Neural Image Captioning: Fixate and Unroll

Saurabh Gupta
Dept. of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
sag043@ucsd.edu

Caroline Kim
Dept. of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
chk101@eng.ucsd.edu

Ruby Pai
Dept. of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
rjpai@eng.ucsd.edu

Apoorve Dave
Dept. of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
a1dave@eng.ucsd.edu

Abstract

Humans can describe a complex scene in a few words without much effort but this task has been significantly difficult for machines. A description should not only capture the objects contained in an image, but it must also express the relationships between objects and the activities they are involved in. Inspired by the recent work in computer vision and machine translation, we explore an "attention" based framework that learns to "attend" to specific regions of an image while generating a textual description, much in the same way human vision focuses when we perceive the visual world. In this work, we replicate the existing attention mechanism based captioning model and analyze the performance impact of modifications to the current attention architecture on Flickr8k dataset. Furthermore, we also train a probabilistic "non-attentive" model for caption generation, which achieves a METEOR score of ##.# on Flickr8k dataset.

1 Introduction

LSTM neural networks have been successfully used for language translation, with the model of one LSTM network acting as an "encoder" while a second LSTM network decodes the representation found by the first into a sentence in the target language. This paradigm inspired recent research in image captioning, in which a convolutional neural network is used to encode an image and a recurrent neural network is used to decode image codes to produce a caption by generating one word at every time step. Typically a LSTM network is used for the decoder, but BRNN have been investigated as well. See [1] for an extensive list of references to this body of work.

Vinyals et al [2] is a straightforward recent application of this paradigm, with a difference from previous papers that image features are only input to the LSTM at the first time step, which [2] says was found empirically to outperform providing image features as inputs to the LSTM decoder at every time step.

Recently the concept of adding attention or localization to neural image captioning has been explored in Xu et al [3] and Johnson and Kaparthy et al [1]. In the architecture of [3], the network learns which part of the image to focus on in generating the next word of the caption; the model focuses on a different region of the input image with every generated word. The attention model used necessitates taking features from lower CNN levels, exploiting information available at this lower level.

On the other hand, [1] proposes "dense captioning" where captions are generated for multiple regions of interest in an image. This is accomplished with the insertion of a localization layer above CNN layer which allows the network to look at arbitrary regions of interest of varying sizes and aspect ratios instead of the fixed grid positions that are accessible using the attention model of [3]. Thus, the authors claim that this approach is more general. For simplicity, [1] follows the approach in [2], where the visual features are passed to the LSTM model once at the first time step.

In this project, we attempt to modify the attention architecture in [3], as described in Section 2. Taking inspiration from [2], we also train a neural and probabilistic model which decodes the image features using a holistic gaze rather than focusing on specific regions at each time step. This allows us to make a comparison between our implementations of [3] and [2].

2 Attention-aided caption generation (Xu et al [3])

2.1 Model Architecture Overview

The architecture of [3] is shown in Fig. 1. VGGNet-19 image features from the final convolutional layer before the max pooling are taken as the input to the caption decoder. The 512 14x14 feature maps this layer of the CNN produces are the 196 feature vectors $a_i \in R^{512}$. These feature vectors are fed to what we can think of as an attention module that computes the context vector $\hat{z}_t$, which encodes features of the regions of the image the LSTM decoder should use to produce the next word in the caption. Inside the attention module, a location distribution over the feature, α, is generated from the feature vectors and the recurrent LSTM hidden state. Then the context vector is produced by either sampling from this distribution or taking the expectation over the distribution. The context vector is the input to the LSTM layer, which also takes recurrent connections from its hidden state and the previous output word. The LSTM hidden state is input to a multilayer perceptron, which computes a posterior distribution over words y_t. Note that "attention-aided" caption generation is our terminology, not used by [3].

2.2 Some Insights

In the discussion that follows, "time", or a step in time, corresponds to a step in producing the output sequence, that is, each time step corresponds to a word in the caption. Image locations correspond to the $L = 196$ positions in the 14×14 feature map of VGGnet-19 [4] before the fully connected layers. (These positions in the feature map correspond to overlapping areas in the 224x224 input image.)

