_docs
trainy = [0 for _ in range(900)] + [1 for _ in range(900)]
save_dataset([trainX,trainy], 'train.pkl')

# load all test reviews
```

```
negative_docs = process_docs('txt_sentoken/neg', False)
positive_docs = process_docs('txt_sentoken/pos', False)
testX = negative_docs + positive_docs
testY = [0 for _ in range(100)] + [1 for _ in range(100)]
save_dataset([testX,testY], 'test.pkl')
```

Running the example cleans the text movie review documents, creates labels, and saves the prepared data for both train and test datasets in *train.pkl* and *test.pkl* respectively.

Now we are ready to develop our model.

Develop Multichannel Model

In this section, we will develop a multichannel convolutional neural network for the sentiment analysis prediction problem.

This section is divided into 3 parts:

1. Encode Data
2. Define Model.
3. Complete Example.

Encode Data

The first step is to load the cleaned training dataset.

The function below-named *load_dataset()* can be called to load the pickled training dataset.

```
# load a clean dataset
def load_dataset(filename):
    return load(open(filename, 'rb'))

trainLines, trainLabels = load_dataset('train.pkl')
```

Next, we must fit a Keras Tokenizer on the training dataset. We will use this tokenizer to both define the vocabulary for the Embedding layer and encode the review documents as integers.

The function *create_tokenizer()* below will create a Tokenizer given a list of documents.

```
# fit a tokenizer
def create_tokenizer(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
```

We also need to know the maximum length of input sequences as input for the model and to pad all sequences to the fixed length.

The function *max_length()* below will calculate the maximum length (number of words) for all reviews in the training dataset.

```
# calculate the maximum document length
def max_length(lines):
    return max([len(s.split()) for s in lines])
```

We also need to know the size of the vocabulary for the Embedding layer.

This can be calculated from the prepared Tokenizer, as follows:

```
# calculate vocabulary size
vocab_size = len(tokenizer.word_index) + 1
```

Finally, we can integer encode and pad the clean movie review text.

The function below named *encode_text()* will both encode and pad text data to the maximum review length.

```
# encode a list of lines
def encode_text(tokenizer, lines, length):
    # integer encode
    encoded = tokenizer.texts_to_sequences(lines)
    # pad encoded sequences
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded
```

Define Model

A standard model for document classification is to use an Embedding layer as input, followed by a one-dimensional convolutional neural network, pooling layer, and then a prediction output layer.

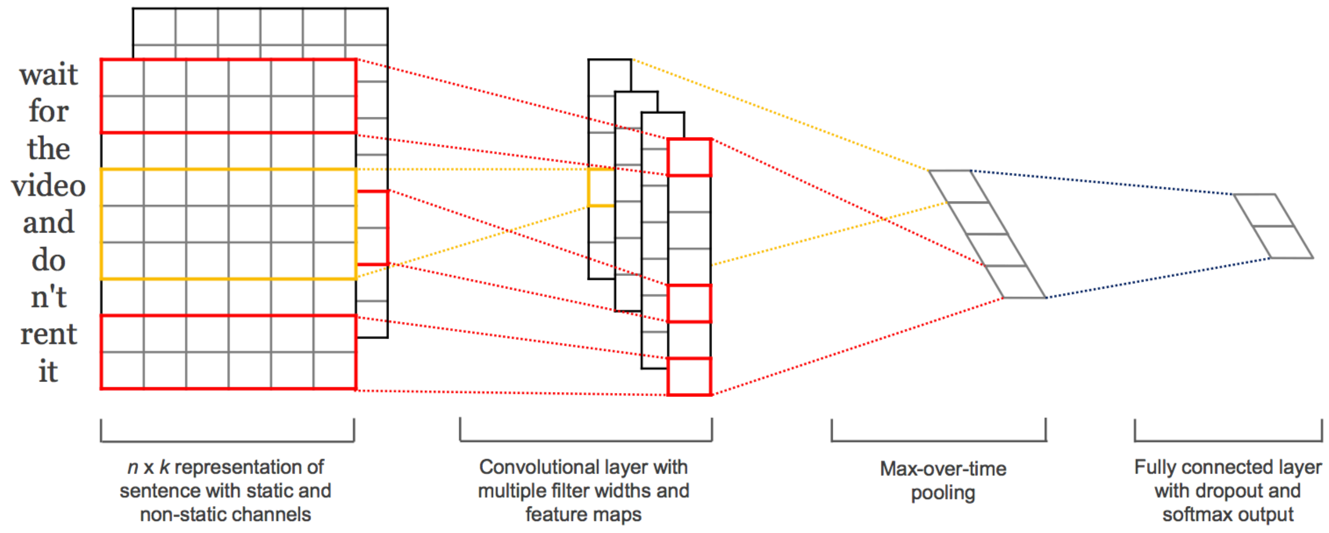
The kernel size in the convolutional layer defines the number of words to consider as the convolution is passed across the input text document, providing a grouping parameter.

A multi-channel convolutional neural network for document classification involves using multiple versions of the standard model with different sized kernels. This allows the document to be processed at different resolutions or different n-grams (groups of words) at a time, whilst the model learns how to best integrate these interpretations.

This approach was first described by Yoon Kim in his 2014 paper titled “Convolutional Neural Networks for Sentence Classification.”

In the paper, Kim experimented with static and dynamic (updated) embedding layers, we can simplify the approach and instead focus only on the use of different kernel sizes.

This approach is best understood with a diagram taken from Kim's paper:


wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Depiction of the multiple-channel convolutional neural network for text.
Taken from "Convolutional Neural Networks for Sentence Classification."

In Keras, a multiple-input model can be defined using the functional API.

We will define a model with three input channels for processing 4-grams, 6-grams, and 8-grams of movie review text.

Each channel is comprised of the following elements:

- Input layer that defines the length of input sequences.
- Embedding layer set to the size of the vocabulary and 100-dimensional real-valued representations.
- One-dimensional convolutional layer with 32 filters and a kernel size set to the number of words to read at once.
- Max Pooling layer to consolidate the output from the convolutional layer.
- Flatten layer to reduce the three-dimensional output to two dimensional for concatenation.

The output from the three channels are concatenated into a single vector and process by a Dense layer and an output layer.

The function below defines and returns the model. As part of defining the model, a summary of the defined model is printed and a plot of the model graph is created and saved to file.

```python
# define the model
def define_model(length, vocab_size):
    # channel 1
    inputs1 = Input(shape=(length,))
    embedding1 = Embedding(vocab_size, 100)(inputs1)
    conv1 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    pool1 = MaxPooling1D(pool_size=2)(drop1)
    flat1 = Flatten()(pool1)
    # channel 2
    inputs2 = Input(shape=(length,))
    embedding2 = Embedding(vocab_size, 100)(inputs2)
    conv2 = Conv1D(filters=32, kernel_size=6, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    pool2 = MaxPooling1D(pool_size=2)(drop2)
    flat2 = Flatten()(pool2)
    # channel 3
    inputs3 = Input(shape=(length,))
    embedding3 = Embedding(vocab_size, 100)(inputs3)
    conv3 = Conv1D(filters=32, kernel_size=8, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    pool3 = MaxPooling1D(pool_size=2)(drop3)
    flat3 = Flatten()(pool3)
    # merge
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(10, activation='relu')(merged)
    outputs = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
    # compile
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize
    print(model.summary())
    plot_model(model, show_shapes=True, to_file='multichannel.png')
    return model
```

Complete Example

Pulling all of this together, the complete example is listed below.

```python
from pickle import load
from numpy import array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.vis_utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers import Embedding
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.layers.merge import concatenate

# load a clean dataset
def load_dataset(filename):
    return load(open(filename, 'rb'))

# fit a tokenizer
def create_tokenizer(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# calculate the maximum document length
def max_length(lines):
    return max([len(s.split()) for s in lines])

# encode a list of lines
def encode_text(tokenizer, lines, length):
    # integer encode
    encoded = tokenizer.texts_to_sequences(lines)
    # pad encoded sequences
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded

# define the model
def define_model(length, vocab_size):
    # channel 1
    inputs1 = Input(shape=(length,))
    embedding1 = Embedding(vocab_size, 100)(inputs1)
    conv1 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    pool1 = MaxPooling1D(pool_size=2)(drop1)
    flat1 = Flatten()(pool1)
    # channel 2
    inputs2 = Input(shape=(length,))
    embedding2 = Embedding(vocab_size, 100)(inputs2)
    conv2 = Conv1D(filters=32, kernel_size=6, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    pool2 = MaxPooling1D(pool_size=2)(drop2)
    flat2 = Flatten()(pool2)
    # channel 3
    inputs3 = Input(shape=(length,))
    embedding3 = Embedding(vocab_size, 100)(inputs3)
    conv3 = Conv1D(filters=32, kernel_size=8, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    pool3 = MaxPooling1D(pool_size=2)(drop3)
    flat3 = Flatten()(pool3)
    # merge
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(10, activation='relu')(merged)
    outputs = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
    # compile
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    # summarize
    print(model.summary())
    plot_model(model, show_shapes=True, to_file='multichannel.png')
    return model

# load training dataset
trainLines, trainLabels = load_dataset('train.pkl')
# create tokenizer
tokenizer = create_tokenizer(trainLines)
# calculate max document length
length = max_length(trainLines)
# calculate vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print('Max document length: %d' % length)
print('Vocabulary size: %d' % vocab_size)
# encode data
trainX = encode_text(tokenizer, trainLines, length)
print(trainX.shape)

# define model
model = define_model(length, vocab_size)
# fit model
model.fit([trainX,trainX,trainX], array(trainLabels), epochs=10, batch_size=16)
# save the model
model.save('model.h5')
```

Running the example first prints a summary of the prepared training dataset.

