

Introduction:

The report tries to analyze the given GLO model from different aspects through different metrics in order to evaluate under which hyper-parameters and losses does he excel, under the current limitations. We start by analyzing the L1+L2 loss and exploring some of his properties, then we tackle his weaknesses and show that a loss based on spatial domain distance is problematic.

Then we propose to monitor the training by hand-picking combinations of content and class embeddings which the model will generate and visualize the activation layers, those metrics should improve our evaluation capabilities. Afterwards we describe how to hand-pick a test data in order to get the most value out of the improved metrics.

We move on to analyzing the hyper-parameters of the model, which are the content embedding, class embedding and noise amount and attempt to find the most fitting ones for our data and model. Equipped with reasonable dimensions and metrics we venture onward to compare and understand from various aspects how does the simple model fare against his weight-decay variation.

Data :

Using MNIST dataset with 2000 images we aim to disentangle the content and class from each the images, and then interpolate them to generate new images based on the content and class we specify.

Class : The different digits serves as the different classes.

Content : The content are more varied and less defined, they're also learned in an unsupervised fashion mostly. Possible common contents that should appear are thick, thin and round hand writing.

Clarification: the concatenated images throughout the report are composed as :

$$Image[:, : 28] = Class \tag{1}$$

$$Image[:, 28 : 56] = Content \tag{2}$$

$$Image[:, 56 :] = Output \tag{3}$$

The class image content is only for illustration, the content in the first image ([:28]) doesn't effect the output at all throughout all the following report.




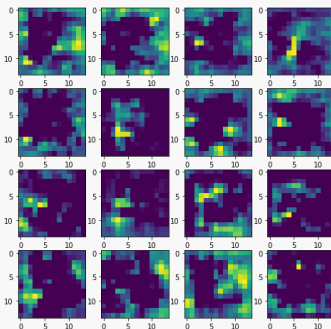
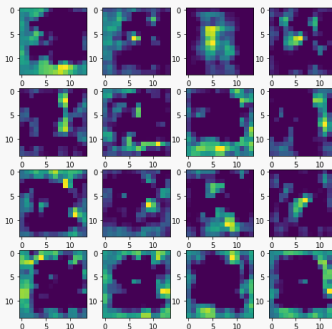
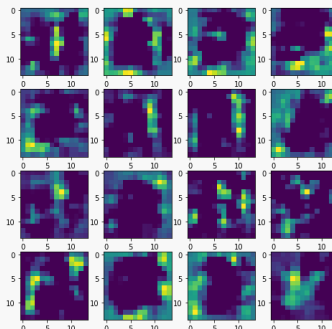
Remark : Examining the metrics empirically before training is not possible, therefore I trained the model in a general way to achieve reasonable results and then analyzed different metrics.

Metrics

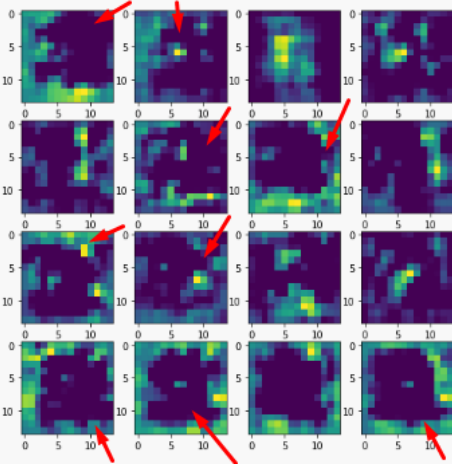
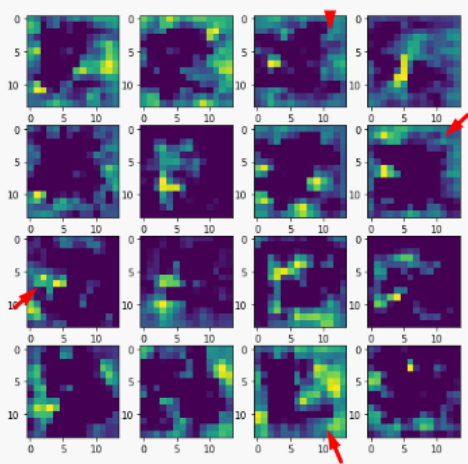
Analysis of the provided loss function

The following section compare the generator results while using exclusively L1 or L2, with out weight decay. I could explain it more thoroughly but the report is becoming too long, even with the large pictures therefore I'll briefly describe the differences.

The training process and convergence became shorter since we optimize only regarding 1 metric, the differences can be seen here

L1 loss	L2 Loss	L1+L2 loss
		
Activation layers , generate digit 2 with thick 4 content vector , epoch 130 	Activation layers , generate digit 2 with thick 4 content vector , epoch 100 	Activation layers , generate digit 2 with thick 4 content vector , epoch 60 
Low freq-waves generation is lacking	High freq-waves generation is lacking	Interpolation of both which yield better results

Activation layers comparison:

	
L2 Activation layers	L1 Activation layers

L1 Activation Layers : 4 out of 16 activation layers of the L1 resemble 2 in some way , and the quality of the resemblance is still lacking. But we can see more refined edges and loss "blob" like activation layers compared to the L2 loss activation layers , so while they resemble the digit less in a global structure , they locally managed to capture the digit edges more accurately. The generated image from the table above also support this.