Figure 1 shows the architecture of [3] as interpreted by this report. The attention mechanism consists of two major parts. First, a probability distribution $\{\alpha_{it}\}_{i=1}^{L}$ is generated at each time step t, over the L image locations. Second, this distribution is used to compute the "context" vector of relevant image features that is input to the caption-generating LSTM. "Soft" attention computes a weighted average of the L image feature vectors by taking an expectation over $\{\alpha_{it}\}_{i=1}^{L}$. "Hard" attention samples from $\{\alpha_{it}\}_{i=1}^{L}$ to select a single feature vector. Thus the probability distribution $\{\alpha_{it}\}_{i=1}^{L}$ is a saliency map dictating the image input for each generated word in the caption.

[3] describes the architecture for generating $\{\alpha_{it}\}_{i=1}^{L}$ as a "multilayer perceptron conditioned on the previous [caption-generation LSTM] hidden state h_{t-1}". Upon study of the details in the source code, it is clear that the attention architecture can be interpreted as a simple convolutional neural network with 1x1 sized kernels, and no pooling. Specifically, weights are shared as the 196×512 matrix of feature vectors is transformed into another set of 512 (14×14) feature maps. The previous hidden state from the caption-generation LSTM h_{t-1}, after dimensionality reduction to dimension 512, acts as the biases in this convolutional layer.

After a tanh nonlinearity, a 1x1 filter with a trained bias is used to create what can be interpreted as a single 14×14 feature map (represented in the code as a L-vector), which is the input to a softmax that produces $\{\alpha_{it}\}_{i=1}^{L}$, the probability distribution over the L positions.

So in this interpretation, the attention mechanism that computes the $\{\alpha_{it}\}_{i=1}^{L}$ distribution is a simple two layer convolutional neural net with the parameters in the table below. At each time step, the input

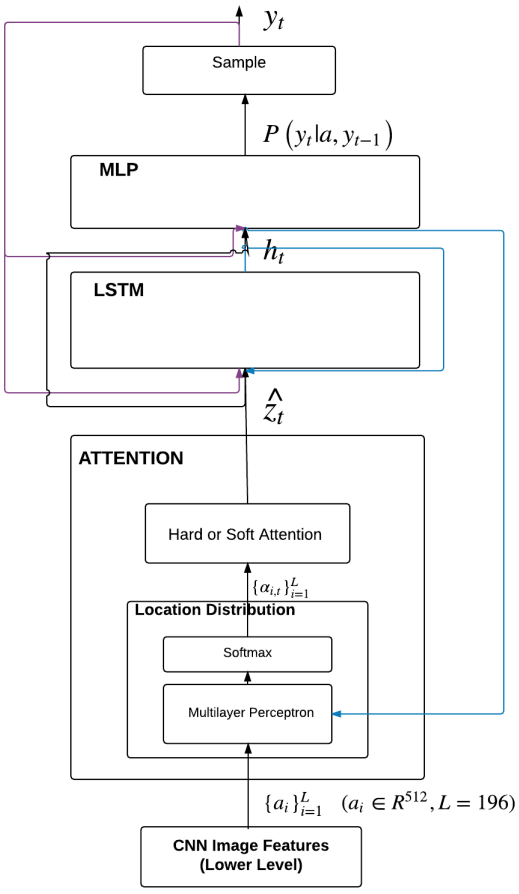

Figure 1: Architecture of Xu et al [3] for an image caption decoder which takes input image features from VGGNet-19. Color denotes recurrent connections (blue: LSTM hidden state, purple: output word, which is fed back as an input after being transformed with a learned embedding matrix).

of VGG lower layer image features is the same, but the bias to the first layer filters changes - it is a fixed projection, or embedding, of the recurrent hidden state of the caption-generation LSTM.

