Probabilistic Graphical Models Final Project
DRAW: A Recurrent Neural Network For Image Generation

Francis de Ladurantaye Younes Driouiche Maximilien Le Clei

Abstract

Most popular approaches to image generation aim to generate entire scenes in a single time step. Artists, on the other hand, tend to draw over time, which has the added advantage of allowing them to make adjustments to their canvas throughout the process.

Additionally, very high resolution images are quite cumbersome to generate all at once. A more scalable approach is to mimic the way people create art, namely by only paying attention to a certain section of the canvas and focusing on refining this part only.

DRAW takes those two considerations into account and extends the standard Variational Autoencoder model by supplementing it with both a **temporal structure** and an **attention structure**.

1. Network Architecture

1.1. Overview

At its core, DRAW combines a Variational Autoencoder to Recurrent Neural Networks.

The two main distinctions with regular Variational Autoencoders are:

- The use of Recurrent Neural Networks to represent the encoder and decoder networks (as opposed to regular Feedforward Neural Networks).
- The **read** and **write** operations.

Those two distinctions are the components behind the added temporal and attention structure.

1.2. Main process

DRAW introduces the notion of a canvas c which is iteratively drawn upon at every time step.

As depicted on Figure 1, at each time step t during training, the encoder **reads** an input image x, the latest version of the canvas c_{t-1}, and the decoder's previous output h_{t-1}^{dec} which it uses to parameterize a distribution $Q(z_t|x, z_{1:t-1})$. Latent codes z_t are then sampled from that distribution and fed to the decoder whose output is used to **write** on the canvas. Equations 1 to 6 explain this process more formally.

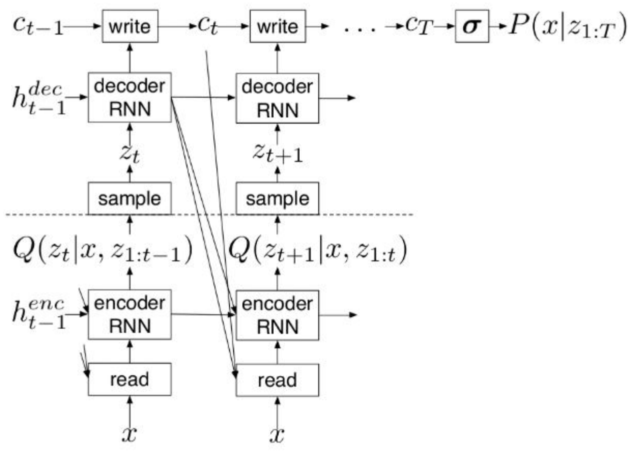

Figure 1. The DRAW Network

$$\hat{x}_t = x - \sigma(c_{t-1}) \tag{1}$$
$$r_t = read(x_t, \hat{x}_t, h_{t-1}^{dec}) \tag{2}$$
$$h_t^{enc} = RNN^{enc}(h_{t-1}^{enc}, [r_t, h_{t-1}^{dec}]) \tag{3}$$
$$z_t \sim Q(Z_t|h_t^{enc}) \tag{4}$$
$$h_t^{dec} = RNN^{dec}(h_{t-1}^{dec}, z_t) \tag{5}$$
$$c_t = c_{t-1} + write(h_t^{dec}) \tag{6}$$

where $\hat{x}_t$ is the error image, $[r_t, h_{t-1}^{dec}]$ is the concatenation of r_t and h_{t-1}^{dec} and σ is the sigmoid function.

RNN^{dec} and RNN^{enc} represent the encoder and decoder networks. The Recurrent Neural Networks in our case are Long-Short Term Memory networks. They were chosen by the authors of the paper due to their ability to better handle long-range time dependencies.

