Advanced Course in Machine Learning - Ex. 2

Amit Roth

July 4, 2024

1 Normalizing Flow

In this part we trained a Normalizing Flow model on the data supplied by the `create_unconditional_olympic_rings()` function.

The architecture of the model is detailed in the following files: `affine_coupling.py`, `permutation.py`, `normalizing_flow.py`.

1.1 Q1: Loss

We present the validation loss over the training epochs in 1 and log-determinant, $log(p_z(x))$ components in 2.

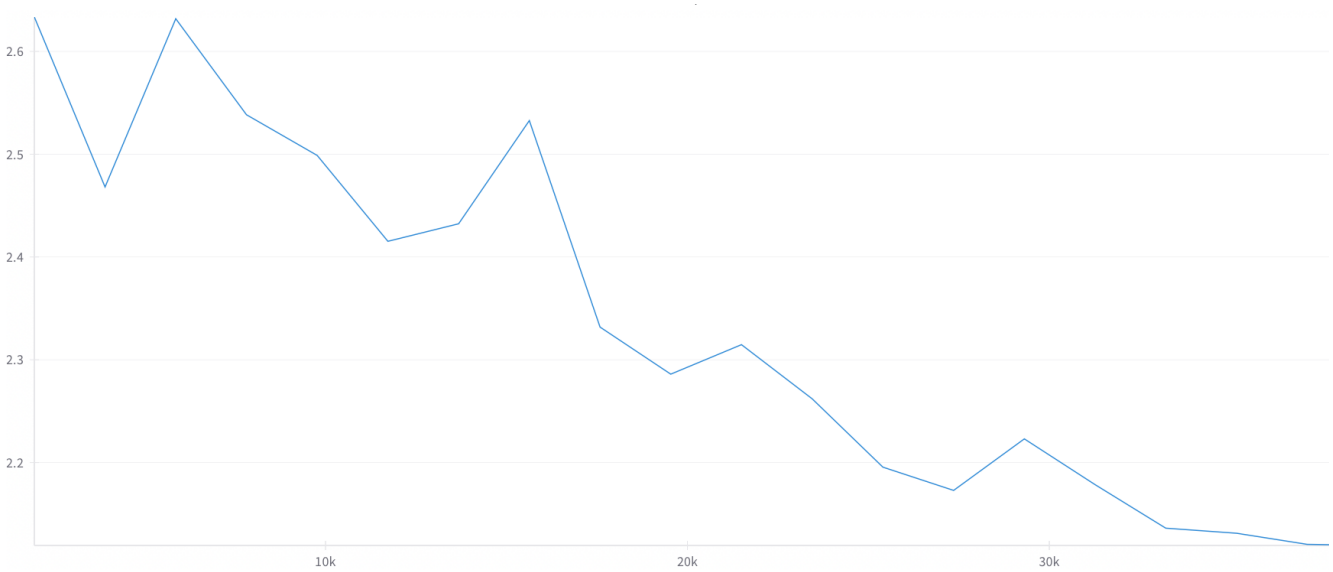


Figure 1: validation loss of normalizing flow model

.

1.2 Q2: Sampling

We present 3 figures with different samplings of 1000 points, using 3 different seeds at 3.

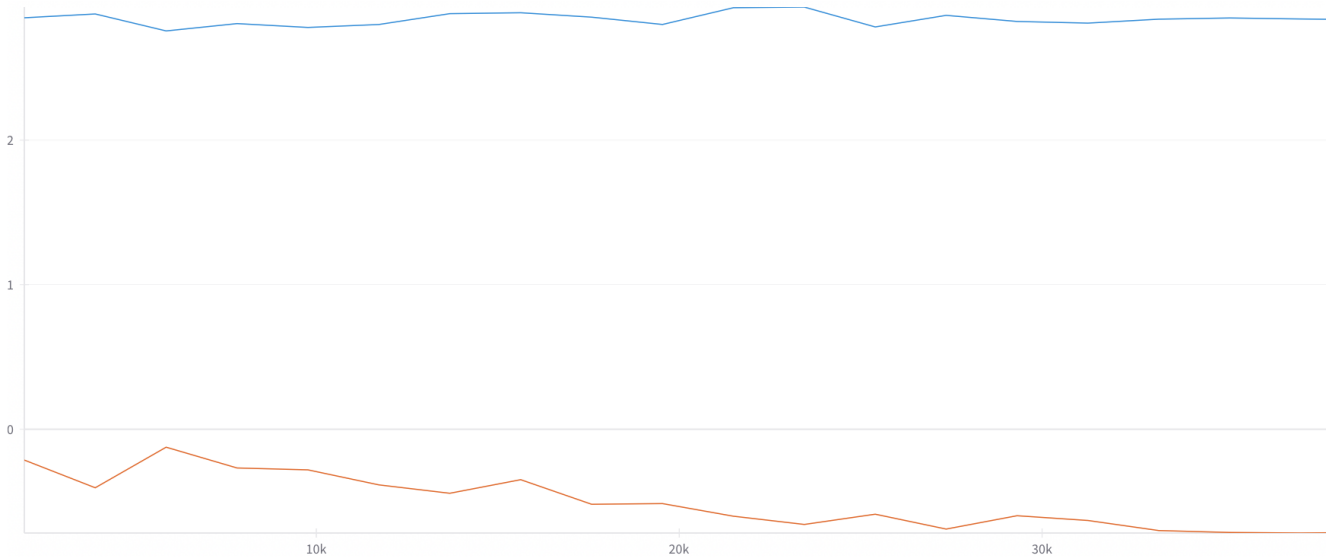


Figure 2: log-determinant (orange) and $log(p_z(x))$ (blue) components of normalizing flow model .

1.3 Q3: Sampling over time

We sampled 1000 points and plotted their propagation over the layers in 4.

1.4 Q4: Sampling trajectories

We sample 10 points from the normalizing flow model and present the forward process layer by layer, as a trajectory in a 2D space in 5.

1.5 Q5: Probability estimation

We chose the following points in table 7 and plotted their in inverse processes layer by layer. Points inside the rings are in red shades, and points outside the rings are in blue shades. original points are with maximum opacity and they became more transparent as flowing through the layers. You can see the plot in 6 and $log(p_z(x))$ in 7. We can see that the outer are points are significantly less likely then the inner points, because the model didn't trained on points outside the rings, and our inverse flow maps the rings into a normal distribution, and points outside the rings are mapped for further points in space.

