Deep Learning (COMP-54) : Spring 2019
Recurrent Models of Visual Attention:
A Julia Implementation

Ahnaf Hannan Lodhi

Spring 2019

Abstract

Convolutional Neural Networks while offering an effective tool for classification are computationally expensive. Furthermore, the cost is compounded by images of large size invoking prohibitively excessive requirements both in terms of hardware and time. This paper [1] offered a Recurrent Neural Network based model which mimics human visual systems by adaptively focusing on region of the image to extract information. The mechanism gathers its motivation from human visual system which progressively builds a picture of its surroundings by building a foveated picture of its surroundings. The implementation of this visual attention scheme is based on a similar phenomenon in which it is assumed that the complete information of the environment is not available. A band-limited sensor is deployed which acquires information from its location of deployment. The information is then passed to a Neural Network operating in Recurrent Mode which ultimately aims to extract useful information and perform classification operations. However, a major challenge faced here is that the adaptation mechanism identified here is not differentiable. The parameters are thus updated using Reinforcement Learning techniques. The implementation of the paper was attempted in 'Julia' language and Knet[2] repository.

1 Introduction

Neural Networks have become a de-facto standard in various classification tasks. The performance of such networks has been remarkable in problems based on visual information. Convolutional Neural Networks have since their inception found widespread applications is this domain. The ability of such networks to assimilate information from images has resulted in widespread deployment of such Networks [3]. Their excellent computational accuracy however comes at a high price. Convolutional networks operate sequentially taking patches of images and progressively covering the entire image. While the high accuracy may be attributed to this phenomenon, the process itself is computationally expensive with very high training times. One major disadvantage of sequential profiling is the employment of same level of computational effort over image patches which do not contain information useful in the context of the problem. One possible approach to dealing with such problems when using large or high resolution datasets is down-sampling. There are however, problems which even being trained on multi-gpu systems required days [4].

The human perceptual system on the other hand does not process the entire image at once. This visual system instead focuses selectively on information rich areas of the environment and successively builds the a scene of the image. During this process, multiple fixations generated across the visual scope are built and the overall image is based on combining information from these fixations. The most obvious advantage of such a systems is the reduction in bandwidth required to process in any image which in turn reduces the computational complexity while processing only meaningful information.

This research conducted at 'Google DeepMind' aimed to replicate the human system of visual attention. The formulation of the problem has been done as a *'Control Problem'* where in the the control is applied to the location requiring focus. Instead of a purely sequential model, the authors have chosen a Recurrent Neural Network Configuration. The model builds builds up an incremental information about its surroundings and integrates them over time.

2 Previous Work

With the increasing mount of details being used by images and ever-growing sizes, computational complexity has been receiving considerable focus in the domain of Computer Vision. Most of the earlier techniques however have delved on image pre-processing to minimize the size before applying computations. The aim primarily was to address the amount of data to be covered by sliding window domains. Another class of approaches to address this problem has been the saliency detectors. These methods prioritize processing of the image regions most likely to contain information critical to classification operations. While being successful, saliency detectors do not typically integrate information which makes employing them for conducting semantics and context based less likely.

The work conducted by the authors adopts a sequential style of processing various regions of image, an approach which has previously been utilized by others attempting to do the same. However, considerably less learning was involved and the setups developed in those cases offer reduced flexibility.

3 Dataset

The dataset used for the experiments comprises of the standard **MNIST** and its variants. The standard MNIST comprises of hand-written digits having 70,000 samples in total. The dataset is distributed into a training set of 60,000 and testing set of 10,000 samples respectively. However, the paper also describes modification to the said dataset in form of translation and clutter to evaluate the efficacy of the network.

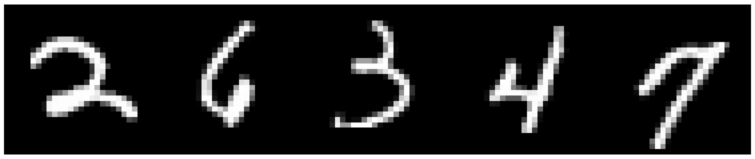


Figure 1: Standard 28 x 28 MNIST

Dataset Modifications The modifications to the dataset include three variations which aim to evaluate the effectiveness of the algorithm under conditions where noise or translation to mimic real world images.

60 x 60 Translation: The first modification entails that the dataset is placed randomly within a blank frame of size **60 x 60**. This is aimed at translating the centered MNIST images.

100 x 100 Translation: Additionally, a 100 x 100 translated dataset was also generated by randomly placing the centered MNIST images in blank 100 x 100 patches.

