Statistical Methods in Image Processing EE-048954

Homework 1: Kernel Density Estimation and Normalizing Flows

Neria Uzan 311598098, Or Glassman 302314174

Part I: Kernel Density Estimation (30 points)

The multivariate kernel density estimate of a density $f(\mathbf{x})$ given a set of samples $\{\mathbf{x}_i\}$ is given by

$$\hat{f}(\mathbf{x}) = \frac{1}{N}\frac{1}{|H|}\sum_{i=1}^{N} K(H^{-1}(\mathbf{x}_i - \mathbf{x})),$$

where H is a bandwidth matrix, $K(\cdot)$ is the kernel and $\{\mathbf{x}_i\}_{i=1}^{N}$ are *i.i.d.* samples drawn from $f(\mathbf{x})$.

Task 1. Consider the following density functions:

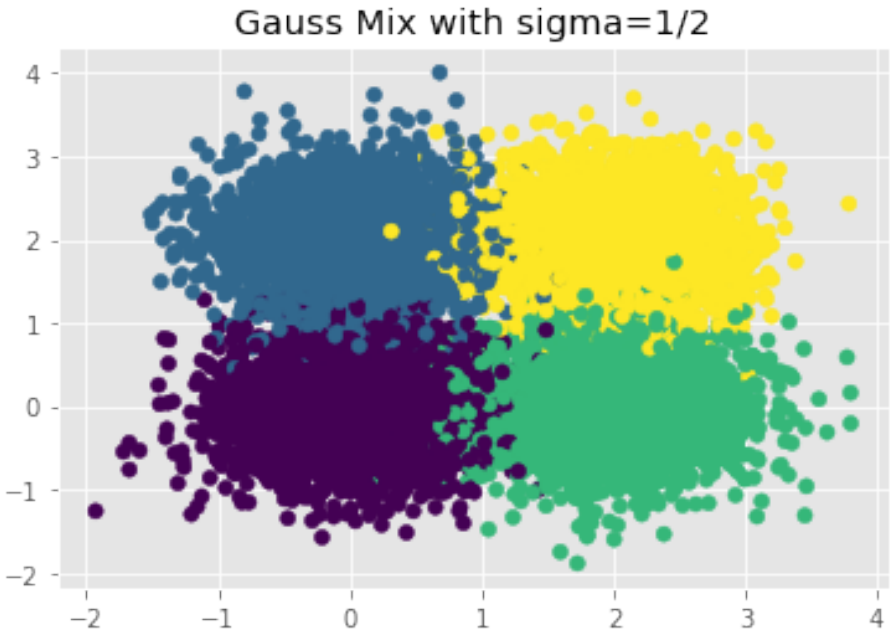
- Gaussian Mixture:

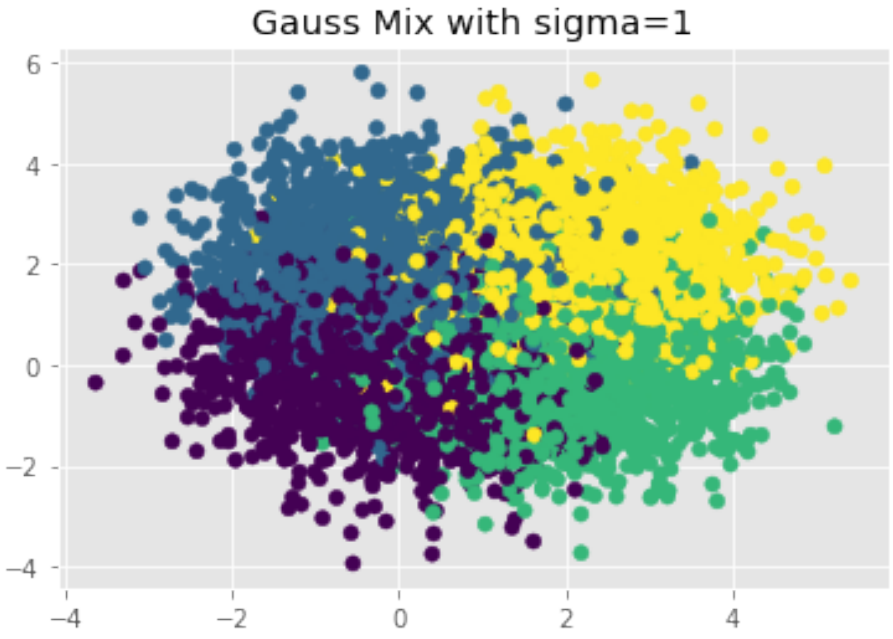
$$f(\mathbf{x};\sigma,\{\mu_i\}) = \frac{1}{2\pi\sigma^2}\sum_{m=1}^{M}\frac{1}{M}\exp\left\{-\frac{1}{2\sigma^2}||\mathbf{x}-\mu_i||^2\right\},$$

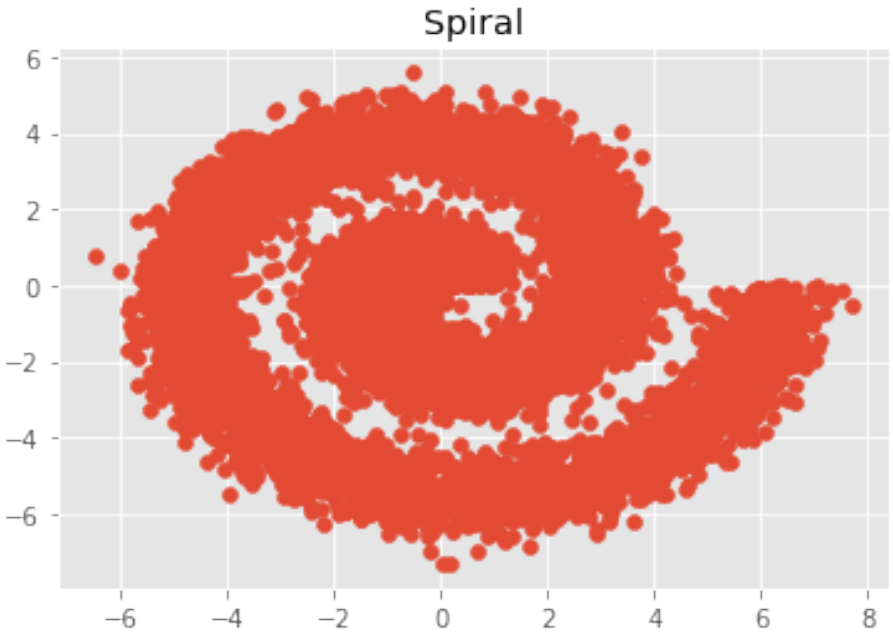
  with $M = 4$, $\sigma = \frac{1}{2}$, and $\{\mu_m\} = \left\{(0,0)^T, (0,2)^T, (2,0)^T, (2,2)^T\right\}$.
- Gaussian Mixture with $M = 4$, $\sigma = 1$, $\{\mu_m\} = \left\{(0,0)^T, (0,2)^T, (2,0)^T, (2,2)^T\right\}$.
- Spiral with $\theta \sim \mathcal{U}[0, 4\pi]$ and $r|\theta \sim \mathcal{N}\left(\frac{\theta}{2}, 0.25\right)$.

For each of the three distributions above, implement a function that draws $N = 10000$ samples $\mathbf{x}_i$ from $f(\mathbf{x})$. Display the drawn samples for each distribution separately.

