Diffusion Models in Natural Language Processing

Li Kaiyu, [1] Chen Lei, [1] Li Jiayi, [1] Wan Zhang [1]

[1] Nanyang Technological University
{s220048, chen1552, jli105, zwan004}@e.ntu.edu.sg

Abstract

Diffusion models have been widely used in generative tasks and have shown their ability to output required information with high quality. However, they are rarely used for NLP tasks. In this report, we compare the performance of Multinomial Diffusion with transformer and GPT-2 on the text generation task to test the potential of using diffusion models in the NLP field and find that Multinomial Diffusion is not as efficient as auto-regressive models on the text8 dataset.

Introduction

With the development of the Internet and social media, information is generated at a large scale everywhere, taking the text as the form mostly. It is attractive to create text based on human language. Text generation is a process of creating a new, logical, and consistent text which is similar and content to existing text. Generally, autoregressive language models or seq2seq models are used for the text generation task(Fedus, Goodfellow, and Dai 2018) and the auto-regressive models have dominated this task for a long time.
At the same time, diffusion generative models incur application in many fields like computer vision and molecular generation. However, it is rare to see that diffusion models deal with some NLP tasks like text generation as usually a set of discrete data is needed as the input for related tasks. Considering diffusion models' good performance, we explore the potential of diffusion models in the NLP field to form useful conclusions and insights for future work direction.
In this report, a diffusion model defined on categorical variables called Multinomial Diffusion is chosen for the text generation task. The Multinomial Diffusion is a continuous diffusion model gradually adding noises and trying to learn the denoising process. We compare the Multinomial Diffusion model's performance on the text8 dataset to a transformer as the baseline and GPT-2(Radford et al. 2019). We found that the Multinomial Diffusion model with sufficient training data can perform better than transformers, but considering the large number of input tokens needed, the diffusion model is not as efficient as auto-regressive models. Also, the auto-regressive model fine-tuned with a pre-trained model like GPT-2 showed an amazing ability to generate high-quality texts.

Related Work

Auto-Regressive Generation

As the mainstream paradigm, autoregressive models have been dominating language modeling for a long time. It generates one token (e.g. a word or a character) at a time, based on a given sequence of input text. The basic idea behind autoregressive models is to predict the probability distribution of the next word or character in a sequence, given the previous words or characters in the sequence.

Denoising Diffusion Generation

Denoising Diffusion Probabilistic Model (DDPM) (Sohl-Dickstein et al. 2015; Ho, Jain, and Abbeel 2020) is a newly emerged generative norm that gradually adds noise over time steps. The model is composed of predefined variational distributions that add small amounts of noise around the previous step. This process destroys information so that the final step carries almost no information about the initial data. The generative counterparts of diffusion models consist of learnable distributions that are trained to remove noise from the data. When small amounts of noise are added, it is possible to use factorised distributions to define the denoising trajectory over the dimension axis.

Diffusion Models in Text Generation

We have seen a recent surging focus on diffusion models. Multiple studies that apply the denoising diffusion generative model in generation tasks in natural language processing (Savinov et al. 2021; Reid, Hellendoorn, and Neubig 2022; Hoogeboom et al. 2021; Austin et al. 2021; Li et al. 2022) have been proposed. All of these papers use denoising diffusion methods to generate text based on the input sequence of words, while they try various ideas to transform the discrete words into a chain of noise distribution as needed. For example, in(Savinov et al. 2021), they replace a random part of tokens in the sentence and unroll the corrupted samples, thus can calculate the noise generated.

