Diffusion Language Model

Zhiping (Patricia) Xiao

UCLA SCAI Lab Course Reading Group Fall 2022

Outline

Introduction

Diffusion-LM Improves Controllable Text Generation (NeurIPS'22)

- Build a Language Model eligible for fine-grained controllable text generation, by applying diffusion model on **continuous** latent space.
- Code: `https://github.com/XiangLi1999/Diffusion-LM`

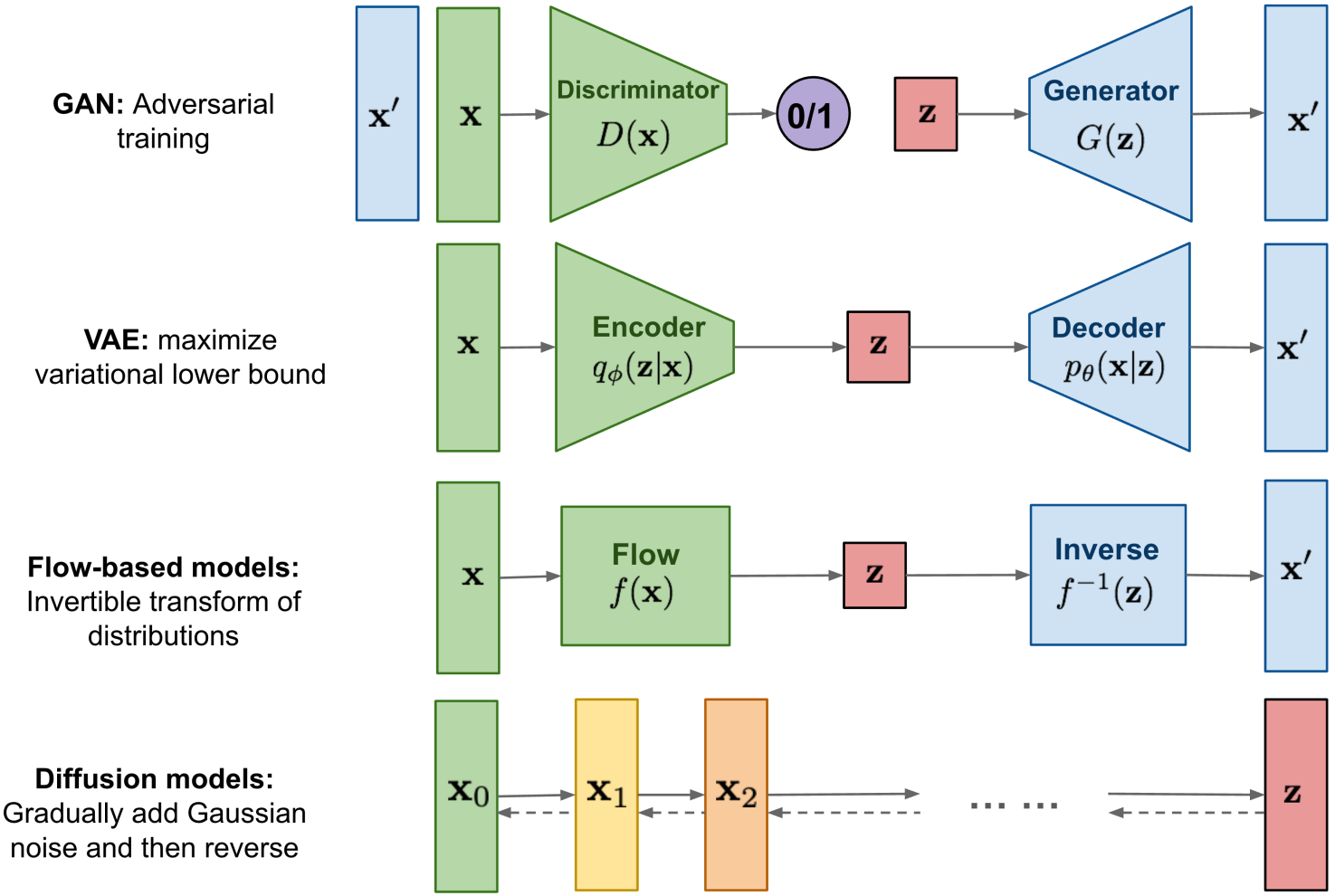

Figure: Overview of different types of generative models.[1]

[1] lilianweng.github.io 2021 post on diffusion models

Given a sequence of discrete words $\mathbf{w} = [w_1, w_2, \ldots w_n]$.

A language model $p_{\text{lm}}(\mathbf{w})$ denotes the probability distribution over sequences of words.

A language model assigns a probability $p_{\text{lm}}(\mathbf{w}) = P(w_1, \ldots, w_m)$ to the whole sequence.

Controllable Text Generation is to compute $p(\mathbf{w}|\mathbf{c})$, aiming at generate $\mathbf{w}$ that satisfies the control target $\mathbf{c}$, where $\mathbf{c}$ is a control variable (e.g. syntax tree, sentiment, topic, politeness, to gender, persona).

Plug-and-Play Controllable Generation: aims to keep the LM frozen and steer its output using potential functions (e.g., classifiers). In this setting:

- $p_{\text{lm}}(\mathbf{w})$ pre-trained and frozen, encouraging fluent;
- For each task, train $p(\mathbf{c}|\mathbf{w})$ on small amount of data, encourage $\mathbf{w}$ to fulfill the control.
- Posterior $p(\mathbf{w}|\mathbf{c})$ approximated from Bayes rule:

$$p(\mathbf{w}|\mathbf{c}) \propto p(\mathbf{c}|\mathbf{w}) \cdot p_{\text{lm}}(\mathbf{w})$$

Quick Recap: Bayes Rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In this case, $p(\mathbf{c})$ can be regarded as a unchanged.

Therefore:

$$p(\mathbf{w}|\mathbf{c}) \propto p(\mathbf{c}|\mathbf{w}) \cdot p_{\text{lm}}(\mathbf{w})$$

Diffusion-LM

Diffusion LM: Motivation

Focusing on a language models (LMs) offering **controllable generation** for text without re-training.

Recent works succeed on controlling simple attributes e.g., sentiment, while little progress is made on complex, fine-grained controls (e.g., syntactic structure).

Solution: a new **non**-autoregressive language model based on **continuous diffusions**.

- **continuous** data domains: images, audio, etc. (*enables efficient gradient-based controllable generation*)
- previous text diffusion models: on discrete state spaces, defines a corruption process on **discrete** data (e.g., *each token has some probability to be corrupted to an absorbing or random token*).

