Assignment 2

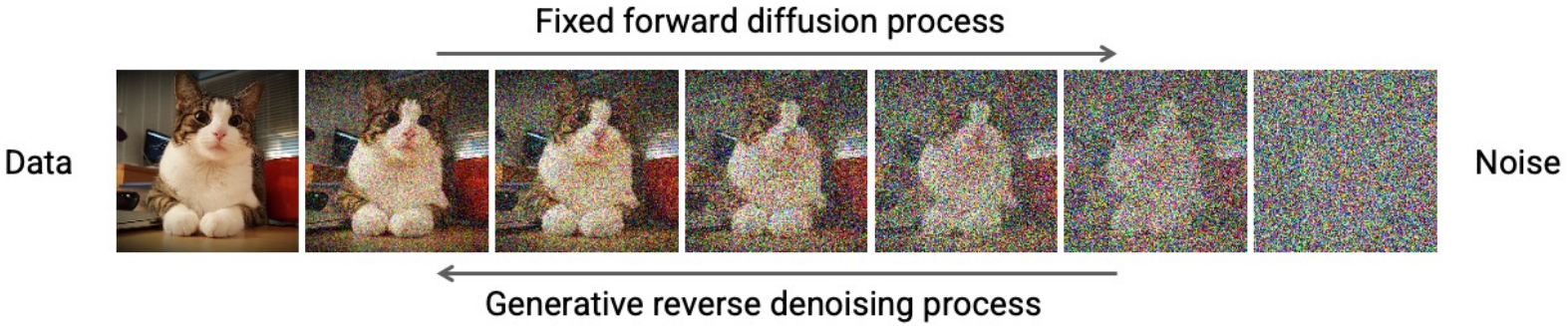
TA: Jaihyun Lew

Department of Electrical and Computer Engineering
Seoul National University

http://data.snu.ac.kr

Diffusion Models

- Denoising Diffusion Models
  - Iteratively removing noise for sample generation.



https://cvpr2022-tutorial-diffusion-models.github.io

Assignment 2-1: Diffusion Models

- Problem 1: Variance (Noise) Scheduling
  - Diffusion Model의 variance schedule (linear, cosine) 구현
- Problem 2: Training & Sampling 구현
  - Training process 구현 (loss)
  - Sampling (noise로부터 이미지 생성) 구현
  - Epsilon prediction, sample (x_0) prediction.
- Problem 3: Training
  - 학습 및 sample generation.
  - 다양한 parameter choice 활용하여 숫자가 명확히 보이면 성공

MNIST dataset

- 28 x 28 black& white image of number digits of 0 ~ 9.
- Consists of 60,000 samples for training & 10,000 for testing.

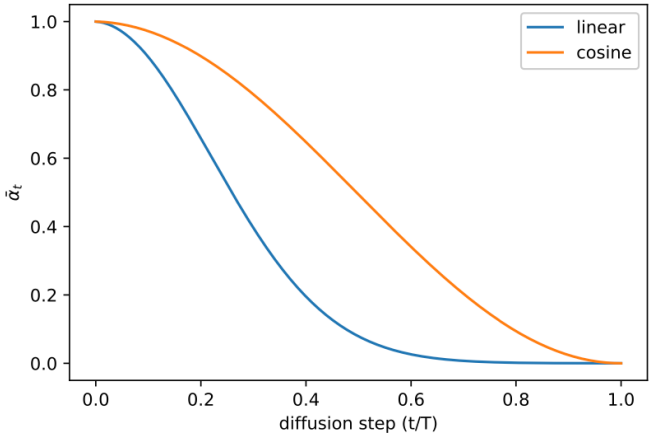
http://yann.lecun.com/exdb/mnist/

2-1-1: Variance Scheduling

- Implement the variance scheduling in diffusion models.
  - Linear & Cosine

- linear schedule (original DDPM):

  e.g. increase β_t linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$



- cosine-based schedule (improved DDPM):

$$\beta_t = \text{clip}(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999), \quad \bar{\alpha}_t = \frac{f(t)}{f(0)}$$