L2 Activation layers : 9 out of 16 of L2 loss activation layers resemble the digit 2 , we can confidently infer that the model manage to preserve the global structure in high quality.

Therefor L1 captures global structure better , L2 captures local structure better and L1+L2 interpolate between them to get the optimal solution.

Remark : The values in the following section only apply to L1+L2 loss , different losses output different values and comparing the values to gain insights about disparities between the models performance is futile.

Quantifying generation quality based on spatial distance is problematic

Since we can't use perceptual loss we're forced to measure image generating quality based on distance in the spatial domain , as we know small distance in the spatial domain doesn't translate generation quality , and therefor isn't a accurate metric to measure content transfer between classes , the following figure from Stanford n231n shows why distance in the spatial domain is *very* unintuitive

All of the images above are at the same distance from the original while their content is very different.

This trend continuous to our current domain as well , the following table demonstrate the problem (Classic L1+L2 loss):

Class dimension 10 , Content dimension 50	Class dimension 15 , content dimension 50	Class dim 5 , Content Dim 25
Loss : 0.029	Loss : 0.0295	Loss :0.04

In the middle and left figures the results quality is miles a part but the loss different is miniscule. On the other hand in the right figure , her lose is higher then former figures yet she yielded the desired result.

But saying not the right loss for the mission is also not quite right , $he(L1+L2)$ is misleading only when we approach very small values , i.e. , the generation quality of models when their loss is within the following range $[0.1, 0)$ doesn't mean much , as the table above shown that a model with 0.04 loss can be far better then a model with 0.029 , therefor the correlation between loss value and image generation is 0 within that range.

L1+L2 loss isn't all bad

Looking outside at wider ranges , especially in very high values , the correlation between loss and generation quality is quite high. A model with extremely high loss regarding L1+L2 will never generate good results , that's why the first epochs losses is always extreme and the results are lacking , i.e , we can count on him to divert us to the right path but we'll need to take matters into our own hands during the last mile.

Activation layers , generate digit 2 with thick 4 content vector , epoch 0

Generated image from the first epoch ,i.e, very high loss.

The activation map of the layers also show us that high loss corresponds to zero patterns learned.

To conclude , while the L1+L2 loss is indicative for the model results in general , he can't quite catch nuances that make or break a good disentanglement model and therefor counting on him for this delicate mission is a slippery slop.

Proposed solutions

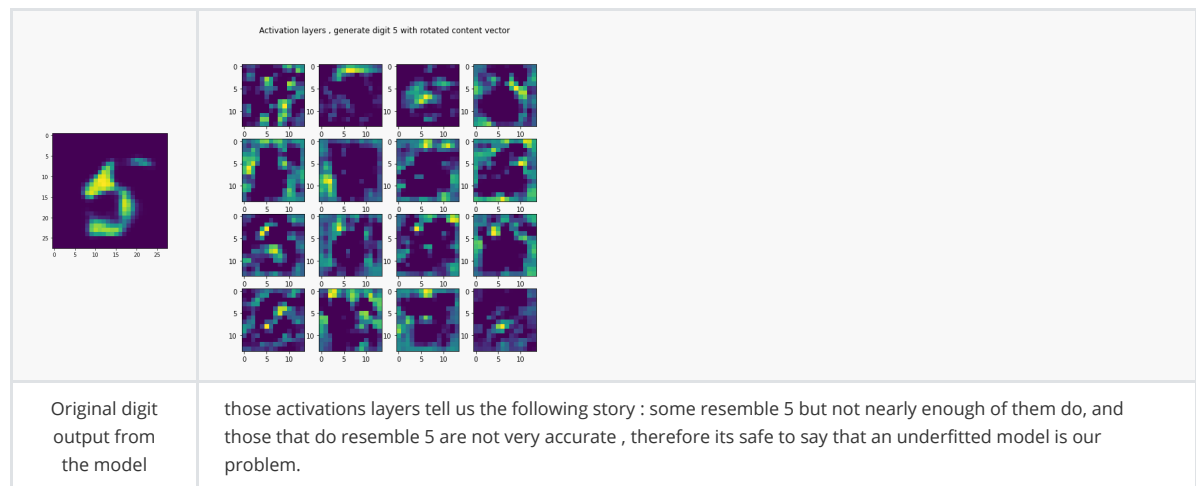
Online image generation

Approximating the performance during training by visualization of hand picked generated examples that we expect the model to succeed on is more reasonable metric for live feedback.

Overfit and disregard to the content vector.	Seems like the content vector is leaking into the class vector	High disentanglement quality.

Layer activation maps

We can also increase the depth of our insights if we visualize the activation layers and combine them with the comparison above. Visualizations of the activation layers during training allow us to inspect the basic building blocks the model combine to generate the output, those insights will allow us to understand it on a deeper level and therefor find the problem he's currently facing more accurately and attempt to fix her.



Perceptual loss :

Overview

1. Pick conv layers from the VGG before the training.
2. Build a loss network that passes images through the VGG19 net and and saves some of the activation layers , i.e. , we observe which neurons the image "light up" in already a pretrain network with very high capabilities , which means that the image representation in the hidden-layers would hold meaningful semantics.
3. Build a loss function by comparing the generated image representation in the hidden layers to the original image by an Euclidean distance. An Euclidean distance is acceptable here compared to the L1+L2 loss since we're in a latent space where distance between different latent vectors is more accurate regarding the semantic information the image holds.