Equations 7 to 12 demonstrate that transformation.

$$\hat{x}_t = x - \sigma(c_{t-1}) \tag{7}$$

$$r_t = read(x_t, \hat{x}_t, h_{t-1}^{dec}) \tag{8}$$

$$h_t^{enc}, cell_t^{enc} = LSTM^{enc}(h_{t-1}^{enc}, cell_{t-1}^{enc}, [r_t, h_{t-1}^{dec}]) \tag{9}$$

$$z_t \sim Q(Z_t | h_t^{enc}) \tag{10}$$

$$h_t^{dec}, cell_t^{dec} = LSTM^{dec}(h_{t-1}^{dec}, cell_{t-1}^{dec}, z_t) \tag{11}$$

$$c_t = c_{t-1} + write(h_t^{dec}) \tag{12}$$

1.3. Parameterizing the latent distribution

As mentioned earlier, $Q(z_t|x, z_{1:t-1})$ is a distribution over latent codes z_t. It is chosen to be a diagonal Gaussian distribution for its simplicity. Its parameters μ_t and σ_t are learned through a linear mapping from the encoder's output h_t^{enc}.

$$\mu_t = W(h_t^{enc}) \tag{13}$$

$$\sigma_t = \exp(W(h_t^{enc})) \tag{14}$$

In addition to their simplicity, Gaussian distributions have the added advantage of being able to make use of the reparameterization trick. This trick is essential for backpropagation to work when stochastic processes are involved.

2. Loss function

DRAW's loss function required to train the network is very similar to a regular Variational Autoencoder loss function in the sense that it is also made up of both a reconstruction loss L^x and a latent loss L^z

The reconstruction loss L^x measures how well the decoder network has learned to reconstruct the input image x. We wish to minimize the difference between x and the normalized final canvas $\sigma(c_T)$.

$$L^x = -\log D(x|c_T) = \text{BCE}(x, \sigma(c_T)) \tag{15}$$

The latent loss L^z measures how different the estimated posterior Q is to a simple prior P ($N(0,1)$). We want the distribution emitted by the encoder network Q to be as similar as possible to the sampling distribution P that we will use during generation. In some sense, this loss acts as a regularizer upon our distribution Q, forcing it to take the shape of P.

$$L^z = \sum_{t=1}^{T} \text{KL}(Q(Z_t|h_t^{enc})||P(Z_t)) \tag{16}$$

The total loss L is then simply the empirical risk of the sum of the two losses for each mini-batch.

$$L = \langle L^x + L^z \rangle_{z \sim Q} \tag{17}$$

3. Image generation

We now draw samples from $P(Z_t)$ instead of $Q(z_t|x, z_{1:t-1})$. This makes sampling independent of the encoder network. The reconstruction loss is therefore only a function of the decoder network and not the encoder network. Was this not to be the case, the reconstruction loss would be much harder to optimize as the sampling distribution would change whenever the encoder network gets trained.

As depicted in the equations below, we start with an empty canvas c_0. A latent code z_t is drawn at every time step from the prior distribution $P(Z_t)$, and fed to the decoder as input. The output of the decoder is then used to write on the canvas. After the end of this process at time step T, the generated image is considered as sample from a distribution $D(X|c_T)$ parameterized by c_T.

For each time step t,

$$z_t \sim P(Z_t) \tag{18}$$

$$h_t^{dec} = \text{RNN}^{dec}(h_{t-1}^{dec}, z_t) \tag{19}$$

$$c_t = c_{t-1} + write(h_{dec}^{t}) \tag{20}$$

Finally, after T time steps,

$$\tilde{x} \sim D(X|c_T) \tag{21}$$

4. Read and Write Operations

As mentioned previously, the **read** and **write** operations are the components responsible for the attention structure of DRAW. The next sections discuss those two operations.

4.1. Attention-less read and write

The implementation of read and write operations without attention is straightforward. The read operation gives the encoder its input, while the write operation takes care of how to use the decoder's output to write on the canvas.

