Deep Recurrent Attentive Writer
CmpE597 Final Project

Mine Melodi Çalışkan [1]
Master Student
Department of Computational Science and Engineering
Bogazici University
Bebek, Istanbul 34432 Turkey

May 28, 2018

[1] minemelodicaliskan@gmail. com

Introduction

In this project deep recurrent attentive writer (DRAW) [1] is implemented on *Quick, Draw!* data set [2]. Draw network is a recurrent auto encoder that uses attention mechanisms. Attention mechanism focus on a small part of the input data in each time step, and iteratively generates image that is closer to the original image. The network is trained with stochastic gradient descent and the objective function is variational upper bound on the log likelihood of the data.

Architecture

Draw network augments the encoder and decoder with recurrent networks. The encoder determines a distribution over latent variables that contains salient information about the input data and the decoder takes samples from this distribution and uses them to condition its distribution. The inference and generation defined by a sequential process. An attention mechanism is added over the input to define this sequential process. Attention mechanism restricts the input region read by the encoder and the output region written by the decoder. LSTMs are used for the encoder and decoder. To generate images, samples are picked from the latent layer based on a prior. These samples are then fed into the decoder. That is repeated for several time steps until the image is finished.

Data Generation

The encoder receives at each time step the image and the output of the previous decoding step. The hidden layer in between encoder and decoder is a distribution $Q(Z_t|h_t^{enc})$ which is a diagonal gaussian. The mean and standard deviation of that gaussian is derived from the encoder's output vector with a linear transformation. Using a gaussian instead of a bernoulli distribution enables the use of the reparameterization trick [4]. The reparametrization trick enables backpropagation algorithm straightforward. At every time step encoder and decoder generates a vector, and the vector generated by the decoder is added to image *canvas*. Fig. 1 show the architecture of the Draw.

Training steps can be summarized as following steps:

1. Encoding:
   (i) $\hat{x} = x - \sigma(c_{t-1})$
   (ii) $r_t = read(x_t, \hat{x}_t, h_{t-1}^{dec})$
   (iii) $h_t^{enc} = RNN^{enc}(h_{t-1}^{enc}, [r_t, h_{t-1}^{dec}])$
2. Sampling: $z_t \sim Q(Z_t|h_t^{enc})$
3. Decoding: $h_t^{enc} = RNN^{dec}(h_{t-1}^{dec}, z_t)$
4. Output: $c_t = c_{t-1} + write(h_t^{dec})$

where $\sigma(x)$ is the logistic sigmoid function and the latent distribution is taken as a diagonal gaussian $\mathcal{N}(Z_t|\mu_t, \sigma_t)$ where

$$\mu_t = W(h_t^{enc})$$
$$\sigma_t = \exp^{W(h_t^{enc})}$$

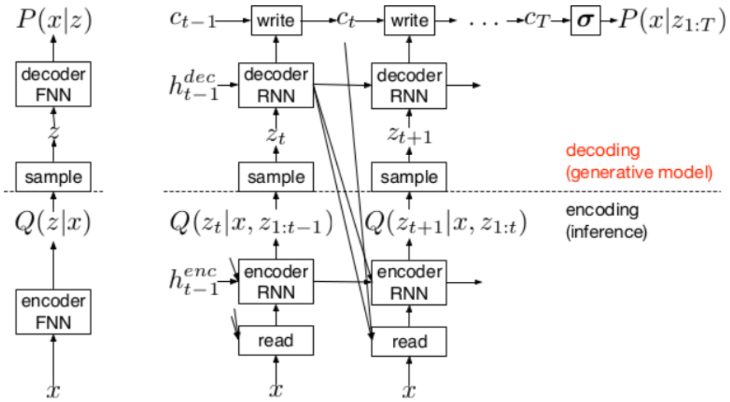


Figure 1: Architecture of the Draw. [1]

Attention

In draw a selective attention model is used to decide which parts to read of the image. These are called *glimpses*. Each glimpse is defined by its center (x, y), its stride δ, its gaussian variance σ^2 and a scalar multiplier γ. Stride value determines zoom level, scalar multiplier scales the intensity of the glimpse results and the higher the gaussian variance the more blurry is the result. These parameters are calculated based on the decoder output using a linear transformation. For an $N \times N$ glimpse it takes $N \times N$ grid of gaussian filters. The center pixel of each gaussian is then used as the respective output pixel of the glimpse. The glimpse parameters are generated from the decoder output in both cases.

