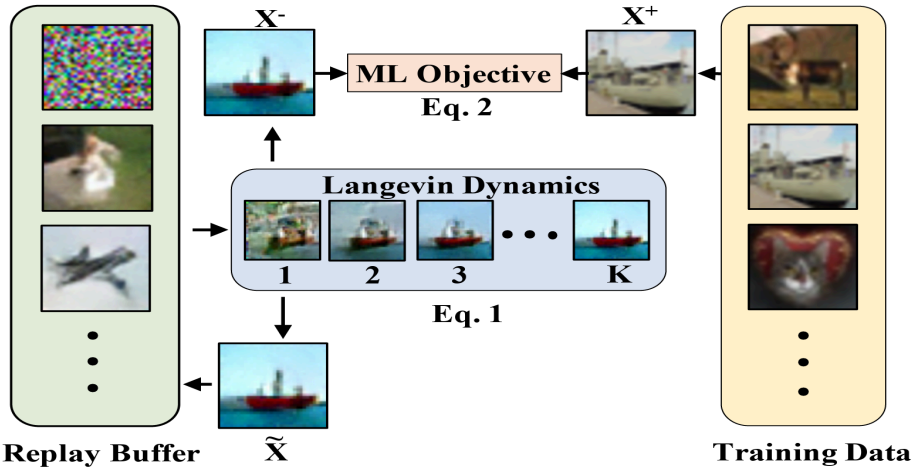
Homework 5

Coding Practice 1: Image generation using the energy-based models (EBMs)



In this part of the homework, you will implement and train energy-based models (EBMs) for image generation, both unconditional and class-conditional. You will learn how to model unnormalized distributions, sample from them via Langevin Dynamics, and train the models using Contrastive Divergence.

1.1. Data Preparation

- **Use the CIFAR-10 dataset, which consists of 32×32 RGB images, as the primary dataset for training the unconditional energy-based model. For class-conditional generation experiments, use the MNIST dataset of 28×28 grayscale digit images (separate conditional energy model on MNIST).**

1.2. Implement an energy model

First, read the article carefully to fully understand the concept and how it works. Implement a neural network $E_\theta(x)$ that maps an input image to a scalar energy. The network should output a single real-valued number, with no activation function

applied at the output layer. You should use a ResNet-like architecture with approximately 9 residual blocks, or alternatively, adopt an architecture presented in the referenced article. The objective is to define an unnormalized energy function over the image space, which will later be used for sampling synthetic data and training the model.

1.3. Implement Stochastic Langevin Dynamics:

- **Next, implement Stochastic Langevin Dynamics (SLD) to draw samples from the unnormalized energy distribution defined by your model. The update rule follows the form:**

$$X_{t+1} = X_t - (\epsilon^2/2)\,\Delta_x E_{theta}(x_t) + \epsilon\,\eta_t\,, \quad \eta_t \sim N(0, I)$$

- **This iterative process should be used to sample negative examples. You will need to design a suitable noise schedule and experiment with different values of the step size ϵ and the number of steps T. Try different configurations to find the best sampling results using trial and error.**

1.4. Train the energy model with Contrastive Divergence:

- **To train your energy-based model, use the Contrastive Divergence (CD) loss. In this setup, the goal is to compare the energy assigned to positive samples (real images from the CIFAR-10 dataset) with the energy assigned to negative samples generated through Stochastic Langevin Dynamics. You are required to implement and evaluate two different initialization schemes for the negative samples.**

- **First, always initialize negative samples from random noise.**

- **In the second scheme, initialize the negative samples from a replay buffer that stores previously generated samples. At each step, initialize 95% of the negative samples from the replay buffer and 5 percent from random noise.**

- **Repeat the previous steps while gradually reducing the noise during Stochastic Langevin Dynamics and design a decreasing sequence $\epsilon_1, \epsilon_2, \ldots, \epsilon_T$, for example :**

$$\epsilon_t = \epsilon_{start} \left(\epsilon_{end} / \epsilon_{start}\right)^{(t/T)}$$

Investigate the effectiveness of this strategy in generating negative samples.

- **Analyze and report the impact of each initialization method on the training stability, convergence speed, and sample quality.**
- **Use this loss to update your model parameters via gradient descent. The objective is to assign lower energy to real images and higher energy to synthesized ones.**

1.5. Conditional Energy-Based Modeling and Sampling on MNIST

Implement and train a conditional energy-based model on the MNIST dataset. The goal is to learn an energy function $E_\theta(x, y)$ that assigns lower energy to digit images x conditioned on their correct labels y, and higher energy otherwise. Then, adapt the Stochastic Langevin Dynamics algorithm to perform class-conditional sampling. During each step of sampling, the label y must be provided as a fixed input to guide the sampling process.

