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

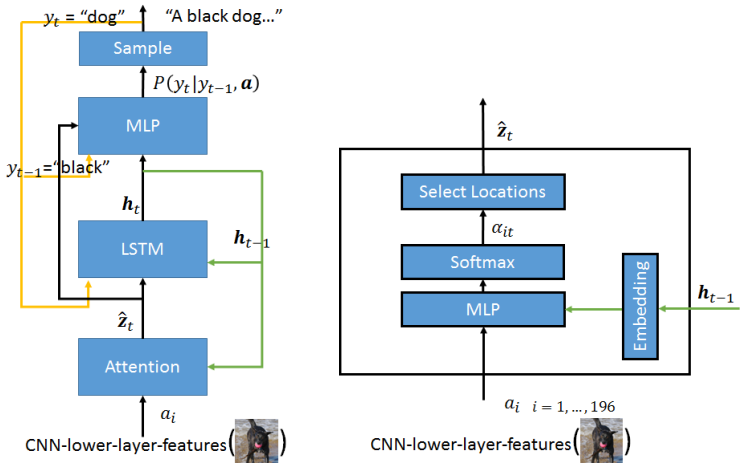
Humans can describe a complex scene in a few words without much effort but this task has been significantly difficult for machines. A description should not only capture the objects contained in an image, but it must also express the relationships between objects and the activities they are involved in. Inspired by the recent work in computer vision and machine translation, we explore two "attention" based frameworks that learn 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. First, we take a recently published attention mechanism based captioning model [1] and analyze the performance impact of modifying different model parameter and hyperparameters, as well as of removing attention. We also train a probabilistic "non-attentive" model for caption generation, inspired by Google's NIC [2], which we call NAPgen. NAPgen serves as a base model for incorporating a Spatial Transformer attention module and achieves a METEOR score of 18.1 on the 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 [3] 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 to neural image captioning has been explored in Xu et al [1]. In the architecture of [1], 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.



(a) (b)

Figure 1: (a) High level architecture of Xu et al [1] for an image caption decoder which takes input image features from VGGNet-19. Color denotes recurrent connections (green: LSTM hidden state, yellow: output word, which is fed back as an input after being transformed with a learned embedding matrix). (b) Attention module details. A probability distribution α is computed over the 14×14 image feature map locations at each time step t.

In this project, we experiment with the architecture in [1], 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 [1] and [2].

2 Attention-aided caption generation (Xu et al [1])

2.1 Model Architecture Overview

The architecture of [1] is shown in Fig. 1. VGGNet-19 [4] 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 vectors, α_{it}, is generated at each time step 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. The argmax of the posterior distribution $P(y_t|y_{t-1}, \mathbf{a})$ is taken to generate the next word. Note that "attention-aided" caption generation is our terminology, not used by [1].

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 [1] 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.

[1] 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 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 Effects of changes in Model Parameters and Hyperparameters

Not all the architecture details and training parameters used to produce the results in [1] 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 in [1].

We use the unaltered model of [1] as our reference benchmark for experiments in section 2. As [1] 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.

This section summarises the variation in BLEU and METEOR scores with changes in various parameters with respect to the base model. See table 6 in the Appendix for the final baseline parameters.

Table 2: Changes with various Model Parameters and HyperParameters

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Benchmark [1]	0.67	0.448	0.299	0.195	0.189
Baseline	0.52	0.289	0.162	0.096	0.162
Adadelta (vs RMSProp)	0.437	0.236	0.126	0.072	0.153
Adam (vs RMSProp)	0.548	0.3	0.17	0.103	0.153
SGD (vs RMSProp)	0.551	0.308	0.173	0.103	0.159
Word Embeddings=3000(vs 512)	0.577	0.322	0.182	0.109	0.156
Attention Layers 1(vs 2)	0.492	0.267	0.154	0.091	0.147
Output Layers 1(vs 2)	0.461	0.256	0.143	0.085	0.143
LSTM Layers 2(vs 1)	0.547	0.309	0.177	0.101	0.169
Initialization Layers 2(vs 1)	0.474	0.258	0.147	0.086	0.146