Given an image of size $A \times B$, we place $N \times N$ grid of gaussian filters positioned on image with grid center (g_X, g_Y) with stride δ and the mean location of the filter is given as:

$$\mu_X^i = g_X + (i - \frac{N}{2} - 0.5)\delta \tag{1}$$

$$\mu_Y^i = g_Y + (i - \frac{N}{2} - 0.5)\delta \tag{2}$$

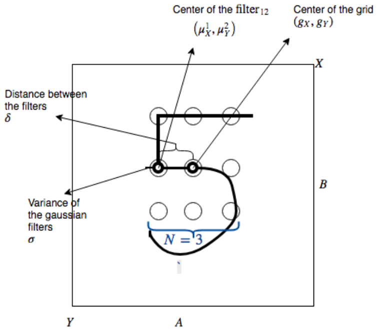

Center of the filter$_{12}$
(μ_X^1, μ_Y^2)
Center of the grid
(g_X, g_Y)
X
Distance between the filters
δ
Variance of the gaussian filters
σ
B
$N = 3$
Y
A

Figure 2: $N \times N$ grid of gaussian filters.

The read attention parameters are determined by a linear transformation of h_{dec} as follows:

$$(\tilde{g}_X, \tilde{g}_Y, \log \sigma^2, \log \tilde{\delta}, \log \gamma) = W(h^{dec})$$

Here log terms of σ, γ computed to guarantee $\sigma = \exp(\log(\sigma))$ and $\sigma = \exp(\log(\gamma))$ to be positive.

$$g_X = \frac{A+1}{2}(\tilde{g}_X + 1)$$

$$g_Y = \frac{B+1}{2}(\tilde{g}_Y + 1)$$

$$\delta = \frac{\max(A, B) - 1}{N - 1}\tilde{\delta}$$

where A, B are width and height of the image respectively.

Using the parameters above and normalization factors Z_X and Z_Y, the horizontal and vertical filter bank matrices are computed.

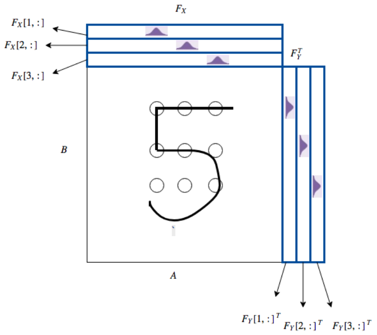
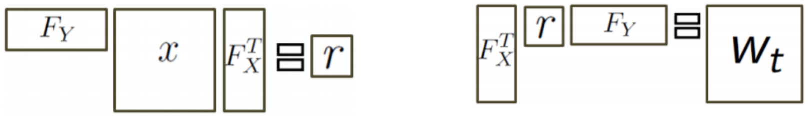


Figure 3: Horizontal and vertical filters.

1. Horizontal filter bank matrix:

$$F_X[i, a] = \frac{1}{Z_X}\exp(-\frac{(a - \mu_X^i)^2}{2\sigma^2})$$



(a) Read attention.

(b) Write attention.

2. Vertical filter bank matrix:

$$F_Y[j,b] = \frac{1}{Z_Y}\exp(-\frac{(b-\mu_Y^j)^2}{2\sigma^2})$$

In the above equations (i,j) is a point in attention patch, and (a,b) is a point in input image. The extracted value for attention grid pixel (i,j) is the convolution of the image with a $2D$ gaussian whose x and y components are the respective i and j rows in F_X and F_Y.

