Due Date: April 12, 2020

Problem 1

Q 1.1-1.5 The code for these parts is submitted to Gradescope. The code for all problems can be found here.

Q 1.6 (15 pts) Consider a latent variable model $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})dz$. The prior is define as $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}_L)$ and $\boldsymbol{z} \in \mathbb{R}^L$. Train a VAE with a latent variable of 100-dimensions ($L = 100$). Use the provided network architecture and hyperparameters described in '`vae.ipynb`'[1]. Use ADAM with a learning rate of 3×10^{-4}, and train for 20 epochs. Evaluate the model on the validation set using the **ELBO**. Marks will neither be deducted nor awarded if you do not use the given architecture. Note that for this question you have to:
Train a model to achieve an average per-instance ELBO of ≥ -102 on the validation set, and report the ELBO of your model. The ELBO on validation is written as:

$$\frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{\boldsymbol{x}_i \in \mathcal{D}_{\text{valid}}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{x}_i) \geq -102$$

Feel free to modify the above hyperparameters (except the latent variable size) to ensure it works.

At the end of training (20 epochs), final average per-instance ELBO= -100.99.

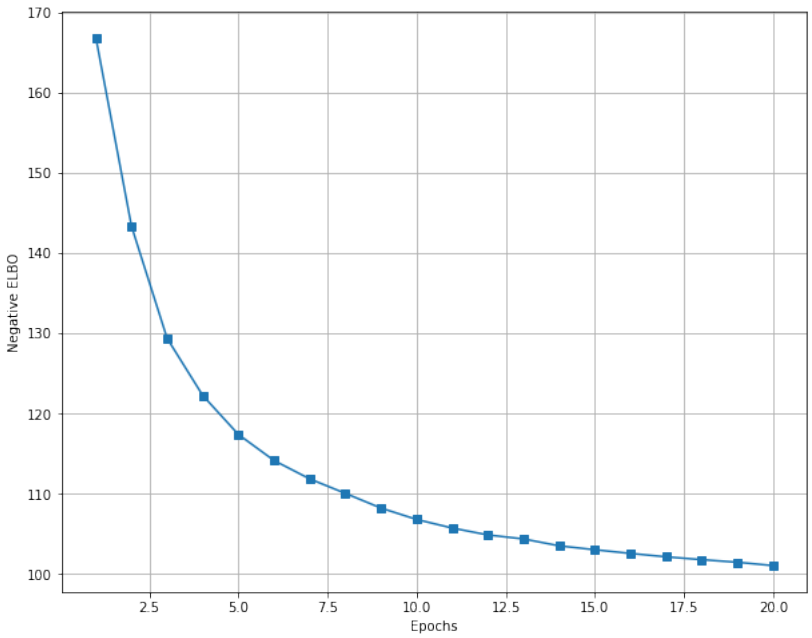


Figure 1: Average per-instance negative ELBO on the **validation set** as training progresses (20 epochs). Final average per-instance ELBO= $-100.99 \geq -102$.

[1]This file is executable in Google Colab. You can also convert `vae.ipynb` to `vae.py` using the Colab.

Q 1.7 (15 pts) Evaluate *log-likelihood* of the trained VAE models by using importance sampling, which was covered during the lecture. Use the codes described in '`vae.ipynb`'. The formula is reproduced here with additional details:

$$\log p(\boldsymbol{x} = \boldsymbol{x}_i) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p_\theta(\boldsymbol{x} = \boldsymbol{x}_i | \boldsymbol{z}_i^{(k)})\ p(\boldsymbol{z} = \boldsymbol{z}_i^{(k)})}{q_\phi(\boldsymbol{z} = \boldsymbol{z}_i^{(k)} | \boldsymbol{x}_i)}; \quad \text{for all } k\text{: } \boldsymbol{z}_i^{(k)} \sim q_\phi(\boldsymbol{z} | \boldsymbol{x}_i)$$

and $\boldsymbol{x}_i \in \mathcal{D}$.
Report your evaluations of the trained model on the test set using the log-likelihood estimate ($\frac{1}{N}\sum_{i=1}^{N} \log p(\boldsymbol{x}_i)$), where N is the size of the test dataset. Use $K = 200$ as the number of importance samples, D as the dimension of the input ($D = 784$ in the case of MNIST), and $L = 100$ as the dimension of the latent variable.

The estimate of the log-likelihood of the trained model on the test set using importance sampling is $\frac{1}{N}\sum_{i=1}^{N} \log p(\boldsymbol{x}_i) = -95.31$

Problem 2

Q 2.1-2.5 The code for these parts is submitted to Gradescope. The code for all problems can be found here.

Q 2.6 (15 pts) Execute the jupyter notebook to train a dirac generator. Substitute the random seed with your matricule number and use the predetermined configurations denoted by the prefix '`dirac-`'. In the report, provide with the final plot of the parameter trajectories for each case and interpret the results.

If theory and practice are aligned by this step, you are ready to proceed in training an image generator.

There are 20 configurations for the '`dirac-`' experiments, 10 of which are related to JSD loss, and 10 are for the W1 loss. '`dirac-jsd-1`' to '`dirac-jsd-5`' and '`dirac-w1-1`' to '`dirac-w1-5`' do not use exponential moving average for the parameters of the generator, but '`dirac-jsd-6`' to '`dirac-jsd-10`' and '`dirac-w1-6`' to '`dirac-w1-10`' do use it. To better visualize the effect of exponential moving average, and the effect of regularizers and their coefficients on stability, below, I have put the figures in pairs. Each pair is concerned with the same loss, regularizer, and regularization coefficient. Figures on the left, are for the case where exponential moving average is *not* used, and on the right, are the same settings where it is used (as also reflected by the captions). There are 5 pairs of figures for each of the loss types. As you go down through the pairs, loss type won't change, but the regularizer and its coefficient will. Putting them this way, we can clearly observe the stability patterns in terms of EMA, loss type, regularizer, and its coefficient. (You can find the corresponding info on the caption as well.)

Interpretation Figure 2: We saw in class and in problem 5 of the theoretical part, that JSD is not stable, and has two oscillatory modes in ψ_0, θ directions, and we're exactly observing the same behavior. We also learned that using exponential moving average of the parameters of the generator adds stability to the system, and on the right, we're seeing that the oscillation in one direction (θ) is dampened which is what we expect.

Figure 3: We saw problem 5 of the theoretical part, that adding $\mathcal{R}1$ regularization with coefficient $\gamma/2$, introduces a negative real part to eigenvalues and stabilizes the training of GAN. Again we saw (with eigenvectors) that it has two *damping* oscillatory modes in ψ_0, θ directions, but converges because of the negative real part of the eigenvalue, and we're exactly observing the same behavior. Again using exponential moving average of the parameters of the generator adds more stability to the system, and on the right, we're seeing that the oscillation in one direction (θ) is dampened which is what we expect and speeds up the convergence by preventing the long circular motion in the parameter space.

Figure 4: Since the negative real part of the eigenvalue has become even larger, damping effect becomes more severe and essentially removes most of the oscillation. Using exponential moving average makes convergence even easier and faster.

Figure 5: GP has a problem which is forcing the gradient to be close to 1 even close to the optimum, and as we saw in class, it fails to converge, and here we also see the divergence from the solution and oscillation around it. In fact we first approach it but later we'll get away from it because we force the gradient to be 1. Again, exponential moving average helps stabilizing at least in one direction, but can't avoid oscillation in the other direction.

Figure 6: Increasing the regularization coefficient will remove the intermediary oscillations, but due to same problem of forcing the gradient to be close to 1, oscillates around the optimum. Using EMA, stabilizes one direction, but doesn't solve the issue completely.

Figure 7,8,9,10,11: Very similar behaviors to that of the JSD, and the reasons are the same as above so we won't repeat them.

Loss type: JSD

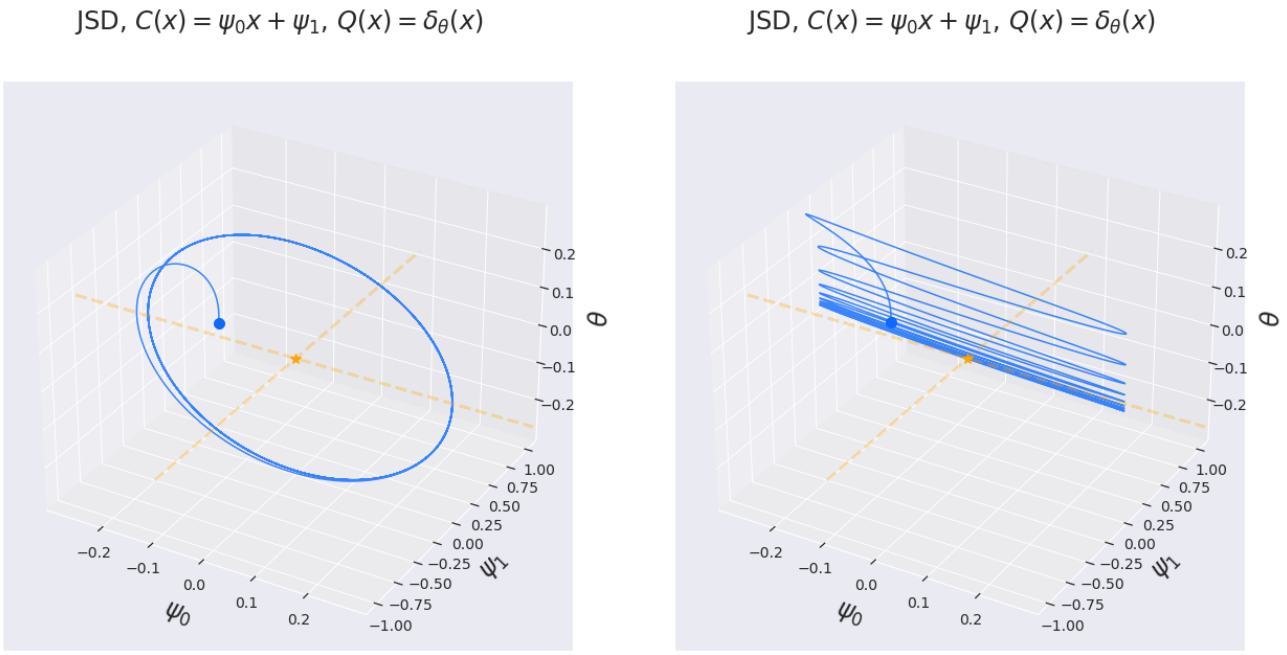


Figure 2: Parameter trajectory for loss type: **JSD**, regularizer: **None**. Left: *not* using EMA for the generator, Right: using EMA for the generator.

