Image Captioning using CNN-LSTM and GRU

Abstract

Image caption generation/ image summarization is a task that involves generating a semantic description of an image in natural language and is currently accomplished by techniques that use a combination of computer vision (CV), natural language processing (NLP), and machine learning methods. The inspiration for such an application can be inferred from Social media platforms like Facebook(Now Meta), that summarize the image posted by the user and infer details like - where you are, what you wear etc. This application also has a profound use in assisting visually impared individuals in comprehending the images of the real world. In this task, we work on a model that generates natural language description of an image. We intend to use a combination of convolutional neural networks to extract features and then use recurrent neural networks to generate text from these features. We incorporated the attention mechanism while generating captions. We evaluated the model on the Flickr 8k database.

Problem Statement

Given an image, we want to obtain a sentence that describes what the image consists of.

Approach

For image caption generation, we will be primarily using two models :

1. LSTM (Long Short Term Memory)
2. GRU (Gated Recurrent Unit)

For better accuracy we will be extending our model through Hyperparameter Tuning and provide visualization for the outputs and the results of the models.

For Hyperparameter Tuning we have the following 4 configurations -

- Convolutional Neural Network(CNN), Long Short Term Memory(LSTM), Optimizer : Adagrad and Epoch runs at 10, 25
- Convolutional Neural Network(CNN), Long Short Term Memory(LSTM), Optimizer :Adamand Epoch runs at 10, 25
- Convolutional Neural Network(CNN), Gated Recurrent Unit(GRU), Optimizer: Adagrad-and Epoch runs at 10, 25
- Convolutional Neural Network(CNN), Gated Recurrent Unit(GRU), Optimizer: Adam and Epoch runs at 10, 25

Evaluation Metrics for Image Captioning

The current codebase uses NLTK to calculate BLEU scores. However, BLEU-1 to BLEU-n can be easily implemented. We are using the NLTK library for doing this which provides a nice interface to achieve this.

Here is the explanation of how BLEU score computation is defined:

BLEU-n is just the geometric average of the n-gram precision.

```
(precisely it's string matching, at different n-gram levels, between
references and hypotheses;
That's why there has been much criticism on this metric.
But, people still use it anyways because it has stuck with the community
for ages)
```

For example, **BLEU-1** is simply the **unigram precision**, **BLEU-2** is the **geometric average of unigram and bigram precision**, **BLEU-3** is the **geometric average of unigram, bigram, and trigram precision** and so on.

Having said that, if you want to compute specific **n-gram BLEU scores**, you have to pass a weights parameter when you call **corpus_bleu** . Note that if you ignore passing this weights parameter, then by default BLEU-4 scores are returned, which is what is happening in the evaluation here.

To compute, BLEU-1 you can call copus_bleu with weights as:

```
weights = (1.0/1.0, )
corpus_bleu(references, hypotheses, weights)
```

To compute, BLEU-2 you can call corpus_bleu with weights as:

```
weights=(1.0/2.0, 1.0/2.0,)
corpus_bleu(references, hypotheses, weights)
```

To compute, BLEU-3 you can call corpus_bleu with weights as:

```
weights=(1.0/3.0, 1.0/3.0, 1.0/3.0,)
```

```
corpus_bleu(references, hypotheses, weights)
```

To compute, BLEU-5 you can call corpus_bleu with weights as

```
weights=(1.0/5.0, 1.0/5.0, 1.0/5.0, 1.0/5.0, 1.0/5.0,)
corpus_bleu(references, hypotheses, weights)
```