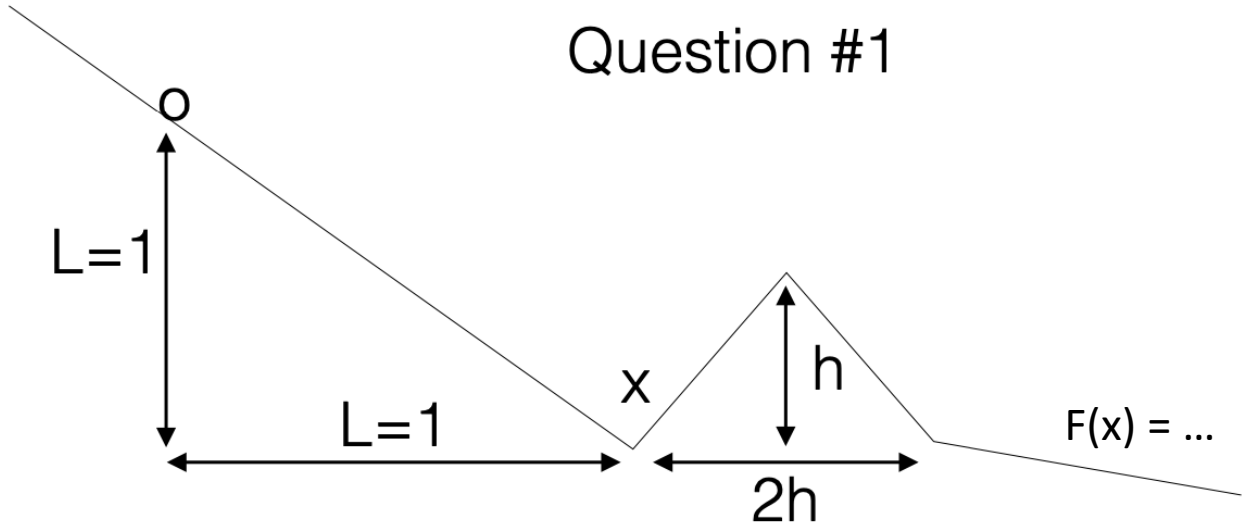
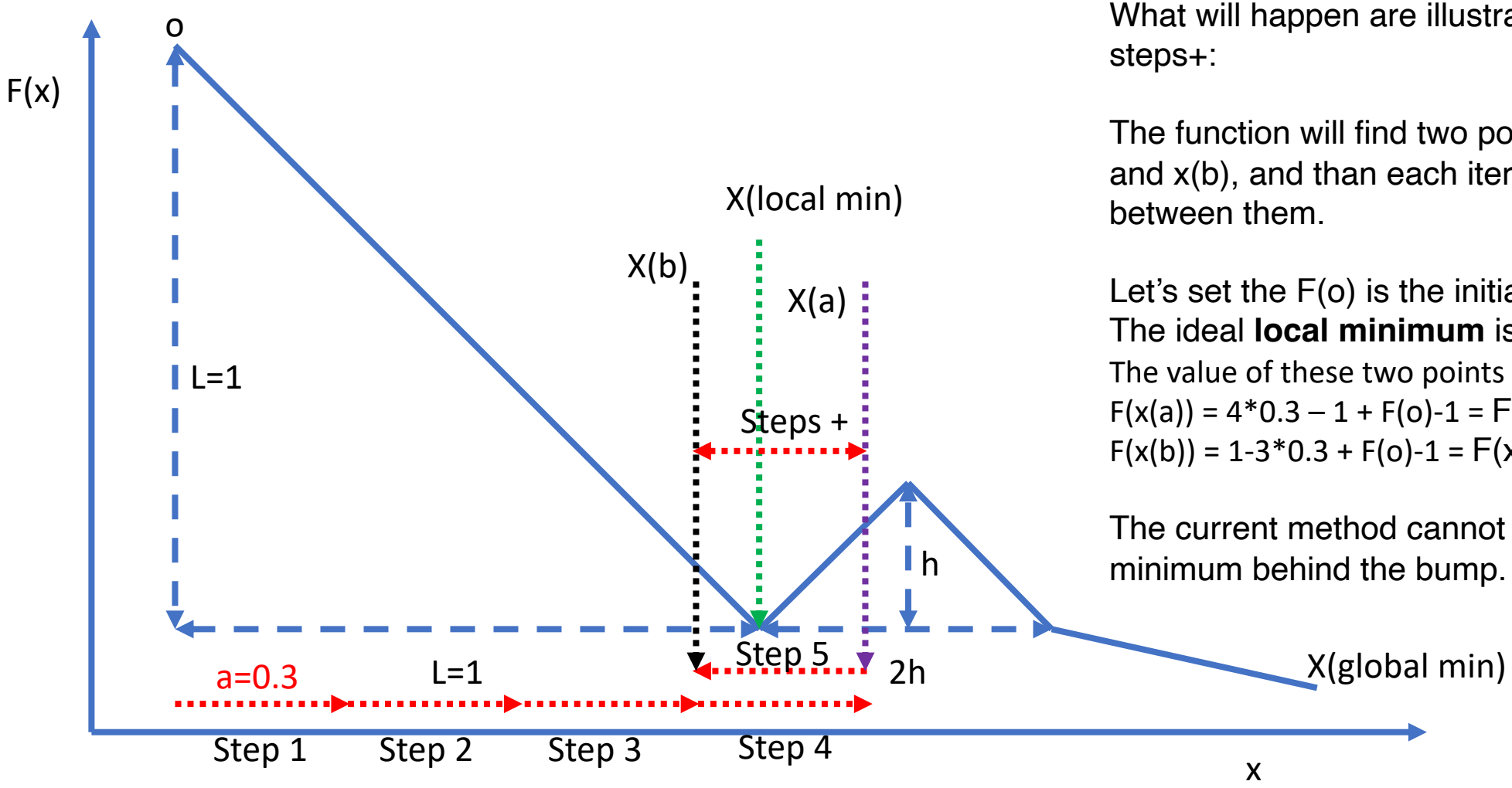
Question #1



The diagram above shows a plot of a 1D function and gradient descend is applied to minimise the function at the point ‘o’. there is a bump a distance L away with bump dimensions given as h x 2h. Let L = 1, a = 0.3 and h > a where a is the learning rate

What will happen if you apply standard gradient descend?

First, we need to understand the question: the give 1D function is F(x),
and gradient descend is used to find a x(min) that obtains minimum F(x(min)).
In the optimization, the gradient of F(x) for x is always horizontal (in this specific case, it's 1 or -1),
each time x will be change by a: x1 -> x0+a or x1 -> x0-a