Reading is simply defined as:

$$read(x, \hat{x}, h_{t-1}^{dec}) = [x, \hat{x}] \tag{22}$$

The decoder's output from the previous time step is not used by the operation, yet we give it as a parameter to be consistent with the attention counterpart of the read operation.

On the other hand, the write operation applies a linear transformation W to the output of the decoder as follows:

$$write(h_t^{dec}) = W(h_t^{dec}) \tag{23}$$

The results of this operation is used to update the canvas as shown in equation 12.

4.2. Attention-based read and write

The attention read and write operations are relatively harder to implement.

To give the network the ability of attention, a $N \times N$ grid of Gaussian filters is applied to the image. To decide where and how to set this attention window, the network has to use some more parameters. Firstly, regarding the Gaussian filters, a variance parameter σ^2 will be needed as well as an intensity parameter γ to scale the filter response. Gaussian filters are generated using a Gaussian kernel, hence the name. Secondly, for the grid of Gaussian filters, a pair of coordinates (g_X, g_Y) will specify its center while a stride parameter δ will determine the size of the squares it contains.

Considering that our image x is of size $A \times B$, all parameters are determined at each time step by applying a linear transformation to the decoder's output and then scaling them appropriately:

$$(\tilde{g}_X, \tilde{g}_Y, \log \sigma^2, \log \tilde{\delta}, \log \gamma) = W(h_{t-1}^{dec}) \tag{24}$$

$$g_X = \frac{A+1}{2}(\tilde{g}_X + 1) \tag{25}$$

$$g_Y = \frac{B+1}{2}(\tilde{g}_Y + 1) \tag{26}$$

$$\delta = \frac{max(A, B) - 1}{N - 1}\tilde{\delta} \tag{27}$$

Looking at equation 24, the first thing we observe is that some of the parameters are given in log-scale by the linear mapping. The reason behind this choice is that we want the variance, stride and intensity to be positive.

Now that we have the centers of Gaussian filters and the stride, we generate the center location μ_X^i and μ_Y^j of the filter at row i and column j as shown below. Note that the centers and δ are real-valued.

$$\mu_X^i = g_X + (i - N/2 - 0.5)\delta \tag{28}$$

$$\mu_Y^j = g_Y + (j - N/2 - 0.5)\delta \tag{29}$$

The next step is to compute the horizontal and vertical filterbank matrices F_X and F_Y of size $N \times A$ and $N \times B$ respectively.

The following equations describe how to generate the filterbanks:

$$F_X[i, a] = \frac{1}{Z_X} \exp\left(-\frac{(a - \mu_X^i)^2}{2\sigma^2}\right) \tag{30}$$

$$F_Y[j, b] = \frac{1}{Z_Y} \exp\left(-\frac{(b - \mu_Y^j)^2}{2\sigma^2}\right) \tag{31}$$

In these two equations, Z_X and Z_Y are normalization constants to ensure that F_X sums to one for every i and F_Y sums to one for every j. In practice, we noticed that in early stages of training, those filters are mostly zeros. To avoid division by zero while normalizing, we add a constant ϵ with value $1 \times e^{-8}$ to the normalization constants Z_X and Z_Y to ensure numerical stability.

With the filterbanks defined, we can describe the attention **read** and **write** operations. Here again, the read operation applies a concatenation on the input and error images, but only by applying attention to them. This process consists of computing matrix products of these images with the filterbanks as follows:

$$read(x, \hat{x}, h_{t-1}^{dec}) = \gamma[F_X x F_Y^T, F_X \hat{x} F_Y^T] \tag{32}$$

As you can see, the result of the concatenation is appropriately scaled by the parameter γ. The argument h_{t-1}^{dec} is given as a parameter to the read function while not being explicitly used by the operation, because the read operation will need this parameter to construct the Gaussian filterbanks.

in the original paper of DRAW, Equation 32 was defined as follows:

$$read(x, \hat{x}, h_{t-1}^{dec}) = \gamma[F_Y x F_X^T, F_Y \hat{x} F_X^T] \tag{33}$$

This definition raises an error if the input image doesn't have dimensions A and B of equal value. Indeed, we have $F_Y x F_X^T$ that yields matrix products with dimensions $(N \times B) \times (A \times B) \times (A \times N)$, which does not work when $A \neq B$.