```
Max document length: 1380
Vocabulary size: 44277
(1800, 1380)
```

Next, a summary of the defined model is printed.

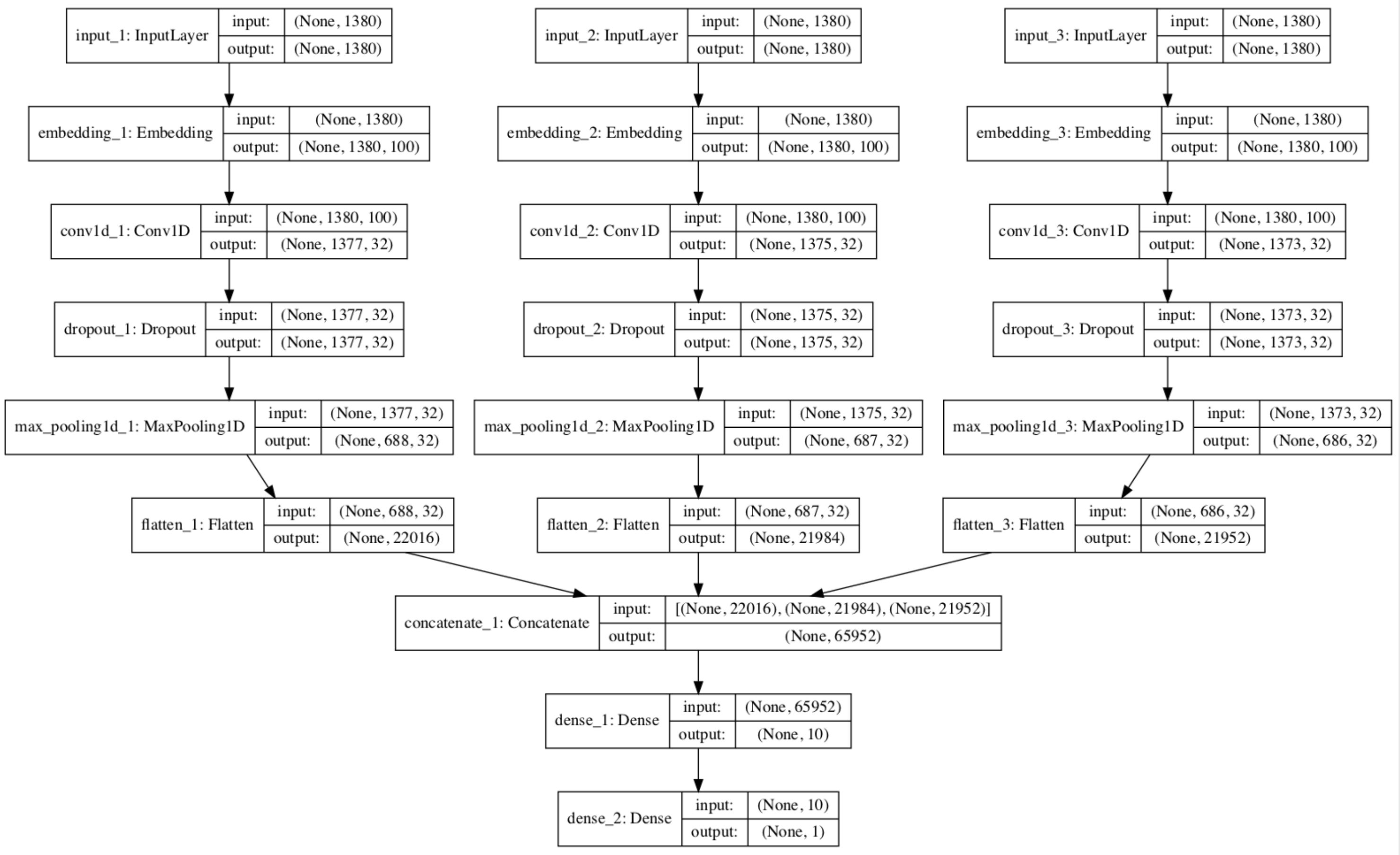
```
____________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
============================================================================================
input_1 (InputLayer)             (None, 1380)          0
____________________________________________________________________________________________
input_2 (InputLayer)             (None, 1380)          0
____________________________________________________________________________________________
input_3 (InputLayer)             (None, 1380)          0
____________________________________________________________________________________________
embedding_1 (Embedding)          (None, 1380, 100)     4427700     input_1[0][0]
____________________________________________________________________________________________
embedding_2 (Embedding)          (None, 1380, 100)     4427700     input_2[0][0]
____________________________________________________________________________________________
embedding_3 (Embedding)          (None, 1380, 100)     4427700     input_3[0][0]
____________________________________________________________________________________________
conv1d_1 (Conv1D)                (None, 1377, 32)      12832       embedding_1[0][0]
____________________________________________________________________________________________
conv1d_2 (Conv1D)                (None, 1375, 32)      19232       embedding_2[0][0]
____________________________________________________________________________________________
conv1d_3 (Conv1D)                (None, 1373, 32)      25632       embedding_3[0][0]
____________________________________________________________________________________________
dropout_1 (Dropout)              (None, 1377, 32)      0           conv1d_1[0][0]
```

```
____________________________________________________________________________________________________
dropout_2 (Dropout)              (None, 1375, 32)      0            conv1d_2[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 1373, 32)      0            conv1d_3[0][0]
____________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)   (None, 688, 32)       0            dropout_1[0][0]
____________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)   (None, 687, 32)       0            dropout_2[0][0]
____________________________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)   (None, 686, 32)       0            dropout_3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 22016)         0            max_pooling1d_1[0][0]
____________________________________________________________________________________________________
flatten_2 (Flatten)              (None, 21984)         0            max_pooling1d_2[0][0]
____________________________________________________________________________________________________
flatten_3 (Flatten)              (None, 21952)         0            max_pooling1d_3[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)      (None, 65952)         0            flatten_1[0][0]
                                                                    flatten_2[0][0]
                                                                    flatten_3[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 10)            659530       concatenate_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 1)             11           dense_1[0][0]
====================================================================================================
Total params: 14,000,337
Trainable params: 14,000,337
Non-trainable params: 0
____________________________________________________________________________________________________
```

The model is fit relatively quickly and appears to show good skill on the training dataset.