Read attention and write attention can be formulated as follows:

1. Read: $read(x,\hat{x},h_{t-1}^{dec}) = \gamma[F_Y x F_X^T, F_Y \hat{x} F_X^T]$

2. Write: $write(h_t^{dec}) = \frac{1}{\gamma}[\hat{F}_Y^T w_t \hat{F}_X]$

Loss Functions

Generative Loss

The loss function of the decoder is the negative log likelihood of the image given in the final canvas matrix : $-\log D(x|c_t)$, where x is the input image and c_t is the final output image of the autoencoder. D is a bernoulli distribution if the image is binary.

Latent Loss

The loss function of the latent layer is the Kullback-Leibler (KL) divergence [5] between that layer's gaussian distribution and a prior, summed over the timesteps.

$$L_z = \sum_{t=1}^{T} KL(Q(Z_t|h_t^{enc})\|P(Z_t))$$

where Z_t = latent layer, h_t^{enc} = result of encoder and a prior $P(Z_t)$.

The KL-divergence loss term measures the difference between the distribution of the latent vector z, to that of an independent and identically distributed gaussian vector with zero mean and unit variance. Optimizing for this loss term

Figure 5: Sample figure of Quick, Draw! data set.

allows to minimizing following difference. Assuming $P(Z_t)$ to be $\mathcal{N}(0,1)$, and using the result in [4]:

$$L_z = \frac{1}{2}\left(\sum_{t=1}^{T} \mu_t^2 + \sigma_t^2 - \log \sigma_t^2\right) - \frac{T}{2}$$

Total Loss

The total loss is the expectation value of the sum of latent and generative loss.

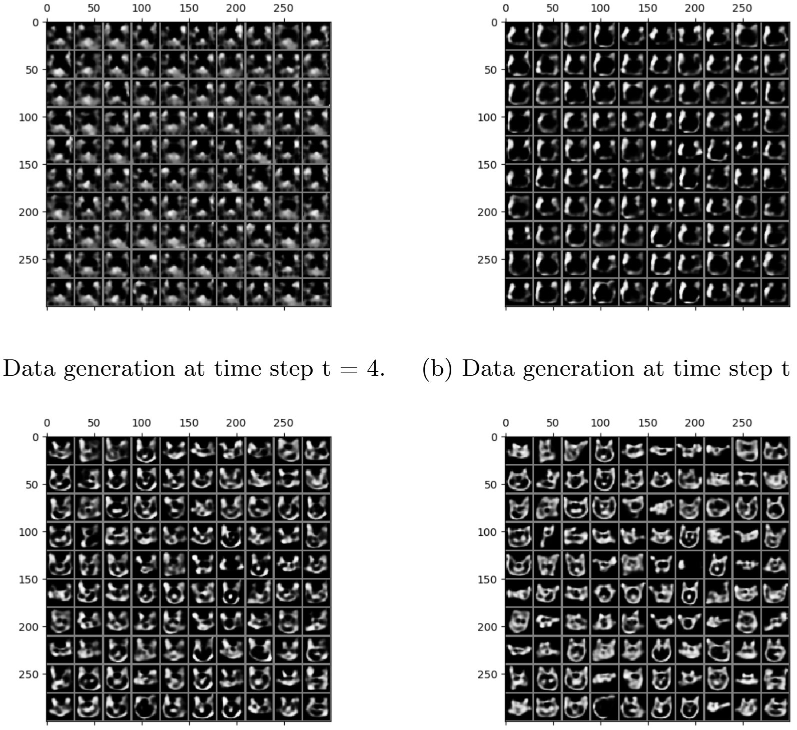
Experiments

Data

Experiments are based on two categories and their mix from Quick, Draw! dataset [2] which is constructed from 50 million vector drawings obtained from Quick, Draw! [3], an online game where the players are asked to draw objects belonging to a particular object class in less than 20 seconds. Fig.5 shows some samples from the data set. For the model I use preprocessed version as Numpy bitmaps, and then converted each image into MNIST data set format.

