**Show and tell: Image captioning**

**IE 534 - Final Project Report**

*December 12th, 2018*

1. **Introduction**

The aim of this project is to describe images using properly formed English sentences by implementing a generative model with a deep recurrent architecture. I reproduced the model architecture and procedure developed in the paper “Show and Tell: A Neural Image Caption Generator, Vinyals et. al.”, which consists of a neural image caption generator (NIC). Given this paper architecture and model I proceed to build my own NIC model using PyTorch, which consists of an image encoder (a regular CNN, I used Resnet), and an image decoder (LSTM) and caption generator. I trained My model both in MSCOCO and Flickr8k datasets, obtaining a working image captioning model. My best testing BLEU score is 16.31 for MSCOCO with a Resnet152 encoder, and BLEU 12.8 for Flickr8k using a Resnet101 encoder. This is compared to BLEU score 27.7on MSCOCO**,** which was obtained in ‘Show and Tell’ by the paper authors.

1. **Model structure**

I used, following Vinyals et. al., a combination of a CNN as an “image encoder” and an RNN (LSTM) “decoder” using BeamSearch inference for sentence generation, which produces “k-best hypothesis” sentences. This forms the so-called Neural Image Caption (NIC) model architecture. The model was trained to maximize the likelihood p(S | I) of the target description sentence S given the training image I. The LSTM is state-of-the-art for sequence tasks, and thus here it will predict the next word of a sentence after it has seen the image as well as the preceding words (starting with the token <start> and ending with <end>). I used a stateful LSTM for sequence processing, and to maximize performance I also used a locked dropout with a common mask for the entire sequence.

For the encoder I used both Resnet152 and Resnet101, as it is defined in the file named train.py before training starts. I deleted the last fully connected layer of the Resnet, and used that model as an image encoder.

The decoder model is defined in the file decoder\_final.py. The decoder accepts an input of an encoded image produced by a Resnet without the last fully connected layer; I flatten the encoded image and put this into a linear fully connected layer which will output an image embedding. This fully connected layer is thus trained with the encoder. After that, I put this image embedding as the first input for stateful LSTM. Then, following the procedure in Vinyals et. al., I put an actual caption sentence that corresponds to the previous image into an embedding layer obtaining an embedding for the caption and then I put this, word by word, into My LSTM cell, until the sentence reaches <end>. This decoder is a class named NIC\_language\_model in decoder\_final.py.

1. **Datasets**

I started by training, validating and testing My NIC architecture on the MSCOCO dataset. This dataset is completely available for public download here: http://cocodataset.org/#download. I used a different number of train/validation/test examples as the one proposed in the paper. I have: ~113287 training images, ~5000 validation images, and ~5000 testing images.

I also trained other models for the Flickr8k dataset with ~6000 training images, ~1000 validation images, and ~1000 testing images.

1. **Pre-processing**

The pre-processing of these datasets is submitted in the create\_input\_files.py and utils.py python files. As an input for pre-processing I use a zip file containing the captions for the MSCOCO, Flickr8k and Flickr30k datasets prepared by Andrej Karpathy (source: http://cs.stanford.edu/people/karpathy/deepimagesent/caption\_datasets.zip).

For pre-processing I take the following steps:

1. Capture each image, caption and caption length of train, validation and test datasets and save them in the different paths.
2. After reading each image and resizing them to (3, 256, 256), I save the images to HDF5 file.
3. I create a vocabulary containing words where the frequency threshold of the words in all the captions is equal to five, i.e. I didn’t control the size of the vocabulary but the minimum frequency of the words in the dataset. Also, I added four additional tokens that correspond to the words <start>, <end>, <pad>, and <unk>. Each sentence starts with <start> and ends with <end>. I established the maximum sequence length of each caption to 50 and if a caption is smaller than 50, I add <pad> to increase the size of each caption to 50. If a word in a caption is not present in the vocabulary, that word is encoded as <unk>.
4. Finally, each image must have five captions. If an image has less than five captions, I generate extra random captions for that image to make sure the number of captions is five.
5. **Training procedure**

The training procedure is included in the train.py file. I first define the image encoder (which is either Resnet101 or Resnet152), and I load a pretrained Resnet on the ImageNet dataset. I also eliminate their last fully connected layer (and include this fully connected layer in the decoder forward function, so that it gets trained even without fine-tuning). I also define the optimizer (either Adam or SGD). In my code there is a so called ‘checkpoint’ that corresponds to a previous trained model in case there is one and in case I start training a model from scratch checkpoint is set to None. I require the encoder Iights to update (fine-tuning) with a boolean value called ‘fine\_tune\_encoder’. I set this to False in the beginning and start training only the decoder, which I did for 30 to 40 epochs at a speed of 2.5 hMys per epoch. I also used the validation dataset to get a validation BLEU score and validation accuracy, and this was the factor that determined My “early stopping” procedure. I defined my best model as the best BLEU score in the validation dataset. I could see by looking at validation that my best model were the ones trained until epochs around 20 for Adam optimizer, and around 40 for SGD optimizer, I report those model BLEU scores. With the result of those models I finally put some of them to train both the encoder and decoder (fine-tuning), which took 8 hours per epoch, but I didn’t see any improvements on validation BLEU score after a few epochs of fine-tuning training.

During training I also adjust the learning rate by a factor of 0.8 if there is no improvement in the best validation BLEU score for 8 epochs. After each epoch I would save a checkpoint with a custom function save\_checkpoint included in the utils.py file.

