|  |  |  |
| --- | --- | --- |
| **Name**: Parashara Ramesh | **Student Id**: A0285647M | **NUSNET Id**: E1216292 |

# 

# Table of contents

[**Table of contents 1**](#_v2wyxfpnc7zk)

[**Note 2**](#_eph9zwel13f)

[**Question 1 (Optimization) 3**](#_r7gac2cbadjv)

[1.0 Finding the equation for the graph 3](#_7jq9rxabxr2q)

[1.1 What happens on gradient descent? 3](#_501c9brhxbb5)

[Derivation 3](#_iiyer7uvirst)

[1.2 What happens on applying adam optimizer? 4](#_e58f8gmbx4xk)

[1.3 Plotting the graphs for adam parameters for ‘L’ 4](#_1fanfex07ttv)

[Explanation 5](#_z562jv8d5onc)

[1.4 Plotting the graphs for adam parameters for different functions 6](#_5hjb085z8xhu)

[1.4.1 Relu 6](#_vnvpcgnwo8s9)

[Explanation 6](#_8l4am562hari)

[1.4.1 Modulus/Abs 8](#_ax9qrbt00bv)

[Explanation 8](#_jhivi2vep4gj)

[1.4.1 Sigmoid 10](#_orca7qr9fgth)

[Explanation 10](#_ujij72cs1e6k)

[**Question 2 (Autoencoders) 12**](#_f68nsc12l8qy)

[2.1 Fully connected Linear Autoencoder 12](#_5p78jfk12a1u)

[With Lambda 2 12](#_mucbtoe3md88)

[Training Loss vs Epochs 12](#_tl6b0hd3f3ii)

[Reconstruction 12](#_9vkrfkccfoql)

[With Lambda 32 14](#_w6napr5gh8n)

[Training Loss vs Epochs 14](#_aqloun6ve5nb)

[Reconstruction 14](#_asmtz498rcz2)

[Explanation 15](#_vbtonwn8c9c2)

[Reconstruction Interpretation 15](#_dl0qzdb5mgji)

[2.2 CNN Autoencoder 16](#_so4ztnhzksoj)

[With Lambda 2 16](#_89bllcuzcj6g)

[Training Loss vs Epochs 16](#_6bz2zpdzatbm)

[Reconstruction 16](#_7sv7z7d4gow8)

[With Lambda 32 18](#_9htt72gpl152)

[Training Loss vs Epochs 18](#_13dyjbi0xme2)

[Reconstruction 18](#_1dlgw7jzdafs)

[Explanation 19](#_7x5k2h4mpz1d)

[Reconstruction Interpretation 19](#_s6ij84kfltvi)

[Training loss vs epochs Interpretation 19](#_lxemsqe8cihb)

[**Question 3 (Common features in an Autoencoder) 20**](#_d4kbdt3iks4h)

[Interpretation 20](#_4tmpw6cxkntr)

[**Question 4 (Understanding Pytorch) 21**](#_dnvscrt4iby2)

[**Question 5 (CIFAR 10) 21**](#_akdckk7tp5dd)

[5.1 Training on the full dataset 21](#_mvqymahutqv0)

[5.2 Training on subsets of the dataset 24](#_bawcdznhqho6)

[**Question 6 (Receptive Field Computation) 25**](#_cimi4m4184xv)

# Note

* All of the code is present inside the respective folders and named as qi.ipynb ( where i is the problem number)
* Since I write some of the common classes in python files, I had to follow this folder structure so that the common classes could be imported in other questions as well
* Alternatively the code is also committed in my github repository (<https://github.com/ParasharaRamesh/NUS-CS5242-Neural-Networks-and-Deep-Learning/tree/main/Assignment%202%20(Autoencoders%20%26%20CNNs)> if that is easier to read instead of the code submitted.

# Question 1 (Optimization)

## 1.0 Finding the equation for the graph

The equation for this graph corresponds to the following formula shown below

a. L = f(x)  
-> 1 - x {x E [0,1)}  
-> x - 1 {x E [1,1+h)}  
-> 1 - x + 2\*h {x E [1+h,1+2h]}

b. Therefore, the partial derivative for this function in the specific ranges mentioned above are

= \frac{\partial L}{\partial x} = \frac{\partial f(x)}{\partial x}
%0ce39ea6-5a75-4a68-9672-3476c7540c1c

-> -1 {x E [0,1)}  
-> +1 {x E [1,1+h)}  
-> -1 {x E [1+h,1+2h]}

## 1.1 What happens on gradient descent?

Assuming h = 0.5 means the equations and partial derivatives change like this

a. L = f(x)  
-> 1 - x {x E [0,1)}  
-> x - 1 {x E [1,1.5)}  
-> 2 - x {x E [1.5,2]}

b. Therefore, the partial derivative for this function in the specific ranges mentioned above are

= \frac{\partial L}{\partial x} = \frac{\partial f(x)}{\partial x}
%0ce39ea6-5a75-4a68-9672-3476c7540c1c  
 -> -1 {x E [0,1)}  
-> +1 {x E [1,1.5)}  
-> -1 {x E [1.5,2]}

### **Derivation**

let x0 = 0

. x1 = x0 - 0.3\*(-1) = 0.3 {as x0 E [0,1)}  
. x2 = x1 - 0.3\*(-1) = 0.3 + 0.3 = 0.6 {as x1 E [0,1)}  
. x3 = x2 - 0.3\*(-1) = 0.6 + 0.3 = 0.9 {as x2 E [0, 1)}  
. x4 = x3 - 0.3\*(-1) = 0.9 + 0.3 = 1.2 {as x3 E [0, 1)}  
. x5 = x4 - 0.3\*(+1) = 1.2 - 0.3 = 0.9 {as x4 E [1, 1.5)}

Here we can see that x5 is the same as x3. Which means that the values will keep switching 0.9 and 1.2 forever and there will be no convergence.

The natural question which arises then would be, assuming we actually stopped at the point when the curve changes equation to x-1 would that have been a natural stopping point?