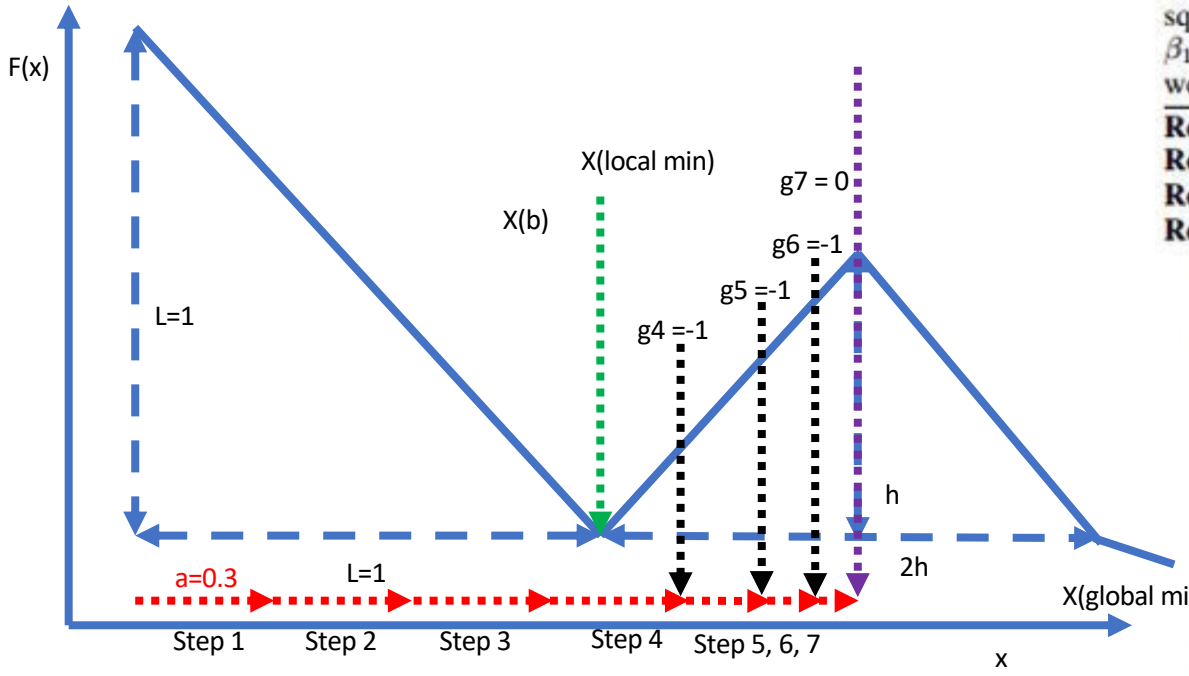
What will happen are illustrated in steps 1 to 5 and steps+:

The function will find two possible minimum x: x(a) and x(b), and than each iteration will be switch between them.

Let's set the F(o) is the initial value,
The ideal **local minimum** is F(x(local min)) = F(o)-1
The value of these two points will be:
F(x(a)) = 4*0.3 – 1 + F(o)-1 = F(x(local min)) + 0.2
F(x(b)) = 1-3*0.3 + F(o)-1 = F(x(local min)) + 0.1

The current method cannot find the global minimum behind the bump.

if you apply adam optimisation with parameters given in the next slide, what is the max height 'h' of the bump in which the adam optimiser will escape the local min at 'x'? use \epsilon = 0 instead of \epsilon = 1e-8 in your calculations.



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

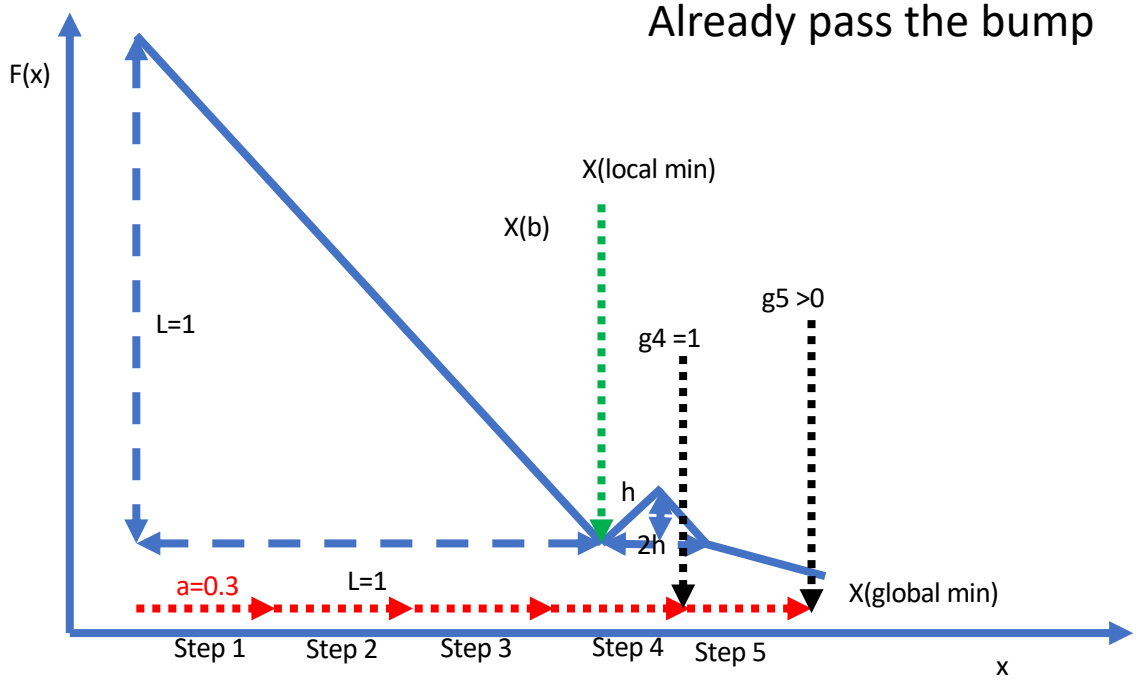
Beta1=0.9,beta2=0.999,alpha=a=0.3

From step 1-4, t in (1,2,…,4):

Step t:
t=t
g(X(t-1))=+1
M(t) = (1-beta1^t)
V(t)=(1-beta2^t)
M^=M(t)/(1-beta1^t) = 1
V^=V(t)/(1-beta2^t) = 1
=>
X(t)=t* alpha

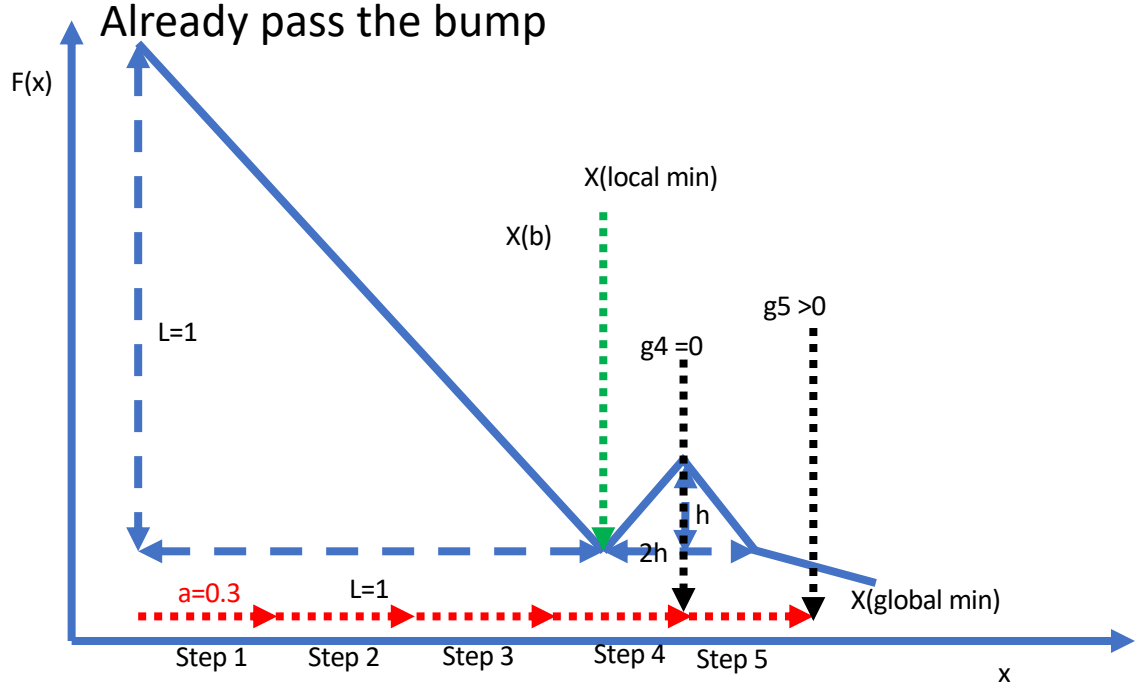
Step 5:
t=5
If step 5 pass the bump:
h< 5*alpha-1
g(X(t-1))=+1

