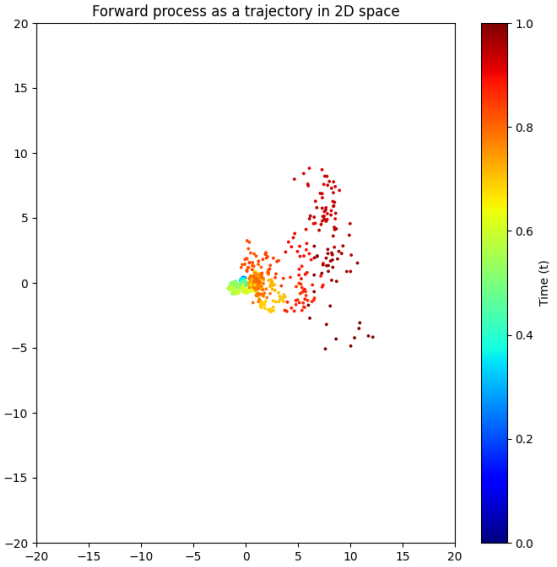
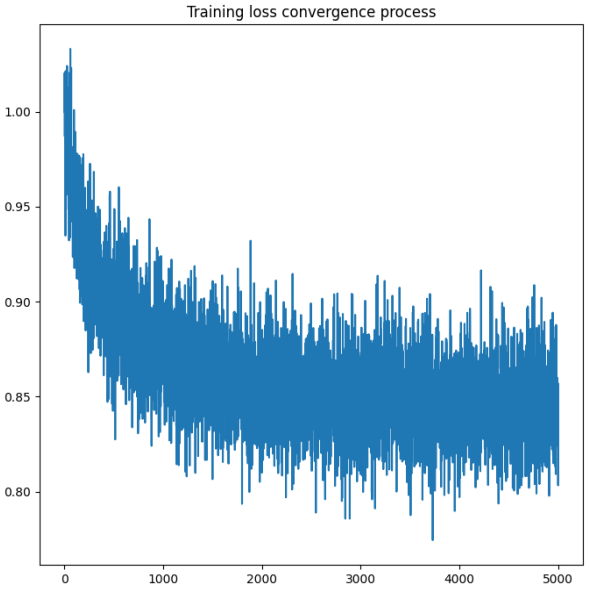
Ex1 - AML
Omer Siton - 316123819

Section 2.2 - Unconditional:

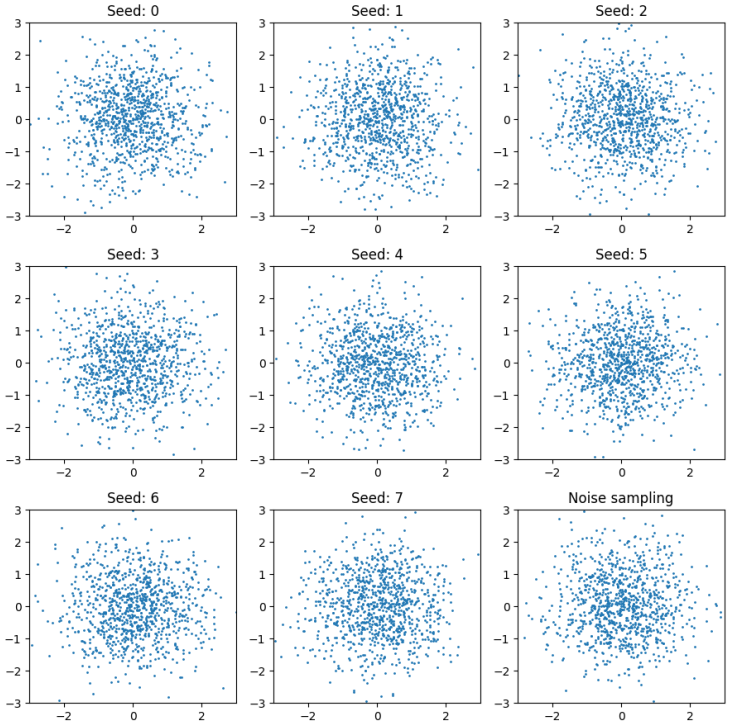
1. For a point of your choice, please present the forward process of it as a trajectory in a 2D space. Start from a point outside the square. Color the points according to their time t.



