Diffusion Model

Nandini Lokesh Reddy

Diffusion model: Definition

- Diffusion models are computer learning algorithms that produce visuals for a given text description.
- These can generate any type of image that we can think of.

There are other working models that are comparable to the diffusion model, including:

Generative adversarial networks

Variational auto encoders

Flow based models.

Diffusion model application:

- Image generation
- Image Denoising
- Inpainting
- Out painting
- Bit diffusion

Some other application that implemented diffusion model are:

1. Dall-E2
2. Dream
3. Studio.

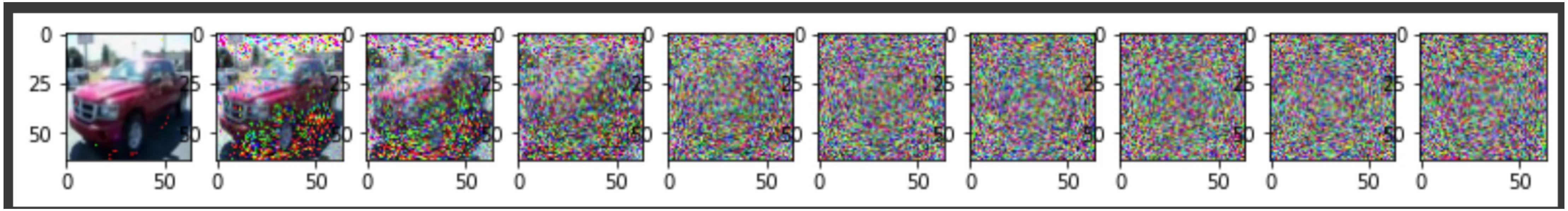
Prompts

- Prompts are how you can control the outputs for diffusion model.

Components of Prompt:

Frame, Subject, Style, Optional Seed.

- Diffusion model uses a batch of input photos and adds noise to each one (forward process).
- The original image is kept from the added noise image using a parameterized backward technique.
- Fig. 1: A automobile image is captured, and then noise is repeatedly added until the image is completely noise-loaded.



- Then, after applying backward processing to the noise image, we obtain the required new image with the desired characteristics.

Implementation

- Building diffusion model consists of 3 steps:
- A. Noise scheduler(forward process)
- B. Neural Networks(backward process)
- C. Timestep encoding.

Step-1: Noise Scheduler

- Forward model: Process of converting the original image to a noise-added image.

The forward model is implemented based on MarKov Process:

Where the noise that can be added depends only on the previous image.

Equation:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Where:

X_0 is the initial input ,

and x_{t-1} is the previous image

All other x’s represent the noisy version of X.

- Noise is sampled by Conditional Gaussian Distribution:

