Reimplementation of the Paper "DRAW: A Recurrent Neural Network For Image Generation" in Julia using Knet.jl

Arda Arslan
Koc University
ararslan13@ku.edu.tr

***Abstract*— This paper introduces the reimplementation of the paper "DRAW: A Recurrent Neural Network For Image Generation" which originally belongs to Google DeepMind. Abbreviation DRAW stands for Deep Recurrent Attentive Writer. It is a neural network architecture which uses two main mechanisms to generate images: spatial attention and iterative refinement. Spatial attention can be defined as improving only a small region of the image at a time. Iterative refinement can be defined as improving the image over time instead of creating the image at one step. By using these two mechanisms, DRAW network managed to improve on the state of the art for generative models on the MNIST and the SVHN(Street View House Numbers) datasets and classification model on the Cluttered-MNIST dataset. The authors of the original paper tested DRAW network with 5 different experiments. In this paper, the reimplementation of these 5 experiments are explained.**

I. Introduction

In this paper, the replication process of the paper "DRAW: A Recurrent Neural Network For Image Generation" is discussed. DRAW is a neural network which reads images with a differentiable attention mechanism and then generates images with again a differentiable attention mechanism. Contrary to the other generative models, DRAW generates images at several steps instead of generating an image at a single step. And this mechanism is called iterative refinement mechanism. To test the generation power of DRAW, the authors used MNIST, SVHN(Street View House Numbers) and CIFAR-10 datasets. And to test the classification power of DRAW, the authors used Cluttered-MNIST dataset. These experiments showed that DRAW network managed to improve on the state of the art for generative models on the MNIST and the SVHN(Street View House Numbers) datasets and classification model on the Cluttered-MNIST dataset. In this paper, first, the main mechanisms used in the original paper will be discussed. Then, the results of the original paper will be examined. After that, the replication process will be discussed in details. For replication of DRAW, I used the programming language named Julia. And to train my model, I used AutoGrad and Knet.jl frameworks which are being developed by a group leaded by Deniz Yuret who is a professor in Koc University.

II. The Original Model

A. Main Mechanisms of the Original Model

Before discussing the reimplementation process, it would be rational to give details about the architecture of the original model. The authors of the original paper define DRAW as "A network which combines a novel spatial attention mechanism that mimics the foveation of the human eye, with a sequential variational auto-encoding framework that allows for the iterative construction of complex images(Gregor et al., 2015)." In this quote, authors mention two mechanisms:

1) Iterative Refinement Mechanism: Most generative models create images in one step. However, if you ask a painter to draw a picture, she would draw it step by step, in other words she would first draw some lines or apply some brush strokes and continue adding these over and over again. By saying "iterative construction of complex images", the authors claim that DRAW network is able to generate images just like a painter would do. This mechanism can also be named as "iterative refinement mechanism". To implement this mechanism, the authors used a specialized version of variational auto-encoders. The term variational auto-encoder can be simply defined as a model which converts an image into codes using an encoder and then converts these codes back into an image using a decoder. There are some differences between the variational auto-encoder that the authors used and the common one. In DRAW, encoder and decoder are both LSTM modules. So they used Recurrent Neural Networks for DRAW. In this network, the encoder receives the previous output of the decoder. And the final difference is that in DRAW, the decoder generates an image several times instead of generating only once. By using this specialized version of the variational auto-encoders, the image generation becomes dependent on multiple latent distributions and this makes the network more powerful in terms of generating images that cannot be distinguished from real data.

2) Spatial Attention Mechanism: By saying "mimics the foveation of the human eye", the authors mention another mechanism used in DRAW. Instead of improving the image as a whole at each step, DRAW improves only a small region of the image that it generates. This mechanism is called "spatial attention mechanism". This mechanism is used both during reading the input image and during writing to the output image. So DRAW network "decides at each

time-step 'where to read', 'where to write' and 'what to write'"(Gregor et al., 2015). In other image generation models such as Recurrent Attention Model(Mnih et al., 2014), the attention mechanism is implemented using reinforcement learning. However the attention model used in DRAW, is fully differentiable. So it is possible to train this model with standard backpropagation(Gregor et al., 2015). The region where DRAW reads at every time step is called a "glimpse". These glimpses are defined by the coordinates of their centers(x, y), its stride, its Gaussian Variance and a scalar multiplier. Stride is simply the zoom level. And Gaussian Variance is the measure of blurriness of the result. For simple experiments like MNIST generation, Gaussian Variance is expected to be low. However for challenging experiments like CIFAR-10 generation, Gaussian Variance is expected to be high, which means the results of CIFAR-10 generation experiment will be blurry. These parameters are calculated using the output of the decoder LSTM.

B. Loss Function of the Original Paper

In the paper, the loss function is defined as "a variational upper bound on the log-likelihood of the data"(Gregor et al., 2015). The loss function in DRAW model is the sum of a reconstruction loss(how good the picture looks) and a latent loss(a measure of how bad our variational approximation is of the true latent distribution)(Jang, 2016).

1) Reconstruction Loss: Since each pixel at the generated MNIST image should have a value between 0 and 1, instead of using mean−normalized cross entropy(-0.5 to 0.5), binary cross entropy(0 to 1) is used for calculating the reconstruction loss(Jang, 2016). For SVHN and CIFAR-10 images, the cross-entropy between the pixel intensities and the model probabilities is used to calculate the reconstruction loss.

