ANN4

Irene Volpe
r0740784

Generative models

1 Restricted Boltzmann Machines

1.1 Effect of different parameters

fig 1

7, n_components: 20; learning_rate: 0.06; n_iter: 10; pseudo-likelihood = -128.93, time = 3.85s
8, n_components: 20; learning_rate: 0.06; n_iter: 20; pseudo-likelihood = -130.42, time = 3.74s
6, n_components: 20; learning_rate: 0.01; n_iter: 20; pseudo-likelihood = -137.32, time = 3.74s
5, n_components: 20; learning_rate: 0.01; n_iter: 10; pseudo-likelihood = -146.39, time = 3.75s
2, n_components: 10; learning_rate: 0.01; n_iter: 20; pseudo-likelihood = -178.26, time = 3.40s
4, n_components: 10; learning_rate: 0.06; n_iter: 20; pseudo-likelihood = -186.34, time = 3.27s
1, n_components: 10; learning_rate: 0.01; n_iter: 10; pseudo-likelihood = -186.68, time = 3.29s
3, n_components: 10; learning_rate: 0.06; n_iter: 10; pseudo-likelihood = -187.26, time = 3.26s

Fig 1: the test set
Fig 2: 8 RBMs —>
Number of components [10,20];
Learning rate [0.01,0.06]
number of iterations [10,20].
probabilities , ie the training performance, is evaluated with the pseudo-likelihood metric
(conditional probabilities can be computed without knowledge of the partition function);