In that particular case the answer is still no as there is no derivative defined for sharp points. ( if it was instead a smooth point instead of a sharp one a derivative could have probably been definable )

## 1.2 What happens on applying adam optimizer?

In my code, I implemented adam and kept logging what the values of x are after each “adam update” for different values of “h”.

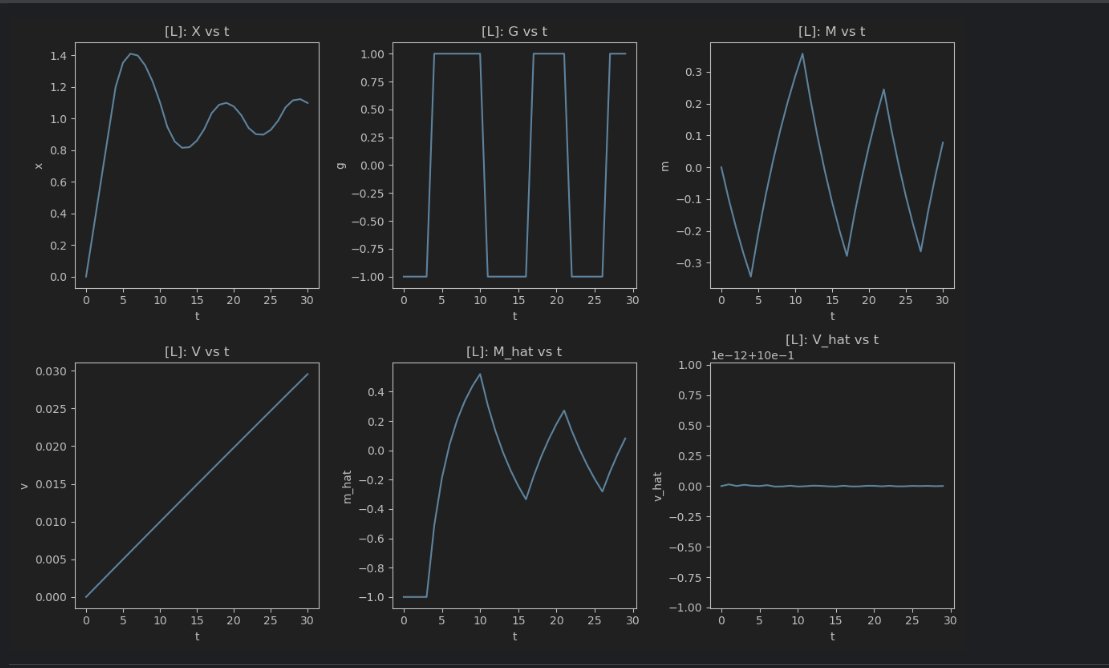
Using binary search and simple trial & error I figured out 2 limits  
0.41018425 (crosses the hump) -> 0.41018450 (where it cant cross)

Therefore after rounding it down to 2 decimal places the answer is just 0.41.

If we have to find this limit out theoretically we would have to use the general equation from 1.0 and solve the value using inequalities but the approach of using trial and error was more easier for me.

## 1.3 Plotting the graphs for adam parameters for ‘L’

(**NOTE**: I will be referring to **m** as velocity and **v** as acceleration as that is more intuitive way to think about it from a physics perspective)