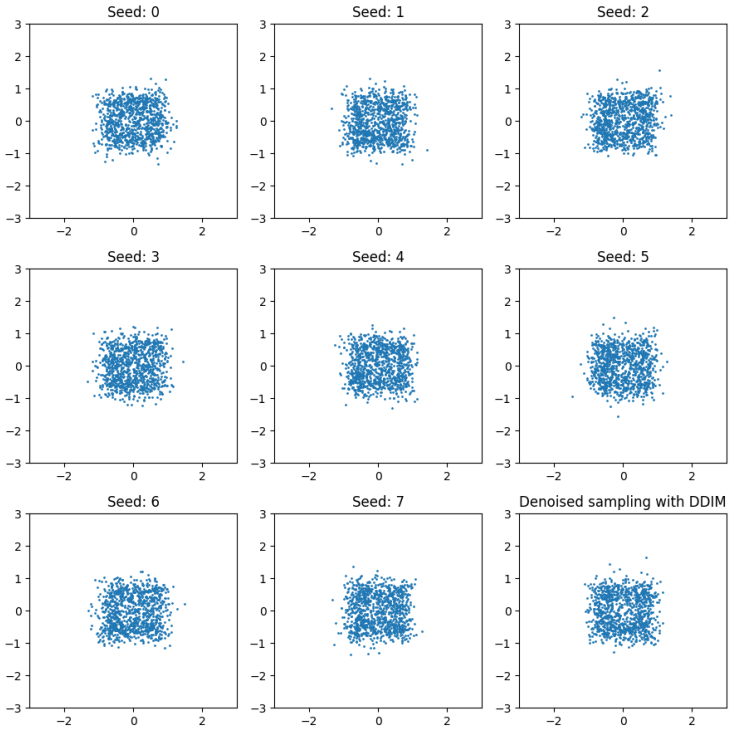
2. Present the loss function over the training batches of the denoiser.



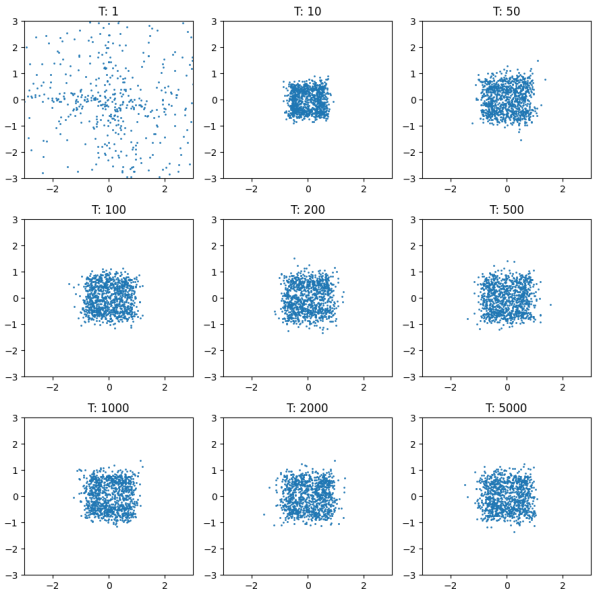
3. Present a figure with 9 (3x3table) different samplings of 1000 points, using 9 different seeds.
   This is the **initial noise sampled from the normal distribution**:



   This is after performing the **reverse sampling** as described in the exercise:

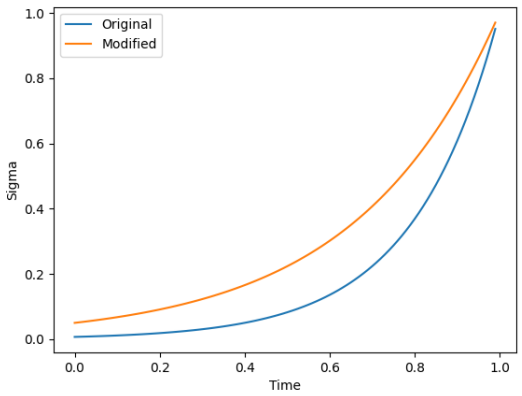


4. Show sampling results for different numbers of sampling steps, T. How does the sampling length affect the results?



We can see that at first for a small number of iterations T, all points move to the center of distribution mass and do not really mimic the real distribution correctly. As T number goes up, the samples move to a more logical and identical place to the real distribution as in a model training process (chain saw shape) in and out of the distribution. We can see that the higher the number T the more identical the distribution to the original one.