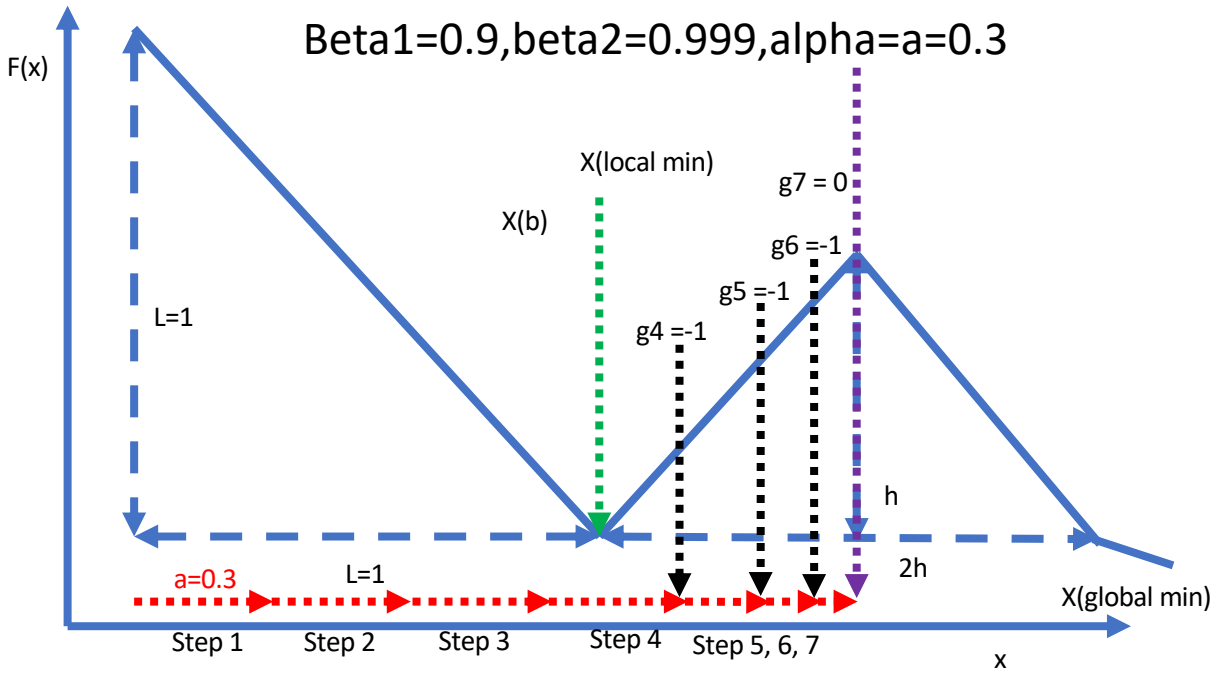
M(t) = (1-beta1^t)
V(t)=(1-beta2^t)
M^=M(t)/(1-beta1^t) = 1
V^=V(t)/(1-beta2^t) = 1
=>
X(5)=5* alpha
Already pass the bump



Step 5:
t=5
If step 5 at the bump top:
g(X(t-1))=0

M(t) = beta1 * (1-beta1^(t-1)) = beta1-beta1^t
V(t)=beta2 *(1-beta2^(t-1)) = beta2-beta2^t
M^=M(t)/(1-beta1^t) = … = 0.7558…
V^=V(t)/(1-beta2^t) = … =0.7996…
=>
X(5)=…=1.453…
1.453…>1.2
Already pass the bump





Step 5: t=5
If step 5 haven't reach at bump top:
g(X(t-1))=-1

M(t) = beta1 * (1-beta1^(t-1))-(1-beta1) = 2*beta1-beta1^t-1

V(t)=beta2 *(1-beta2^(t-1))+(1-beta2) = (1-beta2^t)

M^=M(t)/(1-beta1^t) = (2*beta1-beta1^t-1)/(1-beta1^t) = 0.511614…
V^=V(t)/(1-beta2^t) = 1
=>
X(5)= 4* alpha+ alpha *M^=1.35348…
1.35348 …>1.2, still moving forward
Go to the next step

Step 6: t=6
If step 6 haven't reach at bump top:
g(X(t-1))=-1

M(t) = beta1 * (2*beta1-beta1^(t-2))-(1-beta1) =0.088559
V(t)= (1-beta2^t)

M^=M(t)/(1-beta1^t) = …= 0.189002879…
V^=V(t)/(1-beta2^t) = 1
=>
X(6)= …=1.4101851777…
1.4101851777…>1.35348 …, still moving forward
Go to the next step

Step 7: t=7
If step 7 haven't reach at bump top:
g(X(t-1))=-1

M(t) = …=-0.0202969
V(t)= (1-beta2^t)

M^=M(t)/(1-beta1^t) = …= -0.038905…
V^=V(t)/(1-beta2^t) = 1
=>
X(7)= …=1.395813…
1.395813… <1.4101851777…not moving forward
step 7 already reach or passed the top

Step 7: t=7
If step 7 is at bump top (**decision boundary**):
g(X(t-1))=0

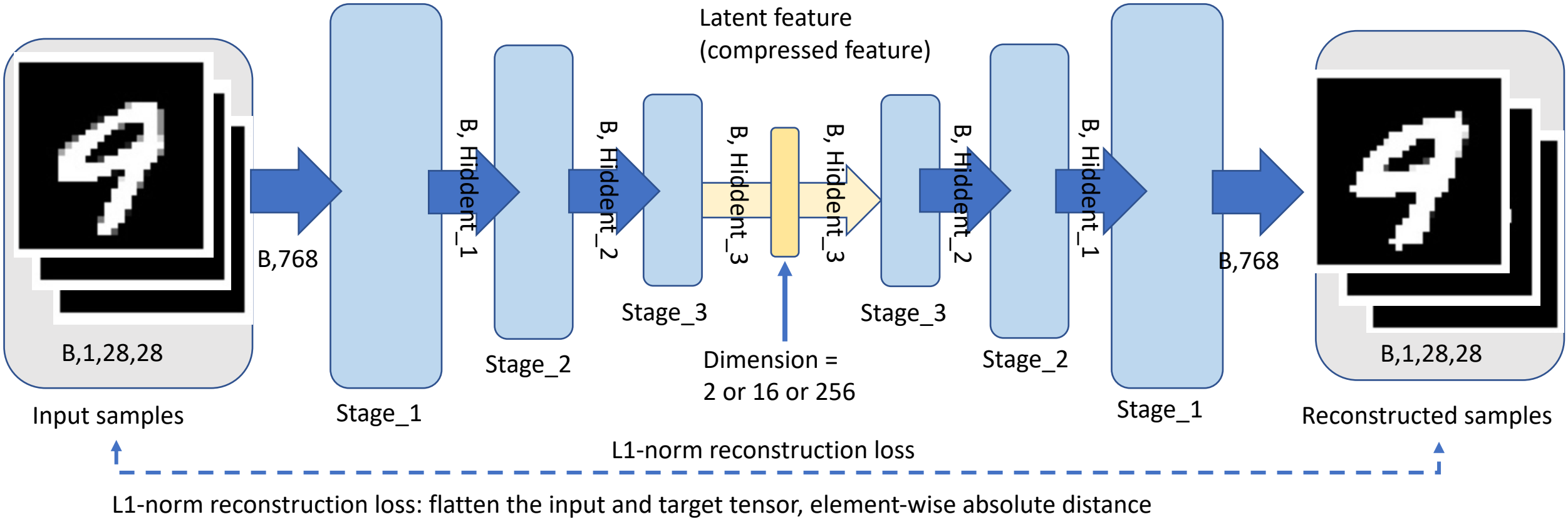
M(t) = bets2*0.0202969=0.079703
V(t)= beta2 *(1-beta2^(t-1)) = beta2-beta2^t
=0.005979…

