Deep Generative Models with focus on StackGAN

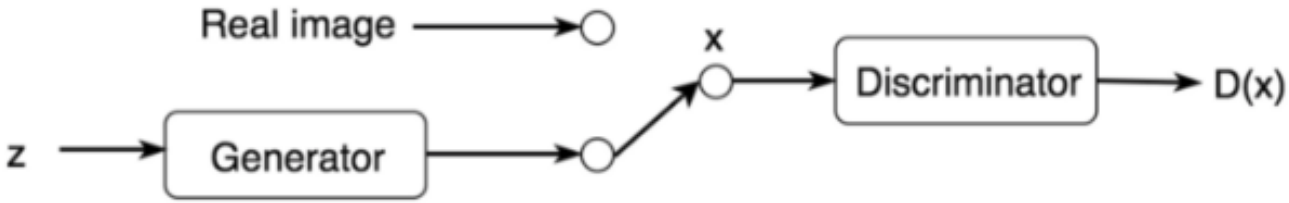
Gururaj K, Sakya Basak, Rahul Bansal, Rajat Nagpal

IISc Bangalore

Introduction

1. Generative Models models P(Y, X) or joint probabilty of class and observed data.
2. Modelling joint distribution allows us to generate (X, Y) pairs of high probability.
3. It is difficult to model joint distribution because of many intractable probabilistic computations that arise in maximum likelihood estimation and related strategies.
4. In 2014, Ian J. Goodfellow proposed a new framework for estimating generative models via an adversarial process, in which they described procedure to train two models: Generator and Discriminator.

GAN



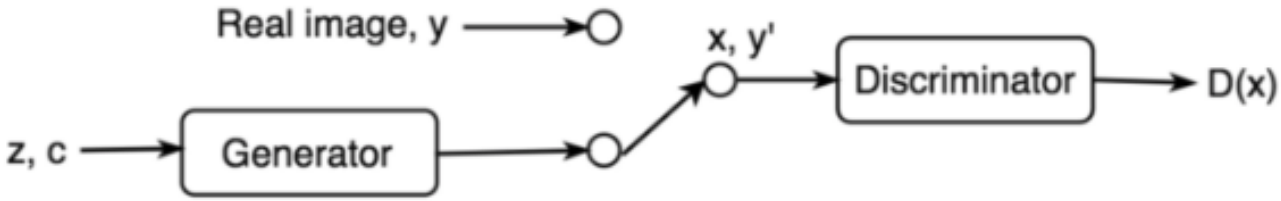
$$\max_D V(D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

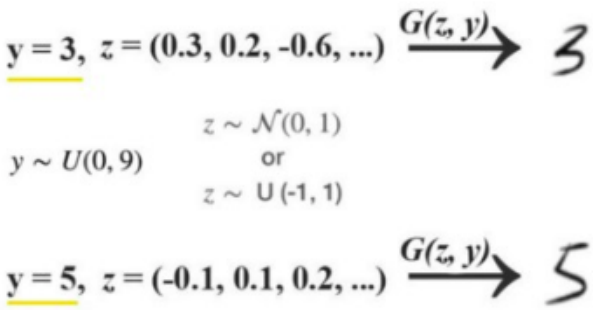
recognize real images better — recognize generated images better

$$\min_G V(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

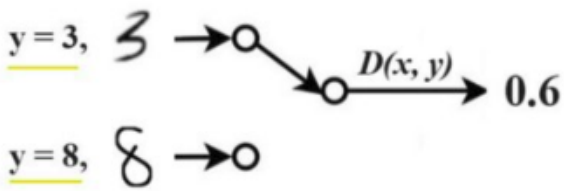
Optimize G that can fool the discriminator the most.