2) Latent Loss: In variational auto-encoders, it is more common to use Bernoulli distributions instead of Gaussians for latent variables(Dayan et al., 1995; Gregor et al., 2014). However the authors of the DRAW thought that by using Gaussian latents, the gradient of a function of the samples can be calculated using the reparametrization trick. This makes it straightforward to back-propagate unbiased, low variance stochastic gradients of the loss function through the latent distribution(Gregor et al., 2015).

C. Results of the Experiments in the Original Paper

1) MNIST Generation: Negative log-likelihood values per test-set example on the MNIST data set for DRAW and other generative models are given in Table I:

As it can be observed in Table I, DRAW without attention mechanism had a similar result comparing with the previous models. However DRAW with attention mechanism managed to improve on the state of the art for generative models on the MNIST dataset.

2) MNIST Generation with Two Digits: In the paper, a numerical result for this experiment was not given. Authors claim that the only difference between this experiment and the MNIST Generation experiment is that, for this experiment, they trained DRAW to generate MNIST images with two 28x28 MNIST images chosen at random and placed at random locations in a 60x60 black background(Gregor et al., 2015).

TABLE I

COMPARISON OF MNIST GENERATION MODELS IN TERMS OF NEGATIVE LOG-LIKELIHOOD VALUES PER TEST-SET EXAMPLE

Model	-logp	$\leq$
DBM 2hl	$\approx$ 84.62	
DBN 2hl	$\approx$ 84.55	
NADE	88.33	
EoNADE 2hl (128 orderings)	85.10	
EoNADE 2hl (128 orderings)	85.10	
EoNADE-5 2hl (128 orderings)	84.68	
DLGM	$\approx$ 86.60	
DLGM 8 leapfrog steps	$\approx$ 85.51	88.30
DARN 1hl	$\approx$ 84.13	88.30
DARN 12hl	-	87.72
DRAW without attention	-	87.40
DRAW	-	**80.97**

3) Street View House Number Generation: In the paper, it is claimed that DRAW network was trained with approximately 350,000 minibatches and the cost per example for the final minibatch is given as 5080. According to the authors, when DRAW is trained on the Street View House Numbers dataset, it generates images that cannot be distinguished from real data with the naked eye(Gregor et al., 2015).

4) CIFAR-10 Generation: In the paper, a numerical result for this experiment was not given either. However the authors shared an example of the results of the CIFAR-10 generation experiment. In this example, the images show that DRAW network managed to capture much of the shape, color and composition of real photographs(Gregor et al., 2015). However the generated images were blurry.

5) Cluttered-MNIST Classification: Cluttered-MNIST dataset consists of images with many digit-like fragments of visual clutter. The challenging part of this experiment is to make the DRAW network find the correct place of the digit in this image and classify it successfully. The results of the experiments with Cluttered-MNIST dataset using different models are given in Table II:

TABLE II

COMPARISON OF CLUTTERED-MNIST CLASSIFICATION MODELS IN TERMS OF THEIR TEST ERRORS

Model	Error
Convolutional, 2 layers	14.35%
RAM, 4 glimpses, 12x12, 4 scales	9.41%
RAM, 8 glimpses, 12x12, 8 scales	8.11%
Differentiable RAM, 4 glimpses, 12x12	4.18%
Differentiable RAM, 8 glimpses, 12x12	3.36%

As it can be observed in Table II, DRAW managed to improve on the state of the art for classification models on the Cluttered-MNIST dataset. One other observation is that increasing the number of glimpses(from 4 to 8) made it possible to decrease the test error.

III. Literature Review

A. Past Work

1) Deep Autoregressive Networks(Gregor et al., 2013): This paper is known as DARN. And two authors of DARN are also authors of DRAW paper. In this paper, the main objective is to use an autoencoder which will help learning some input such as handwriting. DARN is a deep generative architecture with autoregressive stochastic hidden units which is used for understanding complex structure in data to generate high-quality images. The method consists of not just a decoder but also a stochastic encoder as well to allow efficient and tractable inference. While training the model, the authors minimized the Helmholtz variational free energy. To do this the authors applied backpropagation through stochastic units, and to do that they used Monte Carlo method. In DARN, there is a comparison between DARN and the other generative models. It can be said that when the time DARN was written, it had the best accuracy while generating MNIST images. However, in DRAW, in the experiments section, there is a comparison between DARN and DRAW accuracies. These experiments show that DRAW has a higher accuracy on MNIST data set. During the implementation of DRAW, looking at the implementation of DARN may be quite useful.

2) Generating Sequences with Recurrent Neural Networks(Graves et al., 2013): This paper may be useful for understanding selective reading and writing operations. In this paper, there is an introduction about a differentiable attention mechanism which is used in handwriting synthesis and again this may be useful while implementing the spatial attention mechanism. This paper shows that an LSTM RNN is quite successful at generating both discrete and real-valued sequences using next-step prediction. The author also mentions a convolutional mechanism which is used for synthesizing realistic images of handwriting. The author also compares the situations when the bias is high and it is low. According to the experimental results, when there is high bias, it is hard to read the generated handwriting. However if the bias is decreased so much, all the written sentences will look the same with each other.