M^=M(t)/(1-beta1^t) = …= 0.15277483…>0
V^=V(t)/(1-beta2^t)= 0.856713714…>0
=>
X(7)= …=1.459702277…
1.45970… >1.41018517…can moving forward

H= X(7)-1=0.459702277

Question #2

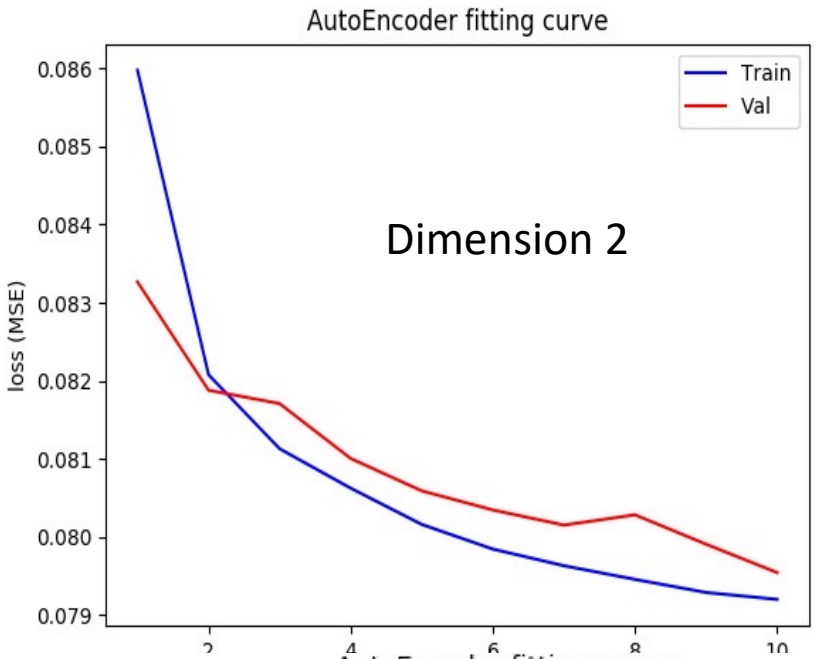
(a) Design an auto encoder to take in MNIST images with latent space dimension of 2,16,256. Train auto encoder with L1-norm reconstruction loss

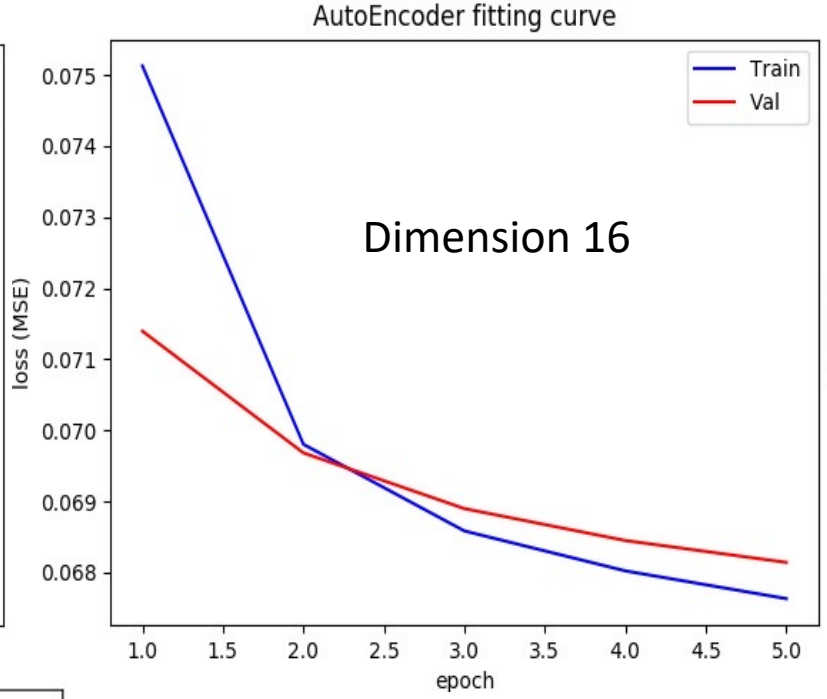


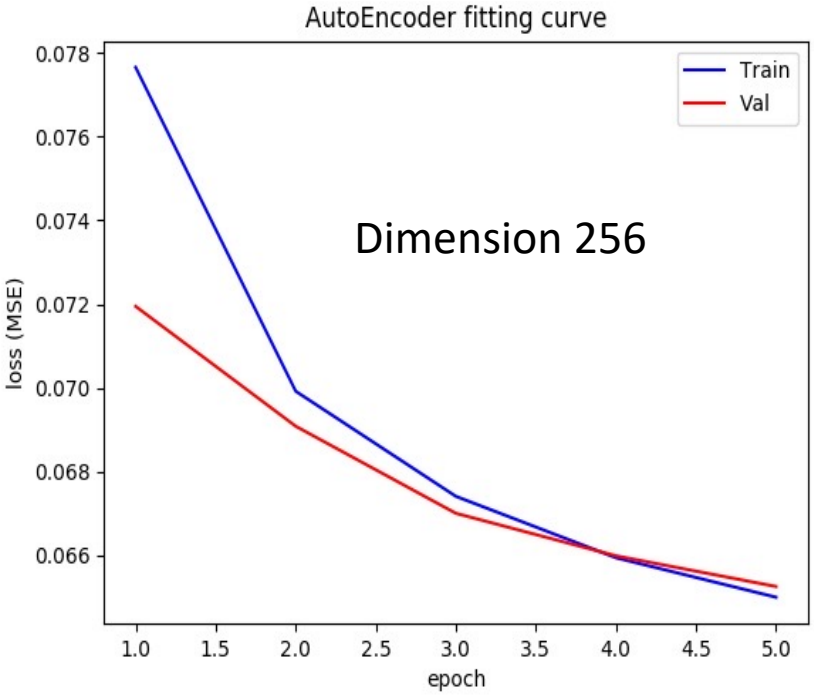
L1-norm reconstruction loss: flatten the input and target tensor, element-wise absolute distance

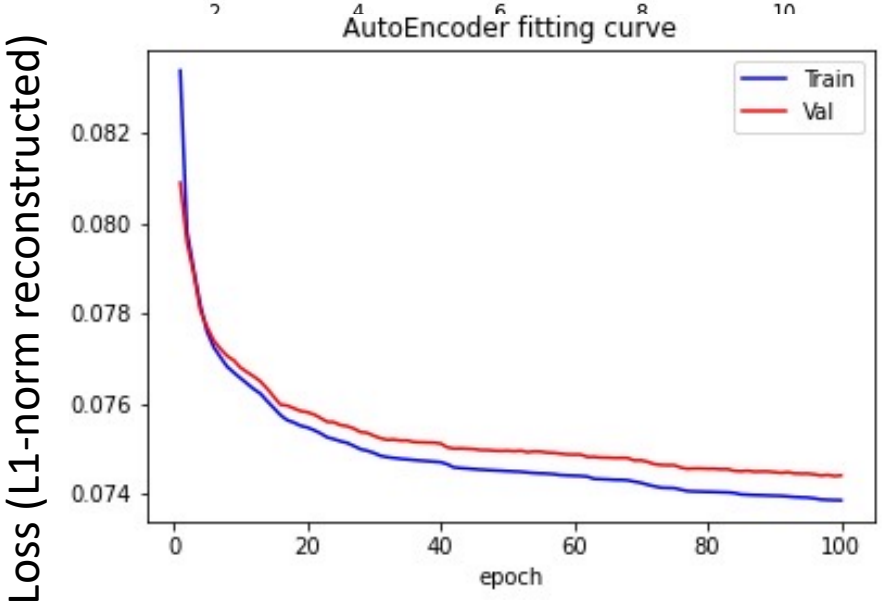
Question #2

(a) Design an auto encoder to take in MNIST images with latent space dimension of 2,16,256. Train auto encoder with L1-norm reconstruction loss