Conditional GAN





Generator



Discriminator

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

DCGAN Architecture

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12544)	1254400
batch_normalization_v1 (Batc	(None, 12544)	50176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTran	(None, 7, 7, 128)	819200
batch_normalization_v1_1 (Ba	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	204800
batch_normalization_v1_2 (Ba	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 1)	1600

Total params: 2,330,944
Trainable params: 2,305,472
Non-trainable params: 25,472

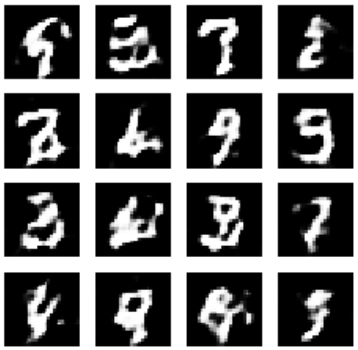
Generator Summary

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273

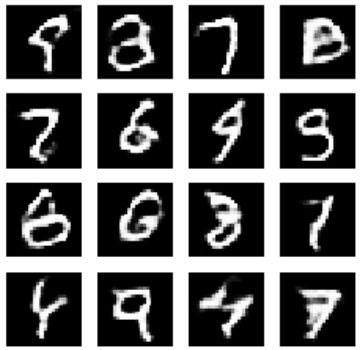
Total params: 212,865
Trainable params: 212,865
Non-trainable params: 0

Discriminator Summary

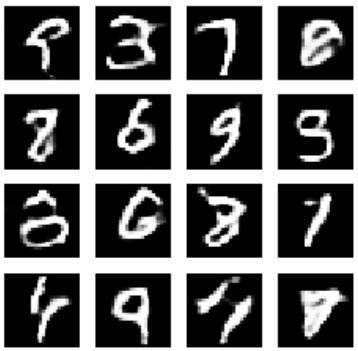
Results



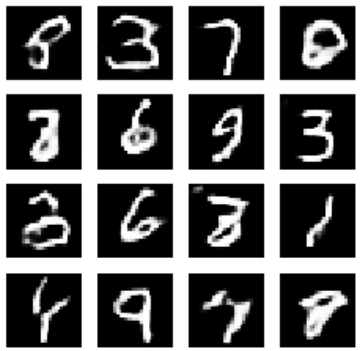
After 25 epochs



After 50 epochs



After 75 epochs



After 100 epochs

VAE:introduction

1 How it works

- VAEs are powerful generative models which is used to create variations of input data in a desired format.Contrary to vanilla autoencoders their latent space is continuous.
- Given an input, we feed it to an encoder which generates a latent space representation of the data.This latent space is usually of a smaller dimension than original input.
- The encoder outputs a probability distribution of the latent attributes .The decoder then samples values from the distribution of the attributes to generate outputs.

VAE: Equations

- Output x as seen from vae is generated from a latent variable z. We try to approximate $p(x|z)$ from $q(x|z)$ and try to minimize the KL Divergence.
- Once we find q we try to optimize a loss function.
  $L_B(\theta, \phi, x^i) = \frac{\sum_{i=1}^{L}(logp_\theta(x^i|z^{(i,l)}))}{L} - KL(q_\phi(z|x^i)||p_\theta(z))$
- The first term consists of a reconstruction error and a second term encourages learned distribution $q(z|x)$ to be equal to $p(z)$ which is chosen as unit Gaussian.
- To train the weights of the encoder and decoder networks we use backpropagation, but since there is random sampling involved we use a **reparameterization trick.**

Implementation and Results

- Code implemeneted using tensorflow.
- Encoder consists of 2 convolutional layers followed by a fc layer.
- Decoder consists of a fc layer followed by 3 Convolutional transpose layers.
- Parameterization trick used to allow gradients to backpropagate through sample to encoder network parameters.
- Trained for 100 epochs. The latent dimension size has been kept at 16.

Figure: Generation Results

Stacked GAN

1. StackGAN is a Conditional GAN based model proposed to synthesize high-quality images from text discriptions.
2. It decomposes the text-to-image generative process into two stages.
3. Stage-I GAN is a Conditional GAN conditioned on the context vector generated from text embedding. It sketches the primitive shape and colors of the object based on the given text description, yielding Stage-I low-resolution images.
4. Stage-II GAN is also a Condtional GAN conditioned on Stage-I results and text embedding. It generates high-resolution images with photo-realistic details.

Motivation for the approach

- Can't do in one stage because of multiple things going on beneath the hood its too complicated .
- Can't do multiple stages because training such a model is tough.
- Incoherence in optimization :Multiple stages trained on different objective functions,which may result in incoherence in optimization, where each module is not trained to match other modules.