Table 1: Interpretation of attention MLP as a convnet with time-varying bias

Layer	Type	Input	Kernel	Filters	Nonlin.	Pooling	Stride	Output	Param.
1	Conv	14*14*512	1*1	512	tanh	None	1	14*14*512	262656
2	Conv	14*14*512	1*1	1	softmax	None	1	14*14*1	513

2.3 Experiments

In this section, the focus is on experimenting with the generation of $\{\alpha_{it}\}_{i=1}^{L}$, leaving the rest of the overall architecture the same. Once $\{\alpha_{it}\}_{i=1}^{L}$ is generated, it can be used with either soft or hard attention. Since [3]'s results for soft and hard attention are comparable, the experiments are restricted to soft attention both due to time constraints and for simplicity, as soft attention has less training parameters to take into account.

One simple experiment that immediately suggests itself: assessing the impact of attention by effectively removing attention from this framework, by setting $\{\alpha_{it}\}_{i=1}^{L}$ to be fixed at the uniform distribution. Note this only really makes sense for soft attention (with hard attention that would be forcing the model to sample uniformly at random from image locations to generate each word, which could be expected to produce poor results). This is different then comparing [3] with a captioning architecture that does not use attention, such as [2] because of the variation in other aspects between the models, as [3] points out. [In progress]

The main experiment in this section, however, is to replace the $\{\alpha_{it}\}_{i=1}^{L}$ generator with an attention LSTM, with the reasoning that that would allow the model to better exploit the temporal relationship between $\{\alpha_{it}\}_{i=1}^{L}$ at each time step. So far this effort has entailed something on the order of 50+ hours of development time, including understanding all the details of the source code [5], considering how the attention LSTM should be implemented, and debugging. Currently, the modified code is able to build and train the model, but a bug in the code prevents the use of the original beam search mechanism to generate captions, resulting in sample captions of sentences completely comprised of the word "a" (which is what the unmodified architecture appears to do without beam search as well). The modified code is available here –.

In order to follow the existing attention architecture as closely as possible in a first experiment with the attention LSTM, attention LSTM cells simply replace the neurons with the tanh activation, from layer 1 in Table 1. For each LSTM cell, there are four weights total: three weights corresponding to the input, output and forget gates, as well as the weight for the input. While these LSTM cell weights are not directly comparable to weights in a multilayer perceptron [6], having four times as many weights still takes up four times as much memory. Due to memory constraints, it was necessary to project the dimension 512 image feature vectors to a lower dimension, currently 64. [Need to finish debugging so that the impact of this can be evaluated in results]

2.4 Footnote about experiments

We had originally proposed inserting the localization layer of [1] before the attention module. CNN output image features would be input to the localization layer to produce feature vectors corresponding to regions of varying sizes, that could then be input to computing a location distribution, leaving the rest of the model unaltered. Our reasoning was that allowing words to be compute from regions of varying sizes inside the image would allow better captions to be generated. However, upon further reflection we realized that this mechanism is encompassed in the soft attention model (by producing the appropriate distribution, which acts as weights over the image feature vectors in the expectation).

2.5 Differentiability and Cost function

[3] contains two different attention models. Common to both, a distribution over different points in the CNN feature map is generated, where each point corresponds to different, overlapping regions of the original image. In the stochastic, "hard" attention model, one point in the feature map is sampled

from the distribution and its feature vector is used as the location input to the LSTM decoder. In the deterministic soft attention model, an expectation is taken over the distribution to produce a blending of feature vectors. The hard attention model is not differentiable and is optimized via the REINFORCE algorithm [7]. The soft attention model is differentiable and can be trained using backprop.

As in [3], the objective functions we will minimize in our experiments are the following,

$$L_{hard} = \sum p(s|a) \log p(y|s,a) \leq \log p(y|a)$$

$$L_{soft} = -\log P(y|x) + \lambda \sum_{i}^{L} (1 - \sum_{t}^{C} \alpha_{ti})^2$$

where s is location variable(where the model focus attention on), C is the length of caption, α_{ti} is weight of each annotation vector (from CNN) at time step t, y is sequence of words(caption).