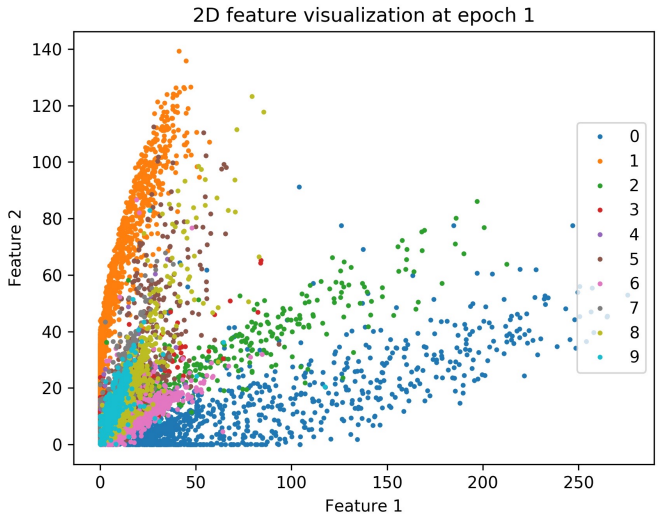


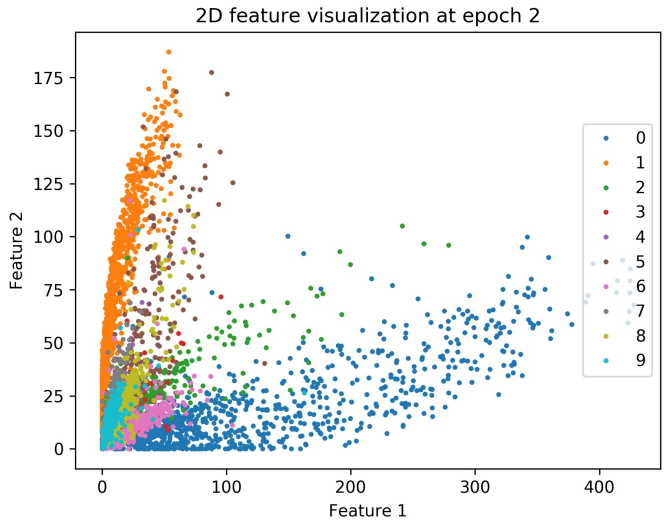


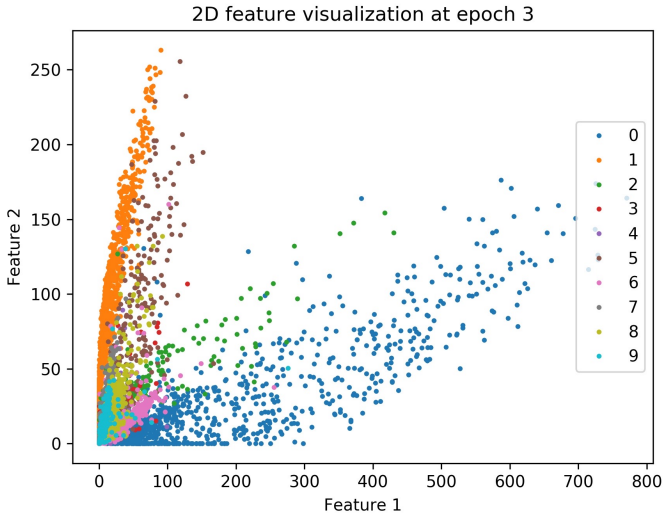


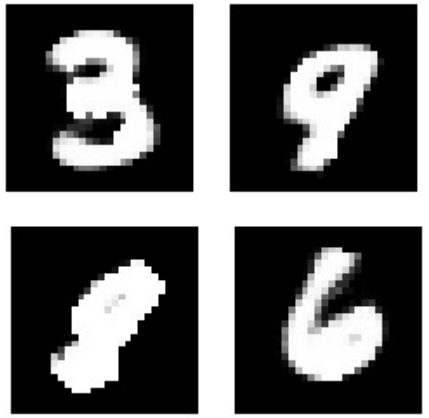
I find using MSE loss can use obvious less epochs, which will be used as the result. L1 version are also illustrated. The training epochs are obviously more.

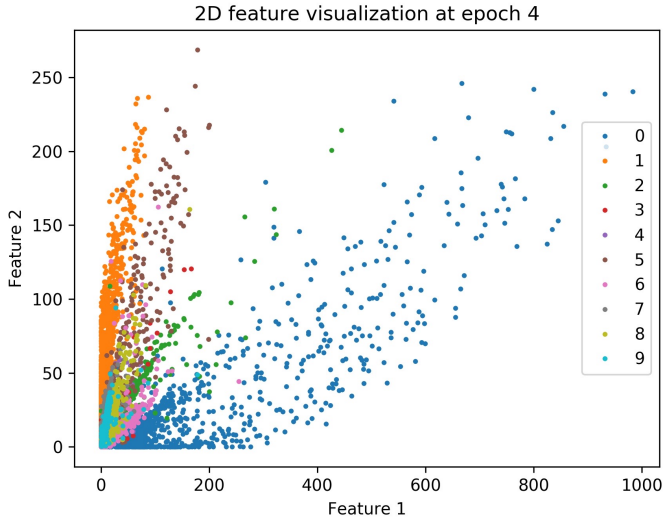
Do a 2D plot of the latent space for different digits for latent space of 2D. Use one color for each digit.