Why does perceptual loss surpasses L1+L2

- Euclidean distance is more meaningful in we forsake the spatial domain and compare the image representation in the hidden-layers.
- Both local and global structure is accurately described in the inner-dimension of VGG19 with a high-level of non-linearity features that far surpass the l loss functions that only find linear patterns such as L1 or L2.

In other words , she estimate local and global structure more accurately thanks to VGG19 and the distance is *much* more representative in the latent space of the inner-dimension then in spatial domain , therefor she surpass out previous losses on all accounts.

I tried to use this loss (The code is in the notebook) , after combining the function into the model and preprocessing the data to fit for VGG19 but I realized that the generator provided for us only generated images of size 28x28 but the VGG19 only receives images of size 224x224 and resizing the image is not a differentiable , therefor I couldn't train the model with perceptual unless I changed the output dimension of the generator which in turn would change the whole architecture.

Metrics evaluation conclusion :

Using those metrics during training allow us to upgrade our evaluation process dramatically and stop the training at an ideal points that's not based on spatial domain distance which is incredibility inaccurate in very small values.

Which images to hand-pick for evaluation ?

The images will be chosen each to meet one of the following criteria's , each criteria present a desirable disentanglement property we would like the model to learn.

Image generation

Image generation under the simplest setting , i.e. , images he saw during the training , i.e. , both class and content vector originate from the same digit. This metric serves as a sanity check , if our model can't even handle that then disentanglement is out of the question.

Disentanglement

Measuring whether different content vectors change the given digit content in a desirable way.

Test set importance

The images to measure this metric should also be done carefully, if the content and class embeddings originate from the same digit then we're just checking our model disentanglement quality over the training set and as we know it's a misleading metric. Therefore mixing content and class embeddings from different digits results in a test set which the model hasn't really seen before and it'll provide us with an accurate measure for the model capabilities regarding disentanglement.

Class embeddings nearest neighbors

A good model should learn a latent space that maps digits with the same geometric properties closer to each other, therefore I measured the convergence and nearest neighbor embeddings after 100 epochs to understand if the model managed to capture the geometric resemblances of the digits. Each digit was given an ideal match that's based on my understanding of their geometric resemblance.

Ideal nearest digits :

0 Ideal nearest digits : 3, 8, 9 all partly resemble an ellipsoid.

1 Ideal nearest digits : 4 and 7 both include a long line in them

2 Ideal nearest digits : 2 is a partial 8 beside his extreme lower part, 2 and 5 are kind of mirrored versions of each other along the y axis, also 2 and 3 highly resemble each other regarding the upper part.

3 Ideal nearest digits : 3 is a partial 8, 0 captures both his up and lower geometric shapes.

4 Ideal nearest digits : 4 is harder to capture since no other digit really resembles him, only 1 and 7 can approximate him since they both lack round geometric shapes.

5 Ideal nearest digits : 2 as explained before, 3 because they both show the same lower part.

6 Ideal nearest digits : 9 is a mirrored 6 along the y axis, 6 highly resembles a partial 8, mirroring 3 along the x axis results in a partial 6.

7 Ideal nearest digits : As explained in 4, a lack of round properties forces him to only approximate 1 and 4 even if their structure isn't really that similar.

8 ideal nearest digits : Same reasons as 0 and 3

9 ideal nearest digits : Same reasons as 3 and 8.

Criteria for measuring the disentanglement

Choosing the contents that should be transferable : Since class embeddings embody an unchanging feature, i.e., digit, the content embeddings will describe variations of writing style. The following variation will be considered to test disentanglement :

1. **Thickness**
2. **Thinness**
3. **Rotation**
4. **Round hand-writing**
5. **"Straight" hand-writing**
6. **Digits with missing chunks which mainly stems from fast writing style.**

Unlearnable disentanglement features due to model limitations:

While I'd like to measure the content vectors describing 4, 5, 6 it is unfortunately not doable given the current setting and I'll attempt to explain my line of thinking :

The **Missing chunks** variation is quite rare in the dataset, therefore learning those patterns to present even mediocre results would be very hard given the sample size and model capacities.

Straight\Round Every hand writing is (kind of) either straight or round, therefore rarity isn't the issue, the issue lies in the digits properties. Round digits carry an inherent bias toward round hand writing, straight digits carry an inherent bias toward straight handwriting even if the original writer isn't a straight writer at all, his 0 still must be round. Therefore quantifying this effect correctly requires us to detect which 7 and 1 digits are written with round properties and which 3, 6, 8, 0, 9 are written with straight properties, therefore the relevant data from which we can learn those patterns is rare, even if every person's hand writing is kind of binary regarding roundness.

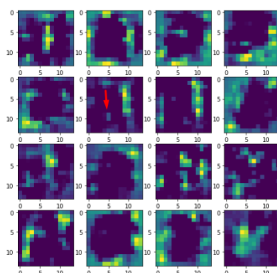
Hyper parameter search analysis

Class Embedding dimension

I did a limited search on the following values : 5, 10, 25, 50 and checked which class dimension produces the best results regarding feature maps learned, I used class embedding 2 and content vector of 4 with thick writing to compare them, I do realize that taking only 1 sample may be inaccurate by time constraints limited me to sample further.