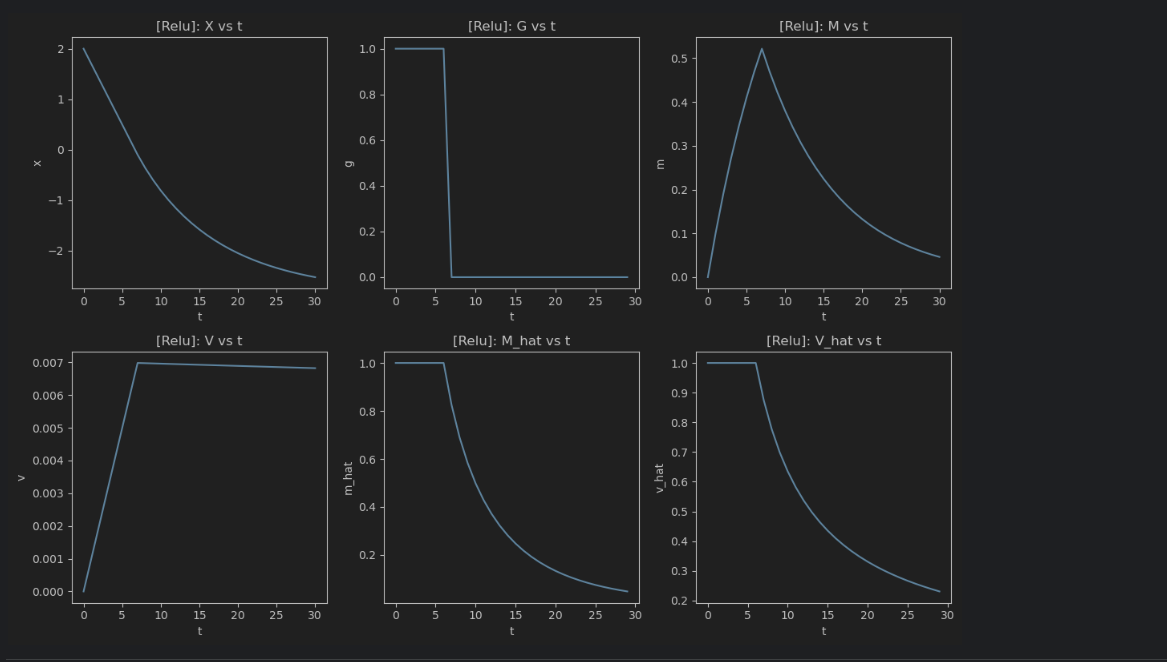


#### **Explanation**

* X vs t
  1. Here we can see that the value of x keeps oscillating primarily because it is unable to cross the bump ( due to the high ‘h’ value of the hump). Therefore it keeps trying to climb it but fails and falls back down the hump which corresponds to increasing and decreasing values of x respectively
* G vs t
  1. Since our piece-wise function has different gradients based on the position of x the values keep jumping between -1 and 1 continuously.
  2. The jump also keeps happening because even x is also oscillating when trying to cross the hump
* M vs t
  1. Even here the velocity greatly varies corresponding to the oscillations in x.
  2. We can see that it increases for a while because there is a slops in the L graph when starting from x =0 till x=1 in the original loss function provided in the question.
  3. The velocity then decreases as x is also not able to escape the hump and increases again when it tries to gather some velocity and acceleration to cross the hump again ( which keeps failing because of the high h value)
* V vs t
  1. This is a gradual increase suggesting that the changes in velocity ( M vs t) is roughly quadratic) .
  2. We can visualize this as a ball rolling back and forth as it is not able to cross the hump.
* M-hat vs t
  1. This is the bias corrected graph for m ( ie velocity) so that the starting point is not at 0
  2. We can see that it is somewhat very similar to M vs t just that the starting point is different and the rest of the graph is slightly transformed
* V-hat vs t
  1. This is the bias corrected graph for v ( ie acceleration) so that the starting point is not at 0
  2. We can see that it this graph is very similar to V vs t just that after bias correcting it is a horizontal line
  3. Which means that the actual change in velocity (M\_hat vs t) is negligible resulting in a near constant ( V\_hat vs t) even though the (V vs t) graph suggests otherwise.
  4. Since we care about the bias corrected values in the end we have to give more importance to the shape of these graphs ( M\_hat and V\_hat) than the original graphs ( M and V )

## 1.4 Plotting the graphs for adam parameters for different functions

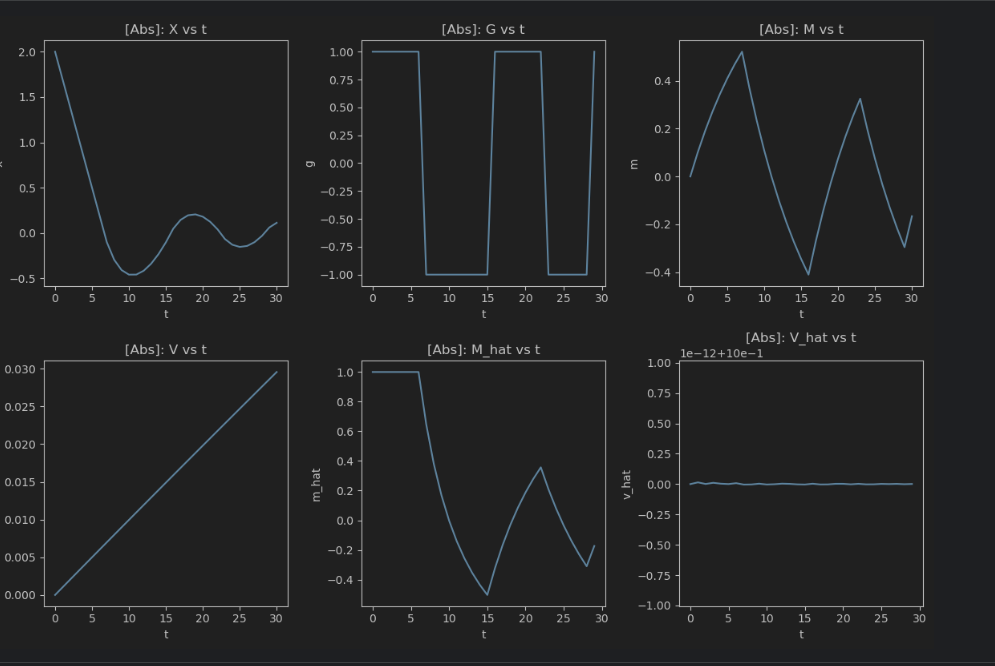
### 1.4.1 Relu



#### **Explanation**

* X vs t
  1. We can see that the x value is monotonically decreasing
  2. This is because once we start at x=2, the min of that function is somewhere close to x=0 so initially the Gradients help it get till x=0
  3. Once we get to x=0 because of the high momentum and velocity it still continues to decrease to negative values even though the gradient at this point in time is 0
  4. In the m(velocity) and v(acceleration) graphs it can be seen that after building up to some value there is a gradual decrease in those values thereby enabling the x value to plateau / flatten out during the last few iterations
* G vs t
  1. The gradient of Relu can either be +1 or 0 which can be clearly visualized in this graph
  2. The moment x cross 0 the gradient switches to 0
* M vs t
  1. m here can be considered as the velocity as it is the first moment. Which largely depends on the previous time stamp’s velocity along with some negligible impact of the gradient
  2. We can see that the velocity keeps building up until it reaches a peak (which happens when the updated x reaches close to 0)
  3. From that point since the gradients are all 0 the velocity value gradually keeps decreasing.
  4. This starts from 0 as the first velocity value (m0) is actually 0
* V vs t
  1. v here can be considered as the acceleration term as it is the second moment. Even here we largely depend on the previous iteration’s acceleration value more than the impact from the gradient
  2. Even here we can see that the acceleration increases rapidly but then stays relatively constant as the m value ( or velocity) is changing only gradually
* M-hat vs t
  1. This is the bias corrected graph for m ( ie velocity) so that the starting point is not at 0
  2. We can see that it is somewhat very similar to M vs t just that the starting point is different and the rest of the graph is slightly transformed
* V-hat vs t
  1. This is the bias corrected graph for v ( ie acceleration) so that the starting point is not at 0
  2. We can see that it this graph is very similar to M\_hat vs t and there is a gradual decrease in acceleration (v-hat) which corresponds to the gradual decrease in velocity(m-hat)

