EE-608: Deep Learning for Natural Language Processing:

Diffusion

James Henderson

Idiap Research Institute

DLNLP, Lecture 12

Outline

Diffusion

Outline

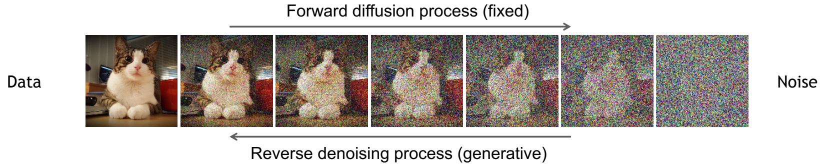
Diffusion

Denoising Diffusion Models

Learning to generate by denoising

Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising



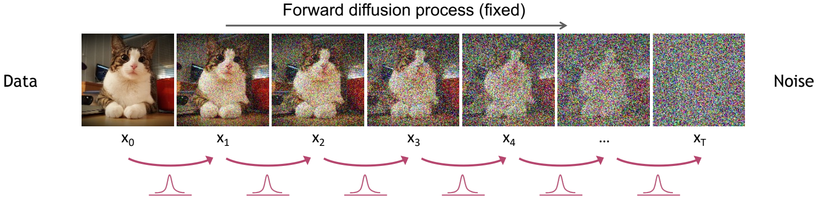
Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

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

Slide from Ruiqi Gao

Forward Diffusion Process

The formal definition of the forward process in T steps:



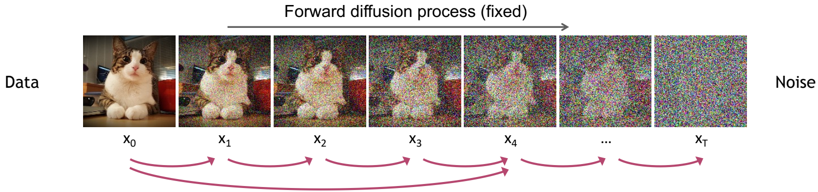
$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \quad \Rightarrow \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \text{(joint)}$$

Similar to the inference model in hierarchical VAEs.

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

Slide from Ruiqi Gao

Diffusion Kernel



Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ ➡ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \rightarrow 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

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

Slide from Ruiqi Gao

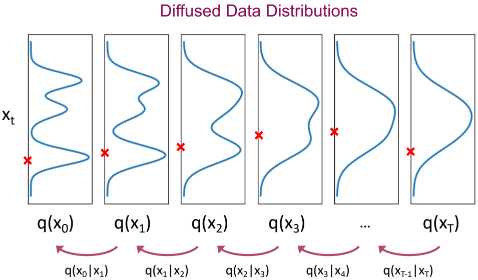
Generative Learning by Denoising

Recall, that the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Generation:

Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$



In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

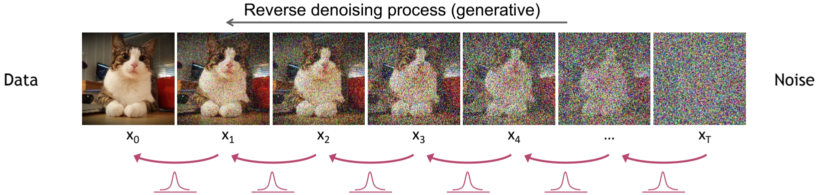
Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Gaussian distribution if β_t is small in each forward diffusion step.

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

Slide from Ruiqi Gao

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:



$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}, \sigma_t^2 \mathbf{I})$$

Trainable network
(U-net, Denoising Autoencoder)

$$\Rightarrow \quad p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

Similar to the generative model in hierarchical VAEs.

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

Slide from Ruiqi Gao

Learning Denoising Model

Variational upper bound

For training, we can form variational upper bound (negative ELBO) that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020 show that:

$$L = \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right]$$

where $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0), \tilde{\beta}_t\mathbf{I}),$$

$$\text{where } \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

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

Slide from Ruiqi Gao

Parameterizing the Denoising Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} ||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2 \right] + C$$

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$. Ho et al. NeurIPS 2020 observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ \epsilon_\theta(\mathbf{x}_t, t) \right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)} ||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon}_{\mathbf{x}_t}, t)||^2 \right] + C$$

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

Slide from Ruiqi Gao

Training Objective Weighting

Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon, t)||^2 \right]$$

The time dependent λ_t ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small t's.