5. Slightly modify the given sampler, or create a sampling schedule of your own. Plot the σ coefficients over time (sample them using some dt step size), in the original sampler, and your modified version. Describe your choice for ablation/sampler. In your viewpoint, what are the rules of thumb you find for an effective sampler?



I choose to change the sampler to exp(3*(t-1)) which leads to a less rapid increase in noise from the original sampler.
Some rules of thumbs I came up with during experiments:

- Match the noise level to the data: If the data has a lot of noise, you might need a more aggressive noise schedule. If the data is very clean, a more conservative schedule might work better.
- Start with a simple schedule: A simple linear or exponential schedule can be a good starting point. You can then adjust it based on your results.
- Experiment: Try out different schedules and see what works best. You might need to tune the schedule just like you would tune hyperparameters.

6. Insert the same input noise into the reverse sampling process 10 times (do not seed the model), and plot the outputs. Are the outputs the same?

   Yes, the outputs are the same, this does make sense based on the fact we used DDIM sampling process, which is deterministic.
   We discussed in class about a stochastic alternative to the reverse sampling process called DDPM.

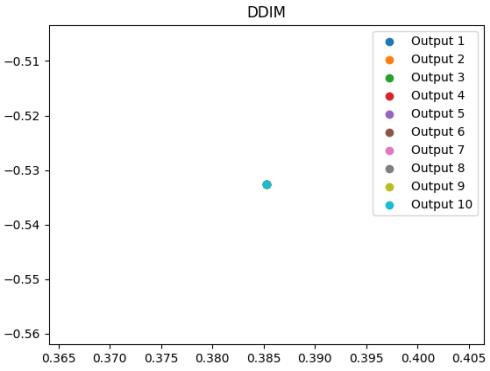
   - Given the probability flow ODE, reverse the process by sign inversion

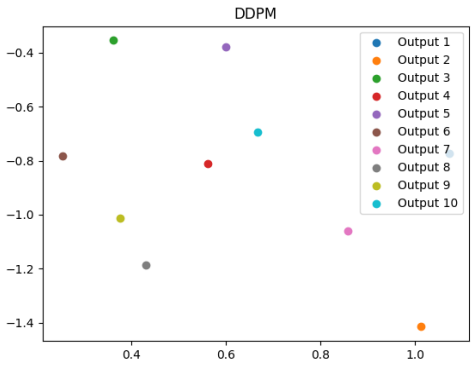
   $$dx = -[f(x,t) - \tfrac{1}{2}g^2(t)\nabla_x \log p(x,t)]dt$$

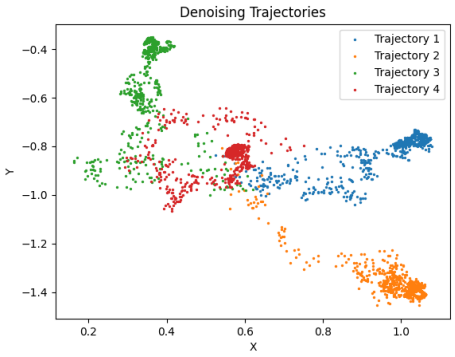
   - Can also construct a stochastic process with the same marginals

   $$dx = -[f(x,t) - \tfrac{1}{2}(1+\lambda^2)g^2(t)\nabla_x \log p(x,t)]dt + \lambda g(t)dW$$

   Using the modified/original (depending on your answer) sampler, plot the denoising trajectories of 4 of the points on the same figure.