Text Embedding

Text embedding are extracted using deep convolutional and recurrent text encoders that learn a correspondence function with images by optimizing the following structural loss.

$$\frac{1}{N}\sum_{i=1}^{N}\Delta(y_n, f_v(v_n)) + \Delta(y_n, f_t(t_n))$$

where (v_n, t_n, y_n) are image,text and class label of a data sample,Δ is 0-1 loss. f_v and f_t are parametrized as follows.

$$f_v(v) = argmax_{y\in Y} E_{t\sim T(y)}[\phi(v)^T \varphi(t)]$$
$$f_t(t) = argmax_{y\in Y} E_{v\sim V(y)}[\phi(v)^T \varphi(t)]$$

where ϕ is the image encoder using CNN and φ is the text encoder using LSTM.

Conditioning Augmentation

Conditioning Augmentation technique is proposed wherein the latent variable $\hat{c}$ is sampled from $\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))$ where μ and Σ which are functions of embedding $\varphi(t)$ are jointly learnt along with rest of the network.

Stage-I

Stage-I GAN is a conditional GAN with conditioning variable $\hat{c}_0$ sampled from $\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t))$. Stage-I GAN trains the Generator G_0 and Discriminator D_0 by alternatively maximizing $\mathcal{L}_{D_0}$ and minimizing $\mathcal{L}_{G_0}$:

$$\mathcal{L}_{D_0} = E_{(I_0,t)\sim pdata}[log\ D_0(I_0, \varphi_t)] + \mathrm{E}_{(z\sim p_z, t\sim pdata)}[log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))]$$

$$\mathcal{L}_{G_0} = E_{(z\sim p_z, t\sim pdata)}[log(1 - D_0(G_0(z, \hat{c}_0), \varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t))||\mathcal{N}(0, I))$$