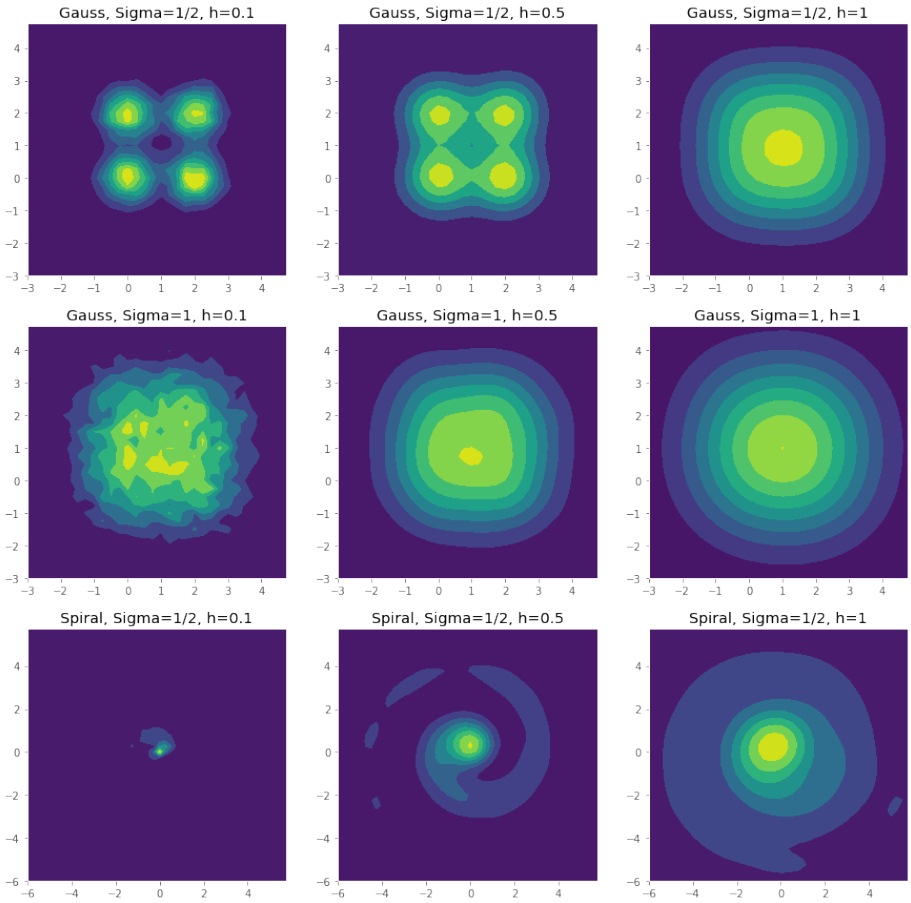
Solution:


Gauss Mix with sigma=1/2
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4


Gauss Mix with sigma=1
6
4
2
0
−2
−4
−4
−2
0
2
4



Task 3. For **each** distribution, compare between $f(\mathbf{x})$ and $\hat{f}(\mathbf{x})$ using the bandwidth matrices $H = \begin{pmatrix} h & 0 \\ 0 & h \end{pmatrix}$ with $h = 0.1, 0.5, 1.$ and display the estimation. Discuss the trade-off of the choice of h.



Discussion

On the simplest level, the higher the h the wider the kernel, therefore for a higher h we get a smoother estimated density function. This is shown mostly on the gauss with sigma=1 in which for the higher h the density is estimated as a one gaussian. This is an example of the bias-variance trade off in which we can see that for the example above it's almost not matter the specific samples of the data, the results will always look like a single gaussian, as oppsed to the lower h in the gaussian with sigma=1/2 in which we can see that each sample can have higher siginicance and the fedeliaty to the dataset will be higher and the bias lower.

Task 4. Which of the distributions was the easiest/hardest to estimate? Why?

Solution: Based on this last observation, the spiral draw is harder to estimate than the Gaussian draw. As can be seen that the tail of the spiral is hard to estimate correctly. We conjecture that this is due to the face that the kernel is

symmetric and the dataset is not, this can cause some kind of "aliasing" in the density function.

Part II: Invertible Neural Networks (70 points)

In this part, we will take a closer look at invertible neural networks, otherwise known as *Normalizing Flows*. The most popular, current application of normalizing flows is to model datasets of images. In this part, we will implement a type of flows called *Coupling Flows* for a simplified problem of sampling from toy 2D datasets, although similar concepts (with deeper models + tricks) can be used to model images.

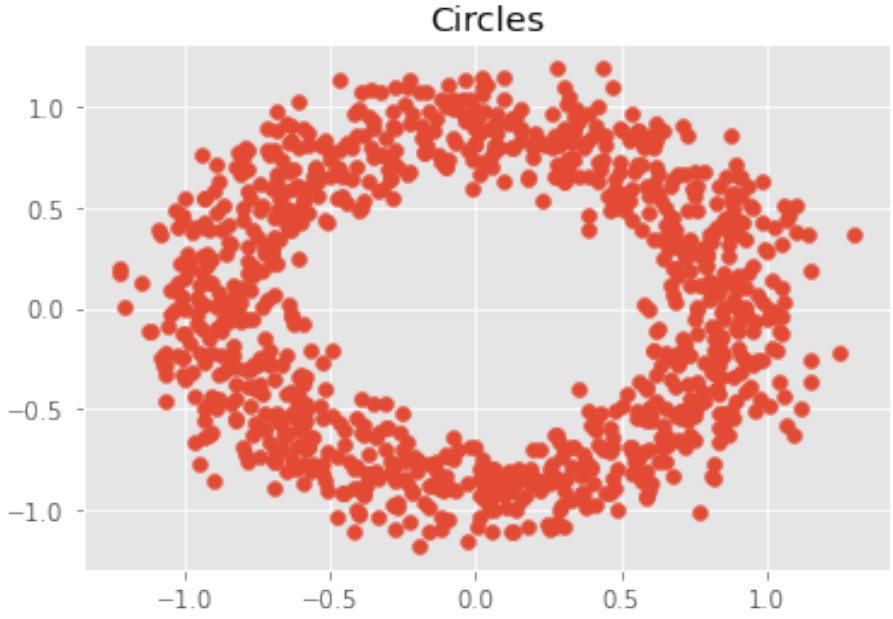
General Concept

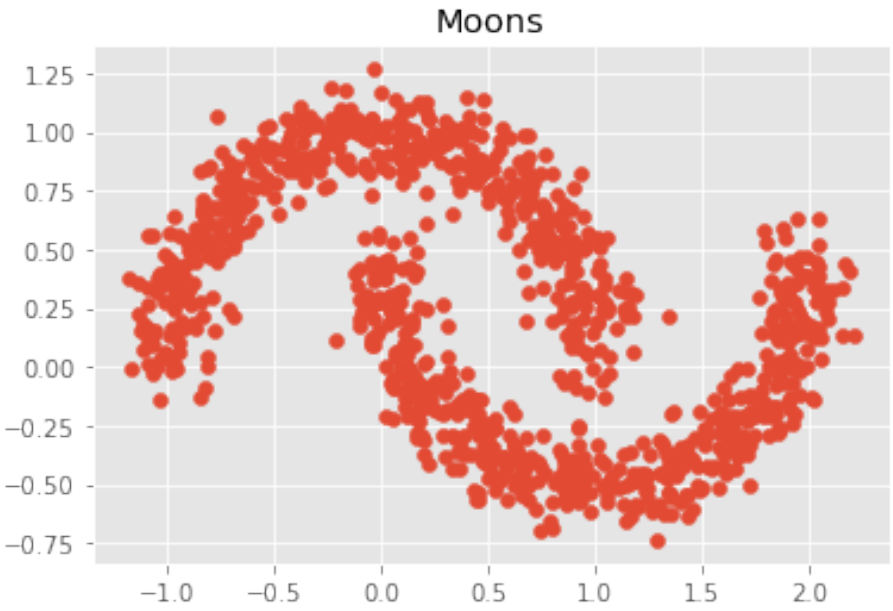
Recall that given a random vector Z with a density $p_Z(\mathbf{z})$ (e.g. Gaussian) and an invertible function f, the density $p_X(\mathbf{x})$ of $X = f(Z)$ is given by:

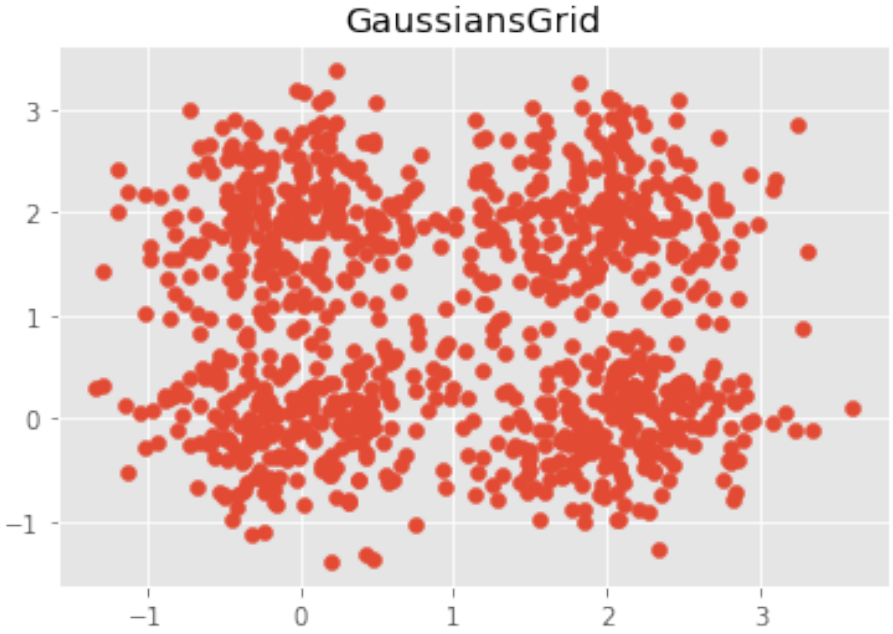
$$\log p_X(\mathbf{x}) = \log p_Z(f^{-1}(\mathbf{x})) + \log \left| \det \frac{df^{-1}(\mathbf{x})}{d\mathbf{x}} \right|$$