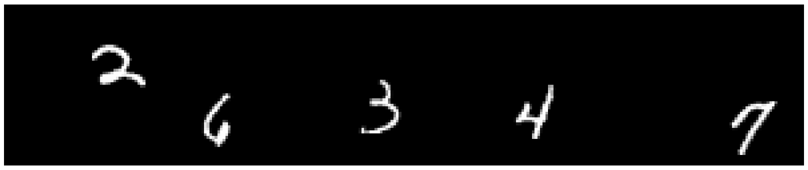


Figure 2: Translated 60 x 60 MNIST

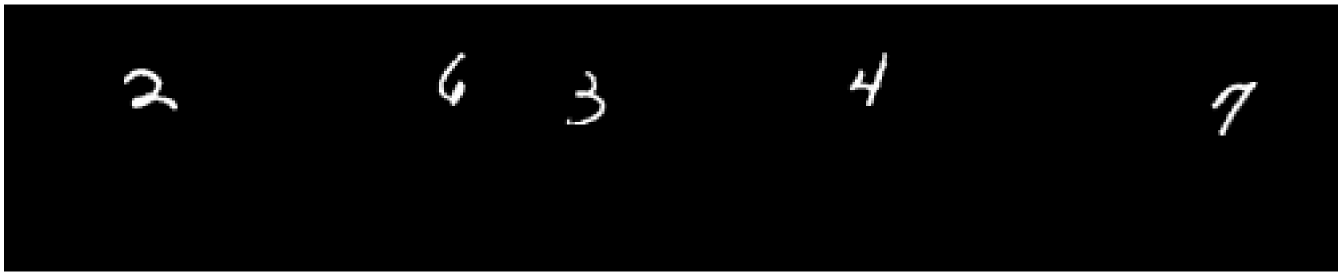


Figure 3: 100 x 100 Translated MNIST

60 x 60 Cluttered MNIST: In order to assess the classification abilities of the network, an additional clutter was introduced on top of the *60 x 60* MNIST. The clutter is introduced by randomly selected patches from the MNIST dataset and randomly placed in **60 x 60** frame.

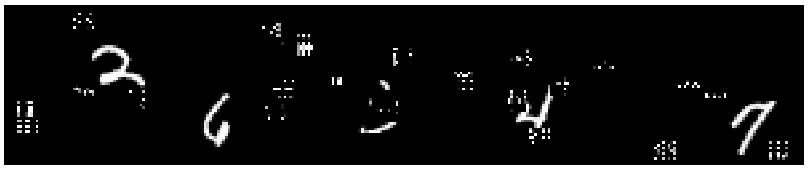


Figure 4: 60 x 60 Cluttered MNIST

100 x 100 Cluttered MNIST Using the same principles employed for generating the 60 x 60 Cluttered dataset, same was employed for generating a 100 x 100 Cluttered dataset.

4 Network Architecture

The work under consideration considers the attention problem as a sequential decision process of goal-oriented agent acting on a visual field. However, the agent is unable to encompass the complete information from the environment instead relying on patches obtained from the image to extract information. However, the model is defined so as to allow control of the agent over the where the sensor is centered. After each computation step which may be multiple time steps depending upon the number of glimpses, a reward is assigned to the agent in case of correct classification. The agent is formulated to attempt to maximize reward. This information is then integrated over a number of times in a recurrent fashion and eventually used for classification purposes. The overall structure of the network is shown in Figure[6].
The flow of information in the model can be summarized as follows from Figure[7]:

1. Information Acquisition through Sensor
2. Internal state update

Figure 5: 100 x 100 Cluttered MNIST

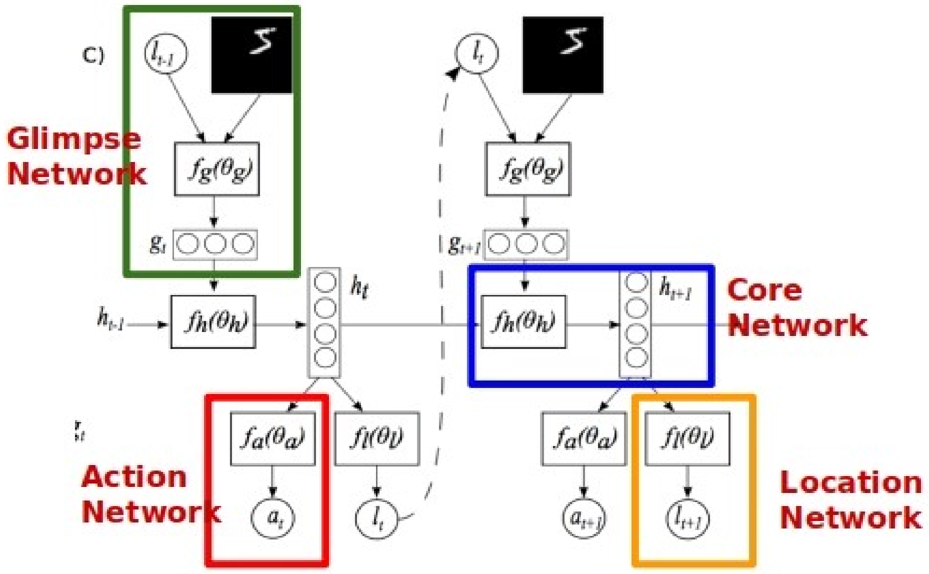


Figure 6: Recurrent[1] Network expanded in time

3. Action identification

4. Reward calculation

The model is structures to emulate each of these steps in distinct sub-networks to maintain an explainable information flow. These individual sub-networks are:

1. Glimpse Network

2. Core Network

3. Location Network

4. Action Network

The individual steps as well the network operation is explained as follows

4.1 Glimpse Network

The Glimpse Network is formulated to integrate the information from image and corresponding location. The combined information is then subsequently used for calculating the current hidden state (h_t) combining with the previous hidden state (h_{t-1}).

The Glimpse Network wields a sensor through which it receives a partial observation of the image at each time step. The sensor is deployed based on the input location (l_{t-1}) from the previous time step. The sensor encodes the information around the location at higher resolution with bandwidth

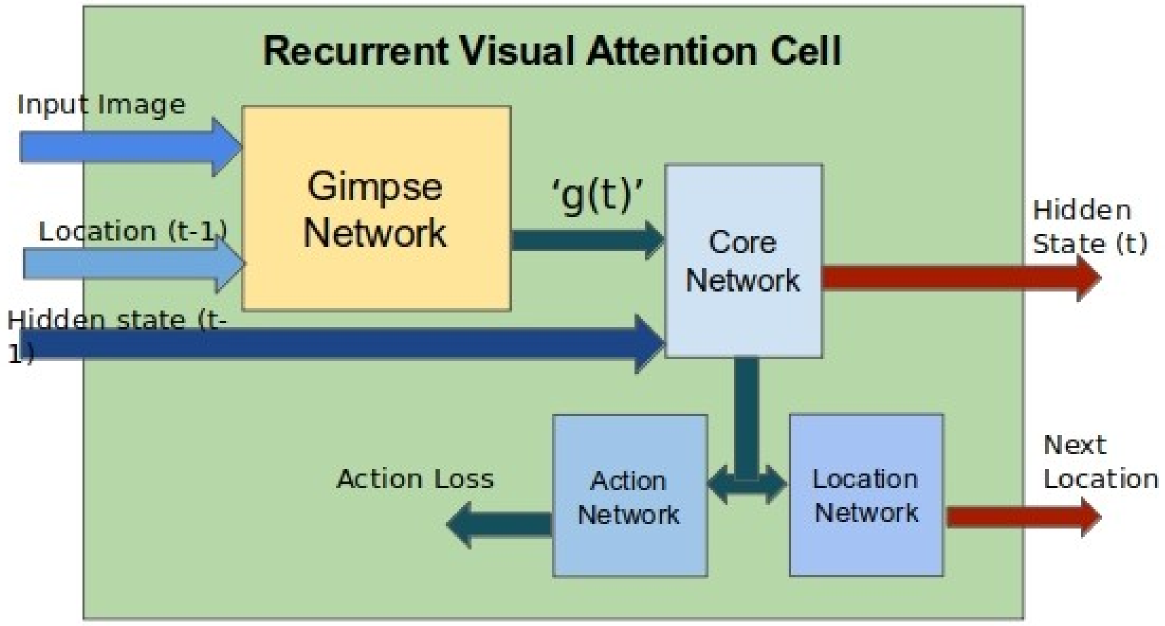


Figure 7: Single Recurrent Cell

whereas the resolution is decreased progressively with a pre-defined scale around the location based on the number of image patches to be recorded. This collection of partial observations of the environment is referred to as a 'glimpse'. Along with the glimpse information, the glimpse network also transforms the location information through a series of neurons with non-linear activation. The information from the glimpses and location is combined in the culmination step of the network. Thus, the Glimpse Network is detailed as $\rho(x_t, l_{t-1})$. The output of the Glimpse Network is the feature vector glimpse feature vector, $g_t : g_t = f_g(x_t, l_{t-1} : \theta_g)$.

A version implemented by [14] instead chooses an alternate option concatenating the outputs of h_g and h_l instead of addition and then passes them through the linear layer activated by a Relu.

