NCSN 4.3: NCSN inference via annealed Langevin dynamics

2024.05.06 박준서

4.2: Learning NCSNs via score matching

- 이상적으로, $\lambda(\sigma_i)l(\theta;\sigma_i)$ 의 크기가 같길 바란다.
  - Equal priority in training process
  - 경험적으로, score network 를 최적화 시키면 $||s_\theta(\tilde{x},\sigma)||_2 \propto 1/\sigma$
  - Set $\lambda(\sigma_i) = \sigma_i^2$
  - $\sigma^2 l(\theta;\sigma) = ||\sigma s_\theta(\tilde{x},\sigma) + \frac{\tilde{x}-x}{\sigma}||_2^2$, $\sigma s_\theta(\tilde{x},\sigma) \propto 1$, $\frac{\tilde{x}-x}{\sigma} \sim N(0,I)$
  - 결과적으로 $\lambda(\sigma_i)l(\theta;\sigma)$ 가 σ 에 의존하지 않는다.

for σ_i,
$$\ell(\boldsymbol{\theta};\sigma) \triangleq \frac{1}{2}\mathbb{E}_{p_{\text{data}}(\mathbf{x})}\mathbb{E}_{\tilde{\mathbf{x}}\sim\mathcal{N}(\mathbf{x},\sigma^2 I)}\left[\left\|\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}},\sigma) + \frac{\tilde{\mathbf{x}}-\mathbf{x}}{\sigma^2}\right\|_2^2\right].$$

for $all\ \sigma_i$,
$$\mathcal{L}(\boldsymbol{\theta};\{\sigma_i\}_{i=1}^{L}) \triangleq \frac{1}{L}\sum_{i=1}^{L}\lambda(\sigma_i)\ell(\boldsymbol{\theta};\sigma_i),$$

4.3: NCSN inference via annealed Langevin dynamics

- Algorithm 분석
- Annealed Langevin dynamics 의 효과

Noise scale

initial point

Sampling using Langevin dynamics

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^L, \epsilon, T.$

Initialize $\tilde{\mathbf{x}}_0$

for $i \leftarrow 1$ to L **do**

  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.

  for $t \leftarrow 1$ to T **do**

    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$

    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$

  end for

  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$

end for

return $\tilde{\mathbf{x}}_T$

4.3: NCSN inference via annealed Langevin dynamics

- Langevin dynamics
  - SDE (Stochastic Differential Equation)의 한 종류
  - 시간의 흐름에 따른 값 추정을 통해 sampling
  - $dt = \epsilon, dw = \sqrt{dt}\, z, z \sim N(0, I)$

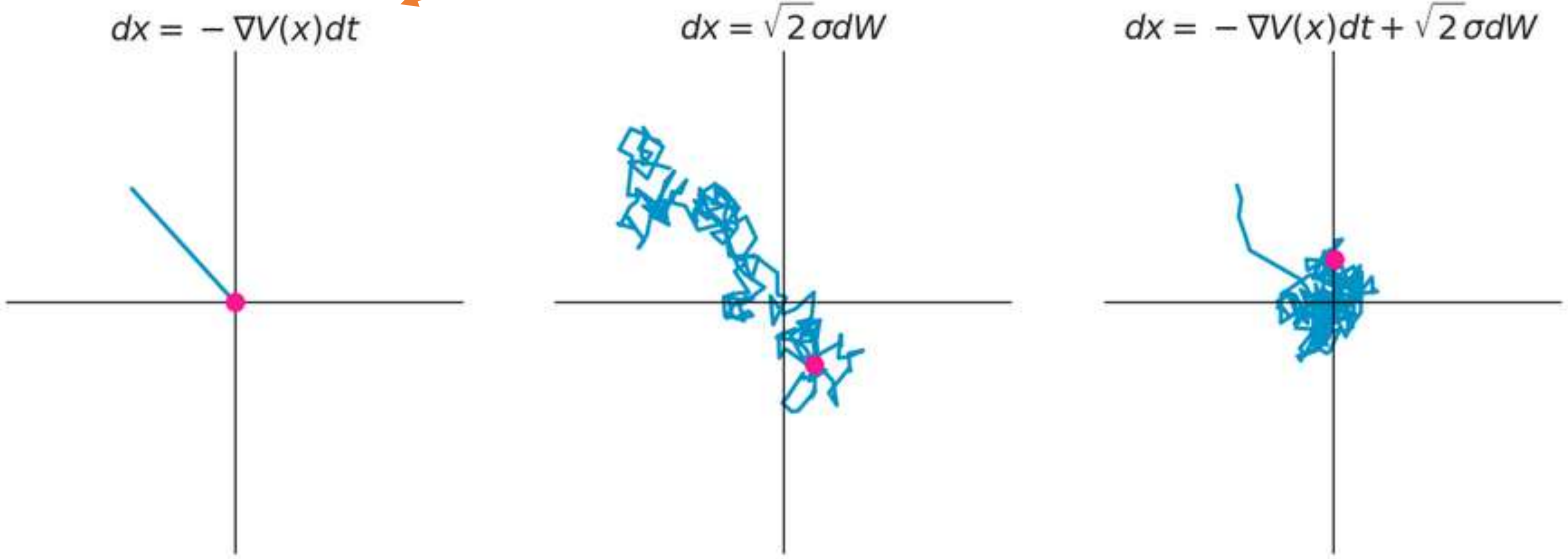
$$d\mathbf{x} = d\mathbf{w} + \frac{1}{2}\nabla_{\mathbf{x}} \log p(\mathbf{x})dt$$