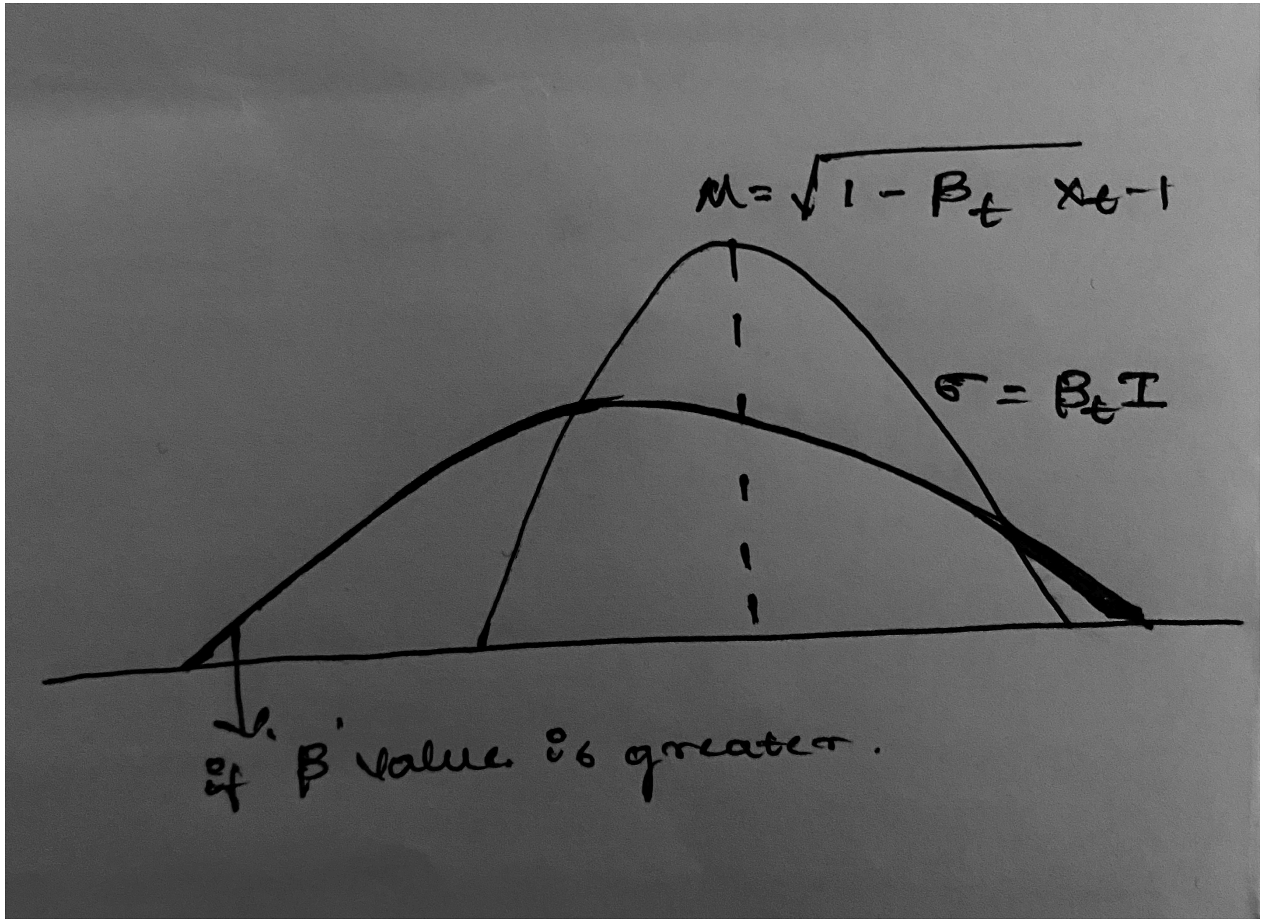
$$q(x_t|x_{t-1}) = N\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right)$$

Where,

X_t is the output, $\sqrt{1} - \beta x t - 1$ is the mean and βtI is the variance.

Beta represents the noise level.

- If the beta size is larger then the pixel distribution is wider and more shifted. This results in more corrupted image.
- How much noise should be added to each image can be precalculated.



```python
def linear_beta_schedule(timesteps, start=0.0001, end=0.02):
  return torch.linspace(start, end, timesteps)

#to get the index from the list with the batch size
def get_index_from_list(vals, t, x_shape):
  batch_size=t.shape[0]
  out=vals.gather(-1,t.cpu())
  return out.reshape(batch_size, *((1,)  *(len(x_shape)-1))).to(t.device)

def forward_diffusion_sample(x_0, t, device="cpu"):
  noise=torch.randn_like(x_0)
  sqrt_alphas_cumprod_t = get_index_from_list(sqrt_alphas_cumprod, t, x_0.shape)
  sqrt_one_minus_aplhas_cumprod_t = get_index_from_list(
      sqrt_one_minus_alphas_cumprod, t, x_0.shape
  )

#mean and variance
  return sqrt_alphas_cumprod_t.to(device) * x_0.to(device)\
  + sqrt_one_minus_aplhas_cumprod_t.to(device) * noise.to(device),noise.to(device)

#beta schedule
T=150
betas=linear_beta_schedule(timesteps=T)

#precaclulate different terms for closed form.
alphas=1. - betas
alphas_cumprod=torch.cumprod(alphas, axis=0)
alphas_cumprod_prev=F.pad(alphas_cumprod[:-1], (1,0), value=1.0)
sqrt_recip_alphas=torch.sqrt(1.0/alphas)
sqrt_alphas_cumprod=torch.sqrt(alphas_cumprod)
sqrt_one_minus_alphas_cumprod=torch.sqrt(1. - alphas_cumprod)
posterior_varience=betas * (1.-alphas_cumprod_prev)/(1.-alphas_cumprod)
```

