International Institute of Information Technology
Bangalore

Advanced Visual Recognition and Rendering
AI-836

Project Report

Sarthak Harne
-IMT2020032

Monjoy Narayan Choudhury
-IMT2020502

Abhinav Mahajan
-IMT2020553

December 17, 2023

Problem Statement

Our work aims to enhance the outputs of a Variational Auto Encoder (VAEs) by performing a Bayesian Optimisation Process [1] on the latent space of the VAE.

Preliminaries

Variational Autoencoders (VAE's): -

In essence, a VAE is equivalent to non-linear Factor Analysis, a classic latent variable probabilistic model, using deep neural networks as nonlinear functions. Given a parameterized probabilistic model $p\theta(x, z)$, where z denotes the latent variables and x the data, we usually want to infer something about the latent having observed data. This can be done with the rules of probability and an appropriate prior belief on the latents:

$$p_\theta(z|x) = \frac{p_\theta(x,z)}{p_\theta(x)} = \frac{p_\theta(z)p_\theta(x|z)}{p_\theta(x)} \tag{1}$$

Here, the generative model $p_\theta(x, z)$ is decomposed into the product of the prior, $p_\theta(z)$, and a stochastic decoder $p_\theta(x|z)$. While the choice of a prior may feel awkward initially, we will see that it can be used as a powerful tool (for example, to encourage disentangled representations.) Since computing $p\theta(z \mid x)$ is an intractable integral over the complete parameter space, it is approximated with a parameterized stochastic encoder $q_\phi(z|x)$. It is possible to jointly train the decoder and encoder parameters θ and ϕ by maximizing a lower bound to the marginal likelihood of the data $p_\theta(x)$. This is referred to as Evidence Lower Bound (ELBO $L_\theta,\phi(x)$) with respect to both parameter vectors:

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z})\right) && (2)\\
&= \log p_{\boldsymbol{\theta}}(\mathbf{x}) - D_{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) && (3)\\
&\leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) && (4)
\end{aligned}$$

We can easily see that (4) indeed is a lower bound due to the non-negativity of the KL divergence term. It is clear from (3) that maximizing this (or minimizing the negative, if treated as a loss) will also minimize the inclusive KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$. Thereby, making the encoder more accurate. The two additive terms of the ELBO are renamed for our purposes as:

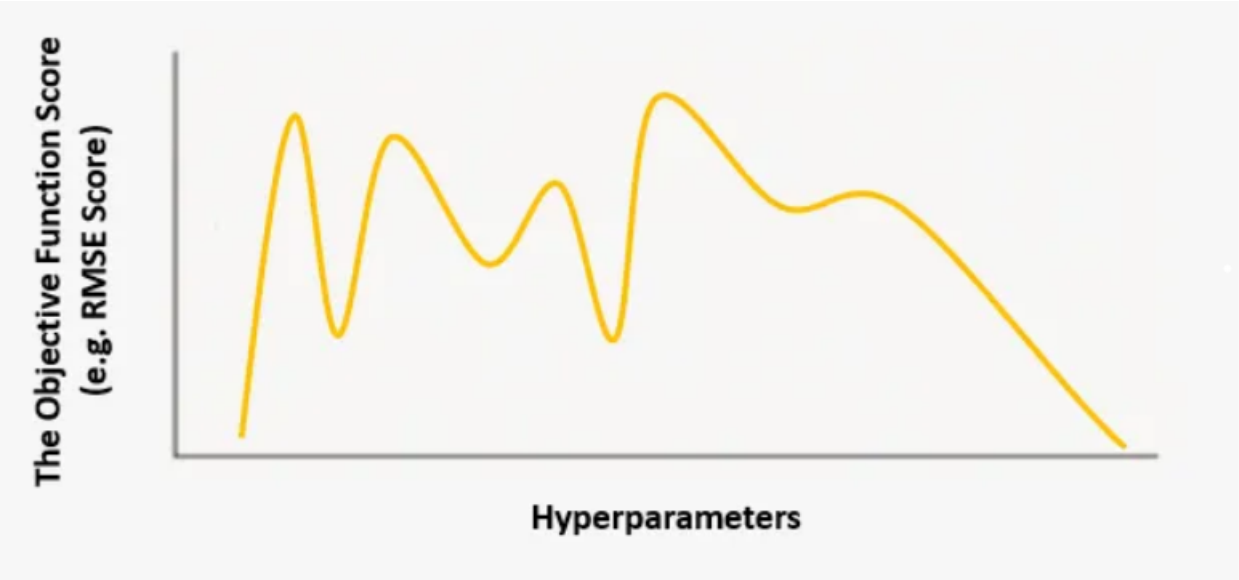
$$\text{ELBO} = \mathcal{L}_{\text{like}}^{\text{pixel}} + \mathcal{L}_{\text{prior}} \tag{5}$$