3 Non-Attentive Probabilistic Caption Generation (NAPGen)

3.1 Model Architecture

This model is based on [2] and [8], in which an end-to-end differentiable system combines state-of-the-art networks for vision and language models. A deep CNN (GoogLeNet v3)[9] is used to produce a rich representation of the input image by embedding it to a fixed-length vector which is fed to a sequence model i.e. LSTM that finally decodes the image features to the desired output sentence. Fig. 2 shows this architecture in which an LSTM model is combined with a CNN image embedder.

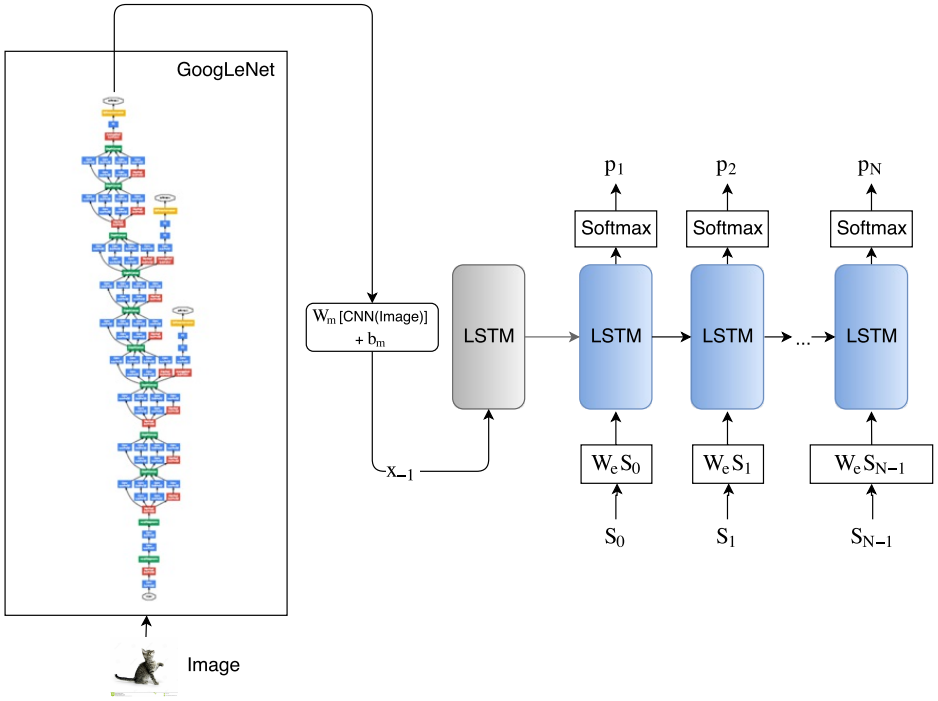

Figure 2: Diagram of the NAPGen model. The LSTM takes a word, context from previous time step and defines a probability distribution p_t over the next word in the sentence. The LSTM is conditioned on the embedded image vector at the first time step.

To represent image features, the image embedding W_m takes a 1000 dimensional vector from the last layer of GoogLeNet and transforms it into a h dimensional vector which is then fed to the LSTM. The matrix W_m has dimensions $h \times 1000$, where h is the size of embedding space(h ranges from 128 to 512 in our experiments).

$$x_{-1} = W_m CNN(I) + b_m \tag{1}$$

At training time, (S, I) is a training example pair where I is the input image and S represents the ground truth caption sequence where S_i is a n-dimensional vector (encoded in a 1-of-k representation). The word embedding matrix W_e transforms each S_i into a h-dimensional vector that is fed to the LSTM at each time step.

3.2 Cost function

This model tries to maximize the probability of the correct description given the image by using the following formulation:

$$\theta^* = \arg\max_{\theta} \sum_{(I,S)} logP(S|I) \tag{2}$$

where θ are the parameters of our model, I is an image and S is the sequence of words. On applying chain rule:

$$\log p(S|I) = \sum_{t=0}^{N} P(S_t|I, S_0, ..., S_{t-1}) \tag{3}$$

The probability $\sum_{t=0}^{N} logP(S_t|I, S_0, ..., S_{t-1})$ is modeled using LSTM as the words upto time $t-1$ can be conditioned upon by the hidden state h_{t-1} of LSTM (Further optimization details are given in Section 3.4).

