Exercise 1 - Generative Modelling

67912 - Advanced Course in Machine Learning

Yuval kenneth – 208435834

2.2.2 – unconditional:

1.

I chose (0,0) as the starting point:

A picture containing text, screenshot, visualization

Description automatically generated

2.

A picture containing text, screenshot, plot, diagram

Description automatically generated

3.

A picture containing pattern

Description automatically generated

A picture containing text, screenshot, display, number

Description automatically generated

In this scatter plot, we can see that sampling (with the same seed) using increasing sampling steps, converges as T increases.

5.

A picture containing text

Description automatically generated

In the presented scatter plots, I wanted to see how sampling works using the same noise scheduler, noisier ones, and a less noisy one. F(1) = 1 for all used schedulers. It is visible that using the same noise schedulers used in training, spreads the samples more uniformly, matching the training data. Also, it is visible that using a less noisy scheduler, the data is more dense in the center of the square, as opposed to the noisier schedulers, that “pushed” the samples to the edges of the square. In summary, in order to sample exactly like the training data, I would use the same noise scheduler, but if I wanted to control how my samples spread, I will choose a scheduler based on those findings.

6.

Using DDIM sampling resulted in the same output every time as expected. I implemented another sampler, with the only change is in eq.8 in the exercise. In each sampling step, I added a small amount of noise (sampled from normal distribution and multiplied by a small factor (<1).

I was thinking that the added noise will still result in the same “Area” (populated areas from the training set), but will add a little variance to the outputs from the same noise input.

A picture containing screenshot, diagram, text, plot

Description automatically generated

2.2.2 – conditional:

1.

A picture containing text, screenshot, colorfulness, pattern

Description automatically generated

I divided the square into 5 classes, such that each class width is 0.4 on the x-axis.

#added loss over batches for conditioned model

A picture containing text, screenshot, plot, diagram

Description automatically generated

2.

In order to insert the conditioning, I implemented a similar denoiser (NN), only this time I added the class of the input as another input, and added an embedding layer to the network. All other functions (such as sampling and estimation) were updated to match both denoisers.

3.

A picture containing text, screenshot, line, diagram

Description automatically generated

It is visible that the points reach their designated class. (class 0 and 4 reach the line between their neighboring classes)

4.

A picture containing text, screenshot, colorfulness

Description automatically generated

5.

Overall, it seems that the spatial distribution is very similar to the input distribution. However, it seems that classes 0 and 4 are slightly more difficult than the others. A possible explanation is that from normal distribution, it is less likely to sample an initial point in those areas, so maybe the denoiser is less likely to succeed in reversing the noise to those classes.

6.



From the first two points we can see that the same point, once assigned to the right class and the other to a different one, the probability of the former is much higher. The least likely point is far away from the square as expected. It seems as if points in the square that are assigned to their class have similar probabilities, as can be expected given that the data was uniformly sampled from the square and divided equally (space-wise) between the classes.

A picture containing text, screenshot, colorfulness

Description automatically generated

(there are two points at (-0.8,0.8)

3.2.4 – GPT-2:

1.

A picture containing plot, line, diagram, screenshot

Description automatically generated

2.

At first, I looked at the embedding layer in order to understand how to initialize the input vector (in terms of dimension, and to sample its values uniformly from -1,1). Then, I experimented with different context windows. I wanted to use enough such that the process will be more flexible in terms of possible words to attend to (for example, if the words 4th word on the desired output sentence is not so probable based on the first 3 words, we can give enough context to attend to instead). On the other hand, I didn’t want to use a large context window, to make the learning process easier. Finally I experimented with different step sizes and iterations. For multiple runs, I got sentences that were very similar to the desired sentence (for example, only one token was different). I figured that could happen because of the optimizing process, and the randomization of the input vector at the start of the process, So finally after a few runs, I got the desired output.

Inversion loss:

A picture containing text, screenshot, plot, diagram

Description automatically generatedA screenshot of a computer program

Description automatically generated with medium confidence

3.

A picture containing text, screenshot, colorfulness

Description automatically generated

4.

A picture containing text, screenshot, colorfulness, purple

Description automatically generated

5.