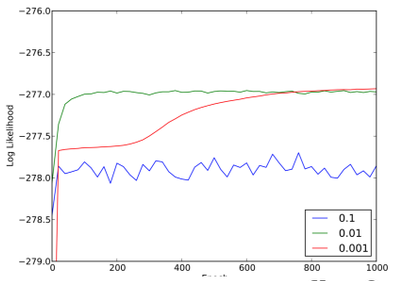


fig 3

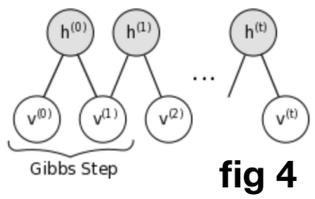


fig 4

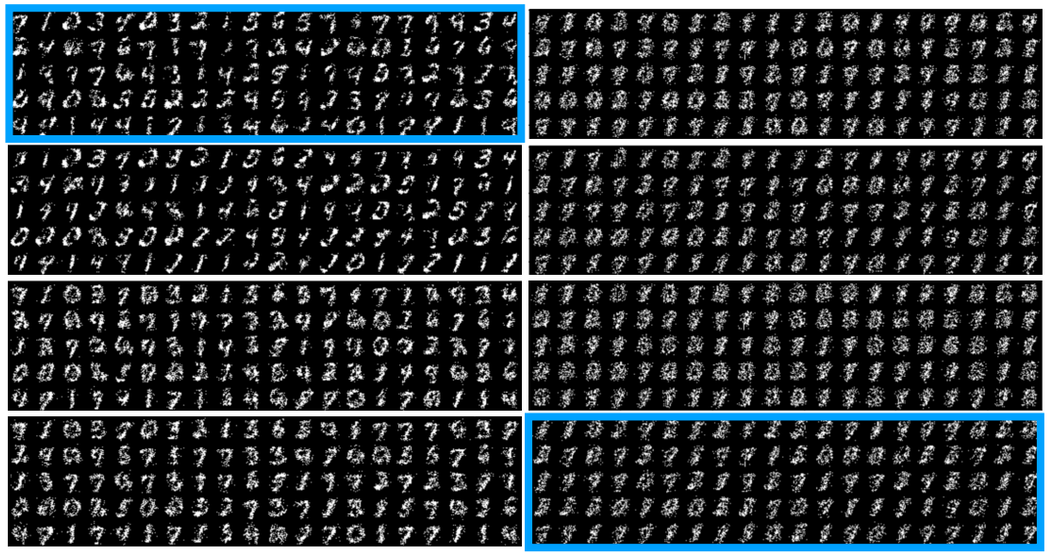


fig 2

n_components: higher values leads to more stable results .
learning_rate —-> Also trying values [0.1,0.01,0.001] I found for higher values (blue and green lines in plot 3) we achieve same results regardless the number of epochs, while smaller values (red line) lead to better results only if a sufficient number of iteration is guaranteed.
Final performance suggest one must either choose the correct intermediate learning rate value (green line) to get optimal performances at early epochs or opt for smaller learning rate values but with more epochs.
With big values (blue line) gradient easily diverges and we get numerical overflows.
n_iter —-> higher values leads to more defined results
better results —> more training time required

Rbm3: The best model in terms of performance, ie the one with fewer number of components, regardless the value assigned to the learning rate.
Rbm.7: The worst model in terms of performance, ie the one with more components and high learning rate.

1.2 Number of Gibbs sampling steps

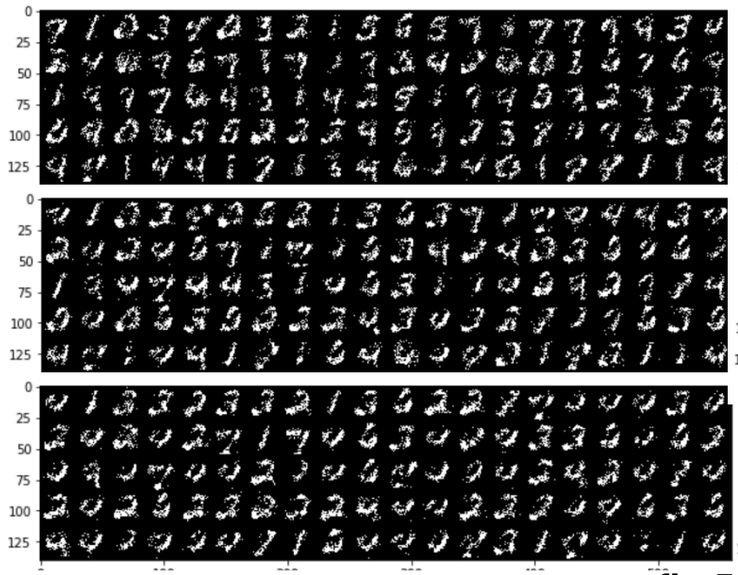


fig 5

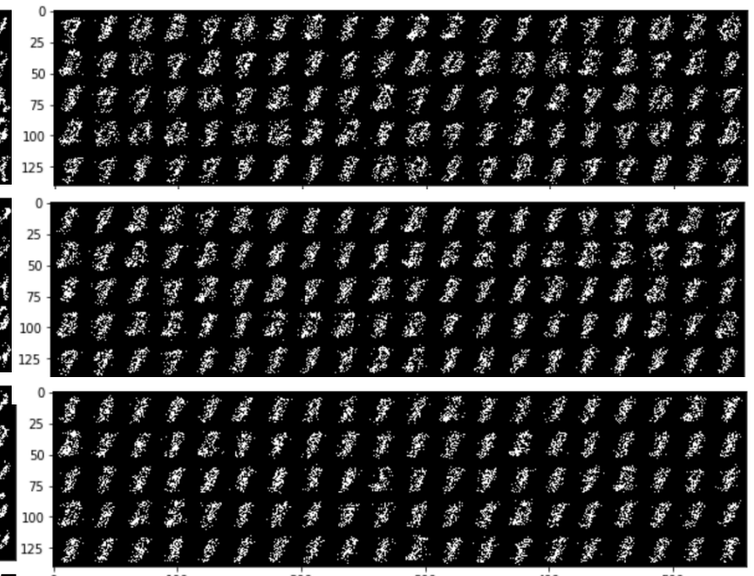


fig 6

Fig 4: Architecture of a RBM with a Gibbs step.
I tried values 10, 100, 1000 for the Gibbs steps,
using my best and worst scoring model in the training step, ie respectively: r
bm 7, n_components: 20; learning_rate: 0.06; n_iter: 10;
rbm 3, n_components: 10; learning_rate: 0.06; n_iter: 10;
Fig 5: rbm7, Gibbs 10,100,1000 starting from the top.
Fig 6: rbm3, Gibbs 10,100,1000 starting from the top.

