CPSC 8810 FINAL PROJECT REPORT

VENKATESH VELIDIMALLA

Project Title : Generative models using Variational Autoencoders (VAEs) & Generative Adversarial Networks (GANs)

ABSTRACT

To truly understand the visual world our models should be able not only to recognize images but also generate them. Deep Learning models perform admirably in supervised learning but not in unsupervised learning.Generative Models have shown to be effective in resolving such issues.
This project aims at applying generative adversarial networks and variable autoencoders(VAE), and aims to generate novel images (GANs). We aim to produce novel images by first implementing a conditional autoencoder and a variational autoencoder with slightly different architectures and applying them to the MNIST handwritten dataset and CIFAR 10 dataset , along with an implementation of GANs in which we attempt to maximize the probability of the discriminator making the wrong choice.

1. INTRODUCTION

1) Generating new images from another is one of the challenging tasks in computer vision. In the last few years deep learning base generative models have grown popular because of its ability to handle large amounts of data, well-designed networks architectures and smart training techniques. They produce highly realistic pieces that may include images , texts and sound. The two most common models that stand out are Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). VAE is an autoencoder whose encodings distribution is regularized during the training in order to ensure that its latent space has good properties allowing us to generate some new data.Variational Autoencoders are able to generate images by tweaking the latent space. Autoencoders consist of an encoder, decoder and a latent space, which are trying to learn the best encoding-decoding scheme using an iterative optimisation algorithm.

2) Generative Adversarial Networks (GANs) belong to the set of generative models.The ability to generate new content makes GANs interesting. Given a training set, this technique learns to generate new data with the same statistics as the training set. It is a good generative model as it distinguishes noise from data as well.
Variational Autoencoders are constructed on the base of autoencoders.When we talk about autoencoders, dimensionality reduction comes into play. Dimensionality reduction is the process of reducing the number of features that describe some data.The main purpose of a dimensionality reduction method is to find the best encoder/decoder pair among a given family.In other words, for a given set of possible encoders and decoders, we are looking for the pair that keeps the maximum of information when encoding and, so, has the minimum of reconstruction error when decoding

3) . We further go on to train VAEs and GANs on the MNIST dataset to generate Novel Images. The MNIST dataset has 60,000 training and 10,000 test images. Once trained, we use the CIFAR10 dataset to test all the networks. It has over 20,000 different categories. We use these two datasets as they find wide use in testing neural networks employed for image generation. Finally, we will evaluate performance of all the implemented models using Fréchet Inception Distance (FID) and Kernel Inception Distance (KID).

2. RELATED WORK

1. Autoencoders are used to compress data by transforming it from a higher to a lower-dimensional space. An encoder and a decoder comprise the autoencoder. The encoder compresses data from a higher-dimensional space to a lower-dimensional space (also known as the latent space), whereas the decoder reverses the process, converting the latent space back to higher-dimensional space. By forcing it to output what was fed as input to the decoder, the decoder ensures that latent space can capture the majority of the information from the dataset space. Autoencoder does not regularize the latent variable. and it has discontinuity. Moreover, It generates garbage output when a random latent variable is chosen. Along with that latent space lacks the generative capability.

2. The issue of non-regularized latent space in autoencoder is addressed by variational autoencoder, which provides generative capability to the entire space. The AE encoder generates latent vectors. Instead of vectors in the latent space, the VAE encoder outputs parameters from a predefined distribution in the latent space for each input. The VAE then applies a constraint to this latent distribution, forcing it to be normal. This constraint ensures the regularization of the latent space.
3. When compared to VAE, GANs produce better results. GANs produce more photorealistic images, but they are more difficult to work with.

3. METHODS

Variational Autoencoders (VAEs): VAE's architecture consists of two components: encoder and decoder. Input data is encoded into hidden states by the encoder. In other words, the encoder projects the input data onto a lower-dimension vector, each element of which has its own distribution. Then, from such distributions, a vector is sampled to obtain a specific input (hidden state) to the decoder. Finally, the decoder attempts to decode its input to the encoder's input data. Unlike GAN, which has predefined hidden state distributions, VAE learns the hidden state distributions p(z) during the process.