Ho et al. NeurIPS 2020 observe that simply setting $\lambda_t = 1$ improves sample quality. So, they propose to use:

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1, T)} \left[||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon}_{\mathbf{x}_t}, t)||^2 \right]$$

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

Slide from Ruiqi Gao

Summary

Training and Sample Generation

Algorithm 1 Training

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

Algorithm 2 Sampling

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

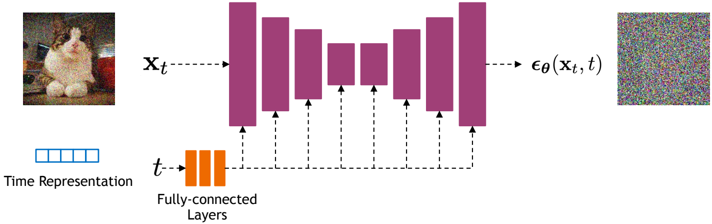
slide from https://cvpr2022-tutorial-diffusion-models.github.io/

Slide from Ruiqi Gao

Implementation Considerations

Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see [Dharivwal and Nichol NeurIPS 2021](#))

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

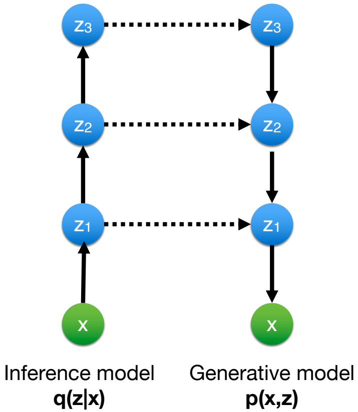
Slide from Ruiqi Gao

Connection to VAEs

Diffusion models can be considered as a special form of hierarchical VAEs.

However, in diffusion models:

- The inference model is fixed: easier to optimize
- The latent variables have the same dimension as the data.
- The ELBO is decomposed to each time step: fast to train
    - Can be made extremely deep (even infinitely deep)
- The model is trained with some reweighting of the ELBO.



Vahdat and Kautz, NVAE: A Deep Hierarchical Variational Autoencoder, NeurIPS 2020

Sønderby, et al.. Ladder variational autoencoders, NeurIPS 2016.

Slide from Ruiqi Gao

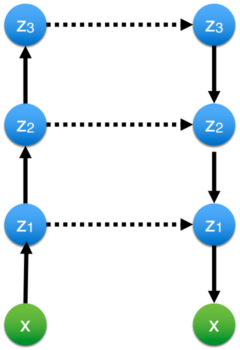
Hierarchical VAEs

- "Flat" VAEs suffer from simple priors
- Making both inference model and generative model hierarchical

$$q_\phi(\mathbf{z}_{1,2,3}|\mathbf{x}) = q_\phi(\mathbf{z}_1|\mathbf{x})q_\phi(\mathbf{z}_2|\mathbf{z}_1)q_\phi(\mathbf{z}_3|\mathbf{z}_2)$$

$$p_\theta(\mathbf{z}_{1,2,3}) = p_\theta(\mathbf{z}_3)p_\theta(\mathbf{z}_2|\mathbf{z}_3)p_\theta(\mathbf{z}_1|\mathbf{z}_2)p_\theta(\mathbf{x}|\mathbf{z}_1)$$

- Better likelihoods are achieved with hierarchies of latent variables



Inference model
q(z|x)

Generative model
p(x,z)

slide by Durk Kingma

[Kingma and Welling, 201

Slide from Ruiqi Gao

How to make sampling faster?

- One bottleneck of diffusion models is its slowness in sampling: need 10-1000+ steps to generate high quality samples
- Generative models need to be fast for practical use.

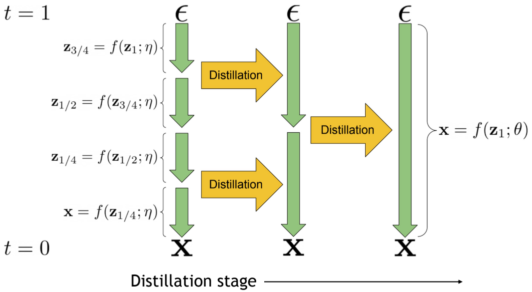
- One solution: distill diffusion models into models using just 4-8 sampling steps!
    - *Progressive distillation for fast sampling of diffusion models, Salimans & Ho, ICLR 2022*
    - *On Distillation of Guided Diffusion Models, Meng et al., CVPR 2023*

Slide from Ruiqi Gao

Progressive distillation

How to make sampling faster?