where $f(t) = \cos^2\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)$

  - small offset s: to prevent β_t from being too small when close to $t = 0$

```
[ ]  def beta_schedule(beta1, beta2, T, schedule='linear'):
         ##############################################################################
         #                            IMPLEMENT YOUR CODE                             #
         ##############################################################################

         if schedule == 'linear':
             betas = None            # TODO


         elif schedule == 'cosine':
             betas = None            # TODO
             s = 0.008


         ##############################################################################
         #                            END OF YOUR CODE                                #
         ##############################################################################
         return betas
```

Nichol et al., "Improved denoising diffusion probabilistic models." *ICML*, 2021.

2-1-1: Variance Scheduling

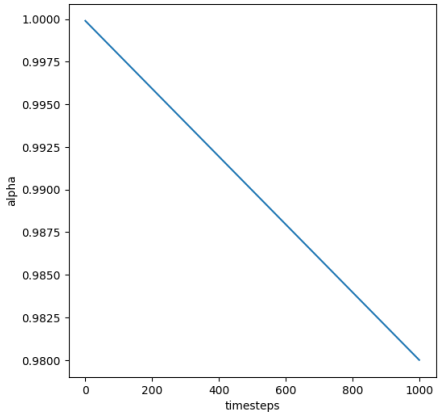
- Pre-compute schedule parameters.
  - $\beta_t,\ \sqrt{\beta_t},\ \ \alpha_t,\ \bar{\alpha}_t,\ \sqrt{\bar{\alpha}_t},\ \frac{1}{\sqrt{\alpha_t}},\ \sqrt{1-\bar{\alpha}_t},\ \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}$

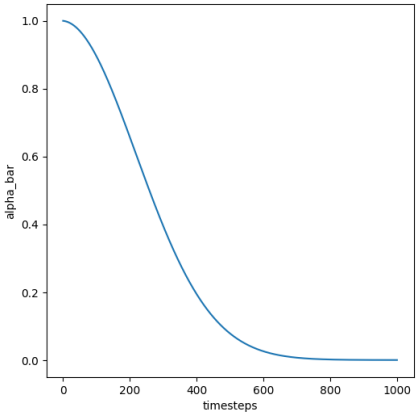
```python
def ddpm_schedules(beta1, beta2, T, schedule='linear'):
    """
    Returns pre-computed schedules for DDPM sampling, training process.
    """
    assert beta1 < beta2 < 1.0, "beta1 and beta2 must be in (0, 1)"


    beta_t = beta_schedule(beta1, beta2, T, schedule)

    ##############################################################################
    #                             IMPLEMENT YOUR CODE                            #
    ##############################################################################

    sqrt_beta_t = None           # TODO
    alpha_t = None               # TODO
    alphabar_t = None            # TODO

    sqrtab = None                # TODO
    oneover_sqrta = None         # TODO

    sqrtmab = None               # TODO
    bt_over_sqrtmab_inv = None  # TODO

    ##############################################################################
    #                             END OF YOUR CODE                               #
    ##############################################################################
```

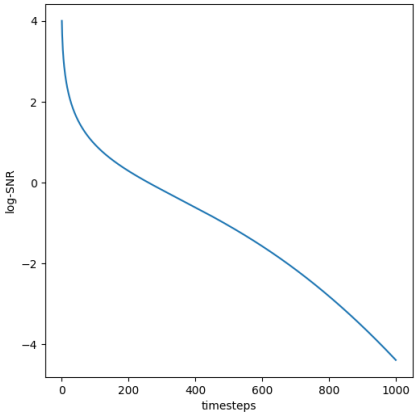
2-1-1: Variance Scheduling

- Check your implementation.
- The graphs should look somewhat like this.