PPLM: the Closest Related Work to Diffusion LM

Plug and play language models: A simple approach to controlled text generation (ICLR'20)

- It runs gradient ascent on an autoregressive LM's hidden activations make the following tokens satisfy the control while maintaining fluency.
- Drawback 1: PPLM is based on autoregressive model, it can only generate left-to-right, thus can never repair previous errors.
- Drawback 2: Work well on simple attribute (e.g. topic) control tasks, fail on more complex control tasks (e.g. syntactic structure).

Diffusion Models (DMs)

Recall that, the case of image:

$$L_{DM} = \mathbb{E}_{x,\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|_2^2\right].$$

The case of text: to maximize $\mathbb{E}_{\mathbf{x}_0\sim p_{data}}\Big[\log p_\theta(\mathbf{x}_0)\Big]$

View as modeling latent variables of the data $\mathbf{x}_0 \in \mathbb{R}^{nd}$ (n is the seq length, d is dimension of t) as a Markov chain
$\mathbf{x}_T, \ldots, \mathbf{x}_0 \in \mathbb{R}^{nd}$ where $\mathbf{x}_T$ is a Gaussian.
The initial state $p_\theta(\mathbf{x}_T) \approx \mathcal{N}(0, \mathbf{I})$, and noise to reduce at step t:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)),$$

where μ_θ and Σ_θ may be computed by a U-Net or a Tranformer. (Ablation Studies in Sec 7.4 and Appendix H)

Diffusion Models (DMs)

To train $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$:

- **Forward Process** q: incrementally adds Gaussian noise to data $\mathbf{x}_0$, until at diffusion step T, samples $\mathbf{x}_T$ are approximately Gaussian.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \,,$$

  where β_t is the amount of noise added at step t. q is pre-defined and contains no trainable parameter.

- **Reverse Process** p_θ: reconstruct the data (i.e. $\mathbf{x}_T \to \dots \mathbf{x}_0$), denoiser U-Net or Transformer.

- The **diffusion model** is trained to maximize the marginal likelihood of the data

$$\mathbb{E}_{\mathbf{x}_0 \sim p_{data}}\Big[\log p_\theta(\mathbf{x}_0)\Big]$$

Measurement of lm: Feed generated text to a teacher LM (i.e., a carefully fine-tuned GPT-2 model) and report the perplexity. This metric is called lm-score (lm), a lower lm-score indicates better sample quality.

Diffusion LM: Architecture

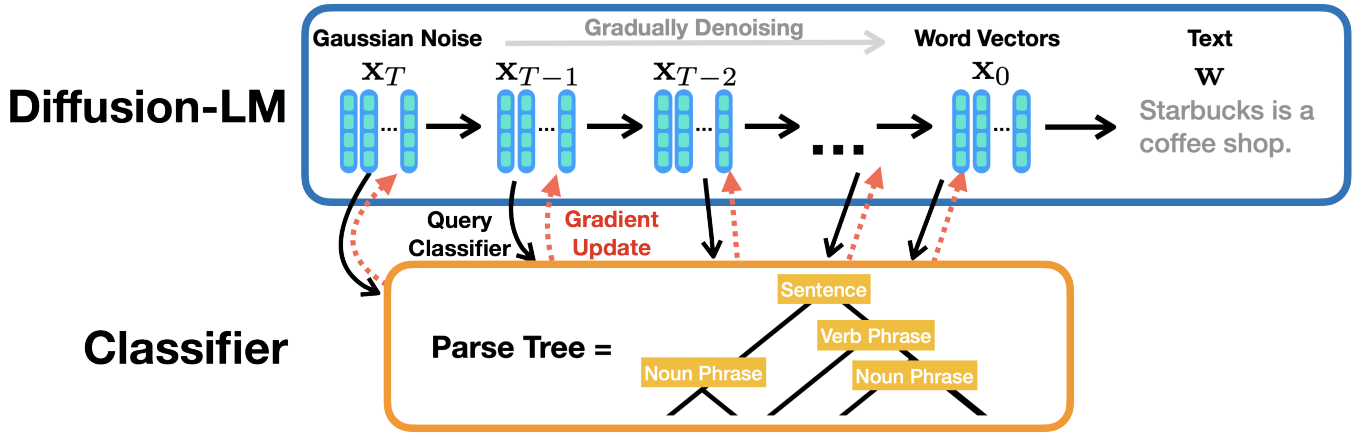

Figure: Iteratively denoises a sequence of Gaussian vectors into word vectors, yielding a intermediate latent variables of decreasing noise level $x_T \dots x_0$. For controllable generation: iteratively perform gradient updates on these continuous latents to optimize for fluency (parametrized by **Diffusion-LM**) and satisfy control requirements (parametrized by a **classifier**).

Diffusion LM: Architecture

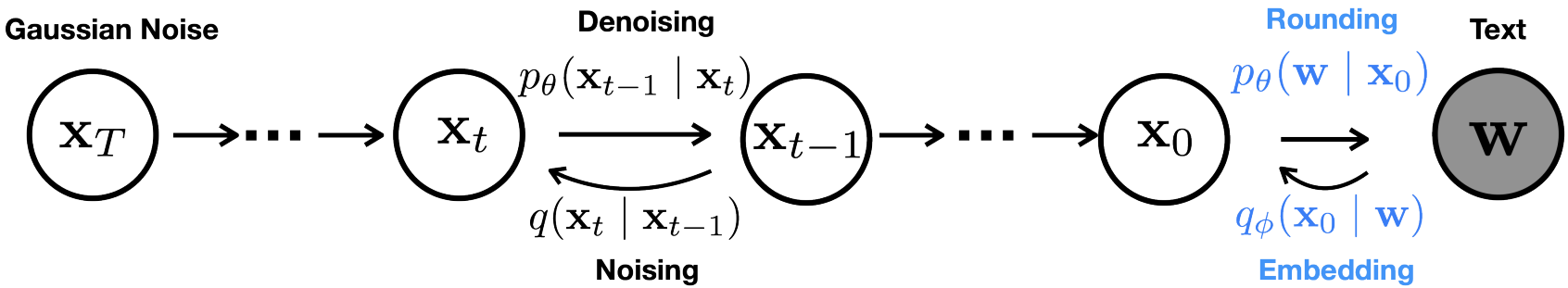

Figure: The forward and reverse diffusion processes. In addition to the original diffusion models, Diffusion-LM add a Markov transition between $\mathbf{x}_0$ and $\mathbf{w}$, defining embedding ($\mathbf{w} \to \mathbf{x}_0$) and rounding ($\mathbf{x}_0 \to \mathbf{w}$).

Diffusion LM: Non-Autoregressive

Autoregressive LMs: $p_{\text{lm}}(\mathbf{w})$

$$p_{\text{lm}}(\mathbf{w}) = p_{\text{lm}}(w_1) \prod_{i=2}^{n} p_{\text{lm}}(x_i | x_{<i}) \,,$$

where the next-token prediction $p_{\text{lm}}(x_i|x_{<i})$ is often parametrized by Transformer architecture.