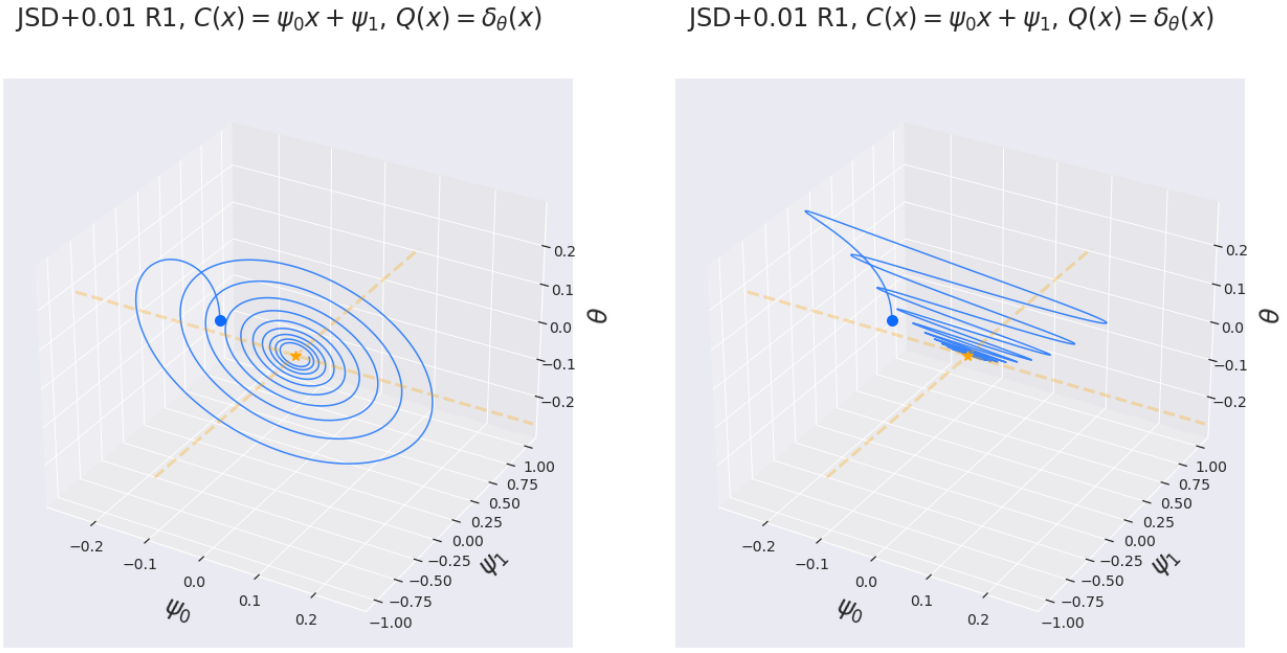


Figure 3: Parameter trajectory for loss type: **JSD**, regularizer: $\mathcal{R}1$, regularizer coefficient: 0.01. Left: *not* using EMA for the generator, Right: using EMA for the generator.

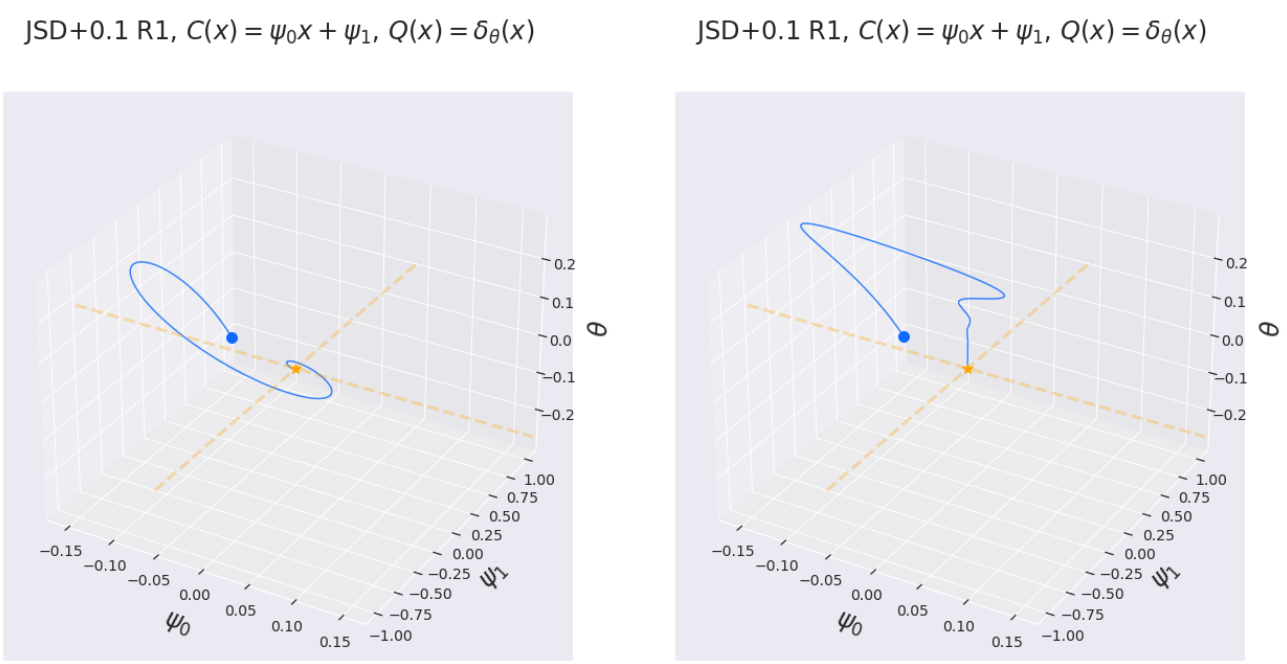


Figure 4: Parameter trajectory for loss type: **JSD**, regularizer: $\mathcal{R}1$, regularizer coefficient: 0.1. Left: *not* using EMA for the generator, Right: using EMA for the generator.

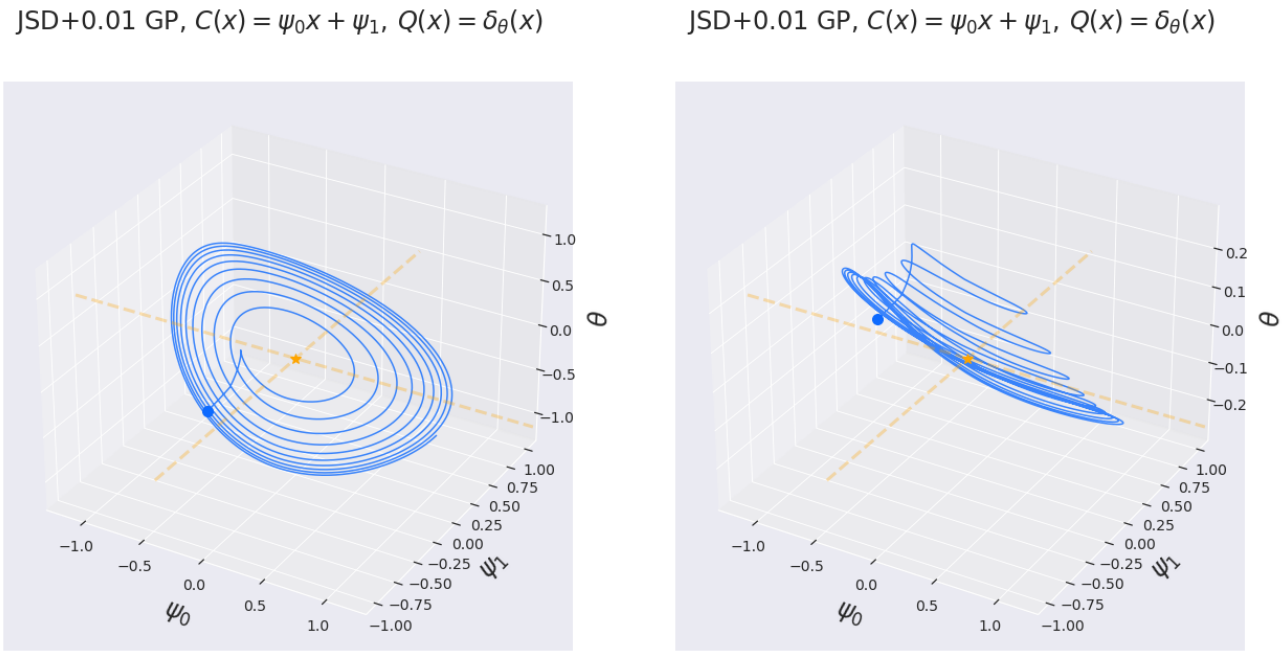


Figure 5: Parameter trajectory for loss type: **JSD**, regularizer: **GP**, regularizer coefficient: 0.01. Left: *not* using EMA for the generator, Right: using EMA for the generator.

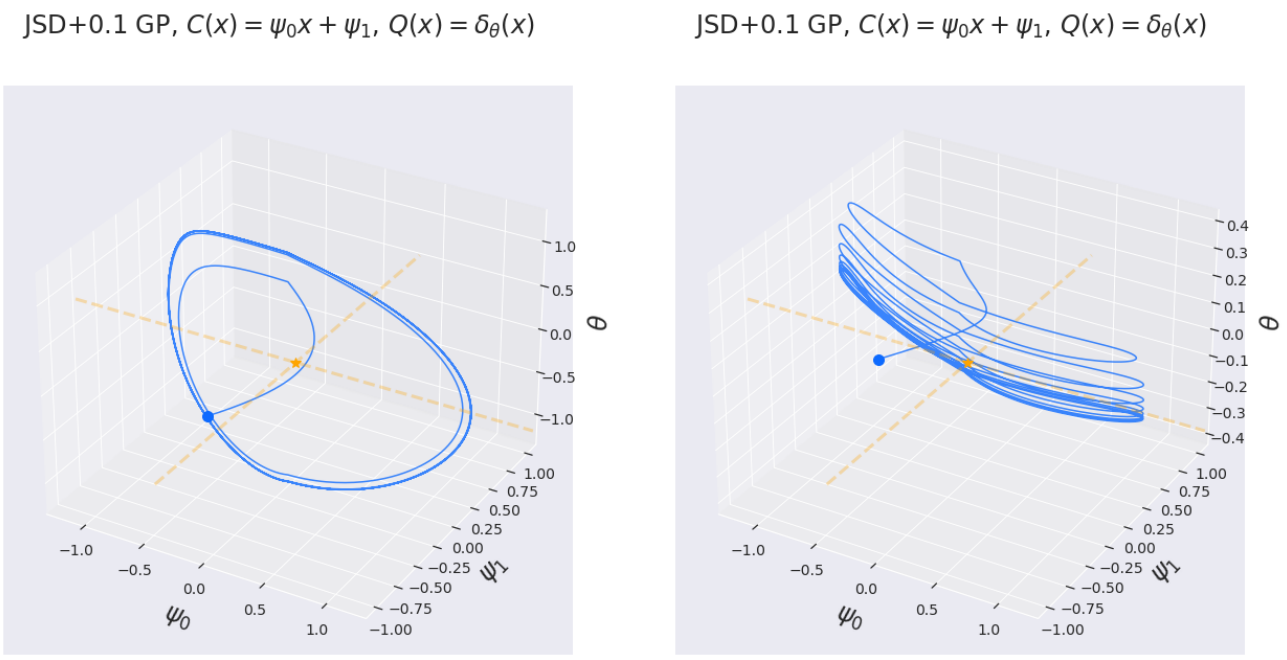


Figure 6: Parameter trajectory for loss type: **JSD**, regularizer: **GP**, regularizer coefficient: 0.1. Left: *not* using EMA for the generator, Right: using EMA for the generator.