```
...
Epoch 6/10
1800/1800 [==============================] - 30s - loss: 9.9093e-04 - acc: 1.0000
Epoch 7/10
1800/1800 [==============================] - 29s - loss: 5.1899e-04 - acc: 1.0000
Epoch 8/10
1800/1800 [==============================] - 28s - loss: 3.7958e-04 - acc: 1.0000
Epoch 9/10
1800/1800 [==============================] - 29s - loss: 3.0534e-04 - acc: 1.0000
Epoch 10/10
1800/1800 [==============================] - 29s - loss: 2.6234e-04 - acc: 1.0000
```

A plot of the defined model is saved to file, clearly showing the three input channels for the model.


input_1: InputLayer | input: (None, 1380) | output: (None, 1380)
input_2: InputLayer | input: (None, 1380) | output: (None, 1380)
input_3: InputLayer | input: (None, 1380) | output: (None, 1380)
embedding_1: Embedding | input: (None, 1380) | output: (None, 1380, 100)
embedding_2: Embedding | input: (None, 1380) | output: (None, 1380, 100)
embedding_3: Embedding | input: (None, 1380) | output: (None, 1380, 100)
conv1d_1: Conv1D | input: (None, 1380, 100) | output: (None, 1377, 32)
conv1d_2: Conv1D | input: (None, 1380, 100) | output: (None, 1375, 32)
conv1d_3: Conv1D | input: (None, 1380, 100) | output: (None, 1373, 32)
dropout_1: Dropout | input: (None, 1377, 32) | output: (None, 1377, 32)
dropout_2: Dropout | input: (None, 1375, 32) | output: (None, 1375, 32)
dropout_3: Dropout | input: (None, 1373, 32) | output: (None, 1373, 32)
max_pooling1d_1: MaxPooling1D | input: (None, 1377, 32) | output: (None, 688, 32)
max_pooling1d_2: MaxPooling1D | input: (None, 1375, 32) | output: (None, 687, 32)
max_pooling1d_3: MaxPooling1D | input: (None, 1373, 32) | output: (None, 686, 32)
flatten_1: Flatten | input: (None, 688, 32) | output: (None, 22016)
flatten_2: Flatten | input: (None, 687, 32) | output: (None, 21984)
flatten_3: Flatten | input: (None, 686, 32) | output: (None, 21952)
concatenate_1: Concatenate | input: [(None, 22016), (None, 21984), (None, 21952)] | output: (None, 65952)
dense_1: Dense | input: (None, 65952) | output: (None, 10)
dense_2: Dense | input: (None, 10) | output: (None, 1)

Plot of the Multichannel Convolutional Neural Network For Text

The model is fit for a number of epochs and saved to the file *model.h5* for later evaluation.

Evaluate Model

In this section, we can evaluate the fit model by predicting the sentiment on all reviews in the unseen test dataset.

Using the data loading functions developed in the previous section, we can load and encode both the training and test datasets.

```python
# load datasets
trainLines, trainLabels = load_dataset('train.pkl')
testLines, testLabels = load_dataset('test.pkl')

# create tokenizer
tokenizer = create_tokenizer(trainLines)
# calculate max document length
length = max_length(trainLines)
# calculate vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print('Max document length: %d' % length)
print('Vocabulary size: %d' % vocab_size)
# encode data
trainX = encode_text(tokenizer, trainLines, length)
testX = encode_text(tokenizer, testLines, length)
print(trainX.shape, testX.shape)
```

We can load the saved model and evaluate it on both the training and test datasets.

The complete example is listed below.

```
from pickle import load
from numpy import array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model

# load a clean dataset
def load_dataset(filename):
    return load(open(filename, 'rb'))

# fit a tokenizer
def create_tokenizer(lines):
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# calculate the maximum document length
def max_length(lines):
    return max([len(s.split()) for s in lines])

# encode a list of lines
def encode_text(tokenizer, lines, length):
    # integer encode
    encoded = tokenizer.texts_to_sequences(lines)
    # pad encoded sequences
    padded = pad_sequences(encoded, maxlen=length, padding='post')
    return padded

# load datasets
trainLines, trainLabels = load_dataset('train.pkl')
testLines, testLabels = load_dataset('test.pkl')

# create tokenizer
tokenizer = create_tokenizer(trainLines)
# calculate max document length
length = max_length(trainLines)
# calculate vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print('Max document length: %d' % length)
print('Vocabulary size: %d' % vocab_size)
# encode data
trainX = encode_text(tokenizer, trainLines, length)
testX = encode_text(tokenizer, testLines, length)
print(trainX.shape, testX.shape)

```

```
# load the model
model = load_model('model.h5')

# evaluate model on training dataset
loss, acc = model.evaluate([trainX,trainX,trainX], array(trainLabels), verbose=0)
print('Train Accuracy: %f' % (acc*100))

# evaluate model on test dataset dataset
loss, acc = model.evaluate([testX,testX,testX],array(testLabels), verbose=0)
print('Test Accuracy: %f' % (acc*100))
```

Running the example prints the skill of the model on both the training and test datasets.

```
Max document length: 1380
Vocabulary size: 44277
(1800, 1380) (200, 1380)

Train Accuracy: 100.000000
Test Accuracy: 87.500000
```

We can see that, as expected, the skill on the training dataset is excellent, here at 100% accuracy.

We can also see that the skill of the model on the unseen test dataset is also very impressive, achieving 87.5%, which is above the skill of the model reported in the 2014 paper (although not a direct apples-to-apples comparison).

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Different n-grams**. Explore the model by changing the kernel size (number of n-grams) used by the channels in the model to see how it impacts model skill.
- **More or Fewer Channels**. Explore using more or fewer channels in the model and see how it impacts model skill.
- **Deeper Network**. Convolutional neural networks perform better in computer vision when they are deeper. Explore using deeper models here and see how it impacts model skill.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Convolutional Neural Networks for Sentence Classification, 2014.
- Convolutional Neural Networks for Sentence Classification (code).
- Keras Functional API

Summary

In this tutorial, you discovered how to develop a multichannel convolutional neural network for sentiment prediction on text movie review data.

Specifically, you learned:

- How to prepare movie review text data for modeling.
- How to develop a multichannel convolutional neural network for text in Keras.
- How to evaluate a fit model on unseen movie review data.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

Tweet Share Share

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

< How to Develop a Neural Machine Translation System from Scratch

How to Generate Test Datasets in Python with scikit-learn >

169 Responses to *How to Develop a Multichannel CNN Model for Text Classification*

Hitkul January 12, 2018 at 5:54 pm # REPLY

Hi
Great article
I think there is a small error, According to this code trainLines would be a list of list, where each list holds tokens for one review. But all the functions to which trainLines is passed(i.e texts_to_sequences, fit_on_texts, max_length) takes a list of strings as input. I think all the lists in trainX and testX should be converted to strings before dumping them into a file.

Jason Brownlee January 13, 2018 at 5:30 am # REPLY

The list of tokens is turned back into a string:

```
tokens = ' '.join(tokens)
```

Minel March 5, 2019 at 9:27 pm # REPLY

Hello Jason
yes but this instruction is missing in your code
I got a traceback with this message list object has no attribute lower

Si I add one line in clean_doc function

tokens = [word for word in tokens if len(word) > 1]
tokens = ' '.join(tokens)
return tokens
It works well
best

Marko Plahuta January 12, 2018 at 6:41 pm # REPLY

Hi,
thanks for the article. Would it be possible to use only one input and one embedding layer, and branch into convolutions after that?

Jason Brownlee January 13, 2018 at 5:31 am # REPLY

How would that work?

Francesco January 25, 2018 at 9:13 pm # REPLY

Wouldn't the model below be exactly the same? (with just one input, used in the three channels?) Of course, if we also had a single embedding (put the embedding before the channels, and let all convs act on that one) the model would be different.

