Assignment 1

Machine Learning II

Indian Statistical Institute

Instructor: Swagatam Das

August 23, 2024

Notes

- Please complete and submit the assignments by the given deadline.
- The written assignments must be done in LaTeX unless otherwise specified.
- For Programming Assignments, please share your Google Colab link or Jupiter Notebook through Github.

1 Problems on Autoencoder

1.1 Problem 1

What is Denoising Autoencoder ? Write the Pseudocode for Denoising Autoencoder training loop.

1.2 Problem 2 (Coding)

Build the model for the patch-based multilayer perceptron (MLP) denoising autoencoder. Take MNIST dataset and only add noise to the randomly chosen 8x8 patch as shown in figure1. Finally design and train an MLP denoising autoencoder to denoise the images with the noisy patches.

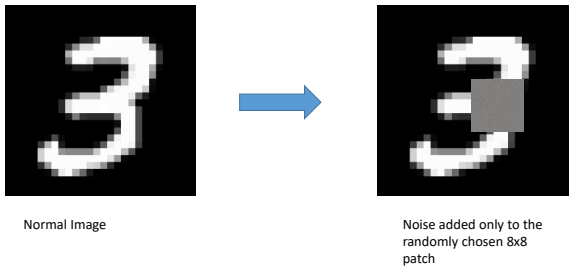


Figure 1: Normal image on left and noise added to randomly chosen 8x8 patch on the right

2 Problems on Generative Adversarial Networks

2.1 Problem 1

A typical Generative Adversarial Network (GAN) uses a Binary Cross Entropy (BCE) loss. A theoretical study by the authors showed that if such a loss is used then the objective of GANs can be reduced to the minimizing of the Kullback–Leibler (or Jensen Shannon) divergence between the distributions of real and generated data.

1. Can the use of BCE loss lead to the unstable training of GAN by being saturated? If yes, then can a use of Least Square (LS) loss mitigate the issues?

2. If LS loss is used in GANs then how can you write the objective function? Further, how such an objective function can be reduced to a divergence measure similar to BCE?

2.2 Problem 2 (Coding)

Generate a training set for one very simple GAN model by drawing random samples using numpy library and then generating the second coordinate using a simple quadratic function for simplicity.

- Now implement the Generator and Discriminator networks preferrably using tensorflow layers. The Generator is a fully connected neural network of 2 hidden layers with given number of nodes. The output of this function is a 2-dimensional vector which corresponds to the dimensions of the real dataset that we are trying to learn. There should be 3 hidden layers for the Discriminator out of which the first 2 layers size you should take as input. You may fix the size of the third hidden layer to 2, so that one can visualize the transformed feature space in a 2D plane. You may use *leaky ReLU* for appropriate activation functions at the hidden nodes as per your discretion. You may use use RMSProp Optimizer for both the networks with the learning rate as 0.001 or *Adam*.

- Plot the training losses at a fixed interval of training iterations. Your plot should indicate how changes in loss decrease gradually and that loss becomes almost constant towards the end of training. Such negligible change in the loss of both Discriminator and Generator indicates equilibrium for this extremely simple GAN.

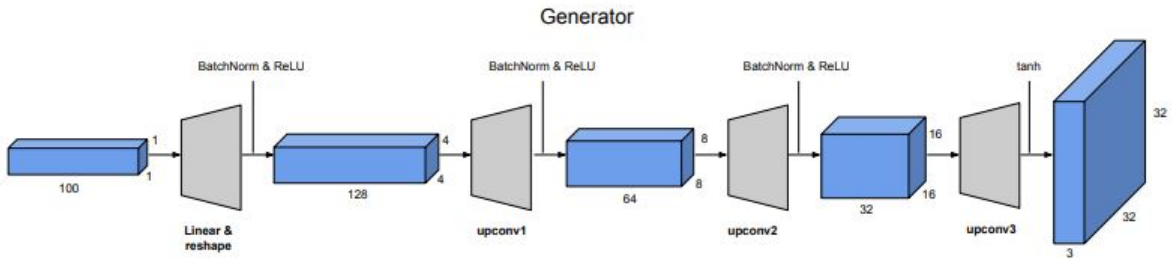
- Plot the real and generated samples after every 1000 iterations of training. These plots visualize how the Generator network starts with a random initial mapping between the input and dataset vector space and then it gradually progresses to resemble the real data samples. As you should see, the "fake" sample starts looking more and more like the "real" data distribution.

2.3 Problem 3 (Coding)

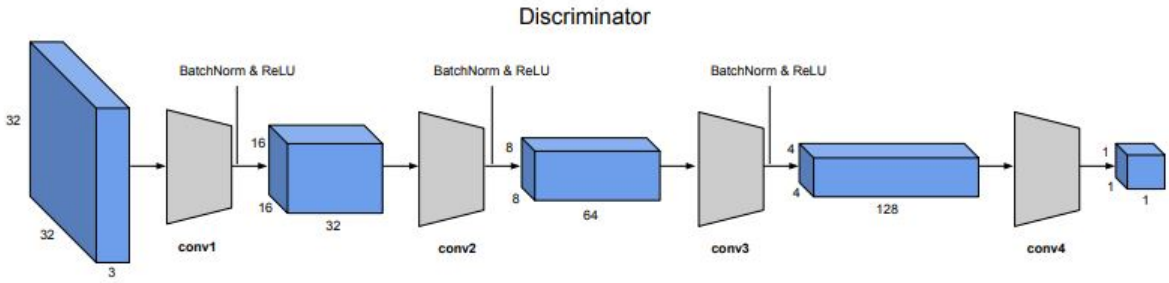
In this assignment, you will implement a Deep Convolutional GAN (DCGAN). A DCGAN is simply a GAN that uses a convolutional neural network as the discriminator, and a network composed of transposed convolutions as the generator. To implement the DCGAN, we need to specify three things: 1) the generator, 2) the discriminator, and 3) the training procedure.