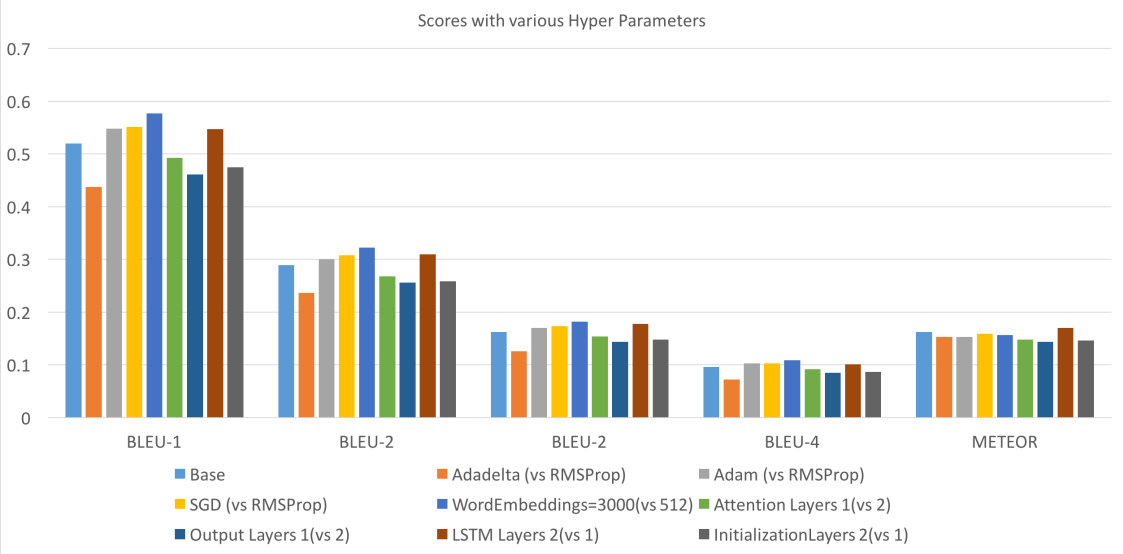


Figure 2: This graph represents changes in each score type with changes in hyper parameters. These changes include architectural as well as other changes. Description of hyperparameters can be found on 8

2.3.1 Hyperparameters

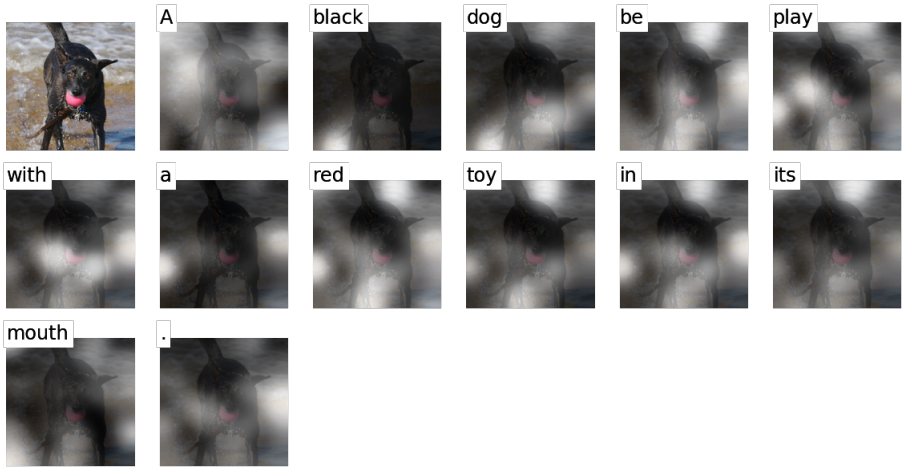
Effects on scores with various architectural changes as well as changes in other model parameters have been tabulated in table 2. Our base model used RMS prop which performed best in comparison with Adadelta, Adam and SGD as visible in the scores. This matches with the expectation of Xu's model[1]. Increasing the network complexity consistently gave better results. Higher number of Multi-Layer Perceptron (MLP) hidden layers helped in increasing the BLEU and METEOR scores, and vice-versa. Increasing the word-embedding dimensionality (input to LSTM) lead to an increase in BLEU scores w.r.t. the base model even though the METEOR scores reduced. A comparative representation of the scores in base model vs these changes can be seen on figure 2.