B. Future Work

1) Show, Attend and Tell: Neural Image Caption Generation with Visual Attention(Xu et al., 2015): This paper introduces a model which learns to describe the content of images using attention. The authors tell how the model can be trained using backpropagation. The model simply takes an image as input. After applying convolutional feature extraction, the output is analyzed with an RNN with attention. And after that the objects in the image are generated as word by word. For experiments, the authors used three benchmark datasets: Flickr8k, Flickr30k and MS COCO. According to the paper, the model uses reinforcement learning while training. There are some similarities between this paper and DRAW. First one is that both models take images as inputs. The second one is that both models are generative models. And the last one is that both models use attention during inference part. To find out how to implement a generative model, looking at the implementation of this paper may be useful.

2) Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks(Denton et al., 2015): This paper introduces a generative model which produces high quality samples of natural images. As it is mentioned in its title, this model uses Laplacian Pyramid framework to generate images. According to the authors, at each level of the pyramid, a separate generative model is trained using the Generative Adversarial Network approach. And authors claim that the samples produced from this model have much better quality comparing to other models. During the experiments, the authors used two models: Generative Adversarial Networks and Laplacian Generative Adversarial Networks. They used CIFAR-10 and STL datasets to test these two models. And the authors showed that by using Laplacian Pyramid, the loss value is decreased. Since this paper does a really similar job with DRAW, looking at the implementation of this paper may be useful.

IV. Methodology

Before starting to reimplement DRAW, the original paper was scanned briefly using Keshav's first pass and a summary of the paper was written. While scanning, all the terms and phrases that were not fully understood were noted. After that, the articles at the paper's related work section were examined and the most important articles cited by this paper were noted. And after examining the previous works, Google Scholar was used to find out who cited this paper and the most important follow-up papers in this area were noted. Then, a research about the datasets that would be used during the experiments was done. After that, the baselines and upper bounds the paper gave were examined. Then, the contribution of the paper to the state-of-the-art was examined. And an ablation study was done. After that, the definitions of the unknown terms that had been listed before were noted. Then, Keshav's first pass was applied to the articles that had been noted as previous work and follow-up papers. As a beginning of reimplementation, a baseline model was implemented. This model was able to read and minibatch the datasets, calculate the average loss value for the model which was not trained yet and generate random images. After that, other reimplementations of DRAW were examined. Since the simplest experiment was to generate MNIST images without using attention mechanism, this experiment was implemented first. After that, attention mechanism is added to the model. Then, the experiment MNIST Generation with Two Digits was implemented since this experiment was easy

to implement by making a few changes in the model of MNIST Generation experiment. Since MNIST images were black and white images, there were only one color channel in MNIST Generation model. To generate SVHN and CIFAR-10 images, the model was altered so that it became able to apply image filters to three color channels. After completing these 4 image generation experiments, the last experiment "Cluttered-MNIST Classification" was implemented. In the following sections, all this process will be discussed in detail.

V. Reimplementation of the Experiments

A. MNIST Generation

Since the easiest experiment among others was "MNIST Generation", it would be rational to implement it first. There are two ways of reading and writing in DRAW. The network can use attention mechanism or not. Since implementation of the model without the attention mechanism was easier, implementing it first would be more rational. By adding just a few methods, the model will be able to generate MNIST images using the attention mechanism.

1) MNIST Generation without Attention Mechanism: Before starting to discuss the details of implementation of this experiment, it is better to give all the parameters in a table(see Table III).

TABLE III
PARAMETERS FOR MNIST GENERATION WITHOUT ATTENTION MECHANISM

Abbreviation	Parameter description	Value
batch_size	Batch size	100
A, B	Image width and Image height	28
img_size	Image size	784
enc_size, dec_size	Encoder size, Decoder size	256
z_size	Number of output classes	10
T	MNIST generation steps	20
iters	Number of minibatches to train	10000
opt	Optimizer	Adam
lr	Learning rate for Adam	0.001
bt1	Beta1 for Adam	0.5
gclip	Gradient clipping for Adam	5
eps	Epsilon value	1e-8
initscale	Weight Initialization Scalar	0.05
initializerdist	Distribution for initializer	Normal(0,1)

For the implementation of this experiment, MNIST training images and test images are read by the model and minibatches of size 100 were created using these datasets. To make all minibatches store the same amount of instances, the first 16 images in training dataset and the first 8 images in the test dataset were ignored. After minibatching, we have 600 minibatches for training and 100 minibatches for testing. One important point is that first dimension of a minibatch is the batch size and the second dimension of the minibatch is the image size in my implementation. Since image size of MNIST images are 28x28, after flatting each image into second dimension, each minibatch had size (100, 784). After minibatching is completed, weights were initialized. Sizes of these weights can be found at Table IV.

TABLE IV
WEIGHT SIZES FOR MNIST GENERATION WITHOUT ATTENTION MECHANISM

Weight Abbreviation	Size
encoder_w	$(2 * (A * B + enc\_size), enc\_size * 4)$
encoder_b	(1, enc_size * 4)
mu_w	(enc_size, z_size)
mu_b	(1, z_size)
sigma_w	(enc_size, z_size)
sigma_b	(1, z_size)
decoder_w	(dec_size + z_size, dec_size * 4)
decoder_b	(1, dec_size * 4)
write_w	(dec_size, A * B)
write_b	(1, A * B)

After creating these weights with an initializer which has a distribution Normal(0,1), each array was multiplied with initscale(0.05). And then, for each array, an Adam parameter was constructed with learning rate 0.01, beta1 0.5 and gclip 5. The reason of using gradient clipping was that the gradients were too large and the model encountered exploding gradients during the training.