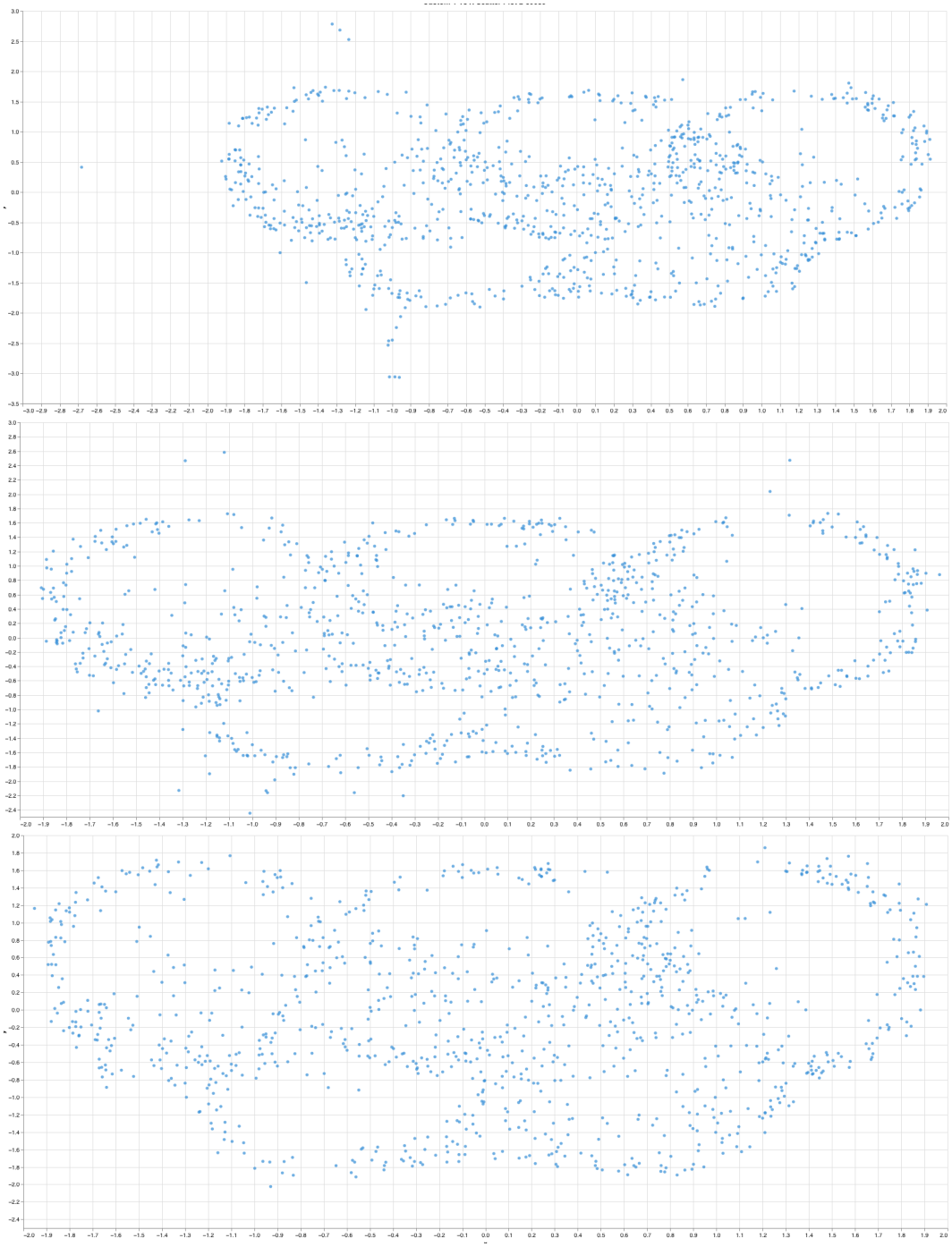
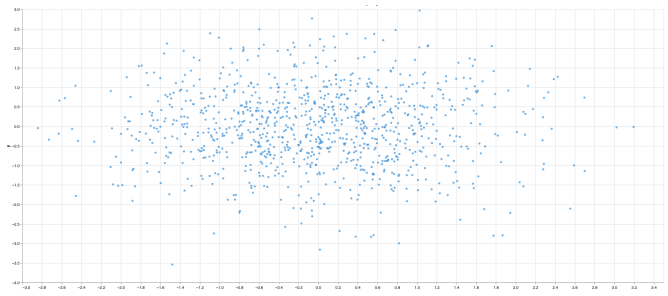
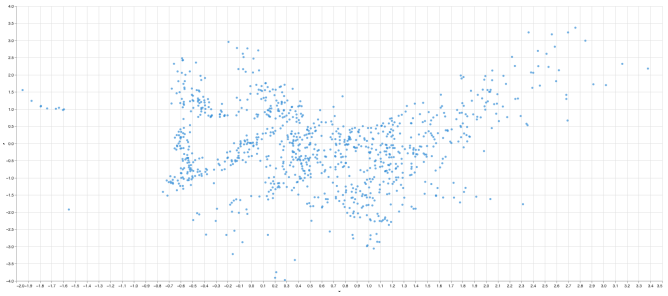


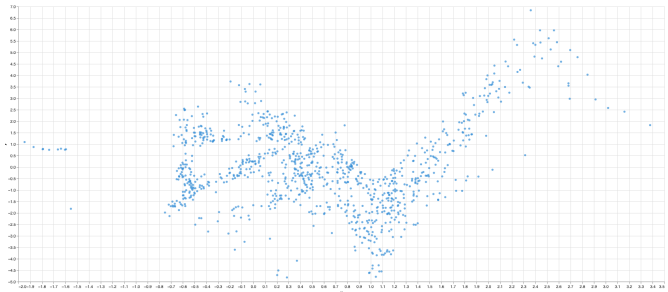
Figure 3: 3 figures with different samplings of 1000 points of normalizing flow model at epoch 20.