Results

The results were reported in Table 1. Maximum iteration step and initial learning rate are fixed to 10^4 and $1e-3$ respectively. As an optimizer RMSprop is used, and gradients are clipped to avoid overshooting. Loss results are calculated on data set both the reconstruction loss and the latent loss. So, for cat category decreasing batch size from 100 to 64 improved the reconstruction score from 166.68 to 146.50. We observed that increasing the fixed number of time steps for image generation T usually gives better results, e.g for dog data set the first result using $T = 10$ was 171.14 for reconstruction loss, doubling T decreased this error sufficiently so that generated sequential image looks similar to the original image. Fig. 7 shows 4 generation steps for dog images. It can be observed from Fig. 8a that this modification also increased the speed of



(a) Data generation at time step t = 4.

(b) Data generation at time step t = 6.

(c) Data generation at time step t = 7.

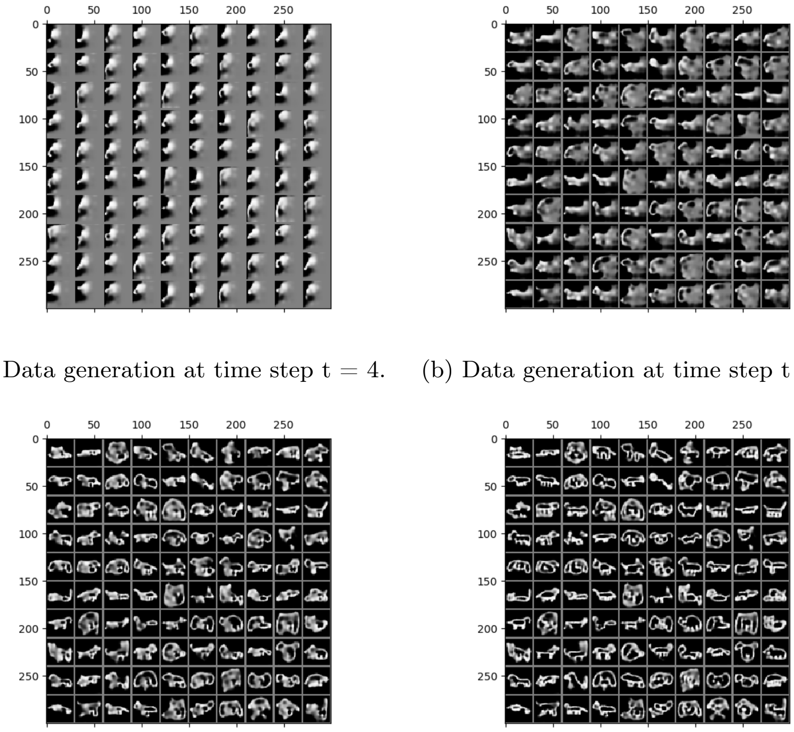
(d) Data generation at time step t = 9.

Figure 6: Generated samples for cat category in Quick, Draw! data set.

learning. Smaller patch size for read attention decreased the performance, an example result can be seen in Table 1 for “Cat & Dog” category.

Conclusion

In this project we implemented a recurrent neural network model for image generation, DRAW for Quick, Draw! data set. We applied solutions to the difficulties we encountered by tuning hyper parameters of the model so that final results show promising results in the sense of “ natural” sequential image generation and also with respect to final reconstructed image.



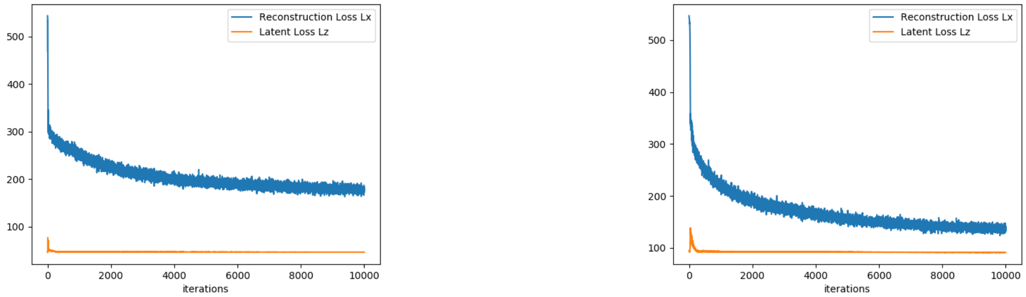
(a) Data generation at time step t = 4.