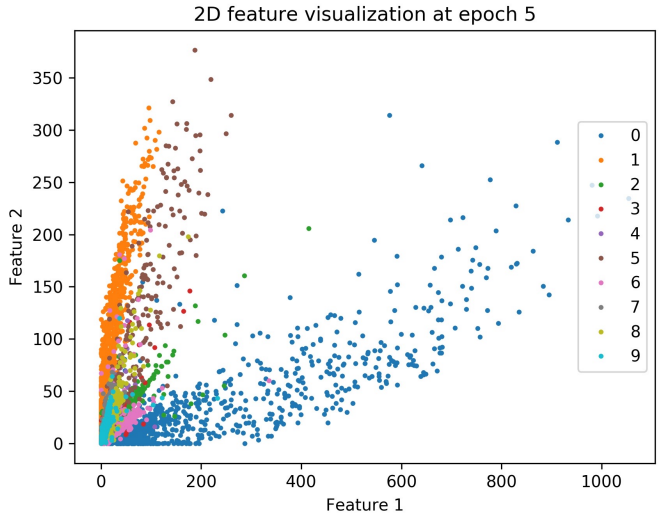


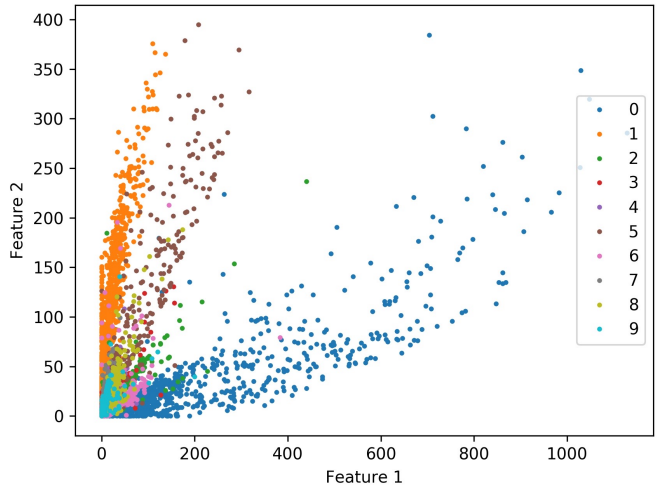


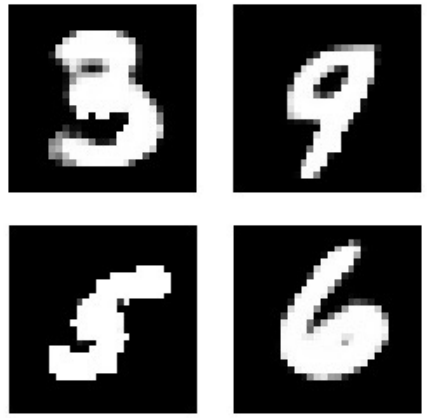






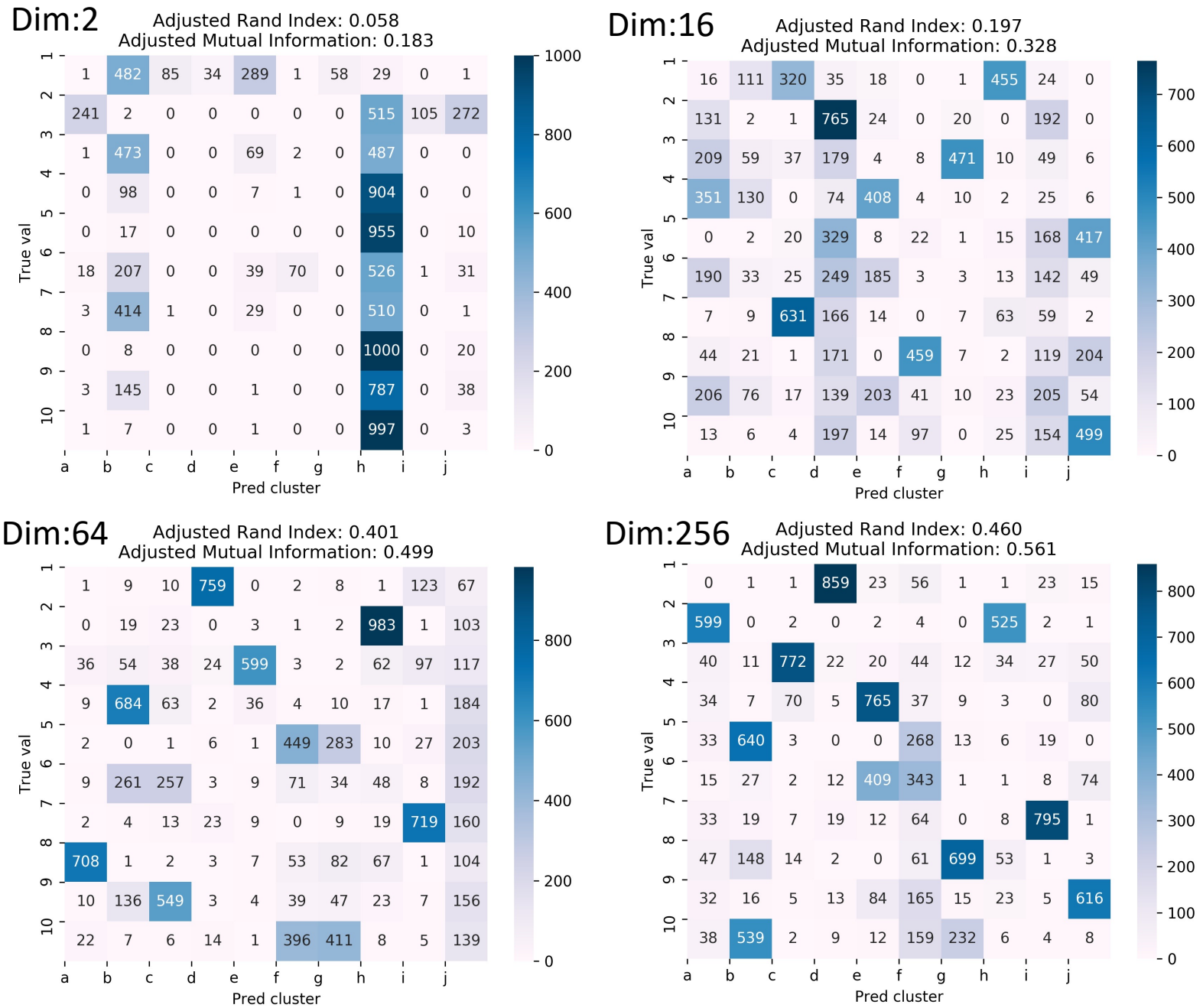






The visualization can roughly indicate the categories distributions for the autoencoder with the latent dimension of 2. Through the epochs, It can be noticed that the distances of the categories are further (the scale also changed), while the reconstructed figures are becoming more apparent.

K-means clustering for latent space of dimensions 16,256.



It can be noticed that the clusters match separate categories better (each cluster the colored blocked corresponds to a different digit) when the latent dimension increase.
The results indicate that more information is recorded, and the clusters are less alike.

What do you notice about the reconstructed images?

Similarity:
All methods (different latent dimensions of 2,16,256) present acceptable reconstraction to the original image.
All methods loss some details toward original image.