3.3 LSTM-based Sentence Generator

The LSTM cell architecture described in [10] is employed in this model. The image embedding x_{-1} is fed to the LSTM at first time step only. Embedded word vectors $W_e S_i$ are fed to the LSTM at other time steps. The output vector of LSTM cells is used to feed to a Softmax, which produces a probability distribution p_t over all words in the vocabulary of size n.

3.4 Training

The loss function, which is the sum of negative log likelihood of the correct word at each time step, can be written as:

$$L(I, S) = -\sum_{t=1}^{N} \log p(S_t) \tag{4}$$

The above loss is minimized w.r.t all the parameters of the LSTM, image embedding W_m and word embedding W_e. We trained all sets of weights using Adam with a fixed learning rate of 0.001 on the Flickr8k dataset. We used 256 dimensions for embeddings and LSTM memory. Dropout was used to prevent overfitting.
For inference, we sample the first word according to p_1, then provide the corresponding word embedding as input to third LSTM cell and sample p_2 and continue this until we sample the special end token or reach maximum length of the sequence.

3.5 Experiments

We have trained a baseline model using the parameters and procedure described in above section. A live web demo of the model is available here). We haven't evaluated the scores of this model yet.

The most important set of experiments include feeding the image embedding to LSTMs cells at every time step. Although this may lead to over-fitting, our hypothesis is that this might work as an attention mechanism since the image information is being passed to LSTM at every time step. However, for this to work, we also need to extract the image features from a lower convolutional layer as the spatial information gets lost after a fully connected layer.

4 Metrics

BLEU scores and METEOR scores, though imperfect, are commonly used metrics for evaluating natural language output with respect to a set of references, and are the metrics used in [3] and [2]. We thus evaluate our results with these metrics, comparing with the results in [3] and our results from running unaltered model of [5] as a baseline (see Section 6).

5 Software

The source code for [3] is available at [5] which is written in Python and uses Theano. As stated in Section 2, we used [5] for our experiments on attention-aided caption generation. We referred to [8] and [2], for the implementation of NAPGen in Lasagne.

6 Datasets

Experiments in [3] were performed on the widely used Flickr8k, Flickr30k, and COCO datasets. Due to computational time constraints, we focused on the Flickr8k dataset [11], which contains 8,000 images, each of which comes with five reference captions. This dataset can be obtained by submitting a request at the website [12].

We use the unaltered model of [3] as our reference benchmark. As [3] points out, various factors (for example, the use of an ensemble of models versus a single model) make it difficult to compare results directly across papers, so it makes sense to compare our results with the paper whose architecture from which we are starting out.

[11] is composed of 8000 images and 5 captions for each image. We use 6000, 1000, and 1000 split for train, validation, and test respectively.

7 Effects of changes in Model Parameters and Hyperparameters

Not all the architecture details and training parameters used to produce the results in [3] were published; instead the source code [5] was made available with some defaults. However, when we trained with the default values, they did not produce the published scores. Therefore, an effort was made to find the model architecture parameter and hyperparameter combinations that would produce results comparable to those published. [Results Pending]

This section summarises the variation in BLEU and METEOR scores with changes in various parameters with respect to a base model.

Table 2: Baseline Scores

Model	B-1	B-2	B-3	B-4	METEOR
Base	0.52	0.289	0.162	0.096	0.162

7.1 Architectural Changes

7.1.1 Hidden Layers for Attention Weights

`n_layers_att`: Normalized weights for various attention vectors, α_i as mentioned in [3], are calculated using a multilayer perceptron with these many hidden layers.

7.1.2 Layers used to compute Logit

`n_layers_out`: Number of hidden layers in multilayer perceptron, which is used to compute probabilities of output word vector $\boldsymbol{y}$.

7.1.3 Number of LSTM Layers

`n_layers_lstm`: As the title suggests, it is the number of LSTM layers in the model.

7.1.4 Number of MLP Layers to Initialize LSTM

`n_layers_init`: The initial memory state $\boldsymbol{c_0}$ and hidden state $\boldsymbol{h_0}$ are predicted by MLP's with number of hidden layers defined by this parameter.