2.3.2 Early Stopping Criterion

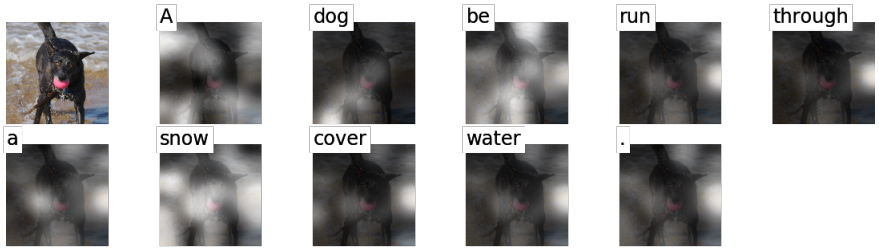
[5] monitors negative log likelihood of the objective function on validation set to determine when to early stop. [1] mentions use of BLEU score for early stopping instead for the published result. To try to achieve baseline closer to the published data, we also trained with early stop on BLEU scores. The result is shown in Table 4. Note that models with early stopping on BLEU-2, BLEU-3, and BLEU-4 require less epochs than the baseline, but scores higher on all BLEU metrics, signifying that the baseline model could be overfitting.

Table 3: Early Stopping Result

Flicker8k Soft Attention	Epochs	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Benchmark [1]	NA	0.667	0.434	0.288	0.191	0.1849
Baseline	83	0.521	0.3	0.162	0.096	0.162
Bleu-1	46	0.601	0.301	0.137	0.061	0.129
Bleu-2	63	0.586	0.329	0.185	0.106	0.154
Bleu-3	69	0.582	0.327	0.189	0.108	0.154
Bleu-4	71	0.586	0.326	0.185	0.109	0.153



(a)



(b)

Figure 3: (a) A sequence of α visualization of the baseline model along with the word generated. (b) A sequence of α visualization of the BLEU-4 early stop model along with the word generated.

2.4 Attention 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 [1]'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.

2.4.1 Removing Attention

One experiment that immediately suggests itself: assessing the impact of attention by effectively removing attention from this framework, by forcing $\{\alpha_{it}\}_{i=1}^{L}$ 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 than comparing [1] with a captioning architecture that does not use attention, such as [2], because of the variation in other aspects between the models, as [1] points out.

As expected, removing attention degrades the performance achieved, verifying that the additional complexity of the attention model does indeed improve performance, keeping other aspects of the architecture fixed.

Table 4: Removing Attention Performance

Flicker8k Soft Attention	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Baseline	0.521	0.3	0.162	0.096	0.162
Attention Removed	0.36	0.19	0.10	0.057	0.14

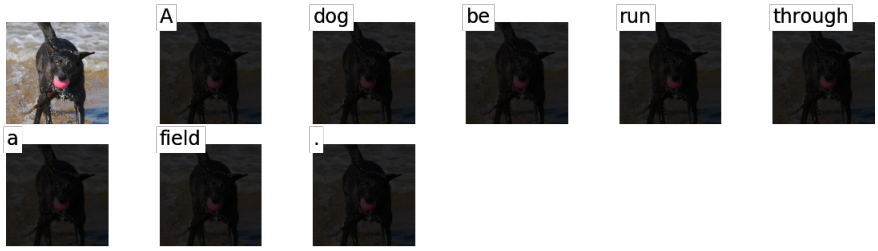


Figure 4: A sequence of α visualization of model trained without attention. This is essentially having uniform distribution of α.