The attention write operation is relatively simpler and cheaper than the attention read operation as it does not require any concatenation and we apply attention to only one input w_t. Remember that the attention-less write simply consisted of applying a linear transformation to the output of the decoder, as shown in equation 23. With attention, the write operation still applies a linear transformation, but, similarly to what was done in the attention read, it modifies the result with filterbanks before producing its output.

$$w_t = W(h_t^{dec}) \tag{34}$$

$$write(h_t^{dec}) = \frac{1}{\hat{\gamma}} \hat{F}_X^T w_t \hat{F}_Y \tag{35}$$

It is also very important to mention that the attention parameters in the write operation are distinct from the ones used in the read operation. In other terms, all linear transformations W are distinct, but are all called W for simplicity. The hat ˆ symbol was used over the attention filterbanks to distinguish them from the read operation filterbanks, same for the gamma parameter.

The problem mentioned in the original DRAW paper for the read operation also arises in their version of equation 35. Regardless of the dimensions of the input image, the result w_t of the linear transformation always has dimensions $N \times N$. After applying the write operation, we get an output of size $B \times A$, while we want to construct an image of size $A \times B$. In our implementation, those problems are solved.

One last thing we want to show is that the read operation produces a result with dimensions $N \times 2N$ ($N \times N$ for the input image as well as the error image), while the write operation produces a result with dimensions $A \times B$, allowing the network to focus it's attention on parts of the input image as intended. This also means that even if the network reads a small patch of the image, it still modifies the full image when writing, but the pixels outside the patch will be modified in smaller proportions.

For RGB images, the same filters are used for every channel of the image.

5. Experimental Results

The results of our experiments with the network are based on three datasets presented in the next subsections. We did not use the CIFAR-10 dataset because the results are not that impressive from what the original paper showed. Instead, we used Fashion MNIST. The implementation of the model was done in Python with the help of PyTorch.

5.1. MNIST Dataset

The MNIST dataset is well-known for being a toy dataset in machine learning, available to test a large variety of networks and techniques. Due to its simplicity, it is often used as a benchmark when testing algorithms in early stages. For these reasons, it was used in the original DRAW paper to show the potential of the DRAW architecture, and we decided to do the same when we implemented it ourselves.

5.1.1. Without attention

As we could expect from an architecture based on a VAE, even in its simpler implementation, DRAW gave good results on the handwritten digits of MNIST. What surprised us is that we actually got slightly better results than what was presented in the original paper. Our results are shown in figure 2.

5.1.2. With attention

The results obtained while using the attention mechanism were better than the ones without. Here we clearly see the attention mechanism in action, the network focusing only on a small patch of the image at a time to refine it as it passes. It allowed the network to better handle curves, resulting in less shaky lines in some spots of the generated digits.

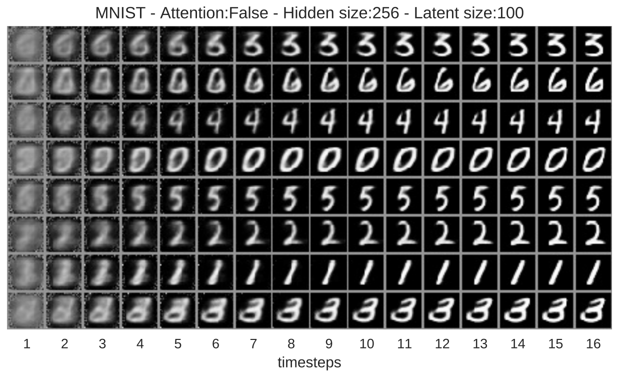

Figure 2. Results on MNIST without attention - hidden size is 200 and latent size is 256

