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[illegible] containing the text "*neg*" and "*pos*" for negative ar[illegible]e with a naming convention *cv000* to *cv999* for each neg and pos.

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.

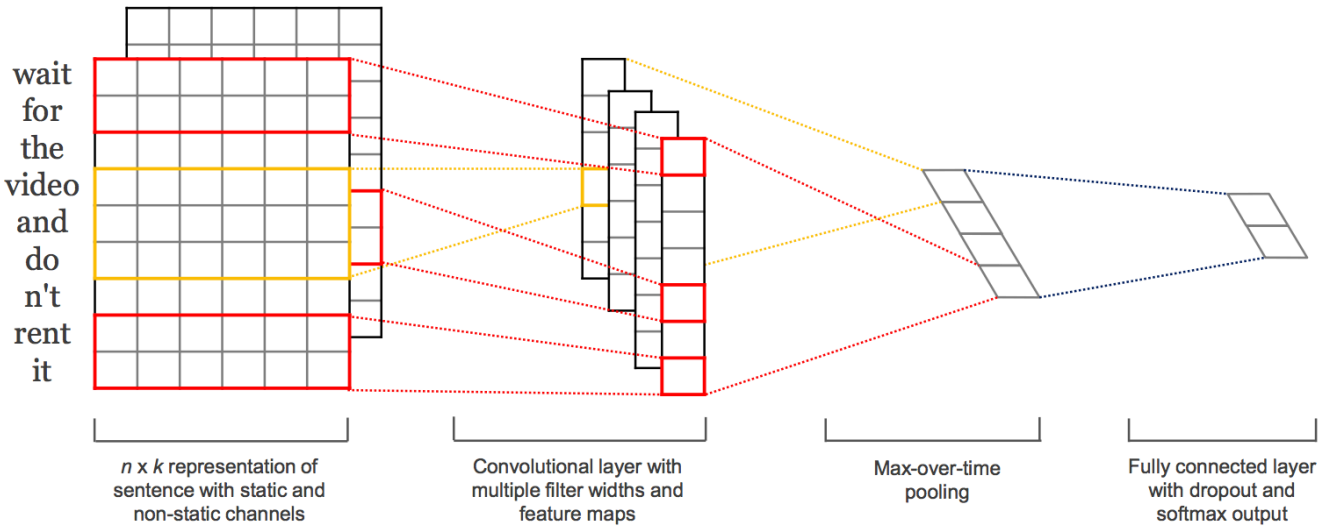
The kernel size in the convolutional layer defines the number of words to consider as the convolution is passed across the input text document, providing a grouping parameter.

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:



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.

```
# 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.

```
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.

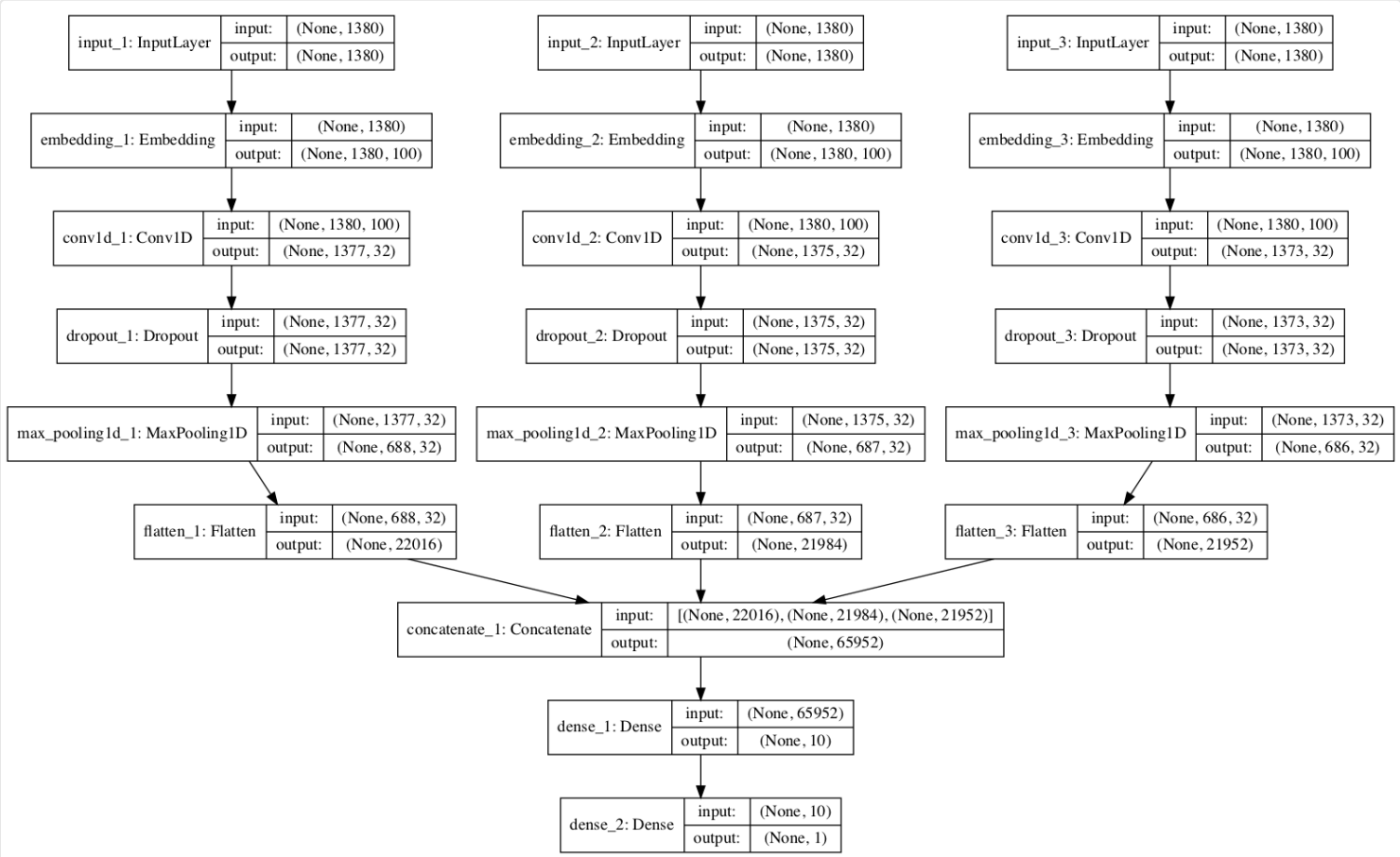
```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (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.



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.

```
# 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.

About Jason Brownlee