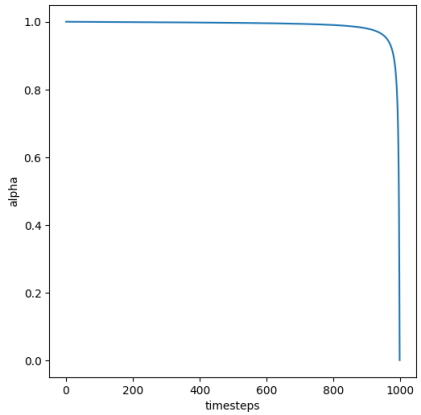
Linear

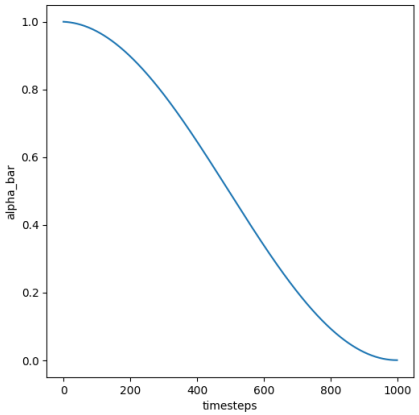


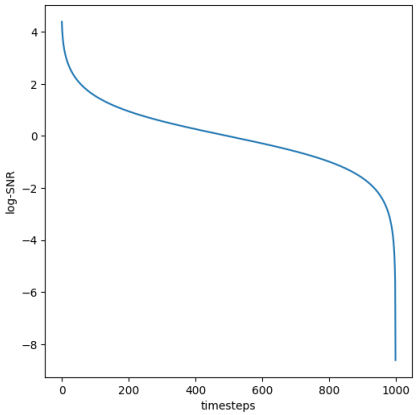




Cosine

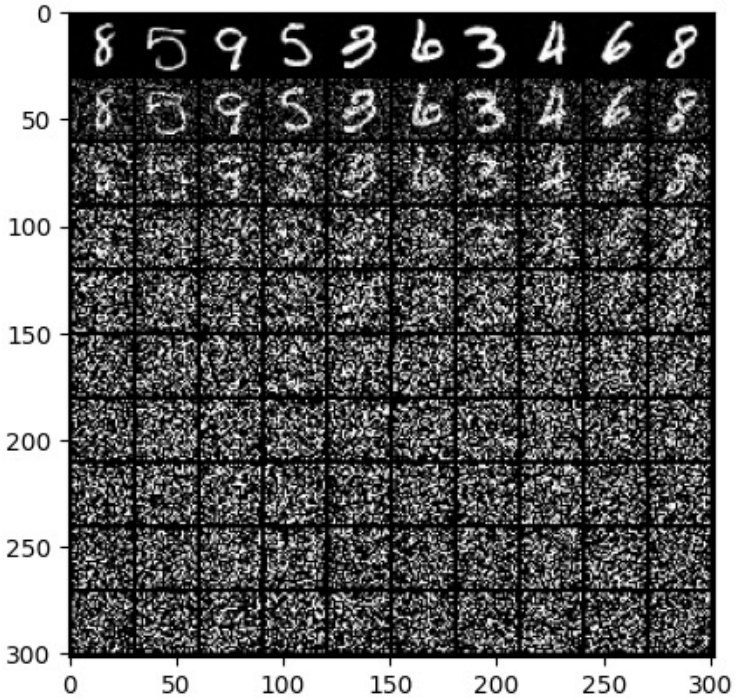


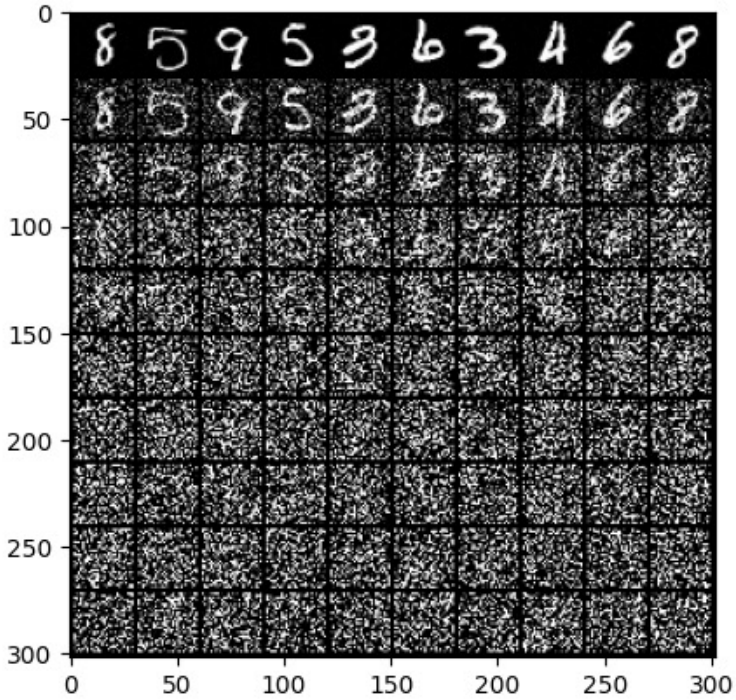




2-1-1: Variance Scheduling

- Check your implementation.
- Visualization of the forward diffusion process should look somewhat like this. (linear)





2-1-2: Training Implementation

- The U-Net architecture is provided in the assignment file.
- You only have to implement the training process: loss computation
- Given an original sample x and its class label c,
  - Compute x_t at random timestep t.
  - Forward through the model and compute the loss of prediction.

```
def forward(self, x, c):
    """
    this method is used in training, sample t and noise randomly.
    """

    ########################################################################
    #                           IMPLEMENT YOUR CODE                        #
    ########################################################################

    if self.prediction_type == 'epsilon':
        pass

    elif self.prediction_type == 'sample':
        pass

    ########################################################################
    #                           END OF YOUR CODE                           #
    ########################################################################
```

2-1-2: Sampling implementation

- From a random noise, generate an image given a class label.
- Save the intermediate results for visualization afterwards.
  - NOTE: saving results for all 1000 steps could be too much; recommendation is saving every 20 iterations.

```
def sample(self, n_sample, size, device):
    x_i = torch.randn(n_sample, *size).to(device)  # x_T ~ N(0, 1), sample initial noise
    c_i = torch.arange(0, 10).to(device)  # the mnist labels