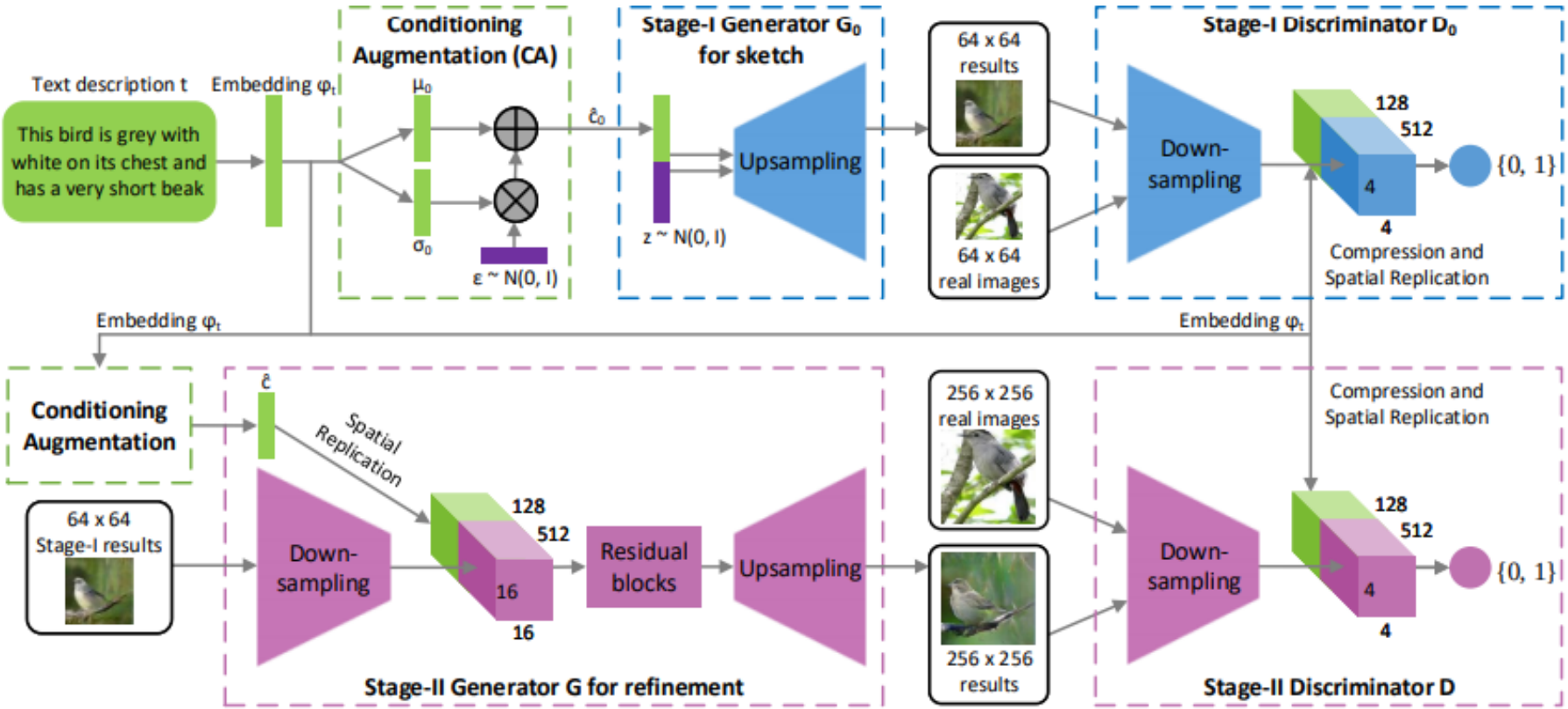
Stage-II

Stage-II GAN is also a conditional GAN which conditions on low-resolution result $s_0 = G_0(z, \hat{c}_0)$ from Stage-I and latent variables $\hat{c}$. Stage-II GAN trains the Generator G and Discriminator D by alternatively maximizing $\mathcal{L}_D$ and minimizing $\mathcal{L}_G$:

$$\mathcal{L}_D = E_{(I,t)\sim pdata}[log\ D_0(I, \varphi_t)] + E_{(s_0\sim p_{G_0}, t\sim pdata)}[log(1 - D(G(s_0, \hat{c}), \varphi_t))]$$

$$\mathcal{L}_G = E_{(s_0\sim p_{G_0}, t\sim pdata)}[log(1 - D(G(s_0, \hat{c}), \varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu(\varphi_t), \Sigma(\varphi_t))||\mathcal{N}(0, I))$$

Architecture



Implementation Details

Upsampling

- The up-sampling blocks consists of nearest neighbor upsampling followed by 3x3 stride 1 convolution.
- Batch Normalization and ReLu are applied at every stride except last one.

Residual Blocks

- They have the same implementation as above.
- Two residual blocks are used in 128x128 StackGAN Models while four are used in 256x256 models.

Down Sampling

- consists of 4x4 stride 2 convolutions,Batch Normalization and Leaky ReLU.

Training

- We iteratively train the generator and discriminator for Stage 1 GAN for 600 epochs by fixing stage 2.
- We do the same for Stage 2 GAN for 600 epochs by fixing stage I GAN.
- All networks are trained using ADAM Solver with batch size 64 and an initial learning of 0.00002.

Ablation Experiments: Conditioning Augmentation

Conditioning Augmentation

- It has been seen that StackGAN without CA produces blurred unrealistic images as claimed in paper.
- In CA, instead of single Gaussian we use mixture of 3 Gaussians to produce samples c from text embedding $\phi(t)$
- The parameters of the Gaussians are learnt via reparameterization trick using FC layers. The KL divergence regularization term for each of these gaussians is added to loss function while training Generators.
- **Results seen** : We do not notice any perceivable change maybe because all the Gaussian parameters generated are almost identical.

Conditioning Augmentation Results

This bird has wings that are black and has a white belly

Without_CA

With CA

CA with gaussian mixture

Ablation Experiments

Generator Upsampling Block

- The original StackGAN model uses nearest neighbour upsampling block with scale factor 2.But we notice the resulting images are slightly pixelated.
- In order to mitigate this problem we experiment with bilinear and bicubic upsampling block.
- **Results seen** : The images obtained are smoother.

Bilinear Upsampling Results

with nearest neighbour upsampling

with bilinear upsampling

Scope of future Work

- Use SRGAN as second stage
- Use attention mechanisms in text input.
- generate multimodal images

Thank You