Claim: Most large pre-trained LMs are left-to-right autoregressive. (e.g. GPT-3, PaLM)

- Fixed generation order (i.e., left to right) limits the flexibility of models.
- For more: `https://huggingface.co/transformers/v3.1.0/model_summary.html`

Diffusion LM: Non-Autoregressive

Non-autoregressive LMs:

Claim: Most of the existing models in this category are task-specific, such as machine translation and speech-to-text (e.g. CTC and Imputer, NAT))

- It has been shown that these models fail for language modeling in CoMMA paper.
- A Study of Non-autoregressive Model for Sequence Generation (ACL'20)

Claim: Diffusion-LM can condition on arbitrary classifiers, which could utilize complex, global properties of the sentence.

Why Diffusion LM uses Non-Autoregressive arch?

According to my personal opinion:

- Diffusion Models are often expensive.
  - Many steps needed for training and generation.
  - In this case, the denoiser – a U-Net or a Transformer.
- Non-autoregressive models generate a whole sequence at a time.

Diffusion LM: Diffusion Noise Schedule

The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS'22)
- The scheduler decide how much noise we add at each step.

Diffusion LM: Diffusion Noise Schedule

The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS'22)
- The scheduler decide how much noise we add at each step.

Problem: Standard noise schedules for continuous diffusion models are not robust for text data.

Diffusion LM: Diffusion Noise Schedule

The noise scheduler / variance schedule is an important hyper-parameter to be determined.

- Elucidating the Design Space of Diffusion-Based Generative Models (NeurIPS'22)
- The scheduler decide how much noise we add at each step.

Problem: Standard noise schedules for continuous diffusion models are not robust for text data.

Hypothesis: discrete nature of text involves rounding, making the model insensitive to noise near $t = 0$.

Solution: introduce a new ***sqrt* noise schedule** suits for text better.

Diffusion LM: Noise Schedule α, β Explained

Recall that during the forward process q incrementally adds Gaussian noise to data $\mathbf{x}_0$, until at diffusion step T, samples $\mathbf{x}_T$ are approximately Gaussian.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}),$$

with β_t to be scheduled, which is equivalent to:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon,$$

where $\epsilon \sim \mathcal{N}(0, I)$. In practice, they find **re-parametrization** helps with model training. By having $\overline{\alpha_t} = \prod_{s=0}^{t}(1-\beta_s)$, we have the closed-form expression of $\mathbf{x}_{t-1} = \sqrt{\overline{\alpha_t}}\mathbf{x}_0 + \sqrt{1-\overline{\alpha_t}}\epsilon$. We estimate $\mathbf{x}_{t-1}$ as:

$$\mathbf{x}_{t-1} = \sqrt{\overline{\alpha_t}}f_\theta(\mathbf{x}_t, t) + \sqrt{1-\overline{\alpha_t}}\epsilon,$$

where $f_\theta(\mathbf{x}_t, t)$ estimates $\mathbf{x}_0$ directly.

Proposed in Auto-Encoding Variational Bayes (ICLR'14)

- The combination of forward process q and backward process p_θ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $\mathbf{x}_0$.

Proposed in Auto-Encoding Variational Bayes (ICLR'14)

- The combination of forward process q and backward process p_θ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $\mathbf{x}_0$.
- The ELBO in this case is the sum of losses at each step:

$$L = L_0 + L_1 + \cdots + L_T$$

Proposed in Auto-Encoding Variational Bayes (ICLR'14)

- The combination of forward process q and backward process p_θ can be seen as a variational auto-encoder (VAE), hence the variational lower bound (i.e. ELBO) can be used to minimize the negative log-likelihood with respect to ground truth data sample $\mathbf{x}_0$.
- The ELBO in this case is the sum of losses at each step:

$$L = L_0 + L_1 + \cdots + L_T$$

- By the construction of q, we observe each L_t $(t = 1, 2, \ldots T)$ is the KL divergence between 2 Gaussian distributions.

$$\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2 = \|\epsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t}\mathbf{x}_0 + \sqrt{(1 - \overline{\alpha}_t)}\epsilon, t)\|^2$$

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Note that: sum of Gaussians is still Gaussian. As is shown by Deep Unsupervised Learning using Nonequilibrium Thermodynamics (ICML'15):

- It means that we do not have to apply q repeatedly to sample $\mathbf{x}_t$ from $\mathbf{x}_0$. Instead we have:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\overline{\alpha}_t}\mathbf{x}_0, (1-\overline{\alpha}_t)\mathbf{I}) \,,$$

  with $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{s=0}^{t} \alpha_s$
- This then allows optimizing random terms of L_t of a randomly sampled t during training.
- This re-param trick turns the model from a step-t mean predictor to a total noise predictor.
- references: huggingface post, Lilian post.

Diffusion LM: SQRT Noise Schedule

For the noise schedule $\sqrt{1-\overline{\alpha_t}}$, sqrt schedule defines $\overline{\alpha_t}$ as:

$$\overline{\alpha_t} = 1 - \sqrt{\frac{t}{T} + s}\,,$$

where s is a small constant that corresponds to the starting noise level.

For the noise schedule $\sqrt{1-\overline{\alpha_t}}$, sqrt schedule defines $\overline{\alpha_t}$ as:

$$\overline{\alpha_t} = 1 - \sqrt{\frac{t}{T} + s}\,,$$

where s is a small constant that corresponds to the starting noise level.

- When $t = 0$, $\overline{\alpha_t} = 1 - \sqrt{s}$, the $\sqrt{1-\overline{\alpha_t}}$ is $\sqrt[4]{s}$.

Other Noise (/Variance) Schedule

Ref: `https://huggingface.co/blog/annotated-diffusion`

- Linear Schedule
  - Example: original DDPM, set the forward process variances to constants increasing linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$.
  - Denoising Diffusion Probabilistic Models (NeurIPS'20)
- Cosine Schedule
  - The equation (usually divided by $\overline{\alpha_0}$ to normalize):

$$\overline{\alpha_t} = \left(\frac{\pi}{2} \times \frac{\cos(\frac{t}{T} + s)}{1 + s}\right)^2$$

  - Shown in Improved Denoising Diffusion Probabilistic Models (ICML'21) that cosine schedule achieves better results.