Loss type: W1

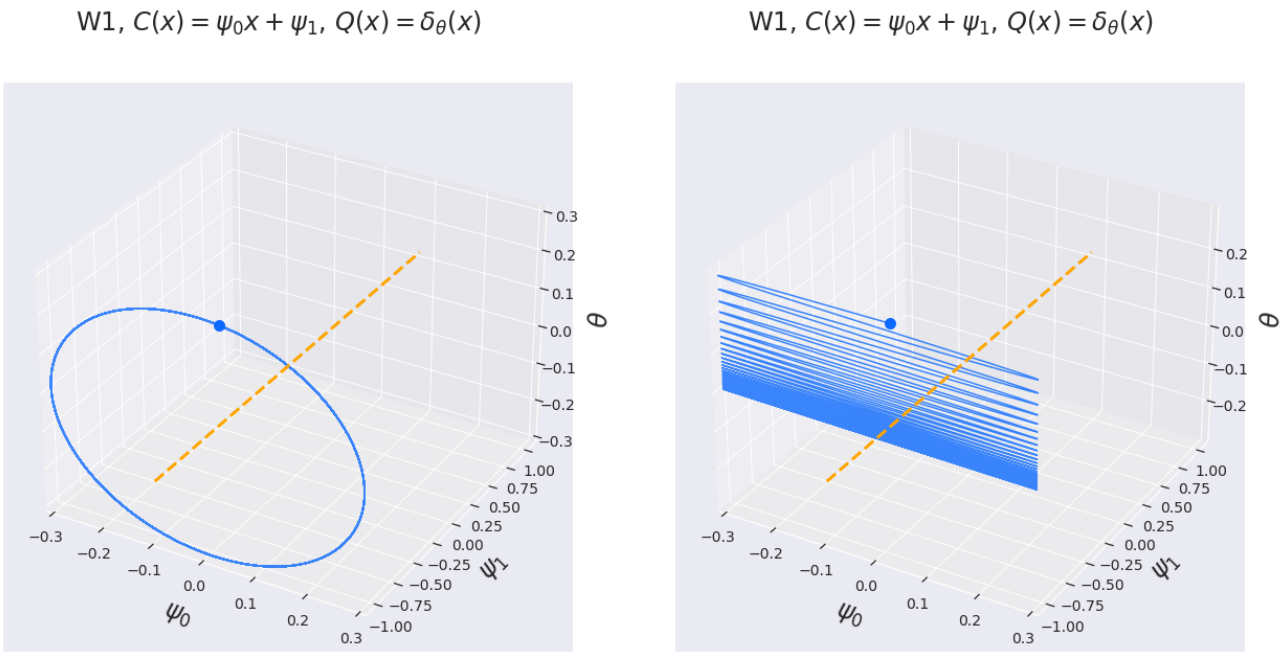


Figure 7: Parameter trajectory for loss type: **W1**, regularizer: **None**. Left: *not* using EMA for the generator, Right: using EMA for the generator.

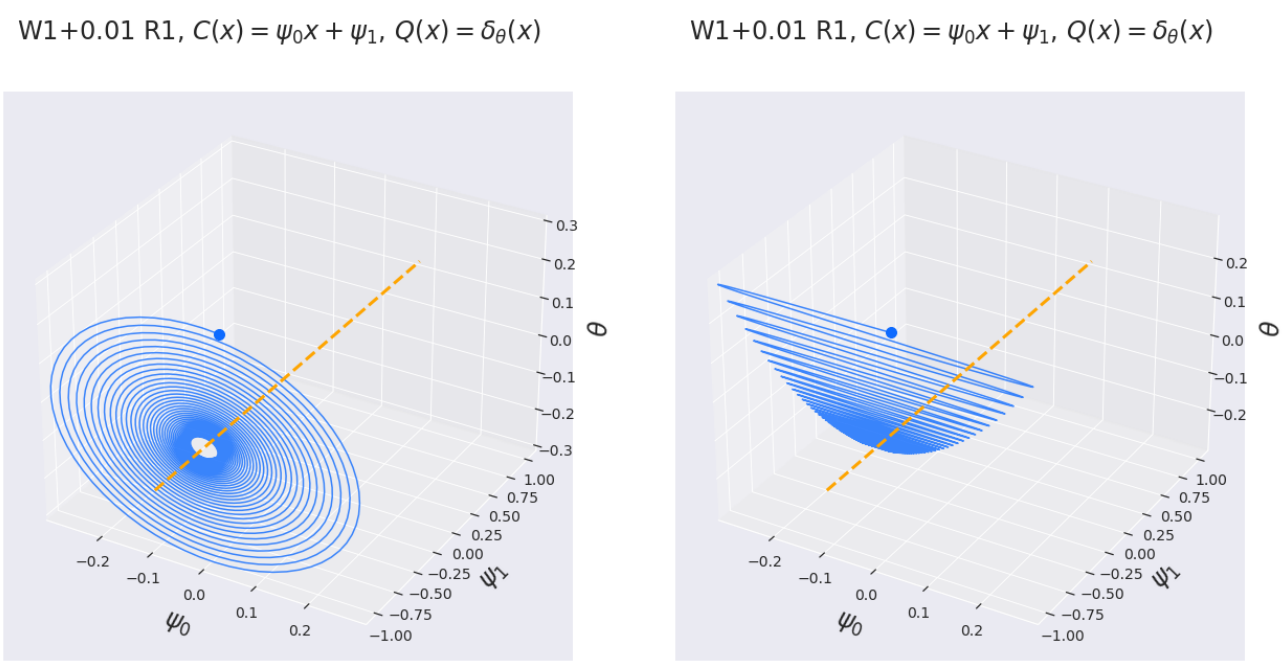


Figure 8: Parameter trajectory for loss type: **W1**, regularizer: $\mathcal{R}1$, regularizer coefficient: 0.01. Left: *not* using EMA for the generator, Right: using EMA for the generator.

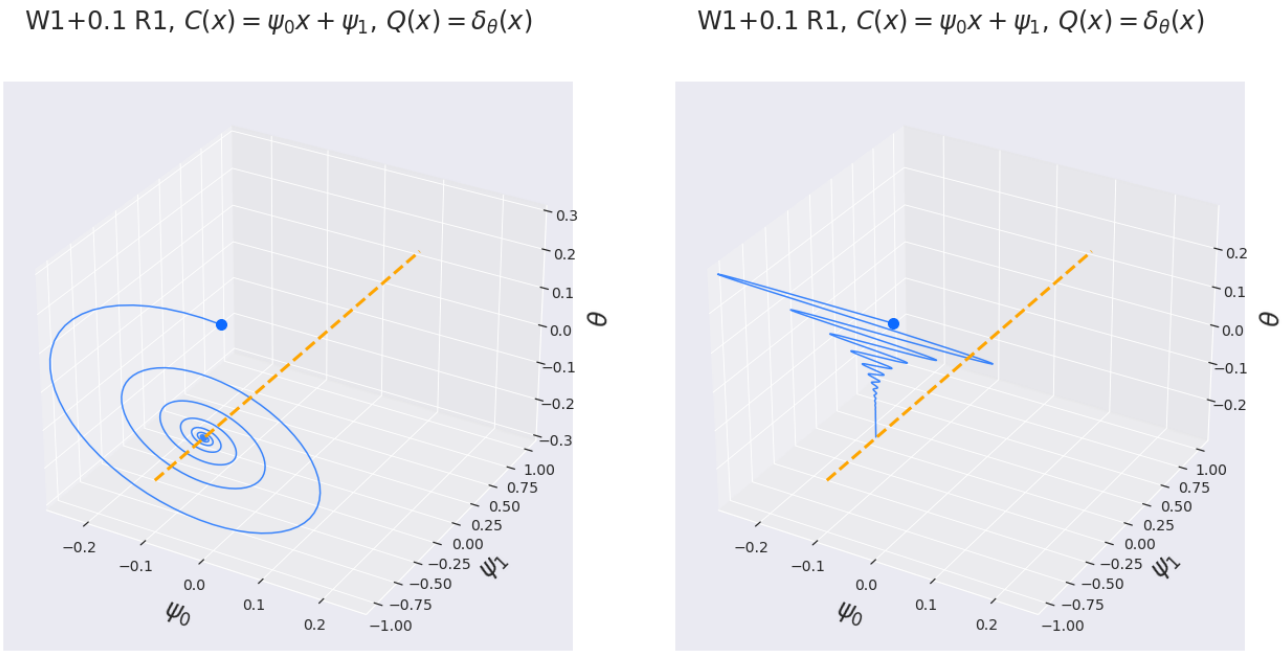


Figure 9: Parameter trajectory for loss type: **W1**, regularizer: $\mathcal{R}1$, regularizer coefficient: 0.1. Left: *not* using EMA for the generator, Right: using EMA for the generator.

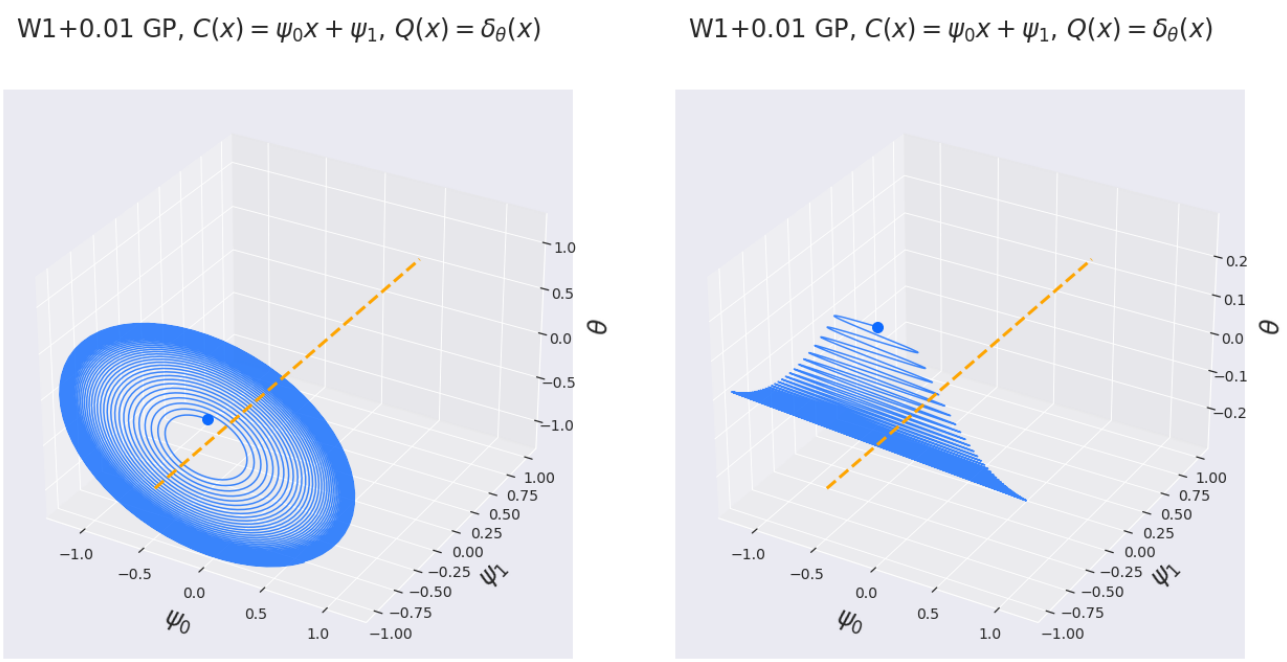


Figure 10: Parameter trajectory for loss type: **W1**, regularizer: **GP**, regularizer coefficient: 0.01. Left: *not* using EMA for the generator, Right: using EMA for the generator.