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}}_{t-1}) + \sqrt{\epsilon}\, \mathbf{z}_t,$$

4.3: NCSN inference via annealed Langevin dynamics

- Langevin dynamics
  - Randomness: stochastic property 제공 및 local minima 를 빠져나갈 수 있다.

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}}_{t-1}) + \sqrt{\epsilon}\, \mathbf{z}_t,$$



4.3: NCSN inference via annealed Langevin dynamics

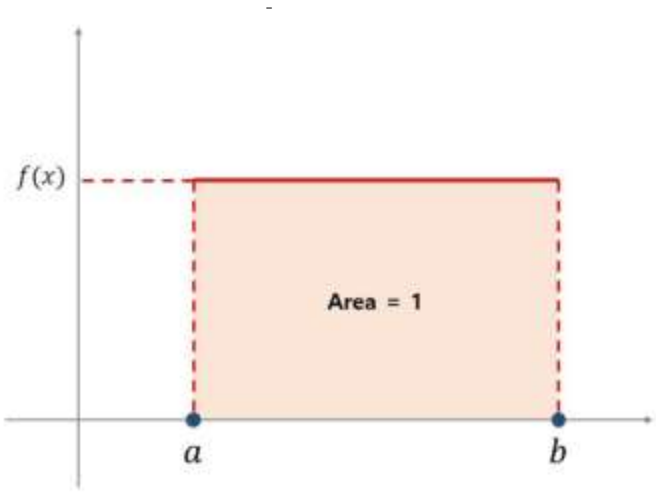
- Langevin dynamics
  - $\epsilon \to 0, T \to \infty$: $\tilde{x}_T$ becomes an exact sample from $p(x)$

$$\tilde{\mathbf{x}}_t = \tilde{\mathbf{x}}_{t-1} + \frac{\epsilon}{2}\nabla_{\mathbf{x}} \log p(\tilde{\mathbf{x}}_{t-1}) + \sqrt{\epsilon}\, \mathbf{z}_t,$$

4.3: NCSN inference via annealed Langevin dynamics

- Annealed Langevin dynamics
  - Step size 가 i 를 반영한다.
- Setting
  - $L: 10, \sigma_1: 1, \sigma_{10}: 0.01, T = 100$
  - $\epsilon: 2 \times 10^{-5}$

Uniform



Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \dfrac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

