EBM for Bitmoji

Papers followed: Class Notes
Dataset: Bitmoji - https://www.kaggle.com/datasets/romaingraux/bitmojis
Framework Used: PyTorch Lightning (1.6.4), PyTorch (1.11.0), TorchVision (0.12.0)

In EBM, we assume the form for probability density $p_\theta(x) = \frac{e^{-E_\theta(x)}}{z_\theta}$.

1. Code Details:

a) MALAConstEBM: Class that creates the EBM model
   a. Constructor Parameters – bs = batch size for the dataloaders, t – Number of Langevin steps to take
   b. The model is $E_\theta(x)$, a simple CNN model consisting of 10 layers of convolution-activation-pooling-batchnorm which outputs a scalar (called as "score" in the following text)
   c. _common_step implements the contrastive loss = mean score on the real samples – mean score on the samples obtained after taking "t" MALA steps
   d. Respective dataloaders are implemented which output the image in a tensor of range -1 to 1
   e. one_mala_step takes a batch of images and runs one MALA step on them. Here, the Langevin step is taken with tau as the step size (which is fixed to be 0.01). The samples obtained undergo a Metropolis-Hasting acceptance step. Every sample obtained is clipped between the range -1 to 1 (the range obtained from dataloaders).
   f. one_langevin_step and one_langevin_decay_step implement Langevin steps with constant and decaying step sizes respectively. The constant step size is chosen based on guidance from the TAs. The decaying step sizes is chosen based on internet sources.
   g. get_transition starts from one U(-1, 1) image and runs respective method to generate the samples. The sampling frequency is controlled using "jump" variable
   h. get_samples starts from a bunch of samples from U(-1, 1) and runs respective method to generate next samples

b) **Training Utilities**: train_and_test: General function which allows running different configurations. It takes parameters such as max_epochs, which GPUs to use etc. Even takes the model_class and the model_kwargs to instantiate the model inside the function. It uses the Trainer class of PyTorch Lightning to train, validate and test the model on the given dataset and the model. It logs everything into a tensorboard (created inside the function), which can then be used to view loss curves

c) **Plots and Analysis Utilities:**
   a. see_some_generations: View 100 images generated randomly from the model in a 10 X 10 grid
   b. see_some_transitions: View 225 transitions for an image generated randomly from the model in a 15 X 15 grid with required sampling frequency

a) Other Models contain models which vary in the way the samples are generated while training. Either using Langevin steps with decaying step size or constant step size and so on.

b) All randomness is controlled by using seeds which ensures reproducibility.

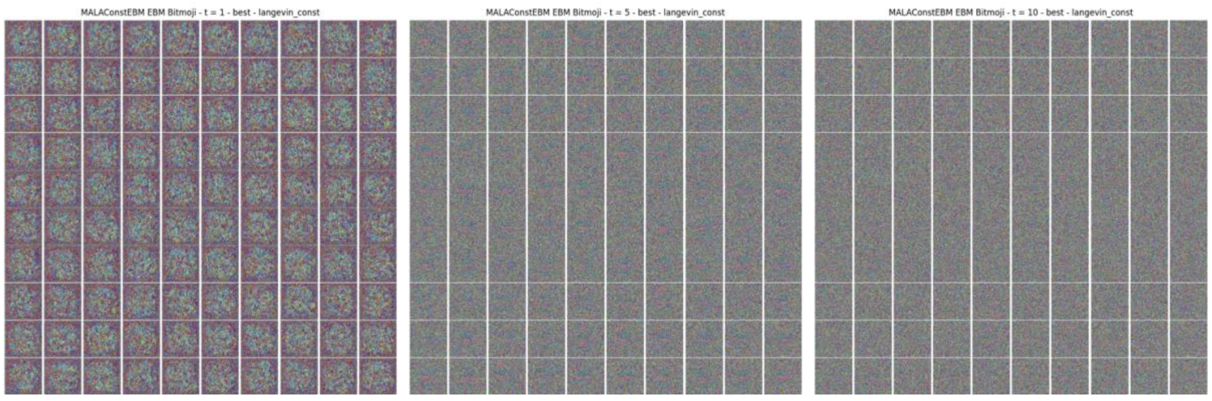
2. Architecture Choices & Details:

As the number of Langevin steps are increased, the model takes a lot of time to train. Hence, all decisions on architecture, sampling method etc. were done on t = 1 and applied to t = 5 and t = 10.

3. Experiments, Results & Inference:

Main hyperparameter which was explored was the number of Langevin steps during sampling. Samples are shown in Figure 1. For t = 1, the model seems to learn the characteristic of the Bitmoji

images, i.e., there is something changing in the middle of the image (face) and surrounded by something which is same across all images (white background).
Loss curves and the scores obtained are shown in Figure 2 & 3. Loss values seem good i.e., value of 0 is the best that can be obtained, yet that does not transfer to good sample quality! Loss for t = 1 goes highly negative which seems to give “better” samples. This was bit of a shock to find out and hope to read up more on this to understand why this could be the case.
The scores for real and fake samples i.e., $E_\theta(x)$ is negative which shows that the model is giving higher probability to the seen and generated samples.