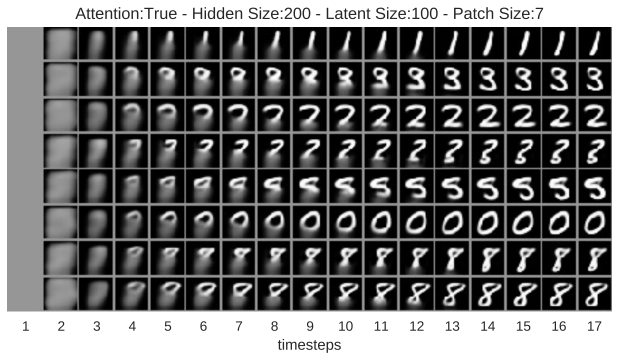
Attention:True - Hidden Size:200 - Latent Size:100 - Patch Size:7
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
timesteps

Figure 3. Results on MNIST with attention

Playing with different values of the hyper-parameters, we discovered that decreasing the patch size under 7×7 started to give us bad results, as well as increasing too much the dimension of latent codes (z = 200+). In the first case, we think this is because when the patch size is too small, it becomes too hard for the network a get the 'big picture' of the image it is generating. In the second case, when there are too many dimensions, the network seems to not be able to take all of them into account correctly. Variational Autoencoders are known to use only a subset of the dimensions of z. In that case our hypothesis is that the extra latent codes are producing noise and therefore decreasing the quality of the generated images.

As an example of results generated with a too small patch size, see the figure below where we see that the network gets a hard time following its course in the proper way.

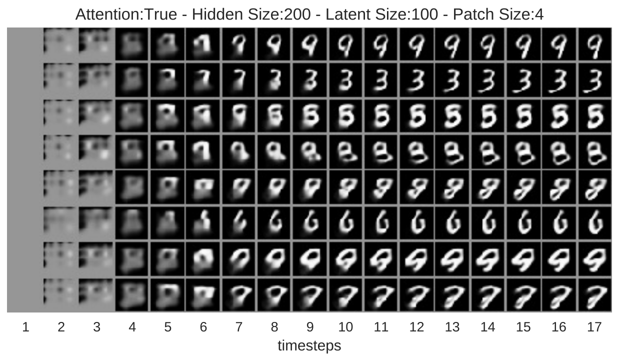
Attention:True - Hidden Size:200 - Latent Size:100 - Patch Size:4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
timesteps

Figure 4. Results on MNIST with attention with a small patch size

5.2. Fashion MNIST Dataset

While still being a monochronous dataset, the Fashion MNIST dataset contains patterns a bit more complex that what we could expect to find in the regular MNIST dataset. This allows us to offer our network a more challenging task in order to test its limits.

5.2.1. Without attention

The network succeeded well in generating Fashion MNIST images in its attention-less version. As shown below, it was even able to handle the spots between the sleeves and the trunk or between the legs with ease in the case of pants. What was less of a success are the short sleeves of t-shirts, which are poorly defined.

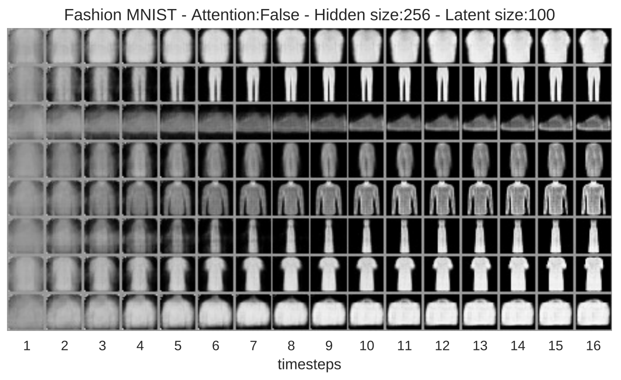
Fashion MNIST - Attention:False - Hidden size:256 - Latent size:100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
timesteps

Figure 5. Results on Fashion MNIST, without attention

5.2.2. With attention

When we put the attention mechanism on for Fashion MNIST, we didn't see many differences compared to without attention. Once again, with a large patch size (12×12 and over), it's really hard to see the attention mechanism at work, as shown below.