2.4.2 Attention LSTM

A major effort was expended on replacing 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. This effort entailed about 60 hours of development time, including understanding the details of the source code [5], considering how the attention LSTM should be implemented, debugging, and attempting to modify sample caption generation during training in hopes of gaining insight into why the model was training poorly and making improvements. After this, there was only time to complete one full training run on a modified version of the design, which did not successfully train the network.

In order to follow the existing attention architecture as closely as possible in what was meant to be 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 system memory. Due to memory constraints, it was necessary to project the dimension 512 image feature vectors to a lower dimension, selected at 64 in order to accommodate other project GPU demands. In order to assess whether this was was a critical change that caused the model to not train successfully, we could have as a next step applied the same dimensionality reduction in the original model. However, we did not have time to do this. After determining whether dimensionality reduction worked in the original model, the next steps would have been to explore different training hyperparameters.

The modified code is available at [7], where the changes are to the file capgen.py.

2.4.3 Footnote about attention experiments

[3] is a recent work that introduces the problem of "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 [1]. Thus, the authors claim that this approach is more general. For simplicity, [3] follows the approach in [2], where the visual features are passed to the LSTM model once at the first time step.

We had originally proposed inserting the localization layer of [3] 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

[1] 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 [8]. The soft attention model is differentiable and can be trained using backprop.

As in [1], the objective function we minimize in our experiments is the following,

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

where 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). [1] call the regularization term "doubly stochastic regularization," stating that it forces the model to look at different locations in the image and produces richer captions.

3 Non-Attentive Probabilistic Caption Generation (NAPGen)

3.1 Model Architecture

This model is based on [2] and [9], in which an end-to-end differentiable system combines state-of-the-art networks for vision and language models. A deep CNN (GoogLeNet v3)[10] 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. 5 shows this architecture in which an LSTM model is combined with a CNN image embedder.

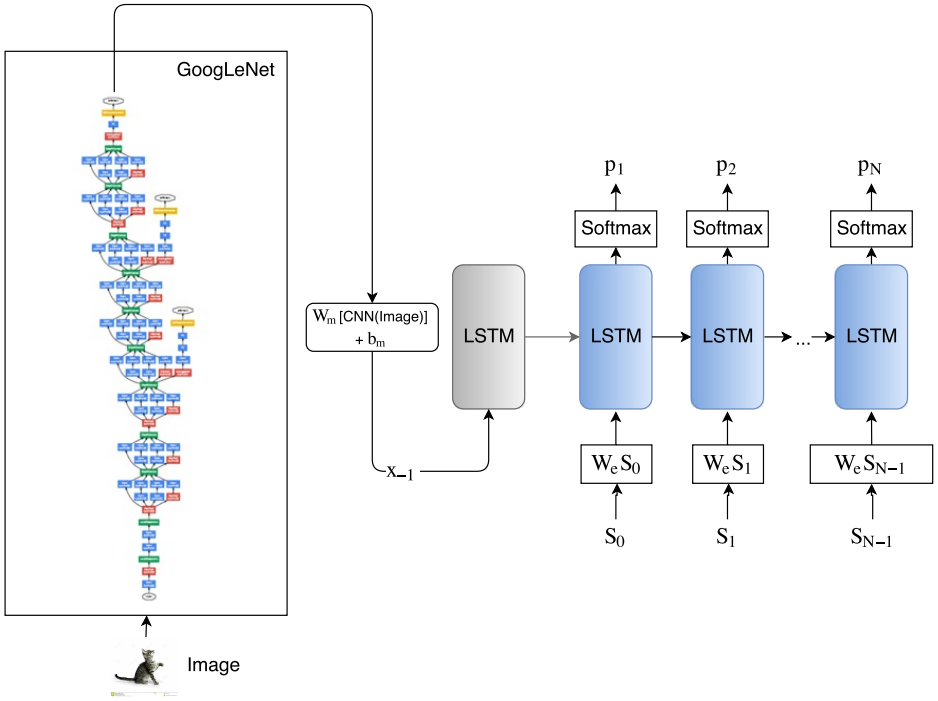


Figure 5: Diagram of the NAPGen model. The LSTM takes a word, context from previous time step and defines a probability distribution p_t over the next word in the sentence. The LSTM is conditioned on the embedded image vector at the first time step.