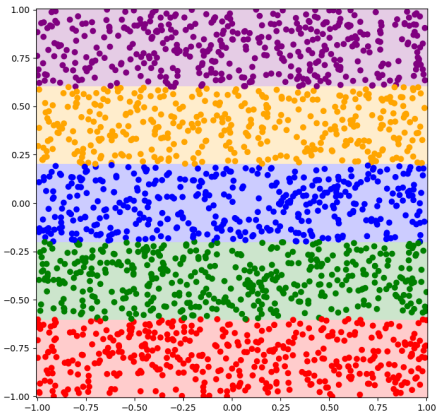






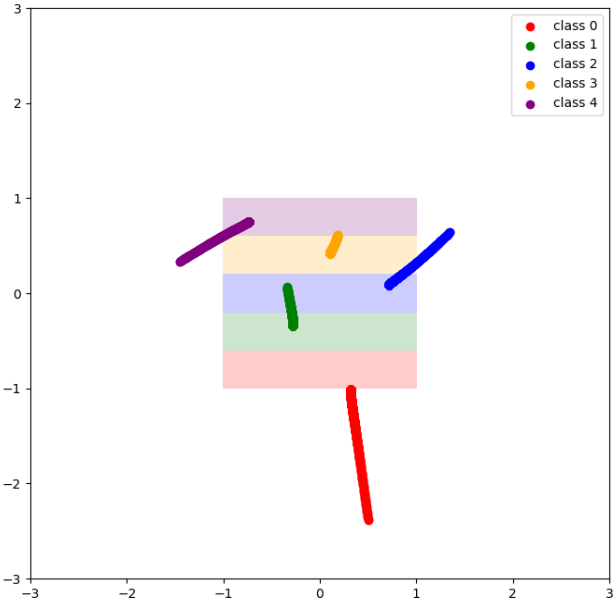
Section 2.2 - Conditional:

1. Plot your input coloring the points by their classes.

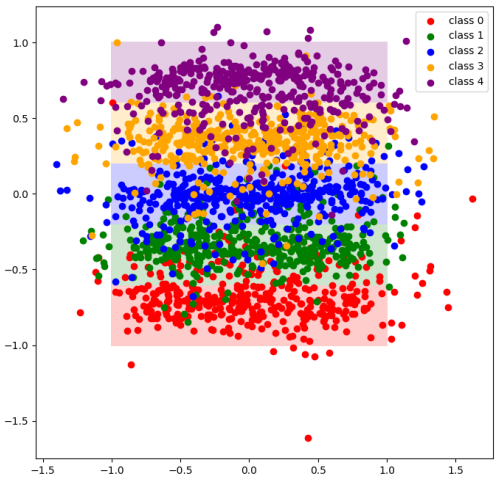


2. How did you insert the conditioning? Which equations have to be modified?
   First, we had to prepare our data in a way that we have the 2d points and the classes as targets. For that task I created torch.Dataset and used a data loader with batch size equals num_points. Second, I changed the model architecture to include an embedding layer for the classes as suggested in the exercise.
   Then, I changed every occurrence of D(xt, t) to D(xt, t, c).

3. Sample 1 point from each class. Plot the trajectory of the points, coloring each trajectory with its class's color. Validate the points reach their class region.

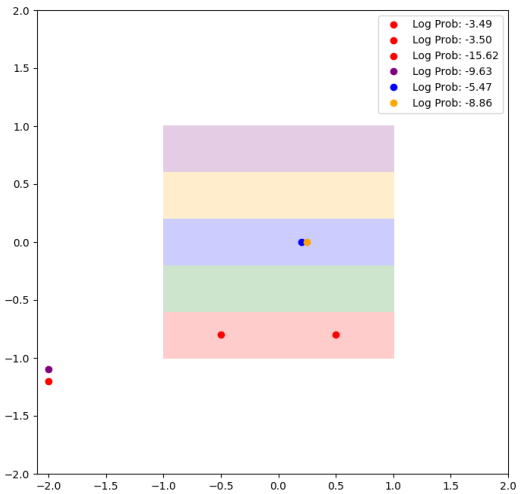


4. Plot a sampling of at least 1000 points from your trained conditional model. Plot the sampled points coloring them by their classes.