Methods

Transformer

Transformer is a groundbreaking introduction to the self-attention and positional encoding methods with the first introduced self-attention mechanism and positional embedding by Vaswani et al. in 2017 ((Vaswani et al. 2017)).
Transformer essentially is an encoder-decoder structure. The encoder consists of multiple layers. The input data will first enter the "self-attention" block to get a weighted eigenvector Z as $Attention(Q, K, V)$:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1}$$

For "self-attention", each input will have 3 different vectors - Query(Q), Key(K), and Value(V), which are given by 3 different weight matrices from the embedding vector X times 3 different weight matrices, W^Q, W^K, W^V.
Obtaining Z, it will be sent to the next block - feed forward neural network with two layers - ReLU activation function, and linear activation function. Now, We have gone through the structure of the encoder.
The structure of the decoder is "self-attention" to "encoder-decoder attention" to "feed forward neural network". For, the new "encoder-decoder attention", its Q is from the output of the "self-attention" block of the decoder, and K and V are from the output of the encoder. Since the decoding process is a sequential operation, that is, when decoding the K_{th} feature vector, only the decoding results of K-1 and those before masked, multi-head attention, in this case, is called masked multi-head attention.
Positional Encoding: The Transformer model does not have the ability to capture sequential sequences, meaning that no matter how the sentence structure is shuffled, the transformer will get a similar result. In order to solve this problem, it introduces a positional encoding method achieved through a sinusoidal function that generates a unique positional encoding vector for each token's position in the sequence. This vector is then added to the token's original embedding vector before being fed into the model. The sinusoidal function consists of sine (Equation 2) and cosine (Equation 3) functions with different frequencies, allowing the model to easily learn and distinguish between positions.

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}}) \tag{2}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}}) \tag{3}$$

After the decoder decodes, the decoded feature vector passes through a fully connected layer with a softmax activation function to obtain an output vector that reflects the probability of each word or character.

Preliminaries

A diffusion model is a kind of latent variable model. At a timestep, the model adds noise in a fixed noising process q e.g. diagonal Gaussian noise. Given that the input data from a real data distribution $X_1, ...X_t$ and the noising process q adds Gaussian noise through X_{t-1} to X_t with a hyperparameter $\beta_t \in (0, 1)$ referring to the amount of noise added at random time t as follows:

$$q(X_t|X_{t-1}) = \mathcal{N}(X_t; \sqrt{1-\beta_t}X_{t-1}, \beta_t\mathbf{I}) \tag{4}$$

As time step$t$ increases, the data sample X_0 losses its feature gradually. If a sufficiently large time T is given and the hyperparameter β_t is suitable, X_T can be close to Gaussian Distribution $X_T\, \mathcal{N}(0, \mathbf{I})$. As the equation4 describes, we can sample a random step of the noised latent variables dependent on the input X_0 directly.By using reparameterisation trick, assume that $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, the marginal distribution can be:

$$q(X_t|X_0) = \mathcal{N}(X_t; \sqrt{\overline{\alpha}_t}X_0, (1 - \overline{\alpha_t})\mathbf{I}) \tag{5}$$

If it is possible to reverse the noising process $q(X_t - 1|X_t)$, we can approximate a sample as in the input noise. Suppose that we know the procedure of denoising the sequence of latent variables X_t to X_1, we can reverse the noising process p and approximate a sample from the original data distribution. However, $q(X_t - 1|X_t)$ is not easy to estimate as it needs the entire dataset. So it instead predicts the probability by p_θ, which is written as:

$$p_\theta(X_0 : T) = p(X_T)\prod_{t=1}^{T} p_\theta(X_t - 1|X_t) \tag{6}$$

$$p_\theta(X_{t-1}|X_t) = \mathcal{N}(X_{t-1}; \mu_\theta(X_t, t), \sum_\theta(X_t, t)) \tag{7}$$

The objective of the diffusion model is to find the transitions maximising the likelihood of the training data. When training in practice, the target of the diffusion model can minimise the variation upper bound:

$$\mathcal{L}_{vlb} = E_{q(X_{0:T})}[log\frac{q(X_{1:T}|X_0)}{p_\theta(X_{0:T})})] \geq -E_{q(X_0)}logp_\theta(X_0) \tag{8}$$

Under that circumstance, the objective is risky to stabilise. (Ho, Jain, and Abbeel 2020) proposes a simpler objective re-weighting every D_KL in $\mathcal{L}_{vlb}$ to obtain a MSE loss:

$$\mathcal{L}_{simple}(X_0) = \sum_{t=1}^{T} E_{q(X_t|X_0)}||\mu_\theta(X_t, t) - \hat{\mu}(X_t, X_0)||^2 \tag{9}$$

where $\hat{\mu}(X_t, X_0)$ stands for the mean of the noise's value added by the process q and $\mu_\theta(X_t, t)$ means the mean of the mean value predicted by the neural network.