4.3: NCSN inference via annealed Langevin dynamics

- Step size: SNR (signal-to-Noise Ratio) 를 고정시키기 위해
  - $\alpha_i \propto \sigma_i^2$
  - SNR = $\dfrac{\alpha_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma_i)}{2\sqrt{\alpha_i}\ \mathbf{z}}$

$$\boxed{\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma) \right\|_2 \propto {}^{1}/_{\sigma}}$$

$$\mathbb{E}\left[\left\| \frac{\alpha_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma_i)}{2\sqrt{\alpha_i}\ \mathbf{z}} \right\|_2^2\right] \approx \mathbb{E}\left[\frac{\alpha_i \left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma_i) \right\|_2^2}{4}\right] \propto \frac{1}{4}\mathbb{E}\left[\left\| \sigma_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}, \sigma_i) \right\|_2^2\right] \propto \frac{1}{4}$$

  - 즉, SNR 에 σ_i 에 의존하지 않는 값이 된다.
  - SNR 이 σ_i 에 의존적이라면?
    - 특정 σ_i 에 따라 SNR 의 값이 바뀌고
    - SNR 값이 크다면, stochastic property x
    - 작다면, noise 가 많아서 방향 잡기가 어렵다

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

```
Initialize x̃_0
for i ← 1 to L do
    α_i ← ε · σ_i^2 / σ_L^2                         ▷ α_i is the step size.
    for t ← 1 to T do
        Draw z_t ~ N(0, I)
        x̃_t ← x̃_{t−1} + (α_i / 2) s_θ(x̃_{t−1}, σ_i) + √α_i z_t
    end for
    x̃_0 ← x̃_T
end for
return x̃_T
```

4.3: NCSN inference via annealed Langevin dynamics

Perturbed data distribution 으로

최대한 가서 다음 iter 에서의 good initial point 로 set

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T.$

Initialize $\tilde{\mathbf{x}}_0$

for $i \leftarrow 1$ to L **do**

$\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ $\quad \triangleright$ α_i is the step size.

$\quad$ **for** $t \leftarrow 1$ to T **do**

$\quad\quad$ Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$

$\quad\quad \tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$

$\quad$ **end for**

$\quad \tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$

end for

return $\tilde{\mathbf{x}}_T$

4.3: NCSN inference via annealed Langevin dynamics

- Annealed Langevin dynamics 의 효과
  - $L: 10, \sigma_1: 10, \sigma_{10}: 0.1, T = 100$
  - Mode 간의 상대적인 차이도 잘 반영한다.