```
def define_model(length, vocab_size):
    inputs = Input(shape=(length,))
    # channel 1
    embedding1 = Embedding(vocab_size, 100)(inputs)
    conv1 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding1)
    drop1 = Dropout(0.5)(conv1)
    pool1 = MaxPooling1D(pool_size=2)(drop1)
    flat1 = Flatten()(pool1)
    # channel 2
    embedding2 = Embedding(vocab_size, 100)(inputs)
    conv2 = Conv1D(filters=32, kernel_size=6, activation='relu')(embedding2)
    drop2 = Dropout(0.5)(conv2)
    pool2 = MaxPooling1D(pool_size=2)(drop2)
    flat2 = Flatten()(pool2)
    # channel 3
    embedding3 = Embedding(vocab_size, 100)(inputs)
    conv3 = Conv1D(filters=32, kernel_size=8, activation='relu')(embedding3)
    drop3 = Dropout(0.5)(conv3)
    pool3 = MaxPooling1D(pool_size=2)(drop3)
    flat3 = Flatten()(pool3)
    # merge
    merged = concatenate([flat1, flat2, flat3])
    # interpretation
    dense1 = Dense(10, activation='relu')(merged)
    outputs = Dense(1, activation='sigmoid')(dense1)
    model = Model(inputs=[inputs], outputs=outputs)
    # compile
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accur
    # summarize
    print(model.summary())
    plot_model(model, show_shapes=True, to_file='multichannel.png')
    return model
```

Jason Brownlee January 26, 2018 at 5:40 am # REPLY

Yes, nice approach.

How does it compare regarding training/prediction?

HSA February 5, 2020 at 9:41 am #

given that I run the code using different dataset and embedding, Mr.Jason code gave me:

ModelMulitCNN MODEL Accuracy: 0.8785238339313173
ModelMulitCNN MODEL precision_score: 0.7633378932968536
ModelMulitCNN MODEL recall_score: 0.6495925494761351
ModelMulitCNN MODEL f1_score: 0.7018867924528303

while Mr.Francesco code gave me:

ModelMulitCNN MODEL Accuracy: 0.8782675550999487
ModelMulitCNN MODEL precision_score: 0.7424242424242424
ModelMulitCNN MODEL recall_score: 0.6845168800931315
ModelMulitCNN MODEL f1_score: 0.7122955784373107

Jason Brownlee February 5, 2020 at 1:41 pm #

Nice, thanks for sharing.

Adnan ÖNCEVARLIK January 13, 2018 at 12:31 am # REPLY

Hi,
Thank you for your effort and good clean article.

Jason Brownlee January 13, 2018 at 5:33 am # REPLY

You're welcome.

Sébastien January 14, 2018 at 3:18 am # REPLY

FYI, the link to the review polarity dataset is wrong. The correct one is:
https://www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz

Jason Brownlee January 14, 2018 at 6:39 am # REPLY

Fixed, thanks.

Ahmet January 14, 2018 at 3:23 am # REPLY

Hi Jason,
Thanks for your this beautiful work. I want to ask you that we are able to evaluate the accuracy of the model but how can we predict the classes of tested documents to analyze results in detail. In a Sequential model we can perform it like model.predict_classes(x_test). However for the Model(inputs=…) object predict_classes feature is not supported. Do you have a suggestion?
Thanks.

Jason Brownlee January 14, 2018 at 6:40 am # REPLY

Yes, the document must be provided in an array 3 times.

yhat = model.predict([doc, doc, doc])

ahmet January 14, 2018 at 7:53 am # REPLY

Yes I also tried like that but I am not sure how to interpret the probabilities for multiclass classification.

Jason Brownlee January 15, 2018 at 6:54 am # REPLY

What is the problem exactly?

Ahmet January 15, 2018 at 7:29 pm #

Thanks for interest, Jason.
I am actually trying to apply your post for a multiclass case (0:negative,1:neutral,2:positive). After training part I want to compare the accuracy rates for each class to measure the how model is accurate in detail. Then I want to calculate the Precision and Recall. However, I can not use predict_classes() function in Model() object it is just allowed for Sequential() object. When I prefer the fuction model.predict([test_doc, test_doc, test_doc]) it gives me some probabilities below but I am not sure how to map them to class labels (0,1,2).

```
[0.03045881]
[0.39043367]
[0.01636862]
…
[0.7408592 ]
[0.17758404]
[0.21271853]
```

Jason Brownlee January 16, 2018 at 7:32 am #

You can use argmax on the result from predict() to get a class index.

rango July 4, 2019 at 2:05 pm # REPLY

hi Jason,
I also try your post for text classification.However when I try to apply argmax() function like you said I get an array of 0.

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,0, 0])

Do you Have any suggestion? Thank in advance.

P.S
Before predict class I prepare text sample like this:

sample = "This is sample text for classification"
tokenizer.fit_on_texts(sample)
sequences = tokenizer.texts_to_sequences(sample)
test_data = pad_sequences(sequences, maxlen=maxlen)

predict = model.predict([test_data,test_data,test_data])
predict.argmax(axis=-1)

If I do something wrong plz correct me! Thank You

Jason Brownlee July 4, 2019 at 2:50 pm # REPLY

Perhaps this will help:
https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

Rui January 14, 2018 at 4:33 am # REPLY

I have tried similar architectures and came to the conclusion that this type of architecture (with parallel paths) are not good because when the error is back propagated one of the good paths ,that would be learning in the right way, will be affected by a bad path that will increase tha global error. So one good path would be in the right way but it will think it is a bad learning because the global error will increase . (is this ideia right ?)

whe I have "parallel layers" I have to find a way to pretrain before … and the freeze the values and add this layers to the model.

Jason Brownlee January 14, 2018 at 6:41 am # REPLY

Perhaps it depends on the dataset and models involved.

Rui January 14, 2018 at 6:59 am # REPLY

There are some types of backprop that work with different learning rates for each connection and with this mitigate better the error on each connection … maybe this can help on "concurrent paths" during learning . here is one type of bprop like this :
https://en.wikipedia.org/wiki/Rprop

Jason Brownlee January 15, 2018 at 6:53 am # REPLY

Yes, I recommend Adam:
https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

JP Raymond January 14, 2018 at 12:34 pm # REPLY

The architecture described in Yoon Kim's paper:
– has one embedding layer (not one for each branch),
– uses global (max-over-time) pooling (not with a pool of size 2),
– applies dropout once, on the concatenation of the max features (not in each branch before the pooling operation).

Do the changes proposed here yield a better predictive performance?

Jason Brownlee January 15, 2018 at 6:56 am # REPLY

Nice!

No idea, why not compare?

Dev January 18, 2018 at 3:10 am # REPLY

Hi Jason,

Thanks for this article. Very helpful.

I am facing an issue with the output, like getting different results everytime when I run the same model. The loss and accuracy are also changing all the time.

Any thoughts on this ?

Jason Brownlee January 18, 2018 at 10:13 am # REPLY

Machine learning algorithms are stochastic by design. This is a feature, not a bug. Learn more here:
https://machinelearningmastery.com/randomness-in-machine-learning/

Ramzy January 21, 2018 at 7:40 am # REPLY

Hi Jason,

Thanks for the great tutorial, I performed all the steps and in the last step when I tried to evaluate the model, I got the below results!

```
Train Accuracy: 50.000000
Test Accuracy: 50.000000
```
What does this mean! I think this is not logical.

Also I got some warning like the below while running the evaluation code.