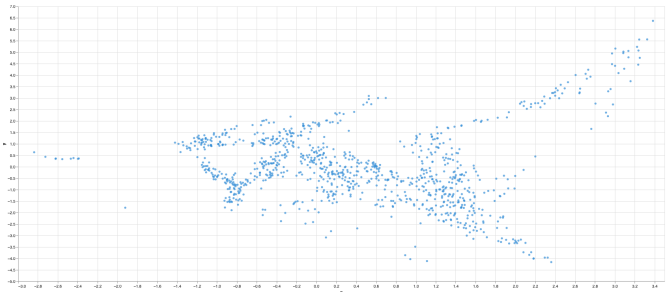
sampling



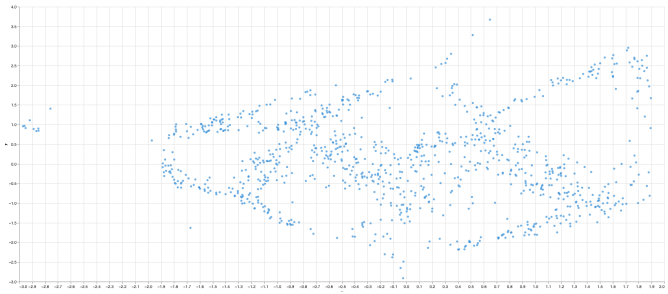
layer 3



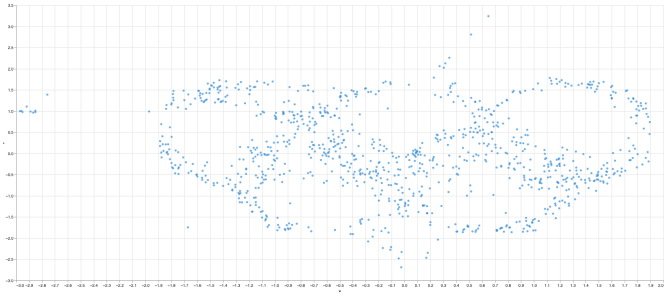
layer 6



layer 9



layer 12



model's output

Figure 4: sample of 1000 points and their flow across the model's layers

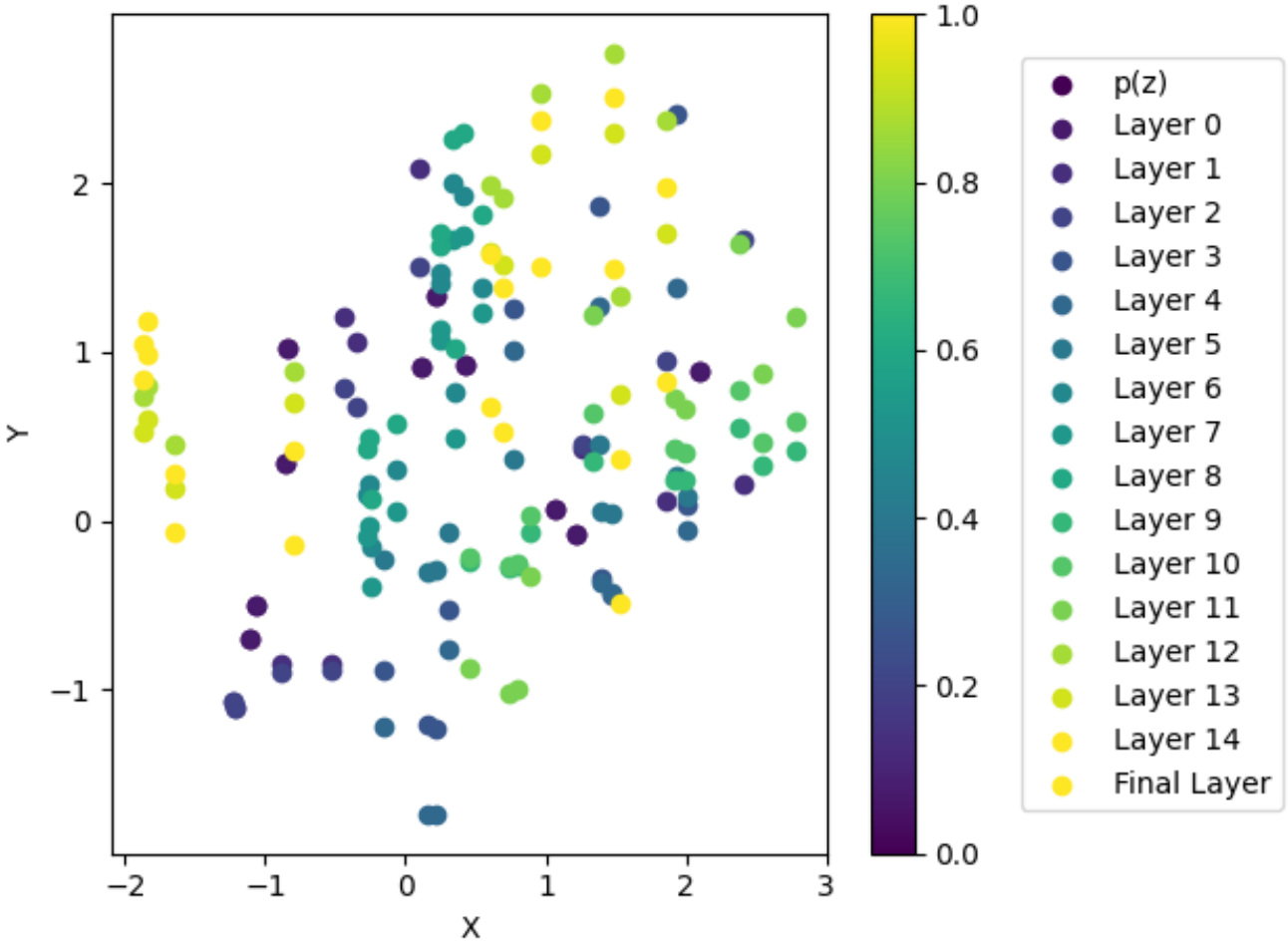


Figure 5: sample 10 points and showing their flow across each layer

.

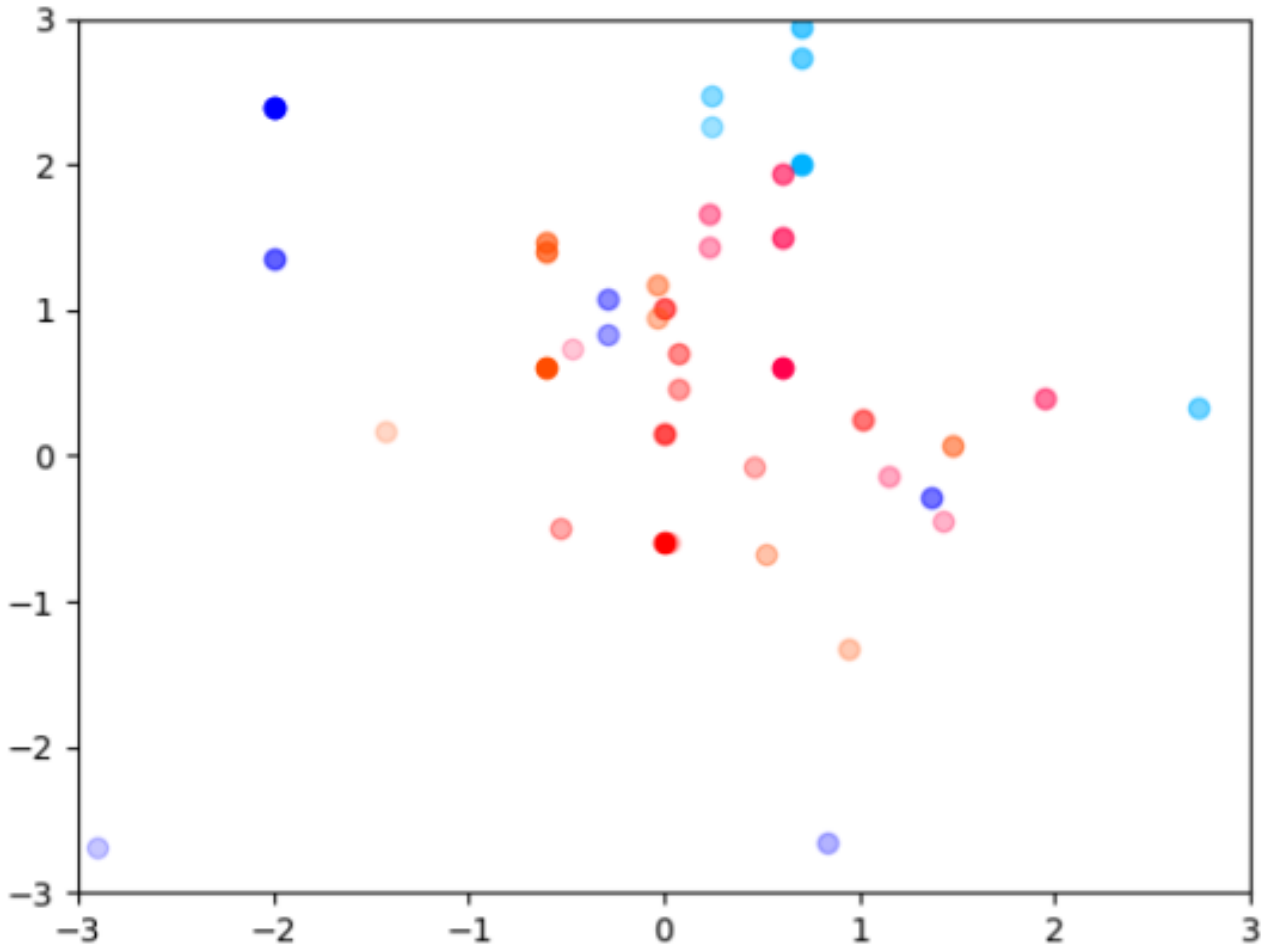


Figure 6: Points from table 7 flowing through the model layers

.

x	y	location	$log(p_z(x))$
0.6	0.6	in	-2.2
0	-0.6	in	-2.5
-0.6	0.6	in	-2.1
-2	2.4	out	-531
0.7	2	out	-52

Figure 7: 5 points

.

