NLP Project 3 Writeup

Reimplementation of "Show, Attend and Tell" Image Captioning Model

Jing Jiang

Project Description

Image Captioning, or image-to-text translation, is a type of task combining natural language processing and computer vision. Given an image, the program can caption it with a sentence of a few words. Usually the image captioning model follows an encoder-decoder architecture. The encoder is commonly a convolutional neural network that can extract the features of an image. The decoder is commonly a sequential model such as LSTM. In this project, I reimplement the work from the "Show, Attend and Tell" model (https://arxiv.org/abs/1502.03044). The uniqueness of this model is that it uses Attention mechanism in the decoder part. I chose PyTorch 0.4 as the deep learning framework.

Dataset

In the task of image captioning, there are three commonly used datasets: MSCOCO, Flickr30k, and Flickr8k. I used Flickr8k, the smallest one among these three. In this dataset, there are 8000 images in total. Each image has 5 corresponding captions to describe the image. An example is shown below.

"a dog is running in a field"
"a multicolored dog running through a flowery field"
"dog with a thick collar running through the grass"
"dog with big collar running"
"the brown and white dog is running through a field of grass and pink flowers"

I split the dataset so in there are 6000 in the training set, 1000 in the validation set and 1000 in the test set. I used Andrej Karpathy's split in JSON format (https://cs.stanford.edu/people/karpathy/deepimagesent/).

Architecture of the Neural Network

For the Encoder, I used the ResNet101 model in my implementation. As the name suggest, ResNet101 has 101 neural layers in total. ResNet is short for Residual Network, it consists of Residual Blocks that enables shortcut connections. The Residual block number and configuration of ResNet101 is shown at the 4th column below.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

I downloaded and applied the pre-trained ResNet101 model in my implementation, considering that the time is short for training both the Encoder and the Decoder, and that the Encoder is doing a computer vision task, which is not the focus of this course.
For the Decoder, I used Attention model. The decoder contains two parts, the LSTM network and the Attention network. I iterated through PyTorch's LSTMCell class instead of using their LSTM class which wraps all cells together, because each output of the cell will also be used in the Attention layers. The Attention networks just consists of linear layers. The exact configuration of the decoder is shown below.

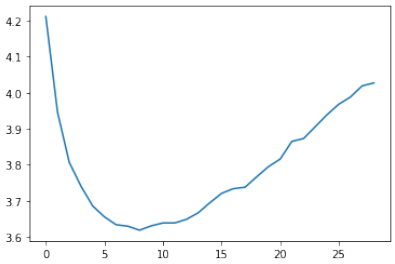
```
[13]: !python printdecodermodel.py

AttentionDecoder(
  (attention): Attention(
    (encoder_att): Linear(in_features=2048, out_features=512, bias=True)
    (decoder_att): Linear(in_features=512, out_features=512, bias=True)
    (full_att): Linear(in_features=512, out_features=1, bias=True)
    (relu): ReLU()
    (softmax): Softmax()
  )
  (embedding): Embedding(2633, 512)
  (dropout): Dropout(p=0.5)
  (decode_step): LSTMCell(2560, 512)
  (init_h): Linear(in_features=2048, out_features=512, bias=True)
  (init_c): Linear(in_features=2048, out_features=512, bias=True)
  (f_beta): Linear(in_features=512, out_features=2048, bias=True)
  (sigmoid): Sigmoid()
  (fc): Linear(in_features=512, out_features=2633, bias=True)
)
```

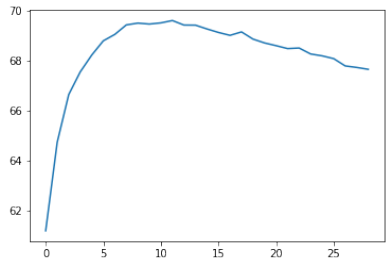
The learning rate for the decoder is set to be 0.0004 initially. Learning rate scheduler is applied, as the program will detect if the model has improvement over the last 5 epochs. If not, the learning rate will be adjusted by a factor of 0.8. Early stopping is also enabled. If the model hasn't improved over the last 20 epochs, the model will stop training. In my training, the model stopped after 30 epochs. I used bleu-4 score as the metric to determine the improvement of the model.

Metrical Results

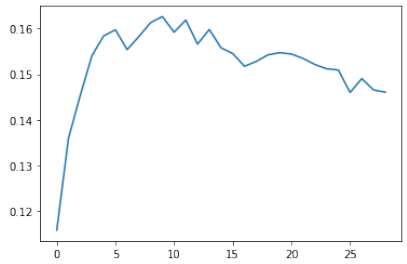
The following graphs shows the metrics of my model.



Epoch Loss



Top-5 Accuracy



Bleu-4 Score

As shown above, the best result is achieved at around epoch number 10. And after that, the model stopped its improvement. Note that the bleu-4 score is of a scale from 0 to 1.

Evaluation of the Model

The model applied Beam Search when inferring the results. The following picture shows an example of the instructional parameter used in the inference program.

```
[32]: !python caption.py \
      --img='./5.jpg' \
      --model='./BEST_checkpoint_run3.pth.tar' \
      --word_map='./modeloutput/WORDMAP.json' \
      --beam_size=5

a dog is running through the water
```

The model captions the objects that appears in the dataset well. It will also generate interesting results when it does not recognize the object. The third image is a good example. Part of the reason is because Flickr8k dataset is a small one compared to others.

“a man sitting on a bench”

“a dog is running through the water”

“a man in a red shirt is jumping off a red slide”