Class embedding dimension 5	Class embedding dimension 10	Class embedding dimension 25	Class embedding dimension 15	Class embedding dimension 50
Activation layers - generate digit 2 with thick 4 content vector - epoch 10 	Activation layers - generate digit 2 with thick 4 content vector - epoch 140 	Activation layers - generate digit 2 with thick 4 content vector - epoch 140 	Activation layers - generate digit 2 with thick 4 content vector - epoch 140 	Activation layers - generate digit 2 with thick 4 content vector - epoch 140

The most interesting thing I could find is that the rest of the dimensions couldn't capture the little circle that's caused when people write 2, but the embedding with dimension 5 manage to capture it, while its not very clear in the generated image a single activation layer tells us this info :



I assume that given enough training data and capacity most of the feature maps would include them to represent this popular writing style of the digit 2, or maybe it appears only in 1 activation layer because its proportional to his likelihood in the dataset? It will remain as an open question since measuring it isn't doable.

Since the generated results of dimension 5 feels most natural and thick as the content vector , on top of capturing the "circle" , I'll conclude that 5 is the best dimension and move on forward with him. Also it make sense that representing 10 different class embeddings can be done in 5 dimensions.

The next part will try to find a good value for the class embedding dimension :

Content embedding dimension

I tired the following dimensions : 10 , 25 , 50 , 100 and trained each model for 150~50 epochs , early stopped at the ideal point performance wise.

Content embedding dimension 10	Content embedding dimension 25	Content embedding dimension 50	Content embedding dimension 75	Content embedding dimension 100
Activation layers - generate digit 2 with thick 4 content vector - epoch 10 	Activation layers - generate digit 2 with thick 4 content vector - epoch 10 	Activation layers - generate digit 2 with thick 4 content vector - epoch 150 	Activation layers - generate digit 2 with thick 4 content vector - epoch 140 	Activation layers - generate digit 2 with thick 4 content vector - epoch 150

Surprisingly so 100 dimensions quality is pretty good , but he didn't manage to capture the thickness aspect of the given content vector 25 dimension will move forward as the chosen dimension. Therefor the final vector is of the following structure:

Representation vector type	Dim
Content	25
Class	5




I was hoping to see clearer patterns in the activation maps that would help me find disparities between the hyper-parameter performance but they weren't helpful to find the optimal dimension and the generated imaged helped more. Maybe its because of the model capacity or maybe its my inexperience that's holding me back from interpreting them clearly.

Noise levels

the following section measure different values of noise impact on the generator results.(I Define σ as notation for noise in the upcoming section)

Very high σ values

The impact of very high $\sigma = 100$ results in the following figure :

	Class 1 , Content current 6 , results a unrecognizable pattern.	Class 2 , Content current 4 , results a unrecognizable pattern.	Class 9 , Content current 4 , results a unrecognizable pattern.
			

Further support for this claim can be found in the activation maps for class 2 and content 4



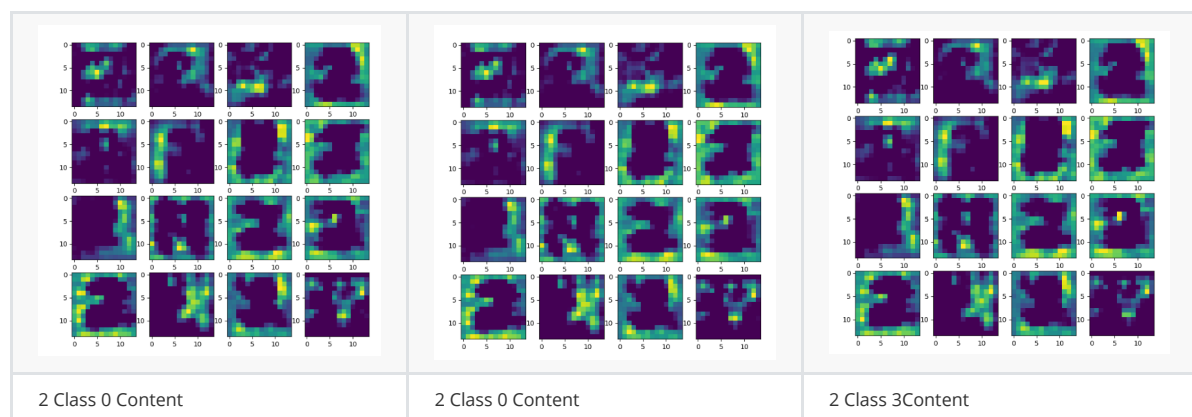
The content embedding doesn't reveal any information anymore , he's just adding *extreme* σ to our generator which force him to produce results with out any understandable patterns.

high σ values :

The impact of high $\sigma = 5$ results in the following figures :

		
2 Class 0 Content	2 Class 0 Content	2 Class 3 Content

Therefore , high noise values , such as 5 for example results in a complete disregard to the given content , i.e. , the class embedding is the *only* factor that decides the properties of the generated image , and if we disregard the content then our generator is only capable of producing 10 different images , which is suboptimal. The reason behind it stems from the inability of the generator to learn from the an extremely noisy content embedding , therefore he manages to only recognize patterns in the class embeddings and as a result they become the deciding factor.



Remark : The content of the left and middle are different.

The activation layers support the idea since they're almost indistinguishable from one another even though each one has a different content vector, therefore we know for certain that the content has no influence at any capacity over our output as the noise increases.