2 Unconditional Flow Matching

2.1 Q1: Loss

We present the loss over the training batches in 8.

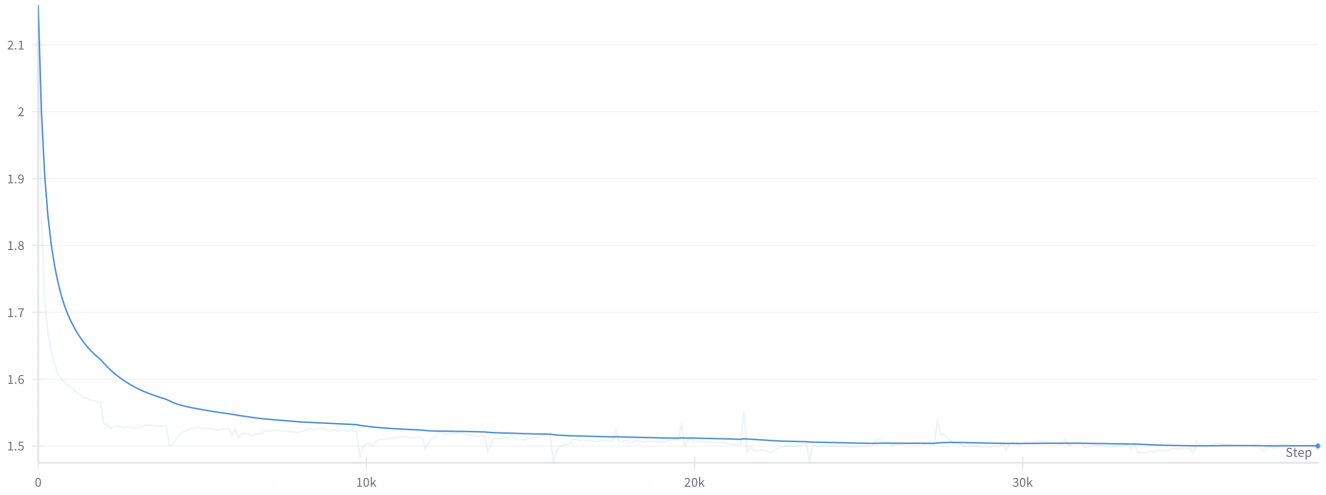


Figure 8: training loss of unconditional flow matching model

.

2.2 Q2: Flow Progression

We sample 1000 points and plot them after each of $t = 0, 0.2, 0.4, 0.6, 0.8, 1$ showing how the distribution progresses in 9.

We can see that the flow converges to the desired probability in a continuous fashion, in contrast to the Normalizing Flow model which had unmeaningfull intermediate representations, due to its permutations and behaviour.

2.3 Q3: Point Trajectory

We sample 10 points the Flow Matching model and present the their forward process as a 2D trajectory in 11. sampled points are with opacity of 0.3 and opacity is rising to 1 while t approaches 1.

Also here we can see how the flow smoothly transforms the points. we can clearly see each point arrive to its final destination. In the noramlizing flow model its hard to see a clear pattern.

2.4 Q4: Time Quantization

We sampled 1000 points from the models using different time quantization $\Delta t = 0.002, 0.02, 0.05, 0.1, 0.2$ in figure 11. We can clearly see that smaller Δt provides better quality results. We are integrating numerically hence when $\Delta t \to 0$ we get better results.