Diffusion LM: Noise Schedule

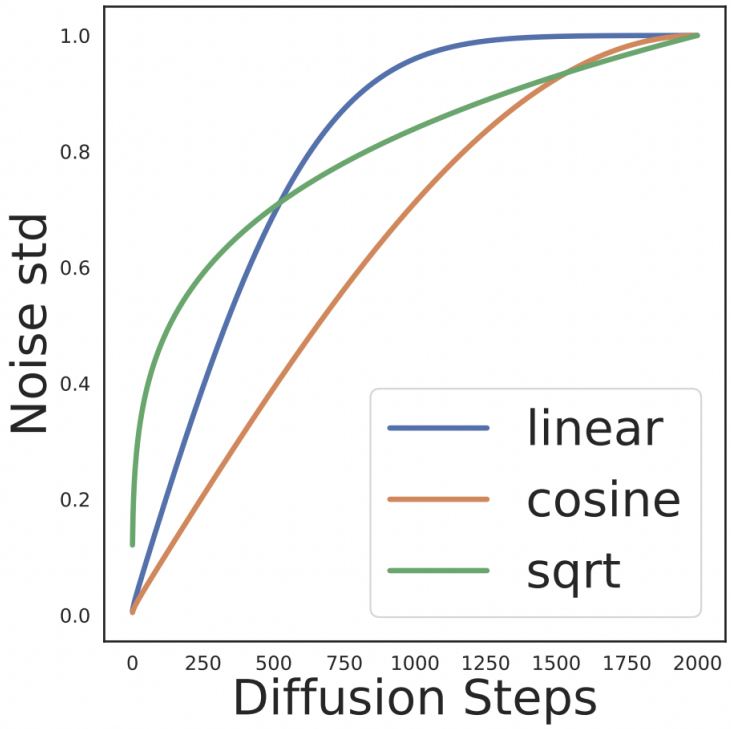

Figure: Noise Schedule. Visualizing the noise schedule $\sqrt{1-\overline{\alpha_t}}$. Figure 5 in the paper.

Diffusion LM: Noise Schedule

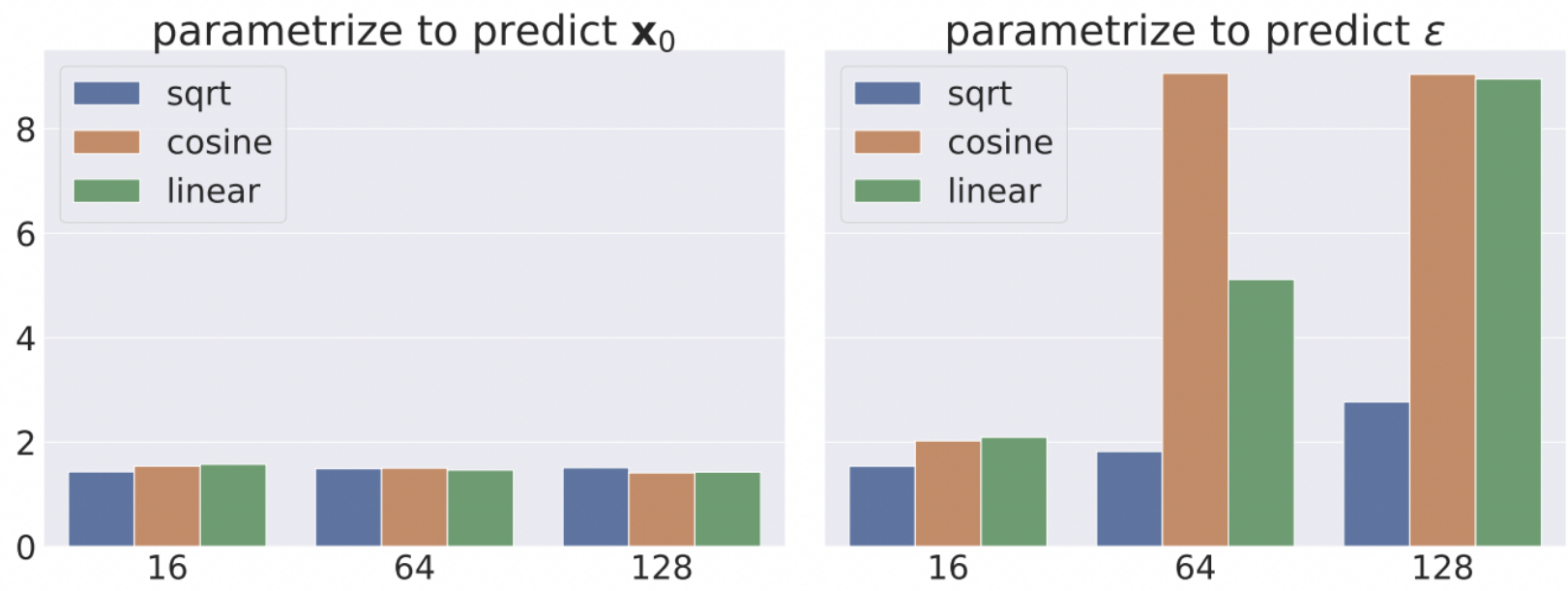

Figure: Noise Schedule, lm score. *sqrt* is the best. Figure 6 Row 2 in the paper. No longer salient when applying $\mathbf{x}_0$-parametrization trick.

Diffusion LM: Approximation

The simplified version of the canonical objective of maximizing

$$\mathbb{E}_{\mathbf{x}_0 \sim p_{data}} [\log p_\theta(\mathbf{x}_0)]$$

is the variational lower bound of $\log p_\theta(\mathbf{x}_0)$, [2] simplified as (NO LONGER a valid lower bound but mathematically proved and empirically more stable):

$$\mathcal{L}_{\text{simple}}(\mathbf{x}_0) = \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\|\mu_\theta(\mathbf{x}_t, t) - \hat{\mu}(\mathbf{x}_t, \mathbf{x}_0)\|^2 \right],$$

where $\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0)$ is the mean of posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$ which is a closed form Gaussian, and $\mu_\theta(\mathbf{x}_t, t)$ is the predicted mean of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, computed by a neural network (Transformer / U-Net).

Proof in Appendix E of **Diffusion-LM**. *Uses the closed form solution of the KL divergence of Gaussian.*

[2] the canonical objective comes from ICML'15

Diffusion LM: Training Objective

The simplified version of the canonical objective of maximizing $\mathbb{E}_{\mathbf{x}_0 \sim p_{data}}[\log p_\theta(\mathbf{x}_0)]$ is:

$$\mathcal{L}_{\text{simple}}(\mathbf{x}_0) = \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\|\mu_\theta(\mathbf{x}_t, t) - \hat{\mu}(\mathbf{x}_t, \mathbf{x}_0)\|^2 \right],$$

where $\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0)$ is the mean of $q(\mathbf{x}_{t-1}|\mathbf{x}_0, \mathbf{x}_t)$, and $\mu_\theta(\mathbf{x}_t, t)$ is the predicted mean of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The end2end version objective:

$$\begin{aligned}\mathcal{L}^{\text{e2e}}_{\text{simple}}(\mathbf{w}) =& \mathbb{E}_{q_\phi(\mathbf{x}_{0:T}|\mathbf{w})} \Big[\mathcal{L}_{\text{simple}}(\mathbf{x}_0) + \|\text{EMB}(\mathbf{w}) - \mu_\theta(\mathbf{x}_1, 1)\|^2 \\ & - \log p_\theta(\mathbf{x}_0|\mathbf{w}) \Big]\end{aligned}$$

Two options:

- Transformer: BERT-base model. Performs better, became the default choice.
- U-Net: make only one change to the standard U-Net, turning all 2D-convolutional layers into 1D-convolutional layers so that the model handles text **sequence** instead of image **matrix**.