Attempting to explain the dispersity between $\sigma = 10$ and $\sigma = 100$

The results for the high-noise values were vastly different, but why were they different, shouldn't the network understand that the content is only misleading and ignore it as she did in $\sigma = 10$? My guess is that for the network to disregard the content embedding a learning process must occur where she realizes that ignoring it is her best option. In extremely high values noise setting, the input range is much wider, therefore for the network to figure out that a vector in the range of $[-100, 100]$ does not contain any pattern she needs considerably more time than for example, figuring out if a vector in a smaller range, e.g., $[-1, 1]$ is patternless (Assuming the same "density" because in theory the interval $[-1, 1]$ is infinite, i.e., he can express spaces with infinite complexity).

Therefore my intuition suggests that prolonging the training enough for $\sigma = 100$ would produce the same results.

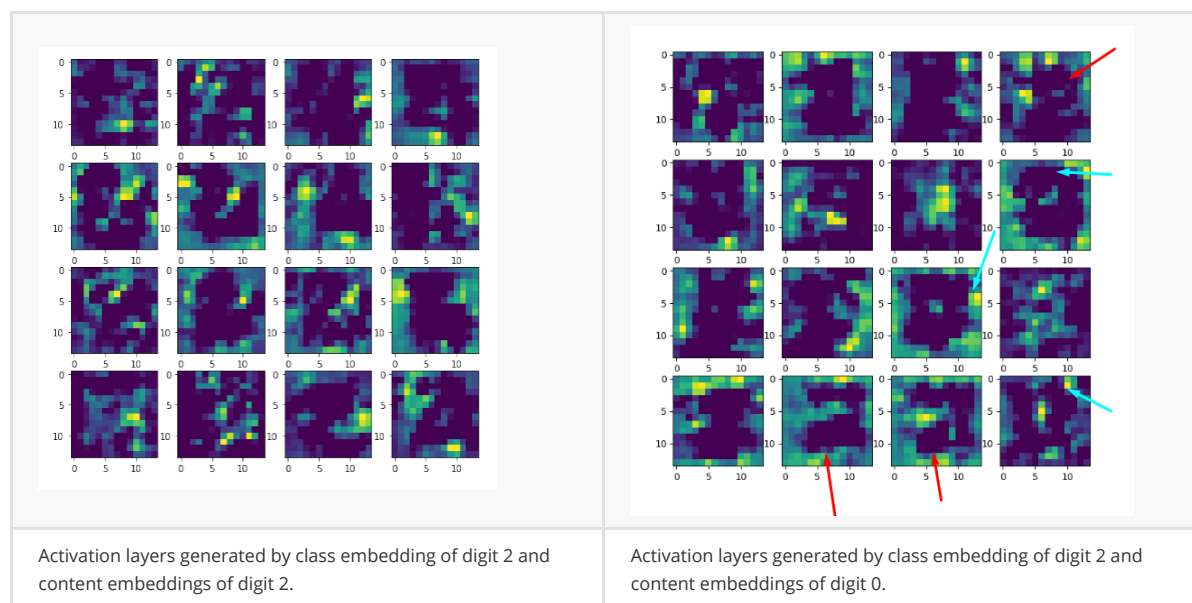
Low σ values

The impact of low noise $\sigma = 0.0001$ results in the following table:

	Class 6 , Content 0	Class 6 , Content 6	Class 6 , content 1
			
	Content embedding completely leaked into class embedding and changed the output with completely without any regard to the class. Fig 10	Reconstruction remain great when we're using the same class	Content embedding partly leaked into the class embedding and distorted the results extremely. Fig12

Therefore since the only images that are affected are the first and third figures, i.e., mismatching digits in their content and class embeddings, we can infer that low σ causes extreme leaking between the content and class embeddings, in some cases like Fig 10 we can even see almost a full disregard to the class embedding influence on the generator.

Further supporting this claim can be done by analyzing the activation layers:



Matching content and class embeddings (digit wise) included a lot of activation layers with a high resemblance toward the digit 2, but when they originate from different classes, we can **clearly** see the leak in the activation layers, the red arrows point toward activation layers that highly resemble the digit 2, while the blue arrow activation layer that somewhat resembles 0, the rest of the layers are kind of a combination between 2 of them without any definite shape. The activation layers indicate what's the building blocks upon the model generates the final image and the model attempts to use the digit 0 and the digit 2 **three times each**, and if we take into account the fact that his main goal is generate the digit 2 while completely disregarding the class of the content vector, then we gain insights about a severe leak that's prevalent in low-noise setting, and that claim is undisputable thanks to the detailed analysis from the activation layers.

Exploring noise levels in between:

The next step here assuming I had enough time or if we weren't supplied already with noise would be to examine different noises in between to find the sweet spot for the tradeoff we described above, since I believe Omri is much more capable than me in this regard I'll stay with the 0.3 noise he supplied us.

Classifying the classes by content embeddings to quantify the effect

The content vectors will be greatly influenced by the class of their original image right now since as we saw in the table they completely leaked. Therefore they'll now hold both class and content information in their embeddings. Clustering over those embeddings would result in clusters that contain the same number and the variety in those classes would be by the content.

Conclusion:

Therefore we found the hyper-parameters which approximately fit the current model. I'll make a wrong assumption that they also fit for the weight-decay variation due to report length and time limit.

