**Assignment 2**

# Question 1

Diagram

Description automatically generated

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×*2 *h*. Let *L*=1, *a*=0.3 and *h*> *a*, where *a* is the learning rate.

1. what will happen if you apply standard gradient descend?

After a few steps, the cost function will converge to the local minimum x, instead of the global minimum, because the learning rate a is lower than h, which means each step is smaller than what is enough to detect another local minimum. Therefore, the function is trapped there.

1. 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 *Ɛ*=0 in instead of *Ɛ*=1e-8 in your calculations.

The maximum height is 0.41. Please refer to the code below:

import numpy as np

def de\_function(x, h):

if x<1:

return -1

elif 1<x< (1+h):

return 1

elif (1+h)< x<(1+2\*h):

return -1

else:

return -0.3

def find\_h(derivative, n\_iter, alpha, beta1, beta2, eps=0):

for h in np.arange(0.3,1,0.0001):

x = 0

m = 0.0

v = 0.0

for t in range(1,n\_iter):

g = de\_function(x,h)

m = beta1 \* m + (1 - beta1) \* g

v = beta2 \* v + (1 - beta2) \* g\*\*2

mhat = m / (1.0 - beta1\*\*t)

vhat = v / (1.0 - beta2\*\*t)

x = x - alpha \* mhat / (vhat\*\*0.5 + eps)

if x>(1+h):

break

if x<(1+h):

print(h)

break

n\_iter = 10000

alpha = 0.3

beta1 = 0.9

beta2 = 0.999

eps = 0

find\_h(de\_function, n\_iter, alpha, beta1, beta2)

**Question 2**

* 1. 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.

Please see the attached notebook. We take part of the code from <https://avandekleut.github.io/vae/> with modification for our purposes of this implementation. We set epoch number to be 20 unless otherwise stated.

Do a 2D plot of the latent space for different digits for latent space of 2. K-means clustering for latent space of dimensions 16 , 256. Use one color for each digit.

**Dim =2:** the result shows no clear clustering, suggesting little information learnt.

**Dim = 16:** the result shows clear clustering. k-mean clustering results are unstable due to random centroids, but I managed to choose one that is closest to the actual results, which is not easy to get unless trying many times. This dimension number shows the best clustering result by k-means.

**Dim = 256:** it seems that little information is learnt and correctly summarized in the latent space, and the k-means results are poor, although it shows some kind of shapes we expect as a result of auto-encoder learning.

**Latent layer size = 2**

Scatter chart

Description automatically generatedScatter chart

Description automatically generated A screenshot of a computer

Description automatically generated with medium confidence

**Latent layer size = 16**

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated A picture containing text, monitor, screen, display

Description automatically generated

**Latent layer size = 256**

Chart, scatter chart

Description automatically generatedChart, scatter chart

Description automatically generated A picture containing text, monitor, display

Description automatically generated

Report all results. What do you notice about the reconstructed images?

**No dis\_net (latent layer size = 2, 16, 256)**

A screenshot of a computer

Description automatically generated with medium confidenceA picture containing text, monitor, screen, display

Description automatically generatedA picture containing text, monitor, display

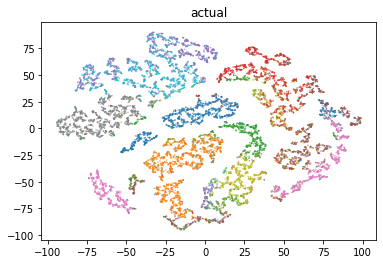
Description automatically generated

The images reconstructed in the latent layer seem to be a blurred, distorted version of the original images, which may indicate certain information loss. As shown below, it is getting more blurred, unrecognizable with the increase of latent layer size from 2 (left) to 16 (middle) to 256 (right).

* 1. Design another neural network “dis\_net” to discriminate between blur images and clear images. Blur images can be generated by taking the original MNIST data and do some gaussian blur. Train autoencoder 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. Compare results between (a) and (b)

Please the attached Notebook. Compared to (a), there are more features captured by the net, which makes the image a bit clearer, but still unrecognizable (see next page) given the same epoch number and latent layer size. Also, as visualised in the latent space below, there are clearer clustering in the latent space for different digits, thus improving prediction accuracy, especially when the latent layer is 16. However, in our model, the discriminator loss is so small when compared to autoencoder loss that it may causes only limited impact. Thus, the autoencoder structure should be optimised for smaller loss.

**With dis\_net (latent layer size = 2; 100 epochs)**

 A picture containing text, vector graphics

Description automatically generatedText

Description automatically generated

**With dis\_net (latent layer size = 16; 100 epochs)**

Chart, scatter chart

Description automatically generated Scatter chart

Description automatically generatedA picture containing graphical user interface

Description automatically generated

**With dis\_net (latent layer size = 256; 100 epochs)**

Scatter chart

Description automatically generated Scatter chart

Description automatically generated A picture containing graphical user interface

Description automatically generated

**No dis\_net (latent layer size = 2, 16, 256; 20 epochs)**

A screenshot of a computer

Description automatically generated with medium confidenceA picture containing text, monitor, screen, display

Description automatically generatedA picture containing text, monitor, display

Description automatically generated

**With dis\_net (latent layer size = 2, 16, 256; 20 epochs)**

A screenshot of a computer

Description automatically generated with medium confidence A screen shot of a computer

Description automatically generated with low confidenceA picture containing text, display

Description automatically generated

**With dis\_net (latent layer size = 2, 16, 256; 100 epochs)**

A picture containing text, monitor, screen, electronics

Description automatically generated A picture containing text, monitor, screen, display

Description automatically generated A picture containing monitor, screen, display

Description automatically generated

Additionally, we use the code from PyTorch-GAN at <https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/aae/aae.py>, which provides a good implementation of the question, as well as the paper by Makhzani et al. (2015). The paper is the first to describe the adversarial autoencoders (AAE). In the implementation, fake images are generated through the reparameterization trick. Please see the attached notebook for details. With the continuation of training for more epochs, the autoencoder can generate highly similar images to those from the original dataset, as shown below (sample indices = 0, 400, 2000, 20000, 80000, 160000).

A picture containing building

Description automatically generated A picture containing grater, keyboard

Description automatically generated

A picture containing grater, electronics, keyboard

Description automatically generated Shape, arrow

Description automatically generated

Shape, arrow

Description automatically generated Shape, arrow

Description automatically generated