Show and Tell: A Neural Image Caption Generator

Zhejian Peng[1], Yutong Dai[2], Qi Tang[1], Xiang Cui[2], Shuhui Guo[2]

Abstract
This is a project report for reproducing and improving the methods proposed in the paper *Show and Tell: A Neural Image Caption Generator*.

Keywords
Convolutional Neural Network, Recurrent Neural Network

[1] *Department of Industrial Engineering, University of Illinois at Urbana Champaign, Champaign, IL*
[2] *Department of Statistics, University of Illinois at Urbana Champaign, Champaign, IL*

Contents

1 Introduction 1
2 Model architectures 1
3 Training Methods 1
4 Experimental Results 2
5 Discussion 3

1. Introduction

In the area of artificial intelligence, automatically describing the content on an image using properly formed English sentences is a challenging task. Leveraging the advances in recognition of objects allows us to drive natural language generation systems, but the current approaches have limited ability in their expressivity. Most closely, neural networks are used to co-embed images and sentences together, but they didn't attempt to generate novel descriptions. For this paper, the authors combined deep convolutional nets for image classification with recurrent networks for sequence modeling to create a single end-to-end network that generates descriptions of images. They take the image I as input and produced a target sequence of words $S = \{S_1, S_2, \ldots\}$ by directly maximizing the likelihood $p(S|I)$. They used a deep convolution neural network as an encoder function to produce a rich representation of the input image by embedding it to a fixed-length vector. This embedding vector will be used as the initial hidden state of a decoder recurrent network that will be used to generated the target sentence. They present an end-to-end system for this sentence caption generation problem. Their neural network is trainable using stochastic gradient descent and also combines the current state-of-art sub-networks for image and language models. These sub-models could be pretrained on larger datasets and take advantage of additional data. Finally, through experiment results, they show their method could perform significantly better compared with the current state-of-art approaches.

2. Model architectures

The goal is to directly maximize the probability of the correct description given the image by:

$$\theta^* = \arg\max_{\theta} \sum_{(I^i,S^i)\in D} \log p(S^i|I^i,\theta) \tag{1}$$

where (I^i, S^i) is *image-caption* pair, D is training dataset and θ is the parameter for our model. Applying the chain rule to model the joint probability $\log p(S|I,\theta)$ over S, we could get

$$\log p(S|I,\theta) = \sum_{t=0}^{N} \log p(S_t|I,\theta,S_0,\ldots,S_{t-1}), \tag{2}$$

where S_t is the t-th word in caption S. So the authors in [2], model the conditional probability $p(S_t|I,\theta,S_0,\ldots,S_{t-1})$ using the LSTM, which is a special form of recurrent neural network. To be more specific, at the time-step $t-1$, treat the hidden state h_{t-1} as a nonlinear representation of $I,\theta,S_0,\ldots,S_{t-2}$ and given the word S_{t-1}, then calculate $h_{t-1} = f(h_{t-2}, S_{t-1})$. Finally, model $p(S_t|I,\theta,S_0,\ldots,S_{t-1})$ using $p_t = \text{Softmax}(h_{t-1})$. The p_t is the conditional probability distribution over the whole dictionary, which suggests the word to generate at the time-step t.
One more thing need to be specifically addressed here is that authors in [2] used Convolution Neural Network to initialize S_0.

3. Training Methods

We firstly provide a brief overview on data preprocessing procedures and then discuss how we trained the model.
Data Preprocessing We used *MSCOCO 2014* datasets to train and test our model. MSCOCO 2014 dataset contains 82783 training images and 40504 testing images. Each image come with 5 human captions. In the validation set, there are 128 images have 6 human captions and 3 images have 7 human captions, but this shouldnt affect our result much.

In order to load this dataset, we wrote our own dataset class `CoCoDataset()` in the `data_loader.py`. We also customized a dataLoader to read a tuple of (images, captions, caption lengths) in batch. Since human captions have various lengths, so we padded all the captions to the maximal length in each batch. The caption lengths variables are used to record the true length of a caption, which is beneficial when computing losses. We also wrote a `coco_batch` function to create mini-batch tensors from a list of tuples. For data augmentation, we used a random crop with the size of (224, 224), random horizontal flip and normalization. Finally, we convert all image to RGB format, so we always have 3 channels. In order to accelerate our training speed, we also wrote a `resize.py` to resize and save all the training image to the size of (256, 256, 3).We use 256 because the image size needs to be greater than 224 for the random cropping.