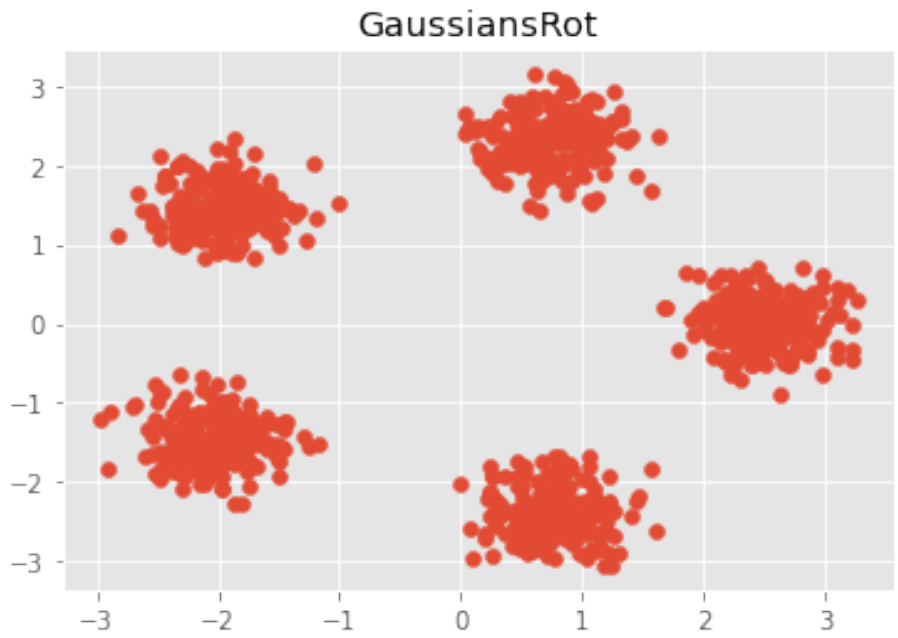
Task 1. To get acquainted with this function, for each of the 4 distributions above, draw $N = 1000$ samples x_i. Display the drawn samples for each distribution in a separate plot.

Solution:




Moons
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


GaussiansGrid
3
2
1
0
−1
−1
0
1
2
3



Coupling Layers

Next, we look at possible transformations to apply inside the flow, focusing on the simplest and most efficient one. A recent popular flow layer, which works well in combination with deep neural networks, is the coupling layer introduced by Dinh et al.. The input $\mathbf{x}$ is arbitrarily split into two parts, $\mathbf{x}_{1:j}$ and $\mathbf{x}_{j+1:d}$, of which the first remains unchanged by the flow. Yet, $\mathbf{x}_{1:j}$ is used to parameterize the transformation for the second part, $\mathbf{x}_{j+1:d}$. In this coupling layer, we apply an affine transformation by scaling the input by $\mathbf{s}$ and shifting it by $\mathbf{t}$. In other words, our transformation $\mathbf{y} = f(\mathbf{x})$ looks as follows:

$$\mathbf{y}_{1:j} = \mathbf{x}_{1:j}$$
$$\mathbf{y}_{j+1:d} = \mathbf{s}_{\theta_1}(\mathbf{x}_{1:j}) \odot \mathbf{x}_{j+1:d} + \mathbf{t}_{\theta_2}(\mathbf{x}_{1:j})$$

$\mathbf{y}$ is the output of the flow layer, the functions $\mathbf{s}$ and $\mathbf{t}$ are implemented as neural networks, and the sum and multiplication are performed element-wise. Here's a block diagram that visualize the coupling layer in the form of a computation graph:

For convience, since we train using the log-density, it is common practice to apply an exponential on the predicted scaling factor prior to multiplication with $\mathbf{x}_{j+1:d}$, as this simplifies the calculation of the log-determinant of the Jacobian that we will derive shortly. Hence, the implemented transformation in practice is usually:

$$\mathbf{y}_{1:j} = \mathbf{x}_{1:j}$$
$$\mathbf{y}_{j+1:d} = \exp(\mathbf{s}_{\theta_1}(\mathbf{x}_{1:j})) \odot \mathbf{x}_{j+1:d} + \mathbf{t}_{\theta_2}(\mathbf{x}_{1:j})$$

Task 2. Write down the inverse of this layer $\mathbf{x} = f^{-1}(\mathbf{y})$ for some $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Draw the inverse function $f^{-1}(\mathbf{y})$ as a computational graph that has $\mathbf{y}$ as input and $\mathbf{x}$ as output.

Solution:

Split the input vector x into $x_{1:j}$ and $x_{j+1:d}$. We have:

1. $$y_{1:j} = x_{1:j}$$
2. $$y_{j+1:d} = S_{\theta_1}\left(x_{1:j}\right) \cdot x_{j+1:d} + t_{\theta_2}\left(x_{1:j}\right)$$

Consequently, we can write that:

$$x_{1:j} = y_{1:j}$$

To evaluate $x_{j+1:d}$, notice that:

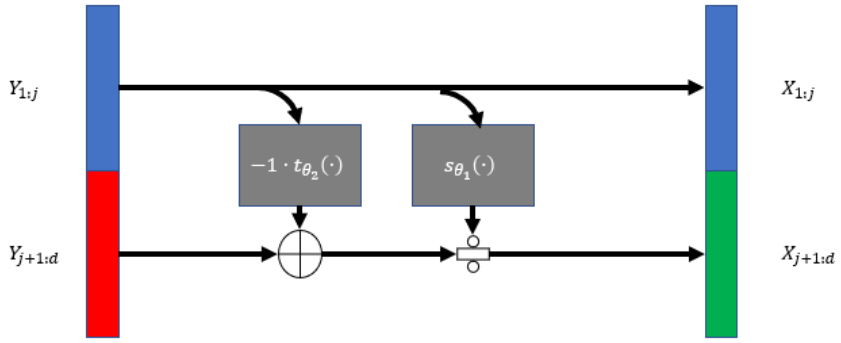
$$y_{j+1:d} = S_{\theta_1}\left(y_{1:j}\right) \cdot x_{j+1:d} + t_{\theta_2}\left(y_{1:j}\right)$$

And as a result:

$$x_{j+1:d} = \left(y_{j+1:d} - t_{\theta_2}\left(y_{1:j}\right)\right) / S_{\theta_1}\left(y_{1:j}\right)$$

Where the division is **element-wise**.

The inverse network can be viewed as the following computational graph:



Task 3. Write down the Jacobian of this layer $\frac{d\mathbf{y}}{d\mathbf{x}}$. Do you recognize a special structure in this matrix?

Solution:

1. For indices $a, b \in \{1 \dots j\}$: $\frac{dy_a}{dx_b} = \delta_{a,b}$
2. For indices $a, b \in \{j+1 \dots d\}$: $\frac{dy_a}{dx_b} = \delta_{a,b} \cdot \exp\{s(x_{b-d})\}$
3. For indices $a \in \{1 \dots j\}, b \in \{j+1 \dots d\}$: $\frac{dy_a}{dx_b} = 0$
4. For indices $a \in \{j+1 \dots d\}, b \in \{1 \dots j\}$: there is no simple form

Consequently, the Jacobian matrix takes the following shape:

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{y_{j+1:d}}}{d\mathbf{x_{1:d}}} \\ 0 & Diag\big(\exp\big\{s\left(x_{1:j}\right)\big\}\big) \end{pmatrix}$$

Task 4. Write down the explicit expression for the *log*-determinant of the Jacobian matrix from the previous section.

Solution:

Because the Jacobian matrix is of block structure, its determinant is the product of each block's determinant $(ac - bd)$:

$$\det\left(\frac{d\mathbf{y}}{d\mathbf{x}}\right) = \det\begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{y_{j+1:d}}}{d\mathbf{x_{1:d}}} \\ 0 & Diag\big(\exp\big\{s\left(x_{1:j}\right)\big\}\big) \end{pmatrix} = \det\left(Diag\big(\exp\big\{s\left(x_{1:j}\right)\big\}\big)\right) =$$

$$= \prod_{i=j+1}^{d} \exp\left\{s\left(x_{1:j}\right)\right\} = \exp\Big\{\sum_{i=j+1}^{d} \left(s\left(x_{1:j}\right)\right)_i\Big\}$$

$$\Rightarrow \log\det\left(\frac{d\mathbf{y}}{d\mathbf{x}}\right) = \sum_{i=j+1}^{d} \left(s\left(x_{1:j}\right)\right)_i$$

Task 5. Write down the explicit expression for the *log*-determinant of the Jacobian matrix of the *inverse* function $\frac{d\mathbf{x}}{d\mathbf{y}}$.