Preview for the following part:

The following part describes the training and results of the following models :

- No weight decay GLO
- Varying weight decay GLO

Each part will include analyze the model and compare the aspects below:

- Training analysis
- Image reconstruction evaluation
- Class embedding analysis
- Disentanglement evaluation
- Activation layers analysis

GLO model analysis

The current variation is defined by the following loss:

$$L = L_1 + L_2 = \frac{1}{2} \|x_i - \hat{x}_i\|_{L_1} + \frac{1}{2} \|x_i - \hat{x}_i\|_{L_2} + 0 \cdot \|c_i\| \quad (4)$$

The loss was already explored above .

Training

Therefore during training he managed to learn (or overfit?) the pictures almost perfectly and how to generate them. The following graph from tensor board describes the loss convergence. The value is 0.008 and the images were generated from the modeled trained here.

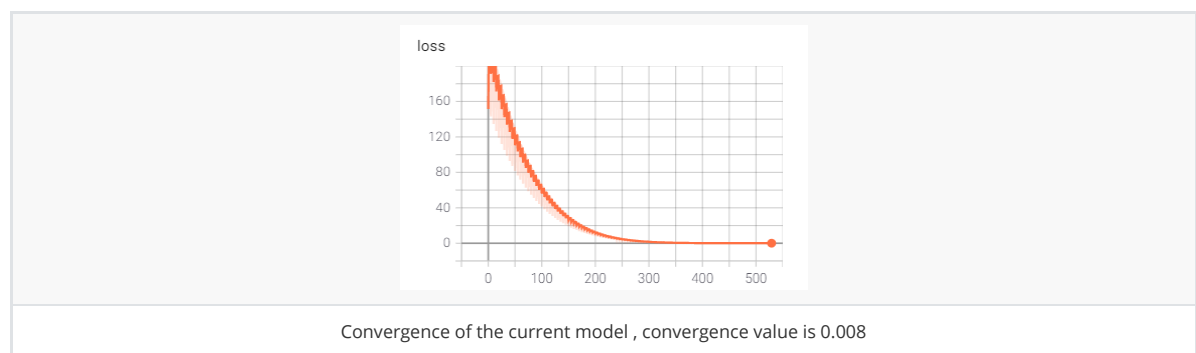





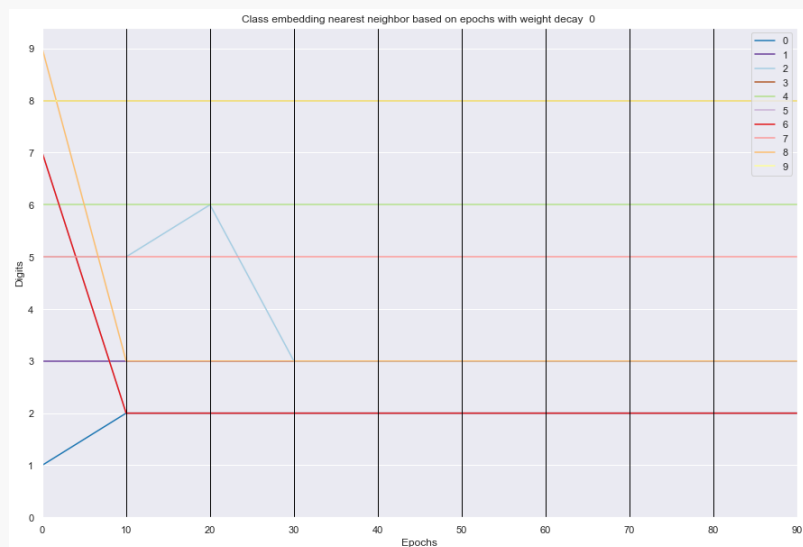
Image reconstruction

Taking images from tensor board side by side during training for reconstruction , weight Decay=0 was used here (Reminder : simplest setting as describe in the metric sections.) :

		
Managed to preserve the angle	Manage to preserve the missing chunk at the bottom	Managed to preserve the missing chunk at the top.

Moving on to intuition to help us understand classes embeddings make sense at all? Otherwise something is inherently wrong with our model.

Class embedding analysis



Nearest class embedding convergence based on epochs , we can see a little bit of uncertainty at the start from 2 , jumping from 5 to 6 and then staying at 3. There isn't some noticeable pattern to point out.

Closest class embeddings neighbors in in convergence :

Digit	0	1	2	3	4	5	6	7	8	9
Nearest digit	2	3	3	8	6	2	2	5	3	8
Ideal nearest digit	3,8,9	7,4	5,3,8	8 or 0	1,7	2,3	9,8,3	4 or 1	3 or 0	3,8



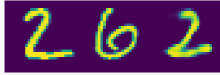


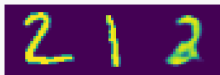

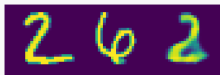
The ideal digits explanation is found in the metrics section , therefor based on those results we can approximate an accuracy metric , notice that 2, 3, 5, 8, 9 managed to match their ideal class , i.e. , **our accuracy is 50% for ideal matching between class embeddings** , taking into account the fact that random guessing would have been 10% accuracy on average (10 classes), then this results is actually very good considering our model!

Activation map progress: Omitted

Omitted since the report is already becoming extremely long and there weren't any major differences between the model activation layers progress , perhaps only the convergence took longer in the no weight-decay variation , or possibly there were but they weren't clear enough so I couldn't notice the patterns.