4.2 Core Network

The internal state for the overall network is maintained by the Core Network. The Core Network encodes the glimpse features vector g_t and the information from the previous time step h_{t-1} in a combined fashion to generate the new hidden state h_t. The size of the both g_t and h_{t-1} is an array of 256 elements each which are combined as follows:

$$h_t = Rect(Linear(g_t) + Linear(h_{t-1}))$$

The paper also describes a slight modification for the core network in case of a dynamic environment in which an LSTM is employed instead of the RNN module for the above equation. The new hidden state h_t is then passed on to the next time step as well as used for calculating the reward and transformed into the next location for sensor deployment.

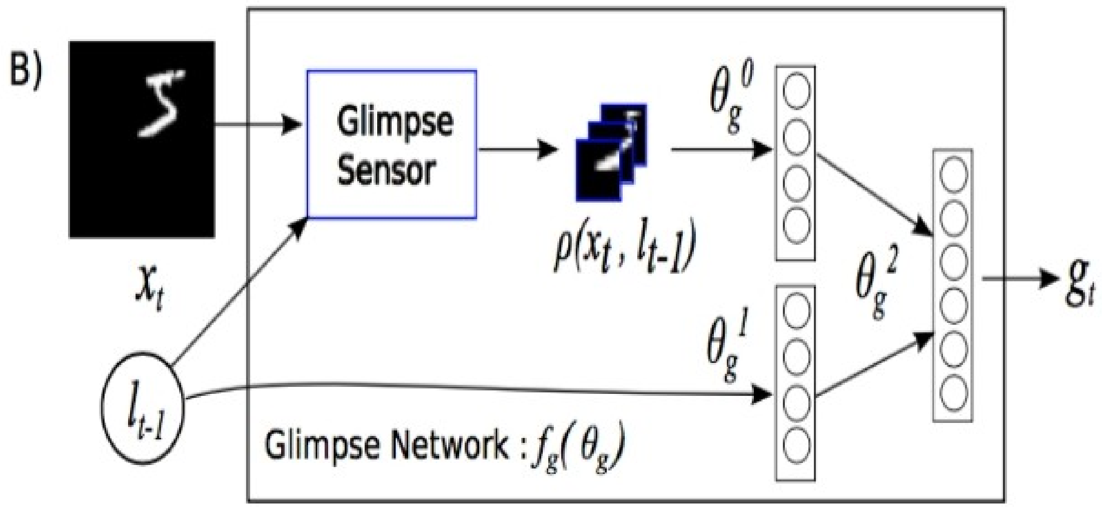


Figure 8: Glimpse Network

4.3 Action Network

The Action Network is one of the ways the agent interacts with the environment. This network is configured to allow the agent to process the internal state h_t and undertake classification operation on it. The Action Network can be described as :

$$\hat{y} \text{ @ t =T} = argmax(Linear(h_t))\text{where T = Number of Glimpses}$$

The Network calculates the scores of of the respective classes and classifies the input sequence based on these results. However, the classification operation takes places after a number of time steps equalling number of glimpses. The output of this sub-network also results in the reward being assigned to the agent which will be covered under the training section.

4.4 Location Network

The Location Network is the other sub-network within this configuration which allows some control to the agent. Based on the hidden state h_t, the agent stochastically decides the next location where the sensor is going to be aimed. However, as opposed to a delayed response by the Action Network, the Location Network generates location at each time step using the current hidden state as a reference.

$$\mu_{loc} = Linear(h_t) \text{ where } h_t \text{ is based on } l_{t-1}$$

$$l_t = \mu + \delta$$

where δ is a perturbation sampled from a Normal distribution defined as:

$$\delta = Normal(mean : \mu = 0, Std : \sigma = 1)$$

The Location Network generates the new location l_t which is then used in time step t+1

5 Loss and Reward Allocation

The above architecture is designed to maximize the reward **R** assigned to the agent after **T** time steps.

$$J(\theta) = \underset{p(s_{1:T});\theta}{\mathbb{E}} \left[\Sigma_{t=1}^{T} = r_t\right] = \underset{p(s_{1:T};\theta)}{\mathbb{E}} [R]$$