- Distill a deterministic ODE sampler (i.e. DDIM sampler) to the same model architecture.
- At each stage, a "student" model is learned to distill two adjacent sampling steps of the "teacher" model to one sampling step.
- At next stage, the "student" model from previous stage will serve as the new "teacher" model.



Salimans & Ho, "Progressive distillation for fast sampling of diffusion models", ICLR 2022.

Slide from Ruiqi Gao

Algorithm 1 Standard diffusion training

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
  $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
  $t \sim U[0,1]$ ▷ Sample time
  $\epsilon \sim N(0, I)$ ▷ Sample noise
  $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data
  $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
  $\lambda_t = \log[\alpha_t^2/\sigma_t^2]$ ▷ log-SNR
  $L_\theta = w(\lambda_t)\|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
  $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization
end while

Algorithm 2 Progressive distillation

Require: $\boxed{\text{Trained teacher model } \hat{\mathbf{x}}_\eta(\mathbf{z}_t)}$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N
for K iterations **do**
  $\theta \leftarrow \eta$ ▷ Init student from teacher
  while not converged **do**
    $\mathbf{x} \sim \mathcal{D}$
    $t = i/N, \quad i \sim Cat[1, 2, \ldots, N]$
    $\epsilon \sim N(0, I)$
    $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
    `# 2 steps of DDIM with teacher`
    $t' = t - 0.5/N, \quad t'' = t - 1/N$
    $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
    $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
    $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
    $\lambda_t = \log[\alpha_t^2/\sigma_t^2]$
    $L_\theta = w(\lambda_t)\|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
    $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
  end while
  $\eta \leftarrow \theta$ ▷ Student becomes next teacher
  $N \leftarrow N/2$ ▷ Halve number of sampling steps
end for

Salimans & Ho, "Progressive distillation for fast sampling of diffusion models", ICLR 2022.

Slide from Ruiqi Gao

Classifier Guidance

Sampler technique

- Assume pairs of data (x, c). A classifier guidance diffusion model consists of
    - A trained conditional diffusion model
    - A trained classifier model on noisy data $\mathbf{x}_t$
- During sampling, at each denoising step, modify the score function to

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_{\theta,\phi}(\mathbf{x}_t|\mathbf{c}) = \underbrace{\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|\mathbf{c})}_{\text{From the conditional diffusion model}} + \omega \underbrace{\nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t)}_{\text{From the classifier model}}.$$

- Upweight samples that the classifier assigns high probability with, better alignment with c.
- Cons: need to train an additional classifier. Increase model complexity.

[Dhariwal and Nichol, "Diffusion models beat GANs on image synthesis", NeurIPS 2021.]

Slide from Ruiqi Gao

Classifier-free Guidance

Sampler technique

- Assume there're two diffusion models, one conditional model and one unconditional model.
- By Bayes' rule we can define an implicit classifier

$$p_\theta(\mathbf{c}|\mathbf{x}_t) \propto p_\theta(\mathbf{x}_t|\mathbf{c})/p_\theta(\mathbf{x}_t).$$

- The modified score function during sampling then becomes

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_\theta(\mathbf{x}_t|\mathbf{c}) = (1+\omega)\underbrace{\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|\mathbf{c})}_{\text{From the conditional diffusion model}} - \omega\underbrace{\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t)}_{\text{From the unconditional diffusion model}}.$$

- The two models can share weights, with the unconditional model taking a null class label c.

Ho & Salimans, "Classifier-Free Diffusion Guidance", 2021.

Slide from Ruiqi Gao

Diffusion for Text

Diffusion for text is challenging due to its discrete nature

- add continuous noise to word embeddings
- add continuous noise to the probability simplex over vocabulary
- add discrete noise as edits to text

Open issues

- how to add noise in the forward model
- predicting error versus predicting x_0 directly
- autoregressive versus non-autoregressive generation
- adding conditioning on previous prediction of x_0

Summary of Diffusion

- Diffusion does generation by doing many steps of denoising applied to random noise
- Diffusion trains a denoiser on real data with artificial noise added
- Diffusion for text is challenging due to its discrete nature
- Diffusion for text is still a very open research topic