GLO capabilities regarding disentanglement no weight decay

Each box contains an image which is a concatenation of 3 images, the first image is the class (the current style of the class picture doesn't affect us at all, the game is here only for illustration of the current class.). The second image is the content , and the third image is the generator output.

		
A rounded 3 content resulted in a more rounded 2. Fig 1	A rounded 2 content resulted in a more rounded 2 . Fig 2	A tilted 6 resulted in a more rotated 2. Fig 3
		
Thick&round 4 resulted in thick&round 2 , great results. Fig 4		Rotated and semi-thick 9 resulted in a bit distorted 2. Fig 5
		
Same thickness 2 and 1 resulted in a failure. Fig6	A bigger "ball" 9 resulted in a complete failure. Fig 7	A heavily rotated 6 resulted in a heavily rotated upper part of 2. Fig8

The results are good if we take into account the model capacity. While failures do exist (Figs : 7 , 6) , success also isn't rare , e.g. Figs : 1 , 3 , 4 , 5, 8 give an example of extracting the content from images of different numbers and managing to do it on a good level. If we simplify the task a bit and extract the content from another image within the same class the results are approaching amazing , as can be seen in Fig 2. The widely better results in the last setting imply that a leak exists between the content and class vector , i.e. , the disentanglement isn't perfect.

GLO model analysis with weight-decay

Loss function:

$$L = L_1 + L_2 + WeightDecay = \frac{1}{2} \|x_1 - \hat{x}_1\|_{L_1} + \frac{1}{2} \|x_1 - \hat{x}_1\|_{L_2} + \lambda \cdot \|c_1\| \quad (5)$$

adding the extra constraints force an adjustment to find the right weights to solve the problem under a constricted dimension, i.e. , no weight decay uphold the following equation for each weight(W is the weights matrix of the GLO) :

$$\forall w \in W \quad w \in (-\infty, \infty) \quad (6)$$

In other words in the former variation we don't care about weights values , we just want to minimize the loss. The latter variation weights could still theoretically lay in the same range , but in practice weights in the latter variation lay in a small range then the former variation, i.e. we limit the range in which the weights lay which result in a longer convergence.

The advantage of weight decay lay in restricting our model overfit capabilities by limiting the range of the weights , and hopefully improve the results of the former variation.

Training

The following graphs from tensor board describes the converges for the $\lambda = 0.1$, which yielded the best result.

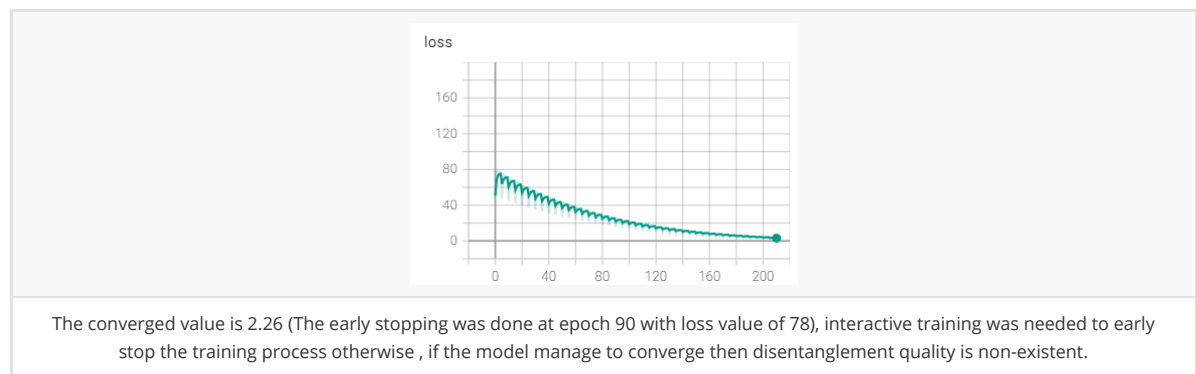




Image reconstruction

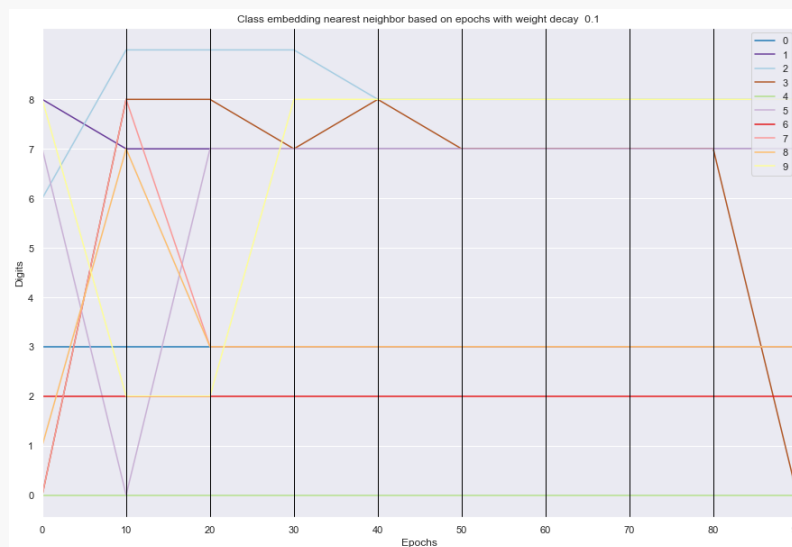
weight Decay=0.1 was used here and resulted in the following generated images when class and content originate from the same number , we should note that it is compared to the image the content was originated from :

Reconstructed Images		

Both models manage to reconstrue the images pretty effortlessly in the simplest setting , the left image is not perfect as a smudge can be seen in the left middle part.