### 1.4.1 Modulus/Abs

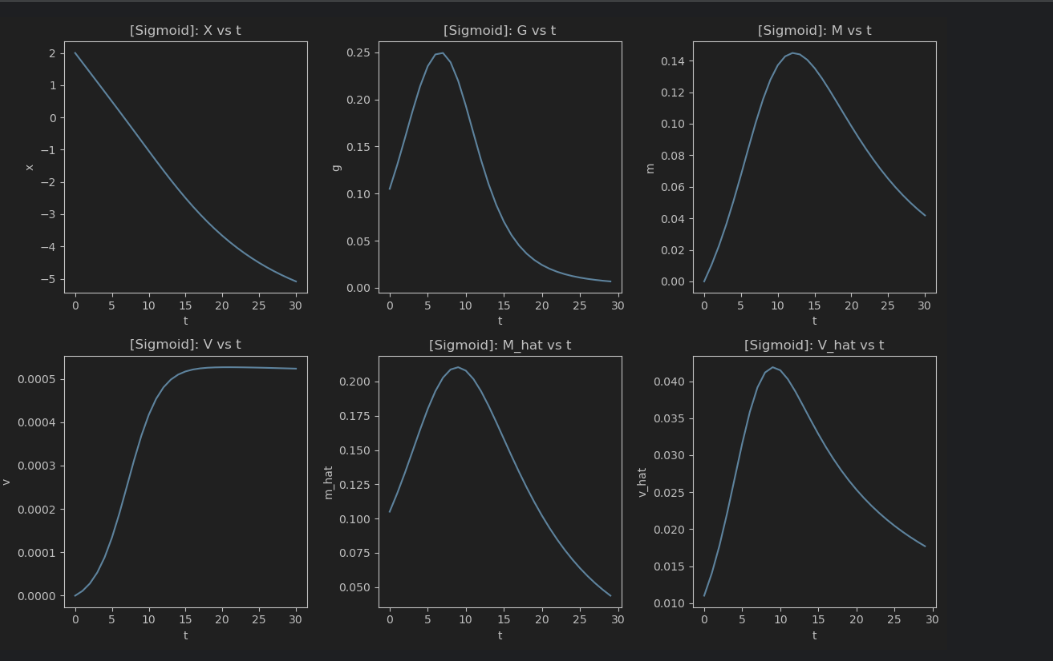


#### 

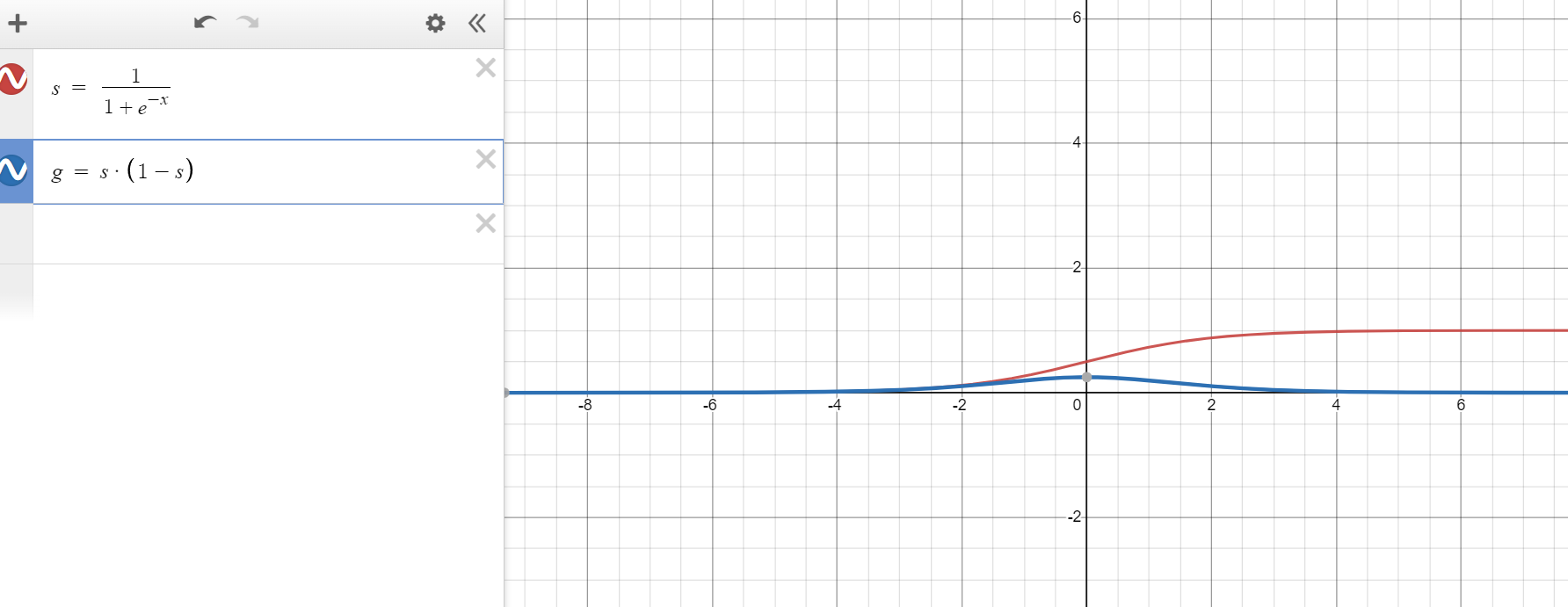
#### **Explanation**

* X vs t
  1. We can see that the x value is not decreasing and also increases in bumps as more iterations pass
  2. This is because once we start at x=2, the min of that function is somewhere close to x=0 so initially the Gradients help it get till x=0
  3. Once we get to x=0 , the gradient becomes -1 as that is the gradient of the Abs function for values lesser than 0 which contributes to the value of x increasing slightly rather than decreasing
  4. In the velocity graph (m) we can see that it widely keeps oscillating back and forth ( which is due to the nature of the G graph) therefore troughs of this graph correspond to the peaks of the M vs t graph ( as during downhill the velocity is high but during uphill the velocity is low)
* G vs t
  1. The gradient of abs can be either -1 or +1 therefore we can see that this graph greatly oscillates between these two values based on the value of x.
* M vs t
  1. m here can be considered as the velocity as it is the first moment. Which largely depends on the previous time stamp’s velocity along with some negligible impact of the gradient
  2. In the velocity graph (m) we can see that it widely keeps oscillating back and forth ( which is due to the nature of the G graph) therefore troughs of this graph correspond to the peaks of the M vs t graph ( as during downhill the velocity is high but during uphill the velocity is low)
  3. This starts from 0 as the first velocity value (m0) is actually 0
* V vs t
  1. v here can be considered as the acceleration term as it is the second moment. Even here we largely depend on the previous iteration’s acceleration value more than the impact from the gradient
  2. Here we can see that there is a linear increase in the acceleration which indicates that the velocity (v) is changing in a roughly quadratic manner. As the v graph can be interpreted loosely to be the change in the M vs t graph
  3. Even here it starts from 0 as the first acceleration value(v0) is 0
* M-hat vs t
  1. This is the bias corrected graph for m ( ie velocity) so that the starting point is not at 0
  2. We can see that it is somewhat very similar to M vs t just that the starting point is different and the rest of the graph is slightly transformed
* V-hat vs t
  1. This is the bias corrected graph for v ( ie acceleration) so that the starting point is not at 0
  2. We can see that it this graph is very similar to V vs t just that after bias correcting it is a horizontal line
  3. Which means that the actual change in velocity (M\_hat vs t) is negligible resulting in a near constant ( V\_hat vs t) even though the (V vs t) graph suggests otherwise.
  4. Since we care about the bias corrected values in the end we have to give more importance to the shape of these graphs ( M\_hat and V\_hat) than the original graphs ( M and V )