Multinomial Diffusion

For a text generation task, the input data is always discrete as it contains single characters and words. A diffusion model needs a set of continuous data close to Gaussian distribution as input. Multinomial Diffusion is a kind of diffusion model designed for categorical variables. It converts the words into a one-hot encoded format. Compared to the X_t described in the former section, it is represented in $X_t \in (0, 1)^K$. For a specific category k, $X_k = 0$ while $X_j = 0$ for every $j \neq k$. Based on the definition, we can define the categorical distribution process as:

$$q(X_t|X_{t-1}) = \mathcal{C}(X_t|(1 - \beta_t)X_{t-1} + \beta_t/K) \tag{10}$$

where $\mathcal{C}$ stands for a categorical distribution with the probability after the symol $|$ and β_t means the change that a category is resampled evenly. Suppose that the input data $X_0, ..., X_t$ is also from a real distribution , given that $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{\tau=1}^{t} \alpha_\tau$, the probability of X_t given X_0 can be calculated by:

$$q(X_t|X_0) = \mathcal{C}(X_t|\overline{\alpha}_t X_0 + (1 - \overline{\alpha}_t)/K) \quad (11)$$

At every timestep, an inference diffusion process adds noise β_t over K classes and samples the former value $X_t - 1$ with $1 - \beta_t$. By combining the Equation10 and 11 we can calculate the categorical posterior by the distribution with the previous probability $\theta(X_t, X_0)$:

$$q(X_t - 1|X_t, X_0) = \mathcal{C}(X_t - 1|\theta(X_t, X_0)) \quad (12)$$

Multinomial Diffusion uses the posterior probability to predict noise and calculate the approximating model based on the predicted probability vector. To ensure the prediction generated by the neural network isn't negative and sums to one, a softmax function μ is used. The denoising process can be written as:

$$p(X_t - 1|X_t) = \mathcal{C}(X_t - 1|\theta_{post}(X_t, \mu(X_t, t))) \quad (13)$$

Experiments

Data

In our experiment we use text8 from http://mattmahoney.net/dc/textdata.html as our benchmark. The text8 dataset contains the first 10^9 bytes of English Wikipedia, approximately 17 million single words of English text. The text8 dataset has removed certain types of formatting and markup, which is widely used for training models to predict the next word in sequence based on the previous word.

Metric

We employ Bits per character (BPC) as an evaluation metric. BPC is widely used to measure the efficiency of text compression algorithms. It measures the average number of bits needed to represent a single character of a given text using the compression algorithm. The lower the BPC value, the more efficient the compression algorithm.
BPC is typically calculated by dividing the total number of bits required to encode the text by the number of characters in the text. The total number of bits required is determined by the size of the compressed file, which is a function of the compression algorithm and the specific text being compressed.
BPC is widely used in NLP tasks, such as language modeling and text generation, as it provides a standardised measure of the quality of the generated text. For example, a language model with a low BPC value is able to generate text that is more concise and easier to store, transmit, and process than a language model with a high BPC value.

Transformer

We trained on the standard text8 dataset using Transformer to work as a baseline for the auto-regressive character-level text generation task.