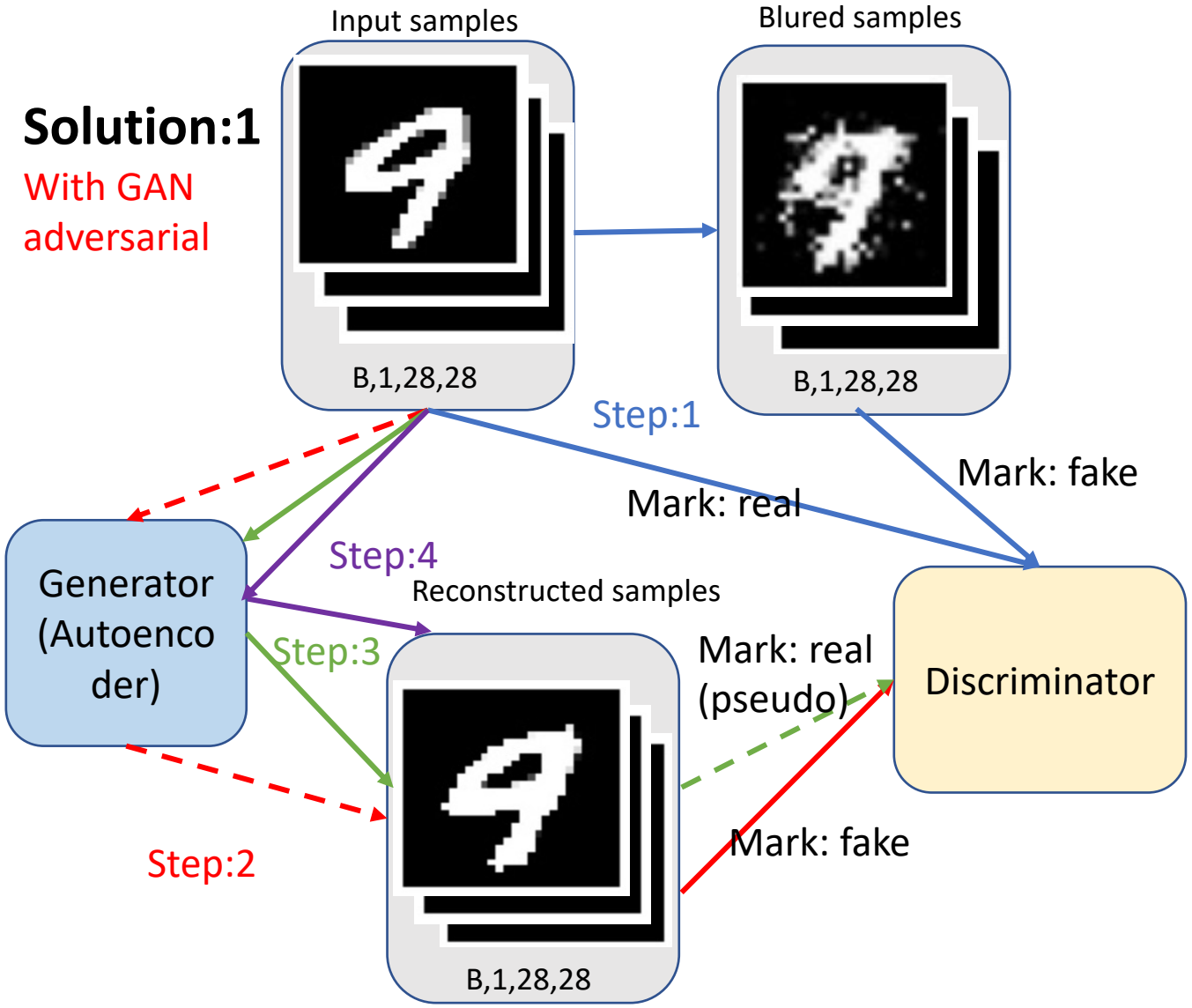
Difference:
The method of lower latent dimension may present worse results, which sometimes may be wrong number. (as illustrated in red boxes) this maybe because the latent dimsension is not enough to correctly compose enough details.
The method of higher latent dimension present better detail. (as illustrated in blue boxes)
From a view of loss, higher latent dimension design have lower loss.

(b) Design another neural network "dis_net" to discriminate between blur images and clear images.

Train auto encoder with L1-norm reconstruction loss + discriminator loss. Make reconstructed images as clear as possible, that is, the auto encoder will need to be trained so that "dis_net" score it as a clear image.

Solution:1

With GAN adversarial



Dash lines means no gradient is recorded.

Following the key concept of DCGAN, a training strategy with 4 steps are designed.

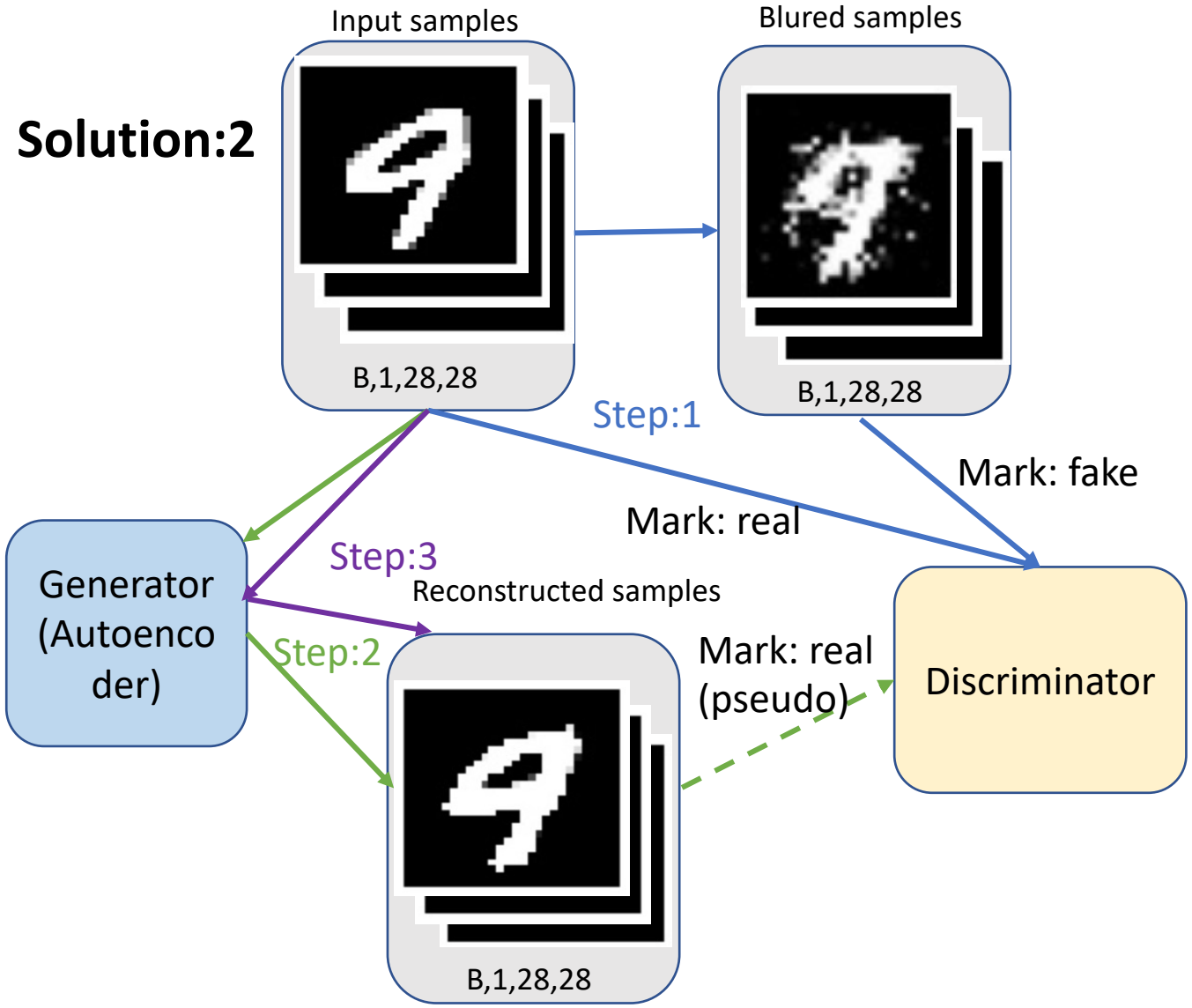
Step. 1: use input images and corresponding blured images to train D with actual real/fake label. The D learns to distinguish blured samples. Update D weight.

Step 2: use G to generate reconstructed samples (G without gradient) and use the samples to train D with actual fake label (because they are generated). Update D only.

Step 3: use G to generate reconstructed samples, and use D to identify them, with pseudo [Real] labels. Update G only, clear the gradient for D. In this case the G's gradient is award by the pseudo loss (G's correctness) from D, while D remain not updated.

Step 4: use G to generate and use reconstraction loss to update G. Notice: the step 4 can be aggreated in step 3.

Solution:2



Dash lines means no gradient is recorded.