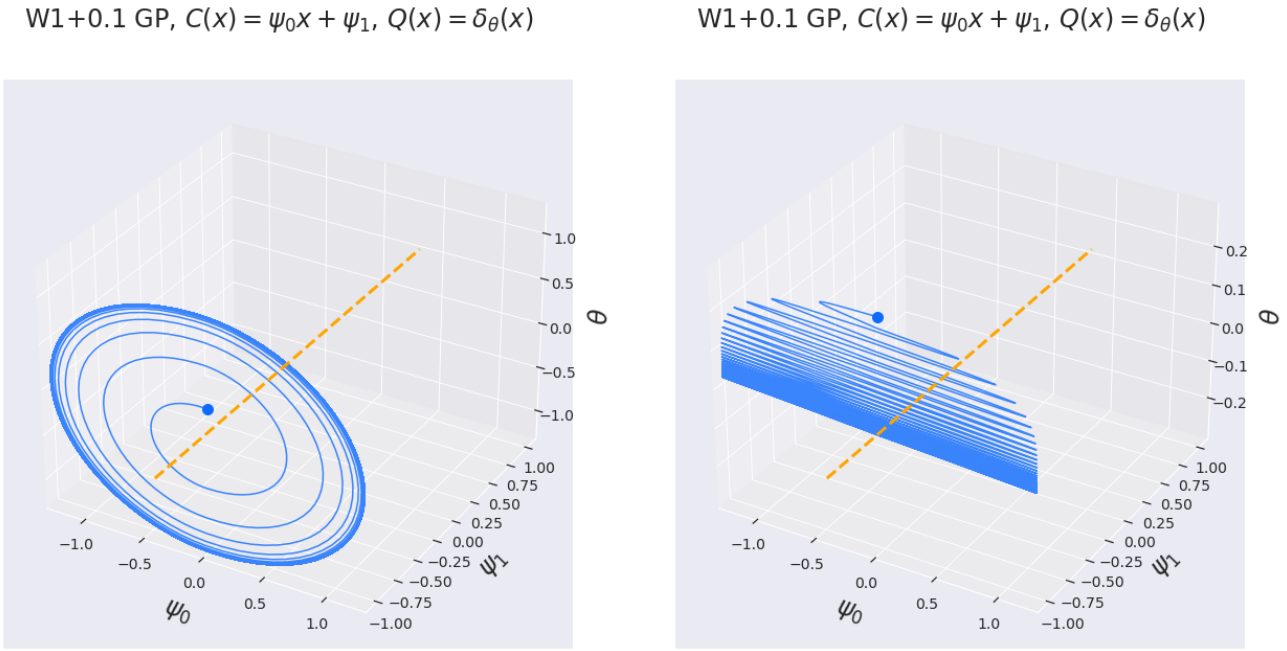


Figure 11: Parameter trajectory for loss type: **W1**, regularizer: **GP**, regularizer coefficient: 0.1. Left: *not* using EMA for the generator, Right: using EMA for the generator.

Q 2.7 (25 pts) We will use CIFAR10 dataset with predetermined generator and critic architectures, as well as the hyperparameter configurations denoted by the prefix '`cifar10-`'. In the report, you are requested to provide with final samples, Inception Score and Fréchet Inception Distance (FID)

and compare between the 5 configurations. Notice that in this section, we have included spectral normalization as a way to constrain critic's Lipschitz constant. Also, provide with a plot over training steps for the approximated metric (JSD or W1) and FID for each run. Contrast the two quantities into the same plot. What do you observe? Which metric is more indicative of training's progress?[2]

Final samples generated are presented in figures 12,13,14,15,16,17 (Note that I have run an extra experiment to explore whether increasing reg. coefficient to 10 could make W1+$\mathcal{R}1$ even better). For final and best Inception Score (IS) and FID during the training refer to 1.
In table 1 I have put the best two models in bold fonts. So JSD with Spectral Normalization and W1 with $\mathcal{R}1$ perform the best as is also reflected by the very high quality images generated by them (13,16). Looking at their training curves (FID and loss) in figures 19,22, we also observe a stable training in terms of FID. 22 shows a slight increase in FID, so one might be cautious about such behaviors. Also we see that JSD loss is not at all a good indicative of the training progress, as it's neither low, nor monotonic, whereas the quality of the generated images is great (and at the same time JSD is increasing!). **So the better measure for the rest of the argument is FID.**
From Dirac experiments we already know that JSD alone without any regularization is unstable, and low IS and high FID 18 along with the low quality image it has generated 12, clearly validate that hypothesis.
JSD+$\mathcal{R}1$ is stable and trains well 14,20, and generates well. The only point is that it's slightly increasing its FID 20, and we can see from table 1 that its best IS and FID are the best overall, but because it is slightly getting away from that optimum, in the end the result isn't as perfect. So we should stop training before it starts to increase, in that case we'll achieve the best performance. Also it seems that Spectral Normalization works better than $\mathcal{R}1$ for JSD 14,20.
It was unexpected that W1 didn't work with Spectral Normalization and generated bad images 15. From its training curve 21, it is clear that FID is increasing and that can explain the low quality image generation.
Additionally, increasing the reg. coefficient to 10 for W1+$\mathcal{R}1$ has degraded the performance which is not totally surprising as it happens when regularization is too strict (In problem 1 of the theoretical part I explored the extreme cases). In fact we see that 23 FID first drops, but then significantly increases resulting in poor images 17.

Experiment name	Loss	Regularization	Reg. Coeff.	IS^f	FID^f	IS^b	FID^b
'cifar10-jsd-1'	JSD	None	-	3.285	113.982	3.541	93.984
'cifar10-jsd-2'	JSD	SN	-	**5.385**	**32.941**	5.428	32.941
'cifar10-jsd-3'	JSD	$\mathcal{R}1$	1.0	4.69	48.705	5.952	25.803
'cifar10-w1-2'	W1	SN	-	1.668	207.39	3.709	85.347
'cifar10-w1-3'	W1	$\mathcal{R}1$	1.0	**5.414**	**33.621**	5.943	25.676
'cifar10-w1-3'	W1	$\mathcal{R}1$	10.0	3.981	59.469	5.443	31.545

Table 1: Higher Inception Score (IS) is better, whereas lower Frechet inception distance (FID) is better. Best two settings are in bold format. Superscript f means the final value of the measure after 100 000 steps, and superscript b is the best value of the measure during training (max for IS, min for FID). A large discrepancy between the best and final value of a measure is a signal of training instability that caused the system to get out of its optimum.

[2]Extra: Why? Find the answer at: Towards Principled Methods for Training Generative Adversarial Networks

Figure 12: Generated image after **100 000** steps, loss: **JSD**, regularizer: **None**. Experiment name: '`cifar10-jsd-1`'

Figure 13: Generated image after **100 000** steps, loss: **JSD**, regularizer: **Spectral Normalization**. Experiment name: '`cifar10-jsd-2`'

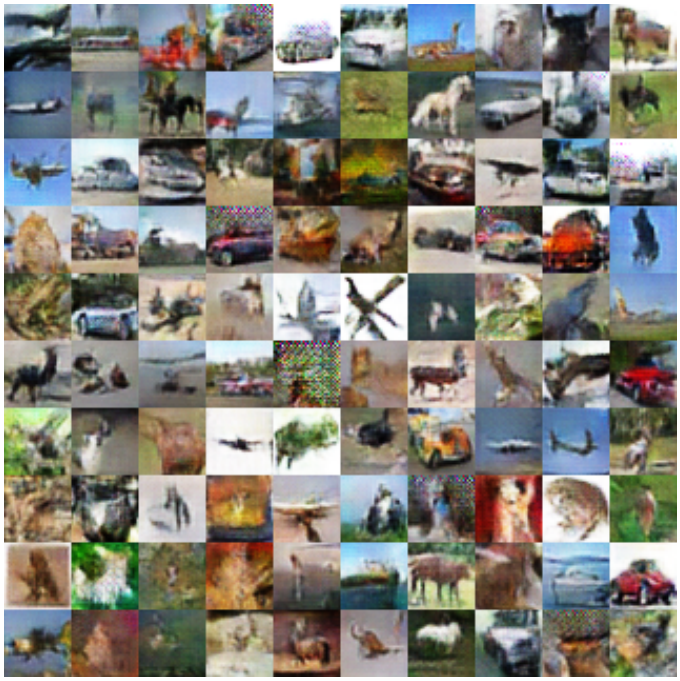


Figure 14: Generated image after **100 000** steps, loss: **JSD**, regularizer: $\mathcal{R}1$, regularization coefficient: 1, Experiment name: '`cifar10-jsd-3`'

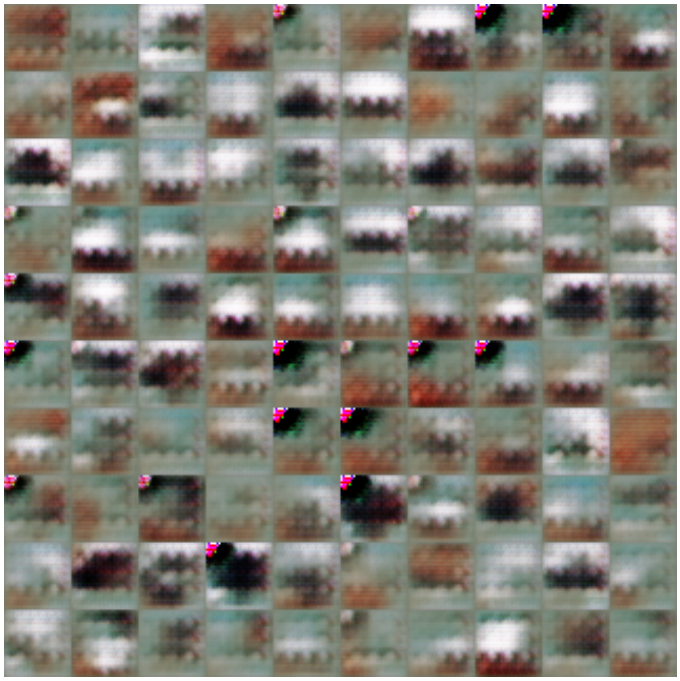


Figure 15: Generated image after **100 000** steps, loss: **W1**, regularizer: **Spectral Normalization**. Experiment name: '`cifar10-w1-2`'