5. Analyze the sampling results. Are certain classes more difficult than others? Do you see the same spatial distribution as the input? Do you have other interesting findings?
   The sampling results look “good” overall, some finding I inferred from the visualization:
   1. The edge classes are distributed a bit worse than the others, so it could be that it’s harder for the model to sample tham correctly.
   2. In the y axis the samples were closer to the square than in the x axis.
   3. All distributions looks more centerd (both axises) in their classes than the original data.

6. Estimate the probability of at least 5 points. Plot the points on top of the data. Choose the set of points so some of them are within the input distribution and others are out of it. Also, choose a pair of points with the same location but different classes, one that matches the input distribution and one that isn’t. Describe the results, explaining what happens when the location/class of a point does not match the initialdistribution.

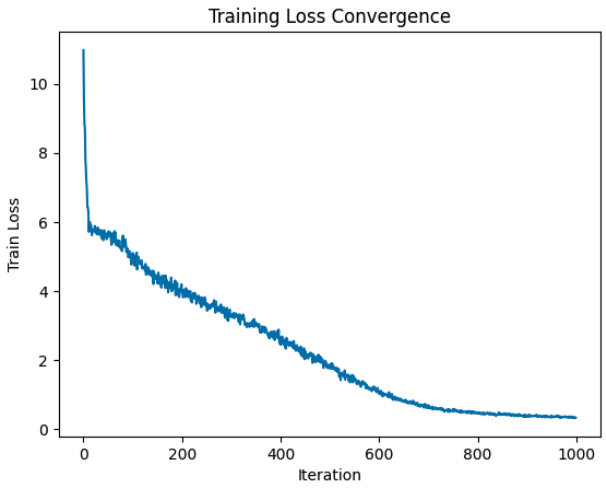


*The points coloring is based on the class given to estimate the probability. As we can see in this figure, we get higher log probabilities (bounds) for points inside the square with the correct class, as expected. For identical points inside the square with different class we get higher prob for the point with the correct class, as expected. For identical points outside the square with different class we get higher prob for the point with the wrong class, weird.

7. Bonus(10pts): Make one of your trained models (conditional or unconditional), output the point(4,5.5) (which is far away from the square), as its sampled output. How did you do it? Describe your attempts (including unsuccessful ones). You may select the noise that is added in each step of you wish. Saying that trivial solutions will receive less points than more advanced ones. Plot the trajectory of the sampled point over time.
   Naive sol: I tried to make a function using the unconditional model and a target point that is similar to the sampling process we implemented which every iteration we add noise in the direction of the target point. This didn't work well at all, probably because the target point is so far away from the actual distribution the model trained on, and respectively the score related to it.
   Another idea is to train the conditional model with another class specific for this far point and include in our training process data that contains many points like the target point with its unique class.

Section 3.2 - AR Model

1. Train the model as standard GPT-2. Present the loss term over training time.



2. Perform an inversion process (Sec.3.2.2) to the following sentence: "I am a little squirrel holding a walnut". Explain any special choice in your experiments. We won't deduct points if you were not able to reach the sentence, but in this case we want you to try and explain why the model was not able to do so.
   Iteration 1/500, Loss: 10.490324974060059
   The current generated sentence is: so have I I!'!'.' there
   Iteration 101/500, Loss: 1.5889984369277954
   The current generated sentence is: I am a little squirrel holding a wal very
   Iteration 201/500, Loss: 0.9442957639694214
   The current generated sentence is: I am a little squirrel holding a walnut
   Iteration 301/500, Loss: 0.3783971071243286
   The current generated sentence is: I am a little squirrel holding a walnut
   Iteration 401/500, Loss: 0.1734674721956253
   The current generated sentence is: I am a little squirrel holding a walnut
   The final generated sentence is: I am a little squirrel holding a walnut

In this experiment, I created an inp_vec with the dimension of (num_tokens_of_target_sentence, n_embd) which is equal to the first layer output dimension. this is a must because we want to optimize this inp_vec we created and the embedding layer can get as an input only int/long which we can't optimize.
In this experiment it's seems like this task was easy for our model and after 200 optimization iterations it could generate the target sentence.