Set diffusion steps to be $2,000$ in practice. When running multiple (i.e. 3), steps of optimization steps (i.e. Adagrad), for each diffusion steps, reduce $2,000$ to 200.

Recap: BERT-base

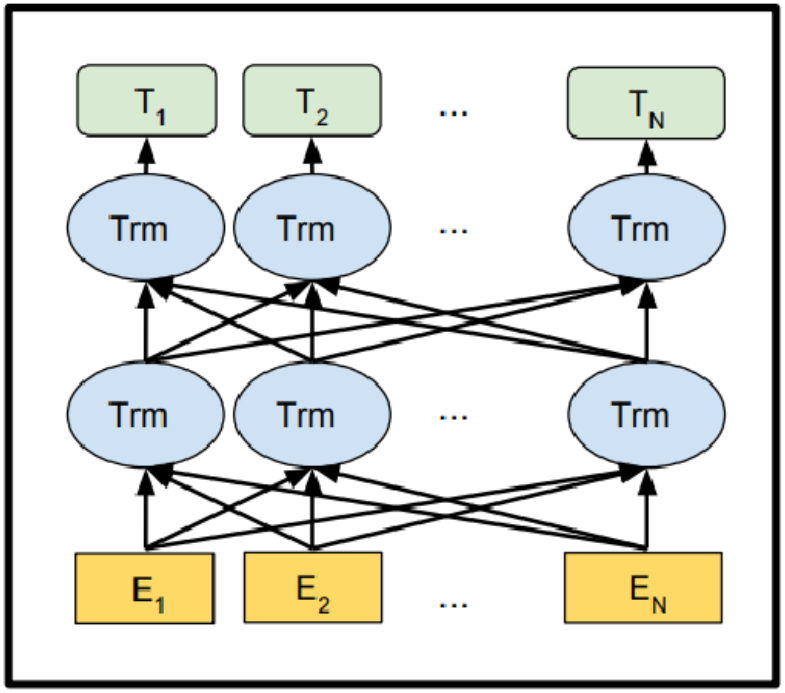

Figure: The architecture is the same as BERT-base, but Diffusion LM trained it from scratch. (link to BERT paper)

Recap: U-Net

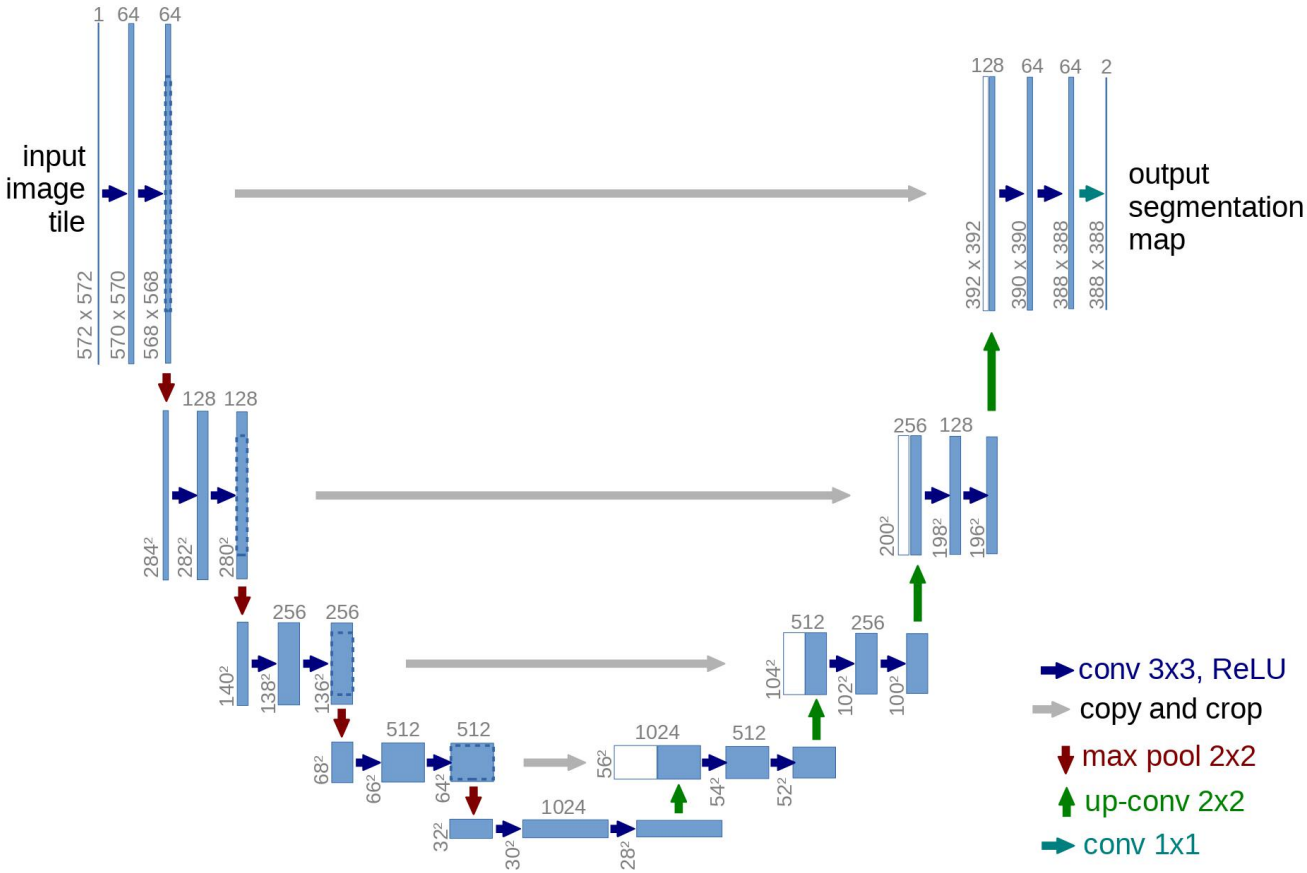

Figure: In Diffusion LM, we replace all the 2D conv layers with 1D conv layers. (link to U-Net paper)