However, maximizing $J(\theta)$ is non-trivial as it involves an expectation operation. In regular instances, this renders the network non-differentiable and thus un-trainable. In order to address this issue, the problem is therefore modeled as a Partially Observable Markov Decision Process **(POMDP)**. Using this formulation, [5] provides a approximation to the gradient as:

$$\nabla_\theta J = \frac{1}{M}\Sigma_{i=1}^{M}\Sigma_{t=1}^{T}\nabla_\theta log\pi(u_t^i|s_{1:t}^i;\theta)R^i$$

where the results are approximated over **M** episodes using s states and θ parameters. It is further detailed that $\nabla_\theta log\pi(u_t^i|s_{1:t}^i;\theta)$ is the gradient of the policy defining network which in this case is being performed by the RNN.
This formulation provides an unbiased estimate of the gradient which may result in higher gradients. The paper therefore utilizes the gradient estimate of the following form for reduction in variance:

$$\nabla_\theta J = \frac{1}{M}\Sigma_{i=1}^{M}\Sigma_{t=1}^{T}\nabla_\theta log\pi(u_t^i|s_{1:t}^i;\theta)(R^i - b_t)$$

where b_t is the baseline defined as $\mathbb{E}_\pi[R_t]$ which is the value function.

5.1 Hybrid Loss

The loss formulation for the training takes the form of a hybrid loss which takes into account the effects the agent can exert on the environment and the rewards it receives in return.

$$J = J_{action} + J_{baseline} + J_{location}$$

where

$$J_{action} = nll(\hat{y}, y_{ref})$$

,baseline loss $J_{baseline}$

$$J_{baseline} = (R^i - b_t)^2$$

and $J_{location}$ is trained through the REINFORCE. Since the new location has been sampled from a Gaussian Distribution, the resulting loss has been:

$$J_{REINFORCE} = -log(\sigma) - \frac{log\sqrt{2\pi}}{2} - \frac{(l_t - \mu_{loc})^2}{2\sigma}$$

One of the REINFORCE loss versions that used by some of the implementations was also evaluated:

$$J_{REINFORCE} = \frac{(l_t - \mu_{loc})^2}{2\sigma^2}$$

However, the results obtained though this variant in terms of loss and accuracy do not improve over the previous versions

6 Experiments and Results

The results obtained by the paper establish baselines using two baseline networks including a Fully Connected Network with 2 layers and a Convolutional Neural Network using 1 Convolutional layer and one fully connected layer for the standard MNIST. Similar baselines were also established for variants of the same dataset to develop reverences thresholds in terms of performance and achieved accuracy.

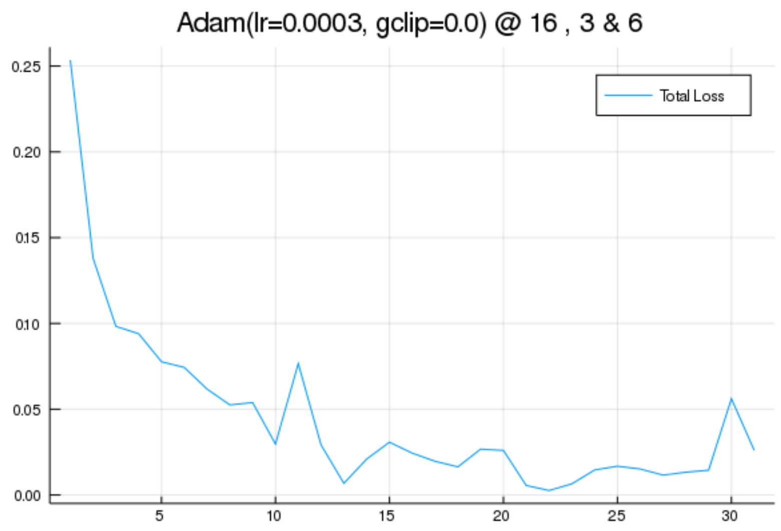
Similar implementation in Julia achieved the following baselines as included in the Baseline file:

Dataset	Configuration	Accuracy
28 x 28 MNIST	FC2 (256)	91% @ 100 epochs
28 x 28 MNIST	Conv(10 x 10 x 8)	91.78% @ 100 epochs

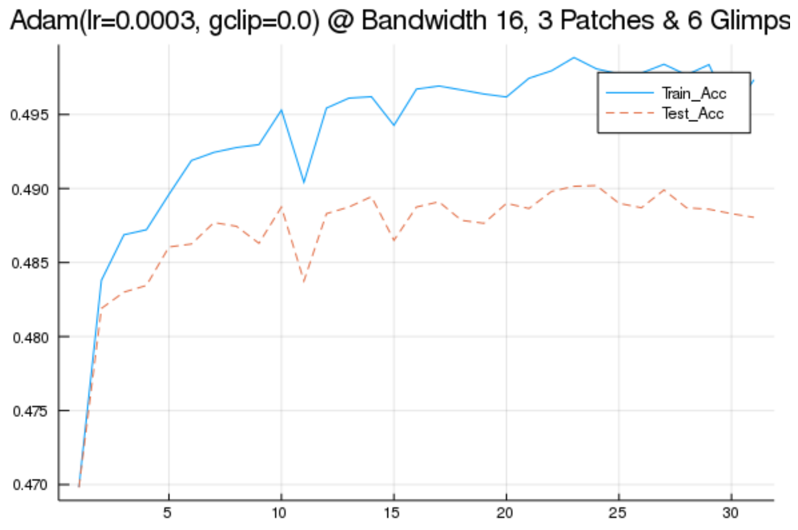
The experiments were aimed at identifying the proper trends in the learning domain. The results are as follows:

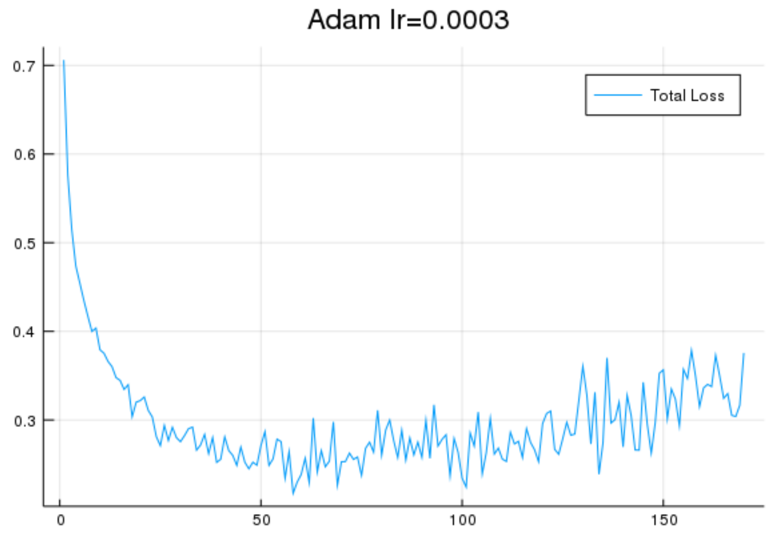
Dataset	Layers	Error
2*28x28 MNIST	FC2	1.69%
	RAM 7 Glimpses	1.07%
2* 60x60 Translated MNIST	FC2	6.42%
	RAM 8 Glimpse	1.2%

Table 1: Results of the Reference Paper



Variants Among other experiments that were conducted was the modification of Glimpse Net to concatenate the resulting h_g and h_l vectors into a combined 256 element array before passing it through a Linear layer with Relu Activation. The experiment yielded no improvement over the regular versions. Another variation was the introduction of batchnorm layers to the original layer to speed-up training.



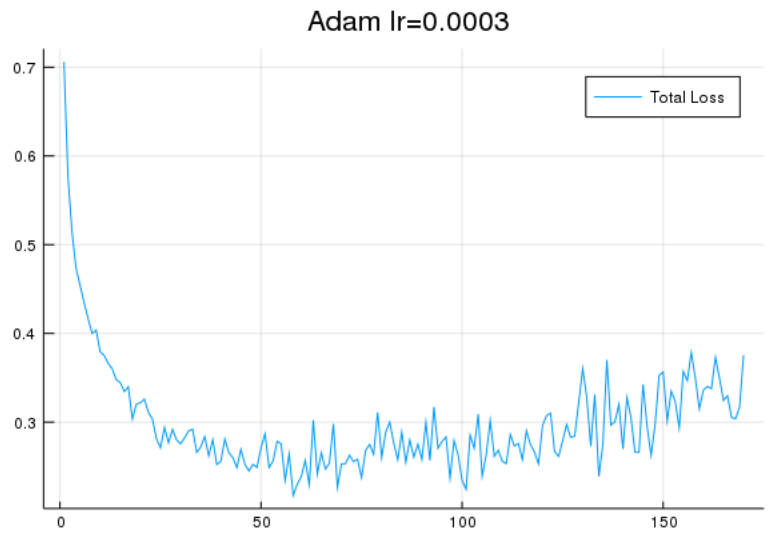


7 Analysis and Conclusion

Based on the replication of this paper, REINFORCE algorithm provides a major solution for developing Hybrid losses where Gaussian distributions are sampled. Owing to the nature of the problem and configuration, various implementations report success to varying degrees. However, the configuration does result in reducing the resultant losses. The maximum accuracy achieved during this implementation was 50%. A major concern in implementing this configuration was the backprop for the REINFORCE part which can cause gradients to be corrupted/distorted. The major aim during this implementation has been to increase the accuracy with minimal sensor size in order to fully capitalize on the strengths of the soft attention mechanism