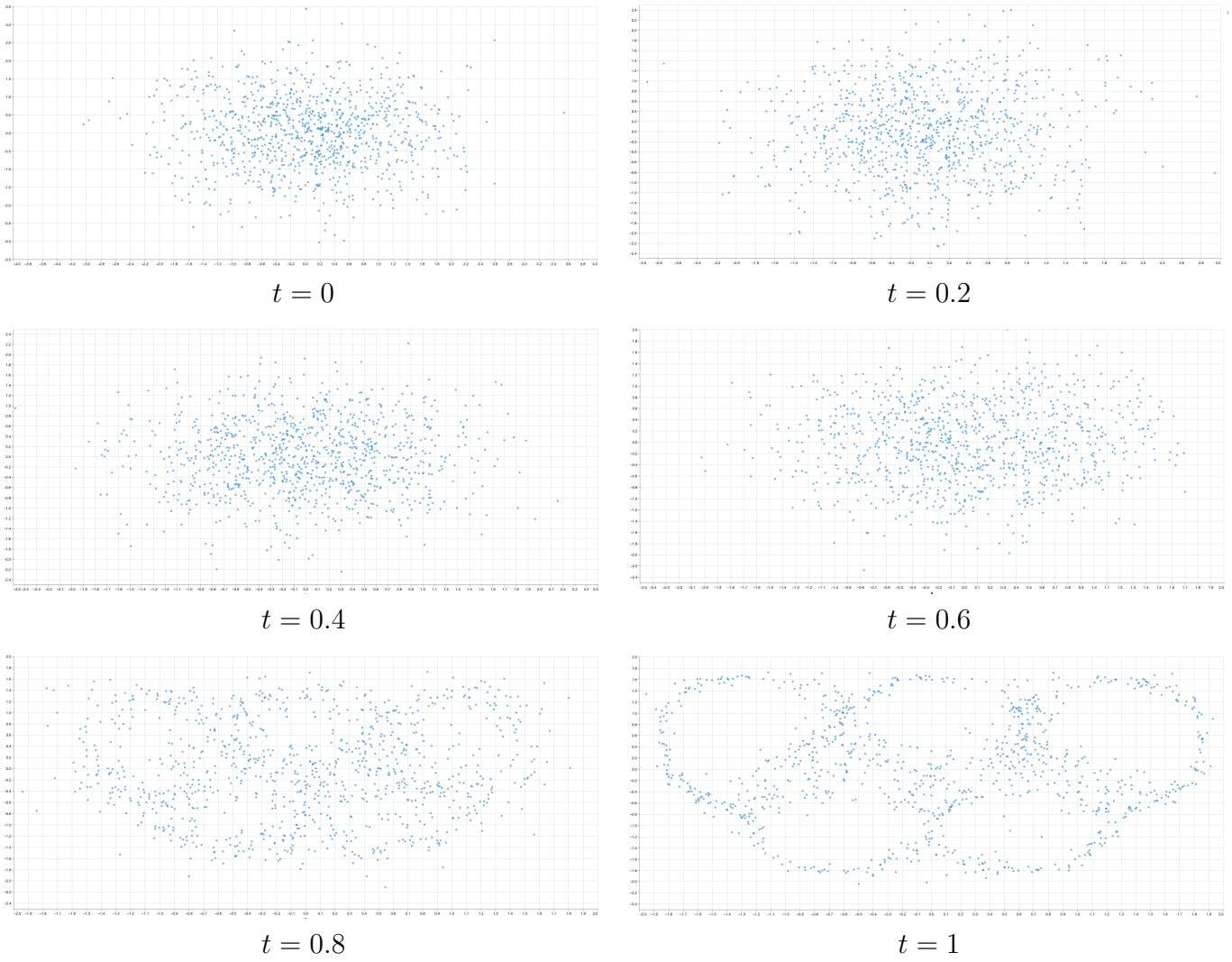


Figure 9: sample of 1000 points and their flow across time in unconditional flow matching model

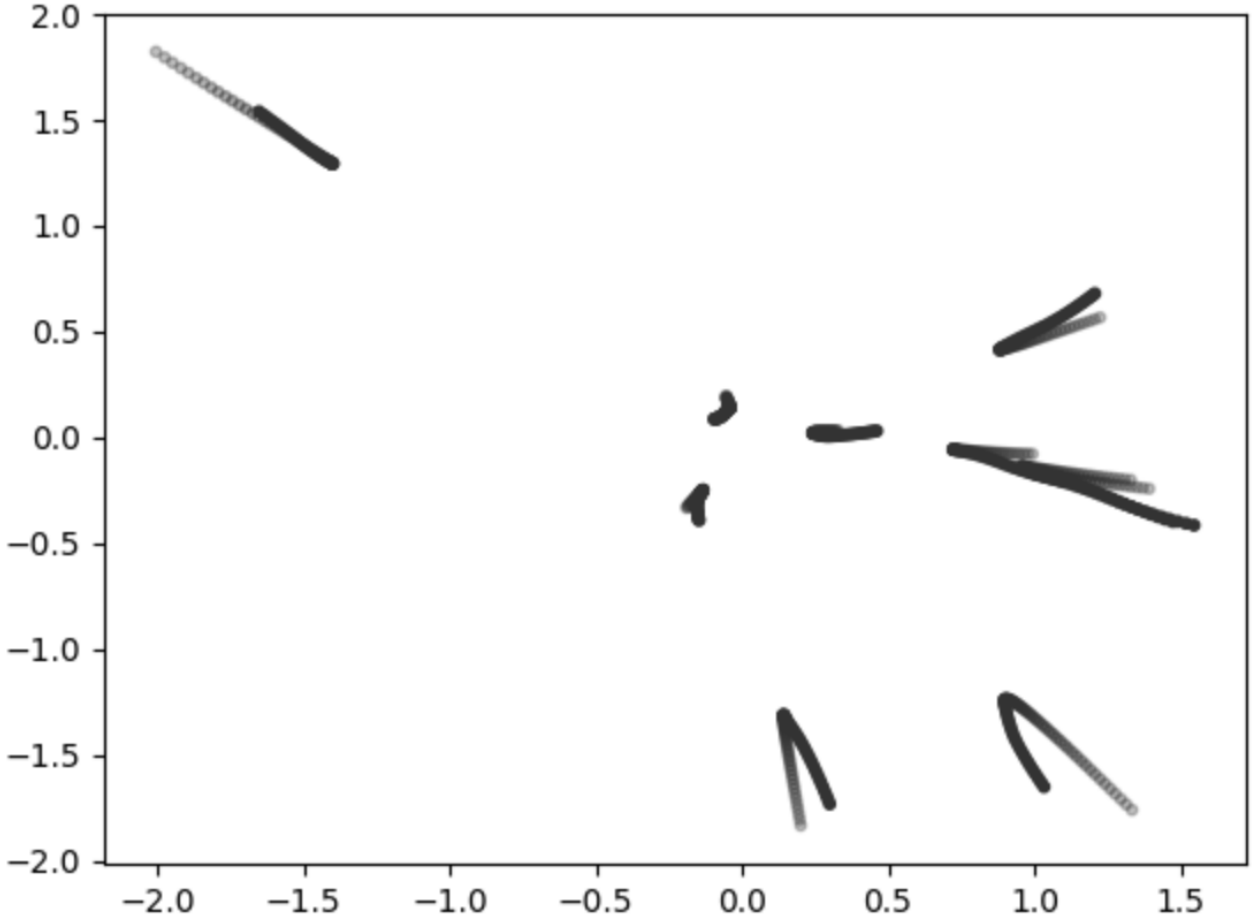


Figure 10: 10 random samples across time in unconditional flow matching model.

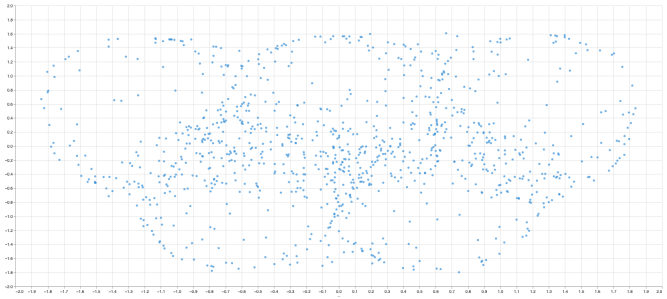
.

2.5 Q5: Reversing the Flow

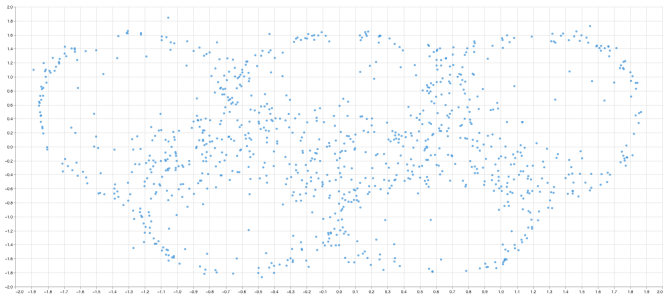
We picked the same 5 points from table 7 and plotted their inverse flow. Each point is marked in black followed by their inverse flow in 12. We can see that the points from inside the rings flows into $0, 0$ and points outside the rings flows against the distribution.

We can see that we get different results, because we worked with different flows that mapped the distributions differently, and we can also see that the sampling process is different hence getting continuous figures and more explainable flows.

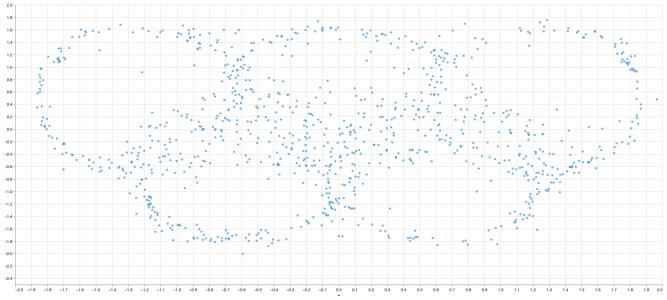
In both of our flows we get the same reconstructed points. The Normalizing Flow model is invertible by definition hence will provide same point. Flow Matching models are not invertible by definition, but because that we learn a continuous vector field, our flow is statistically always smooth, hence providing us with the same reconstructed point.