Diffusion LM: Transformer v.s. U-Net

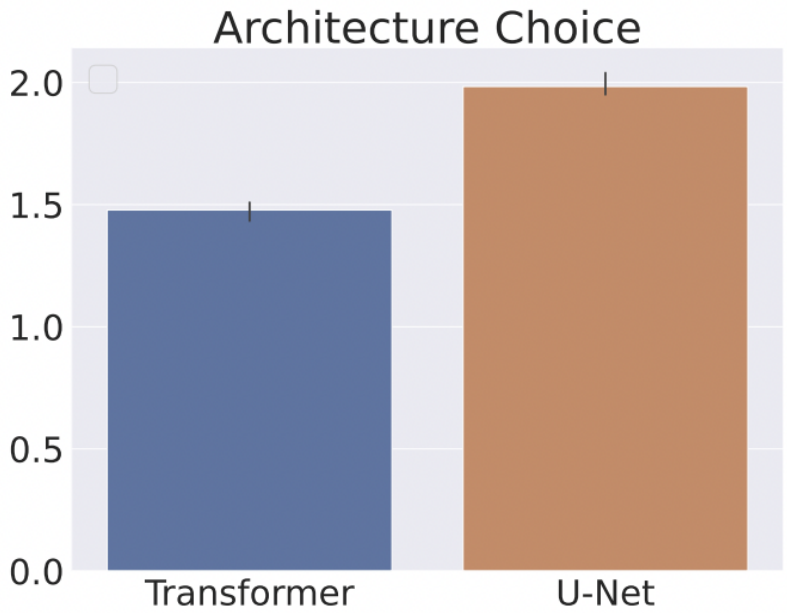

Figure: Transformer v.s. U-Net. Note that the less lm score, the better. Therefore, Transformer is better.

Diffusion LM: Embedding & Rounding

Embedding:

- $\text{EMB}(w_i) \in \mathbb{R}^d$ embed discrete word w_i into vector space.
- $\text{EMB}(\mathbf{w}) = [\text{EMB}(w_1), \dots \text{EMB}(w_n)] \in \mathbb{R}^{nd}$ denotes the embedding of a length-n sequence. A Markov transition is applied:

$$q_\phi(\mathbf{x}_0|\mathbf{w}) = \mathcal{N}(\text{EMB}(\mathbf{w}), \sigma_0 I)\,,$$

which is trained end-to-end with the other components.

Rounding: achieved by choosing the most probable word according to

$$\arg\max p_\theta(\mathbf{w}|\mathbf{x}_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i)\,,$$

where $p_\theta(w_i|x_i)$ is a softmax distribution.

Diffusion LM: Learned v.s. Random Embeddings

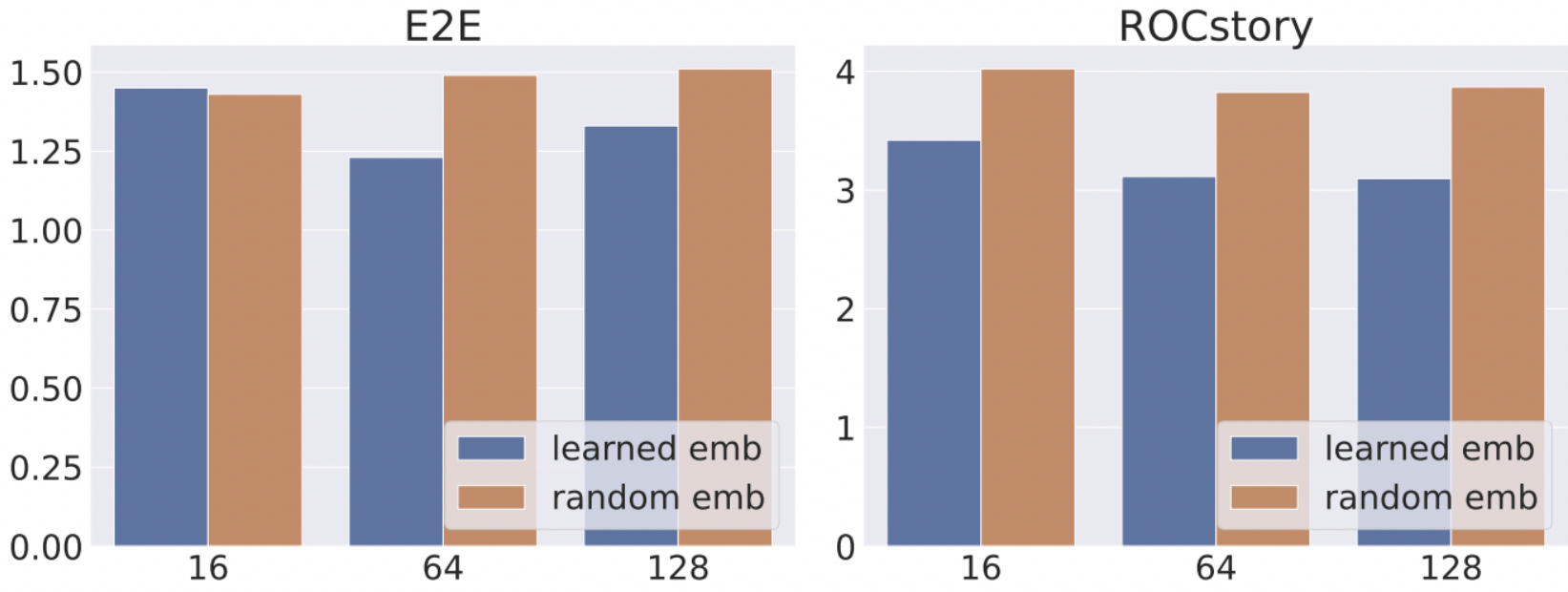

Figure: Learned v.s. Random Embeddings. Note that the less lm score, the better. Learned Embedding is better. Figure 6 Row 1 in the paper. Conclusion: Fixed pre-trained embedding or random Gaussian embeddings are worse than the **embedding trained via an end-to-end framework**.