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 [11] 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 with respect to all the parameters of the LSTM, image embedding W_m and word embedding W_e. N refers to the maximum number of words in a caption and was chosen to be 32 in our model. 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. 50% dropout was applied on the input and output layers of LSTM 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). The main hyperparameter modified was the size of embeddings and LSTM memory. From experiments, 256 dimensions for both gave the best results. Details are reported in the results section.

3.5.1 Use of Spatial Transformer Network(STN) as Attention mechanism

To study the effects of attention, we leveraged Spatial Transformer Networks [12] in our model, which can help in attending to specific regions of image while generating captions.

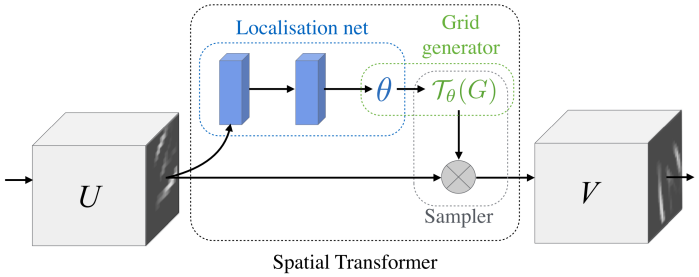


Figure 6: Architecture of Spatial Transformer module. The localisation network generates the parameters θ using the input feature map U. These parameters are then used by the grid generator and sampler to transform U into the cropped feature map V. Source: [12]

Spatial Transformer is a differentiable module that can be inserted into convolutional architectures, giving neural networks the ability to spatially transform feature map conditioned on the feature map itself, without any extra supervision. The transformations can include cropping, scaling and rotation.

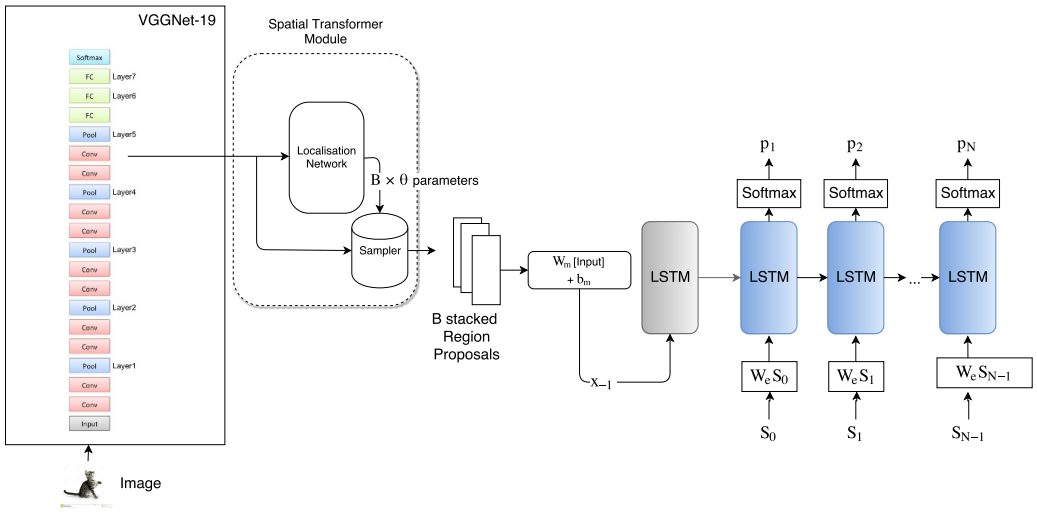


Figure 7: Architecture of NAPGen with Spatial Transformer module, which transforms(crops) the original image features to create a stack of B image maps, an embedding of which is fed to the LSTM.