Solution:

The inverse transformation is very simillar except that t's sign is negative, and we divide rather than multiply by the exponent:

$$\frac{d\mathbf{x}}{d\mathbf{y}} = \begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{x_{j+1:d}}}{d\mathbf{y_{1:j}}} \\ 0 & Diag\big(\exp\big\{-s\left(y_{1:j}\right\}\big) \end{pmatrix}$$

And since we saw this structure in the above section we can deduce:

$$\Rightarrow \log\det\left(\frac{d\mathbf{x}}{d\mathbf{y}}\right) = -\sum_{i=j+1}^{d} \big(s\big(y_{1:j}\big)\big)_i$$

Coupling Flows

As you might have guessed by now, a coupling layer is powerful yet still limited in its ability to significantly alter the input. This is because it only operates on a chunk of it with element-wise manipulations due to the invertability constraint. We can go on with making our function f more complex. How can we implement more complex invertible functions? The answer is: invertible function composition. We can stack multiple invertible functions $f_1, \ldots, f_K$ (e.g. Coupling Layers) after each other, as all together, they still represent a single, invertible function. Specifically, if $\mathbf{y} = f_1(\mathbf{z})$ and $\mathbf{x} = f_2(\mathbf{y})$ are invertible functions, then $\mathbf{x} = f_2 \circ f_1(\mathbf{z})$ is an invertible function and its inverse is given by $f_1^{-1} \circ f_2^{-1}$. More importantly, the calculation of the log-determinant of the Jacobian in this case is simple using the chain rule.

Task 7. Assuming $\mathbf{y} = f_1(\mathbf{z})$ and $\mathbf{x} = f_2(\mathbf{y})$ are coupling layers, calculate the log-determinant of the Jacobian $\frac{d\mathbf{x}}{d\mathbf{z}}$. How is it related to the log-determinant of the Jacobians $\frac{d\mathbf{y}}{d\mathbf{z}}$ and $\frac{d\mathbf{x}}{d\mathbf{y}}$?

Solution:

By using the chain rule, we can write:

$$\frac{dx}{dz} = \frac{dx}{dy}\frac{dy}{dz} = \begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{x_{j+1:d}}}{d\mathbf{y_{1:j}}} \\ 0 & Diag\big(\exp\big\{-s_2\left(y_{1:j}\right)\big\}\big) \end{pmatrix} \cdot \begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{y_{j+1:d}}}{d\mathbf{z_{1:j}}} \\ 0 & Diag\big(\exp\big\{-s_1\left(z_{1:j}\right)\big\}\big) \end{pmatrix} =$$

$$= \begin{pmatrix} \mathbb{I}_j & \frac{d\mathbf{y_{j+1:d}}}{d\mathbf{z_{1:j}}} + \frac{d\mathbf{x_{j+1:d}}}{d\mathbf{y_{1:j}}} \cdot Diag\big(\exp\big\{-s_1\left(z_{1:j}\right)\big\}\big) \\ 0 & Diag\big(\exp\big\{-s_2\left(y_{1:j}\right)\big\}\big) \cdot Diag\big(\exp\big\{-s_1\left(z_{1:j}\right)\big\}\big) \end{pmatrix}$$

Note that the top-right element can be dismissed as it vanishes when calculating the Jacobian.

Coupling layers generalize to any masking technique we could think of. However, the most common approach is to split the input $\mathbf{x}$ in half using the mask. For our toy 2D datasets comprised of samples $\mathbf{x} = (p_x, p_y) \in \mathbb{R}^{d=2}$, this means that either $\{\mathbf{x}_{1:j} = p_x, \mathbf{x}_{j+1:d} = p_y\}$, or $\{\mathbf{x}_{1:j} = p_y, \mathbf{x}_{j+1:d} = p_x\}$. These correspond to masks $[1, 0]^T$ and $[0, 1]^T$ respectively. Note that when we apply multiple coupling layers, we invert the masking every other layer so that each variable is transformed a similar amount of times.

Intuition in 1D Intuitively, using multiple, learnable invertible functions, a normalizing flow attempts to transform $p_z(z)$ slowly into a more complex distribution which should finally be $p_x(x)$. We visualize the idea below (figure credit - Lilian Weng):

Starting from z_0, which follows the prior Gaussian distribution, we sequentially apply the invertible functions $f_1, f_2, ..., f_K$, until z_K represents x.

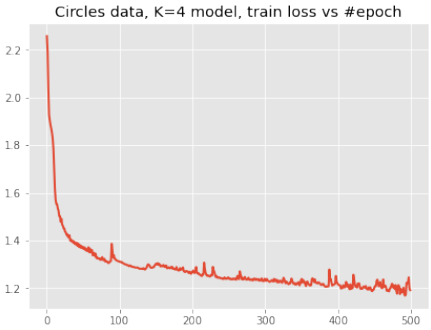
Training Coupling Flows

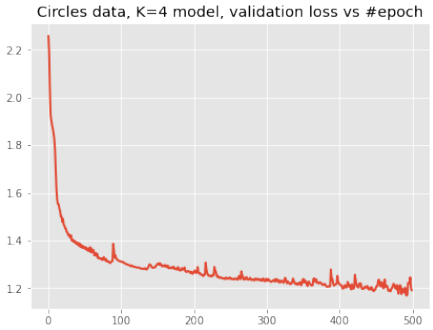
Task 9. For each of the 4 provided distributions (use the default value of `num_gaussians=5` for `'GaussiansRot'`), use a similar snippet to repeat the following:

- Create a `ToyDataset` instance with $N = 1500$ samples from the distribution.
- Learn a `CouplingFlow` model with $K = 4$ layers, by training for 500 epochs. For the `'Moons'` dataset train for 1000 epochs.
- Plot the estimated density $p(\mathbf{x})$ in $\mathbb{R}^2$. To achieve this, use the method `log_probability` to calculate the log-probability $\log p(\mathbf{x})$ at a predetermined grid of points $\mathbf{x} \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \subset \mathbb{R}^2$ (e.g. using `np.meshgrid`). Sample each coordinate with at least 100 points (i.e. a grid of 100×100 positions in 2D). Plot the resulting 2D distribution $p(\mathbf{x}) = \exp(\log p(\mathbf{x}))$ as an image where the value in each pixel is $p(\mathbf{x})$.
- Plot samples from intermediate flow layers, including the prior $p(\mathbf{z})$ and the modelled $p(\mathbf{x})$. To achieve this, use the method `sample_x_each_layer` with $N = 1000$ samples. Plot the resulting samples from each layer in the **same** axis limits to visualize the transformation of each coupling layer separately. You can use `plt.subplot(..., sharex=True, sharey=True)` to achieve link the axis of all subplots.

Circles Solution:

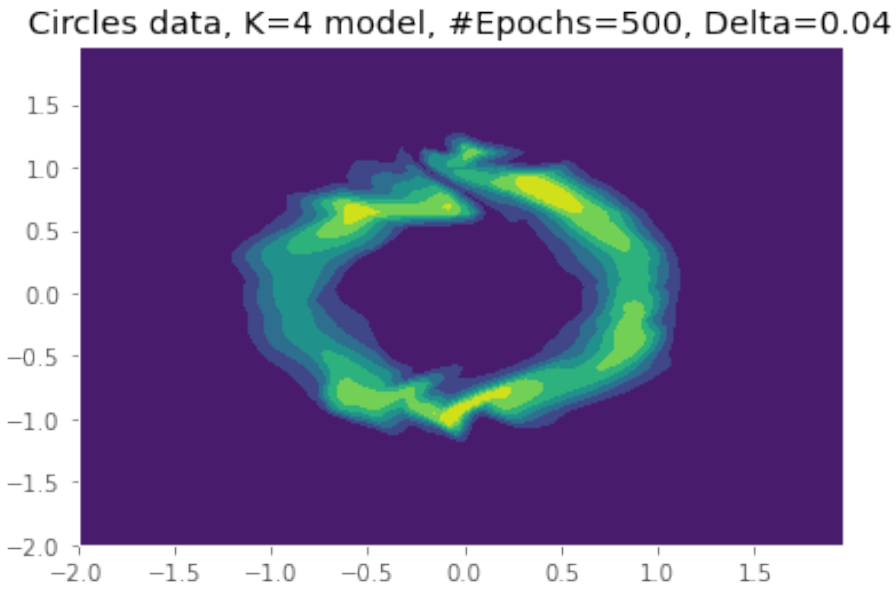
Loss graphs:





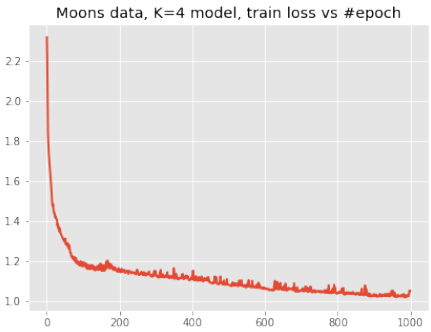
Sampling each layer:

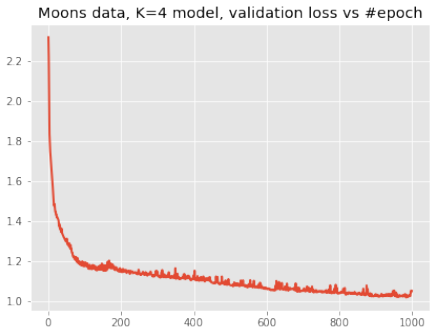
Estimated density at output:



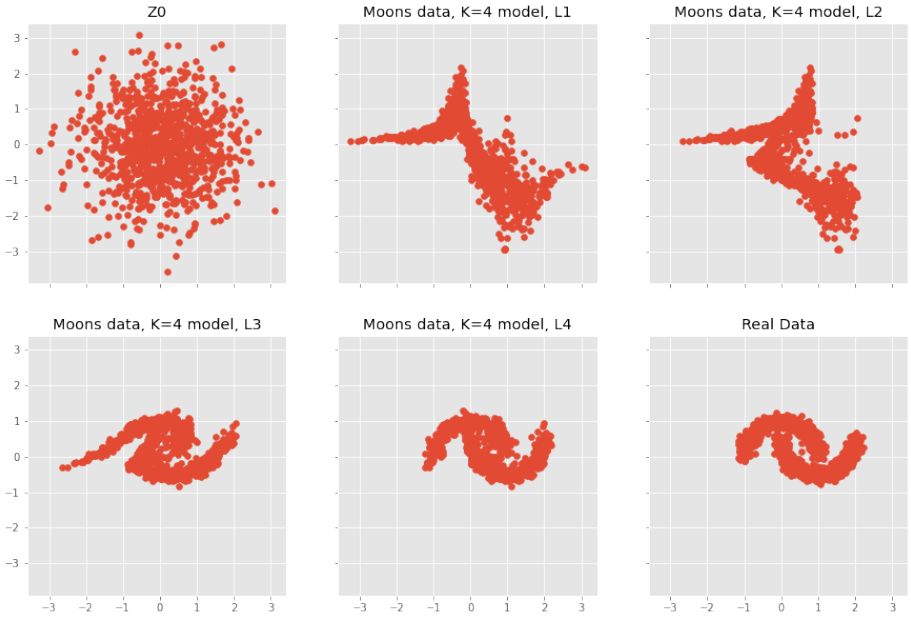
Moons Solution:

Loss graphs:

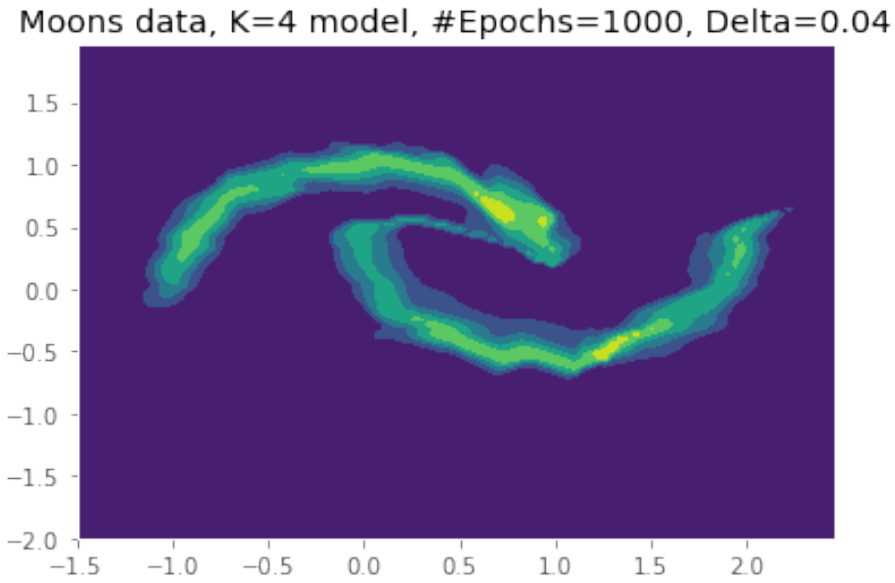




Sampling each layer:

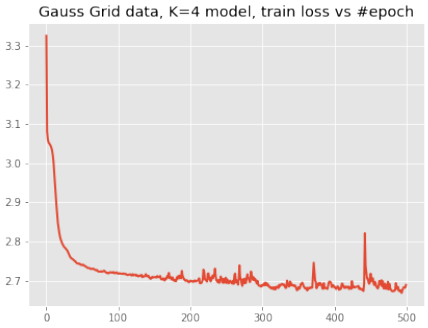


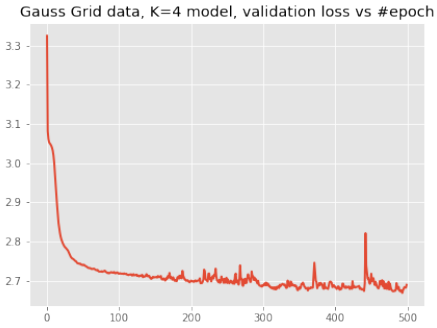
Estimated density at output:



Gaussian Grid Solution:

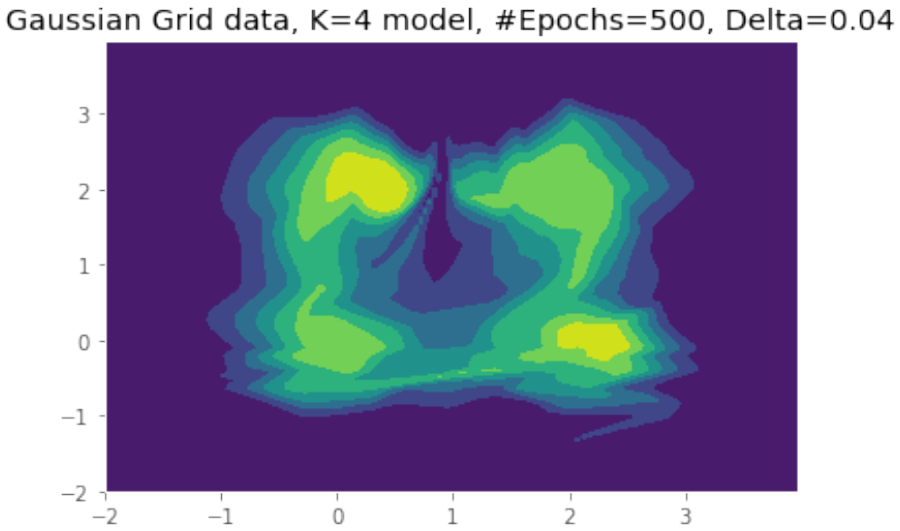
Loss graphs:





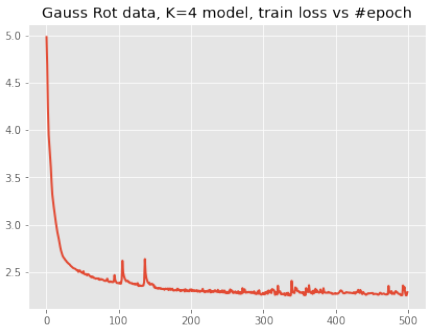
Sampling each layer:

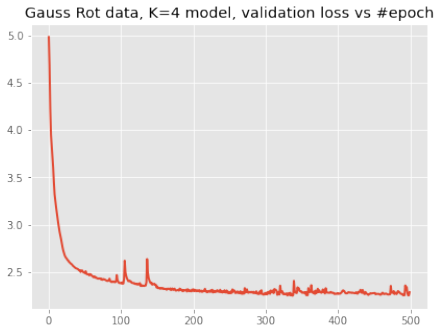
Estimated density at output:



Gaussian Rot Solution:

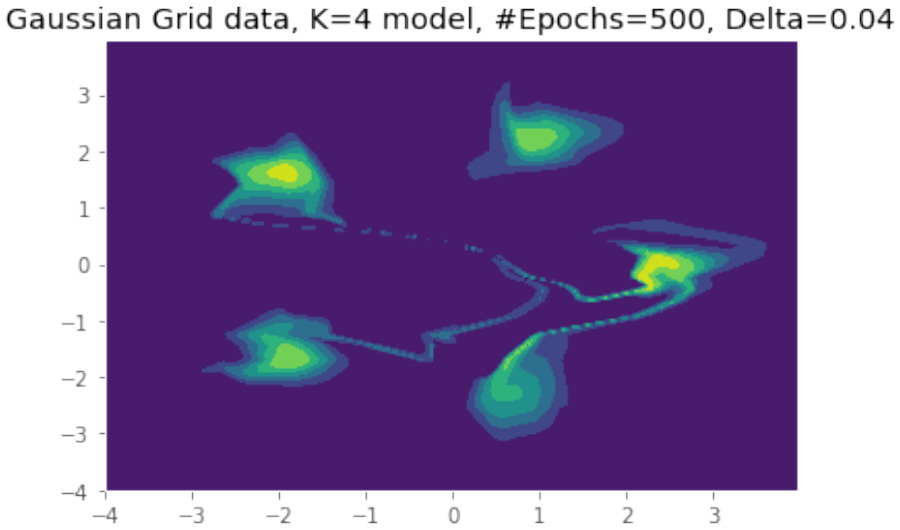
Loss graphs:





Sampling each layer

Estimated density at output:



Analyzing Coupling Layers

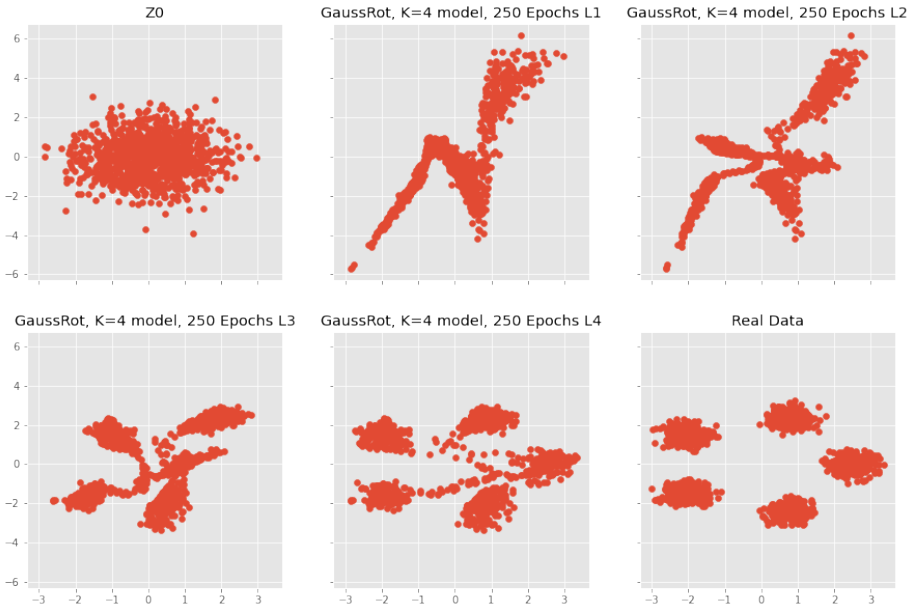
Task 10. Train two flow models with a varying number of coupling layers $K = \{2, 4\}$ for 250 epochs, using $N = 1500$ samples from the `'GaussiansRot'` dataset with `num_gaussians=5`. Compare the resulting estimated density $p(\mathbf{x})$ (using a grid of 100×100 points) and the intermediate distributions of $N = 1000$ samples throughout the coupling layers of the model. Which model fits $p(\mathbf{x})$ better? What do you conclude regarding the effect of model depth? explain the result in your report and attach the resulting plots.

Solution:

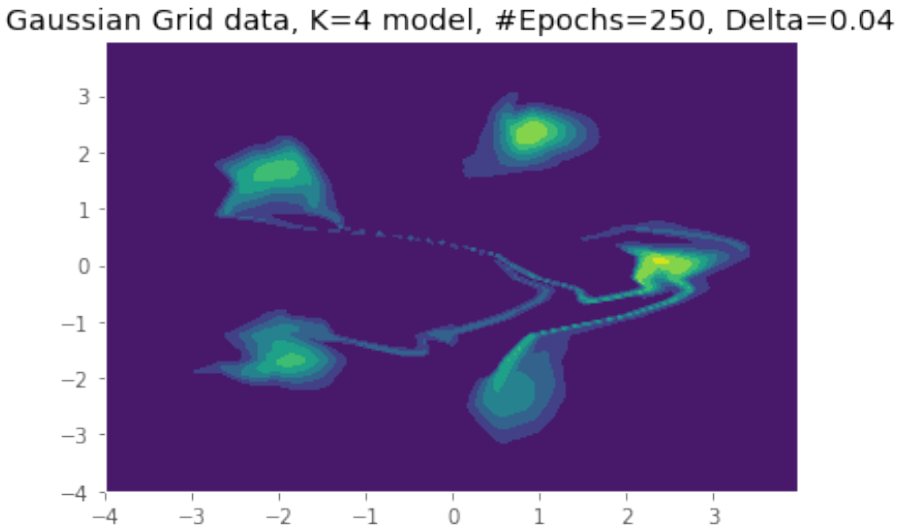
4 layers:

Best Model achieved 2.3887 validation loss at epoch 243.

Sampling each layer:



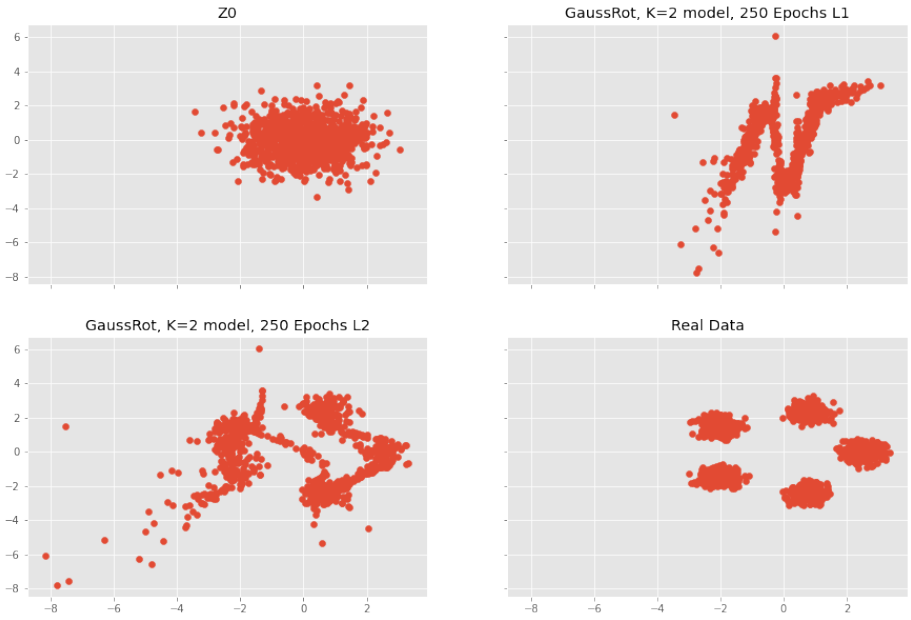
Output density:



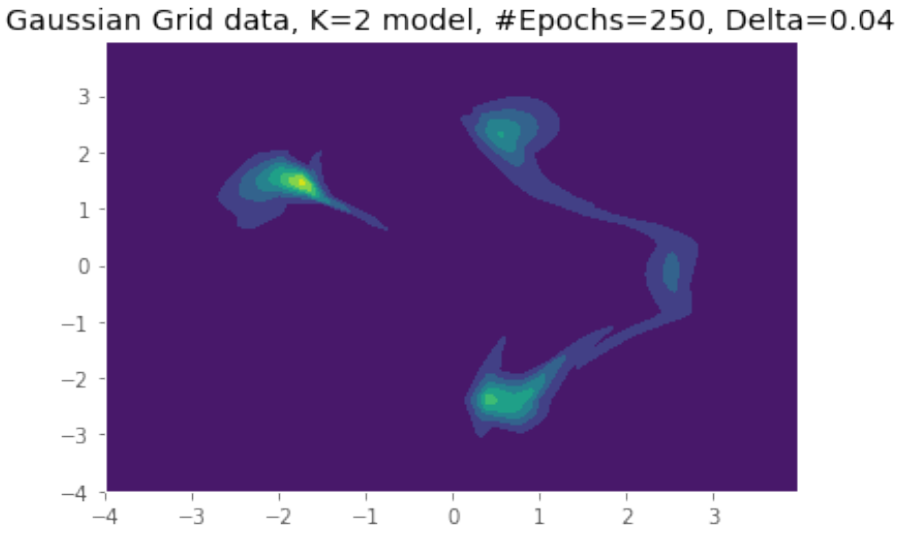
2 layers:

Best Model achieved 2.5809 validation loss at epoch 244.