The actual training can be divided into two stages, using the CNN to code image and using the LSTM to generate captions. Also the loss is calculate after we generate the caption for each training (I,S) pair. Details are described below.

Image Encoding

`resnet152` pre-trained on the ImageNet is used a image encoder, where we only change and trained the last fully connected layer to accommodate our needs that pass the image embedding to as the input for the LSTM at the time-step 0. In later experiments, we also attempted to fine tune the last block of the `resnet152` to the how this affects the final model performance.

Caption Generation Firstly, we one-hot encode all captions to vectors to desired fixed size, which was done by the file `generate_vocab_dict.py`. To be specific, We tokenized all the captions and calculate the word frequency. Then set a threshold on frequency and merge those less-frequent words to the `<<unknown>>` token. We also introduced `<<start>>`, `<<end>>` and `<<padding>>`tokens. Finally we derived a vocabulary, `vocab`, of 9957 different tokens.

Second, to train the LSTM neural nets, we fed each one-hot-encoded word $S_t, t = 1,2,...,N$ vector to a word-embedding layer, which produces a word embedding vector. Then we sent this word embedding vector to the LSTM cell to predict S_{t+1}. This procedure can be formally described below,

$$\begin{aligned} x_{-1} &= \text{CNN}(I) \\ x_t &= W_e S_t \\ h_t &= \text{LSTM}(x_t) \qquad t \in \{0,1,2,...,N\} \\ p_{t+1} &= \text{Softmax}(h_t) \end{aligned} \tag{3}$$

The *Class* `Decoder` in the file `model_V2_dropout0.py` implements aforementioned procedures. In each batch, since we padded all the captions, so we stripped the `<<padding>>` before we fed captions to the LSTM cells.

Loss and Metric. The loss is calculate as

$$L = \sum_{(I^i,S^i)\in D} L(I^i,S^i) = \sum_{(I^i,S^i)\in D} \sum_{t=1}^{N^i} -\log p_t(S_t^i). \tag{4}$$

We use the training loss as a criteria to terminate the training process, that is when there is no significant improvement on the training loss we stop the training.
We mainly use BLEU-4 scores, which are calculated on the validation datasets, as the criteria to evaluate the quality of the generated captions, where the captions are generated based on the maximal probability rule. To be more specific, $\text{Word}_t = \text{Vocab}[\arg\max(p_t)]$, where the `Vocab` is generated from the one-hot encoded captions.

4. Experimental Results

We did following groups of experiments.

- Use various number of the LSTM layers to see how the complexity of our decoder affects the training loss and BLEU-4 score.
- Add dropout layer at the end of LSTM layers and use different dropout rate to see how training loss and the BLEU-4 score are affected.
- Fine tune the `resnet152` to see how much we can further improve the training loss and BLEU-4 score.
- Compare the model we implemented here and the advanced version proposed by [3] to conceptually see how attention affect the performance.

Choice of Hyperparameters
We used `resnet152` with fine tuned last fully connected layers as our image encoder throughout the most of the experiments, only one exception will be mentioned later.

- Using 1, 3, 5 and 10 layers of LSTM cell(s) (without dropout)
- Using 0.2, 0.5 dropout rate in the decoder with 3 layers of LSTM cells
- Comparing the locked dropout with the normal dropout in the decoder with 3 layers of LSTM cells
- no dropout for both the model we implemented (`resnet152` + 3-layered-LSTM) and the one proposed in [3]

Lastly. We compare the encoder with the only fine-tuned last fully connected layer with one with the fine-tuned last block and the last fully connected layer.
Traning and Validaiton Results.
First, we present the loss curve for the model with/without attention and corresponding BLEU-4 score gain the basic ideas. The rest of comparisons are similar, hence will be summarized in tables for simplicity.