Contrastive Divergence uses two tricks to speed up the sampling process:
1) since we eventually want p(v) ~ ptrain(v) (the true, underlying distribution of the data), we initialize the Markov chain with a training example (i.e., from a distribution that is expected to be close to p, so that the chain will be already close to having converged to its final distribution).
2) CD does not wait for the chain to converge. Samples are obtained after only k-steps of Gibbs sampling. In practice we observe that just k=1 works surprisingly well.

Both for my best and worst model, figure 4 and 5 accordingly, we notice that increasing number of steps k leads to worst results.
This could be because the initial approximated posterior summaries, ie our models rbm3 and rbm7, used to approximate posterior summaries of interest, were very inaccurate from the beginning, leading to worsen posterior summaries for bigger k values.

1.3 Reconstructing missing parts of image

1.3.1 different Gibbs step values

Learning rate didn't affect much my training,
higher number of iterations and hidden units led to better performances without overfitting.
It's better to use a lower number of gibbs steps to reconstruct the image (here no more than 100), both for the best and worst trained RBM models;
results are pretty much equal.

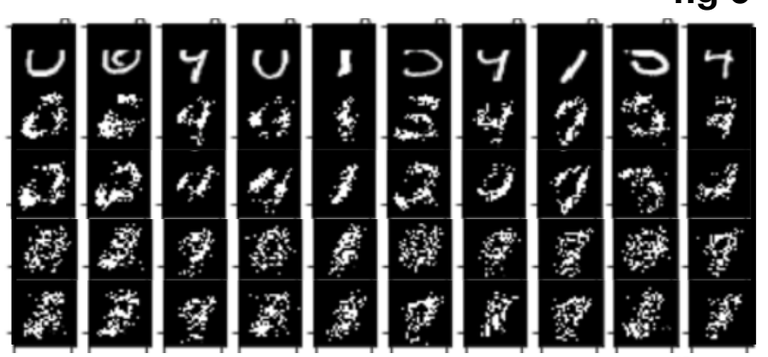


fig 7

Fig 7:
first row is initial state (10 rows of pixels removed),
second row is reconstruction from rbm7 with k = 10
third row is reconstruction from rbm7 with k = 100
forth row is reconstruction from rbm3 with k = 10
fifth row is reconstruction from rbm3 with k = 100

1.3.2 best k value fixed (10), different number of missing pixels

Rbm3 generalizes better when many test rows are missing, rbm7 overfits therefore only gets to reconstruct unseen images if the number of pixel rows removed is very little.
Using rbm3, I get to reconstruct removing 20 rows, with rbm7 only 1 becase then it starts making wrong guesses and predicts the wrong number.

Fig 8a: rbm7 reconstructs images —>
first row = 0 rows of pixels removed,
second row = 1 row of pixels removed
third row = 10 rows of pixels removed
forth row = 15 rows of pixels removed
forth row = 20 rows of pixels removed

Fig 8b: rbm3 reconstructs images —>
first row = 0 rows of pixels removed,
second row = 1 row of pixels removed
third row = 10 rows of pixels removed
forth row = 15 rows of pixels removed
forth row = 20 rows of pixels removed