Figure 16: Generated image after **100 000** steps, loss: **W1**, regularizer: $\mathcal{R}1$, regularization coefficient: 1. Experiment name: '`cifar10-w1-3`'

Figure 17: Generated image after **100 000** steps, loss: **W1**, regularizer: $\mathcal{R}1$, regularization coefficient: 10. Experiment name: '`cifar10-w1-3`'

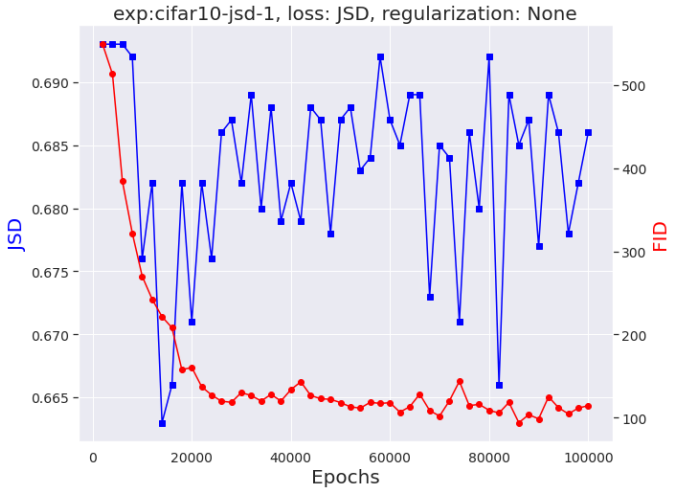


Figure 18: FID and JSD over training steps, loss: **JSD**, regularizer: **None**. Experiment name: '`cifar10-jsd-1`'

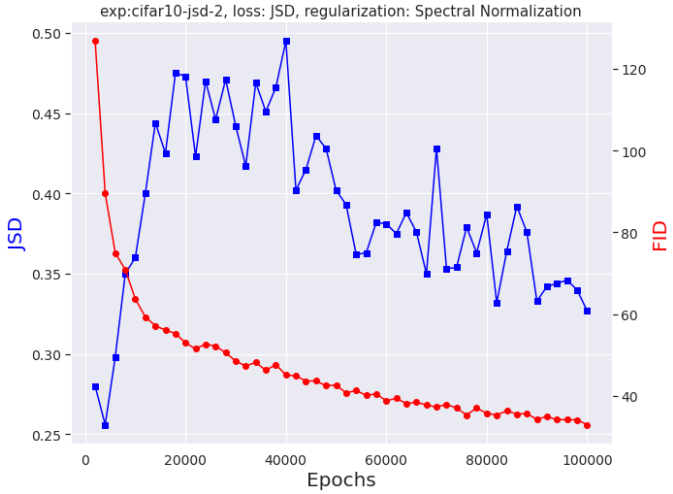


Figure 19: FID and JSD over training steps, loss: **JSD**, regularizer: **Spectral Normalization**. Experiment name: '`cifar10-jsd-2`'

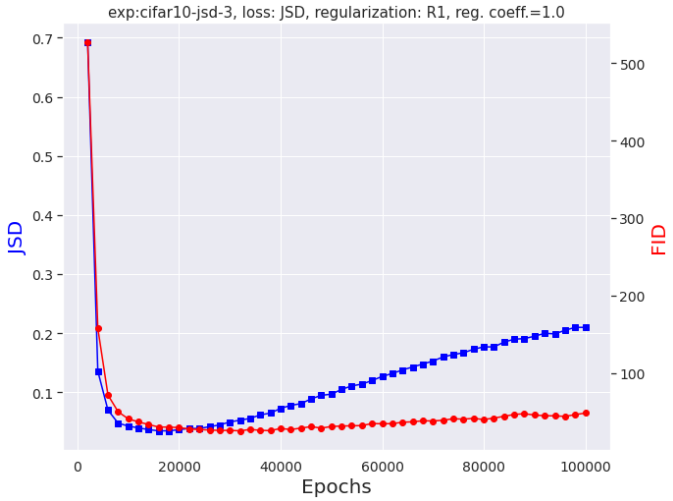


Figure 20: FID and JSD over training steps, loss: **JSD**, regularizer: $\mathcal{R}1$, regularization coefficient: 1, Experiment name: '`cifar10-jsd-3`'

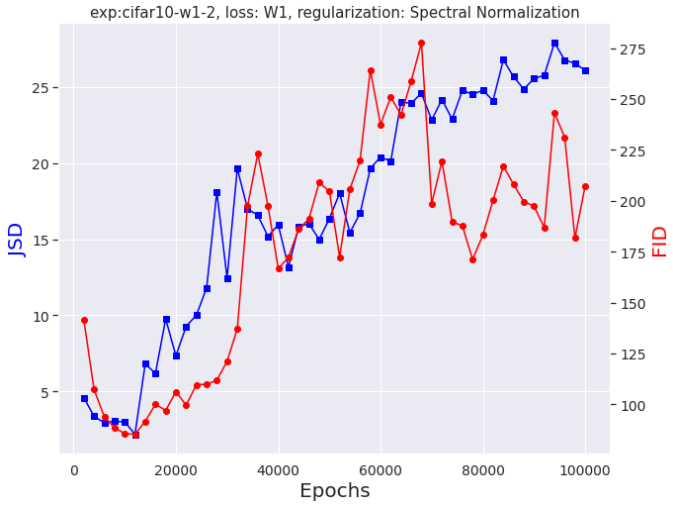


Figure 21: FID and W1 over training steps, loss: **W1**, regularizer: **Spectral Normalization**. Experiment name: '`cifar10-w1-2`'

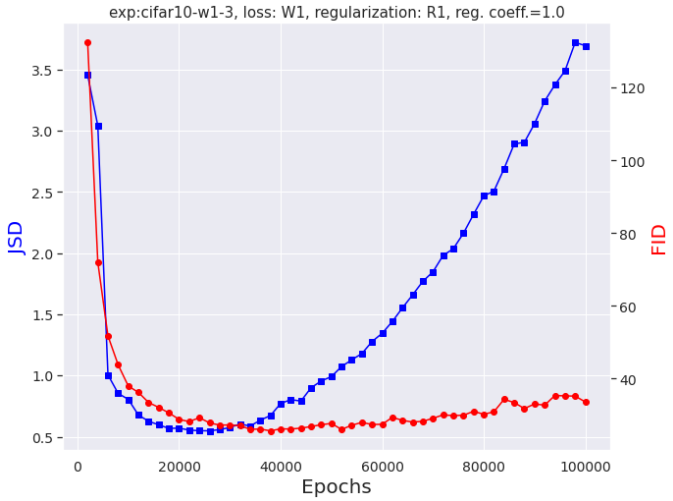


Figure 22: FID and W1 over training steps, loss: **W1**, regularizer: $\mathcal{R}1$, regularization coefficient: 1. Experiment name: '`cifar10-w1-3`'

Figure 23: FID and W1 over training steps, loss: **W1**, regularizer: $\mathcal{R}1$, regularization coefficient: 10. Experiment name: '`cifar10-w1-3`'

Problem 3

Normalizing flows are expressive invertible transformations of probability distributions. In this exercise, you will implement one instance of a normalizing flow: a planar flow. Most of the code is already given, you only need to complete some functions. Likewise, to train the model, you should use the default hyperparameters that are given. For questions noted as `unittest`, no public unit tests are given, but you have to make sure to follow the instructions in the docstrings of each function. A planar flow is a composition of planar transformation, each defined as:

$$f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u}h(\boldsymbol{w}^\top \boldsymbol{z} + b) \tag{1}$$

where $\boldsymbol{w} \in \mathbb{R}^d$, $\boldsymbol{u} \in \mathbb{R}^d$, $b \in \mathbb{R}$, and h is a smooth non-linear function.

Q 3.1, 3.4, 3.6, 3.8 The code for these parts is submitted to Gradescope. The code for all problems can be found here.

Q3.2 (2 pts) Test your function by applying it to samples from a $N(\mathbf{0}, \boldsymbol{I})$ for different value of $\boldsymbol{u}$, $\boldsymbol{w}$, b. Describe what the transformation correspond to and add plots to show the effect of different transformations.

For figures related to this problem, please refer to the last 4 pages of this document. Essentially, a planar transformation has a residual term plus a contraction or expansion (depending on the value of $\boldsymbol{u}$). If bias term b is zero, then the symmetry in the original space is preserved, but otherwise, the densities will become unbalanced as layers keep contracting/expanding with a bias. What is the direction of contraction/expansion? It's perpendicular to the hyperplane (line in a 2-D space) defined by $\boldsymbol{w}^\top \boldsymbol{z} + b = 0$. What is the effect of $\boldsymbol{u}$? If $\|\boldsymbol{u}\| < 1$, then we have contractive transformations, and if $\|\boldsymbol{u}\| > 1$, we have an expanding transformation which puts the distributions further apart as the number of layers in the flow increases. To assess these behaviors, I have put the figures for the following settings:

- $\boldsymbol{w} \sim (0, 0.1), \mu_{\boldsymbol{u}} \in \{0, 2, 5\}, \sigma^2_{\boldsymbol{u}} \in \{0.1, 0.5, 1\}$
- $\boldsymbol{w} \sim (0, 0.5), \mu_{\boldsymbol{u}} \in \{0, 2, 5\}, \sigma^2_{\boldsymbol{u}} \in \{0.1, 0.5, 1\}$
- $\boldsymbol{w} \sim (0, 1), \mu_{\boldsymbol{u}} \in \{0, 2, 5\}, \sigma^2_{\boldsymbol{u}} \in \{0.1, 0.5, 1\}$
- $\boldsymbol{w} \sim (5, 0.1), \mu_{\boldsymbol{u}} \in \{0, 2, 5\}, \sigma^2_{\boldsymbol{u}} \in \{0.1, 0.5, 1\}$

So for each fixed distribution of $\boldsymbol{w}$, we explore different distributions of $\boldsymbol{u}$ and different values of the bias. For $\boldsymbol{w}$ we change its variance and mean in a range of values to see the patterns.
In all cases you can observe, if $\boldsymbol{u}$ is small (or its mean is zero), there's not much of a transformation and things remain more or less the same. If $\boldsymbol{w}, b$ are both small, then the derivative of tanh would be close to 1, and we will have strong contraction/expansions as also reflected by the figures. (distribution is converted through a logdet derived in the next problem). When bias is too large or too negative, then h' will be pushed to its flat areas and effectively causing the logdets to vanish, and thus we see little transformations in such cases and the only driving force would be $\boldsymbol{w}^\top \boldsymbol{u}$. So to see more interesting results, one should focus on the cases where b is close to zero. In those settings, again large $\boldsymbol{u}$ causes strong contraction/expansion, and if the variance is higher, we get a thicker strip (evident in figures).
In short, the norm of $\boldsymbol{u}$ determine the strength of expansion/contraction, $\boldsymbol{w}$ determines the direction of expansion/contraction, and b changes the balance towards one side of the expansion.