Sampling each layer:



Output KDE:



Discussion:

We can see that the number of layers is critical. We conjecture that is critical because the Gaussian Rot data need to be describe by at least 3 parameters - outside radius, inner variance, and coordinates of the circles. Therefore requires high expresivity of the network and therefor require more layers. We verified

this conjecture by trying the same expirement on the Circles data which require basiclly one parameter - radius. The results between the circles data with K=2 and K=4 was more simmilar that the difference between the Gauss Rot with K=2, 4.

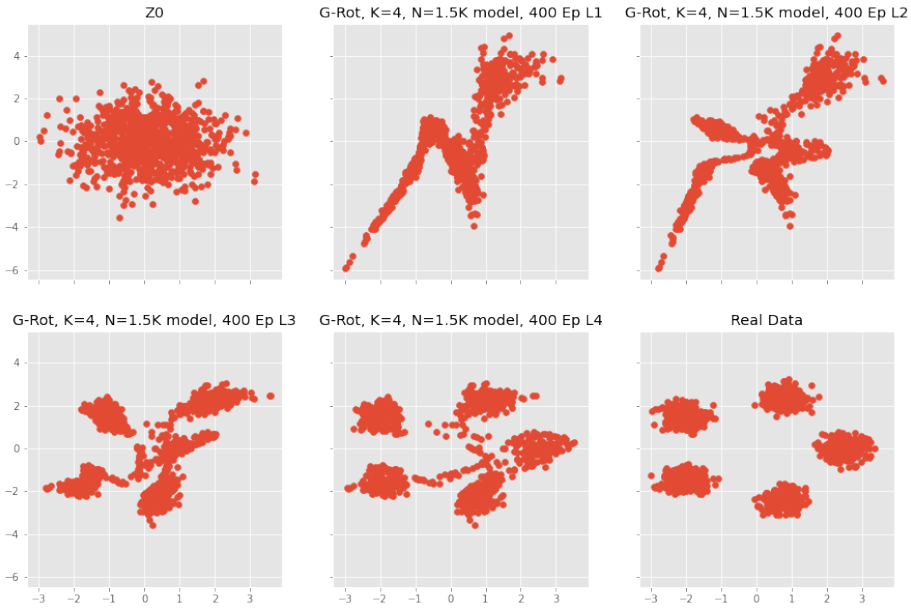
Task 11. Train two flow models with $K = 4$ layers for 400 epochs, using a varying number of $N = \{1500, 3000\}$ samples from the `'GaussiansRot'` dataset with `num_gaussians=5`. Compare the resulting estimated density $p(\mathbf{x})$ (using a grid of 100×100 points) and the intermediate distributions of $N = 1000$ samples throughout the coupling layers of the model. Which model fits $p(\mathbf{x})$ better? What do you conclude regarding the effect of training set size? explain the result in your report and attach the resulting plots.

Solution:

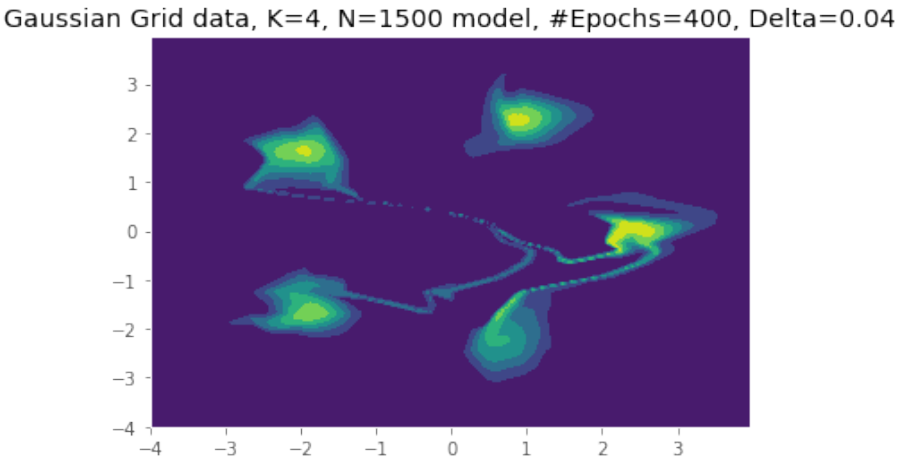
1500 samples:

Best Model achieved 2.3523 validation loss at epoch 271.

Sampling each layer:



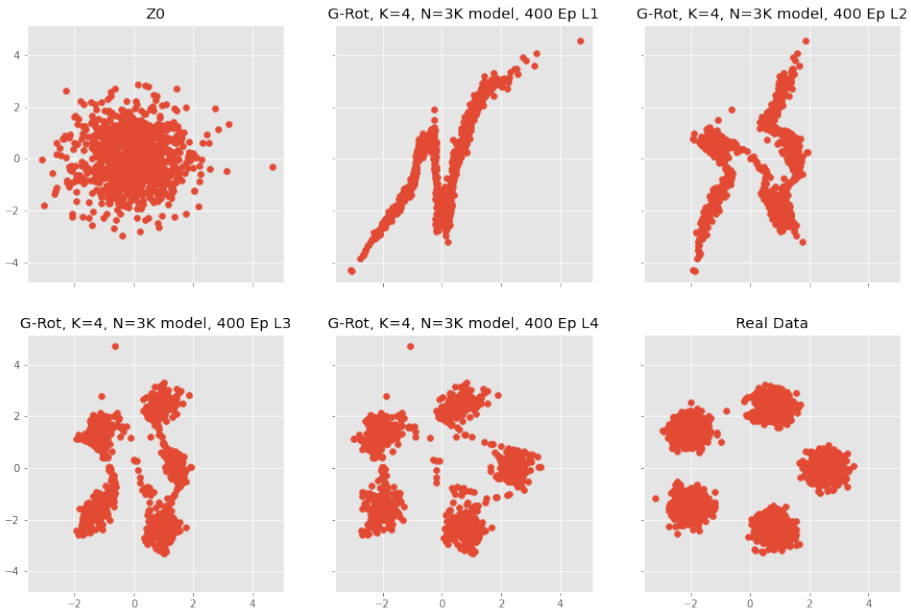
Output KDE:



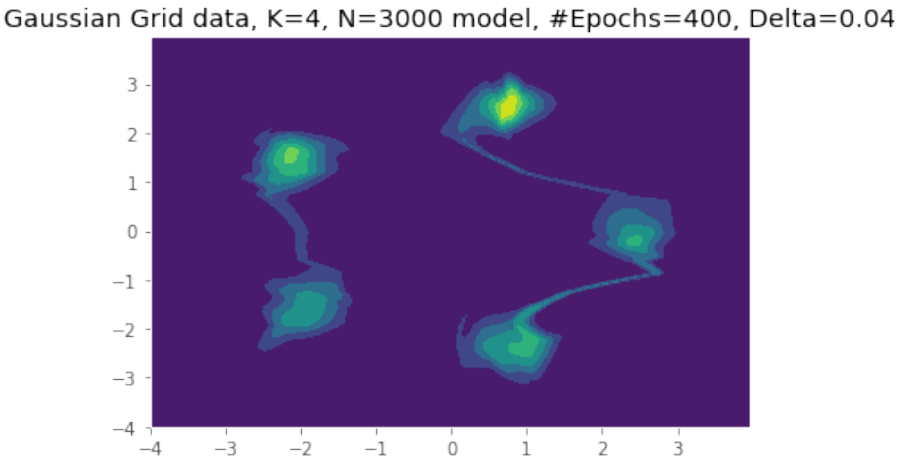
3000 samples:

Best Model achieved 2.5283 validation loss at epoch 305.

Sampling each layer:



Output KDE:



Discussion:

As we can see the number of samples didn't change the basic structure of the probability function, but only the intensity of the values in the middle of the inner gaussians.