with $L_{like}^{pixel} = E_{q\phi}(z|x) \log p\theta(x \mid z)$ (interpretable as the data reconstruction loss) and $L_{prior} = \text{DKL } q\phi(z|x)kp\theta(z)$ (interpretable as the discrepancy of the posterior from the prior). Finally, note that the objective in 5 cannot be optimized directly because of the sampling from $q\phi(z|x)$. Therefore, the reparameterization trick (Kingma & Welling, 2014[2]) is used to express z as a deterministic variable by introducing an auxiliary random variable with a fixed distribution.

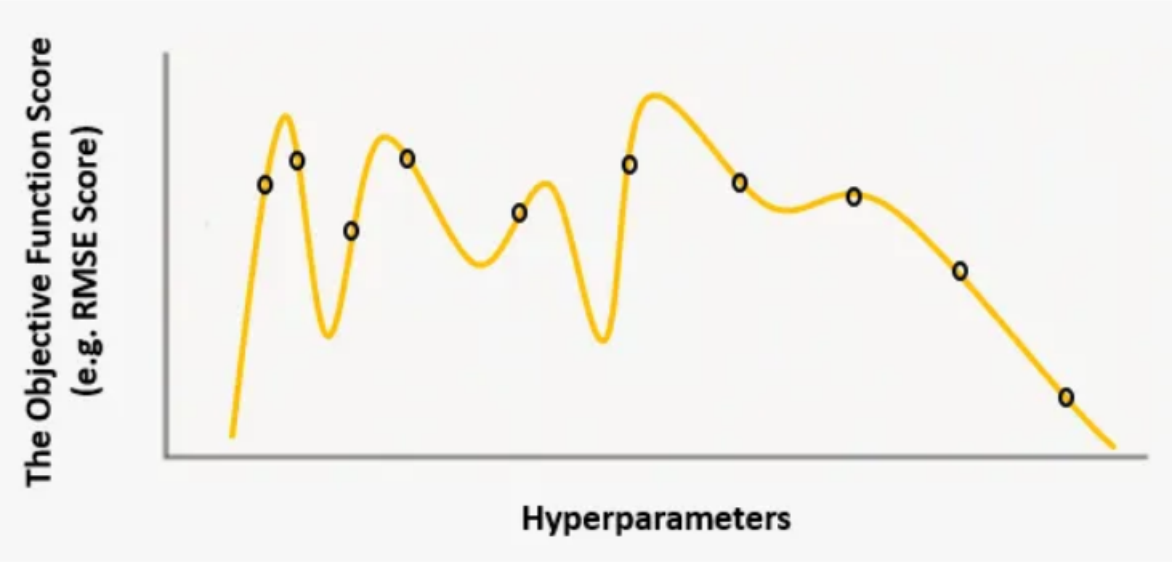
Bayesian Optimisation

Hyperparameter tuning is often a crucial factor in achieving optimal performance in machine learning models, especially with deep neural networks. However, the process of finding hyperparameters that generalize well on unseen data is both expensive and difficult. The most popular approach is cross-validation or k-fold cross-validation. The downside of these methods is that they are computationally expensive and decrease the amount of data available for training. A principled approach to hyperparameter tuning that uses all data (without the need for a validation set) is Bayesian optimization (BO). It is a general approach to optimizing unknown, expensive functions.
The other traditional tuning algorithms include manual searching, grid searching, and random searching. manual searching is time-consuming, and may not be exploratory enough. Grid searching is time-consuming, especially