Figure2: code for implementing the noise to the image

```python
IMG_SIZE=64
BATCH_SIZE=128
#converts pillow image to tensor image
def load_transformed_dataset():
  data_transforms=[
      transforms.Resize((IMG_SIZE, IMG_SIZE)),
      transforms.RandomHorizontalFlip(),#data augumentation(data filtering)
      transforms.ToTensor(),#scales the data into[0,1]
      transforms.Lambda(lambda t: (t *2)-1) #for beta we need data b/w [-1 1]
  ]
  #training and testing the data
  data_transform = transforms.Compose(data_transforms)
  train=torchvision.datasets.StanfordCars(root=".", download=True,
                                          transform=data_transform)
  test=torchvision.datasets.StanfordCars(root=".", download=True,
                                          transform=data_transform, split='test')
  return torch.utils.data.ConcatDataset([train, test]) #merge them into a dataset


#converts tensor image to pillow image [reverse transformation]
def show_tensor_image(image):
  reverse_transforms=transforms.Compose([
      transforms.Lambda(lambda t: (t+1) / 2),
      transforms.Lambda(lambda t: t.permute(1, 2, 0)),
      transforms.Lambda(lambda t: t *255.),
      transforms.Lambda(lambda t:t.numpy().astype(np.uint8)),
      transforms.ToPILImage(),
  ])
  #take 1st image:

  if len(image.shape)==4:
    image=image[0,:,:,:]
  plt.imshow(reverse_transforms(image))
```

Figure3: implementing noise to our dataset image

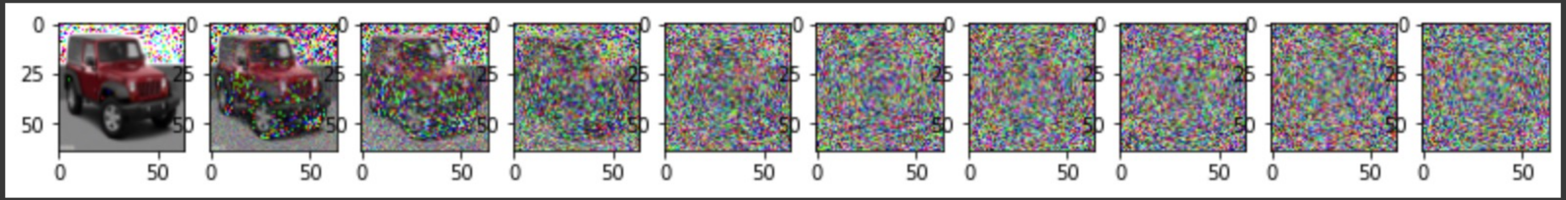
OUTPUT: Noise implementation

```
#simulating forward diffusion:

image = next(iter(dataloader))[0]

plt.figure(figsize=(15,15))
plt.axis('off')
num_images=10
stepsize=int(T/num_images)


for idx in range(0, T, stepsize):
  t=torch.Tensor([idx]).type(torch.int64)
  plt.subplot(1, num_images+1, (idx/stepsize)+1)
  image, noise = forward_diffusion_sample(image, t)
  show_tensor_image(image)
```



Step-2: Neural Network:

- Model takes the 3-channeled(red, green, blue) noisy image and predicts the noise in the image.
- Simpler models like U-Net can be used to predict the noise, as it takes input and predicts the output in the same dimensionality, also useful for image segmentation.
- It takes the image, down sample the data until a bottleneck is reached, and then tensor’s are up sampled again and passed to more convolutional layers.
- The input tensor gets smaller but also deeper as more channel’s are added.