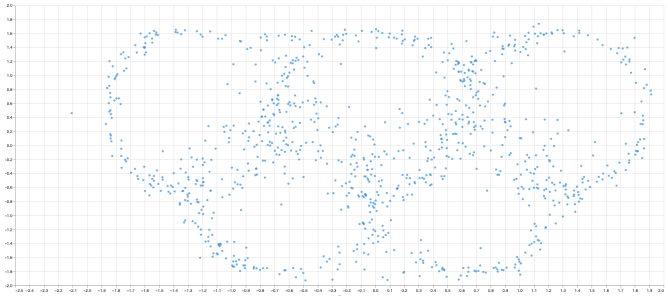
$\Delta t = 0.2$



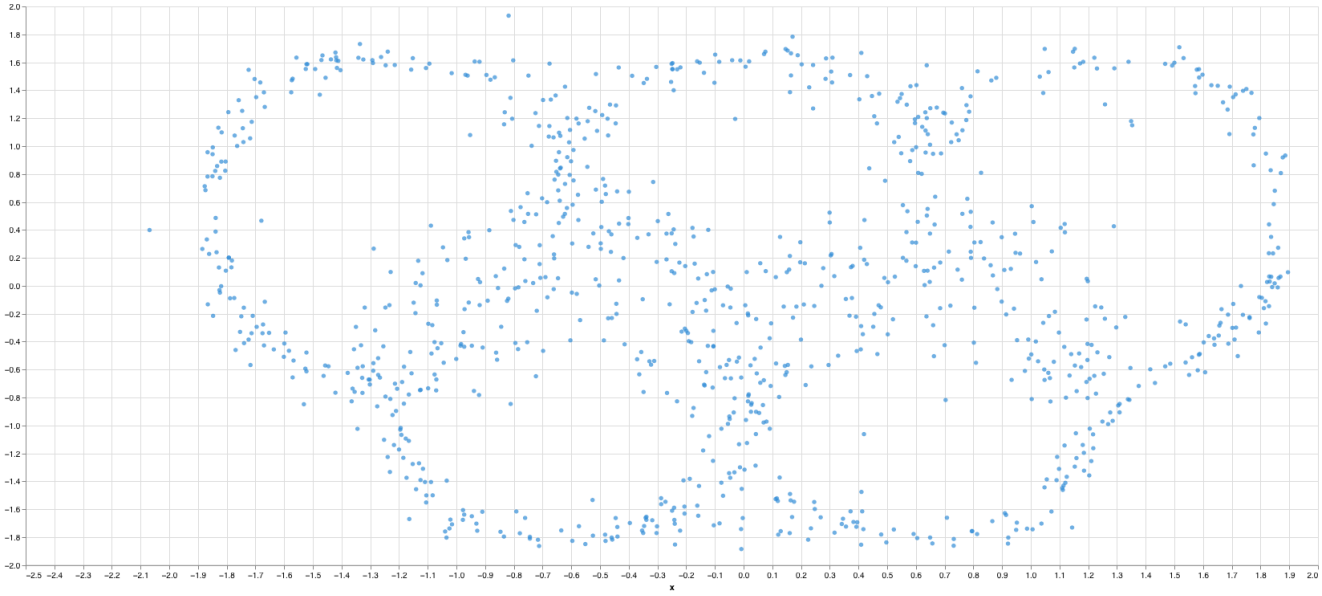
$\Delta t = 0.1$



$\Delta t = 0.05$



$\Delta t = 0.02$



$\Delta t = 0.002$

Figure 11: unconditional model's output with different time

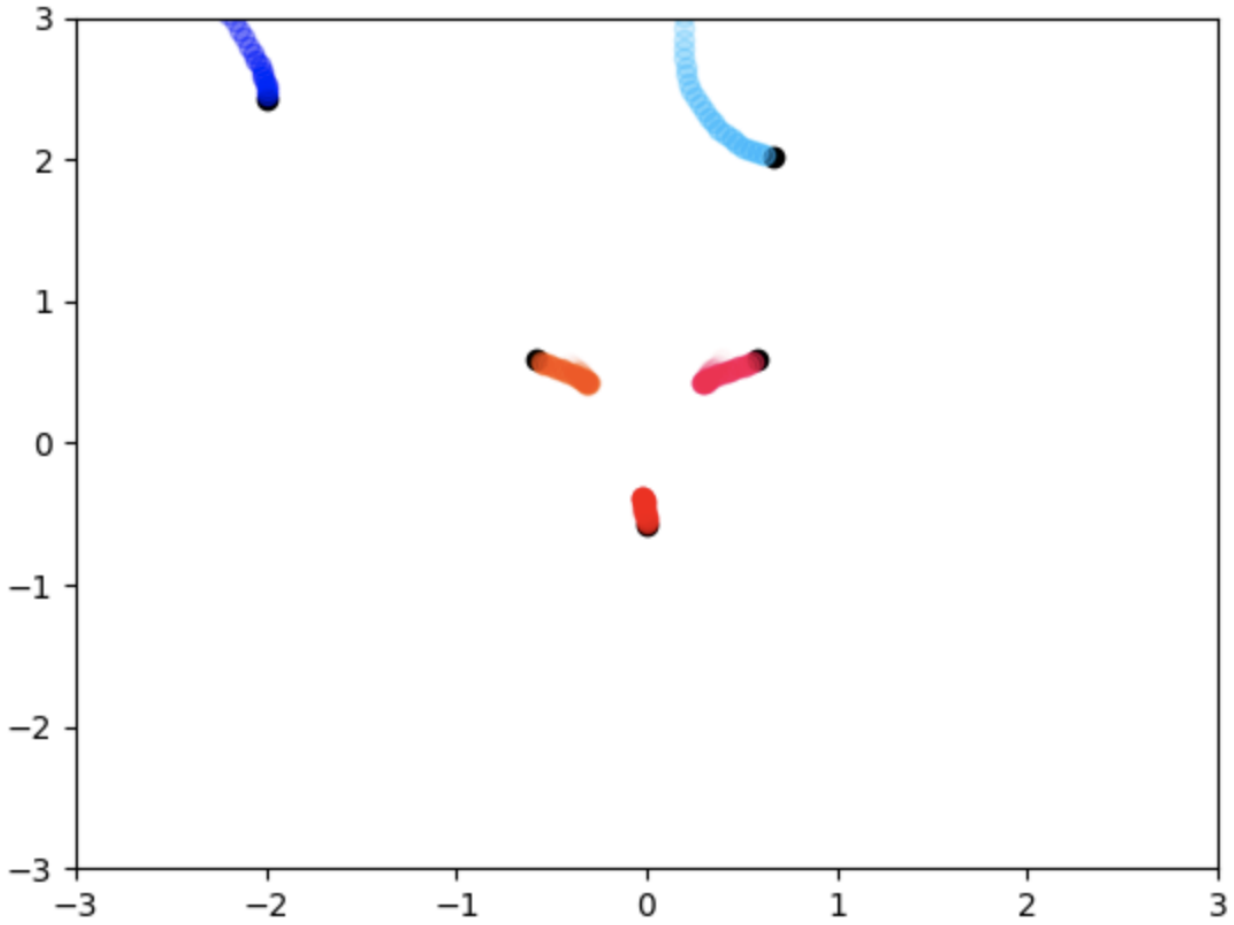


Figure 12: 5 points in an inverse flow of unconditional flow matching model

.