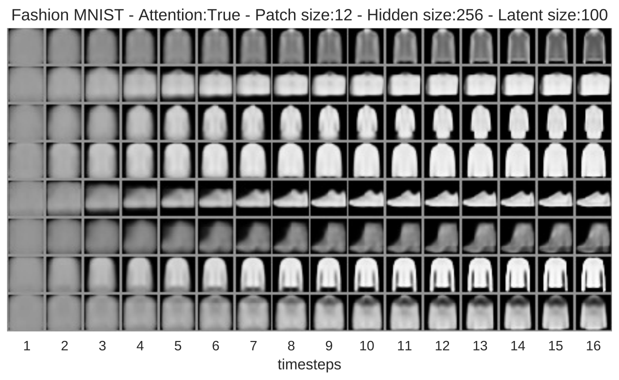
Fashion MNIST - Attention:True - Patch size:12 - Hidden size:256 - Latent size:100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
timesteps

Figure 6. Results on Fashion MNIST, with attention

When trying with a larger latent space and a much larger number of hidden features in the encoder and decoder, it helped fix the biggest issue we were having earlier, namely the above-mentioned poorly defined short sleeves of t-shirts. Moreover, we now observe a great many more shades and details inside the generated pieces of clothing that weren't there before.

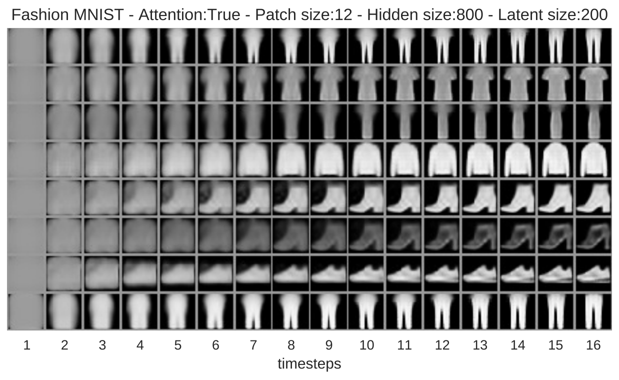
Fashion MNIST - Attention:True - Patch size:12 - Hidden size:800 - Latent size:200
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
timesteps

Figure 7. Results on Fashion MNIST, with attention and high hidden feature space for encoder and decoder

Once again, using a small patch size offered nothing but poorly defined results, as shown in figure 8.

5.3. Street View House Numbers (SVHN) Dataset

For simplicity, we used the SVHN dataset in its standard format, where images contain between one or two digits and are of size 32×32. In the original paper, they preprocessed the images so that most of them were containing between two and four digits, having a size of 54×54.