7.1 Acknowledgements

The version on which these results have been compiled has closely followed implementations at [10] and [9]. The structure of the code has followed the approach adopted by [9] for the Julia which was originally used by [10] for the PyTorch implementation. The sections including loss and validate have



been principally taken from [9] implementation. While, validate has been not been changed for its efficiency and comprehensive approach, loss has multiple revisions in so far as how the REINFORCE loss has been calculated as well the attempting to understand the flow of gradients for it as well as other components of the loss. This implementation has also referenced [11] as well as [12] for developing a further understanding of the information flow through the network. Two other versions by [13] and [14] were implemented to identify whether structural changes which they use can bring in any improvements.

The layer definitions, individual functions have been written particularly for this project. Additionally, the main Network call, though follows the structure from the references, uses local implementation and thus different arguments.

7.2 Related Works

Some of the major works that have since followed this publication [6], [7] and [8]

References

[1] Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention." Advances in neural information processing systems. 2014.

[2] https://denizyuret.github.io/Knet.jl/latest/

[3] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[4] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[5] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992): 229-256.

[6] Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." arXiv preprint arXiv:1502.03044 (2015)

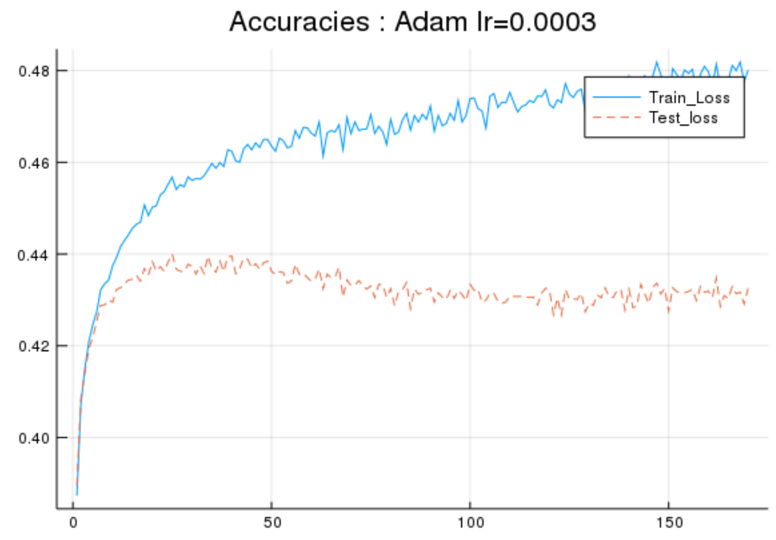


Figure 9: Caption

[7] Ba, Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu. ”Multiple object recognition with visual attention.” arXiv preprint arXiv:1412.7755 (2014).

[8] Vaswani, Ashish, et al. ”Attention is all you need.” Advances in neural information processing systems. 2017.

[9] https://github.com/ilkerkesen/RAM

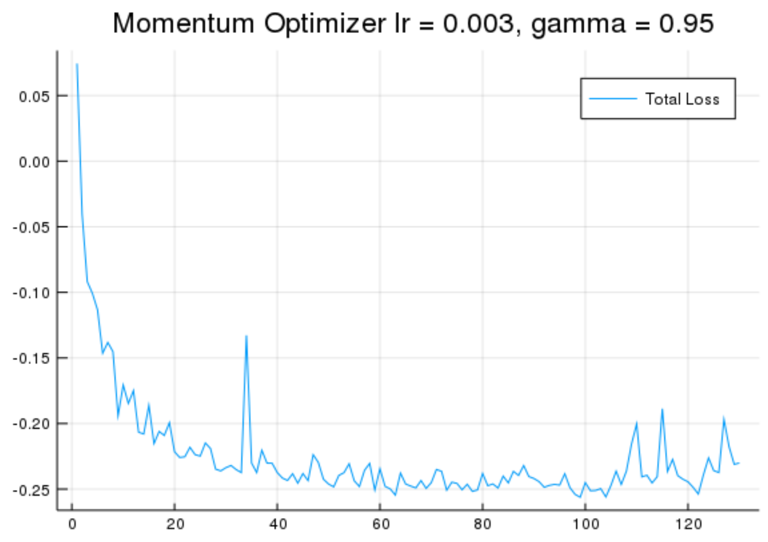
[10] https://github.com/kevinzakka/recurrent-visual-attention