Q 3.3 (4 pts) Calculate the log determinant of the Jacobian of a planar transformation using the matrix determinant lemma. What is the complexity (using the big $\mathcal{O}$ notation) of evaluating it?
Let's first calculate the Jacobian:

$$\frac{\partial f}{\partial \boldsymbol{z}} = \frac{\partial}{\partial \boldsymbol{z}} \boldsymbol{z} + \frac{\partial}{\partial \boldsymbol{z}} \boldsymbol{u} h(\boldsymbol{w}^\top \boldsymbol{z} + b) = \boldsymbol{I} + \boldsymbol{u} \frac{\partial}{\partial \boldsymbol{z}} h(\boldsymbol{w}^\top \boldsymbol{z} + b)$$

But since $h(x) = \tanh(x), h'(x) = 1 - \tanh^2(x)$, then using the chain rule we have:

$$\frac{\partial f}{\partial \boldsymbol{z}} = \boldsymbol{I} + \boldsymbol{u}\frac{\partial}{\partial \boldsymbol{z}} h(\boldsymbol{w}^\top \boldsymbol{z} + b) = \boldsymbol{I} + \boldsymbol{u}h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\frac{\partial}{\partial \boldsymbol{z}}(\boldsymbol{w}^\top \boldsymbol{z} + b) = \boldsymbol{I} + \boldsymbol{u}h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top$$

Note that $h'(\boldsymbol{w}^\top \boldsymbol{z} + b)$ is a scalar, not a vector or matrix, now calculating the determinant of the Jacobian:

$$\det \frac{\partial f}{\partial \boldsymbol{z}} = \det(\boldsymbol{I} + \boldsymbol{u}h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top)$$

Using the matrix determinant lemma:

$$\begin{aligned}\det \frac{\partial f}{\partial \boldsymbol{z}} &= \det(\boldsymbol{I} + \boldsymbol{u}h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top) = \det(\boldsymbol{I} + h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{u}^\top \boldsymbol{w}) \det(\boldsymbol{I}^{-1}) \\ &= \det(\boldsymbol{I} + h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top \boldsymbol{u}) = 1 + h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top \boldsymbol{u}\end{aligned}$$

Now taking the logdet:

$$\log|\det \frac{\partial f}{\partial \boldsymbol{z}}| = \log|1 + h'(\boldsymbol{w}^\top \boldsymbol{z} + b)\boldsymbol{w}^\top \boldsymbol{u}|$$

The complexity of calculating $\boldsymbol{w}^\top \boldsymbol{u}$ is $O(d)$ because it's a dot product of vectors of length d, same goes for the complexity of $\boldsymbol{w}^\top \boldsymbol{z} + b$, and since $y = \boldsymbol{w}^\top \boldsymbol{z} + b$ is a scalar, the complexity of $h'(y) = 1 - \tanh^2(y)$ is $O(1)$, and thus finally the complexity of calculating the logdet of the Jacobian would be only $O(d)$. Note that the kep property here was the matrix determinant lemma, otherwise we had to take the determinant of a $d \times d$ matrix and we had to also compute matrix multiplications of size $d \times d$, which would induce a complexity of $O(d^3)$ *per layer* of the flow. So with this lemma we have significantly reduced the complexity.

Q 3.5 (2 pts) Let $U_0 \sim N(\boldsymbol{0}, \boldsymbol{I})$ be the base distribution. Express the induced log density $\ln q_k(\boldsymbol{u}_k)$ obtained by applying a flow constituted of k planar transformation, where $\boldsymbol{u}_k = f_k \circ f_{k-1} \circ \cdots \circ f_1(\boldsymbol{u}_0)$. What is the complexity of evaluating the log determinant of the normalizing flow?

Using the same relation from the previous problem and density conversion from class slides, and the fact that $|\det \frac{\partial f^{-1}}{\partial y}| = (|\det \frac{\partial f}{\partial x}|)^{-1}$, we have:

$$\begin{aligned}\ln q_k(\boldsymbol{u}_k) &= \ln q_{k-1}(\boldsymbol{u}_{k-1}) + \ln|\det \frac{\partial f_k^{-1}}{\partial \boldsymbol{u}_k}| = \ln q_{k-1}(\boldsymbol{u}_{k-1}) - \ln|\det \frac{\partial f_k}{\partial \boldsymbol{u}_{k-1}}| \\ &= \ln q_{k-2}(\boldsymbol{u}_{k-2}) - \ln|\det \frac{\partial f_k}{\partial \boldsymbol{u}_{k-1}}| - \ln|\det \frac{\partial f_{k-1}}{\partial \boldsymbol{u}_{k-2}}| = \dots \\ &= \ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \ln|\det \frac{\partial f_m}{\partial \boldsymbol{u}_{m-1}}| = \ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \ln|1 + h'(\boldsymbol{w}_m^\top \boldsymbol{z}_{m-1} + b)\boldsymbol{w}_m^\top \boldsymbol{u}_m|\end{aligned}$$

The complexity of calculating $\ln q_0(\boldsymbol{u}_0)$ is $O(1)$, and computing each term in the summation takes $O(d)$ (from Q 3.3), thus since there are k layers in the flow, the complexity of evaluating the logdet of the would be $O(1 + kd) = O(kd)$

Q 3.7 (4 pts) Now that the flow is completed, train the model on the dataset '`data_arcs`' and '`data_sine`' with $k = \{2, 8, 32\}$ for 20000 iterations using the function '`launch_experiments_from_samples`' from '`train.py`'. For each setting of k and dataset, plot the learned density (these plots are automatically generated by the code) and report the negative log likelihood. Briefly comment the effect of k.

It is generally clear that increasing the number of layers in the flow results in a more flexible density estimator that can better capture the intrinsic structure. The first dataset '`density_arsc`' is a bit simpler, so going from 8 to 32 layers hasn't yielded much more complexity, most of the data distribution had been captured by 8 layers. In the other dataset '`density_sine`' however, capturing the density is harder and we can clearly see the advantage of introducing more layers to capture more parts of the sine more accurately.
Negative log-likelihood and the number of layers for each figure is reflected in the respective captions with nll and k. Both from the plots and negative log-likelihoods.

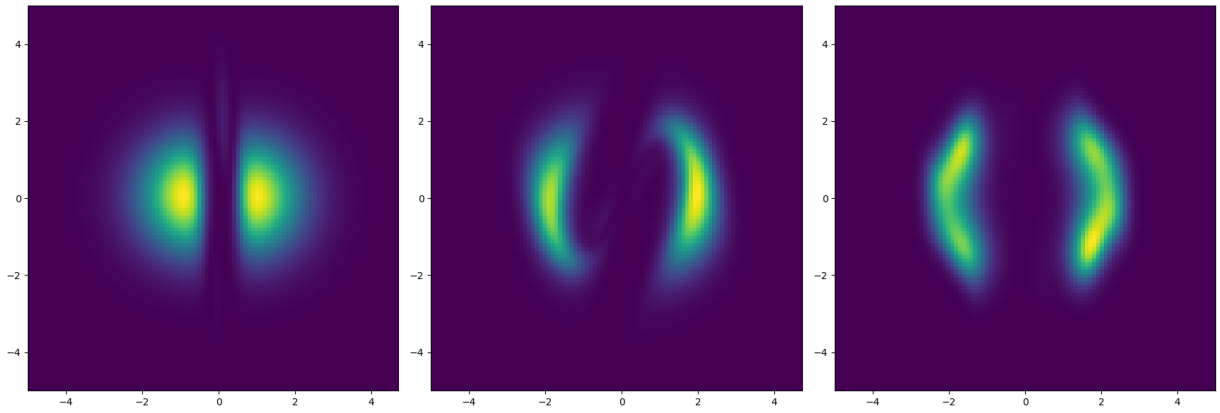


Figure 24: Learned densities from samples of the dataset '`data_arcs`'. From left to right: $\{k = 2$, nll$= 3.46\}$, $\{k = 8$, nll$= 2.73\}$, $\{k = 32$, nll$= 2.72\}$.

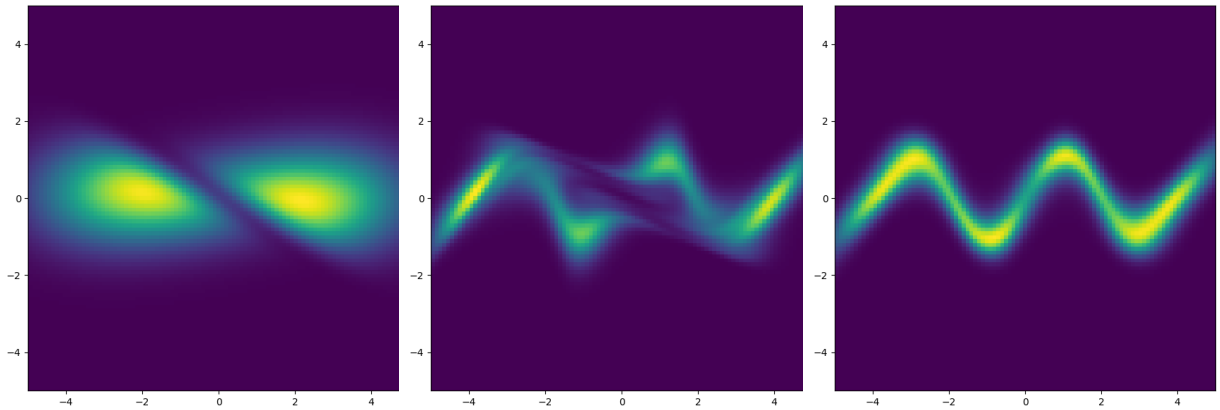


Figure 25: Learned densities from samples of the dataset '`data_sine`'. From left to right: $\{k = 2$, nll$= 3.54\}$, $\{k = 8$, nll$= 3.04\}$, $\{k = 32$, nll$= 2.91\}$.

Q 3.9 (2 pts) Train the model on the density '`density_arcs`' and '`density_sine`' with $k =$

$\{2, 8, 32\}$ for 20000 iterations using the function '`launch_experiments_from_density`' from '`train.py`'. For each setting of k, report the plots of the samples generated by the normalizing flow (these plots are automatically generated by the code).

Again a similar argument to that of Q 3.7 holds. The only difference is that here we're concerned with *generating* data, but the same principles apply here as well.
It is generally clear that increasing the number of layers in the flow results in a more flexible density estimator that can better capture the intrinsic structure and generate better samples. The first dataset '`density_arsc`' is a bit simpler, so going from 8 to 32 layers hasn't yielded much more complexity, most of the data distribution had been captured by 8 layers.
In the other dataset '`density_sine`' however, capturing the density is harder and we can clearly see the advantage of introducing more layers to capture more parts of the sine more accurately.