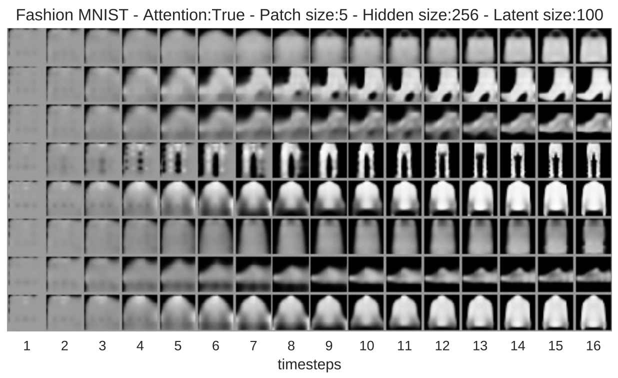

Figure 8. Results on Fashion MNIST, with attention and a small patch size

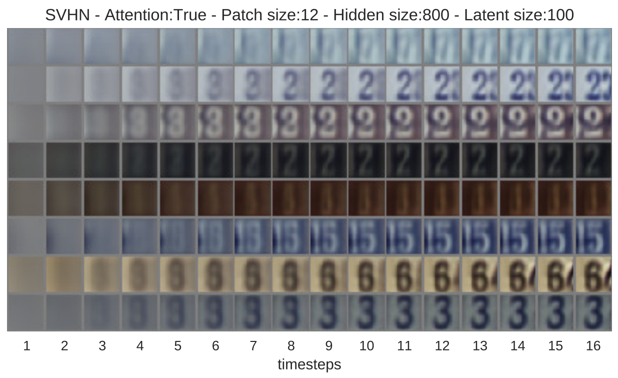

Figure 10. Results on SVHN, big hidden size

5.3.1. Without attention

The results without attention are poor on this dataset and do not look very clear. Figure 9 shows the results without attention, with 32 time steps. A lower number of time steps does not yield better results, nor a higher number.

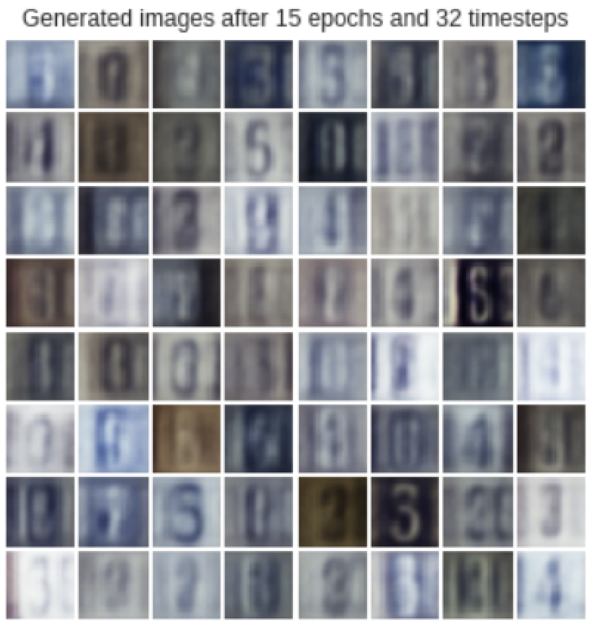

Figure 9. Results on SVHN without attention. Hidden size is 800 and Latent size is 100

5.3.2. With attention

Attention greatly helped the network in learning how to generate better quality SVHN images as shown in Figures 10, 11, 12 and 13. Networks with different hidden size, latent size and patch size where trained to understand the impact of each hyper-parameter on the quality of generation.

6. Discussion

One of the goals of the original paper was to motivate the attention mechanism. The results on MNIST shown on the paper with the attention-less mechanism are quite blurry. We found our results on MNIST to be much better than the paper's without attention. On the other hand, the attention-based results shown on the paper for SVHN are less blurry than ours. We think the reason behind this fact is that they trained their network on randomly extracted 54×54 patches from images of size 64×64 while we trained ours on a version of this dataset with images of size 32×32. Roughly speaking, the results obtained with the attention implementation are pretty good on the three datasets used, when hyperparameters were chosen appropriately. In the original paper, the results were really blurry on CIFAR10, guiding of choice to not use it in our experiments.

Another thing we noted from our experiments is that, when the patch size is greater, it is harder to observe the attention mechanism in action. This effect didn't surprise us because a greater patch size implies that the network will focus its attention on a greater part of the input image. The result is that a greater part of the generated image will be noticeably modified by the write operation. Taking this into account, we did some more experiments with a decreased patch size in order to compare the results and check if it reduced the quality of generated images. What we found is below the threshold of size 7×7 for the patches, the results became blurry. We think that decreasing the patch size too much makes it harder for the network to learn how to effectively use attention to build harmonious images. As you saw in MNIST and Fashion MNIST, decreasing the patch size made the images look somewhat inconsistent.