when there are a lot of hyperparameters. The inefficiency stems from the fact that it does not prune away hyperparameters that are non-factors but instead uses every hyperparameter to make all possible types of combinations and permutations which is why it is time-consuming and hard to use, it has to be kept in mind that each run is very expensive, for example, if you want to train the learning rate, you need to perform the entire training procedure all over again for a new learning rate. Hence a smarter method is needed to perform tuning. Random Searching is a step in the right direction but again lacks enough parameter exploration field. Therefore, Bayesian Optimisation was a technique used to mitigate the issues posed by hyperparameter tuning. Basic Bayesian philosophy in any context(signal processing, signal estimation, adaptive filters, Total probability, etc.) is the same, use all the prior information in the modeling procedure. In the case of hyperparameter tuning, we first make a few runs, we randomly choose combinations of hyperparameters and get their objective scores, then use this as prior information to find smarter combinations that are more likely to give better objective scores values. For example, suppose we want to minimize the score by tuning the β hyperparameter, which is the task we have at hand. First, we make our initial few guesses of hyperparameter β (eg. β=1, 5, 10, 50, 150) and get the Objective Function scores for each case. Now we have to try training with a new value of β, and we want to try with a value such that we are more likely to get a lower value of the Objective Function score or try a value such that we are exploring the hyperparameter space more, as with each run we learn something. Bayesian Optimisation provides a framework that handles all of this. It makes smart initial assumptions/guesses and provides a methodology to optimize the objective functions in a time-efficient and smart way, by weighing both exploration as well as objective function value.
Let's try to understand how all of this happens with a more technical explanation. Suppose we have an objective function (Objective Function score, RMSE score, F1 score, etc.) completely parameterized by a few parameters like learning rate, architecture details like number of layers, momentum, β. Our task is to find the configuration of parameters that minimize/maximize the objective function. The problem is we don't have the actual function, we just know its value at a few discrete points(let's say 10), and it's an expensive process to re-evaluate at other points.

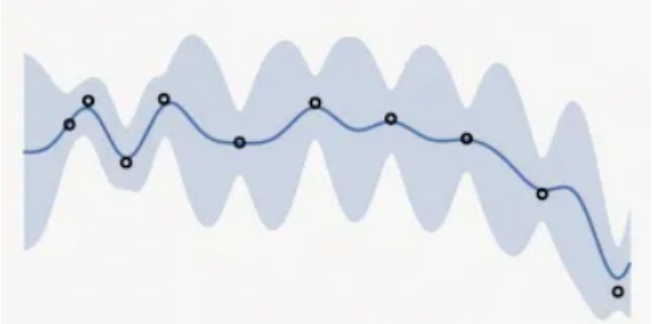


True objective function



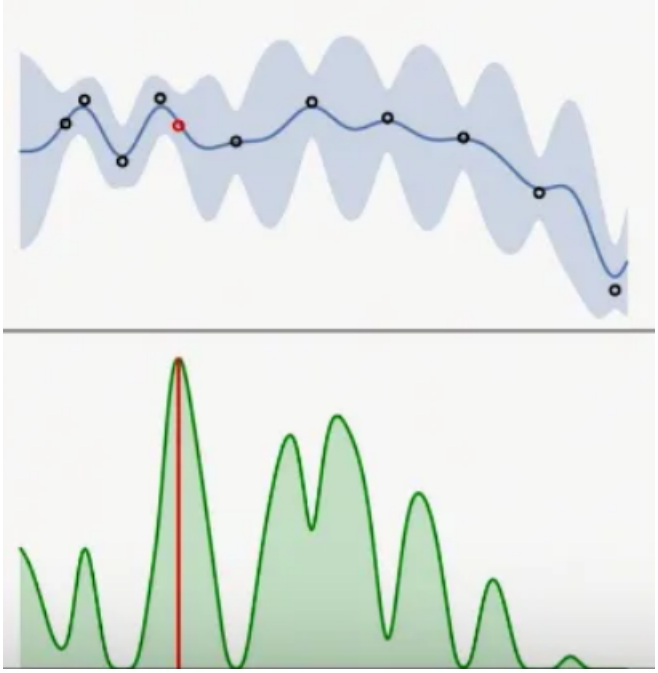
We only know the value at the discrete points

Using the prior information, the values at the discrete points, we generate a surrogate model (also called the response surface model). Which is a weak probabilistic approximation of the true objective functions. It looks like this below :



Surrogate function, where the blue shade signifies the standard deviation/conObjective Functionence

Now we have 10 samples of the objective function and how should we decide which parameter to try as the 11^{th} sample? We need to build an acquisition function (also called the selection function). The next hyperparameter of choice is where the acquisition function is maximized. In the figure below, the green shade is the acquisition function and the red straight line is where it is maximized. Therefore the corresponding hyperparameter and its objective function score, represented as a red circle, is used as the 11^{th} sample to update the surrogate model.



