Pay Attention to Your Surroundings: An Exploration of Different Decoding Techniques for Image Captioning

Aneek Barua
SUNY Stony Brook University

Abstract

Being able to verbally describe an image is an imperative issue in the fields of both computer vision and natural language processing. This report aims to clear some debates between two different encoder-decoder systems, specifically LSTM-cell based decoders and systems where an attention Layer is placed between the encoder and decoder. There's an additional analysis on using a ResNet and Transformer pairing for encoding-decoding, and the efficacy of using transformers as opposed to RNNs with LSTM like GRU. The datasets used in this paper include MSCOCO's 2014 image and annotation dataset, as well as Flickr8k, a smaller version of Flickr30k, which is a fairly large dataset of images paired with 5 reference sentences (human annotated). The code for this report is adapted from Google's *Show and Tell* CV paper from 2015 (originally written in TensorFlow), as well as adapted from Assignment 3 of this course (Transformers and DistilBert).

1 Introduction

Generating accurate captions for images is a pretty challenging task that appears often in publications in both computer vision and natural language processing conferences. Many papers aim to tackle the issue in different structures (i.e. segmented models, joint model structure for encoding-decoding), and many yield accurate and understandable results. A problem such as image caption generation comes with many pitfalls, such as prioritizing certain aspects of an image, or conveying complex meaning behind the juxtaposition and overall meaning behind a picture. One such example would be a model identifying a blanket, grass, and food, but having difficulty correlating these with a picnic if in the sun, or a bedroom with abnormal decor. This contrasts heavily with ongoing computer vision research, which tackles much more well-studied tasks like object recognition or image classification. This lack of familiarity carries over the field of natural language processing, considering the cooperation needed between a language model that understands and can express semantic knowledge and a vision model that can accurately capture important parts of an image that it deems important for said language model.

Many common solutions to this problem include a fairly intuitive structure (courtesy of Vinyals et al. from *Show and Tell*) which starts with a Convolutional Neural Network (CNN) for decoding the image to extract key features and convert it to a vector representation in order to semantically describe the image. The second step in the most common implementation of this type of problem is a RNN (usually GRU or another refactored large language model/RNN built from scratch) which is responsible for decoding these vector representations to accurately reconstruct a sentence that holds the semantic meaning accurately. The joint of these two models serves the purpose of optimizing the following function (from Vinyals et al.):

$$\Theta* = \underset{\Theta}{argmax} \sum_{(I,S)} \log p(S|I;\Theta)$$

The scale of the models used (specifically the decoder) depends largely on the dataset size and computational power the authors' of papers have access to. While this usually achieves accurate results, this is a fairly dated method for language decoding, ever since the widely adopted usage of transformers.

Within the field of machine translation, Transformer based models have started replacing LSTM models in almost every sector. This is due to the speed at which transformers can train, as well as the large advantages it give in sequence-to-sequence tasks. In a study done by Karita et al. (*A Comparative Study on Transformer vs RNN in Speech Applications*), although their analysis was on tasks tangentially related to image caption generation, the investigations done in this paper reflect across almost all natural language processing tasks and model fields. Karita et al. specifically mentions the transformers' reliance on multi-head and self-attention for the encoder and decoder portion, and the significantly faster training time on transformer-based models. Generally, transformer-models offer lower memory usage, and parallelism benefits when compared to RNNs.

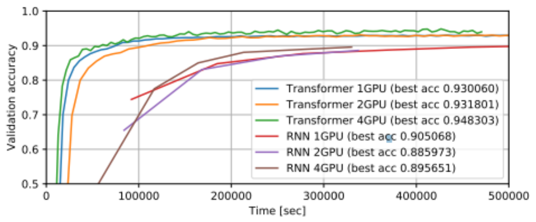
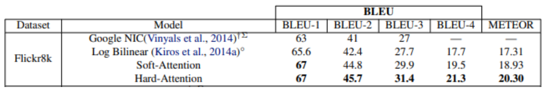


Figure 1: *ASR Training Curve with LibriSpeech from Karita et al.*

Given the multitude of methods available to tackle image caption generation, an interesting question shows up regarding which configuration is best for what cases. While it may sound like transformer-based models are much better than RNN models, there could be a few caveats only visible when the models are pitted against each other. For instance, in the computer vision side of this problem, transformers are having a much slower uptick in replacing neural networks. This is mostly because they cannot pick up on the image-specific biases or pixel-related accentuations that CNNs can. In a paper by Bai et al. (*Are Transformers More Robust Than CNNs?*), the authors explore the usage cases of transformers, and specifically the other studies on transformers vs CNNs (and how disadvantageous the playing field is for CNNs in visual recognition and other tasks). They go more in depth about the training practices associated with transformers (which can easily be translated to CNNs), and the unfairness of experimental settings from papers that claim transformers will replace CNNs soon. A highlighted snipped from their abstract claims that ”stronger generalization is largely benefitted by the Transformer's self-attention-like architectures per se, rather than by other training setups)”. This, along with other situational specifics that help either model, seems to insinuate that the models are not direct competitors, but rather replacements for each other in specific cases.

