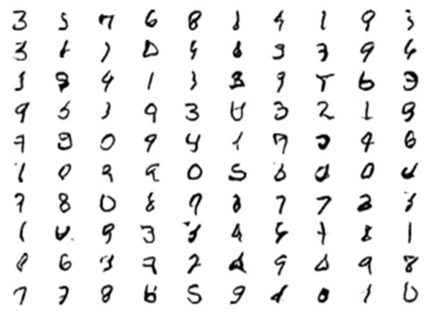
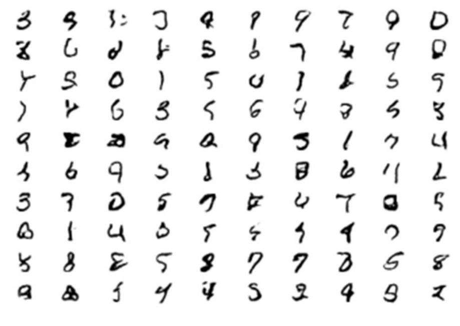
Task 5

1) I am not too entirely sure what to talk about here since no direct questions were asked. I updated the activation method in define_generator() to tanh and had adjusted the scaling in load_real_samples to range from [-1,1]. I then used this as a base for comparison in question 2. The model started out producing very bad images and as it progressed continued to improve and by the e100 was producing very passable images and still showing signs of improving.

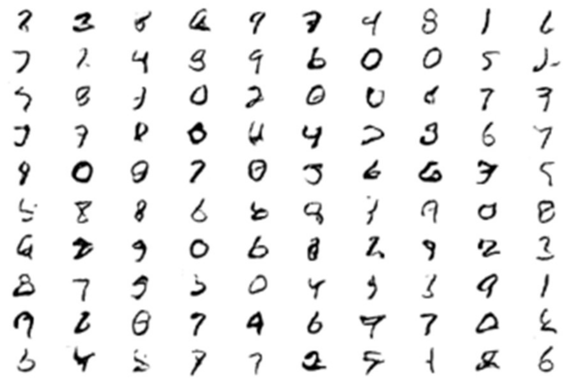
2) For this question I changed the latent_dim from 100 to 70 to reduce the size of the latent space. Now first I will talk about the time to run because it is remarkably almost the same. For question 1 it took 44 minutes and 37 seconds while for question 2 it took 44 minutes and 31 seconds. It is questionable in my mind if this small difference is due to the change in the code or due to the random nature of the code itself. I will assume it is due more to randomness than the change in the code since its console output accuracy had taken a much lower score which should of resulted in more training. Comparing the quality of the two runs we can see that this reduction in size of the latent space resulted in more blotchy results. The results also appear less human. The first question had about the same number of artifacts generated (random symbols with no meaning instead of legible numbers), however the second question generated numbers that were jagged and inconsistent even in line thickness. Comparing the console outputs for these runs it appears that the first question resulted in overly positive results. Although it was generating very passable digits it was not in 80-99% that it was finding. This will result in the model not making much training progress due to receiving bad results since the discriminator is confused. Now in the second run the results appear to be overly negative. There are a lot of vary passable numbers however the results for real were showing wild swings from 15% to 80% between runs. This would result in the generator does not know what direction to train in.
Below was my question 1 e100 run on the left and question 2 e100 run on the right.



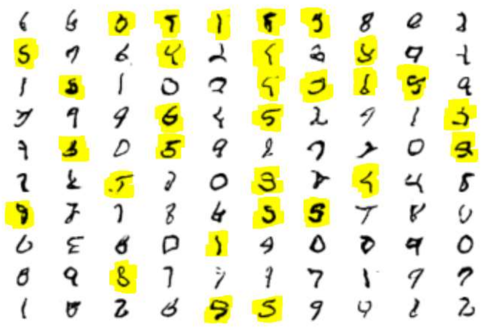


3) For this I imported the BinaryCrossentropy from keras.losses to replace the loss in the discriminator with one that had a label_smoothing of 0.9. Doing this resulted in the most drastic change in run time. From question 2s 44 minutes and 31 seconds down to 44 minutes flat. I believe this is due to the loss being 10% smaller than it previously was at

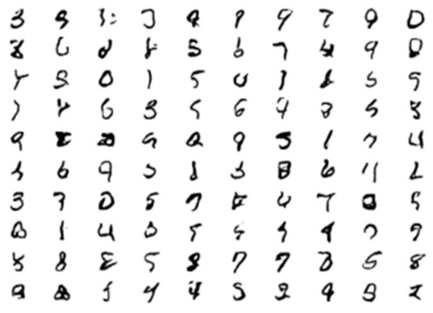
1.0 making the numbers small and more even across the board for easier calculations. Now comparing the quality of these runs resulted in an interesting find. In question 2 the results ended up staying consistent with blotchy numbers and artifacts. Meanwhile in question 3 the results came out less blotchy with equal to or more artifacts. However, it is apparent at e70 (as shown below) the generator was constructing very human like digits.