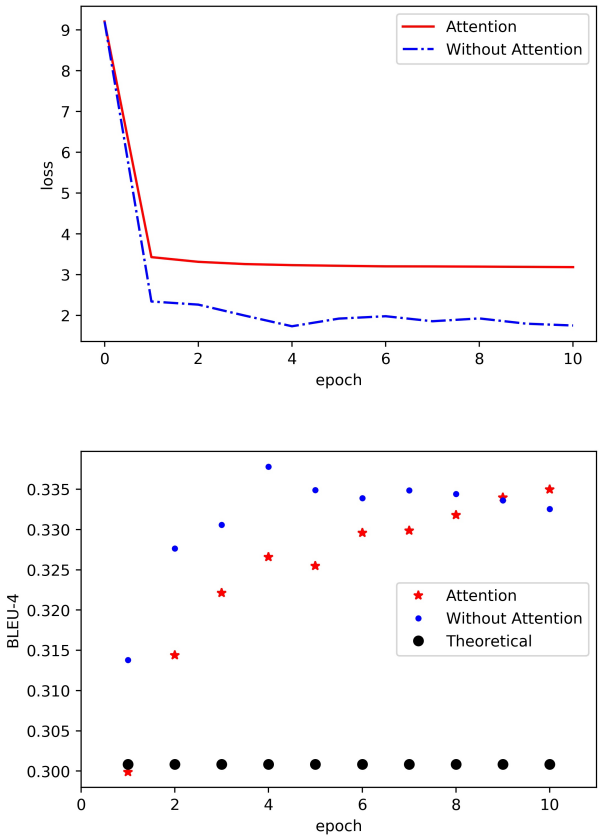

Figure 1. The loss curve and the average BLEU-4 score for two models.

The Table 1 tries to show the impact of LSTM layers. We can see the 3 Layers gives the best BLEU-4 score.

number of LSTM layers	BlEU-4 Score	Training Loss
1	0.33103	1.8058
3	0.33109	1.9128
5	0.32192	2.1484
10	0.247716	4.5569

Table 1. Compare how number of LSTM layers affects the training loss and BLUE-4 scores.

The Table 2 shows that only use dropout at the image encoder will give the best BLEU-4 score.

Apply	Dropout Rate	BlEU-4 Score	Training Loss
Encoder	0.2	0.33117	1.9500
Encoder	0.5	0.33692	1.7687
Decoder	0.2	0.32837	3.4942
Decoder	0.5	0.32675	5.5146
Both	0.2	0.33550	3.3468
Both	0.5	0.32923	5.4706

Table 2. Compare how dropout rate affects the training loss and BLUE-4 scores.

The Table 3 shows that fine tuning the last block and the last fully connected layer indeed improves the model a lot. Of course, it takes more time to train.

Fine-tune	BlEU-4 Score	Training Loss
Last fc layer	0.33117	1.9500
Last fc layer + Last Block	0.35371	1.7059

Table 3. Compare how fine-tune affects the training loss and BLUE-4 scores.

Inference: Caption Generation. We do observe that generally the model with attention mechanism can generate captions making more sense, but the model we trained here seems to have better generalization ability. For example, we fed the following figure that is not from the validation set to both models. The generated caption for our model is `<<start>>` **two dogs are playing with a frisbee in the grass.** `<<end>>` while the attention model can only generate `<<start>>` **a dog with a frisbee in its mouth.**`<<end>>`. It shows that our model can identify two dogs while the model with attention can only identify one.

Computation Time. The experiments are done separately on the AWS and Blue Waters. The total time spent on the Blue waters are approximately 450 hours while we spent around 230 hours on the AWS (Approximately 230 * 2.5=575 computation hours on Blue Waters).

Supplements. All the supporting code can be found at the Github repository.

5. Discussion

For this final project, our group implemented the Neural Image Caption(NIC) generator model, one end-to-end neural network that could automatically generate a reasonable description for the input image using plain English. This neural network model combines a convolution neural network(CNN) with a recurrent neural network(RNN). The CNN could encode the image into a compact representation and the RNN

as a decoder could generate a meaningful sequence. Experiments are based on the *MSCOCO 2014* dataset. Reasonable captions generated from our trained model shows that the NIC model is robust in qualitative results. Also different choices of hyper-parameters are tested, where the best test BLEU-4 score is 0.35371 comparing to the theoretical BLEU-4 0.301. For the future, we believe better encoding and decoding neural network could be further developed along with refined image representation, RNN architecture and improved optimization algorithms. Therefore, such kind of Neural Image Caption model is expected to perform better than human.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[3] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057, Lille, France, 07–09 Jul 2015. PMLR.