We specifically aimed to clear the playing field by using the same dataset on all model types. We started by implementing the baseline algorithm, the one highlighted in *Show and Tell* by Vinyals et al. Their model highlights a Vision Deep CNN which reads the image to encode into a vector. From there, it is directly linked to a Language Generating RNN, which handles the decoding task. By loosely following their code (implemented in TensorFlow), understanding it, and translating it to our specific usage case (Flickr8k), we were able to reconstruct the essence of their code in a different codebase. This included a simple CNN, in which case we used a pretrained model, VGG-16. VGG-16 is a CNN model proposed by Simonyan and Zisserman, which has been a go to for visual classification and detection for years now. We took a pretrained fork of that model and removed the last layer. We then added a final layer, which downsampled the output from our pretrained model to 300 features. This was fed into our decoder model, an RNN with LSTM. This model consisted of a hidden layer, an LSTM layer, and a fully connected final layer to 1000 features.

The second model to test against was the intermediary attention model. The structure of this followed the same as in *Show and Tell*, but with an extra attention network in between. This attention network was a series of 3 linear layers, followed by a ReLU and a softmax, for soft attention. This was passed into an RNN as well, for the decoding phase. The implementation of this model was roughly based on *Show, Attend, and Tell*, by Xu et al. Xu et al., explored both Deterministic ”Soft” and Stochastic ”Hard”. Xu et al. included BLEU scores, metrics for evaluting machine translated text, for their results:



		BLEU				
Dataset	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Flickr8k	Google NIC(Vinyals et al., 2014)$^{\dagger\Sigma}$	63	41	27	—	—
	Log Bilinear (Kiros et al., 2014a)$^{\circ}$	65.6	42.4	27.7	17.7	17.31
	Soft-Attention	**67**	44.8	29.9	19.5	18.93
	Hard-Attention	**67**	**45.7**	**31.4**	**21.3**	**20.30**

Figure 2: *BLEU Scores for configurations (Xu et al.)*

The third, and final model, was a transformer model. This model had a much bigger pre-processing time and list, but nevertheless ran quicker. We started by filtering the data as we did for all models, but we also went through and used a ResNet Feature Extraction method to get our images as inputs. To do this, we pulled a ResNet18 model and pulled layer 4 to encode all our image files and data. From here, we created a Position Embedding Encoder, which consisted of a dropout and embedding layer. This went in conjunction with our Decoder, which was the full tranformer. Our decoder had a decoding layer, positional embeddings within the class, and a trasnformer decoder. With these, and the embedding layer and fully connected layer for output, we calculated masks for each input, padded said masks, and fitted them to the model to train. This model trained much quicker than the others, but was far more complex to code due to the intricacies of formatting input for a transformer for decoding.

Overall, from our experience, it seems both model types are equally viable, and the real difference lied in training time. In fact, some of our models have still not finished the benchmark 5 epochs, while the transformer model has completed the epochs.

2 Your Task

By creating all the specific models and training them, we were able to see the average loss values for each configuration, as well as a few example images pulled from their results.

3 CNN-RNN

The idea behind this model, in the simplest form, is to pair a vision-tuned network (CNN) to a language model (RNN or GRU) to encode important information from the image to decode against the corpus after completion.

Our CNN was a fork of the pretrained VGG-16 model, paired with a final fully connected layer to lower dimensionality of our output. This was paired with our RNN, which was a basic single layer RNN with LSTM and fully connected layer. This was trained (still is training interrupted at 4 epochs) for 5 epochs, ultimately yielding a loss value of 0.6.

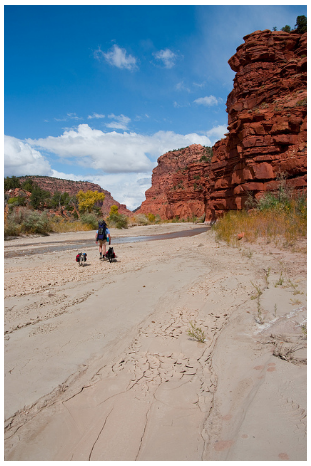


Figure 3: *Predicted Caption: A beach with two people in the water.*

4 CNN-RNN with Attention

This model is very similar to the last one, but with an attention layer between the encoder and decoder. The training on this was just about the same, but the performance of the loss was better. We used the same CNN pretrained model and RNN, with an Attention Network between which consisted of 3 linear layers and a softmax.