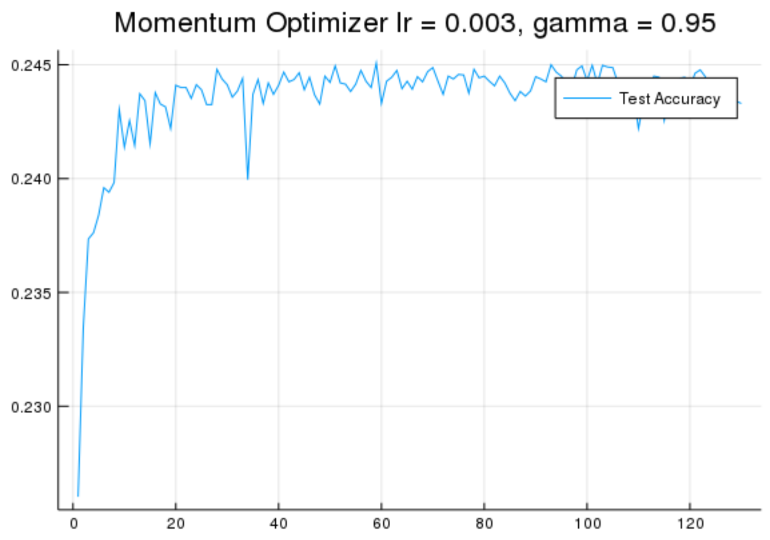
[11] https://github.com/jlindsey15/RAM

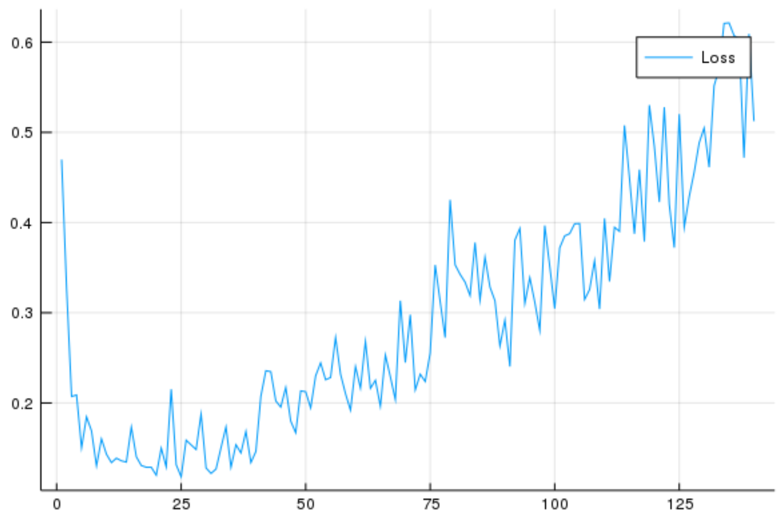
[12] https://github.com/SunnerLi/RAM

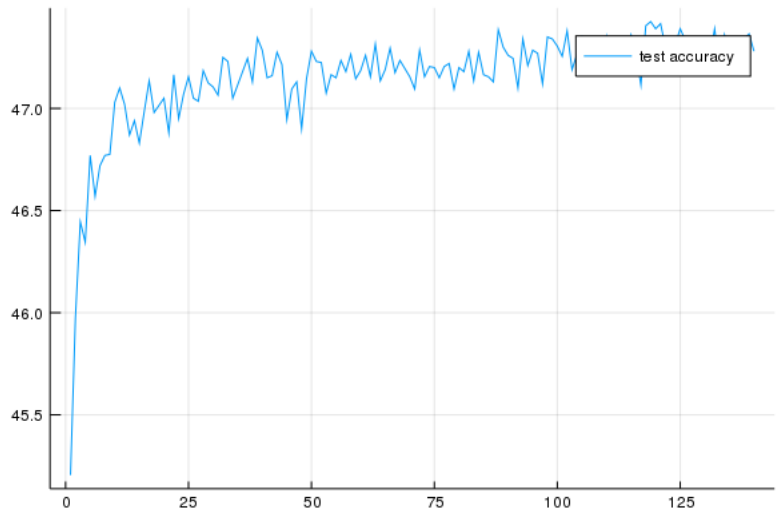
[13] https://github.com/fleer/RAM

[14] https://github.com/amasky/ram


Momentum Optimizer lr = 0.003, gamma = 0.95
Total Loss
0.05
0.00
-0.05
-0.10
-0.15
-0.20
-0.25
0
20
40
60
80
100
120


Momentum Optimizer lr = 0.003, gamma = 0.95
Test Accuracy
0.245
0.240
0.235
0.230
0
20
40
60
80
100
120


Loss
0.6
0.5
0.4
0.3
0.2
0
25
50
75
100
125


test accuracy
47.0
46.5
46.0
45.5
0
25
50
75
100
125