    assert n_sample % c_i.shape[0] == 0, 'number of samples must be a multiple of 10.'
    c_i = c_i.repeat(int(n_sample/c_i.shape[0]))  # make n_samples
    x_i_store = [] # keep track of generated steps to plot the reverse process: save intermediate results

    ################################################################################
    #                            IMPLEMENT YOUR CODE                               #
    ################################################################################



    ################################################################################
    #                            END OF YOUR CODE                                  #
    ################################################################################

    x_i_store = np.array(x_i_store)
    return x_i, x_i_store
```

2-1-3: Train & Sample

- Set hyper-parameters and try training & sampling.

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

```
device='cuda'
n_epoch = None              # total training epoch
n_T = None                  # total timesteps of diffusion process
betas = None                # beta schedule
n_classes = 10              # the number of class in dataset
n_feat = None               # feature dimension of UNet
lr = None                   # learning rate
prediction_type = None      # prediction type of diffusion models: epsilon (noise) or sample (x_0)
schedule = None             # noise schedule

# to save results
save_dir = f'./results_{prediction_type}_{schedule}/'  # DO NOT change this part, or else it could be left out when compressing.
os.makedirs(save_dir, exist_ok=True)
save_every = 5

nn_model = ContextUnet(in_channels=1, n_feat=n_feat, n_classes=n_classes)
ddpm = DDPM(nn_model=nn_model, betas=betas, n_T=n_T, prediction_type=prediction_type, schedule=schedule, device=device)
ddpm.to(device)

optim = None                # set optimizer for training.
```