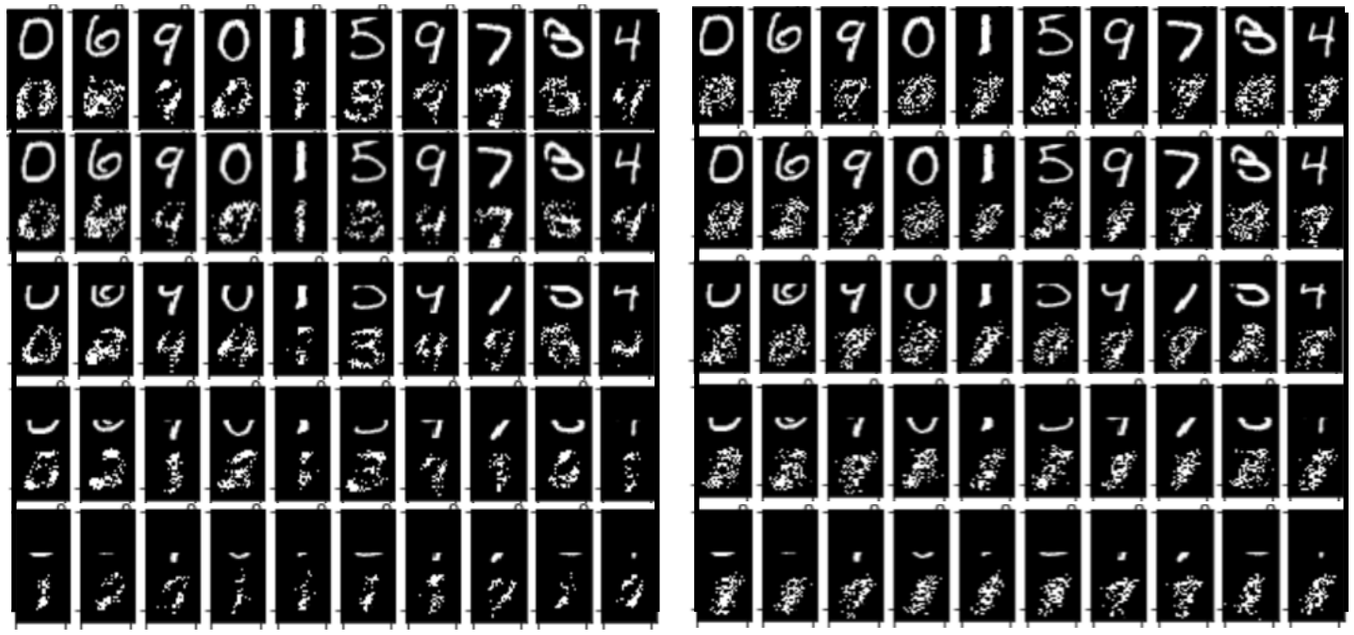


fig 8

2 Deep Boltzmann Machines

Boltzmann machine uses randomly initialized Markov chains to approximate the gradient of the likelihood function which is too slow to be practical.
DBM uses greedy layer by layer pre training to speed up learning the weights. It relies on learning stacks of Restricted Boltzmann Machine with a small modification using contrastive divergence.
The key intuition for greedy layer wise training for DBM is that we double the input for the lower-level RBM and the top level RBM.
Lower level RBM inputs are doubled to compensate for the lack of top-down input into first hidden layer.Similarly for top-level RBM, we double the hidden units to compensate for the lack of bottom-up input.
For the intermediate layers, the RBM weights are simply doubled.