### 1.4.1 Sigmoid



#### **Explanation**

* X vs t
  1. We can see that x monotonically keeps decreasing smoothly as we know that its minimum occurs when x reaches -infinity asymptotically
  2. Since it is a smooth, continuous curve even the gradients are much smoother
* G vs t
  1. 
  2. Here the blue graph is the derivative of sigmoid which has a small bump at 0. Therefore its minimum lies on both sides of 0
  3. We can see that the graph of G vs t also corresponds to this bump. Once the x value gets over this bump when coming from x=2 it is able to decrease smoothly
* M vs t
  1. Here we can see that the velocity is increasing until reaching a peak when x is approximately -3 which corresponds to the speed a ball would roll if it is let to roll on the red graph from the right side
  2. Once it crosses the x=-3 threshold the velocity decreases gradually
  3. Even here the value starts from 0 as the first value of velocity m0 is 0
* V vs t
  1. Here we can see the corresponding increase in the acceleration until the peak in the velocity graph ( M vs t) after which it stays constant indicating that there is not much change in the velocity values after that
  2. Even here the value starts from 0 as the first value of acceleration v0 is 0
* M-hat vs t
  1. This is the bias corrected graph for m ( ie velocity) so that the starting point is not at 0
  2. We can see that it is somewhat very similar to M vs t just that the starting point is different and the rest of the graph is slightly transformed
* V-hat vs t
  1. This is the bias corrected graph for v ( ie acceleration) so that the starting point is not at 0
  2. We can see that it this graph is very similar to M\_hat vs t and there is a gradual decrease in acceleration (v-hat) which corresponds to the gradual decrease in velocity(m-hat)

# 

# 

# 

# 

# 

# Question 2 (Autoencoders)

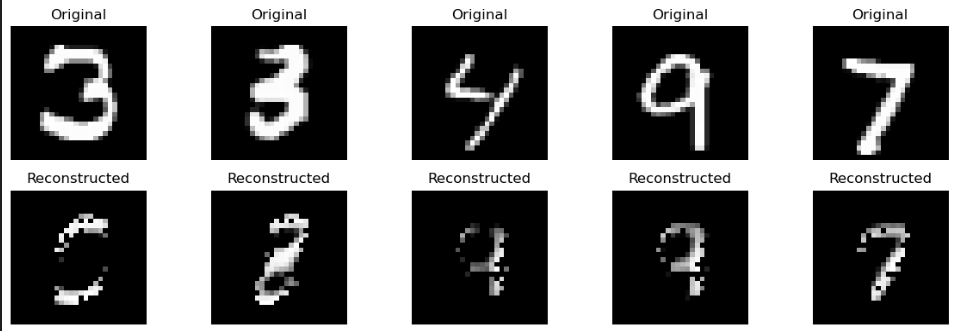
## 2.1 Fully connected Linear Autoencoder

### With Lambda 2

#### Training Loss vs Epochs

#### 

#### Reconstruction

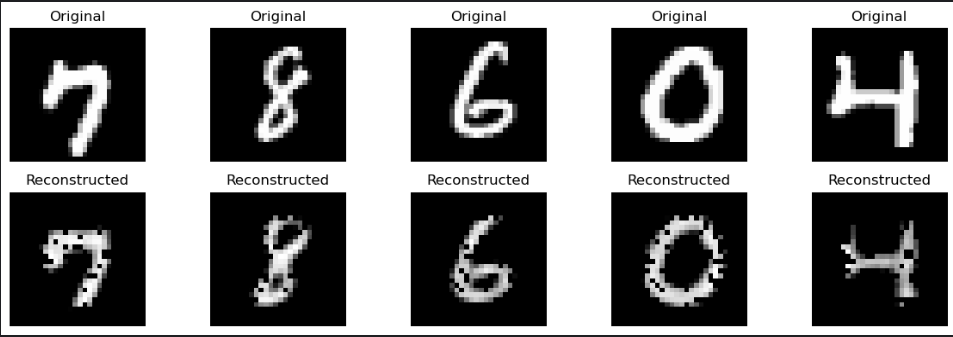


### With Lambda 32

#### Training Loss vs Epochs

#### 

#### Reconstruction



#### Explanation

##### Reconstruction Interpretation

It is clear that for lambda 32 the reconstruction is much better than lambda 2. The reason for that is because it is better to have a latent feature dimension size of 32 instead of 2 as that is a more natural fit encoded/compressed representation of an MNIST image.

For the lambda2 model many times even the reconstruction does not match the ground truth as representing 10 different classes of digits in just a small latent dimension of 2 can confuse the model when it attempts to pass it through the decoder.

On the other hand for lambda 32 the reconstruction images are very good and follows the exact shape of the ground truth image ( just that it has some minor holes which indicates that there is still scope for improving this model)

Training loss vs epochs Interpretation

For the lambda 32 model, the loss is consistently decreasing as there is a lot of scope for the model to learn the latent features better as a latent dimension of 32 is a much better fit than a latent dimension of size 2.