Top: surrogate function, Bottom: acquisition function

As described above, after using an acquisition function to determine the next hyperparameter, the true objective function score of this new hyperparameter is obtained. Since the surrogate model has trained on the (hyperparameter, true objective function score) pairs, adding a new data point updates the surrogate model. Repeat the above steps until the max time or max iteration is reached. And voila! You now have an accurate approximation of the true objective function and can easily find the global minimum from the past evaluated samples. Your Bayesian Optimization is completed!

$$\mathrm{EI}_{y^*}(x) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p_M(y|x) dy.$$

p(y|x): the surrogate model. y is the true objective function score and x is the hyperparameter

y: the minimum observed true objective function score so far*

y: new scores

Expected Improvement (EI) formula

Methodology

We use the MNIST dataset and some standard PyTorch examples to show a synthetic problem where the input to the objective function is a 28 x 28 image. The main idea is to train a variational auto-encoder (VAE) on the MNIST dataset and run Bayesian Optimization in the latent space. Let our synthetic expensive-to-evaluate objective function take the form:
image → image classifier → scoring function → score.

- The classifier is a convolutional neural network (CNN) trained using the architecture of the PyTorch CNN example.
- Our VAE architecture is the same as the PyTorch VAE example as training the entire VAE from scratch will be an expensive process and transfer learning would be helpful.
- We define a scoring function that maps digits to scores. It can be given in the form

  $$sf(y, k) = e^{-2*(y-k)^2}$$

  Here y is the random variable referred while k is fixed number. K depends on which digit sample we would like to generate. Let us say we want k=3, then the equation will reduce to

  $$sf(y) = e^{-2*(y-3)^2}$$

  Given the scoring function, we can now write our overall objective, which as discussed above, starts with an image and outputs a score. Let's say the objective computes the expected score given the probabilities from the classifier.
- Now on that we need to extract the parameters μ and Σ of the variational distribution and performs reparameterization and the decoding. We use batched Bayesian optimization to search over the parameters μ and Σ such that we get a good generation of the digit k = 3 in our case.