Class embedding analyze

The following graphs describes the class embedding nearest neighbor based on epochs ,



the transition between class embedding isn't stable at the start but converges to a stable pattern in epoch 50 for most of the digits beside 3 which switches from 7 to 0 at epoch 90, like he should since 7 and 3 class embedding shouldn't contain a similar information regarding class embedding based on their shape.

The final convergence value can be seen here :

Digit	0	1	2	3	4	5	6	7	8	9
Nearest digit	3	7	8	0	0	7	2	3	3	8
Ideal nearest digits	3,8,9	7,4	5,3,8	8 or 0	1,7	2,3	9,8,3	4,1	3 or 0	3,8

The ideal digits explanation is found in the metrics section , therefor based on those results we can approximate an accuracy metric , 0 , 1 , 2 , 3 , 8 , 9 were able to match their ideal nearest digits , **therefor the accuracy of the current variation is 60%** , which is quite good and imply that the model managed to learn the needed geometric structures of the class embeddings .

Activation map progress: Omitted

Omitted since the report is already becoming extremely long and there weren't any major differences between the model activation layers progress , perhaps only the convergence took longer in the no weight-decay variation , or possibly there were but they weren't clear enough so I couldn't notice the patterns.

GLO capabilities regarding disentanglement with weight decay

$\lambda = 1$ Interactive training (Early stopping)

Lacking results , made the 2 thicker by costing a lot of form and density.	Ok results , made the class rounder but also made her a bit thicker in some prats.	Bad results , lost all the form.

$\lambda = 0.5$ Long training (Overfit)

Hard overfit , blurry and can't distinguish between any style.	Hard overfit , blurry and can't distinguish between any style.

$\lambda = 0.1$ Interactive training (Early stopping)

While not as defined as I'd liked it to be , it is a nice result.	the generated is missing a bit of thickness but he did learn to make it this.	A rotated 2 upper part , a very nice result.

The results are much better at a lower weight decay and in some aspects outperform the no weight decay variant , like thinness .

Therefore $\lambda = 0.1$ with interactive learning is yielded the best results based on my experiment.

Comparing the models

Generation comparison

Reconstructed Images		
	GLO	GLO weight-decay

Both images manage to reconstruct a given image from the training set almost perfectly with a slight edge given to the no weight-decay variation ,

Training Comparison

Convergence time

Convergence time is considerably longer in the latter variation which is explained by the constricted dimension our weights live in , therefore searching and finding the best is a harder then searching in an infinite dimension.

Loss

The different loss function hopes to avoid overfitting in the latter variation , the values are higher and good disentanglement is generated only at a sizeable loss (78) which isn't the convergence loss (2.26) , yet the former variation would yield good result at convergence.

Interactive training





The weight decay model would yield good results **only** if we early stopped the training , the loss at that point would be 78.00~, resuming the training would overfit class embeddings and ignore content completely and eventually the converged value would be 2.26.

The difference's between the models regarding training process ,loss values and convergence are enormous and indicate the need to closely monitor the given loss needs and properties and adjust them based on his particular quirks.

Class embedding comparison

The former managed to reach 50% accuracy while the latter 60% , which imply that the weight decay does manage to differ between the classes geometric shape better then the no-weight decay variation , which imply that maybe under the optimal λ the latter variation could far out-match the former.

Disentanglement comparison:

	$\lambda == 0.1$	$\lambda == 0$
		
	Generated digit is rounded and thicker but undefined.	Generated image is thicker and rounder as desired.
		
	Generated digit is bit rotated but lost her intensity levels.	Generated image is clearly rotated.

Disentanglement results clearly favor the former variation based on my experiments by 2 different criterions , rotation and thickness.

Models comparison conclusion

The former model is easier to train and reached better results based on the limited hyper-parameter search I have done , still declaring him better would be a mistake. I believe that theoretically there exists a λ which lives in $[0,0.1]$, combining him with early stopping at the optimal point would outperform the former variation. Unfortunately since we need to early stop on top of hyper-parameter searching reaching optimal results would not contribute a lot to the learning process and mainly result in a sizeable chunk of time wasted, therefore I'll

leave the results as is.

Summary

We attempted to disentangle the content and class from given digit. First we begin by finding the right metrics on which to evaluate our performance, then hyper-parameter fit was done, afterwards the training process began, we saw that monitoring the results is necessary in order to gain insights about the behavior of the model when he engages the current data and loss function, then we saw that the line between overfit and good is thin, and therefore we must early stop at the right time to gain optimal performance.. Based on the insights we adapted the learning process to yield optimal results, we checked if the class embeddings and the disentanglement make sense and reasonable results were achieved.

Afterwards we compared the learned models variations (weight-decay vs no weight-decay) based on criteria that were described at the start, and tried to understand their differences, we saw that finding the right weight-decay hyper parameter is tedious and therefore his performance wasn't as we expected, and the no weight decay variation performed better.

Thanks!