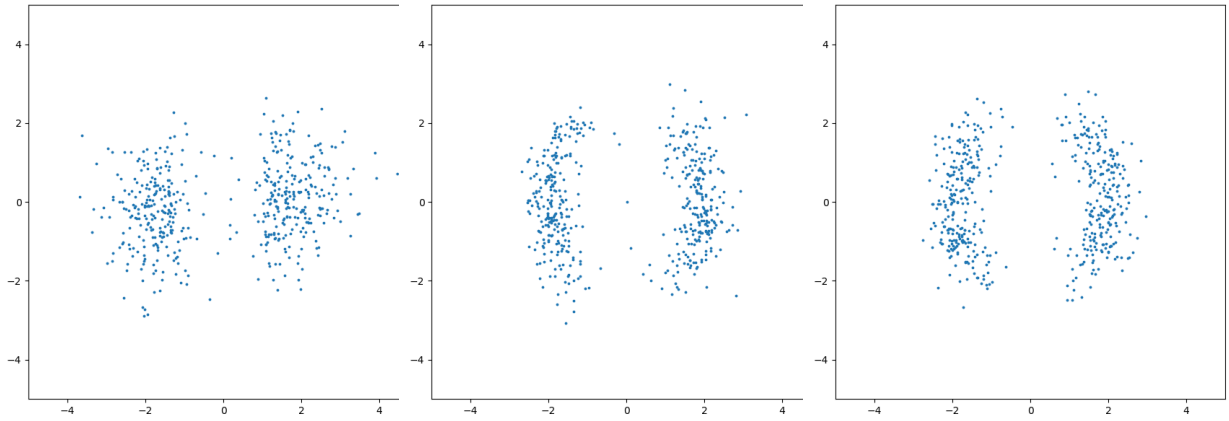


Figure 26: Samples generated by the normalizing flow from the density '`density_arcs`'. From left to right: k ={2,8,32}.

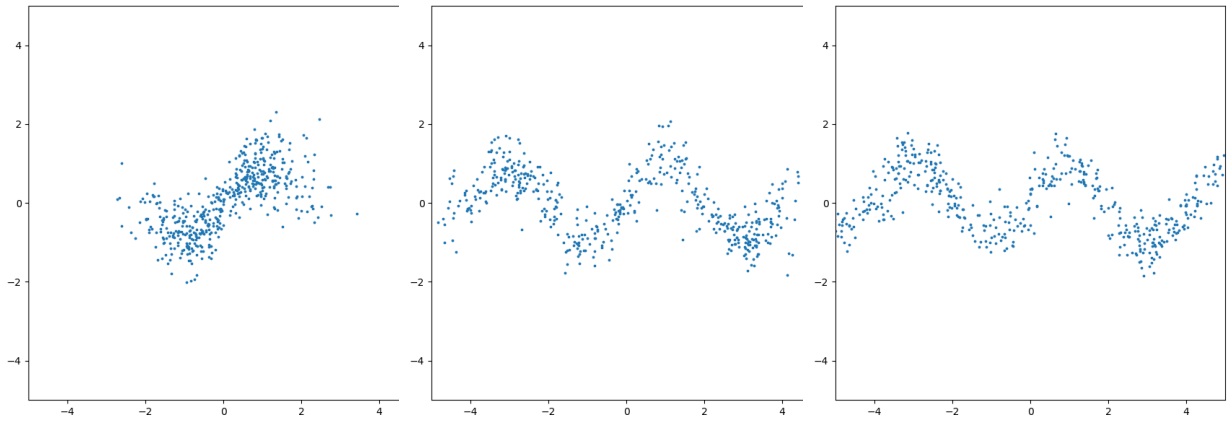


Figure 27: Samples generated by the normalizing flow from the density '`density_sine`'. From left to right: k ={2,8,32}.

Q 3.10 (4 pts) Normalizing flows can also be used as a posterior of variational autoencoders. Write the loss of a VAE using a planar flow. What is the advantage of using a normalizing flow as the posterior of a VAE?

Let's first start from the ELBO for the VAE:

$$\mathcal{L}(\theta, \phi, x) = -D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$$

Now we should replace $q_\phi(z \mid x)$ by that of a planar flow derived in Q 3.5 above, so $q_\phi(z \mid x) = q_k(\boldsymbol{u}_k)$ (also note that $z = \boldsymbol{u}_k$ because it's sampled from $q_k(\boldsymbol{u}_k)$):

$$\begin{aligned}
\mathcal{L}(\theta, \phi, x) &= -D_{\mathrm{KL}}(q_\phi(z \mid x) \| p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] = -D_{\mathrm{KL}}(q_k(\boldsymbol{u}_k) \| p_\theta(\boldsymbol{u}_k)) + \mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log p_\theta(x \mid \boldsymbol{u}_k)] \\
&= -\mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log q_k(\boldsymbol{u}_k) - \log p_\theta(\boldsymbol{u}_k)] + \mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log p_\theta(x \mid \boldsymbol{u}_k)] \\
&= -\mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log q_k(\boldsymbol{u}_k)] + \mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log p_\theta(x, \boldsymbol{u}_k)] \\
&= -\mathbb{E}_{q_k(\boldsymbol{u}_k)}[\ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \ln |\det \frac{\partial f_m}{\partial \boldsymbol{u}_{m-1}}|] + \mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log p_\theta(x, \boldsymbol{u}_k)]
\end{aligned}$$

But now in the first term, we have expectation w.r.t. $q_k(\boldsymbol{u}_k)$, but inside, we have samples from $q_0(\boldsymbol{u}_0)$, so we need to change the expectations with the law of unconscious statistican, the one we also used in the theory part; if $Y = g(X)$ and g is some invertible function, then the expectation of any function $h(Y)$ w.r.t. the distribution of Y is:

$$\mathbb{E}_{y \sim P_Y}[h(Y)] = \int h(y) p_Y(y) dy = \int h(g(x)) p_X(x) dx = \mathbb{E}_{x \sim P_X}[h(g(X))]$$

Using that we get the following VAE loss for planar flow:

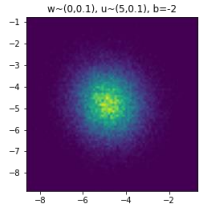
$$\begin{aligned}
\mathcal{L}(\theta, \phi, x) &= -\mathbb{E}_{q_k(\boldsymbol{u}_k)}[\ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \ln |\det \frac{\partial f_m}{\partial \boldsymbol{u}_{m-1}}|] + \mathbb{E}_{q_k(\boldsymbol{u}_k)}[\log p_\theta(x, \boldsymbol{u}_k)] \\
&= -\mathbb{E}_{q_0(\boldsymbol{u}_0)}[\ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \ln |\det \frac{\partial f_m}{\partial \boldsymbol{u}_{m-1}}|] + \mathbb{E}_{q_0(\boldsymbol{u}_0)}[\log p_\theta(x, \boldsymbol{u}_0)]
\end{aligned}$$

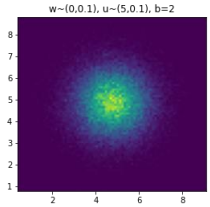
But the loss, is the negative of ELBO, so by taking that into account and replacing the logdet $\log |\det \frac{\partial f}{\partial \boldsymbol{z}}| = \log |1 + h'(\boldsymbol{w}^\top \boldsymbol{z} + b) \boldsymbol{u}^\top \boldsymbol{w}|$, we get:

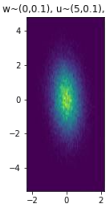
$$\text{loss} = -\mathcal{L}(\theta, \phi, x) = \mathbb{E}_{q_0(\boldsymbol{u}_0)}[\ln q_0(\boldsymbol{u}_0) - \sum_{m=1}^{k} \log |1 + h'(\boldsymbol{w}_m^\top \boldsymbol{z}_{m-1} + b) \boldsymbol{w}_m^\top \boldsymbol{u}_m| - \log p_\theta(x, \boldsymbol{u}_k)]$$

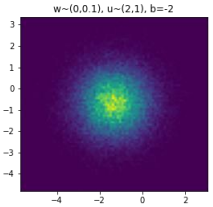
The advantage is that before using flows, we had to choose a family of densities and optimize that to get to the true posterior, but if our family is not expressive enough, then even asymptotically we can't reach posterior, but with normalizing flows, since we can have much flexibility, it's much more likely that the true posterior falls in the family of densities that the chosen flow can capture, and thus as opposed to mean Gaussian field approaches or other simple family densities, we can get very close to the true posterior, and hence get a better generative performance.
In problem 2 of the theory part, we showed that maximizing the ECLL coincides with that of the marginal likelihood only if $q(z \mid x)$ (approximate posterior) perfectly matches $p(z \mid x)$ (true posterior). So the point of normalizing flows is that they enable such condition by their flexibility, and thus if $q(z \mid x)$ perfectly matches $p(z \mid x)$, then we can maximize ECLL instead of the marginal log-likelihood.

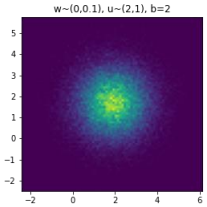
$w \sim (0,0.1), \mu_u \in \{0,2,5\}, \sigma_u^2 \in \{0.1,0.5,1\}$