Architecture Diagram

Image Classifier for Enforcing Digit

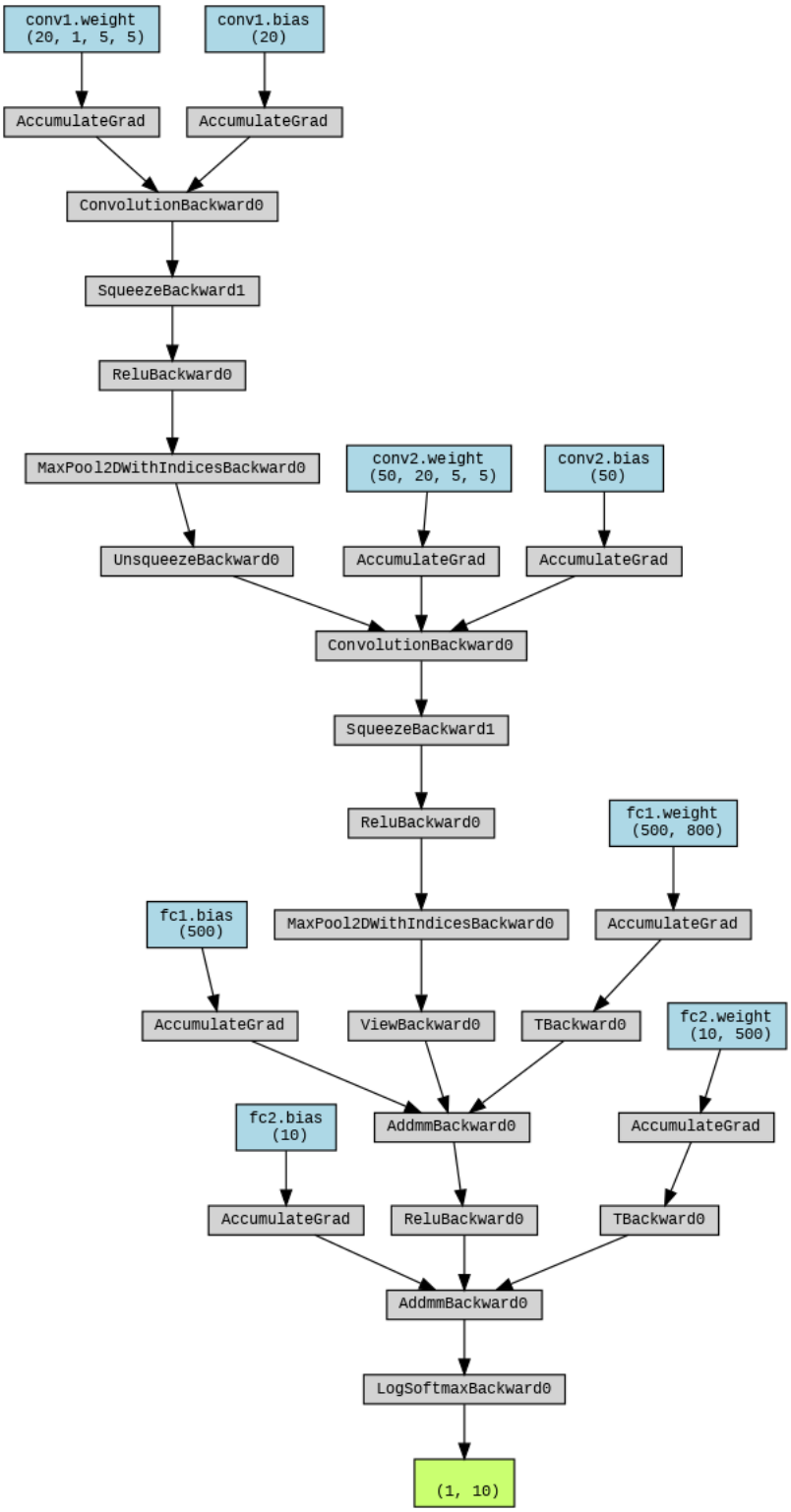


Figure 1: Classifier Architecture

VAE Architecture

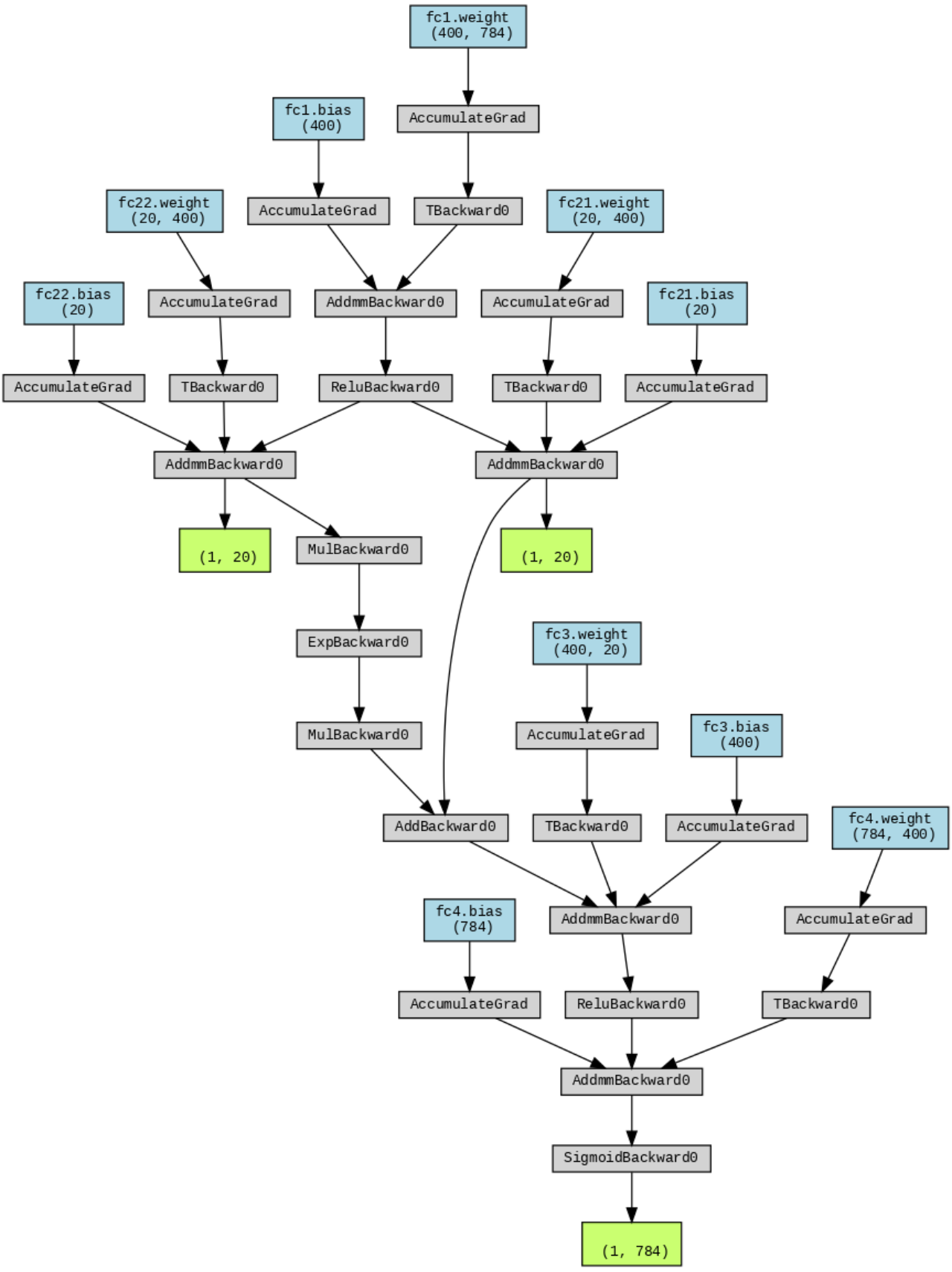


Figure 2: VAE Architecture

Understanding the Bayesian Optimisation Loop in case of VAEs

The loop consists of the following steps:

1. The acquisition function proposes a new candidate latent vector based on the current model and objective function.
2. The VAE generates a digit from this vector, and its quality is evaluated using the chosen objective function.
3. The BO model is updated with the new information, refining its understanding of the latent space and its predictions.
4. This process repeats, iteratively searching for latent vectors that produce high-quality MNIST digits

Implementation Details

We use the pretrained weights from pytorch and the examples for the model architectures while for Bayesian Optimisation we use a library called BoTorch which adds the optimisation technique over the normal pytorch backend allowing us to add our own code for extracting latents and analysing intermediate results given by the technique. All results were run on a colab notebook with T4 GPU.

Results and Conclusion

The results for the model run for all the MNIST digits are given below. The first image in each strip is generated by using parameters from random sampling. Subsequent images are generated after 10%, 25%, 50%, 75%, and 100% of the budget. The total budget that we work with is 75 iterations.