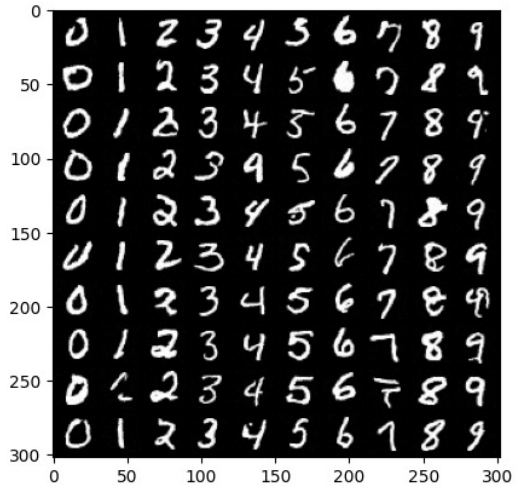
2-1-3: Train & Sample

6. Inference

With your trained model, try generating random samples with your model.

```
[ ]  num_samples = 100
     x_gen, x_gen_store = ddpm.sample(num_samples, (1, 28, 28), device)
     plt.imshow(make_grid(x_gen.cpu(), nrow=10).permute(1, 2, 0))
     plt.show()
     plt.close()
```

- Once training is complete, try sampling images.
  - The result should look somewhat like the example below.
  - 10 samples per class, 100 samples will be generated.
    - ✓ DO NOT clear outputs!
    - ✓ Will be evaluated based on 100 samples.



Assignment 2-2: Text-to-Image Synthesis

- Enjoy the publicly available text-to-image diffusion model! (Stable Diffusion)

T prompt string Shift + Return to add a new line

an astronaut riding a horse on mars, hd, dramatic lighting

Input prompt

Default: "a vision of paradise. unreal engine"

height integer

768

Height of generated image in pixels. Needs to be a multiple of 64

Default: 768

width integer

768

Width of generated image in pixels. Needs to be a multiple of 64

Default: 768

T negative_prompt string

Specify things to not see in the output

https://replicate.com/stability-ai/stable-diffusion

Assignment 2-2: Text-to-Image Synthesis

- Here is an example of inference.

```
NUM_IMAGES = 4

prompt = "a photo of nike air jordan shoes"
GUIDANCE_SCALE = 7.5
NUM_DENOISING_STEPS = 50
SEED = 1234


images = pipe(
    prompt,
    num_images_per_prompt=NUM_IMAGES,
    guidance_scale=GUIDANCE_SCALE,
    num_inference_steps=NUM_DENOISING_STEPS,
    height=IMAGE_SIZE,
    width=IMAGE_SIZE,
    generator=torch.Generator('cuda').manual_seed(SEED),
    ).images
```

Assignment 2-2: Text-to-Image Synthesis

- Here are some guidelines that could be helpful for writing the text prompts.

Appendix A: Stable Diffusion Prompt Guide

In general, the best stable diffusion prompts will have this form:

> *"A [type of picture] of a [main subject], [style cues]*"*

Some *types of picture* include *digital illustration, oil painting (usually good results), matte painting, 3d render, medieval map*.

The *main subject* can be anything you're thinking of, but StableDiffusion still struggles with compositionality, so it shouldn't be more than one or two main things (say, *a beaver wearing a suit*, or *a cat samurai with a pet pug*). The *main subject* should be mostly composed of adjectives and nouns. Avoid verbs, as Stable Diffusion has a hard time interpreting them correctly.

Style cues can be anything you want to condition the image on. I wouldn't add too many, maybe only 1 to 3. These can really vary a lot but some good ones are: *concept art, steampunk, trending in artstation, good composition, hyper realistic, oil on canvas, vivid colors*.