This model fared far better than the previous model, most likely due to the attention component on the encoded vectors (highlighting the important parts of the image). The training loss on this was about 0.13. Training time, on the other hand, was still just as slow (only made 4 epochs in time).

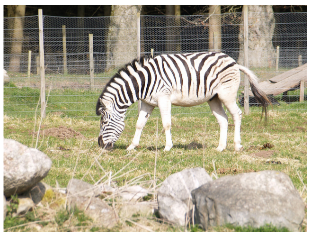


Figure 4: *Predicted Caption: A zebra that is standing in the grass.*

5 Tranformer Encoder-Decoder

This model consisted of a pre-encoder, which was the ResNet18 layer 4, which was ran on our images beforehand and saved as a serialized pickle file.

This acted as our encoder, while the decoder included a Transformer Decoder and a Positional

Embedding Network. By utilizing both, this network fared much better with the dataset in terms of time and accuracy. The training loss of this model was roundabout 2.5 by the end of our 5 epochs, but this was not normalized correctly due to the padded_mask factor taken into account when computing loss.

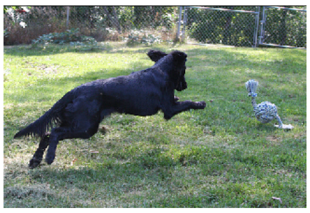


Figure 5: *Predicted Caption: black dog is running through the grass .*

6 Experimental Setup

State the purpose of your evaluation. How should one evaluate a system for your task. What are the questions to ask?

6.1 Dataset Details

6.2 Evaluation Measures

6.3 Model Implementation Details

6.4 Implementation Details

For all of the models here, we used PyTorch in conjunction with PIL Image for Image reading and rendering. We also used pandas for cleaning the dataset up (reorganizing the labels and data the way we see fit). Most of the code utilized in these experiments are based on the papers referenced, and the Flickr8k Kaggle Notebook directory (used as reference but not as copy-paste).

For the RNN based models, we used a learning rate of 0.001 arbitrarily. We tried a lower and higher learning rate, but a lower learning rate seemed to not converge within 2 epochs, while a higher training rate caused gradient explosion in some cases. We stuck with this learning rate from empirical experience. As for the optimizers, we used Adam in both cases (or rather all 3). Adam is generally regarded as the better optimization algorithm, and has the faster computing time. The other options we looked into (like SGD and RMSprop) required far more parameters and had minimal benefit or even detriment in some cases.

For our transformer model, we tried to work with scheduling and altering learning rate over time. If it works perfectly, it will be included in the code demo, but we ultimately decided to stick with a learning rate of 0.0001. Another parameter we explored in the transformer model was the train-val split, specifically comparing a 90-10 to an 80-20. We started off with a 90-10 and had minimal issues, but saw the 80-20 was also sufficient, and gave us more access to different test cases to validate against.

7 Results

Overall, while we did not have the time to run metrics like BLEU or F1-score, we could tell that the transformer easily deserves its spot as the one to dethrone the RNN. Many hot language models have started using transformers in lieu of RNNs (BERT vs. GRU), and the training time difference is significant when computing on a low-end machine.

8 Code

The code is located at a GitHub Repository by AbsolutUnit called NLPProject. The link is here: https://github.com/AbsolutUnit/NLPProject.

1. Flickr8K
2. *Show, Attend and Tell*
3. *Show and Tell*
4. Show, Attend, and Tell Code Demo
5. Software Requirements: Python in Google Colab, PyTorch, GPU runtime.

9 Conclusions

Overall, caption generation from images is a fairly complex task, but the many ways to go about it have their tradeoffs. The RNN-CNN implementation of encoder-decoder is fairly simple in code, but far more strenuous on one's system and memory, as compared to the transformer implementation, which is much harder to conceptually grasp and write, but runs much faster. Another small nitpick I saw on the transformer model is the performance of it in terms of writing sentences gramatically correct, but that may be a code related issue. In general, the transformer does far better for tasks in smaller datasets or quick and easy completed captions, while the RNN, being slow, still does a great job at capturing the semantic meaning and providing more than trivial captions for an image.

10 References

1. *Show and Tell* by Vinyals et al.

2. *Show, Attend, and Tell* by Xu et al.

3. *A Comparative Study on Transformer vs RNN in Speech Applications* by Karita et al.

4. *Are Transformers More Robust Than CNNs?* by Bai et al.