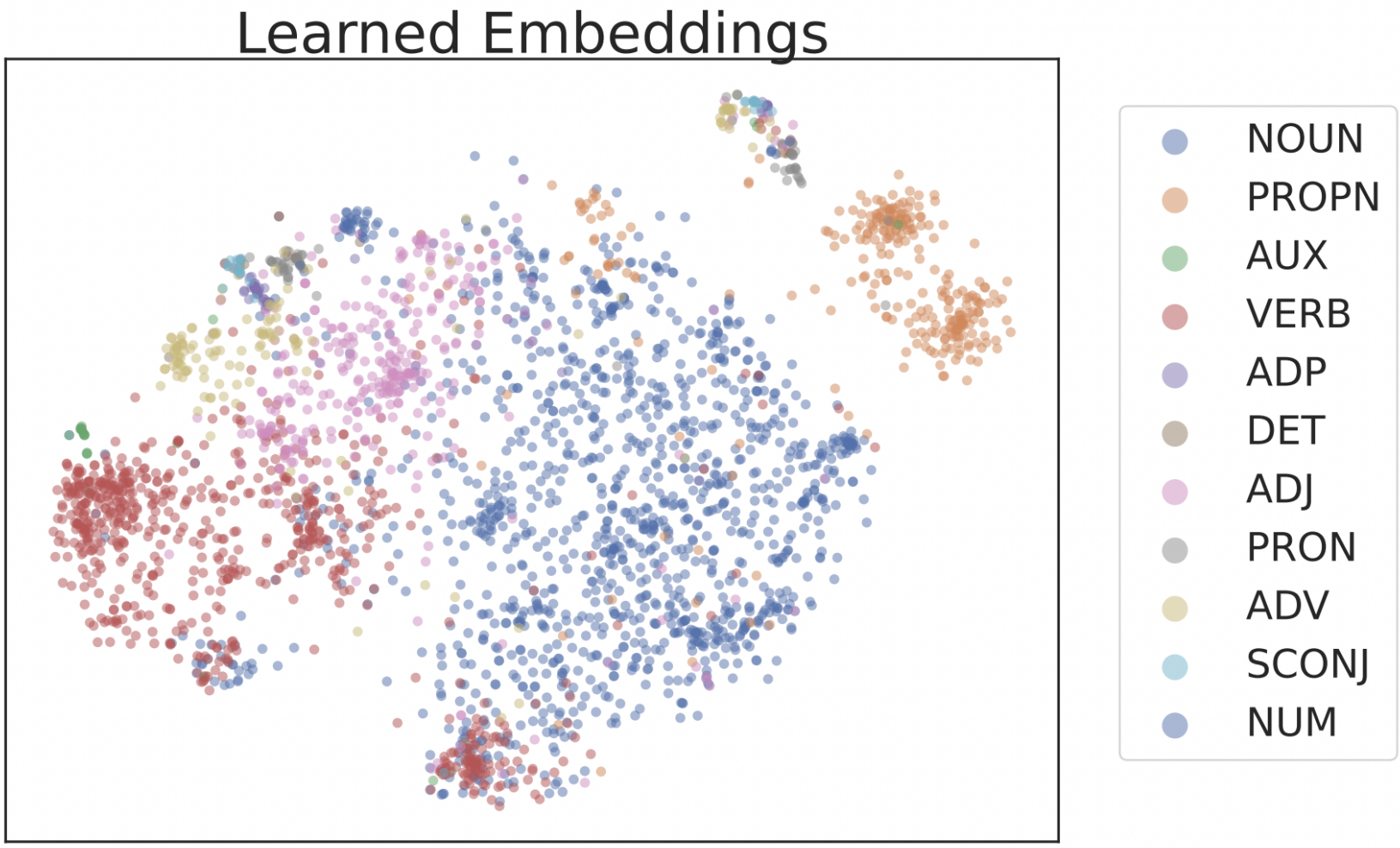

Figure: Figure 3 in the paper: A t-SNE plot of the learned word embeddings. Each word is colored by its POS (part-of-speech).

Recall that rounding is achieved by choosing argmax of

$$p_\theta(\mathbf{w}|\mathbf{x}_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i)\,,$$

where $p_\theta(w_i|x_i)$ is a softmax distribution. Ideally, this rounding is sufficient to map back to **discrete** text. The denoising steps should make $\mathbf{x}_0$ lie exactly on the embedding of some word.

Recall that rounding is achieved by choosing argmax of

$$p_\theta(\mathbf{w}|\mathbf{x}_0) = \prod_{i=1}^{n} p_\theta(w_i|x_i)\,,$$

where $p_\theta(w_i|x_i)$ is a softmax distribution. Ideally, this rounding is sufficient to map back to **discrete** text. The denoising steps should make $\mathbf{x}_0$ lie exactly on the embedding of some word.

But the problem is that, empirically, the model fails to generate $\mathbf{x}_0$ that commits to a single word.

One Explanation: not emphasizing single commit on $\mathbf{x}_0$ enough.

$$\mathcal{L}_{\text{simple}}(\mathbf{x}_0) = \sum_{t=1}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\|\mu_\theta(\mathbf{x}_t, t) - \hat{\mu}(\mathbf{x}_t, \mathbf{x}_0)\|^2 \right]$$

- The model $\mu_\theta(\mathbf{x}_t, t)$ directly predicts the mean of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ at every denoising step t.
- The constraint that $\mathbf{x}_0$ has to commit to a single word embedding will only appear in the terms with t near 0, and require careful tuning to emphasize those terms. (Appendix H)
- Quick Fix: $\mathbf{x}_0$-parameterized model

$$\mathcal{L}^{\text{e2e}}_{\mathbf{x}_0-\text{simple}}(\mathbf{x}_0) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t} \|f_\theta(\mathbf{x}_t, t) - \mathbf{x}_0\|^2 .$$

Diffusion LM: Result of the Quick Fixation

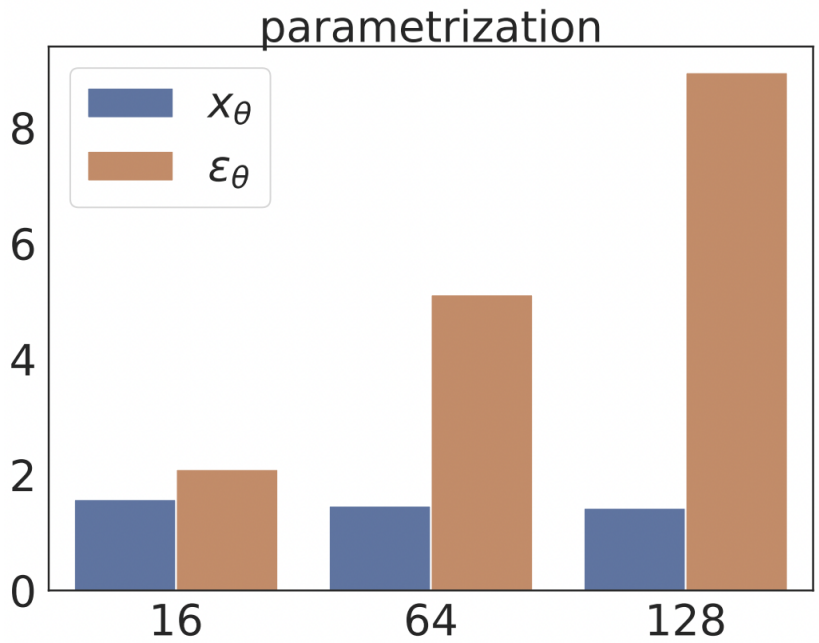

Figure: Figure 4 right half in the paper. Parametrizing by $\mathbf{x}_0$ consistently performs well, whereas parametrizing by ϵ works fine for small dimensions, but quickly collapses for larger dimensions.