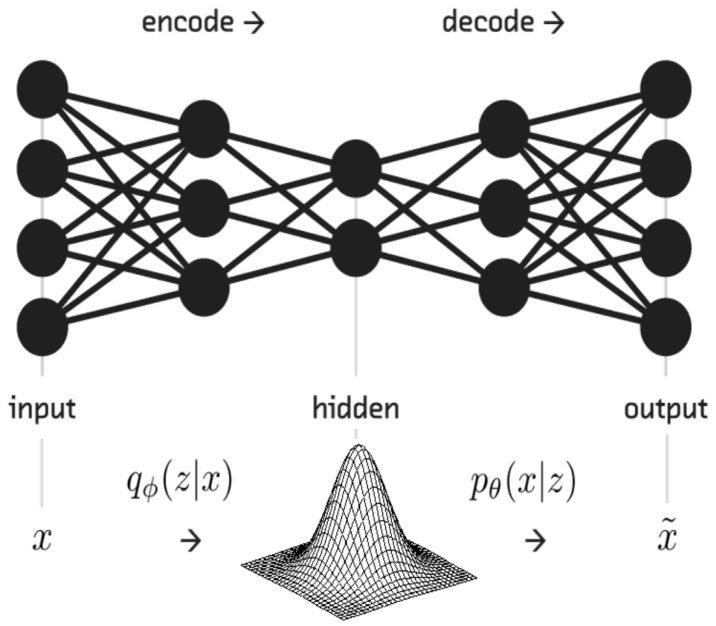


Fig. VAE Architecture

The VAE value function looks like this:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \mathbf{x}^{(i)}) \simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2\right) + \frac{1}{L} \sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)} | \mathbf{z}^{(i,l)})$$

where $\mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)}$ and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$

While a VAE learns to encode the given input (say, an image) and then reconstructs it from the encoding, a GAN works to generate new data which can't be distinguished from real data. The key behind their operation is that in case of an autoencoder, we use the latent representations generated by an encoder for various tasks.However, in case of a GAN we work on the generated 'fake' images which are not so fake. And thanks to the joint effort of the discriminator which reprimands the generator till it learns to generate the 'real - fake' images!Variational Autoencoders use KL-divergence as their loss function. The aim is to minimize the difference between the supposed and original distribution of the dataset.

Now we would be implementing various Fully Connected Variational Autoencoders (FCVAE) and Conditional Variational Autoencoders (CVAE).

FCVAE: First implementation of VAE is fully connected layers. We'll take the 1x 28 x 28 shape of our input and flatten the features to create an input dimension size of 784. Encoder will take our images as input and pass them through three Linear+ReLU layers. With the help of reparameterization trick in order to estimate the posterior zz during our forward pass, given the μμ and σ2σ2 estimated by the encoder. The forward pass should pass the input image through the encoder to calculate the estimation of mu and logvar, reparametrize to estimate the latent space z, and finally pass z into the decoder to generate an image.

FCVAE Encoder Architecture:
- Flatten (Hint: nn.Flatten)
- Fully connected layer with input size 784 (input_size) and output size H
- ReLU
- Fully connected layer with input_size H and output size H
- ReLU
- Fully connected layer with input_size H and output size H
- ReLU

FCVAE Decoder Architecture:
● Fully connected layer with input size as the latent size (Z) and output size H
● ReLU
● Fully connected layer with input_size H and output size H
● ReLU
● Fully connected layer with input_size H and output size H
● ReLU
● Fully connected layer with input_size H and output size 784 (input_size)
● Sigmoid
● Unflatten (nn.Unflatten)

Conditional Variational Autoencoder (CVAE):
It is a Variational Autoencoder extension. Both the encoder and the decoder of the conditional variational autoencoder have an additional input. We will condition our latent space and image generation on the class here. The CVAE architecture will be similar to our FC-VAE architecture, with the addition of a one-hot label vector to both the x input and the z latent space.
Generative Adversarial Networks (GANs):
The architecture of GAN is made up of two parts: the generator and the discriminator. GAN learns by engaging in a "minimum-maximum two-player game." Its generator generates fake images that can heuristically fool the discriminator. Its discriminator tries to tell the difference between real and fake images.