Parameterized backward process: Mechanism

$$p_\theta(x_T) = N(x_t; 0, I)$$

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

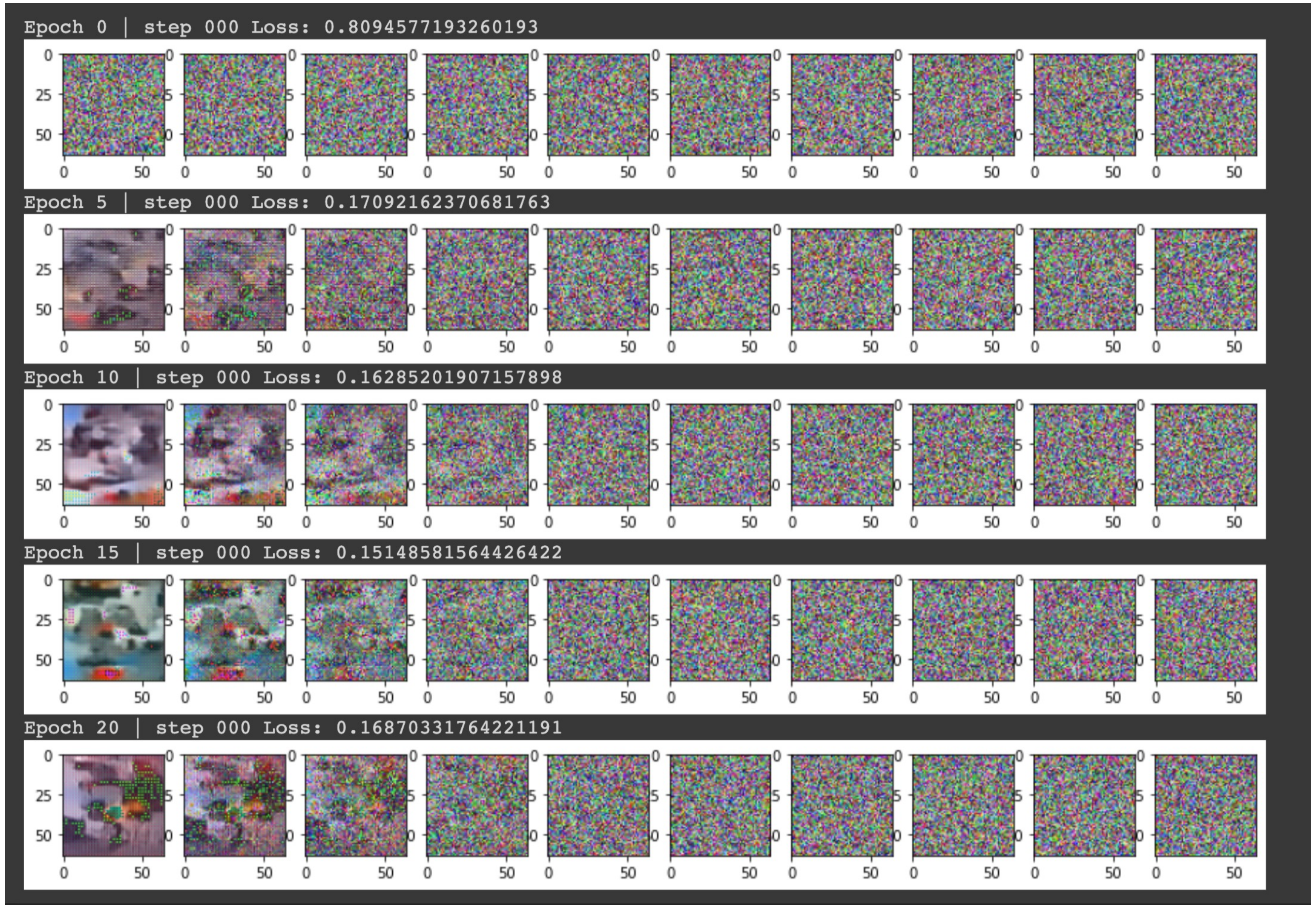
To get the actual image:

$$x_{t-1} \approx x_t \ - noise$$

Step-3: Time Step Encoding

- The neural network can’t distinguish between different time step, as they have shared parameters across time.
- As it needs to filter out the noise from images, with different noise intensities
- So we can use positional embeddings or time step encodings to differentiate different time steps.

After subsequent epochs, the model can print the modified image.



Conclusion:

We can paint a picture in our minds with just a single text, and this is applicable to many fields, including e-commerce and education.

Although we can make whatever image we want with a simple model, if we work on it more, we can make the image more realistic and artistic.

References:

- Denoising Diffusion Probabilistic Models by Jonathan Ho, Ajay Jain, Pieter Abbeel https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf
- Diffusion Models Beat GANs on Image Synthesis by Prafulla Dhariwal, Alex Nichol https://proceedings.neurips.cc/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf
- https://openaccess.thecvf.com/content_cvpr_2018/papers/Xu_AttnGAN_Fine-Grained_Text_CVPR_2018_paper.pdf
- https://github.com/CompVis/stable-diffusion