3 Conditional Flow Matching

I added a small embedding layer to the model 13 and concatenated the embedding into x and t passing them together to the first layer as input.

3.1 Plotting the Input

Now we work with conditioned data, you can see it in 14

3.2 A Point from each Class

I really loved that section, it shows exactly how the network maps the distribution for each class, we can see in 15 that for random samples of noise with different conditions, each point arrives to her desired ring.

```
class FlowMatchingCond(nn.Module):
    COND_EMB_SIZE = 3
    def __init__(self, layer_dim, dt):
        self.dt = dt
        self.layers = nn.Sequential(
            nn.Linear(layer_dim + 1 + FlowMatchingCond.COND_EMB_SIZE,
            ↪  64).double(),
            nn.LeakyReLU(),
            nn.Linear(64, 64).double(),
            nn.LeakyReLU(),
            nn.Linear(64, 64).double(),
            nn.LeakyReLU(),
            nn.Linear(64, 64).double(),
            nn.LeakyReLU(),
            nn.Linear(64, layer_dim).double(),
        )

        self.cond_layer = torch.nn.Embedding(5,
        ↪  FlowMatchingCond.COND_EMB_SIZE)
```

Figure 13: FlowMatchingCond Architecture

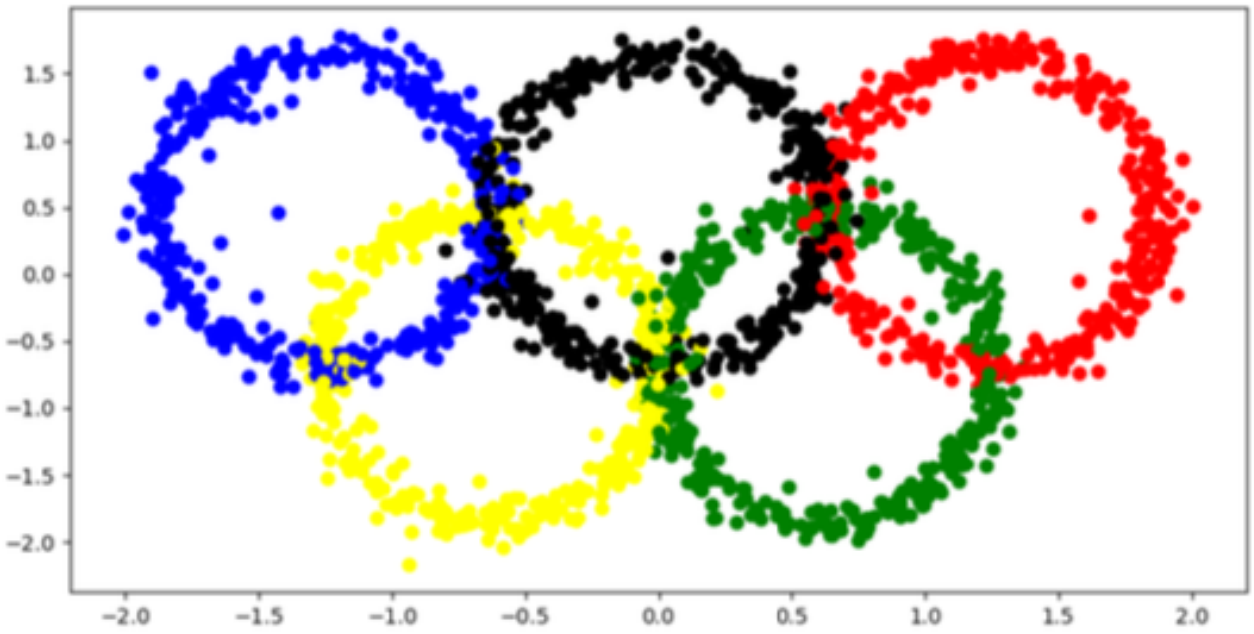


Figure 14: conditioned flow model data distribution

.

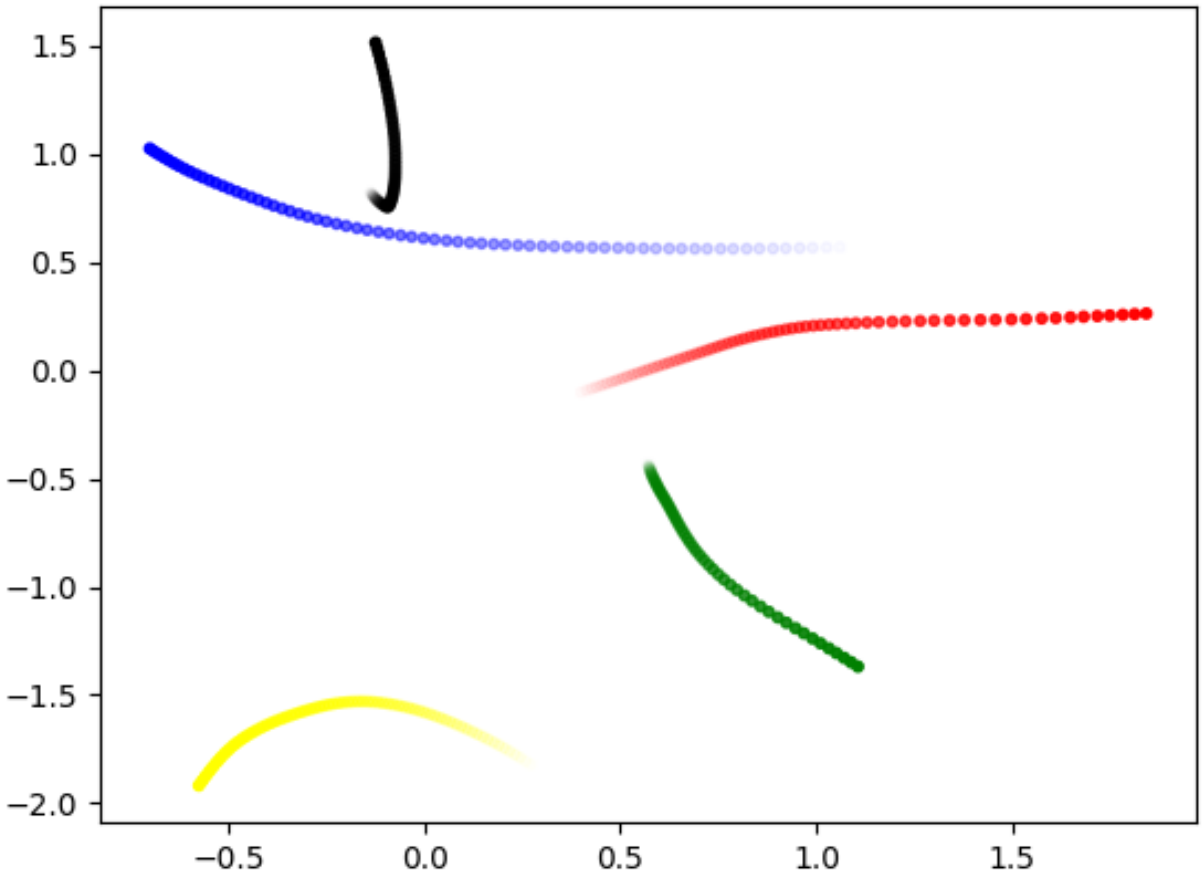


Figure 15: A sample for each class propagates through a conditional flow matching model

.