Our hypothesis is that if we employ B (= 4 in our experiments) spatial transformers in parallel on the image features to produce B transformed(cropped) image maps which can be stacked and fed to the LSTM at initial time step, then the LSTM might be able to attend to this stack of region proposals by giving different weights to each region vector at different time steps using its memory state, thus emulating the soft-attention mechanism. An important change to be noted is that instead of using the final layer of GoogLeNet, we now use feature maps from a lower convolutional layer of VGGNet i.e. "conv5_4" , as we want the spatial information to be preserved before cropping or rotating the image regions. The final architecture of NAPGen with attention is depicted in figure 7.

3.6 Result

(Live Demo)

Table 5 summarizes the results of our experiments with NAPGen. We first train a model similar to [2] without using any attention mechanism, the performance of which is comparable to the state-of-the-art. Next, we introduce attention into this model by using Spatial Transformers. The use of spatial transformers increases the complexity of the model since we have several layers of the localization network, which produce the best parameters for transformation. Hence, the degradation in performance can be attributed to the increase in number of trainable parameters. This claim can be verified by the use of a bigger dataset like MS COCO.

Table 5: NAPGen Experiments on Flickr8k

Model	B-1	B-2	B-3	B-4	METEOR
Google NIC from [2]	0.63	0.41	0.27	-	-
NAPGen	0.61	0.387	0.232	0.13	0.181
NAPGen with Attention	0.47	0.261	0.139	0.07	0.172

4 Datasets

Experiments in [1] were performed on the widely used Flickr8k, Flickr30k, and COCO datasets. Due to computational time constraints, we focused on the Flickr8k dataset [13], which contains 8,000 images, each of which comes with five reference captions. We obtained this dataset by submitting a request at the website [14]. We use 6000, 1000, and 1000 split for train, validation, and test respectively.

5 Metrics

Scoring metrics originally designed to evaluate quality "machine translated" text from a natural language, such as BLEU and METEOR, have been adopted as a standard evaluation metric for image caption generation, and are the metrics used in [1] and [2].

More recently, CIDER has been designed particularly for evaluating image description, and address the similarity of sentences with the use of TFIDF. However [15] suggest having 50 reference captions is ideal for this metric.

We thus evaluate our results on FLickr8k with BLEU and METEOR, comparing with the results in [1], [2], and our results from running unaltered model of [5] as a baseline (see Section 4).

6 Software

The original source code for [1] is available at [5] which is written in Python and uses Theano. As stated in Section 2, we used and modified [5] for our experiments on attention-aided caption generation. We referred to [9] and [2], for the implementation of NAPGen in Lasagne. For employing the Spatial Transformer module, we used the "TransformerLayer" of Lasagne. We use [16] for calculating metric scores.

7 Discussion and Conclusions

In this project, we focused on [1] and [2], which are the current state-of-the-art image captioning models. Concretely, to gauge the impact of attention with respect to each of these models, we performed several experiments to modify and remove the existing attention mechanism in [1], and add our own attention mechanism to [2].

In an effort to replicate results in [1], we grid searched the hyperparameter and model architecture space. Although our trained model does not reach [1]'s BLEU and METEOR score, we use the best scoring model as the baseline comparison for our following experiments. Following the suggestion in [1], we trained the model with early stopping on various BLEU scores instead of negative log likelihood of the objective function. We suspected this would possibly combat overfitting of the model, and these models indeed stopped training at lower number of epochs. Although there were not significant differences in METEOR scores, the model performed better in terms of all BLEU scores.

For [1], we showed that setting the location distribution to a uniform distribution with soft attention, thereby effectively removing attention, degrades performance, verifying that it was not another aspect of model architecture responsible for its state of the art performance. We modified the architecture of [1] to replace the attention mechanism with a second LSTM, but did not have time to either finish successfully training this model or analyze its unsuitability.