```
2018-01-20 22:21:41.333198: W c:\tf_jenkins\home\workspace\release-win\m\windows
\py\36\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library
wasn’t compiled to use AVX2 instructions, but these are available on your machin
e and could speed up CPU computations.
2018-01-20 22:21:41.333808: W c:\tf_jenkins\home\workspace\release-win\m\windows
\py\36\tensorflow\core\platform\cpu_feature_guard.cc:45] The TensorFlow library
wasn’t compiled to use FMA instructions, but these are available on your machine
and could speed up CPU computations.
```

I’ll appreciate your reply. Thanks in advance

Jason Brownlee January 21, 2018 at 9:16 am # REPLY

Ignore those warnings.

Perhaps try running the code a second time given the stochastic nature of the algorithm.

Ramzy January 24, 2018 at 8:20 pm # REPLY

Sorry sir, I didn't get a notification that you replied to me.
Could you please tell me what you meant by running the code given the stochastic nature of the algorithm. Thanks in advance.

Jason Brownlee January 25, 2018 at 5:53 am # REPLY

Yes, see this post:
https://machinelearningmastery.com/randomness-in-machine-learning/

Mahfuza January 24, 2018 at 11:40 am # REPLY

Hi, First of all thank you so much for the article.

I tried to utilize the multichannel concept in a different way. I am working with relation extraction in natural language where relations are marked already but needs to figure out whether or not there is a valid sentence structure in between those entities. I am thinking to get different representations of a same sentence. For an example sentence (Paris is the capital of France), the real words of the sentence (Paris, is, the, capital, of, France) as first channel's data, the POS tags of those same tokens (propn, verb, det, noun, adp, propn) as second channel's data and, dependency tree tags (nsubj, root, det, attr, prep, pobj) as third channel's data.

I am confused whether I need to encode all the words, pos tags, and dep tags thinking them as a part of the same vocabulary? Or I need to encode those different tokens/representations in their own vocabulary scope?

Regards

Jason Brownlee January 24, 2018 at 4:39 pm # REPLY

Interesting approach.

I guess you could try a few different representations and test whether there is any impact on model skill?

Mahfuza January 25, 2018 at 5:51 am # REPLY

Actually I am confused on the text to integer encoding step. I mean, should I consider all the different representations of the same sentence as a single list of documents? For example lets say I got three different representations of the sentence "Paris is the capital of France" and collected them in a single document list like below –

all_docs = [[Paris is the capital of France]…, [propn verb det noun adp propn]…, [nsubj root det attr prep pobj]…]

Now if I fit the tokenizer like tokenizer.fit_on_texts(all_docs) into these documents then all the tokens are having different unique integers, right? Is it then valuable if I feed those three encoded representations separately into different channels like input1 = [ints of tokens], input2 = [more ints of tokens], input3 = [some ints of tokens]?

OR

I should consider fitting the tokenizer in three different representations separately and then encode them separately based on the fitted tokenizer. I think the tokenizer would provide some common integers then. Because the encoding scopes are isolated. Will that help in the concatenated layer some how?

Thanks again Jason for your reply and effort for this blog.

Jason Brownlee January 25, 2018 at 5:59 am # REPLY

Perhaps this would be a good place to start to get a handle on data prep:
https://machinelearningmastery.com/start-here/#nlp

Francesco January 25, 2018 at 9:19 pm # REPLY

Hi Jason,

would it be possible, looking at the activation patterns, to identify the n-ngrams that mostly affected the classification? With pictures it is possible, but I suppose that with text it should be very difficult, with the embedding inbetween…

Thanks
Francesco

Jason Brownlee January 26, 2018 at 5:40 am # REPLY

It may be, but some hard thinking and development would be required. It's not obvious sorry.

Massa February 1, 2018 at 7:32 am # REPLY

Hi,

Thank you for this neat article! 🙂

I am facing an issue.

AttributeError: 'int' object has no attribute 'ndim'
in model.fit

Jason Brownlee February 2, 2018 at 8:02 am # REPLY

Did you try coping all of the code from the final example?

Mia February 9, 2018 at 5:31 pm # REPLY

I also had the same issue. The reason is that variable is a list object while Keras expects a numpy array. I solved the problem by importing numpy and inserting "trainLabels = numpy.array(trainLabels)"

Mia February 9, 2018 at 5:35 pm # REPLY i mean variable trainLabels .

Jason Brownlee February 14, 2018 at 11:03 am # REPLY

I have update the example to correct this issue.

It seems to be a change in Keras 2.1.3.

JanneK February 5, 2018 at 5:44 am # REPLY

Thank for this useful post Jason. It's interesting that even such a minimalistic preprocessing always seem to work fine. When doing text classification (or regression) tasks, do you generally see any value in keeping/including additional information during preprocessing, such as:

(1) Basic sentence-separating punctuations, such as ".", "!" and "?"
(2) Paragraph separators between groups of sentences
(3) POS tag augmented tokens, e.g., "walking" –> "walking_VERB", "apple" –> "apple_NOUN"

Do these typically have any significant impact?

Jason Brownlee February 5, 2018 at 7:53 am # REPLY

It really depends on the application I'm afraid.

For simple classification like tasks, often simpler representations result in better model skill.

Jason H February 7, 2018 at 7:13 am # REPLY

You should be able to comment out inputs2, embedding2, inputs3 and embedding3, and then feed embedding1 to conv2 and conv3, right? Then you go from [x_in, x_in, x_in] to just x_in for fit(), and predict().

Franco Arda February 14, 2018 at 7:01 pm # REPLY

Hi Jason,

In your NLP book (Chapter 16), you cover n-gram CNN in depth. LSTM's memory mimic n-grams.

May I ask you why you not use a LSTM classification instead?

Thanks!
Franco

Jason Brownlee February 15, 2018 at 8:39 am # REPLY

You can use LSTMs, here I was demonstrating how to do it with CNNs.

Franco February 15, 2018 at 8:16 pm # REPLY

Thanks Jason!

Indeed, you have a NLP sample in your LSTM book.

Boris February 21, 2018 at 2:15 am # REPLY

Hi Jason,

Thanks for yet another amazing post. Apart from purely educational purposes, what do you think would be the differences between using LSTM and CNN in sentiment analysis? Are both approaches completely interchangeable or each of the models might hold advantages/limitations in different setups.

Best,
Boris

Jason Brownlee February 21, 2018 at 6:41 am #

CNN seems to achieve state of the art results. I would start there.

Satheesh March 24, 2018 at 10:25 pm # REPLY

Can we apply this approach to multi class problems? The only change is during fitting and using ‘categorical cross entropy” as loss.Am I correct

Jason Brownlee March 25, 2018 at 6:29 am # REPLY

Sure.

Marwa May 5, 2018 at 3:24 am # REPLY

Hi Jason,
I am beginner on keras and CNN ,I want to know How to give multinputs to train_test_split?

my example:(Xtrain_user,Xtrain_item,Xtest_user,Xtest_item),y_train,y_test=train_test_split((user_reviews, item_reviews),rating , test_size=0.2, random_state=42)

which X= (user_reviews,item_reviews)
and Y=rating

Jason Brownlee May 5, 2018 at 6:24 am # REPLY

Perhaps split the data manually using array notation in NumPy:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

Marwa May 5, 2018 at 7:41 am # REPLY

But I have a dictionnaire for user and for item ,how can I split it ?

Jason Brownlee May 6, 2018 at 6:18 am # REPLY

Perhaps pick one axis (e.g. users) and divide a list of users in half?

Junaid Akhtar May 11, 2018 at 3:19 pm # REPLY

First of all, thank you for what you did here, and generally what you do 🙂

I tested your code on the imdb data, got the similar results, then changed the output to 2 neurons with softmax/argmax training/testing, got marginally better results.

Then I switched to my own 5-class sentiment dataset, but got 98%-ish traning accuracy but 40%-ish testing accuracy, which means its over-fitting, but then read in many papers that generally in the 5-grained SST1 dataset everyone reports an accuracy of 40%-ish. With binary +/- sentiments of course everyone reaches late 80s-90s in accuracy.

What do you suggest to that? Or a setting I could try on top of your code.

Jason Brownlee May 12, 2018 at 6:26 am # REPLY

The model architecture may require tuning to each different problem.

Ethan Schmit May 31, 2018 at 7:53 pm # REPLY

Hi Jason ,

Thanks for the great article.

If I have more than 2 classes, e.g. positive/negative/neutral, what will my output list (trainLabels) look like? Will it be a a list containing 0's, 1's and 2's or will it be a n*3 matrix, with each column containing 0's and 1's?

Thanks a lot

Jason Brownlee June 1, 2018 at 8:18 am # REPLY

I would recommend a one hot encoding, you can learn more here:
https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

az June 28, 2018 at 4:01 am # REPLY

Hi,
i have created a network which takes two sequences of integers (2 inputs) for one sentence (one related to word embeddings and other related to POS tags) and corresponding embedding layers and then merges them both together before applying the convolution layer. I had called it multi input model.

```
inp = Input(shape=(data.shape[1],))
e= Embedding(vocab_size, 200, input_length=maxSeqLength, weights=[embedding_matrix],
trainable=False)(inp)
inp2 = Input(shape=(embedding_pos.shape[1],))
e1= Embedding(54, 54, input_length=maxSeqLength, weights=[embedding_matrix_pos],
trainable=False)(inp2)

mer=concatenate([e,e1],axis=-1)