MALAConstEBM EBM Bitmoji - t = 1 - best - langevin_const
MALAConstEBM EBM Bitmoji - t = 5 - best - langevin_const
MALAConstEBM EBM Bitmoji - t = 10 - best - langevin_const

(a) (b) (c)

Figure 1: 100 Samples obtained using MALA when training and Langevin while sampling

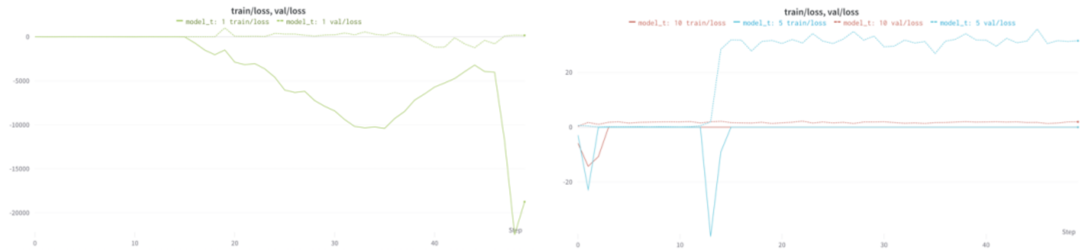


Figure 2: Loss curves for t = 1 & t = 5 and 10 Langevin steps

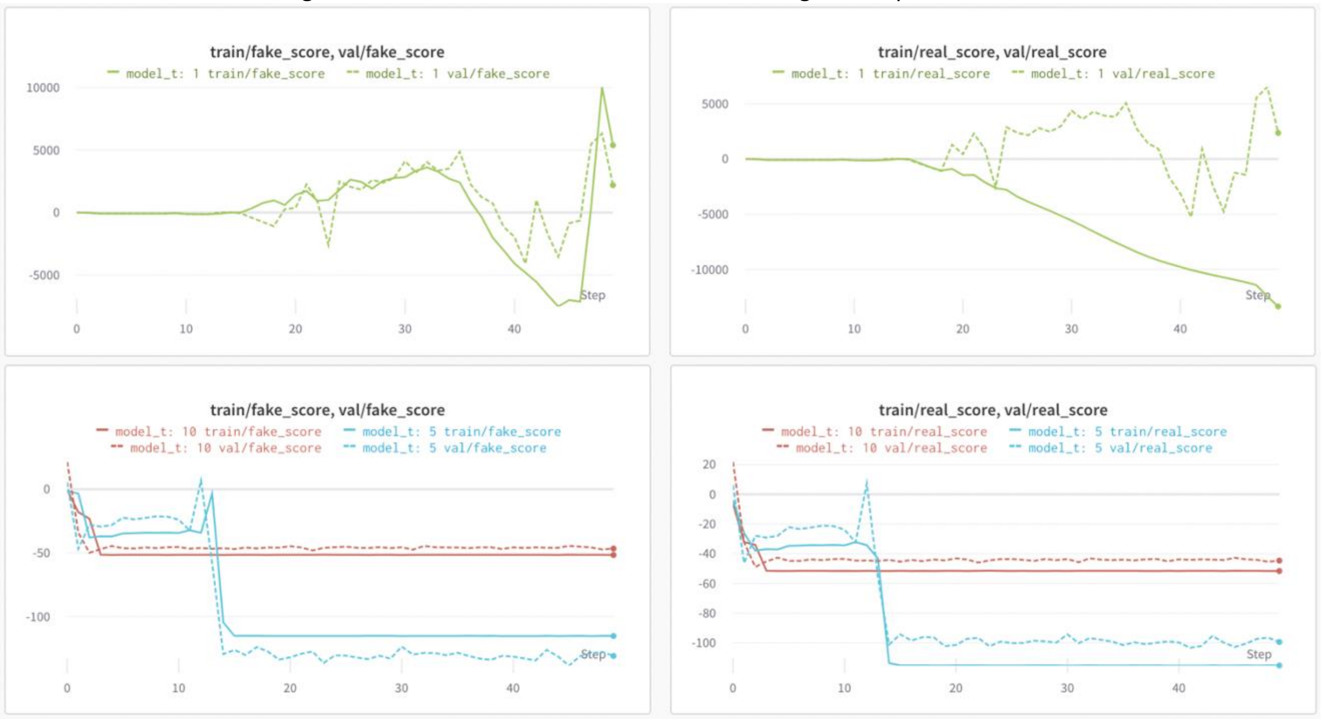


Figure 3: The scores for real and fake (= MALA samples) over epochs for different Langevin steps

4. Problems faced, solutions used:

a) Initial attempts gave only noise as output. We changed to Resnet architecture (using pretrained weights) thinking that the model may be too simplistic to capture enough information. Still noise output. Reverted to the simple architecture. Then we explored different ways to get the samples. We explored three main approaches as outlined below. MALA seems to give better results for now.
   i. Langevin steps with decaying step size (0.0001 / $t^{0.75}$)
   ii. Langevin steps with constant step size (0.95) and constant noise factor of 0.0005
   iii. MALA steps with step size of 0.01

b) For t = 5 and t = 10, each epoch takes a lot of time, so we subsampled and chose only 25000 samples for training and 1000 for test.

c) Another approach was to try using regularisation on the score output i.e., real score^2 + fake score^2. This did not give any better results

d) Another was to use Instance Norm instead of Batch Norm. The hypothesis was that batch norm could be introducing dependencies between samples of a batch and may be causing generation issues. But no better results was obtained and we stuck with Batch Norm.

Conclusion, Intuitions & Future Work:

a) This gave a very good wakeup call in terms of the complexity of the systems we are trying to build. We saw a stark difference between theory and practical viability of an algorithm. Although the algorithm is much simpler (there is no encoder-decoder sort of architecture, no generator and fooling of some network etc.), the sample quality is worse.

b) One **Intuition** as to why things are not working: The model is free to choose any score for a sample. If it trivially learns to assign the same number to all samples, the loss would be 0. The loss curves would look great but sample quality would be worse. I wonder what else would be needed to make this EBM work.

c) A personal win was to implement MALA (i.e., Langevin with Metropolis-Hastings) and see that it gave “better” results compared to other approaches

Future work is to try out a more diverse hyper-parameter search and investigate how to better this model.