Fig 9: filters of RBM
Fig 10: filters of DBM

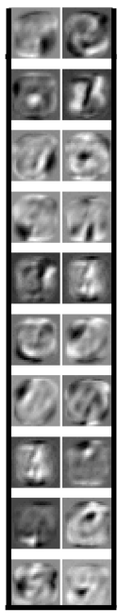


fig 9

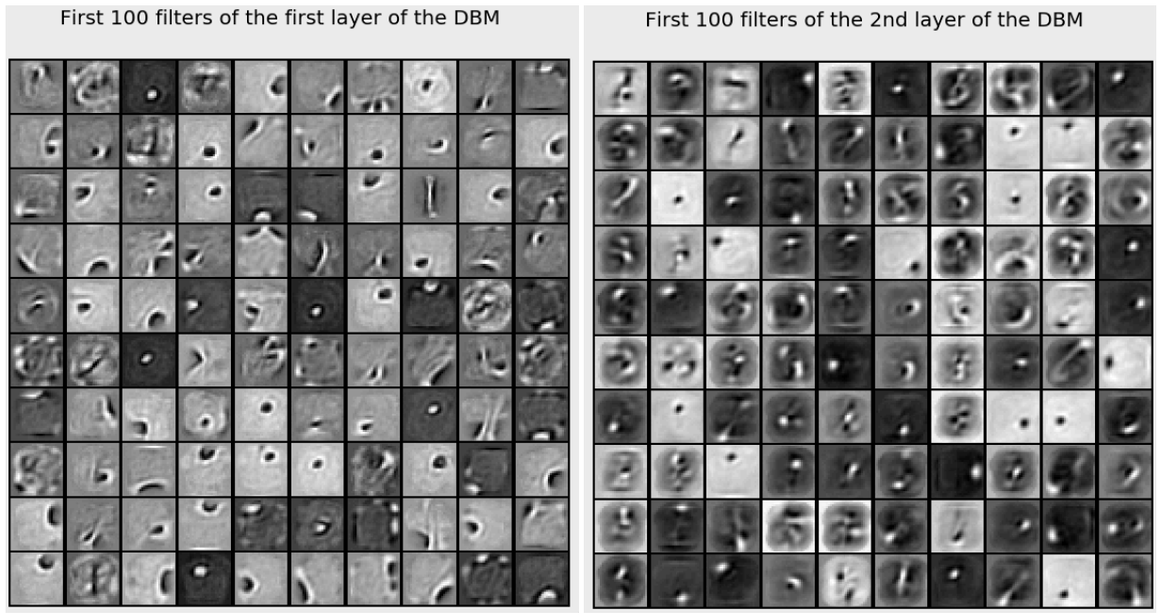


fig 10

2.1 Difference between RBM and DBM

DBM is built upon RBM to increase its representation power by increasing depth.

2.2 Difference between DBM first and second filter

filters on the second layers give more details

2.2 Difference between RBM and DBM sampled images

Deep belief network is a deep architecture built upon RBM to increase its representation power by increasing depth.
In a DBN, two adjacent layers are connected in the same way as in RBM.
The network is trained in a greedy, layer-by-layer manner, where the bottom layer is trained alone as an RBM, and then fixed to train the next layer.
After all layers are pre-trained, the resulted network contains one RBM at the top layer while the rest layers form a directed neural network.
There is no joint training for all layers in DBN.

To generate images, we need to first run Gibbs sampling in the top RBM till convergence and then propagate it down to the bottom layer.
DBN has also been extended to use convolutional RBMs for better scalability and higher quality features.

Fig 11: first plot shows images sampled by worst RBM model, second by best RBM model, third by DBM model, with fixed k value of 100. We see a mode collapse for DBM (ie multiple different digits/modes are learnt as the same digit/mode).

fig 11

3 Generative Adversarial models

3.1 monitor the loss and accuracy of the generator vs the discriminator

Fig 12: Selected class to reconstruct by the DBM

Fig 13:
first plot is the initial reconstruction step—>
batch 1 ,
D loss: 1.2105 D acc: 0.3438
G loss: 0.499 G acc: 0.7344;
second plot is an intermediate reconstruction step —>
batch 10000 ,
D loss: 0.6916 D acc: 0.4688
G loss: 0.7905 G acc: 0.2344;
third plot is the final reconstruction step —>
batch 20000 ,
D loss: 0.7291 D acc: 0.4219
G loss: 0.7381 G acc: 0.4062;

Fig 14:
plot1: D's Accuracy is higher and more stable than G's.
plot2: D's Loss is lower and more stable than G's.
plot3: D's Loss is higher and more stable than D's Accuracy.
plot4: G's Loss is higher and more stable than G's Accuracy.