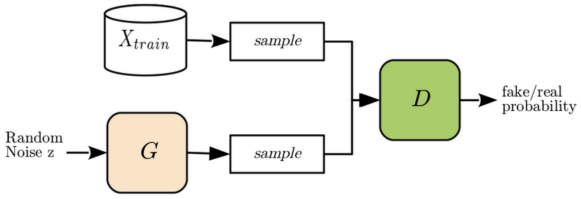
The GAN loss function looks like this:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}))$$

where x is a real-world image whose distribution is affected by pdata (x). z is a random vector (or hidden state) whose distribution follows a predefined distribution pz (z). D(x) represents the discriminator, and its output represents its confidence that the input is a real image. G(z) stands for generator and produces images. When D(x) returns a higher confidence rate for x, the first term in the GAN value function gains value. In other words, if D(x) is more confident in classifying real images as "real," we are pleased. The second term increases if D(x) is less certain that the image generated by G(z) is "real." Furthermore, the second term decreases if G(z) generates more realistic images, which may make D(x) more confident in classifying them as "real."

Now we would be implementing various Fully Connected Generative Adversarial Networks(FC-GAN), Least Squares GAN (LS-GAN) and Deeply Convolutional GAN (DC-GAN).

Fully Connected Generative Adversarial Networks will have two networks. Discriminator and generator Discriminator will examine images and determine whether they are real or fake. Generator will take random noise as input and transform it into images using a neural network. The generator's goal is to trick the discriminator into thinking the images it generated are real.To optimize this minimax game, I will alternate between taking gradient descent steps on the objective for *G* , and gradient ascent steps on the objective for *D*.



Architecture of discriminator:
Fully connected layer with input size 784 and output size 256
LeakyReLU with alpha 0.01
Fully connected layer with input_size 256 and output size 256
LeakyReLU with alpha 0.01
Fully connected layer with input size 256 and output size

Architecture of Generator:
Fully connected layer from noise_dim to 1024
ReLU
Fully connected layer with size 1024
ReLU
Fully connected layer with size 784
TanH (to clip the image to be in the range of [-1,1]).

Least Squares GAN (LSGAN): Now we will retrain the model by changing the loss function. LSGANs have two advantages over conventional GANs. First, compared to regular GANs, LSGANs are able to produce images of higher quality. Second, LSGANs exhibit more consistent performance while learning. Deeply Convolutional GAN (DC-GAN): Now we further go on to reimplement the model by adding convolutional networks.

4. EXPERIMENTS

Hyperparameters for VAE:
Optimizer: Adam optimizer
learning rate: 1e-3
num_epochs = 10
latent_size = 15
input_size = 28*28(MNIST), 3*32*32(CIFAR10)
Hyperparameters for GAN
Optimizer: Adam optimizer
batch_size = 128
NOISE_DIM = 96
num_epochs = 10
learning rate: 1e-3
input_size = 28*28(MNIST), 3*32*32(CIFAR10)

Datasets: The MNIST database contains a large collection of handwritten images. It is commonly used to train and test machine learning and deep learning models. There are 60,000 training images and 10,000 testing images in total. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class.
Frechet Inception Distance (FID) and Kernel Inception Distance (KID) are two commonly used metrics for evaluating VAEs. Inception score and Frechet Inception can also be used for GANs.

We trained the models VAE and GAN on both MNIST and CIFAR dataset and submitted the results and comparisons we obtained ,but failed to evaluate and execute using FID and KID.

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2672–2680.aa

[2] Kingma, D. P., & Welling, M. (2013). Auto-Encoding Variational Bayes. arXiv: Machine Learning. Retrieved 11 19, 2022, from https://arxiv.org/abs/1312.6114

[3] https://wandb.ai/shambhavicodes/vae-gan/reports/An-Introduction-to-VAE-GANs--VmlldzoxMTcxMjM5

[4] https://towardsdatascience.com/deep-generative-models-25ab2821afd3

[5] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative Adversarial Nets. Retrieved 11 19, 2022, from https://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[6] https://towardsdatascience.com/generative-adversarial-networks-gans-a-beginners-guide-f37c9f3b7817#: ~:text=We%20can%20make%20the%20neural,which%20still%20gives%20high%20accuracy