3.3 Sampling

We plot the sample of $3,000$ points in 16. In order to insert the condition we changed only the equation of u to get the condition as well.

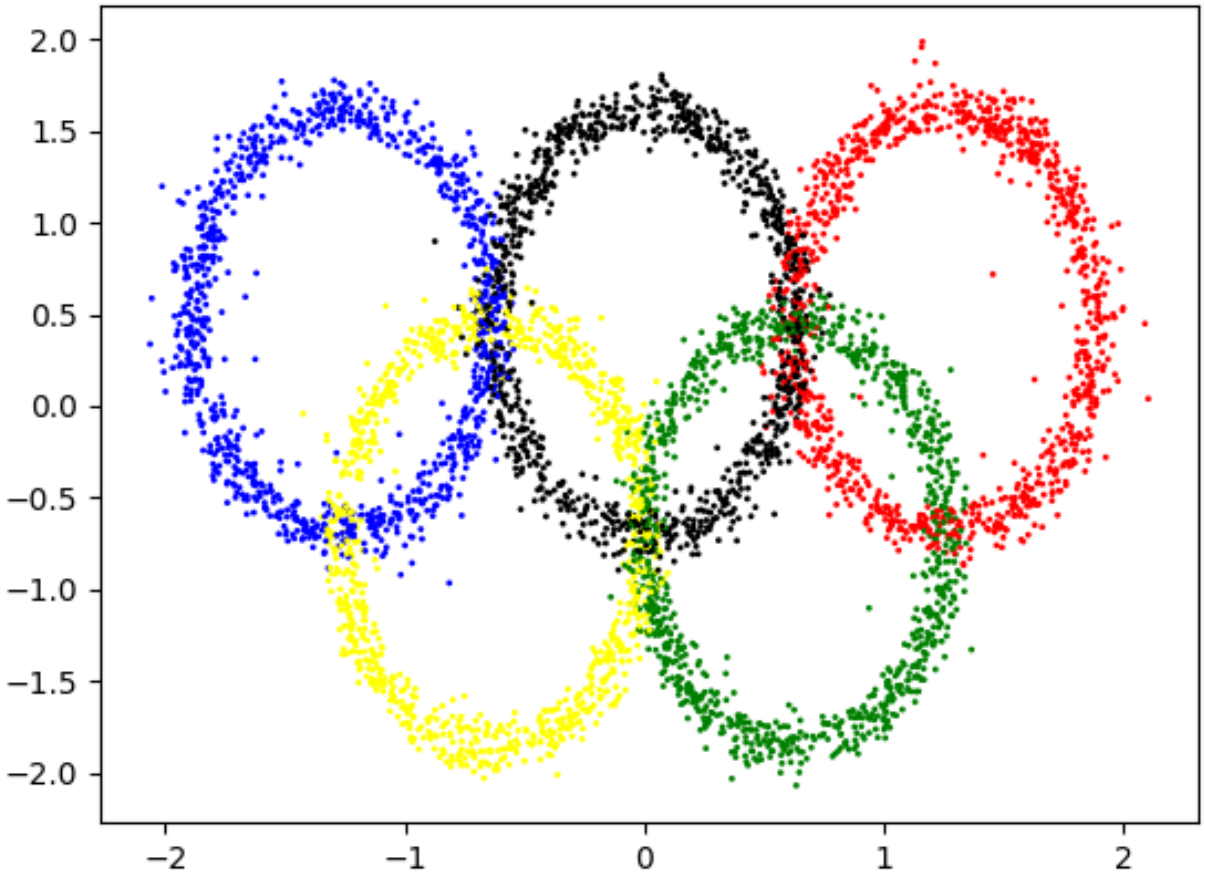


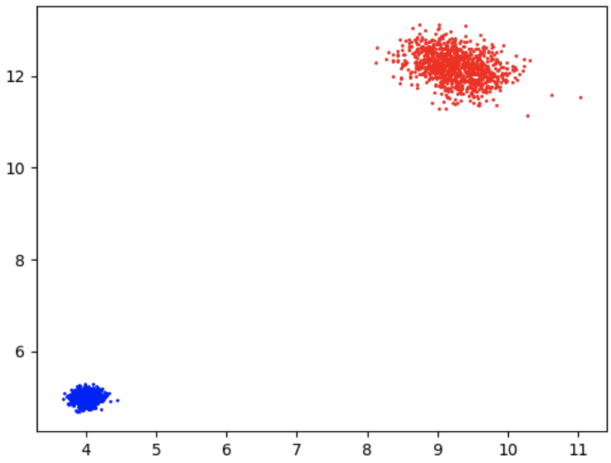
Figure 16: A sample of $3,000$ conditioned points

.

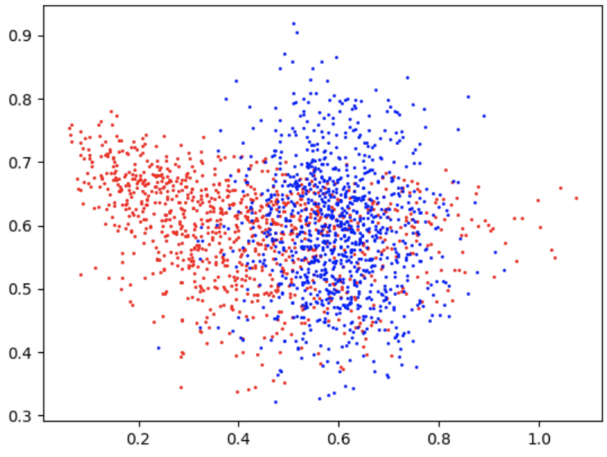
4 Bonus

In order to find a sample that arrives to $(4, 5)$, I just computed the inverse of $(4, 5)$ using the flow. After observing the reconstruction of $(4, 5)$ I seen that there is a slight shift in results, so I sampled $1,000$ points from a normal distribution around $(4, 5)$ and computed their reconstruction. I have observed that in the Flow Matching model the distributions stay close. I plotted the noise (red), and the reconstructed points (blue) for 2 distributions around $(4, 5)$ which is outside the rings, and for $(0.6, 0.6)$ which is inside the ring in figure 17. We can clearly see that the inverse of the outside distribution is around $(4, 4)$ which is not our prior distribution. and the inverse samples around $(0.6, 0.6)$ is a distributed normally.

Finlay, if we will pass the input $(10, 12)$ we would get $(4, 5)$ as an output.



(4,5) out



(0.6, 0.6) in

Figure 17: sample of 1000 points and their flow across the model's layers