Additionally, adding the name of an artist as a cue will make the picture look like something that artist made, though it may condition the image's contents, especially if that artist had narrow themes (Beatrix Potter gets you spurious rabbits, for instance).

https://strikingloo.github.io/stable-diffusion-vs-dalle-2

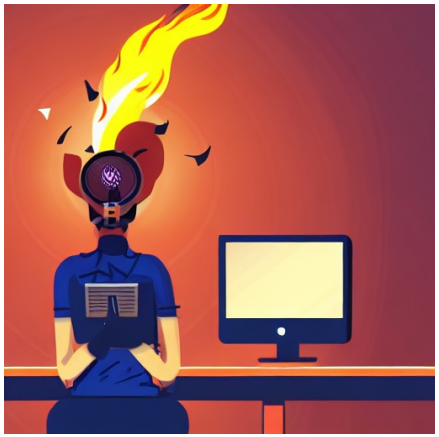
https://docs.google.com/document/d/17VPu3U2qXthOpt2zWczFvf-AH6z37hxUbvEe1rJTsEc/edit

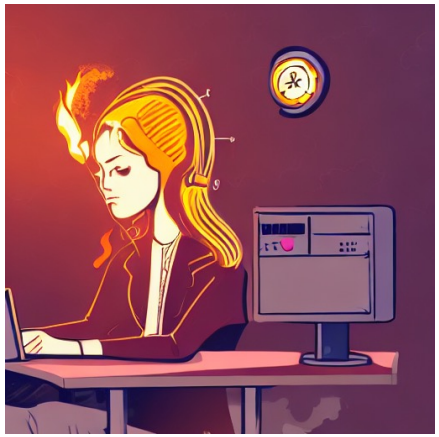
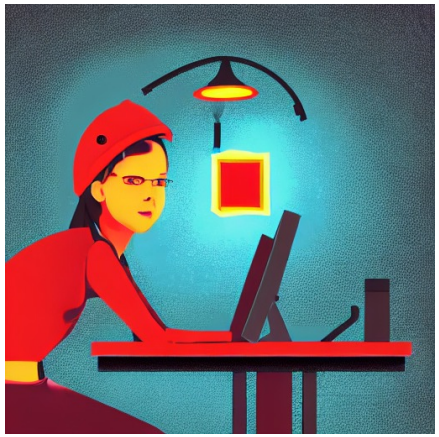
Assignment 2-2: Text-to-Image Synthesis

- The collection of works will be shared with the class.
- Here are some works from last year:




" A photo of Cristiano Ronaldo wearing the red top uniform of South Korean football team "



" A digital illustration of a computer engineering student with fire burning on her head, dramatic lightning "

Assignment 2-2: Text-to-Image Synthesis

- The collection of works will be shared with the class.
- Here are some works from last year:




“ A rabbit on skate board, temple by Tomas Kinkade ”




“ A photo of Elon Musk playing cards”

Score criteria

- Assignment 2-1: 90 points
  - Problem 1 (20 points)
    - ✓ Beta schedule (10 points)
    - ✓ DDPM schedule parameters (10 points)
  - Problem 2 (20 points)
    - ✓ Loss computation for epsilon & sample prediction (10points)
    - ✓ Sampling from noise implementation (10points)
  - Problem 3 (50 points)
    - ✓ 4가지 setting 학습 시도 (10 points each)
      - Prediction (Epsilon, Sample) + Schedule (linear, cosine)
      - Will be automatically saved.
    - ✓ 생성된 이미지 퀄리티 (10 points)
      - 명확한 숫자의 개수 (n개 / 100개)

- Assignment 2-2: 10 points

How to install assignment files

- 포함된 파일: 3개
  1. Assignment2-1_diffusion.ipynb
  2. Assignment2-2_TTI.ipynb
  3. CollectSubmission.sh

- 다운 후 설치 방법
  1. $tar zxvf Assignment2.tar.gz (decompress tar.gz file)
  2. $cd Assignment1
  3. $chmod 755 CollectSubmission.sh (get permission of script file)
  4. $conda activate {가상환경명}
  5. $jupyter notebook

- IPython notebook상에서 과제 수행

Important Dates

- Due: 12/6 (수) 23:59
- PLEASE read the notes on the notebooks carefully
- Google first before mailing TAs

- Submitting your work
  - DO NOT clear the final outputs
  - After you are done all two parts:
    1. $ ./CollectSubmission.sh 2000-00000 (학번)
    2. Upload the 2000-00000.tar.gz on eTL

- TA email : deeplearning.snu@gmail.com


Thank You!