(b) Data generation at time step t = 8.

(c) Data generation at time step t = 15.

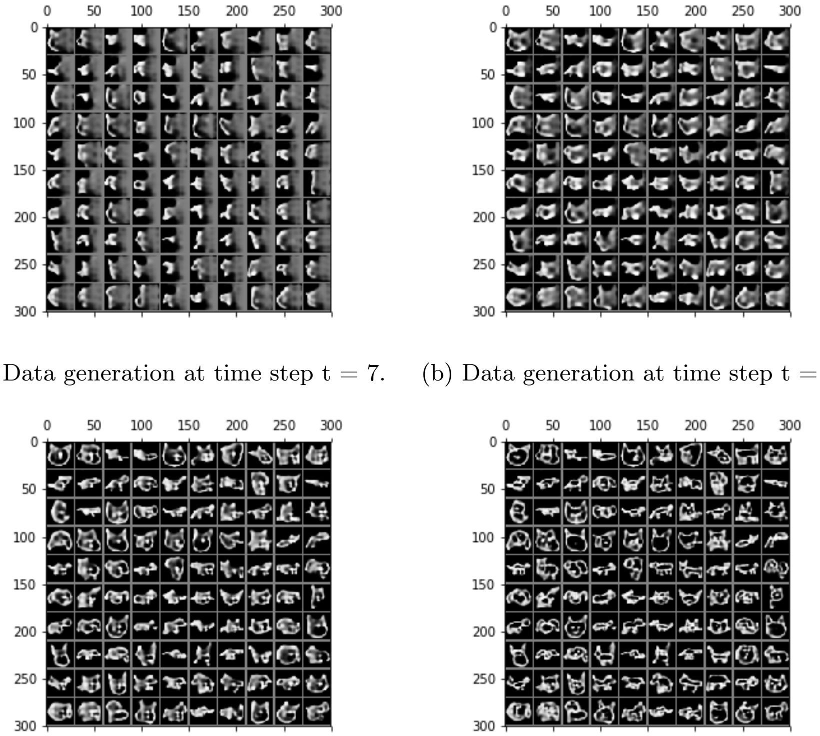
(d) Data generation at time step t = 20.

Figure 7: Generated samples for dog category in Quick, Draw! data set.



(a) Reconstruction and latent loss using 10 iteration for dog category during training.

(b) Reconstruction and latent loss using 20 iteration for dog category during training.



(a) Data generation at time step t = 7.

(b) Data generation at time step t = 9.

(c) Data generation at time step t = 15.

(d) Data generation at time step t = 20.

Figure 9: Generated samples for cat and dog categories together in Quick, Draw! data set.

Category	Lx	Lz	read	write	z size	T	batch
Cat	**146.501862**	92.220596	5	5	10	20	64
Cat	166.688004	91.576004	5	5	10	20	100
Dog	171.142306	92.418405	5	5	10	10	100
Dog	**158.912208**	91.2177324	5	5	10	20	100
Cat& Dog	181.984634	91.815247	2	5	10	20	100
Cat& Dog	**141.932007**	91.562057	5	5	10	20	100

Table 1: Experimental Hyper-parameters and Test Losses.

Bibliography

[1] Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. *DRAW: A Recurrent Neural Network For Image Generation.* Google DeepMind, arXiv:1502.04623v2 [cs.CV] 20 May 2015.

[2] Ha, D., & Eck, D. *A Neural Representation of Sketch Drawings.* ICLR 2018, arXiv:1704.03477

[3] Jongejan, J., Rowley, H., Kawashima, T., Kim, J., & Fox-Gieg, N. *The Quick, Draw!.* `https://quickdraw.withgoogle.com/`

[4] Kingma,D. P., & Welling, M. *Auto-Encoding Variational Bayes.* Proceedings of the 2nd International Conference on Learning Representations (ICLR), ArXiv:1312.6114v10, 1 May 2014.

[5] Kullback, S. *The Kullback-Leibler Distance.* The American Statistician, Vol. 41, No. 4. (1987), pp. 340-341.