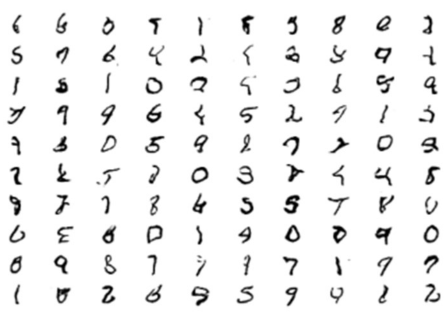


However, after this it started to move away from human like figures and into more squiggly lines that were in vague shapes of digits. I believe that this shows when the point of convergence was crossed in this run. In fact, it is starting to collapse towards the digit 5. I show this by highlighting question 3 e100 where it generates 5s (this includes artifacts that look vaguely like 5). The other digits seen in this are 7 and 9 with almost no attempts at any of the other digits. These 3 digits make up for at least 90% of this run's generation meaning it is collapsing.



In finality of this comparison I observed that question 3 was able to generate better results faster, however it also over trained past the point of convergence and started collapsing towards 3 different digits with a strong emphasis on the digit 5. While, question 2 was still learning and producing passable digits.
Below was my question 2 e100 run on the left and question 3 e100 run on the right.





4) This is going to be a common issue when you have 2 algorithms trying to achieve different goals. The discriminator is trying to weed out all fake outputs and the generator is trying to generate only outputs the discriminator believes to be real. As such when the generator finds an output that the discriminator always classifies as real it will start leaning towards this output till it will only output this. There are a lot of different approaches on how to prevent this. Personally, I believe that having the generator/discriminator save the result for review later to see if this result is usable for its given use case. For this example if the digit is adequate enough. However, this would also require extra head room of letting the generator know that this generated result was a specific digit and it still needs to find the rest. But by saving this result you can use it to have the discriminator to ignore this result and tell the generator to stop trying to produce it. I believe this approach to be best for this specific cause since if the generator stops generating that result it is still likely it will generate something similar unless the discriminator flags these results. This is also the reason why I would have these collapses saved. If the discriminator is constantly flagging against them then it is possible it will flag good results that could be used. Another approach is to limit how many of one digit the discriminator is willing to pass. For this given case something like it will any single digit 10% of the total number of generations. Let's just say it looks at 100 generations then it will only pass a specific digit 10 times. Now these might be a very naïve approach to solving this issue because it is only solving the symptoms and not the root cause. Looking into the issue online shows that this issue is still an area of ongoing research with no definitive answer yet.

5) Now the failure of convergence is generally because the goal of the discriminator is to "beat" the generator and the goal of the generator is to "beat" the discriminator. In order for it to converge both the generator and discriminator need to both be in an equilibrium. Specifically, for GANs the discriminator needs to still give meaningful feedback while the generator can sufficiently fool the discriminator. One approach to doing this is by using different formulas for cost. This approach is shown by the various GANs such as LSGAN, WGAN, BEGAN, DCGAN, and many more. While this approach has its merits no formula to my knowledge has been found that completely solves this issue. Instead they are all more efficient in their respective areas of use cases. Another approach is by changing the weights of the experiment. I personally believe this is a better approach. So,

an example of changing the weights is making the discriminator favor a certain type of number. Looking at the digits 0-9 you can break them down into 2 different categories. Those with straight line (1,2,4,5,7) and those with curves (3,6,8,9,0). We can have the discriminator lean towards one of these 2 categories by weighing them differently. Using this approach, you can train 2 or more different discriminators to train the same generator. This will try to keep the generator from not receiving rubbish feedback as well as avoiding a pitfall in the generator only wanting to produce a single digit. Now with all of this said it still will not solve the issue of convergence because it is very easy to over train your GAN. Which is why you are supposed to periodically save your models to be reviewed later to find the best one that matches your criteria. You can change how fast or how slow the model converges but that point of convergence will most likely happen for only a very short amount of time before the model moves on. There have been mathematical models to show a representation of when it is possible for the GAN to converge however due to nature of GANs even these representations are not accurate and cannot be used practically since each run will be different resulting in these representations being only trends of the specific GANs. Similar to question 4 this issue is still an active area of researching leading to a lot being open to interpretation. This is just my potential solution to better GAN convergence.