3.2 comment on the stability of the training

GAN is based on the zero-sum non-cooperative game (aka minimax game) : if one wins the other loses.
Your opponent wants to maximize its actions and your actions are to minimize them.
GAN model converges when the discriminator and the generator reach a Nash equilibrium (optimal point for the minimax equation).

Cost functions may not converge using gradient descent in a minimax game, because minimax game is a non-convex game: your opponent always countermeasures your actions which make the models harder to converge. Yet in Fig 14 a and b we see a convergence at epoch 6 (black circle).

Mode/generator collapse is one of the hardest problems to solve in GAN. A complete collapse is not common but a partial collapse happens often.
Fig 15: images 2-6,4-7 and 5-8 look similar and the mode starts collapsing.
Fig 14b: two partial collapses (arrows).

fig 12

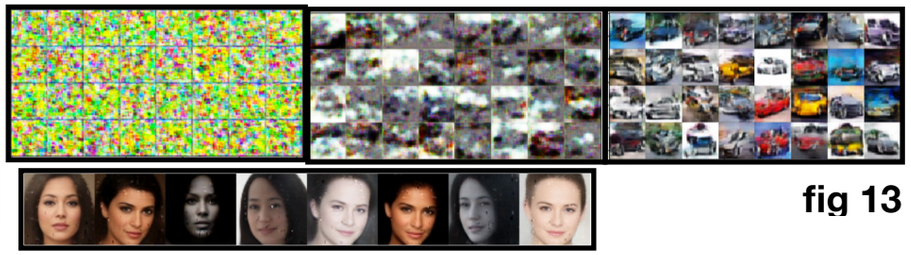


fig 13

fig 15

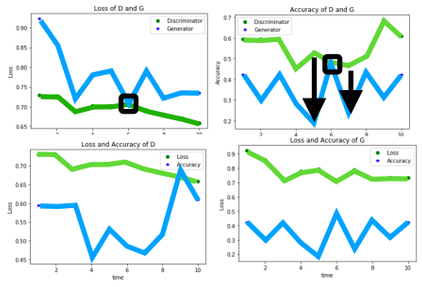


fig 14

fig 16 a,b,c,d

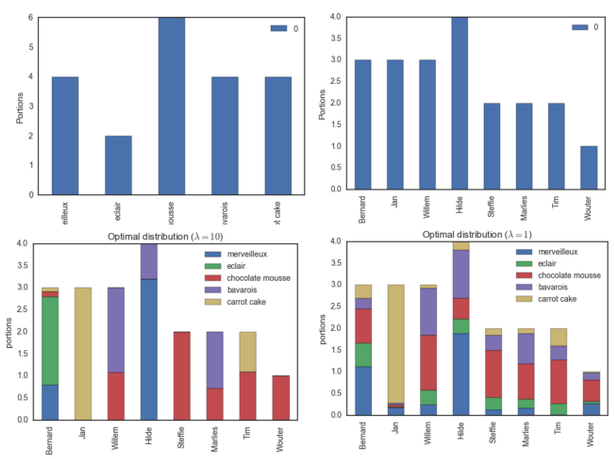
4 Optimal Transport

4.1 transfer colors between images

In OT the goal is to transform one probability distribution into another with a minimal cost. In Fig 16 Two OT algorithms are used: EMD (16c) and Sinkhorn (16d); we represent an image as a distribution of pixels in a 3D color space. we can use OT and a simple multivariate regression method to map one color scheme into another. A wider range of green shades from image a is mapped to fewer shades of yellow into image b (underlined by the two rectangles), since the latter was posterized before the upload. EMD (unregularized) and Sinkhornt (regularized) methods don't differ significantly.