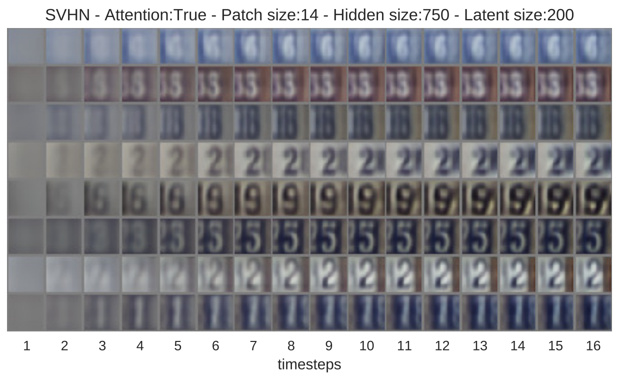
SVHN - Attention:True - Patch size:14 - Hidden size:750 - Latent size:200
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
timesteps

Figure 11. Results on SVHN, big hidden size and latent size

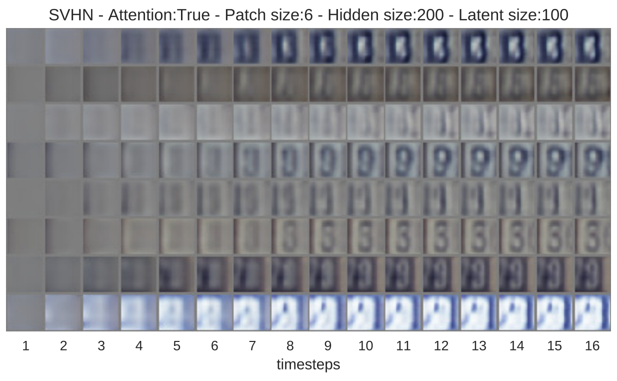
SVHN - Attention:True - Patch size:6 - Hidden size:200 - Latent size:100
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
timesteps

Figure 13. Results on SVHN, small hyperparameter values

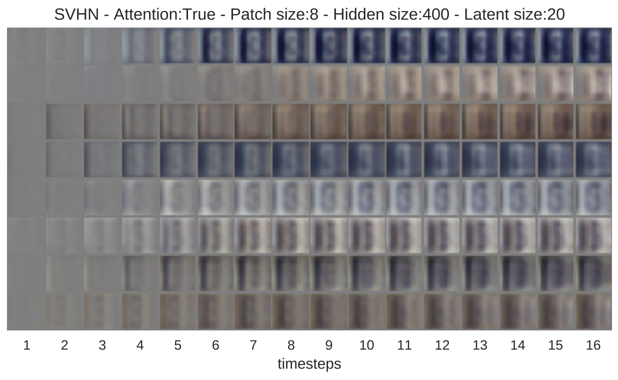
SVHN - Attention:True - Patch size:8 - Hidden size:400 - Latent size:20
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
timesteps

Figure 12. Results on SVHN, average hidden size and small latent size

The original paper proposed to use latent codes of size 100, and our experiments confirmed this choice. When we tried to make them larger, images generated by our network became less clear. It seems the problem is caused by the VAE architecture itself, that is known to use only a small subset of the dimensions of z. On the other hand, using a higher number of hidden features in the encoder and decoder dramatically increased the quality of generated images, as shown by our experiments on the SVHN dataset, but came with the trade off of making the execution longer.

Finally, on more complex images like the ones found in the SVHN dataset, the attention window tended to remain large until the end, as opposition to what was observed on MNIST. We suppose that it is because the patterns in SVHN are much more complex, yielding a higher loss. This higher loss reduces the confidence of the network producing the attention parameters, and so makes the attention window to remain large.

References

K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. DRAW: a recurrent neural network for image generation. arXiv:1502.04623, 2015.