conv1_1 = Conv1D(filters=100, kernel_size=3)(mer)
conv1_2 = Conv1D(filters=100, kernel_size=4)(mer)
```

From this article i understood that it might be multichannel. I am confused that is multiple inputs is necessarily multichannel or multiple parallel convolution layers even on same input is multichannel? If i consider the second definition i would call my network multi-input multichannel?
Also in this case all the convolutions are applied to original input.What is the difference if i stack up convolutions so the result of one is input to another?Thanks.

Jason Brownlee June 28, 2018 at 6:25 am # REPLY

If the inputs are different, it might be better to have a multi-headed CNN rather than a multichannel CNN.

---

az June 28, 2018 at 8:51 am # REPLY

The inputs are sequence of the same sentence and both are padded to same length..so would it be called different inputs still? and can you please clarify if multichannel is multi-input regardless of parallel or sequential convolutions?

Jason Brownlee June 28, 2018 at 2:07 pm # REPLY

Each head of the model will "read" the data differently.

Perhaps referring to the multiple inputs to the model as "channels" was a poor choice. Different inputs as in the above model are indeed different to different channels for one input. Importantly, we can use different sized kernels and number of filters with different inputs, whereas each channel on one input is fixed to use the same kernel and number of filters.

az July 12, 2018 at 4:45 am # REPLY

Thanks alot. I understand that now that multichannel means one input with one or more embeddings for that one input.In your example you are using input multiple times but still its the same input. I have taken your advice and applied convolution of different filter size to each embedding separately instead of merging them together which has improved accuracy. Only one thing,it means it is multi input, i can understand how it can be seen as multi headed..so basically having multiple inputs makes it multiheaded?Thanks

Jason Brownlee July 12, 2018 at 6:30 am # REPLY

Yes.

az July 28, 2018 at 4:11 am # REPLY

Thank you so much! The precision and recall graphs i plotted through training epochs show sudden spikes in precision while gradual increase in recall.Though recall is higher than precision. Now my dataset has more positive samples than negative ones which lead me to believe that there is chance higher number of FN than FP i.e. lower recall and higher precision but my results are opposite. Can you please elaborate if i have the wrong understanding?

az July 28, 2018 at 4:44 am #

or should i see it this way that since it has more positive samples,classifier is biased towards positive class,leading to more FP and hence lower precision and higher recall?Thanks

Jason Brownlee July 28, 2018 at 6:40 am #

You will need to find a trade-off that makes sense for your specific application and problem.

Yuval July 11, 2018 at 11:20 pm # REPLY

I plotted precision-recall curve for both pos and neg class and found the results interesting, While the curve for the pos class looks very good with oprtimal point at 0.9 and 0.8 for recall and precision respecitvally. The curve for the neg class is a stright line that gives 0.6 for precision and recall or 0.8 with 0.6 and vice versa.
Any idea how this could be?

Jason Brownlee July 12, 2018 at 6:25 am # REPLY

It may suggest that one class is easier to predict than the other.

Zenon Uchida July 22, 2018 at 9:39 pm # REPLY

I did the exact thing on the tutorial but i got a lower test accuracy which is 83.5%. Traning is 100%

Jason Brownlee July 23, 2018 at 6:09 am # REPLY

You could try running the example a few times to see if you get differing results.

Zenon Uchida August 5, 2018 at 2:45 am # REPLY

(forgot to reply) Yes, actually got different results after running it a few times.

andi wijaya July 28, 2018 at 2:03 am # REPLY

def max_length(lines):
return max([len(s.split()) for s in lines])
I just copy and paste the code and have this error, I believe that code is actually right, I dont know why I got an error
AttributeError: 'list' object has no attribute 'split'

Environment
Python 3.6
Tensorflow 1.9
Keras 2.2

Jason Brownlee July 28, 2018 at 6:37 am # REPLY

Ensure the indenting is correct and that you have all of the surrounding code and data.

I have more suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Zenon Uchida August 4, 2018 at 11:59 pm # REPLY

I have a question on the "Plot of the Multichannel Convolutional Neural Network For Text". Why is there always a "none" per box? What does it denote to?

Zenon Uchida August 5, 2018 at 2:02 am # REPLY

Found the answer: The None dimension in the shape tuple refers to the batch dimension which simply means that the layer can accept input of any size.

Jason Brownlee August 5, 2018 at 5:33 am # REPLY

Yes.

Jason Brownlee August 5, 2018 at 5:31 am # REPLY

Good question!

"None" refers to a dimension that is not specified, and in turn is variable.

Zenon Uchida August 5, 2018 at 2:50 am # REPLY

I have a question. Based on this paper, https://arxiv.org/pdf/1510.03820.pdf (A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification), they stated "we set the number of feature maps for this region size to 100". How do i set the number of feature maps? Is the number of feature maps based on the number of filters per region size?

Jason Brownlee August 5, 2018 at 5:35 am # REPLY

The number of feature maps is the first argument to the convolutional layer Conv1D().

Zenon Uchida August 17, 2018 at 1:39 am # REPLY

I really got lots of questions sorry xD.

What's an intuitive way of understanding how this multi-channeled CNN (or perhaps a basic CNN architecture in general) identify the following (given that the sentiment expressed are only positive or negative):
1.) If more than one sentiment is expressed in the tweet but the positive sentiment is more dominant, then it is a positive tweet, or
2.) If more than one sentiment is expressed in the tweet but the negative sentiment is more dominant, then it is a negative tweet.

Jason Brownlee August 17, 2018 at 6:34 am # REPLY

We cannot know what the CNN has learned, only the evaluation of the model performance.

Zenon Uchida August 17, 2018 at 3:40 pm # REPLY

What I'm trying to understand here is how CNN is applied to sentence classification. I still don't get the idea. What i know is CNN when it comes to computer vision, it finds features of pictures. For example a dog with features such ears, nose and eyes.

Zenon Uchida August 17, 2018 at 4:56 pm # REPLY

I think i got.
In convolutional neural networks every network layer acts as a detection filter for the presence of specific features present in the original data. The first layer in a CNN detect (large) features that can be recognized and interpreted relatively easy. Subsequent layers detect increasingly (smaller) features that are more abstract. The last layer of the CNN is able to make an ultra-specific classification by combining all the specific features detected by the previous layers in the input data.

Jason Brownlee August 18, 2018 at 5:34 am # REPLY

Perhaps try this simpler tutorial:
https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

Ishay Telavivi August 19, 2018 at 1:23 am # REPLY

Hi,

I have a question about the input of the model, based on an error I got.

I ran this code on a different data and I had no problems. Then I tried an extention with RandomizedSearchCV, as the following:

model = KerasClassifier(build_fn=create_model, verbose=1, epochs=3, batch_size=32)
param_dist= {"n_strides": sp_randint(1,3)}

random_grid = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter = 3)
random_grid_result = random_grid.fit([X_train, X_train, X_train], y_train)

I got the following error:
ValueError: Found input variables with inconsistent numbers of samples: [3, 25000]

What may the error be? 25000 is the length of my train data

Jason Brownlee August 19, 2018 at 6:26 am # REPLY

Sorry, I'm not sure what is going on.

Perhaps post your code and error to stackoverflow?

Ishay Telavivi August 19, 2018 at 7:13 am # REPLY

Sorry for not being understood.

I have explored this further and found out that Keras wrapper don't support multi input network. That was the problem (https://github.com/keras-team/keras/issues/6451)

So I decided to use Francesco's comment above (January 25), using one Input for all three channels and that it's working!

Jason Brownlee August 20, 2018 at 6:30 am # REPLY

Nice!

Ishay Telavivi August 19, 2018 at 1:33 am # REPLY

Hi,

I have a queation about the nagtion words.
When you clean your data with excluding stopwords, you are loosing the nagation words ('not' etc.). Don't you want to leave these words to have better interpretation of the sentence?

For example: "I didn't like the movie" and "I liked the movie" would look the same after cleaning.

Jason Brownlee August 19, 2018 at 6:27 am # REPLY

It really depends on whether they add value on the specific problem you are solving or not.

Perhaps try modeling with and without them?

Ishay Telavivi August 19, 2018 at 7:05 am # REPLY

Great Thanks

Suleyman Suleymanzade September 6, 2018 at 7:18 am # REPLY

This is two neuron networks that I tried to merge by using concatenate operation. The network should classify IMDB movies reviews by 1-good and 0-bad movies
I have an error in model's fit (training):
history = conc_model.fit([X_train, X_train], [y_train, y_train], np.reshape([y_train, y_train], (25000,2)),epochs=3, batch_size=(64,64))
TypeError: fit() got multiple values for argument 'batch_size'.
This is the method that should return trained model. BTW x_train shape (25000, 5) and y_train shape (25000,)

def cnn_lstm_merged():
embedding_vecor_length = 32
cnn_model = Sequential()
cnn_model.add(Embedding(top_words, embedding_vecor_length,input_length=max_review_length))
cnn_model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
cnn_model.add(MaxPooling1D(pool_size=2))
cnn_model.add(Flatten())

lstm_model = Sequential()
lstm_model.add(Embedding(top_words, embedding_vecor_length, input_length=max_review_length))
lstm_model.add(LSTM(64, activation = 'relu', return_sequences=True))
lstm_model.add(Flatten())

merge = Concatenate([lstm_model, cnn_model])
hidden = Dense(1, activation = 'sigmoid')

conc_model = Sequential()
conc_model.add(merge)
conc_model.add(hidden)

conc_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = conc_model.fit([X_train, X_train], [y_train, y_train], np.reshape([y_train, y_train], (25000,2)),epochs=3, batch_size=(64,64))
return history

Jason Brownlee September 6, 2018 at 2:09 pm # REPLY

Sorry, I don't have the capacity to debug your new code, perhaps post it to stackoverflow?

Emmanuel September 14, 2018 at 2:56 pm # REPLY

Very Nice Tutorial!

I adapted your version to fit on my data saved in pandas' dataframe. Since I have a limited data set, the evaluated score has reached to 100% which is a great improvement as compared to my previous, BOW model, 98%.

You can find the complete code at my github:

https://github.com/eanunez/ArticleClassifier/blob/master/cnn/Multi-Channel%20CNN.ipynb

Thank you so much Jason!

Kind Regards,
Emmanuel

Jason Brownlee September 15, 2018 at 6:01 am # REPLY

Well done!

sanjie December 3, 2018 at 1:40 am # REPLY

hello Emmanuel,

thanks for the updated version on your github, when making multi-classes fit, you should add the code

```
from keras.utils import to_categorical
 if len(self.classes_) > 2:
             train_y = to_categorical(train_y)
```

after your code