Fig 17 a: big λ —> everybody only has desserts they like;

Fig 17b: low λ —>encourage a more homogeneous distribution, though some people will have to try some desserts which are not their favorites;



The steps involved includes:
1) link the instances of the train and test sets using optimal transport,
2) learn the mapping from train to test/apply a transformation to the training set to minimize the Wasserstein or Sinkhorn distance.
3) Build a classifier on the training data transformed to match the testing distribution.

The Earth Mover's Distance (EMD) evaluates dissimilarity between two multi-dimensional distributions in some feature space where a distance measure between single features called the ground distance is given. The EMD ``lifts'' this distance from individual features to full distributions. In case of Sinkhornt, Given: M, r, c and λ , initialize: $P\lambda=e^{-\lambda M}$ and repeat until convergence: scale the rows such that the row sums match r, scale the columns such that the column sums match c. Increasing the entropy parameter in sinkhornt means making the distribution more homogeneous.
The Wasserstein distance (or the EM distance) is the cost of the cheapest transport plan. Wasserstein Distance is a measure of the distance between two probability distributions. It is also called Earth Mover's distance.
Even when two distributions are located in lower dimensional manifolds without overlaps, Wasserstein distance can still provide a meaningful and smooth representation of the distance in-between.

4.2 compare GAN and WGAN

Compared to the original GAN algorithm, the WGAN undertakes the following changes:

- After every gradient update on the critic function, clamp the weights to a small fixed range, [−c,c][−c,c].
- Use a new loss function derived from the Wasserstein distance, no logarithm anymore. The "discriminator" model does not play as a direct critic but a helper for estimating the Wasserstein metric between real and generated data distribution.

Real-life data like MNIST are multimodal; there are 10 modes from digit '0' to digit '9'. We see in Figure 17, shows training results of GAN and WGAN over different number of iterations. that standard GAN classifies all digits as '0's' thus, suffering from modal collapse i.e. he generator collapses which produces limited varieties of samples.

Figure 17. shows that WGAN generates more samples and with optimal architectural choices, they can be more stable than GAN. This is because it allows to train critic to optimality which then generates loss to the 'generator' that can then be trained as a neural network. The better the critic is trained, the higher the quality of gradients used to train the generator are. Thus, we then don't have to balance the generator-discriminator capacity.

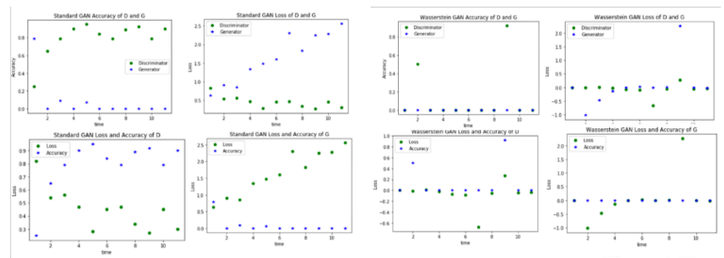


fig 17

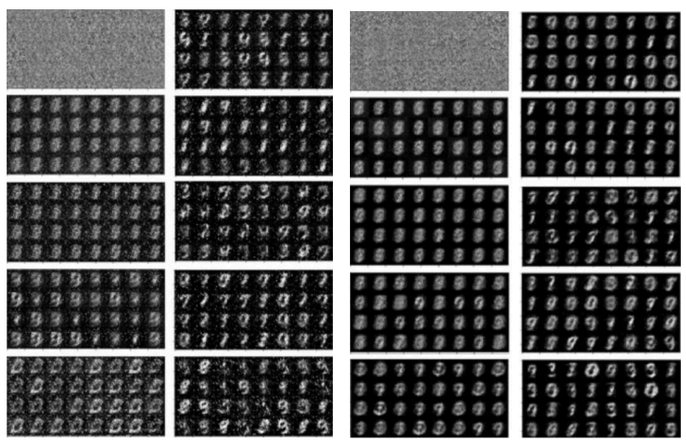


fig 18