3. Generate a sentence with at least 12 words (the 11th and 12th words must be generated by the model). In the12th word prediction (find the tokens responsible for it), extract the attention scores SoftMax((Q∗K)/sqrt(dim)) for the last transformer block. Average the scores over the different attention heads. Paint the tokens by their relation to the 11th word. Explain why did the model choose to predict the11th word in its previous prediction step,based on this analysis. If you find painting a sentence too complex, you may present the scores below each word instead (which can be done with paint as well). What is the sum of all attention scores?

   number of tokens generated: 12 / 12
   Generated Response:
   Alice, and
   looked at poor Alice, who was reading

   Last transformer block
   Score for each token predicting the 12th token - ' reading':
   'Alice': 0.011750717647373676
   ',': 0.008491908200085163
   ' and': 0.08203516155481339
   '\n': 0.13695891201496124
   'look': 0.09191030263900757
   'ed': 0.0623285174369812
   ' at': 0.1762125939130783
   ' poor': 0.04064375162124634
   ' Alice': 0.03135298565030098
   ',': 0.14201991260051727
   ' who': 0.07619347423315048
   ' was': 0.14010180532932281
   Sum of attention scores: 1.0000001192092896
   In the final transformer block, we see that the token ' at' has the highest attention score (0.176) when predicting the 12th token ' reading'. This suggests that the context of direction or location might have influenced the prediction of an action or state - 'reading'. The high attention to the tokens 'look' and 'ed' (forming the past tense verb 'looked') also point towards the same, as it sets up the context of someone observing Alice, which could be logically followed by describing what Alice was doing, in this case, 'reading'.
   (*) in the code, the words are colored

4. Repeat the last question, this time using the attention scores from the first transformer block. What difference do you see in the results?
   First transformer block
   Score for each token predicting the 12th token - ' reading':
   'Alice': 0.02561720460653305
   ',': 0.031182043254375458
   ' and': 0.01320886705070734
   '\n': 0.0033750555012375116
   'look': 0.07492997497320175
   'ed': 0.05171241983771324
   ' at': 0.02511376142501831
   ' poor': 0.13130314648151398
   ' Alice': 0.03737381845712662
   ',': 0.10674639791250229
   ' who': 0.2644014060497284
   ' was': 0.23503591120243073
   Sum of attention scores: 1.0

   In the first transformer block, the model paid the most attention to ' who' (0.264) and ' was' (0.235) when predicting 'reading'. This is logical because 'who was' often precedes a verb describing the action of the person, in this case, Alice was 'reading'. It is interesting to note that the first block focuses on 'who was' which sets up a description, while the final block pays more attention to 'looked at', suggesting the evolving context being built up in the intermediate layers of the transformer. The sum of the attention scores in **both** the transformer blocks is 1, which is expected as the attention mechanism uses softmax to distribute attention among the tokens and softmax ensures that the sum of all values (here, attention scores) equals 1.
   (*) in the code, the words are colored

5. Sample any sentence from the model. Compute the log-probability score of the sentence by multiplying the model's probability prediction for each word in the sentence. Don't forget to work in log-scale,working with the actual probabilities will lead to unstable and extremely low values.

   number of tokens generated: 10 / 10
   Generated Response:
   'Omer,' she said, as politely as she
   could'
   Log Probability of this sentence: -4.114333152770996

   GPT:
   I used chat GPT a lot in that exercise, first, it helped me work with PyTorch. Second and more importantly it helped me with all the figuers in this exercise, usually I don't remember the exact syntax for generating the specific graph I need and this tool saved me a lot of time scrubbing the web for examples.

Submission:
I want to clarify that I spent over **1 full day** of work on transferring the code I wrote in Google colab that worked great to Python for submission. Moreover, I don't know how our code supposes to train our AR model without the Alice.txt fil. Please don't deduct points if the submission format is not how you expected I will change it accordingly to what you want. what I've submitted now is what I understood from what you wrote in the exercise document. Because I spent so much time and enjoyed the learning process I don't want to lose points on submission issues.