```
train_y = self.encode_classes(train_df['label'].values)
```

Thank Jason Brownlee again, i am really one of your fans.

Astariul October 16, 2018 at 7:15 pm #

REPLY

Hi Jason,

My first comment on this amazing blog of yours, even though I read a lot of your article, all of them useful.

Anyway I have a question about MaxPooling layers : why do we need them at all ?

Is it just for reducing the dimension ? Because I feel that using it will make us loose some information we learned in the Convolution layer.

###

I my application I want to work with embeddings but n-grams as well. If I have the sentence 'I like you', I want to end up with a tensor of dimension [?, 6, d] (d is the dimension of embeddings). The tensor would represent :
'I'
'like'
'you'
'I like'
'like you'
'I like you'

So I want to use the basic embeddings for the 3 first token, and apply a 2-gram convolution layer to get the 2 next token, and finally a 3-gram convolution layer for the last token. Then I concatenate everything (choosing a kernel size adequately).

In this case, why would I want to apply a MaxPooling layer ?
Do you think my approach could work ?

Thank you and keep up the good work 🙂

Jason Brownlee October 17, 2018 at 6:48 am # REPLY

Yes, we use the pooling layer to distil the large feature maps down to the most essential.

Try with and without the pooling layer and use the model that gives the best performance.

Prem Kumar October 23, 2018 at 10:59 pm # REPLY

Hi Jason,

Its very much helpful for me to learn about NLP and its tasks. Thank you very much for your work.. please do the same for computer vision problems too. I can't find a blogs like yours anywhere for computer vision problems. Please consider it..

Jason Brownlee October 24, 2018 at 6:28 am # REPLY

Thanks for the suggestion.

prisilla December 31, 2018 at 5:00 am # REPLY

Hi Jason,

For one of my input values when i run the CNN code
The output i received is

```
Epoch 1/100
250/250 [==============================] – 77s 308ms/step – loss: -2.8596 – acc: 0.7843
Epoch 2/100
250/250 [==============================] – 76s 303ms/step – loss: -2.9239 – acc: 0.7863
Epoch 3/100
250/250 [==============================] – 75s 300ms/step – loss: -2.8824 – acc: 0.7904
Epoch 4/100
250/250 [==============================] – 74s 294ms/step – loss: -2.9322 – acc: 0.7851
```

Do you think the model is correct , as the loss is in negative values.

Thanks,

Jason Brownlee December 31, 2018 at 6:15 am # REPLY

Negative loss is interesting. Something odd might be going on.

Niko January 9, 2019 at 3:46 am # REPLY

Hello Jason,

I wanted to apply grid search on this model (3-channels) by following your other tutorials but I'm getting this error

>ValueError: Found input variables with inconsistent numbers of samples: [3, 8000]

This is the code:
# grid search
epochs = [1, 10]
batch_size = [16, 32]
param_grid = dict(epochs=epochs, batch_size=batch_size)
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring='accuracy')
grid_result = grid.fit([trainX,trainX,trainX], array(trainLabels))

I have tried googling but I didn't find any answers for it.

Thank you in advance!

Jason Brownlee January 9, 2019 at 8:48 am # REPLY

Perhaps try running the grid search loops manually.

Roi Ong February 1, 2019 at 12:39 am # REPLY

Hi Jason! Does a prediction value close to either class mean that it has higher confidence? say 0.99 for this sentence A while 0.65 predicting this sentence B which means that the model predicts with higher confidence on sentence A compared to sentence B OR does it have anything to do with overfitting? a value produced like sentence A is due to being too closely fitted to the data which may cause erroneous predictions for the model in the future because it can prioritize sentences like sentence

A?

Jason Brownlee February 1, 2019 at 5:39 am # REPLY

A larger predicted probability could be interpreted as higher confidence in the prediction.

Roi Ong February 3, 2019 at 4:44 am # REPLY

Thank you so much Jason!

Asmaa M. Elmohamady February 1, 2019 at 11:15 pm # REPLY

can i used shared input instead of using input on every CNN channel ??

Jason Brownlee February 2, 2019 at 6:17 am # REPLY

What do you mean exactly?

You have the data once in memory and provide multiple references to it.

Asmaa M. Elmohamady February 5, 2019 at 12:17 am # REPLY

sorry, I didn't receive notification about your reply.
I mean if i use one input with one embedding can i use it once with parallel different kernel convolution and this is also called multichannel or not?

def define_model(length, vocab_size):
inputs1 = Input(shape=(length,))
embedding1 = Embedding(vocab_size, 100)(inputs1)

# channel 1
conv1 = Conv1D(filters=32, kernel_size=4, activation='relu')(embedding1)
drop1 = Dropout(0.5)(conv1)

```
pool1 = MaxPooling1D(pool_size=2)(drop1)
flat1 = Flatten()(pool1)
# channel 2
conv2 = Conv1D(filters=32, kernel_size=6, activation='relu')(embedding1)
drop2 = Dropout(0.5)(conv2)
pool2 = MaxPooling1D(pool_size=2)(drop2)
flat2 = Flatten()(pool2)
# channel 3
conv3 = Conv1D(filters=32, kernel_size=8, activation='relu')(embedding1)
drop3 = Dropout(0.5)(conv3)
pool3 = MaxPooling1D(pool_size=2)(drop3)
flat3 = Flatten()(pool3)
# merge
merged = concatenate([flat1, flat2, flat3])
# interpretation
dense1 = Dense(10, activation='relu')(merged)
outputs = Dense(1, activation='sigmoid')(dense1)
model = Model(inputs=[inputs1, inputs2, inputs3], outputs=outputs)
# compile
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# summarize
print(model.summary())
plot_model(model, show_shapes=True, to_file='multichannel.png')
return model
```

Jason Brownlee February 5, 2019 at 8:24 am # REPLY

Yes, I have to call this a mutli-headed model and save "channel" to refer to the depth of a given input.

Asmaa M. Elmohamady February 5, 2019 at 10:37 pm # REPLY

what do you mean with save "channel" to refer to the depth of a given input.

Jason Brownlee February 6, 2019 at 7:45 am # REPLY

As in an input image has 3 channels, one for red/blue/green.

And, the depth of a stack of feature maps is referred to as the channels.

sanker February 13, 2019 at 7:50 pm # REPLY

sir,
i am doing a project on “paraphrase detection using deep learning”.
i have two inputs, as two sentence . both sentence want to be separate training. how i fit my model???

Jason Brownlee February 14, 2019 at 8:42 am # REPLY

Perhaps start with a strong definition of your prediction problem:
http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/

Minel March 5, 2019 at 4:55 am # REPLY

Hello jason
There is a small typo in the beginning of the example
table = str.maketrans(”, ”, string.punctuation)
but it is fixed in the whole example
table = str.maketrans(”, ”, punctuation)
best

Jason Brownlee March 5, 2019 at 6:41 am # REPLY

I don’t believe so, are you sure?

Minel March 5, 2019 at 7:30 pm # REPLY

Yes only in the first example

# turn a doc into clean tokens