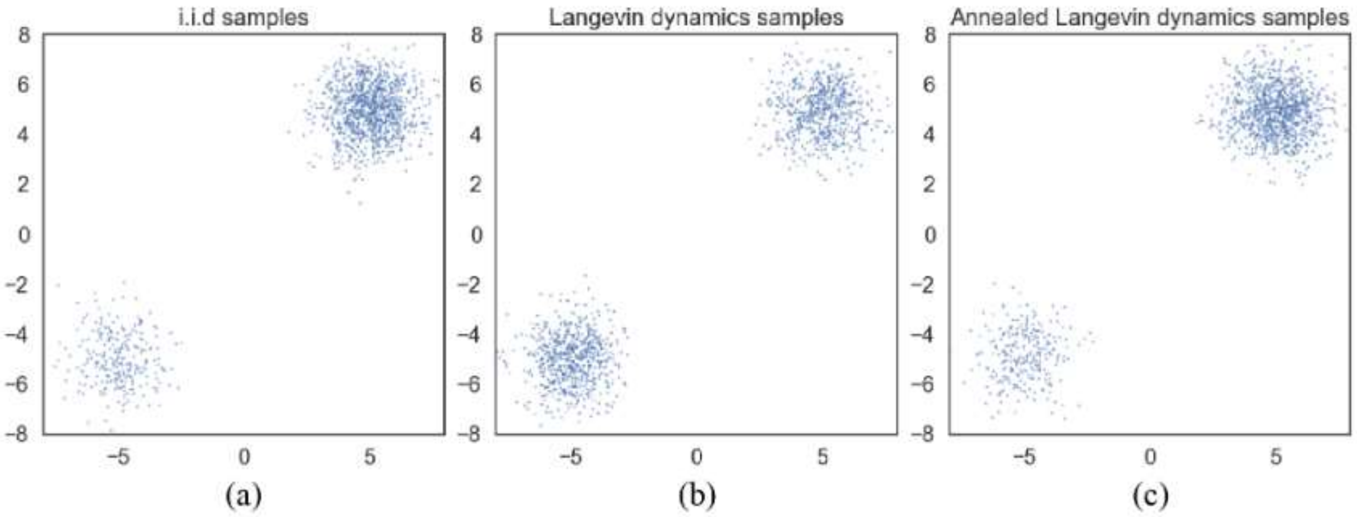


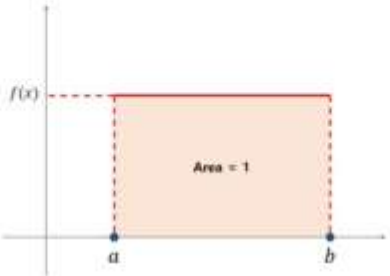
Figure 3: Samples from a mixture of Gaussian with different methods. (a) Exact sampling. (b) Sampling using Langevin dynamics with the exact scores. (c) Sampling using annealed Langevin dynamics with the exact scores. Clearly Langevin dynamics estimate the relative weights between the two modes incorrectly, while annealed Langevin dynamics recover the relative weights faithfully.

4.3: NCSN inference via annealed Langevin dynamics

- Code review: noise level

- Annealed Langevin dynamics
  - Step size 가 i 를 반영한다.
  - L: 10, σ_1: 1, σ_{10}: 0.01, T = 100
  - ϵ: 2×10^{-5}

Uniform noise



Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ ▷ α_i is the step size.
    for $t \leftarrow 1$ to T **do**
        Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
        $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
    end for
    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

```
model:
  sigma_begin: 1
  sigma_end: 0.01
  num_classes: 10
```

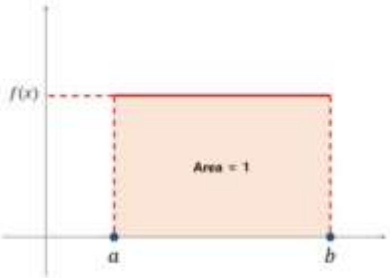
```
sigmas = np.exp(np.linspace(np.log(self.config.model.sigma_begin), np.log(self.config.model.sigma_end),
                            self.config.model.num_classes))
```

4.3: NCSN inference via annealed Langevin dynamics

- Code review:
  - initial sample: [0,1) uniform
  - MNIST: 1 x 28 x 28
  - 25 samples
  - T: $100, \epsilon$: 2×10^{-5}

- Annealed Langevin dynamics
  - Step size 가 i 를 반영한다.
  - L: $10, \sigma_1$: $1, \sigma_{10}$: $0.01, T = 100$
  - ϵ: 2×10^{-5}

Uniform noise



Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T.$
Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
    for $t \leftarrow 1$ to T **do**
        Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
        $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\, \mathbf{z}_t$
    end for
    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

```
score.eval()
grid_size = 5

imgs = []
if self.config.data.dataset == 'MNIST':
    samples = torch.rand(grid_size ** 2, 1, 28, 28, device=self.config.device)
    all_samples = self.anneal_Langevin_dynamics(samples, score, sigmas, 100, 0.00002)
```

4.3: NCSN inference via annealed Langevin dynamics

- Code review: annealed Langevin dynamics sampling

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
    for $t \leftarrow 1$ to T **do**
        Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
        $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \dfrac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
    end for
    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

```python
def anneal_Langevin_dynamics(self, x_mod, scorenet, sigmas, n_steps_each=100, step_lr=0.00002):
    images = []

    with torch.no_grad():
        for c, sigma in tqdm.tqdm(enumerate(sigmas), total=len(sigmas), desc='annealed Langevin
            labels = torch.ones(x_mod.shape[0], device=x_mod.device) * c
            labels = labels.long()
            step_size = step_lr * (sigma / sigmas[-1]) ** 2
            for s in range(n_steps_each):
                images.append(torch.clamp(x_mod, 0.0, 1.0).to('cpu'))
                noise = torch.randn_like(x_mod) * np.sqrt(step_size * 2)
                grad = scorenet(x_mod, labels)
                x_mod = x_mod + step_size * grad + noise
                # print("class: {}, step_size: {}, mean {}, max {}".format(c, step_size, grad.ab
                #                                                          grad.abs().max()))

        return images
```