The formulas for the forward calculation and the loss function is given in the original paper in detailed. In this formula, there are methods named read, write, sampleQ, encode and decode. For MNIST generation without attention mechanism, at step T, read method simply concatenate the input image and image generated by the network at step T-1. Write method simply applies a linear transform to hidden state of the decoder LSTM. SampleQ method applies two linear transforms to hidden state of the encoder LSTM. Encode and decode methods call LSTMs and update the states of the DRAW network. Each of these methods are called for T times. And after T steps, reconstruction loss and latent loss are calculated using the formulas given in the original paper. Sum of these two loss values is the value that our model tries to decrease. There is one additional note: To prevent the the gradients to explode, one should add a small positive epsilon-value(1e-8) to the number which is passed into the log function in binary cross entropy.

During these T steps, all the images generated by DRAW are stored in a list named "cs". After each 100 epochs, the loss values are printed and T images are generated. One can see the effects of iterative refinement mechanism by looking at these T images.

2) MNIST Generation with Attention Mechanism: For this experiment, in addition to the parameters used in MNIST Generation without Attention Mechanism experiment, also the parameters given in Table V are used.

TABLE V
ADDITIONAL PARAMETERS FOR MNIST GENERATION WITH ATTENTION MECHANISM

Abbreviation	Parameter description	Value
read_n	Width and height of reading glimpse	5
write_n	Width and height of writing glimpse	5

And the weight sizes are given in table VI.

TABLE VI
WEIGHT SIZES FOR MNIST GENERATION WITH ATTENTION MECHANISM

Weight Abbreviation	Size
read_w	(enc_size, read_n)
read_b	(1, read_n)
encoder_w	(562, enc_size * 4)
encoder_b	(1, enc_size * 4)
mu_w	(enc_size, z_size)
mu_b	(1, z_size)
sigma_w	(enc_size, z_size)
sigma_b	(1, z_size)
decoder_w	(dec_size + z_size, dec_size * 4)
decoder_b	(1, dec_size * 4)
writeW_w	(dec_size, write_n * write_n)
writeW_b	(1, write_n * write_n)
write_w	(dec_size, write_n)
write_b	(1, write_n)

To implement the attention mechanism, after updating these parameters and initializing these weights in the given sizes, we should only change read and write methods. During reading with attention, the network chooses a glimpse and applies a filter on this glimpse. During writing with attention, the network chooses another glimpse and writes to this region of the image only. Again the images generated by the network are stored in a list named "cs". At each 100 epochs, the loss values are printed and T images are generated. One can see the effects of both iterative refinement and spatial attention mechanisms by looking at these T images.

B. MNIST Generation with Two Digits

The main objective of this experiment to have 2 MNIST digits inside one grid. So, for this experiment, nor the parameters neither the weight sizes changed. Normally at each 100 epochs, 1 image generation process was displayed as output. However for this experiment, at each 100 epochs, we calculate the loss value twice. So we get two sequence of MNIST generations at this epoch. Considering we store the first sequence in an array named cs1 and the second sequence in an array named cs2. We generate random numbers so that the images stored in cs1 and the images stored in cs2 are displayed at random locations of a grid. After that we add these two sequences for each image and and clip them so that a pixel value is not greater than one.

C. Street View House Number Generation

Since this experiment is a bit challenging, we will assume that we will use the attention mechanism to generate SVHN images. Updated parameters are given in Table VII. All other parameters are same with the experiment "MNIST Generation with Attention Mechanism".

For the implementation of this experiment, SVHN training images and test images are read by the model and mini-batches of size 100 were created using these datasets. To make all minibatches store the same amount of instances, the first 57 images in training dataset and the first 32 images in the test dataset were ignored. After minibatching, we have 732 minibatches for training and 260 minibatches for testing. As in the previous experiments, first dimension of a minibatch is the batch size and the second dimension of the minibatch is the image size in my implementation. Since image size of SVHN images are 32x32, after flatting each image into second dimension, each minibatch had size (100, 3072). After minibatching is completed, weights were initialized. Updated weight sizes are given in Table VII. All other weight sizes are same with the experiment "MNIST Generation with Attention Mechanism".

TABLE VII
UPDATED PARAMETERS FOR SVHN GENERATION

Abbreviation	Parameter description	Value
A, B	Width and height of the image	32
read_n	Width and height of reading glimpse	12
write_n	Width and height of writing glimpse	12
enc_size	Size of encoder LSTM	800
dec_size	Size of decoder LSTM	800
num_colors	Number of color channels	3

TABLE VIII
UPDATED WEIGHT SIZES FOR SVHN GENERATION

Weight Abbreviation	Size
encoder_w	(2464, enc_size * 4)
writeW_w	(dec_size, write_n * write_n * num_colors)
writeW_b	(1, write_n * write_n * num_colors)

To implement SVHN generation, we should only make changes on filter_img and write_attn methods. In MNIST generation, there were only one color channel. However in SVHN generation, there are 3 color channels. So we should change filter_img and write_attn methods so that while applying a filter during reading an image and while writing to an image, all the channels will be trained separately.

D. CIFAR-10 Generation

For this experiment we use the SVHN Generation model with a few changes in parameters and weight sizes. Updated parameters used in this experiment are given in Table IX. All other parameters are same with the "MNIST Generation with Attention Mechanism" experiment.