The train.py is organized around three main functions: main, train, and validate. The procedure of the main function was described in this section, and the train and validate procedures are the usual training (with loss.backward) and validation (without loss.backward). In both of these procedures I calculate print the loss and calculate the top5-accuracy, which is a measure of the percentage of generated captions that have the first most probable words included in the actual captions. I also calculate and report the time that each training and validation epoch takes using an AverageMeter() function included in utils.py. After validation is finished I calculate the BLEU validation score and the best score is what I report in the quantitative results section. Some of this code is based on a github implementation of the paper ‘Show, Attend and Tell’ ([github implementation](https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning)) that I adapted for my purposes. The model however was written from scratch.

* *BLEU scores explanation*

I are using the concept of ‘Teacher Forcing’, for the validation in the training part. The main advantage of this method is to speed up the process of validating during the training. An important distinction to make here is that I’re still giving the encoded captions(ground-truth) as the input during validation, irrespective of the word last generated.

* *Computation time*

I used a total of 800 training hours in BlueWaters and 50 hours in vectordash, which were divided in all the models I trained (both for successful and defective models). I am only presenting the results of the best models, that are trained after debugging the codes for this project.

* *Evaluation*

I calculate the testing BLEU score using a k-beam size algorithm (with k reported in quantitative results table). This code is included in the eval.py file. The BLEU score in this case is calculated by providing the encoded image and a <start> token to the decoder, which then generates the next word that is fed again into the LSTM. I have stopped the generation process at time step of 50.

1. **Quantitative results**

In the following table I report each of the best trained models, their hyper-parameters, other training properties, and final BLEU scores, and finally I compare these scores with the ones reported in the original paper.

Table 1- Hyper-parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training properties | Training 1 | Training 2 | Training 3 | Training 4 |
| Encoder | Resnet101 | Resnet152 | Resnet152 | Resnet101 |
| Dataset | MSCOCO | MSCOCO | MSCOCO | Flickr8k |
| Optimizer | Adam | Adam | SGD, 0.9 momentum | Adam |
| Learning rate | 5e-4 | 5e-4 | 5e-4 | 5e-4 |
| Batch size | 128 | 128 | 128 | 128 |
| Encoder fine-tuning | No | No | No | No |
| Early stopping | Epoch 20 | Epoch 13 | Epoch 37 | Epoch 6 |
| Validation BLEU score  (Teacher Forcing) | 19.08 | 20.02 | 11.10 | 10.9 |
| Beam Size | 2 | 2 | 2 | 2 |
| Testing BLEU score | 14.99 | 16.31 | - | 12.87 |

I can observe from the BLEU score for the SGD optimizer on MSCOCO dataset that this model performed worse than the rest of the models. A possible reason is that I used the same learning rate for both Adam and SGD. But, SGD might require a higher learning rate than the given, thus causing this discrepancy. The BLEU scores reported by the original paper Ire 27.7.

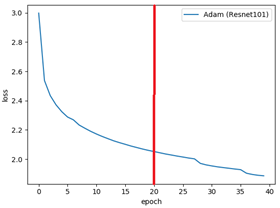
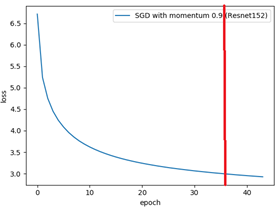
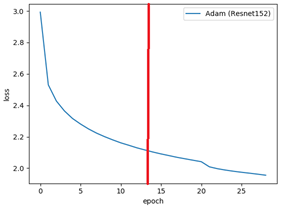
**  **

Figure 1- Loss chart for different models (Red line is early stopping which corresponds to the best validation BLEU score)

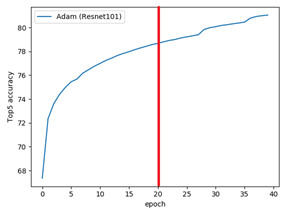
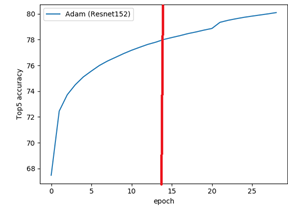
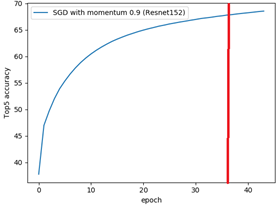
  

Figure 2- Top-5 accuracy score chart for different models (Red line is early stopping which corresponds to the best validation BLEU score)

1. **Qualitative results**

Some good captions generated by My model are shown below. Beam size for left, middle and right pics are 4, 3 and 4 respectively.

Generated captions:

[['<start>'], ['fMy'], ['dogs'], ['in'] [['<start>'], ['a'], ['small'], ['dog'], ['in'], [['<start>'], ['a'], ['boy'], ['in'],

, ['the'], ['snow'], ['<end>']] ['a'], ['grassy'], ['field'], ['<end>']] ['midair'], ['<end>']]

Original captions:

['a', 'group', 'of', 'dogs', 'stand', [‘a’ ,‘dog’, ‘playing’,’in’,’the’,’field’] ['a', 'boy', 'on', 'a', 'bicycle',

'in', 'the', 'snow'] 'in', 'midair']

Also, there are bad captions generated shown below. Beam size for left and right pics are 5, 5 and 4 respectively.

Generated captions:

[['<start>'], ['a'], ['little'], ['girl'], ['in'], [['<start>'], ['a'], ['black'], ['and'], [['<start>'], ['a'], ['white'], ['and'], ['white'],

['a'], ['green'], ['dress'], ['<end>']] ['a'], ['dog'], ['<end>']] ['and'], ['white'], ['dog'], ['<end>']]

Original captions:

['the', 'boy', 'is', 'riding', 'on', ['a', 'dog', 'catching', 'a', 'ball'] ['there', 'are', 'two', 'dogs',

'a', 'toy', 'truck']] 'running', 'on', 'the', 'beach', 'outside']