7.1.5 Dropout in MLP Layers

Flag to toggle dropout option on all of the MLP Layers of the model.

7.1.6 Dropout in LSTM gates

Flag to toggle dropout option on LSTM gates of the model.

7.1.7 Word Embedding Dimensionality

This gives the dimensionality of embedding $\boldsymbol{Ey_t}$ obtained from output word vector $\boldsymbol{y_t}$.

7.2 Other Hyperparameters

7.2.1 Optimizers

This section highlights the variation in scores with various optimizers: SGD, Rmsprop, Adadelta and Adam.

Table 3: Comparison with Adadelta

Model	B-1	B-2	B-3	B-4	METEOR
Base(RMSProp)	0.520	0.289	0.162	0.096	0.162
Adadelta	0.437	0.236	0.126	0.072	0.153

7.2.2 Decay Coefficient

L2 regularization coefficient for weight decay.

7.2.3 Learning Rate for SGD

Learning rate parameter, works only with SGD.

7.2.4 Selector

Flag to use doubly stochastic attention model. As suggested by [3], keeping this flag on helps in improving BLEU scores and provides richer and more descriptive captions.

7.2.5 Input Batch-Size

7.2.6 Early Stopping Criterion

[5] monitors negative log likelihood of the objective function on validation set to determine when to early stop. [3] mentions use of BLEU score for early stopping instead for the published result. To try to achieve baseline closer to the published data, we also trained with early stop on Bleu-1, Bleu-2(tentative), Bleu-3, and Bleu-4.

Table 4: Early Stopping Result

Flicker 8k Soft Attention	Epochs	Bleu-1	Bleu-2	Bleu-3	Bleu-4	METEOR	CIDEr	ROUGE_L
Xu paper	NA	66.7	43.4	28.8	19.1	18.49	NA	NA
Log-likelihood	80?	52.1	30.0	16.2	9.6	16.2	20.7	43.5
Bleu-1	46	60.1	30.1	13.7	6.1	12.9	9.0	44.6
Bleu-2								
Bleu-3	69	58.2	32.7	18.9	10.8	15.4	21.1	44.3
Bleu-4	71	58.6	32.6	18.5	10.9	15.3	20.6	44.4

8 Results

8.1 Using Attention Mechanism

8.2 Using Non-Attentive Probabilistic model (NAPGen)

9 Discussion and Conclusions

10 Contributions

Saurabh and Caroline worked on data pre-processing to generate the input as required by [5], while Ruby wrote the bulk of both drafts of the proposal, based on consensus from team meetings. Ruby initially discovered through reading [5] that some key training parameters had defaults in the published source code that differed from what was published in [3]. Subsequently, Apoorve, Caroline and Saurabh worked on hyper-parameters search to find the most optimal parameters for Flickr8k dataset. A lot of time was spent on this since these hyper-parameters are not mentioned in [5]. Caroline worked on modifying the early stopping criterion based on Bleu-n scores to achieve higher accuracy. Saurabh and Apoorve worked on Non-Attentive Probabilistic Caption Generation model (NAPGen) and Web GUI. Ruby developed the attention LSTM addition to the architecture of Xu et al [5], helped by Caroline in understanding parts of the original code, and wrote Section 2 with contributions from Caroline.

References

[1] Justin Johnson, Andrej Karpathy, and Fei-Fei Li. Densecap: Fully convolutional localization networks for dense captioning. *arXiv preprint arXiv:1511.07571*, 2015.

[2] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014.

[3] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[5] https://github.com/kelvinxu/arctic-captions. Accessed 2-18-2016.

[6] Conversation with Gary Cottrell.

[7] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning, 8(3-4):229256*, 1992.

[8] https://github.com/karpathy/neuraltalk. Accessed 2-18-2016.

[9] C. Szegedy, W. Liu, and Y. Jia. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[10] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] P. Young M. Hodosh and J. Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. *Journal of Artificial Intelligence Research*, 2013.

[12] https://illinois.edu/fb/sec/1713398. Accessed 2-18-2016.