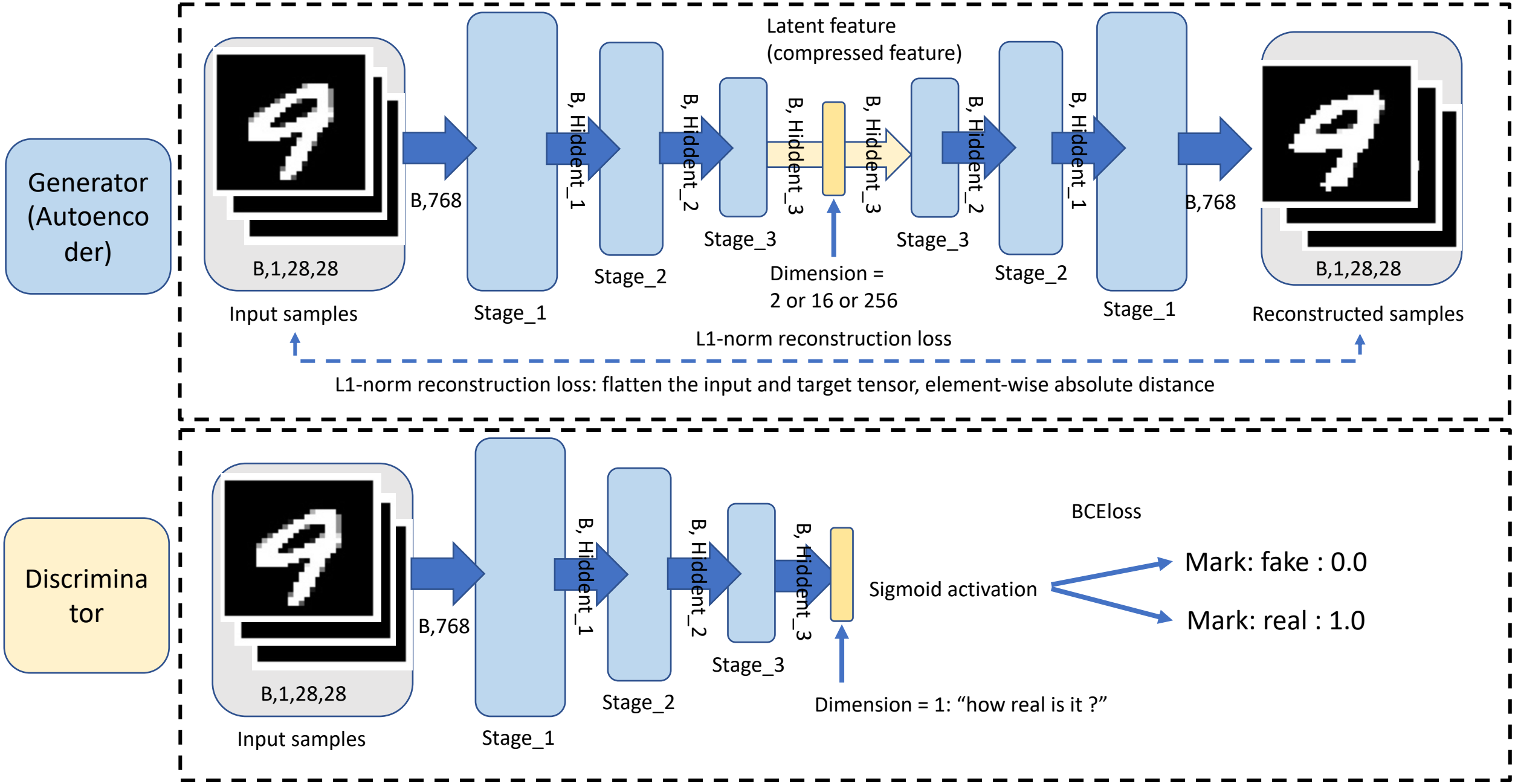
By simplifying the GAN adversarial process, a training strategy with 3 steps (corresponding to 1,3,4 in solution 1) are designed.

Step. 1: use input images and corresponding blured images to train D with actual real/fake label. The D learns to distinguish blured samples. Update D weight.

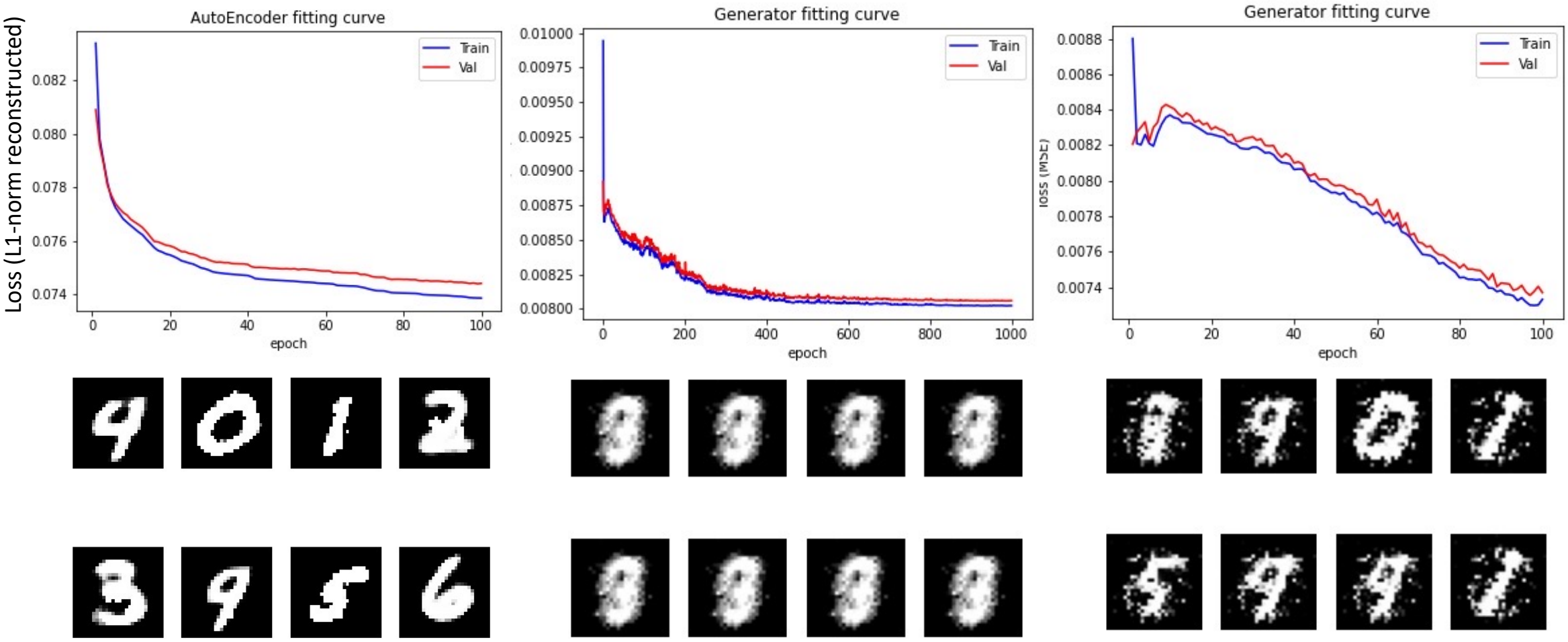
Step 2: use G to generate reconstructed samples, and use D to identify them, with pseudo [Real] labels. Update G only, clear the gradient for D. In this case the G's gradient is award by the pseudo loss (G's correctness) from D, while D remain not updated.

Step 3: use G to generate and use reconstraction loss to update G. Notice: the step 3 can be aggreated in step 2.

Network structure:



Compare results between (a) and (b)



From the view of loss, the results are similar. However, the visualization shows another story. Due to the limitation of GAN (training difficulty), the results of solution 1 and 2 with D are both worse than Q2_a, without the adversarial design, solution 2 is better.