TABLE IX
UPDATED PARAMETERS FOR CIFAR-10 GENERATION

Abbreviation	Parameter description	Value
A, B	Width and height of the image	32
read_n	Width and height of reading glimpse	5
write_n	Width and height of writing glimpse	5
enc_size	Size of encoder LSTM	400
dec_size	Size of decoder LSTM	400
num_colors	Number of color channels	3

For the implementation of this experiment, CIFAR-10 training images and test images are read by the model and

minibatches of size 100 were created using these datasets. Since the number of training images and number of test images are both multiples of 100, there is no need to ignore any images. After minibatching, we have 500 minibatches for training and 100 minibatches for testing. As in the previous experiments, first dimension of a minibatch is the batch size and the second dimension of the minibatch is the image size in my implementation. Since image size of CIFAR-10 images are 32x32, after flatting each image into second dimension, each minibatch had size (100, 3072). After minibatching is completed, weights were initialized. Updated weight sizes are given in Table X. All other weight sizes are same with the experiment "MNIST Generation with Attention Mechanism".

TABLE X
UPDATED WEIGHT SIZES FOR CIFAR-10 GENERATION

Weight Abbreviation	Size
encoder_w	(950, enc_size * 4)
writeW_w	(dec_size, write_n * write_n * num_colors)
writeW_b	(1, write_n * write_n * num_colors)

E. Cluttered-MNIST Classification

To implement this experiment, one should make a few changes on the loss function of "MNIST Generation with Attention Mechanism" experiment and add a softmax classifier to the end of this loss function. To be more specific, in this experiment, there is no write method and there is no decoder LSTM. A glimpse is read from the image, sent to the encoder LSTM and this process is repeated for T times. After T steps, a softmax classifier is applied to the hidden state of the encoder LSTM. The model tries to decrease the softmax cost we get at the end of the process. To make the model learn more accurately, a decay in the learning rate is used. At every 1/25 epoch, learning rate was multiplied by 0.985 constant. And the minimum value that the learning rate was allowed to be assigned was 0.0001. The model stopped learning at epoch 6. At the end of epoch 6, the training accuracy was 96.968% and the test accuracy was 96.28%. So at the end of epoch 6, the test error was 3,72%. In the original paper, the test error was 3.36% for the same parameters(12x12 8 glimpses). In the original paper, there was also a result for 12x12 4 glimpses. However since the training took more than 5 days, this result could not be replicated. Parameters used in this experiment can be found in the following table:

Abbreviation	Parameter description	Value
batch_size	Batch size	100
A, B	Image width and Image height	100
enc_size,	Encoder size	256
T	MNIST classification steps	8
opt	Optimizer	Adam
lr	Initial learning rate for Adam	0.001
dc	Learning rate decay constant	0.985
bt1	Beta1 for Adam	0.5
gclip	Gradient clipping for Adam	5
initscale	Weight Initialization Scalar	0.03
initializerdist	Distribution for initializer	Normal(0,1)

VI. COMPARISON WITH THE ORIGINAL PAPER AND THE ANALYSIS OF THE RESULTS

1) MNIST Generation without Attention Mechanism

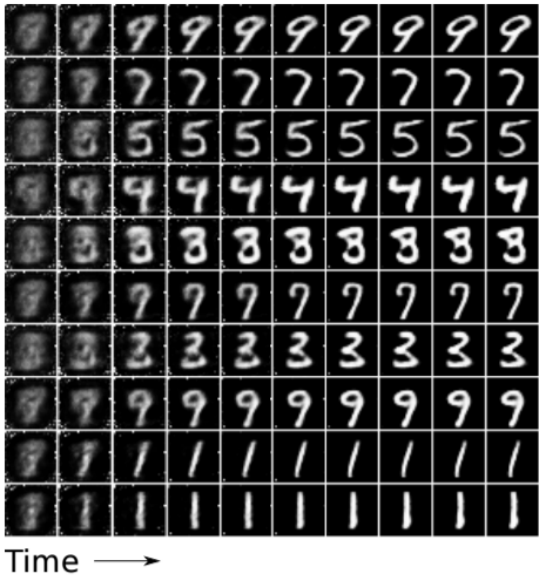

Fig. 1. A sample image generation process in the original implementation of "MNIST Generation without Attention Mechanism" experiment(Gregor et al., 2015).

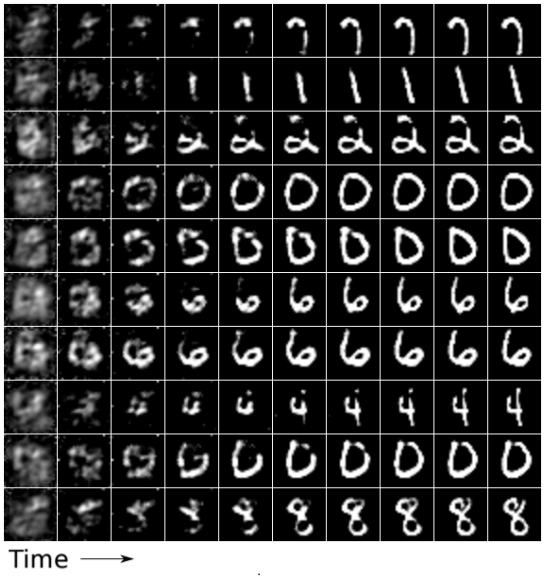

Fig. 2. A sample image generation process in my implementation of "MNIST Generation without Attention Mechanism" experiment.