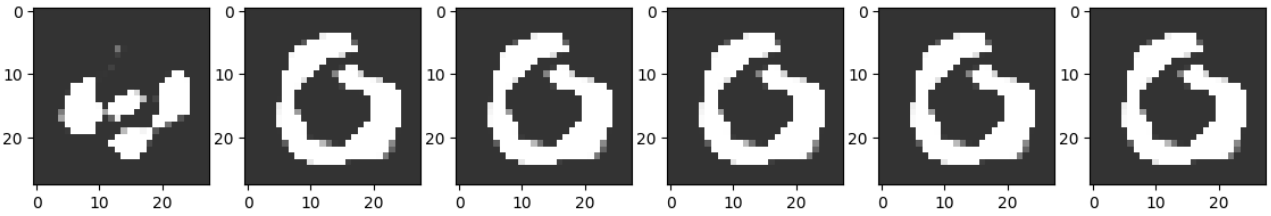


Figure 3: Results for 0

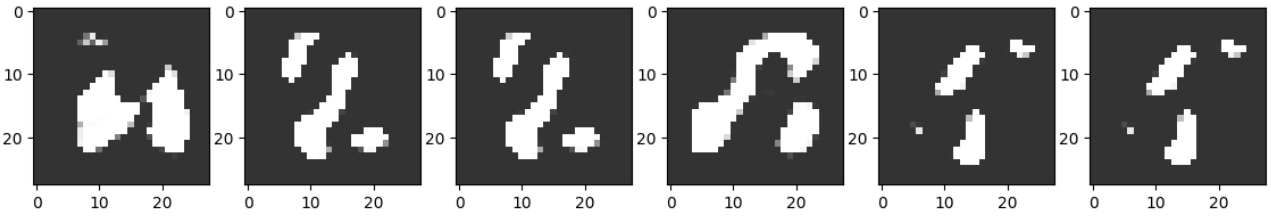


Figure 4: Results for 1

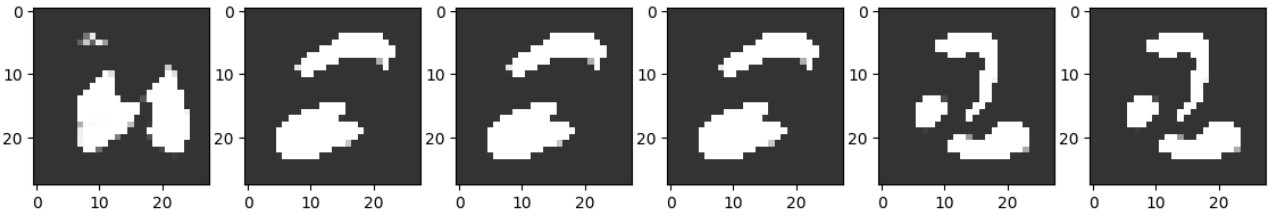


Figure 5: Results for 2

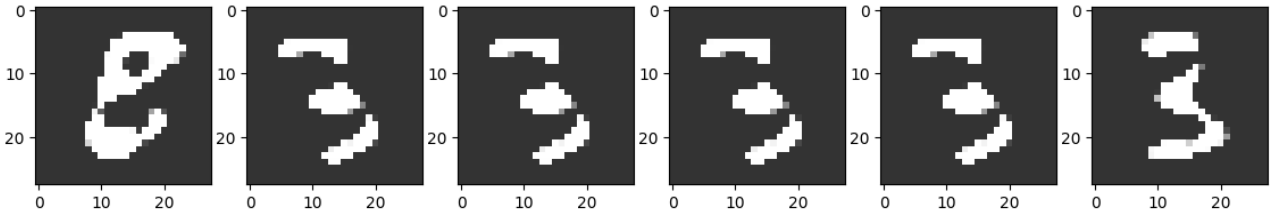


Figure 6: Results for 3

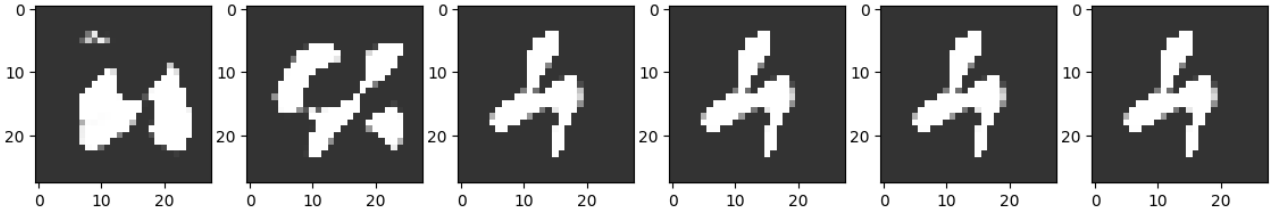


Figure 7: Results for 4

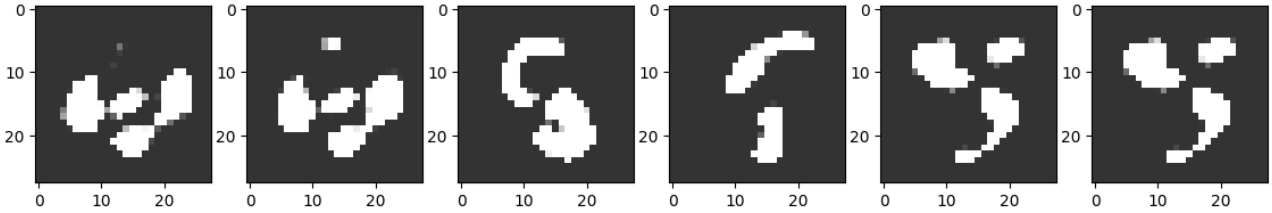


Figure 8: Results for 5

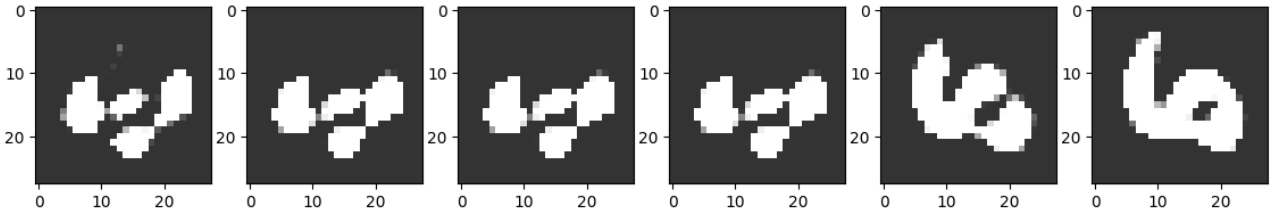


Figure 9: Results for 6

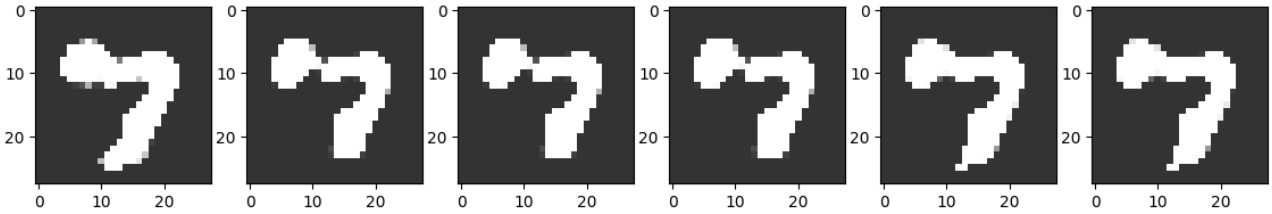


Figure 10: Results for 7

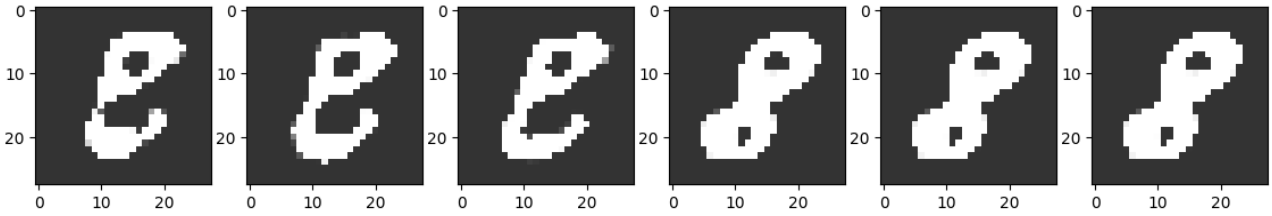


Figure 11: Results for 8

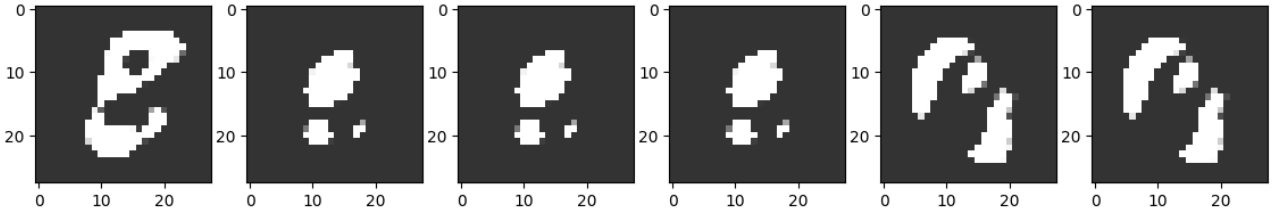


Figure 12: Results for 9

From the image above, we can see that the model works well with the numbers 3, 4, 6, 7, 8. It works decently with 2 and 5 and gives bad results for 1 and 9.
From the above results, it is apparent that the Bayesian Optimisation for selecting the parameters for sampling works better, as the structure of the digit becomes better iteratively. This showcases how Bayesian Optimisation effectively navigates the latent space, focusing on areas likely to produce high-quality digits, saving computational resources compared to an exhaustive search. The flexibility of this approach allows us to experiment with different VAE architectures, latent space dimensions, and evaluation metrics to achieve the desired outcomes in image generation.

Future Work

- The idea of Bayesian Optimisation for the selection of parameters can be used for β-VAEs. As the values of β are in our control, Bayesian Optimisation can be used to control the disentangled features.
- Classifier-free guidance also involves weighting the terms of condition-free and conditioned losses using hyperparameters. Bayesian Optimisation can be used to control these weights and reach an optimal point of exploration and exploitation.

Code Link

1. Colab Notebook Link
2. OneDrive Link containing required weight files

References

[1] Jasper Snoek and Hugo Larochelle and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms.

[2] Kingma, Diederik P. and Welling, Max. Auto-encoding variational bayes. In 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, 2014.