Borrowing from [2], we trained a Non-Attentive Probabilistic model (NAPGen), which uses GoogLeNet to encode the image into a compact representation, followed by an LSTM that generates the corresponding caption. We verify that this model is quite accurate and comparable to the state-of-the-art, by quantitative evaluations which use BLEU or METEOR scores. We also incorporate attention mechanism by introducing the Spatial Transformer module, which provides the ability to generate multiple affine transformations of the original image features, different linear combination of which can be used by the LSTM at different time steps, thus emulating the soft attention mechanism. Although, the introduction of attention in NAPGen degrades the performance on Flickr8k due to increase in model complexity, it will be interesting to see its performance on other bigger datasets like COCO.

8 Contributions

Saurabh and Caroline worked on data pre-processing to generate the input as required by [5], while Ruby wrote drafts of the proposal, based on consensus from team meetings. Ruby initially discovered through studying [5] that some key training parameters had defaults in the published source code that differed from what was published in [1]. 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 [1]. Apoorve worked on effects of changes in various hyperparameters with respect to a baseline model achieved earlier. Caroline worked on modifying the early stopping criterion based on Bleu-n scores to achieve higher accuracy as well as attention visualizations. Caroline worked on removing attention from [1], helped by Ruby. Ruby worked on the attention LSTM modification to the architecture of Xu et al [5], helped by Caroline in understanding parts of the original code and in debugging. Saurabh and Apoorve worked on Non-Attentive Probabilistic Caption Generation model (NAPGen), Spatial Transformer module and Web GUI.

References

[1] 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.

[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] Justin Johnson, Andrej Karpathy, and Fei-Fei Li. Densecap: Fully convolutional localization networks for dense captioning. *arXiv preprint arXiv:1511.07571*, 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] https://bitbucket.org/ruby314eng/nic-attention/branch/model2. Accessed 3-17-2016.

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

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

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

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

[12] Max Jaderberg et al. Spatial transformer networks. *arXiv preprint arXiv:1506.02025v3*, 2016.

[13] 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.

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

[15] Ramakrishna Vedantam and Parikh Devi Zitnick, C. Lawrence. Cider: Consensus-based image description evaluation. *CVPR2015*, 2015.

[16] https://github.com/tylin/coco-caption. Accessed 3-2-2016.

APPENDIX

Description of Xu et al Model Parameters

1. **Attention MLP Layers:** `n_layers_att`: Normalized weights for various attention vectors, α_i as mentioned in [1], are calculated using a multilayer perceptron with these many hidden layers.
2. **Initialization MLP Layers:** `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.
3. **LSTM Layers:** `n_layers_lstm`: As the title suggests, it is the number of LSTM layers in the model.
4. **Output MLP Layers:** `n_layers_out`: Number of hidden layers in multilayer perceptron, which is used to compute probabilities of output word vector $\boldsymbol{y}$.
5. **Word Embedding Dimensionality:** This gives the dimensionality of embedding $\boldsymbol{Ey_t}$ obtained from output word vector $\boldsymbol{y_t}$.

Baseline model parameter and hyperparameters

Table 6: Baseline model parameter and hyperparameters

n_words	10000
decay_c	0.0
patience	10
n_layers_init	1
RL_sumCost	True
max_epochs	1000
dispFreq	100
attn_type	deterministic
validFreq	1000
temperature	1.0
n_layers_att	1
ctx_dim	512
valid_batch_size	64
lstm_encoder	False
n_layers_lstm	1
optimizer	rmsprop
alpha_c	1.0
dictionary	None
batch_size	64
selector	True
lrate	0.01
dataset	flickr8k
ctx2out	True
prev2out	True
dim	1000
use_dropout	True
dim_word	100
sampleFreq	100
semi_sampling_p	0.5
n_layers_out	1
maxlen	100
alpha_entropy_c	0.002
use_dropout_lstm	False