Classify a book's category based on its title

Yunkai Wang
Carleton University, School of Computer Science

Jules Kuehn
Carleton University, School of Computer Science

Recently Convolutional neural networks(CNN) have been well-studied and be shown that they can archive incredible results on tasks like sentence classification (Kim, 2014). Another interesting topic that has been studied is if we can categorize a book based on its cover picture (Iwana et al.). However, as shown in the paper, they showed that they were able to draw a relationship between a book cover image, and its category, but the accuracy was less than 50%. Thus, we conduct this experiment to see if we can use CNN to learn the potential relationship between a book's title and its categories. For this project, we experiment the task with different model setups and layers using word embedding, and compare their accuracies.

I. Introduction

Book titles will given the reader a first impression of what the book may be about, and most of the time, a good book title will attract more readers from buying and reading the book. But what does the book title really tells you?

II. The Dataset

The *Data Mining* is a dataset which can be found on *Github.com*. The set consists of detailed information of 207572 books from 32 categories, to help determining the potential relationship between different information related to a book, like the relationship between a book's cover image and its category. All books in the dataset have informations like the image url of the cover image of the book, the book's author, and the book's category (Brian et al.). The dataset doesn't have the training dataset and testing set splitted by default, so we just use make a use of the *sklearn* library, which provides a nice *train_test_split* function to create the training set and test set.

Transforming the data

All the titles are made up with words, which is hard to be feed to the neural network as input, to fix this problem, we make the following transformations to the titles:
1. Create a vocabulary set containing all possible words that have appeared in one of title. The titles in the dataset used a total of 71056 words.
2. We extend the titles so that all titles all have the same length. The longest title in the dataset has 96 words, but most of the titles are short (5-10 words). Figure 1 is a graph showing the distribution of title length among the entire dataset.
3. We convert the titles into array of integers, where each integer corresponds to the index of the word in the vocabulary list that we created, the index is between 1 and 71057, and we map all the special character that we added in step 2 to 0. This vectorization process takes $\approx$ 2 hrs to run, therefore, we decide to store the vectorized data into a new .csv file, which we can use to train our data directly.

III. Model

We began with 2 naive implementations, in which each "word vector" was an arbitrary unique integer. Each title is then a vector of these integers (one for each word), zero padded to the length of the longest title (96 words). This representation of the titles has little semantic value as the integers representing the words are arbitrary.
The first model (NaiveMLP) was a fully connected MLP with the following configuration. Output size of each layer is indicated in parentheses.

- Input (96)
- FC Sigmoid (625)
- FC Sigmoid (300)
- FC Softmax (32)

The second model (NaiveCNN) was based on a good model for the CIFAR-10 classification.

- Input (96,1)
- Conv ReLU: 3 kernel, 1 stride (96,32)
- Conv ReLU: 3 kernel, 1 stride (96,32)
- Max pool: 2 kernel, 2 stride (48,32)
- Dropout: pKeep 0.8 (48,32)
- Conv ReLU: 3 kernel, 1 stride (48,64)
- Max pool: 2 kernel, 2 stride (24,64)

- Conv ReLU: 3 kernel, 1 stride (24,64)
- Max pool: 24 kernel, 24 stride (1,64)
- Dropout: pKeep 0.8 (1,64)
- Output Softmax (32)

A better word representation is to embed each word in a n-dimensional space, where the position of each word reflects its meaning in relation to the other words. Each word is then represented by a dense vector of length n, with similar words having similar vectors. Similarity between words is inferred from context. Many pre-trained embeddings are available, trained through Word2Vec or GloVe on datasets pulled from Wikipedia, Twitter, or other sources. An embedding can also be trained from scratch, or initialized to a pre-trained dataset then trained further. We compare these three approaches.

The first embedding model (EmbedTrain) trained the embeddings from scratch, using only the book titles from the training set. Note that we truncated the titles from a maximum of 96 words to 26 words. We chose 32 for the embedding dimension, based on the rule of the thumb that the embedding should be the fourth root of the vocabulary size. There were roughly 70,000 words in the training dataset, which suggest that the embedding dimension should be 16, but we found that 32 offered a slight improvement in practice.

- Input (26, 1)
- Embedding (26, 32)
- Flatten (832)
- Output Softmax (32)

For each of the three embedding models, we also tested 2 variations to compare the performance of MLP vs CNN on the title embeddings. The first variation (Embed<Model>_FC) adds a single fully connected (FC) layer:

- Input (26, 1)
- Embedding (26, 32)
- Flatten (832)
- FC ReLU (512)
- Output Softmax (32)

The second variation (Embed<Model>_CNN) adds a single convolutional layer, followed by maxpool over the entire length:

- Input (26, 1)
- Embedding (26, 32)
- Conv ReLU: 3 kernel, 1 stride (24, 512)
- Max pool: 24 kernel, 24 stride (1, 512)
- Flatten (512)
- Output Softmax (32)

The second embedding model (EmbedGloveFixed) loaded 400,000 pre-trained 100-dimensional word-vectors from the GloVe.6B dataset. These word vectors were trained on Wikipedia and Gigaword, a newswire dataset. The model is identical to EmbedTrain except that the embeddings are pre-trained and fixed, with the embedding dimension increased to 100.

The third embedding model (EmbedGloveTrain) is identical to EmbedTrain except that the embeddings are initialized to the GloVe dataset as seen in EmbedGloveFixed. Embeddings are then retrained on the book titles.

IV. Implementation

The naive models were implemented using the course-provided cnn.py and mlp.py code. All other models were implemented in Keras, in which the code for the model is extremely concise:

```
model = Sequential()
model.add(Embedding(vocab_size, 32,
   input_length=max_title_length))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(32, activation='softmax'))
model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam())
model.fit(
    trX, to_categorical(trY), epochs=5)
```

V. Experiment

a

VI. Conclusion

a

References

Convolutional Neural Networks for Sentence Classification By Yoon Kim
Judging a Book by its Cover By Brian et al.