For the lambda 2 model, the training loss is much more erratic which indicates that the model is getting confused or rather learning something and then quickly unlearning and relearning the same concepts again. This is primarily because of the fact that since the latent dimension is very small the distance between the encoded representation of each class is very close in the latent space. Therefore the model does not clearly know how to proceed decoding this image which is why it gives the wrong reconstruction for an entirely different image at times.

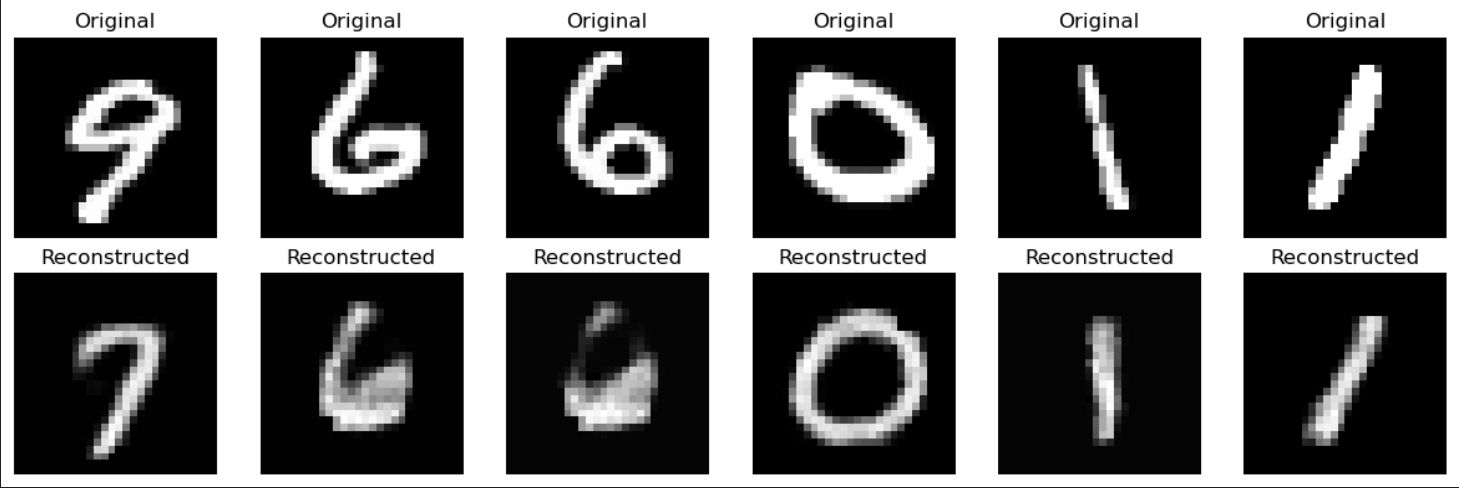
## 2.2 CNN Autoencoder

### With Lambda 2

#### Training Loss vs Epochs

#### 

#### Reconstruction

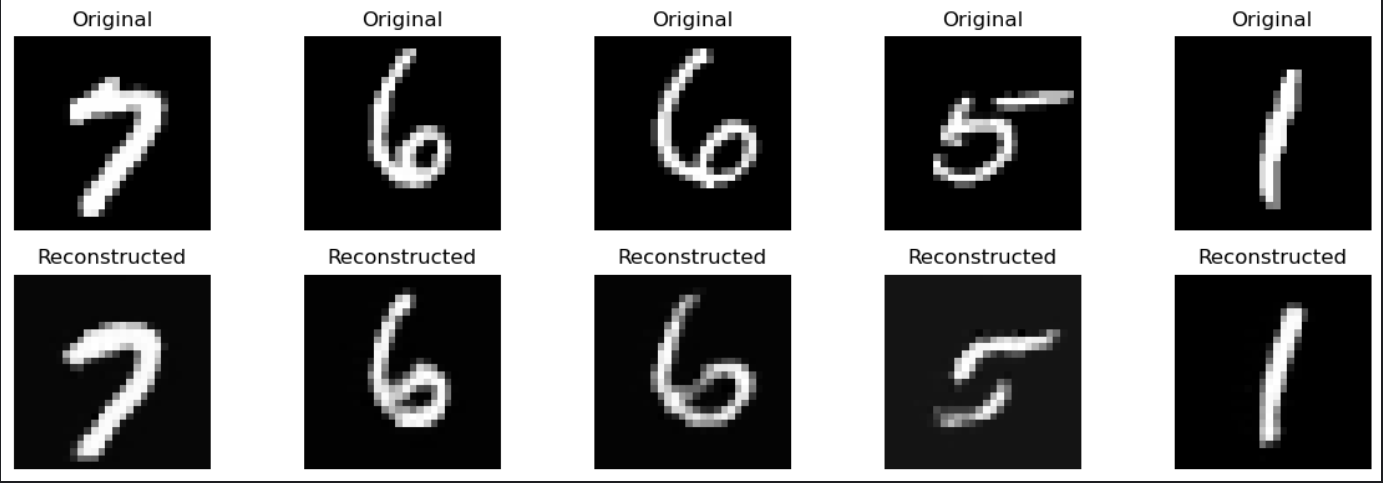


### With Lambda 32

#### Training Loss vs Epochs

#### 

#### Reconstruction



#### Explanation

### Reconstruction Interpretation

Since CNNs in general are better than fully connected neural networks for images it is quite visible that the reconstructed images are much better than their fully connected counterparts.