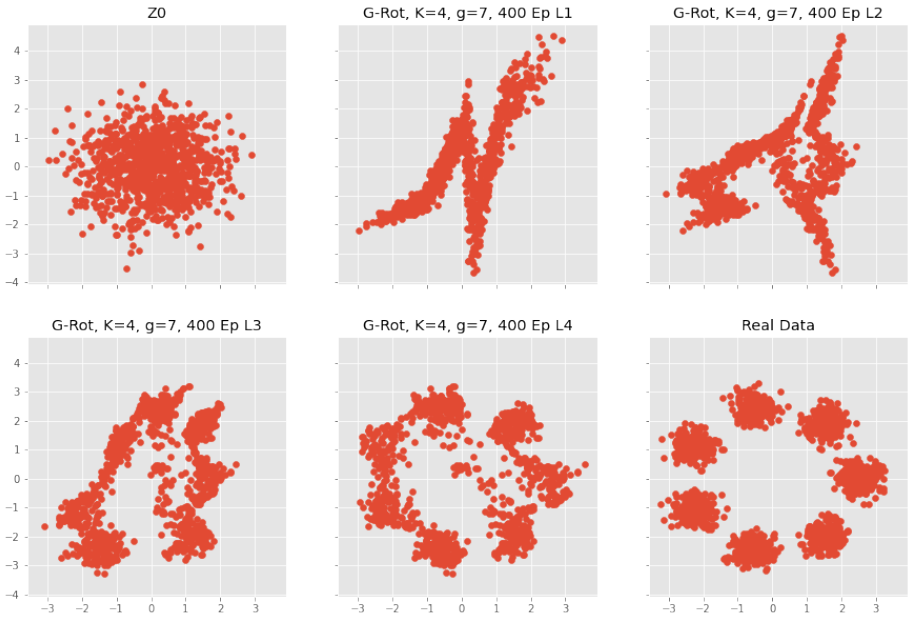
Task 12. Train two flow models with $K = 4$ layers for 250 epochs, using $N = 1500$ samples from the `'GaussiansRot'` dataset with a varying number of Gaussians `num_gaussians`$= \{3, 7\}$. Compare the resulting estimated density $p(\mathbf{x})$ (using a grid of 100×100 points) and the intermediate distributions of $N = 1000$ samples throughout the coupling layers of the model. Which distribution $p(\mathbf{x})$ is fitted better by the model? What do you conclude regarding the effect of data complexity? explain the result in your report and attach the resulting plots.

Solution:

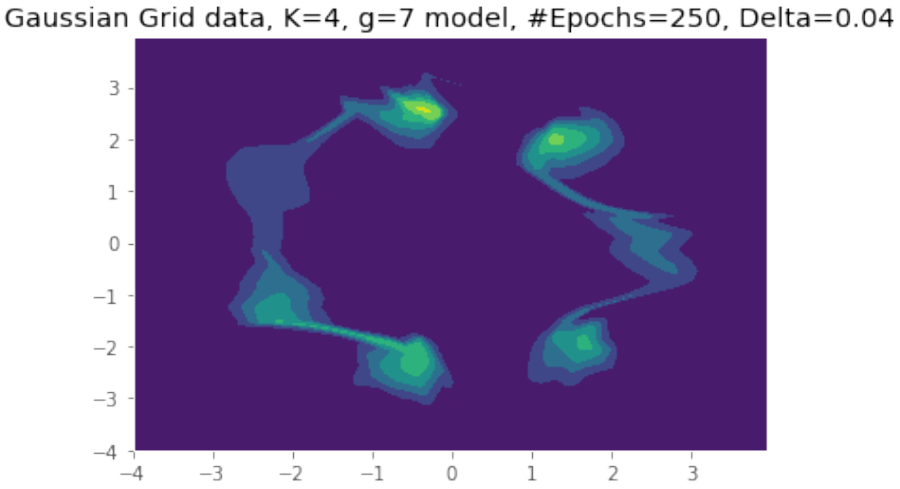
7 Gaussians:

Best Model achieved 2.6497 validation loss at epoch 248.

Sampling each layer:



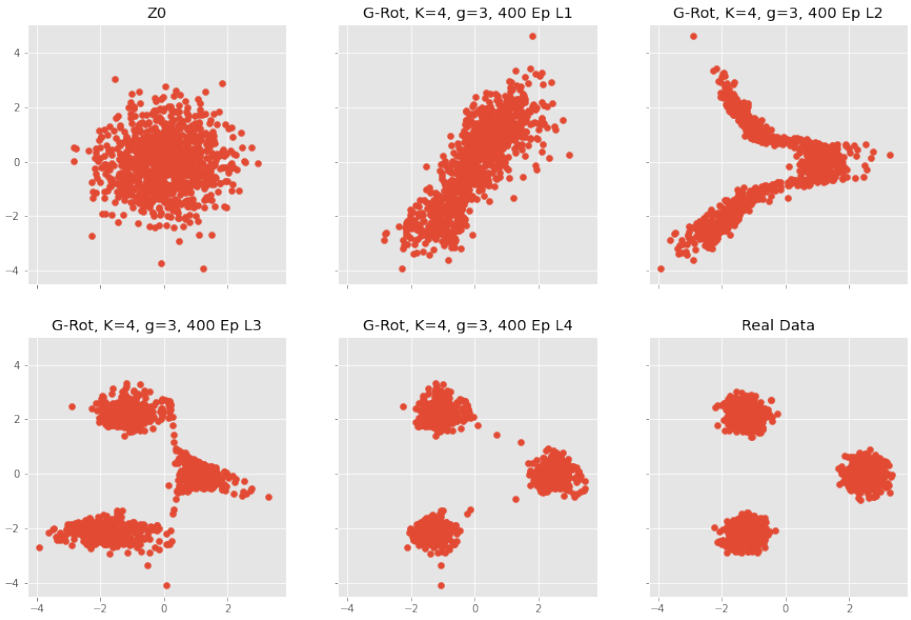
Output KDE:



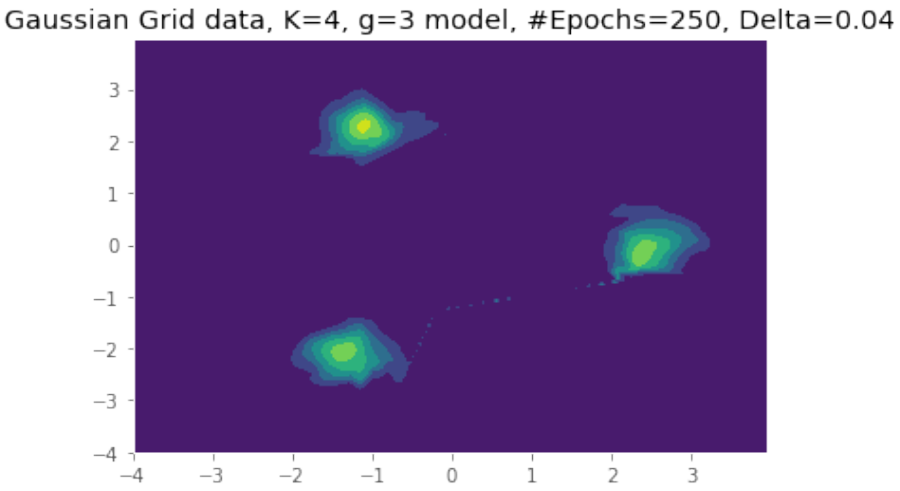
3 Gaussians:

Best Model achieved 1.6312 validation loss at epoch 243

Sampling each layer:



Output KDE:



Discussion:

The data with the 3 gaussians was learned pretty good compare to the data with 7 gaussians. As can be seen some gaussian lost some volume and their mass was drifted to the other gaussians. This tells us that with data that is more complex and sensitive to noise we need more complex model or higher number of epochs or higher number of samples.

Conclusion

In conclusion, we have seen how to implement our own normalizing flow on toy 2D datasets. However, as mentioned in the beginning of Part II, similar models with significantly more layers and additional tricks can be used to model images (e.g. Glow). The most common flow element, the coupling layer, is simple to implement, and yet effective. Normalizing flows are an interesting generative model compared to GANs, as they allow an exact likelihood estimate in continuous space, and we have the guarantee that every possible input x has a corresponding latent vector z. Recent advances in Neural ODEs allow a flow with infinite number of layers, called Continuous Normalizing Flows, whose potential is yet to fully explore.

References and Credits

[1] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). "Density estimation using Real NVP," In: 5th International Conference on Learning Representations, ICLR 2017. Link

[2] Kingma, D. P., and Dhariwal, P. (2018). "Glow: Generative Flow with Invertible 1x1 Convolutions," In: Advances in Neural Information Processing Systems, vol. 31, pp. 10215--10224. Link

[3] University of Amsterdam, Deep Learning 1, Tutorial 11. Link

[4] Technical University of Munich, Machine Learning for Graphs and Sequential Data, Generative Models. Link