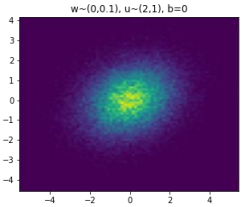


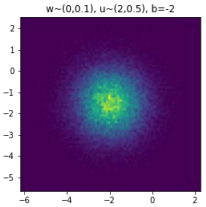


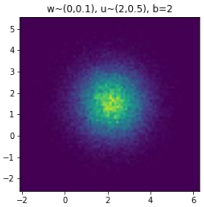


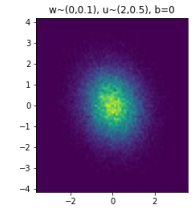


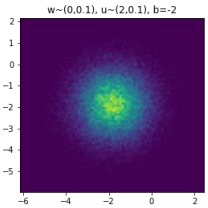


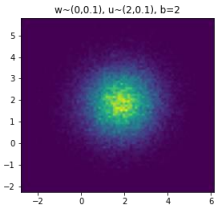


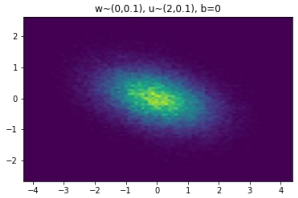


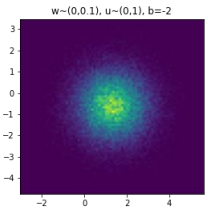


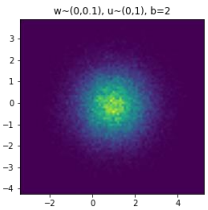


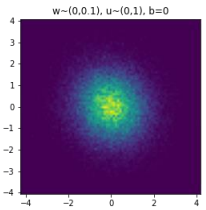


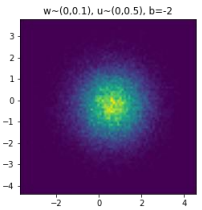


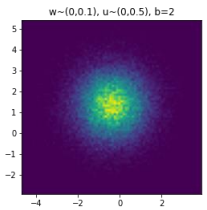


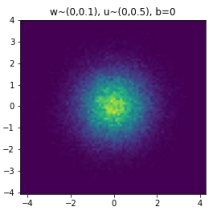


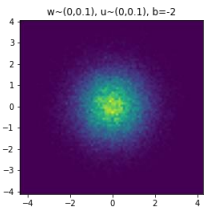


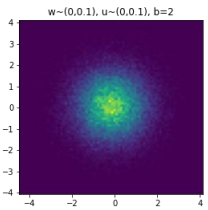


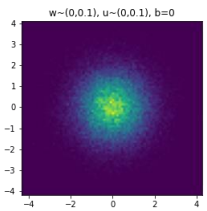


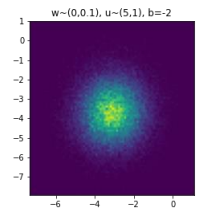


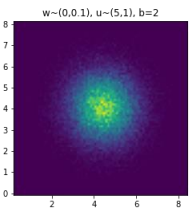


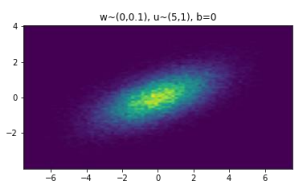


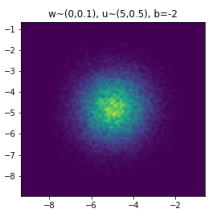


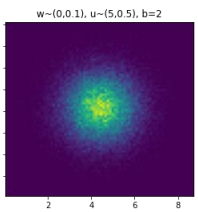


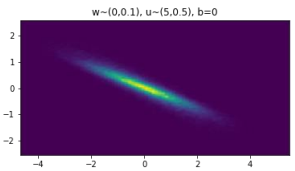




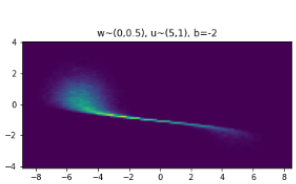


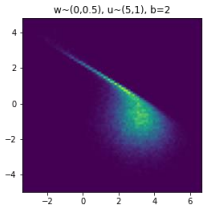


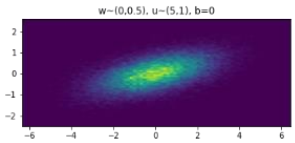


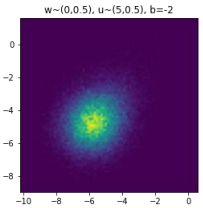


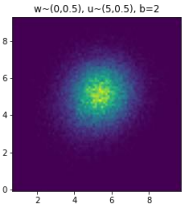
$w \sim (0,0.5), \mu_u \in \{0,2,5\}, \sigma_u^2 \in \{0.1,0.5,1\}$

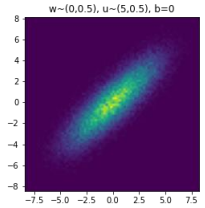


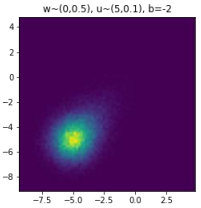


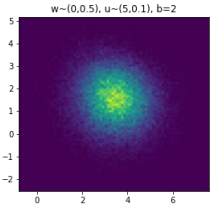


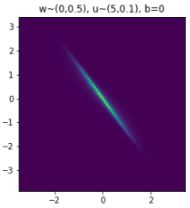


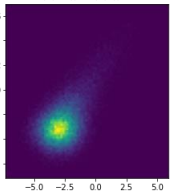


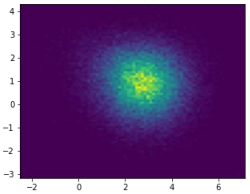


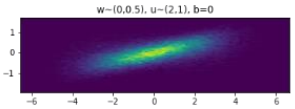


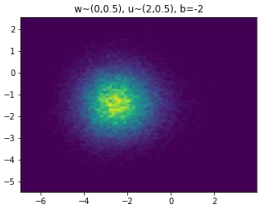


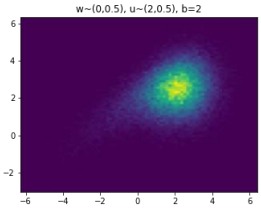


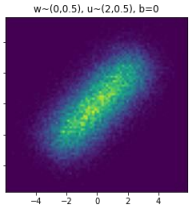


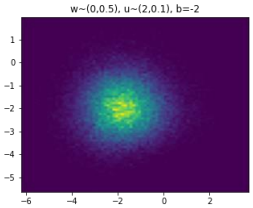


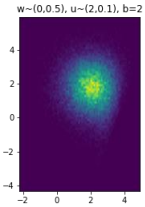


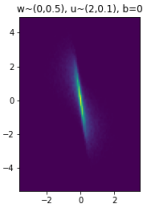


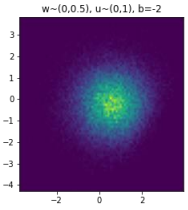


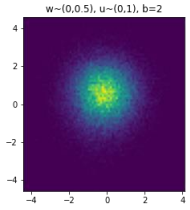


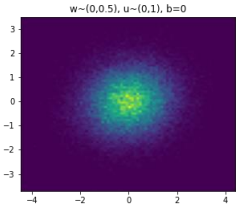


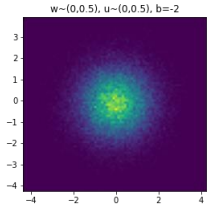


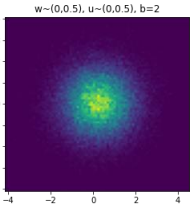


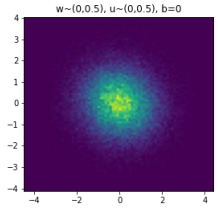


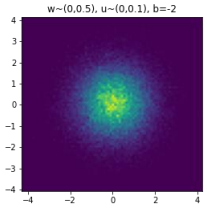


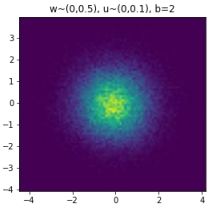


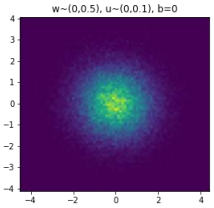




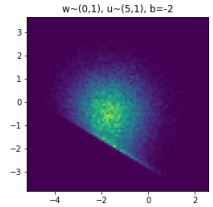


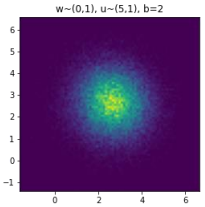


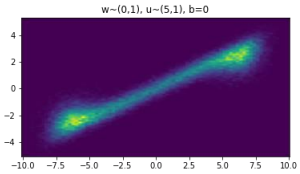


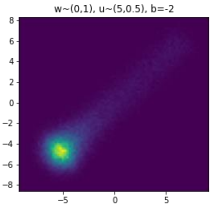


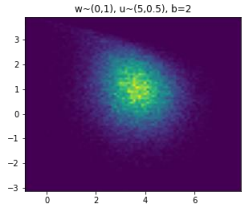
$w \sim (0,1), \mu_u \in \{0,2,5\}, \sigma_u^2 \in \{0.1,0.5,1\}$

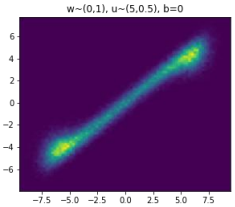


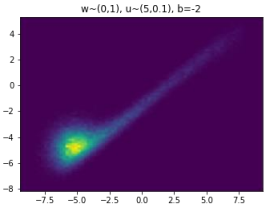


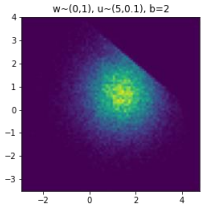


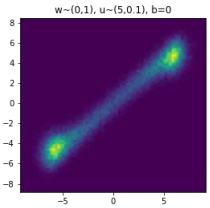


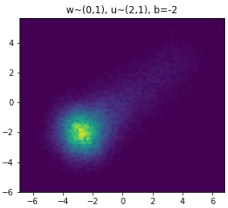


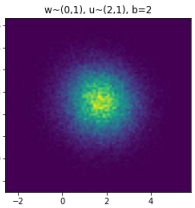


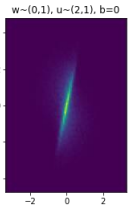


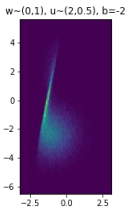


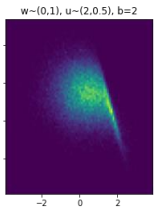


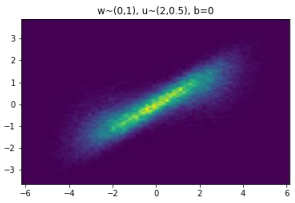


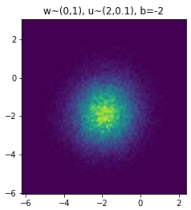


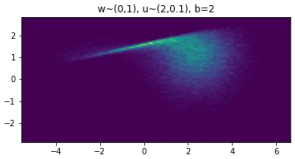


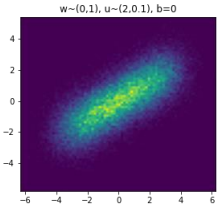


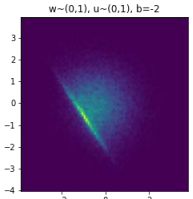


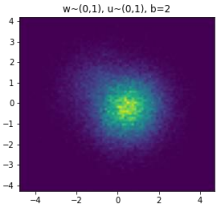


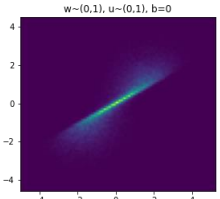


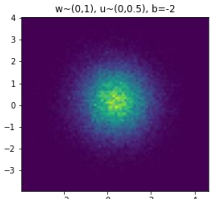


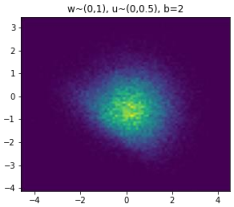


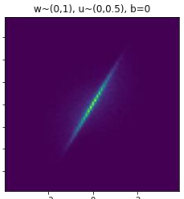


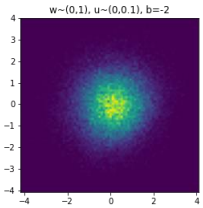


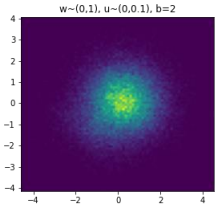


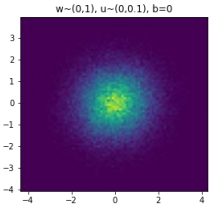












$w\sim(5,0.1), \mu_u \in \{0,2,5\}, \sigma_u^2 \in \{0.1,0.5,1\}$