4.3: NCSN inference via annealed Langevin dynamics

- Code review: annealed Langevin dynamics sampling

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
    $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ α_i is the step size.
    for $t \leftarrow 1$ to T **do**
        Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
        $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
    end for
    $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

```python
def anneal_Langevin_dynamics(self, x_mod, scorenet, sigmas, n_steps_each=100, step_lr=0.00002):
    images = []

    with torch.no_grad():
        for c, sigma in tqdm.tqdm(enumerate(sigmas), total=len(sigmas), desc='annealed Langevin
            labels = torch.ones(x_mod.shape[0], device=x_mod.device) * c
            labels = labels.long()
            step_size = step_lr * (sigma / sigmas[-1]) ** 2
            for s in range(n_steps_each):
                images.append(torch.clamp(x_mod, 0.0, 1.0).to('cpu'))
                noise = torch.randn_like(x_mod) * np.sqrt(step_size * 2)
                grad = scorenet(x_mod, labels)
                x_mod = x_mod + step_size * grad + noise
                # print("class: {}, step_size: {}, mean {}, max {}".format(c, step_size, grad.al
                #                                                        grad.abs().max()))

        return images
```

4.3: NCSN inference via annealed Langevin dynamics

- Code review: annealed Langevin dynamics sampling

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

1: Initialize $\tilde{\mathbf{x}}_0$
2: **for** $i \leftarrow 1$ to L **do**
3: $\quad \alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ $\quad \triangleright\ \alpha_i$ is the step size.
4: $\quad$ **for** $t \leftarrow 1$ to T **do**
5: $\quad\quad$ Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
6: $\quad\quad \tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \dfrac{\alpha_i}{2} \mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \underline{\sqrt{\alpha_i}\, \mathbf{z}_t}$
7: $\quad$ **end for**
8: $\quad \tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
9: **end for**
return $\tilde{\mathbf{x}}_T$

```python
def anneal_Langevin_dynamics(self, x_mod, scorenet, sigmas, n_steps_each=100, step_lr=0.00002):
    images = []

    with torch.no_grad():
        for c, sigma in tqdm.tqdm(enumerate(sigmas), total=len(sigmas), desc='annealed Langevin
            labels = torch.ones(x_mod.shape[0], device=x_mod.device) * c
            labels = labels.long()
            step_size = step_lr * (sigma / sigmas[-1]) ** 2
            for s in range(n_steps_each):
                images.append(torch.clamp(x_mod, 0.0, 1.0).to('cpu'))
                noise = torch.randn_like(x_mod) * np.sqrt(step_size * 2)
                grad = scorenet(x_mod, labels)
                x_mod = x_mod + step_size * grad + noise
                # print("class: {}, step_size: {}, mean {}, max {}".format(c, step_size, grad.ab
                #                                                          grad.abs().max()))

        return images
```

4.3: NCSN inference via annealed Langevin dynamics

- Code review: annealed Langevin dynamics sampling

Algorithm 1 Annealed Langevin dynamics.

Require: $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
for $i \leftarrow 1$ to L **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2/\sigma_L^2$ $\quad \triangleright$ α_i is the step size.
  for $t \leftarrow 1$ to T **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \dfrac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  end for
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
end for
return $\tilde{\mathbf{x}}_T$

```python
def anneal_Langevin_dynamics(self, x_mod, scorenet, sigmas, n_steps_each=100, step_lr=0.00002):
    images = []

    with torch.no_grad():
        for c, sigma in tqdm.tqdm(enumerate(sigmas), total=len(sigmas), desc='annealed Langevin
            labels = torch.ones(x_mod.shape[0], device=x_mod.device) * c
            labels = labels.long()
            step_size = step_lr * (sigma / sigmas[-1]) ** 2
            for s in range(n_steps_each):
                images.append(torch.clamp(x_mod, 0.0, 1.0).to('cpu'))
                noise = torch.randn_like(x_mod) * np.sqrt(step_size * 2)
                grad = scorenet(x_mod, labels)
                x_mod = x_mod + step_size * grad + noise
                # print("class: {}, step_size: {}, mean {}, max {}".format(c, step_size, grad.ab
                #                                                          grad.abs().max()))

        return images
```