You will train a DCGAN to generate emojis. Architecture of Generator and Discriminator along with the training loop you will be using are as follows:

- **Generator:** Consists of a sequence of transpose convolutional layers that progressively upsample the input noise sample to generate a fake image. The generator has the following architecture:



- **Discriminator:** The discriminator in this DCGAN is a convolutional neural network that has the following architecture:



- **Training Loop;** Next, you will implement the training loop for the DCGAN. A DCGAN is simply a GAN with a specific type of generator and discriminator; thus, we train it in exactly the same way as a standard GAN. The pseudo-code for the training procedure is shown below.

Algorithm 1 GAN Training Loop Pseudocode

procedure TRAINGAN

Draw m training examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from the data distribution p_{data}

Draw m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from the noise distribution p_z

Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \ldots .m\}$

Compute the (least-squares) discriminator loss:

$$J^{(D)} = \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(x^{(i)}) - 1\right)^2\right] + \frac{1}{2m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)}))\right)^2\right]$$

Update the parameters of the discriminator

Draw m new noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from the noise distribution p_z

Generate fake images from the noise: $G(z^{(i)})$ for $i \in \{1, \ldots .m\}$

Compute the (least-squares) generator loss:

$$J^{(G)} = \frac{1}{m}\sum_{i=1}^{m}\left[\left(D(G(z^{(i)})) - 1\right)^2\right]$$

Update the parameters of the generator

Experiments you will be performing:

1. Train a DCGAN to generate emojis. You should run training loop for 20000 iterations (should take approximately half an hour on Colab). Your script should saves the output of the generator

for a fixed noise sample every 200 iterations throughout training; this allows you to see how the generator improves over time.

2. To assess the quality of images created by a generative model store around 3000 generated image and evaluate the FID (Fréchet inception distance) of the generated images with respect to the training images. (Lower the FID better the quality.)

Dataset:
You will be using EmojiOne dataset from
https://joypixels.com/download, or
download from this drive link:
https://drive.google.com/file/d/1APkxEWu8Is7pZ-0tCjL8CabSRnyI6IRE/view?usp=sharing
After downloading the zip file, unzip it and use images of size 32x32, i.e. use images in the folder "joypixels-7.0-free\png\unicode\32".

If we recall, variational inference (VI) in simpler words can be defined as a statistical technique that is used for approximating complex distributions through a known one. To elaborate, if we have a parametrized family of distribution available to us then given an unknown distribution we can approximate it by minimizing its divergence (for example KL) from the family. For example, our family of distributions can be Gaussians with different mean and standard deviations. Now we know in the case of VAE we apply VI to approximate the probabilistic encoder $p(z|x)$ with a Gaussian distribution $q_x(z) \equiv \mathcal{N}(g(x), h(x))$, where g and h are the two functions respectively controlling the mean and the standard deviation. Evidently, we seek to find g and h by gradient descent on the KL divergence between $p(z|x)$ and $q_x(z)$. 1. Show that the KL divergence between $p(z|x)$ and $q_x(z)$ can be expressed as the objective function of VAE if we assume $p(x|z) \equiv \mathcal{N}(f(x), CI)$. You may assume $p(z) \equiv \mathcal{N}(0, 1)$. 2. In the context show how reparametrization trick is necessary to find g and h in practice. 3. How can we relate this formulation to the case where we want to maximize the probability of generating the real data (or $E(logp(x|z))$ while keeping the distance between the real and estimated distributions small, under a threshold ϵ. Hint: you may consider this as a constrained optimization problem.

3 Problems on Variational Auto-Encoder

3.1 Problem 1 (Coding)

1. Obtain (or write! but this isn't required) a Python (tensorflow preferrable) code for a variational autoencoder (VAE). Train this autoencoder on the MNIST dataset. Use only the MNIST training set.?

2. We now need to determine how well the codes produced by this autoencoder can be interpolated.

   - For 10 pairs of MNIST test images of the same digit, selected at random, compute the code for each image of the pair. Now compute 7 evenly spaced linear interpolates between these codes, and decode the result into images. Prepare a figure showing this interpolate. Lay out the figure so each interpolate is a row. On the left of the row is the first test image; then the interpolate closest to it; etc; to the last test image. You should have a 10 rows and 9 columns of images.
   - For 10 pairs of MNIST test images of different digits, selected at random, compute the code for each image of the pair. Now compute 7 evenly spaced linear interpolates between these codes, and decode the result into images. Prepare a figure showing this interpolate. Lay out the figure so each interpolate is a row. On the left of the row is the first test image; then the interpolate closest to it; etc; to the last test image. You should have a 10 rows and 9 columns of images.

4 Questioning the quality of the generated samples

4.1 Problem 1

Two standard techniques to quantify the quality and diversity of the generated image samples are by Inception Score and Frechet inception distance. Can you define the two ideas and briefly discuss their pros and cons over each other?