Even here the lambda 32 model has better reconstructed images than the lambda 2 model because it has a larger latent feature dimensions which can represent the MNIST data much better than having a really small latent feature dimension of just 2

### Training loss vs epochs Interpretation

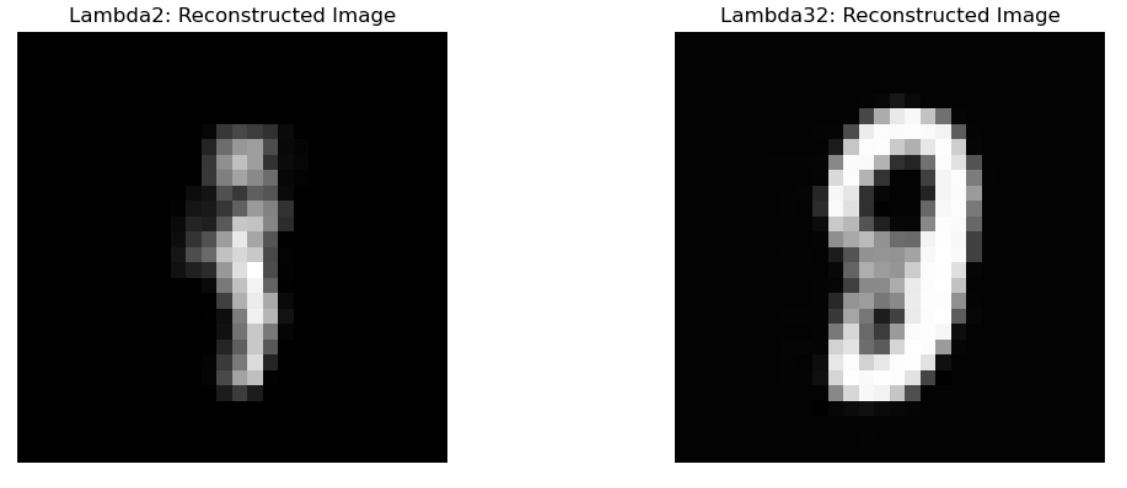
Even here the training loss for the lambda 2 model is slightly erratic as we can see that the loss increases and then suddenly decreases which is akin to the model learning something , then getting confused and then relearning the concepts again.

Whereas the lambda 32 model has loss values which are continuously decreasing in a monotonic fashion as the value of 32 is much better than 2 for the latent feature dimensions

Additionally, since the model is now a Convolutional Auto Encoder it is able to learn spatial data much better than the fully connected linear autoencoder.

### 

# Question 3 (Common features in an Autoencoder)



## Interpretation

For both images since we are taking the average of all latent features for the entire training dataset there will be some kind of blurring effect in the reconstructed image as in some sense we are finding a compromise between all input digits and trying to find the most "average" looking digit.

Additionally, we can see that the images are lacking clarity as well as we cannot clearly classify this reconstruction as belonging to one particular class.

In case of lambda 2 since the model itself is not that powerful it has only learnt the most basic features which is why the blurred reconstruction image just shows the parts which is common in most digits. ( which in this case is most probably the right hand side line present in numbers like 7 and 9 etc)

However, in the case of lambda 32 model it has learnt that the most "average looking" number is something which looks close to an 8 which makes sense since in most digital clocks all digits can be representable using the parts of the digit "8".

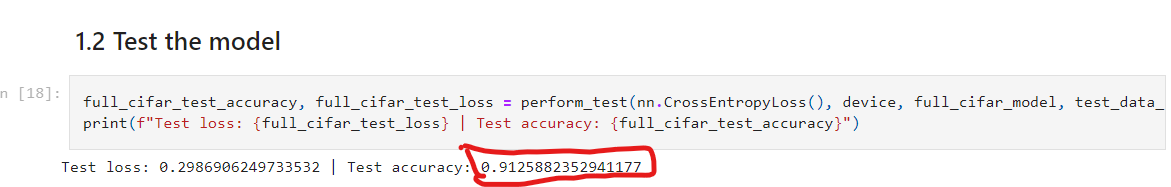
This kind of high level/abstract concept would have been learnt across all the handwritten digits in the dataset.

# Question 4 (Understanding Pytorch)

Refer to the q4.ipynb as each cell also has an assert statement which checks if the output generated from pytorch is similar to the output generated using my implementation. ( which is true for all the 6 functions given the input weights provided)

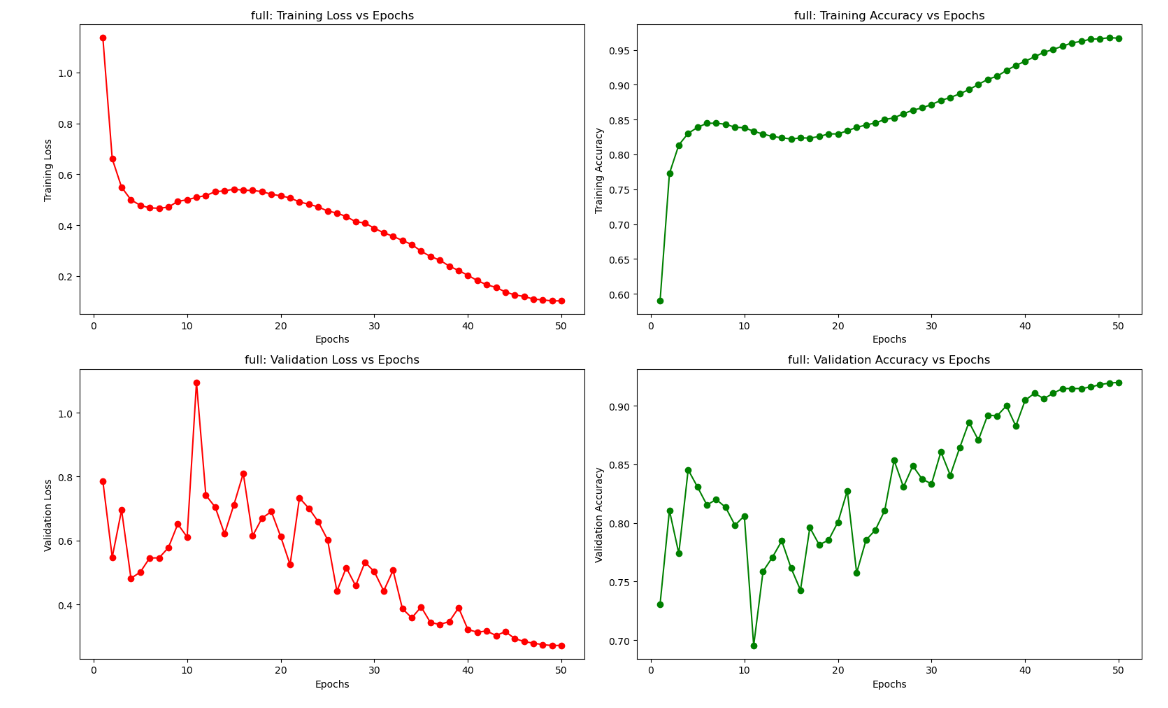
# Question 5 (CIFAR 10)