As it can be seen in Fig. 1 and Fig. 2, my implementation of "MNIST Generation without Attention Mechanism" gave a similar result to the original experiment. The only difference between these two sequences is that in the original paper, the MNIST images become clear in earlier time steps.

2) MNIST Generation with Attention Mechanism

Fig. 3. A sample image generated during the original implementation of "MNIST Generation with Attention Mechanism" experiment(Gregor et al., 2015).

Fig. 4. A sample image generated during my implementation of "MNIST Generation with Attention Mechanism" experiment.

As it can be seen in Fig. 3 and Fig. 4, my implementation of "MNIST Generation with Attention Mechanism" gave a similar result to the original experiment. In the results of the experiment "MNIST Generation without Attention Mechanism", the authors preferred to show the generation sequence in their paper. So for that experiment, I showed the same process in my implementation. However the authors preferred to show the end result of "MNIST Generation with Attention Mechanism" in their paper. And this is the reason why I showed only the end result of this experiment.

3) MNIST Generation with 2 Digits

Fig. 5. A sample image generated during the original implementation of "MNIST Generation with 2 Digits" experiment(Gregor et al., 2015).

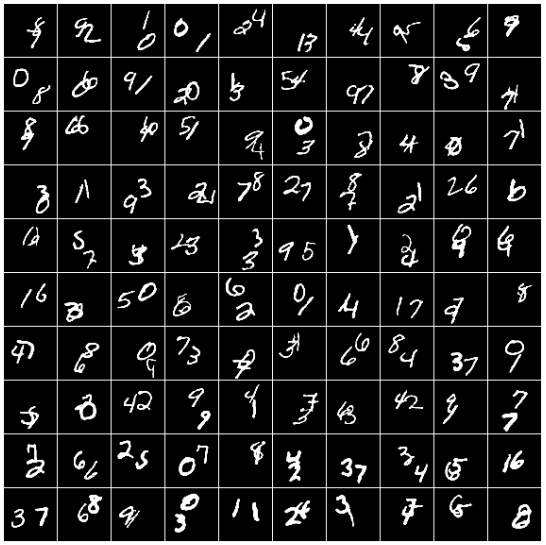

Fig. 6. A sample image generated during my implementation of "MNIST Generation with 2 Digits" experiment.

As it can be seen in Fig. 5 and Fig. 6, my model managed to draw two MNIST images at random locations in a single grid. By comparing these two figures, it can be said that my implementation of "MNIST Generation with 2 Digits" experiment is a successful replication of the original one.

4) Street View House Number Generation

Fig. 7. A sample image generated during the original implementation of "SVHN Generation" experiment(Gregor et al., 2015).

Fig. 8. A sample image generated during my implementation of "SVHN Generation" experiment.

As it can be observed in Fig. 7 and Fig. 8, my implementation of SVHN Generation experiment did not result as good as the original one. The numbers can be read easily, however they do not look clear as the numbers in the original experiment. During training, the authors made DRAW look for 32 glimpses per image. However since backpropagation of these 32 steps made the training process took too long, in my implementation I made DRAW look for only 20 glimpses per image. And this resulted a difference between these two the results. One additional note is that, while plotting the images in my implementation, I summed all the pixels with 1, and then divided the result with 2. This may explain the poor contrast in my results.

5) CIFAR-10 Generation

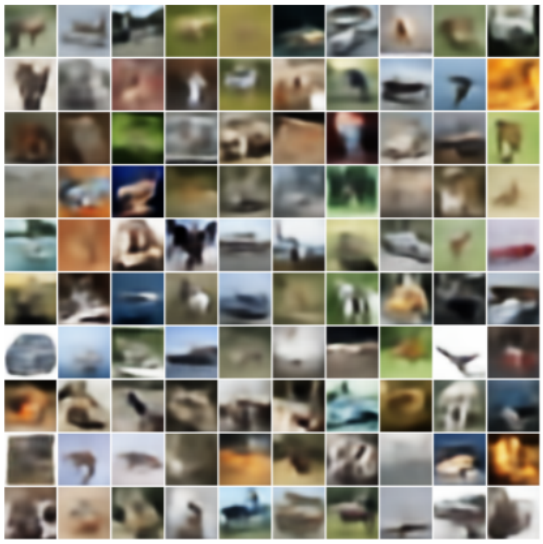

Fig. 9. A sample image generated during the original implementation of "CIFAR-10 Generation" experiment.

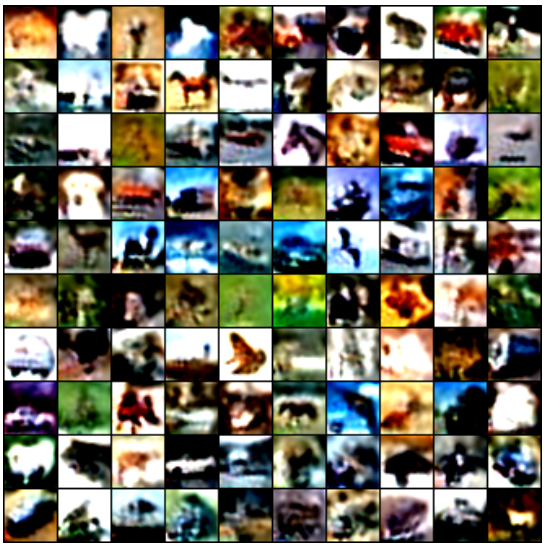

Fig. 10. A sample image generated during my implementation of "CIFAR-10 Generation" experiment.