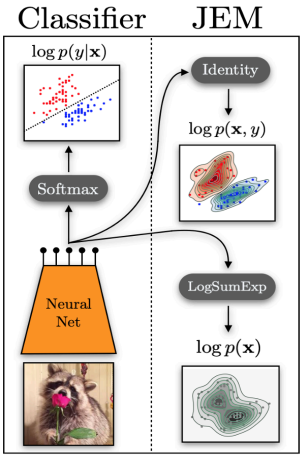
1.6. Report the performance and convergence, and visualize the image generated on the :

- **Use this loss to update your model parameters via gradient descent. The objective is to assign lower energy to real images and higher energy to synthesized ones.**

The output will be as follows:

For the implementation, you are free to use the official codebase, but you must be able to interpret and explain every line of your code. Although the official codebase uses the TensorFlow framework, you have to use PyTorch for inference.

Coding Practice 2: Secretly extend your classifier into an energy-based generative model!



Joint Energy-Based Models (JEM) reinterpret classification models as energy-based models by treating the classifier logits as unnormalized log probabilities. This allows the model to perform both classification and generation using the same network. In this exercise, we explore an advanced version called JEM++ and examine how it improves performance.

2.1. Data Preparation

- **You can use one of the CIFAR-10 or MNIST datasets.**

2.2. Providing a pretrained model

- **First, read the technical reports on JEM and JEM++.**
- **Use a pretrained classifier (with softmax output)—either ResNet-18 or a small CNN—on a subset of CIFAR-10 or MNIST, depending on which dataset you find more convenient.**
- **The model should achieve at least 85% accuracy on MNIST or 70% accuracy on CIFAR-10.**

2.3. Convert the discriminative model to a generative one:

- **Reinterpret the classifier as an energy-based model by defining an energy function. To treat the classifier as a Joint Energy-Based Model (JEM), incorporate a log-partition function over the class dimension.**
- **Modify the training loss function accordingly. The JEM loss is formulated as a combination of the original cross-entropy loss and a term representing the negative log-likelihood of the input x, which is approximated via maximum likelihood using Stochastic Gradient Langevin Dynamics (SGLD).**

2.4. Starting to generate:

- **For training, you will also need to sample negative images that contrast with your data distribution. Use Stochastic Gradient Langevin Dynamics (SGLD) for this sampling step. The update rule is given by During this phase, ensure that your classifier still performs well on the original classification task.**

2.5. Applying Modifications to the Main Article

1. **Following the JEM++ approach, initialize the SGLD process using a Gaussian Mixture Model (GMM) estimated over the training data, instead of using a mix of noise and a replay buffer.**

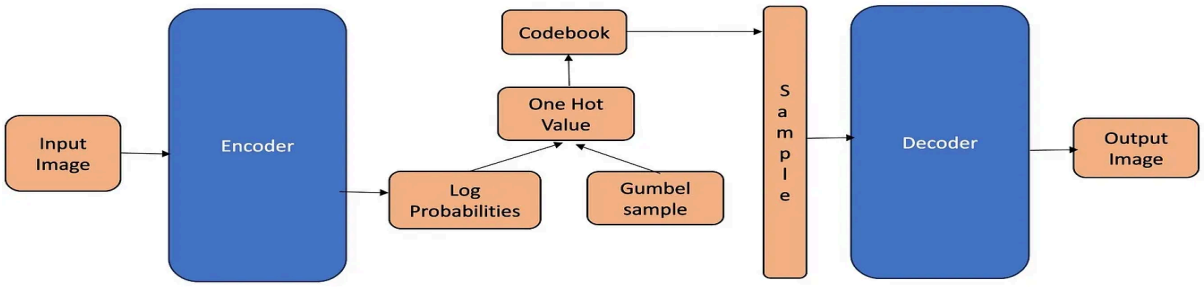
2. **Implement Proximal SGLD as proposed in JEM++, and compare its performance with the original JEM method.**

Evaluate the combined impact of these two modifications on the original JEM framework, and analyze the resulting performance improvements.