## 5.1 Training on the full dataset





1. I was able to achieve 91.25% on the full dataset. But I had to iterate over 4-5 different model architectures before I was able to get this accuracy.
   1. Here are some of the reasons why achieving this is hard:
      1. The images are colored images of size 32x32 and the number of images/the size of the dataset is not that large. Therefore it is hard to learn the general insights for classifying 10 different classes just using the standard test dataset
      2. Just using a standard architecture was giving a max test accuracy of only 75-85% as increasing the depth of the model only increases the complexity and the time needed for the model to actually learn the underlying representations. Which was why I decided to use skip connections to get the context from layers even before the immediate previous layer
      3. Since the images are relatively small, performing too many convolutions ensures that we quickly lose track of the big picture which is identifying the class the picture belongs to. Therefore in some sense the model also needs to be not too deep as we may encounter the “vanishing gradient” problem.
      4. Since the task to achieve requires that we train our model from scratch we have to train our model for a lot of epochs for it to even learn something. On the other hand if we had another task to perform ( say finding the bounding boxes of objects) along with an enhanced dataset, we could have performed multi-task multi-label classification which would have enabled us to learn even more generalizable robust representations. But since we are only focussed on the classes its harder for the model to learn those concepts from scratch.
   2. Here are some of the tricks I used to get it > 90% test accuracy
      1. Performed a series of transformations like cropping, rotating etc on the dataset and augmented the original dataset with the randomly transformed images as well (i.e. 2x the size of the original dataset)
      2. Used a custom architecture inspired by [this kaggle notebook](https://www.kaggle.com/code/kmldas/cifar10-resnet-90-accuracy-less-than-5-min). The changes I made were to add more skip connections
      3. Here is the architecture with the red edges being the new connections I introduced on top of the existing architecture. I also added another linear layer with some dropout at the end as well (refer to the diagram above, and for a clearer base model refer to the kaggle notebook link)
      4. Adding dropout also helps the model generalize better during inference which is why I added it at the end of my architecture.
      5. With skip connections the model can take into account the “context” from previous layers before its immediate parent layer which helps the model to see the “big picture” or “birds eye view” which helps the model during inference time
      6. As suggested in this kaggle notebook I also used a learning rate scheduler along with weight decay in the adam optimizer to ensure that the learning rate keeps changing appropriately during training along with regularization being applied to the underlying cross entropy loss function.
      7. I initially also used multi-precision training to speed up the training on my machine but later I skipped this step as I was training on colab
      8. I also used a gradient clipping operation as suggested by the same kaggle notebook link to ensure that the gradients don’t explode due to the fact that my model is moderately deep.



As we can see here the training loss smoothly decreases over time and the training accuracy also smoothly increases over time.

However the validation loss and the validation loss keeps jumping around which is expected as I am using a separate validation dataloader on which the model has not been trained.

The model was trained for a total of 50 epochs

## 5.2 Training on subsets of the dataset

### 

This graph was constructed by plotting the following points which was derived by looking at the values mentioned in the logs:

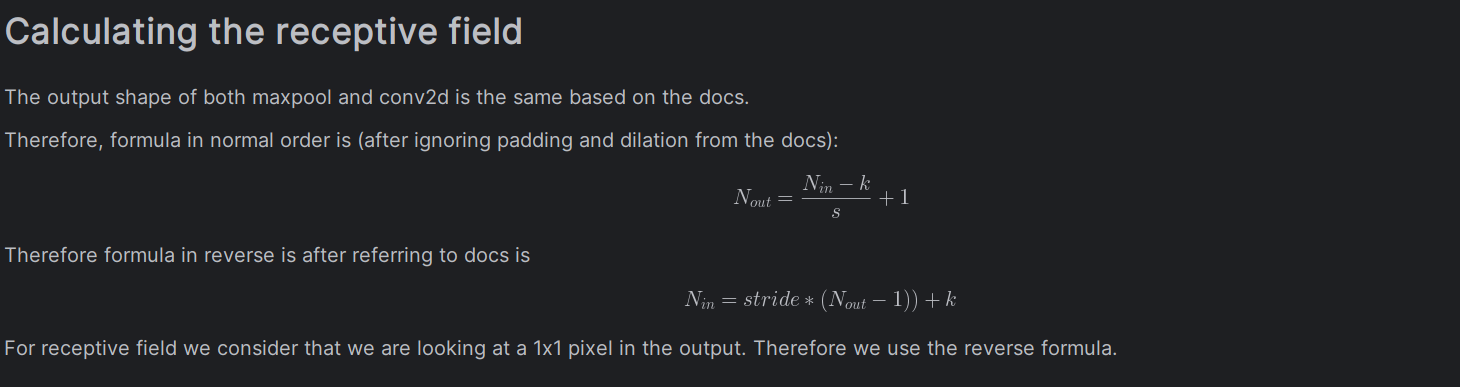
**Num instances: [Train accuracy, test accuracy]**

*1: [1, 0.1308],  
10: [0.48, 0.21423],  
100: [0.72, 0.5189],  
1000: [0.8795, 0.8250],  
50000: [0.96682, 0.9125882]*

# Question 6 (Receptive Field Computation)

Since receptive field refers to that region in the input image latent space which gives rise to one pixel in the output latent representation.

Using this knowledge to work backwards and found the inverse formula.



Q6.ipynb essentially just uses a loop to apply this inverse/reverse formula layer by layer from layer #5 to layer #1 assuming we start with 1.

Here are the answers after computing the receptive fields.

1. 8x8 ( Given )
2. 26x26
3. 15x15,
4. 27x27
5. 20x20
6. 147x147
7. 282x282
8. 18x18
9. 18x18 (Same as question #8)
10. 91x91