As it can be observed in Fig 9. and Fig 10. both my implementation and the original implementation of the CIFAR-10 Generation experiment gave blurry results. However there is a contrast difference between two results. Most probably the reason is that again I summed all the pixels with 1, and then divided the result with 2 in my implementation. Despite this contrast difference, one can easily recognize some of the objects that my implementation plots. It can also be said that, DRAW network managed to capture much of the shape, color and composition of real photographs(Gregor et al., 2015).

6) Cluttered MNIST Classification

TABLE XI
COMPARISON OF CLUTTERED-MNIST CLASSIFICATION MODELS IN TERMS OF THEIR TEST ERRORS

Model	Error
Convolutional, 2 layers	14.35%
RAM, 4 glimpses, 12x12, 4 scales	9.41%
RAM, 8 glimpses, 12x12, 8 scales	8.11%
Differentiable RAM, 4 glimpses, 12x12	4.18%
My implementation of DRAM, 8 glimpses, 12x12	**3.72%**
Differentiable RAM, 8 glimpses, 12x12	3.36%

As it can be see in the table, in the original paper, the test error was 3.36% for 8 glimpses of size 12x12 per image. In my implementation, the test error was 3.72% for 8 glimpses of size 12x12 per image. In the original paper, there was also a result for 12x12 4 glimpses. However since the training took more than 5 days for the parameters 12x12 and 8 glimpses, this result could not be replicated.

VII. LOSS AND ACCURACY GRAPHS

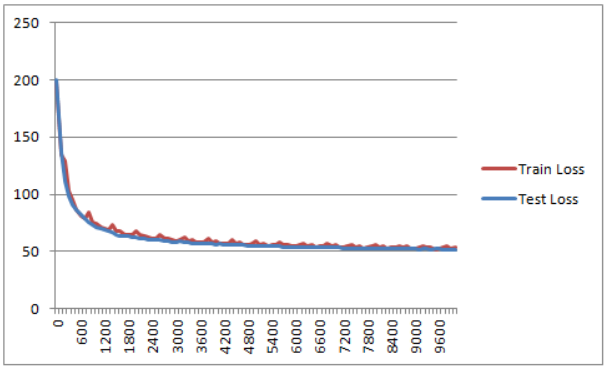

Fig. 11. Training and test loss values during "MNIST Generation without Attention Mechanism" experiment.

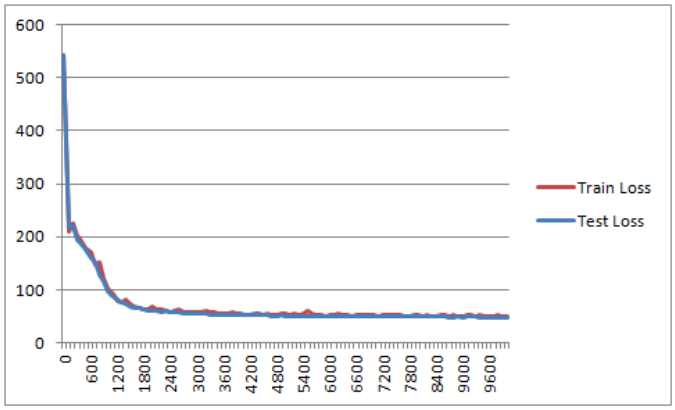

Fig. 12. Training and test loss values during "MNIST Generation with Attention Mechanism" experiment.

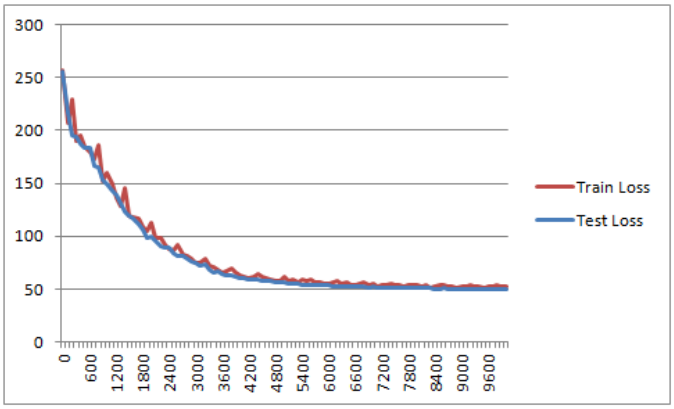

Fig. 13. Training and test loss values during "MNIST Generation with 2 Digits" experiment.

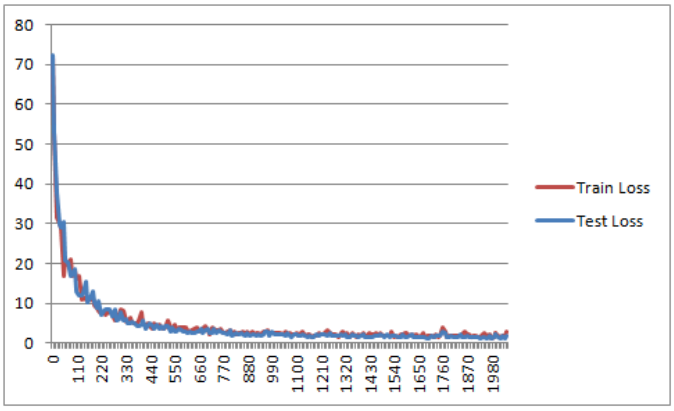

Fig. 14. Training and test loss values during "SVHN Generation" experiment.