2.6. Visualization and Accuracy

- **Show the classification accuracy of your model.**
- **Display the generated images as examples of the model's generative capability.**
- **Use the results from the article as a reference for evaluating the generation quality.**
- **Plot the training loss and accuracy curves over the course of training.**

Coding Practice 3: Exploring the latent space of DVAEs!



In this section, we train a Discrete Variational Autoencoders (DVAE) model. The encoder generates latent feature maps, which are then quantized using a learned vector codebook. These quantized representations capture high-level image features as discrete codes, also known as codebook indices or tokens. These tokens act as symbolic representations of the image content. Subsequently, we perform clustering on these discrete tokens to group similar features together. This clustering process produces pseudo-labels that can be utilized for various downstream tasks, including unsupervised learning, classification, and segmentation.

3.1. Data Preparation

- **You can use either the MNIST or Fashion MNIST datasets for this purpose.**

3.2. Model

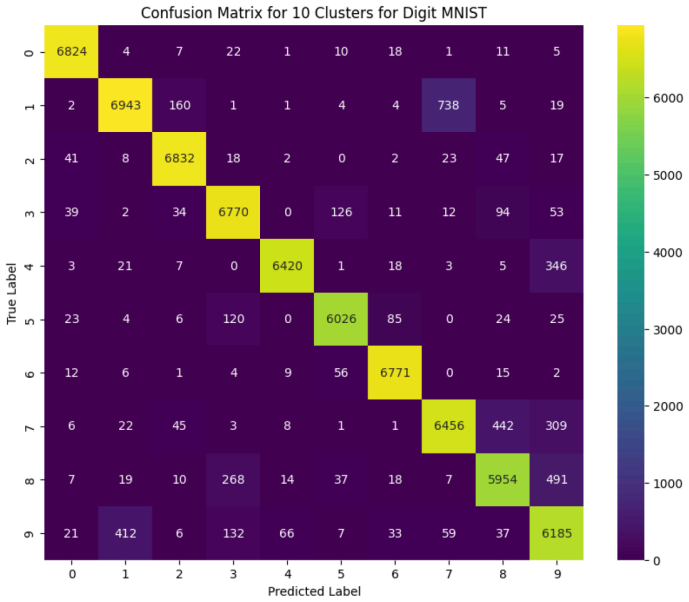
- **Study the concept of discrete variational autoencoders and train one of their variants on the MNIST and Fashion MNIST datasets. Just Ensure that the models achieve good reconstruction results.**

3.3. Creating pseudo labels

Using the quantized representations obtained from each image:

- **Apply KMeans clustering to group the images into *K* clusters.**
- **Explore different values of *K* to determine the most suitable number of clusters.**
- **Calculate the silhouette score for each clustering result corresponding to different *K* values.**
- **Plot the silhouette scores against the respective *K* values.**
  (Note: It is acceptable if the results do not reveal clear insights.)

- **Use the cluster IDs as pseudo-labels by selecting the best clustering results (i.e., the value of *K* that yields the highest silhouette score or best separation). For easily separable datasets like MNIST, you should obtain results similar to those above.**
- **Assuming the dataset is initially unlabeled, annotate it using the cluster assignments from the best clustering result, taking into account the cluster consistency with the training data.**



3.4. Creating pseudo labels

Train a classifier twice—once using the true labels and once using the pseudo-labels generated from clustering—and compare their performance.

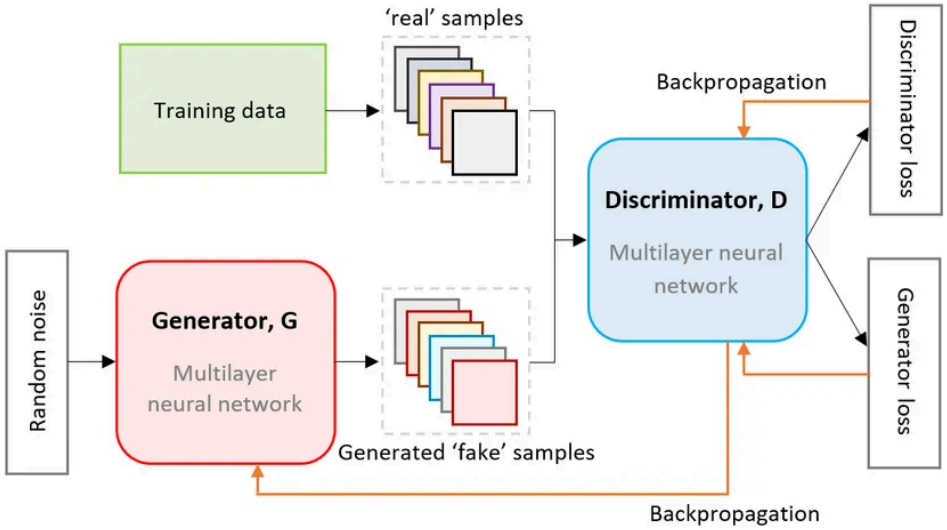
3.5. Evaluation

For each dataset (MNIST and Fashion MNIST), perform the following steps:

- **Clustering evaluation charts (e.g., silhouette scores versus the number of clusters).**
- **Confusion matrix comparing pseudo-labels (from clustering) with true labels.**
- **Percentage distribution of each true label within every cluster (manual step).**
- **t-SNE visualization of the latent space**
- **Classification accuracy when training with true labels versus training with pseudo-labels.**

Perform this process, and report the results with appropriate discussion.

Coding Practice 4: The Generator and Discriminator Are Finally Well-Matched Rivals!



Implement an Image GAN model, to generate images of Persian digits (0–9).

4.1. Data Preparation

- **You will be provided with a dataset containing binary images of Persian digits.**
- **Each image has a black background and a white foreground representing the digit.**
- **For simplicity and consistent results, you can resize the input image to 28x28.**
- **Shuffle and split the dataset into training, validation, and test sets with a ratio of 85%, 10%, and 5%, respectively.**

4.2. Simple GAN Model

- **Study the concept of GANs and implement a GAN model to generate binary images of Persian digits. (No need to use any conditional input.)**

- **A Generator that takes random noise as input and outputs a 28×28 binary image.**

- **A Discriminator that takes an image as input and outputs a probability indicating whether the image is real or generated.**
- **Use appropriate loss functions and optimizers. Choose hyperparameters (learning rate, batch size, etc.) at your discretion, but justify them.**

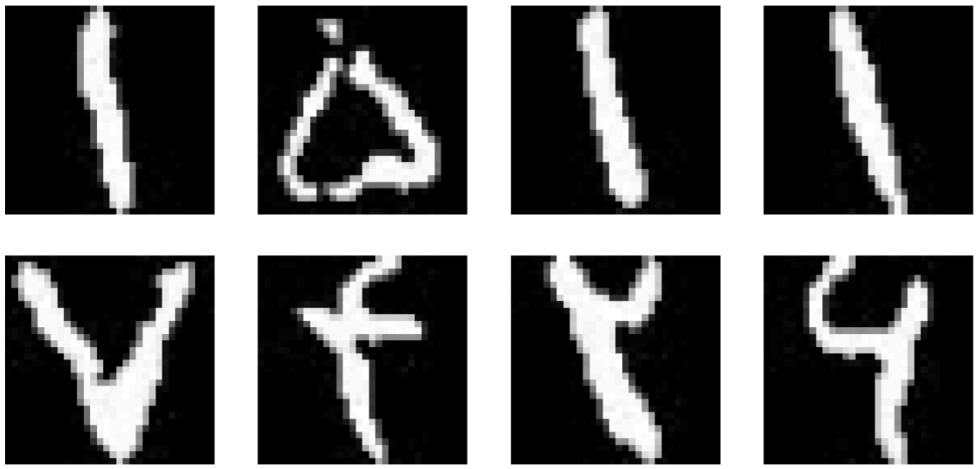
4.3. WGAN Model

- **One of the versions of GANs are WGAN. Now Based on the loss function introduced in the paper, correct your code for this version, retrain the model with new settings.**
- **In your opinion, what are the issues still present in the WGAN model you implemented? Explain them, and if you have any suggestions to resolve them, describe those as well.**

4.4. Visualization and Evaluation

- **Generate sample images from both the trained GAN and WGAN models.**
- **Visualize and compare image quality using plots.**
- **Discuss differences in image realism, training stability, and convergence behavior.**

The output for GAN will be as follows:



references

- Du, Y., & Mordatch, I. (2019). Implicit generation and modeling with energy based models. Advances in neural information processing systems, 32.

- Yang, X., & Ji, S. (2021). Jem++: Improved techniques for training jem. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 6494-6503).

- Grathwohl, W., Wang, K. C., Jacobsen, J. H., Duvenaud, D., Norouzi, M., & Swersky, K. (2019). Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*.

- Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In International conference on machine learning (pp. 214-223). PMLR.