$\mathbf{x}_0$-parameterized model:

$$\mathcal{L}^{\text{e2e}}_{\mathbf{x}_0-\text{simple}}(\mathbf{x}_0) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t} \|f_\theta(\mathbf{x}_t, t) - \mathbf{x}_0\|^2 ,$$

where our re-parameterized model $f_\theta(\mathbf{x}_t, t)$ learns $\mathbf{x}_0$ directly.

In the decoding phase, same intuition could be used in **clamping trick**:

- Maps the predicted $f_\theta(\mathbf{x}_t, t)$ to its **nearest** word embedding sequence at **every** step.
- Forces the predicted vector to commit to a word for intermediate diffusion steps, making predictions more precise and reducing rounding errors.

Diffusion LM: Clamping Trick

Before clamping trick:

$$\mathbf{x}_{t-1} = \sqrt{\overline{\alpha_t}} f_\theta(\mathbf{x}_t, t) + \sqrt{1 - \overline{\alpha_t}}\epsilon\,,$$

After clamping trick:

$$\mathbf{x}_{t-1} = \sqrt{\overline{\alpha_t}} \cdot \mathrm{Clamp}\big(f_\theta(\mathbf{x}_t, t)\big) + \sqrt{1 - \overline{\alpha_t}}\epsilon\,,$$

Controlling $\mathbf{x}_{0:T}$ over $\mathbf{c}$ is equivalent with decoding from the joint inference problem posterior:

$$p(\mathbf{x}_{0:T}|\mathbf{c}) = \prod_{t=1}^{T} p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) \,,$$

which can be decomposed to a sequence of control problems at each diffusion step:

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) \propto p(\mathbf{x}_{t-1}|\mathbf{x}_t) \cdot p(\mathbf{c}|\mathbf{x}_{t-1}, \mathbf{x}_t)$$

And according to a Yang Song et al's paper from ICLR'21 (Section 5), there are *conditional independence assumptions* that we can use to simplify:

$$p(\mathbf{c}|\mathbf{x}_{t-1}, \mathbf{x}_t) = p(\mathbf{c}|\mathbf{x}_{t-1})$$

Therefore, for the t-th step, we run gradient update on $\mathbf{x}_{t-1}$:

$$\nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}) \,,$$

where the two differentiable terms are parameterized by:

- $\log p(\mathbf{x}_{t-1}|\mathbf{x}_t)$: Diffusion LM (for **fluency**);
- $\log p(\mathbf{c}|\mathbf{x}_{t-1})$: an arbitrary neural network classifier (for **control**);

respectively.

Therefore, for the t-th step, we run gradient update on $\mathbf{x}_{t-1}$:

$$\nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{c}) = \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}) \,,$$

where the two differentiable terms are parameterized by:

- $\log p(\mathbf{x}_{t-1}|\mathbf{x}_t)$: Diffusion LM (for **fluency**);
- $\log p(\mathbf{c}|\mathbf{x}_{t-1})$: an arbitrary neural network classifier (for **control**);

respectively.

In practice, we add *fluency regularization* where λ is a hyper-param:

$$\lambda \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{x}_{t-1}|\mathbf{x}_t) + \nabla_{\mathbf{x}_{t-1}} \log p(\mathbf{c}|\mathbf{x}_{t-1}) \,,$$

Steps:

1. Collect a set of samples $\mathcal{S}$ drawn from the Diffusion-LM, instead of having one single best option.
2. Select the sample that achieves the minimum expected risk under a given loss function (e.g., negative BLEU score).

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w} \in S} \sum_{\mathbf{w}' \in S} \frac{1}{|S|} \mathcal{L}(\mathbf{w}, \mathbf{w}')$$

3. A low quality sample would be dissimilar from the remaining samples, and penalized by the loss function.

Diffusion LM: Highlighted Result

	Semantic Content		Parts-of-speech		Syntax Tree		Syntax Spans		Length	
	ctrl ↑	lm ↓	ctrl ↑	lm ↓	ctrl ↑	lm ↓	ctrl ↑	lm ↓	ctrl ↑	lm ↓
PPLM	9.9	5.32	-	-	-	-	-	-	-	-
FUDGE	69.9	2.83	27.0	7.96	17.9	**3.39**	54.2	4.03	46.9	3.11
Diffusion-LM	**81.2**	**2.55**	**90.0**	**5.16**	**86.0**	3.71	**93.8**	**2.53**	**99.9**	**2.16**
FT-sample	72.5	2.87	89.5	4.72	64.8	5.72	26.3	2.88	98.1	3.84
FT-search	89.9	1.78	93.0	3.31	76.4	3.24	54.4	2.19	100.0	1.83

Table 2: Diffusion-LM achieves high success rate (ctrl ↑) and good fluency (lm ↓) across all 5 control tasks, outperforming the PPLM and FUDGE baselines. Our method even outperforms the fine-tuning oracle (FT) on controlling syntactic parse trees and spans.

Diffusion LM: Highlighted Result

Syntactic Parse	(S (S (NP *) (VP * (NP (NP * *) (VP * **(NP (ADJP * *) *)**))))) * (S (NP * * *) (VP * (ADJP (ADJP *)))))
FUDGE	Zizzi is a cheap restaurant . **[incomplete]**
Diffusion-LM	Zizzi is a pub providing **family friendly Indian food** Its customer rating is low
FT	Cocum is a Pub serving **moderately priced meals** and the customer rating is high
Syntactic Parse	(S (S (VP * (PP * (NP * *)))) * **(NP * * *)** (VP * (NP (NP * *) (SBAR (WHNP *) (S (VP * (NP * *)))))) *)
FUDGE	In the city near The Portland Arms is a coffee and fast food place named The Cricketers which is not family - friendly with a customer rating of 5 out of 5 .
Diffusion-LM	Located on the riverside , **The Rice Boat** is a restaurant that serves Indian food .
FT	Located near The Sorrento, **The Mill** is a pub that serves Indian cuisine.

Table 3: Qualitative examples from the Syntax Tree control. The syntactic parse tree is linearized by nested brackets representing the constituents, and we use the standard PTB syntactic categories. Tokens within each span are represented as * . We color failing spans red and **bold** the spans of interest that we discuss in §7.1.

FT refers to fine-tuned GPT-2 without plug-and-play setting.