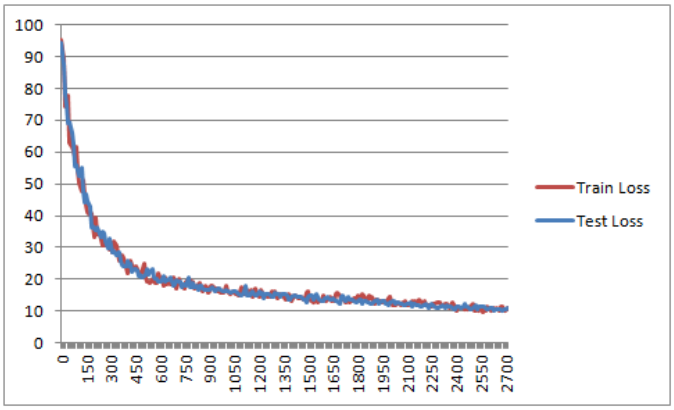

Fig. 15. Training and test loss values during "CIFAR-10 Generation" experiment.

As it can be seen in Fig. 15, CIFAR-10 Generation experiment in my implementation did not converge yet. The reason is that training DRAW with 2700 minibatches with a GPU took more than 4 days and most probably it would have taken 4 more days to converge.

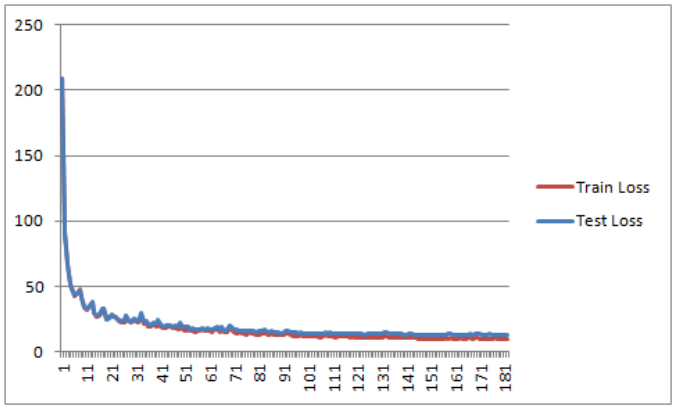

Fig. 16. Training and test loss values during "Cluttered-MNIST Classification" experiment.

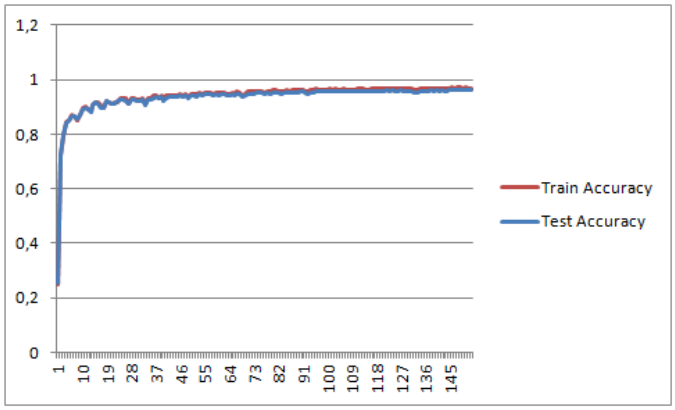

Fig. 17. Training and test accuracy values during "Cluttered-MNIST Classification" experiment.

VIII. Contributions

The only experiment where the accuracies could be calculated was "Cluttered MNIST Classification". In that experiment, my implementation gave a test error slightly higher than the test error in the original implementation. Since all other experiments were made by using a generative model, there were only loss values to be calculated. However the loss values that the authors found and the loss values that I found do not seem comparable. This is most probably because of some differences in these two implementations. However before I replicated this work, there were replications in programming languages Python and Lua. So my single contribution is that my implementation became the first replication of this paper in Julia language. The implementation of DRAW in Julia can be found in the following repository: https://github.com/ardarslan/draw. All the datasets except Cluttered-MNIST dataset are downloaded automatically by the Julia code I provided. Cluttered-MNIST dataset can be found at goo.gl/4GJbSP.

References

[1] Denton, Emily L., Soumith Chintala, and Rob Fergus. "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks." Advances in neural information processing systems. 2015.

[2] Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

[3] Gregor, Karol, et al. "Deep autoregressive networks." arXiv preprint arXiv:1310.8499 (2013).

[4] Gregor, Karol, Danihelka, Ivo, Graves, Alex, Rezende, Danilo Jimenez, and Wierstra Daan. "DRAW: A Recurrent Neural Network for Image Generation." ICML, 2015.

[5] Jang, Eric. "Understanding and Implementing Deepmind's DRAW Model." N.p., 25 June 2016. Web. 30 Apr. 2017. <http://blog.evjang.com/2016/06/understanding-and-implementing.html >.

[6] Mnih, Volodymyr, Heess, Nicolas, Graves, Alex, et al. Recurrent Models of Visual Attention. In Advances in Neural Information Processing Systems, pp. 2204 − 2212, 2014.

[7] Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International Conference on Machine Learning. 2015.