```
def clean_doc(doc):
# split into tokens by white space
tokens = doc.split()
# remove punctuation from each token
table = str.maketrans(", ", string.punctuation)
tokens = [w.translate(table) for w in tokens]
```

Jason Brownlee March 6, 2019 at 7:48 am # REPLY

The first example in section "Loading and Cleaning Reviews" import's string "import string" then uses "string.punctuation".

It is correct Python code.

Perhaps I misunderstand your comment?

Minel March 6, 2019 at 11:05 pm #

Hello Jason

Yes you are right. I merged the code of two examples that is why I got a traceback
Sorry about that

Jason Brownlee March 7, 2019 at 6:50 am #

No problem.

Steve March 24, 2019 at 3:00 pm # REPLY

Hi Jason,

I've followed your article and it was really helpful.
Right now i'm stucked, the model val_loss keeps increasing while the val_acc keeps increasing as well.

I followed your article on improving overfitting, but adding dropout layers didn't work at all. I tried improving the amount of training data which in fact made the results even worst.

I've posted a question explaining top to bottom about my problem in stackoverflow.
It'll be really helpful if you can take a look at the question.

Here's the link
https://stackoverflow.com/questions/55320567/validation-loss-increases-after-3-epochs-but-validation-accuracy-keeps-increasin

I'm looking forward for your answer.

Thanks.

Jason Brownlee March 25, 2019 at 6:41 am # REPLY

Perhaps try using SGD, reducing the learning rate, and increasing the number of training epochs.

saurabhk April 18, 2019 at 3:14 pm # REPLY

CNNs would try learning the padded sentences directly which would result in noisy learned representations, how do we ignore padded value so it has no impact on CNN filter learning?

Jason Brownlee April 19, 2019 at 6:04 am # REPLY

Typically CNNs ignore zero padded values, e.g. they use padding all the time as part of performing convolutions.

SK April 19, 2019 at 10:12 pm # REPLY

Hey Jason,

Thanks for this easy-to-follow tutorial. I do not get any improvement with a multi-channel convnet compared to a single convnet with a 3-gram kernel and more filters (128 instead of 32) with GlobalMaxPooling. Do you have any ideas why that would be ? Can you suggest any effective tweaks to improve a multi-label text classifier ?

Best regards

SK

Jason Brownlee April 20, 2019 at 7:38 am # REPLY

I have some suggestions here that might help:
https://machinelearningmastery.com/start-here/#better

SK April 25, 2019 at 9:37 pm # REPLY

Hey Jason,

Thanks for the very useful link.

Best
Sudheer

Jason Brownlee April 26, 2019 at 8:33 am # REPLY

You're welcome.

maunish June 10, 2019 at 7:04 pm # REPLY

Hi , jason great article

can you make a article on what are the most common ways to approach a text related Machine Learning problem

Jason Brownlee June 11, 2019 at 7:45 am # REPLY

Yes, start here:
https://machinelearningmastery.com/start-here/#nlp

maunish June 14, 2019 at 8:03 pm # REPLY

I wrote a fake review as

"this was wonderfull movie
this movie is amazing
actors have acted very well and performance was outstanding
i will watch this movie again"

then is used same model to predict if review was good or bad

result was 0.50316983 but i was expecting more as review is straight foreword
i have used more posotive words and avoided any confusting words

so what is the problem here ?

Jason Brownlee June 15, 2019 at 6:32 am # REPLY

Perhaps the model was overfit, you could try fitting it again?

SP June 26, 2019 at 5:06 pm # REPLY

Hi Jason,

Thanks for your wonderful tutorial. One query: Can it support multilingual ?. I mean if the dataset in other than English, does it require any change in word embedding ?. Or still, it works.

Thanks

Jason Brownlee June 27, 2019 at 7:45 am # REPLY

You can train your own word embedding for any language as far as I know.

SP June 27, 2019 at 8:26 pm # REPLY

Thank you, Jason, for your response. Have you used any word embedding(ex:w2v/glove) in your code ?. I could able to see only Keras tokenizer function (correct me if I am wrong). When I ran adapting your code to another language and different dataset, it ran smoothly, so asked if it required any word embedding specifically.

Thanks

Jason Brownlee June 28, 2019 at 6:01 am # REPLY

Yes many times. Perhaps start here:
https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

Zeinab July 7, 2019 at 12:39 am # REPLY

What is the difference between CNN and the multichannel CNN?

Why I need to used the multichannel CNN?

Jason Brownlee July 7, 2019 at 7:52 am # REPLY

It can use a separate kernel size on the same input data and effectively "see" the data at different resolutions at once.

Zeinab July 7, 2019 at 7:52 pm # REPLY

Thank you

Jason Brownlee July 8, 2019 at 8:40 am # REPLY

You're welcome.

zeinab July 7, 2019 at 11:08 pm # REPLY

Below is a function for calculating the correlation coefficient. I use it to measure the accuracy degree for a regression problem (text similarity). I use the lstm and the multichannel cnn, however the correlation degree results with -ve values starting from the second epoch.
can you help me and check the correctness of this function?

```
def correlation_coefficient(y_true, y_pred):
    pearson_r, update_op = tf.contrib.metrics.streaming_pearson_correlation(y_pred, y_
    metric_vars = [i for i in tf.local_variables() if 'pearson_r'  in i.name.split('/'

    for v in metric_vars:
        tf.add_to_collection(tf.GraphKeys.GLOBAL_VARIABLES, v)

    with tf.control_dependencies([update_op]):
        pearson_r = tf.identity(pearson_r)
        return 1-pearson_r**2
```

Jason Brownlee July 8, 2019 at 8:42 am #

REPLY

This is a common question that I answer here:
https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

zeinab July 7, 2019 at 11:10 pm #

REPLY

```
def correlation_coefficient(y_true, y_pred):
    pearson_r, update_op = tf.contrib.metrics.streaming_pearson_correlation(y_pred, y_t
    metric_vars = [i for i in tf.local_variables() if 'pearson_r'  in i.name.split('/')
    for v in metric_vars:
        tf.add_to_collection(tf.GraphKeys.GLOBAL_VARIABLES, v)
    with tf.control_dependencies([update_op]):
        pearson_r = tf.identity(pearson_r)
        return 1-pearson_r**2
```

zeinab July 8, 2019 at 12:18 am #

REPLY

when I use the correlation coefficient as a metric,
the resulted correlation coefficient values are ranged from -1 to -85.

Does the resulted values means that there is a problem in the model?

Jason Brownlee July 8, 2019 at 8:43 am # REPLY

Perhaps start by reviewing the loss?

zeinab July 8, 2019 at 3:28 am # REPLY

Can the Pearson correlation coefficient decrease and the loss at the same time? what does this means?

I think the loss should decrease while the correlation coefficient should increase during the training time?

Jason Brownlee July 8, 2019 at 8:44 am # REPLY

Correlation would not make a good loss function.

zeinab July 8, 2019 at 3:50 am # REPLY

I run the multichannel cnn model on a text similarity problem.

Every time I run the model, I have different results(loss, accuracy).

Does it is normal to have different loss and accuracy results every time I run the code?

Jason Brownlee July 8, 2019 at 8:44 am # REPLY

This is to be expected, see this:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Bibin Antony October 3, 2019 at 11:30 pm # REPLY

how to do sentiment analysis for Galssdoor data with maximum accuracy considering the ratings also ?

Jason Brownlee October 4, 2019 at 5:42 am # REPLY

Perhaps test a range of data preparation methods and explore a range of CNN models to see what works.

Start here:
https://machinelearningmastery.com/start-here/#nlp

Then use the tutorials here to get better results:
https://machinelearningmastery.com/start-here/#better

zeinab October 15, 2019 at 5:54 pm # REPLY

What is the advantage of using multichannel CNN over just one CNN?

Jason Brownlee October 16, 2019 at 7:58 am # REPLY

It can read the input in different ways in parallel, e.g. extract different features from the same input within one model.

moSaber December 9, 2019 at 12:49 am # REPLY

Hi Jason,

When I run the complete model as is, I get 50% accuracy rate (see following output). I've tried to add more layers to the CNN, used different kernals, tweaked the hyper parameters in the embedding layers .. etc. still getting the same accuracy rate. Would you be able to advise how I can boost the accuracy? Also, I've noticed that when I run the code for the first time, I get 99% accuracy rate on test data and 50% on test data.

Max document length: 1380
Vocabulary size: 44277
(1800, 1380) (200, 1380)
Train Accuracy: 50.000000
Test Accuracy: 50.000000

Jason Brownlee December 9, 2019 at 6:52 am # REPLY

You can discover suggestions for diagnosing and improving deep learning performance here:
https://machinelearningmastery.com/start-here/#better

James December 10, 2019 at 5:54 am # REPLY

Hi Jason, thanks for the tutorial. Would you be able to write a tutorial on aspect-based sentiment analysis with Keras? That would be awesome!

Jason Brownlee December 10, 2019 at 7:36 am # REPLY

What is "aspect-based sentiment analysis"?

Matip December 15, 2019 at 9:57 am # REPLY

How to make a prediction with a sentence using your tutorial

Jason Brownlee December 16, 2019 at 6:05 am # REPLY

Prepare the text in the same way as the training data, then call model.predict()

For help see this:
https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

Ryan Lambert January 21, 2020 at 10:12 am # REPLY

Hi Jason – I tweaked this to run predictions on other text documents using the above movie review data to train on. I had tried to scoring predictions on text strings such as “this movie sucks do not recommend” and got quite poor results (model predicted 65% probability it was a positive review).

Just wondering if there are additional training resources to build into the model to get a truly rounded approach to positive/negative sentiment? It seems to work well with large paragraph/eloquent reviews but not so well with short 5-10 word reviews like “this movie was fantastic”.

Jason Brownlee January 21, 2020 at 2:52 pm # REPLY

Nice work, and interesting finding.

Yes, train on more diverse data or data more appropriate for the way the model is intended to be used.

HSA February 13, 2020 at 8:58 pm # REPLY

How about in each channel I put different embedding?
embedding1=random embedding (channel1)
embedding2=word2vec embedding (channel2)
does this seem right Idea or not?

Jason Brownlee February 14, 2020 at 6:33 am # REPLY

Try it. Results matter more than opinions.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

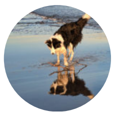
Welcome!

My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.

Read more

Never miss a tutorial:

Picked for you:



How to Develop a Deep Learning Photo Caption Generator from Scratch

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras

How to Develop a Neural Machine Translation System from Scratch

How to Use Word Embedding Layers for Deep Learning with Keras

Text Generation With LSTM Recurrent Neural Networks in Python with Keras