1. Model settings: We build the transformer based on the nn.module in PyTorch, which is built totally based on the vanilla transformer.
   - Transformer Model: The transformer model class defines the overall architecture of the Transformer model. It initialises the necessary components, including the positional encoder, the transformer encoder, masking, input embeddings, and the output decoder.
   - PositionalEncoding: We built a PositionalEncoding class responsible for adding positional information to the input embeddings. The positional encoding uses a combination of sine and cosine functions to create a unique encoding for each position in the sequence, which is then added to the input embeddings.
   - Transformer Encoder: We directly apply the TransformerEncoder in the nn.module, which consists of multiple layers of TransformerEncoderLayer with multi-head attention.
   - Masking: We wrote a "_generate_square_subsequent _mask" function in the model class for generating a square mask to prevent the model from attending to future positions during training.
   - Transformer Decoder: As the transformer works for auto-Regressive generation, we don't need the decoder. Therefore, we set the decoder by one linear layer.
   - The total number of trainable parameters is **18,941,979**.
   - Parameters: The key parameters used in our model are shown below with values and descriptions:

Field	Value	Description
ntoken	27	Number of tokens in the vocabulary.
d_model	512	Dimension of the input embeddings and the model's hidden states.
nhead	8	Number of attention heads in the multi-head attention mechanism.
nhid	2048	Dimension of the feed-forward networks inside the transformer layers.
nlayers	6	Number of layers in the transformer encoder

Table 1: Transformer - Parameters

2. Train:
   - Optimiser: We used the Adam optimiser with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 1e - 9$. We adjust the learning rate by the 'ReduceLROnPlateau' scheduler with an initial value of 1e-4, which reduces the learning rate when the monitored metric plateaus or stops improving.
   - Loss Criterion: We chose "CrossEntropyLoss" here to calculate the bpc for evaluation. In each batch, calculate the loss by comparing the model outputs and the targets using the given criterion. The outputs and targets are reshaped to match the expected dimensions of

the loss function by updating the accumulated loss and accuracy metrics then. Finally, return the average loss by dividing the total loss by the number of batches as the final bpc value.

For our transformer model using the hyperparameters described, each training step took about 20 minutes. We trained the base models for no more than 50 epochs or 16 hours.

3. Results: On the text8 auto-regressive character-level text generation task, the model, trained by the first 100,000 characters in the text8 dataset, converges within 30 epochs and gives a best validation bpc of 2.3768, and the model, trained by the first 1,000,000 characters in the text8 dataset, converges within 25 epochs and gives a best validation bpc of 2.3604. It shows initial results that a larger dataset gives better performance. Because of the time limit, we don't test the transformer model with the whole text8 dataset. Finally, we choose the one trained by the first 1,000,000 characters as the baseline of our project. The loss curves are shown in Figure 1.

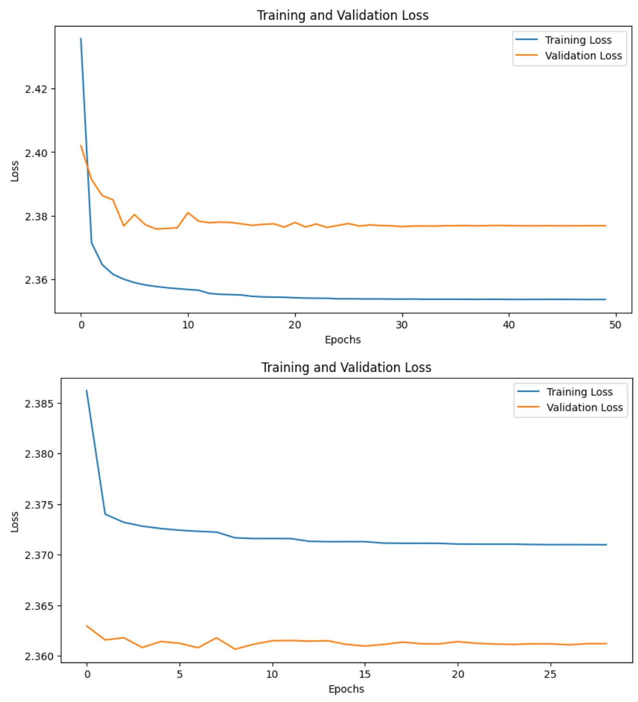

Figure 1: Transformer - Loss Curve (Top: First 100,000 characters; Down: First 1,000,000 characters)

GPT-2

Apart from the baseline transformer, we also conduct experiments on GPT-2, which is a considerably larger transformer-based model, with fine pre-trained weights on large datasets free to use. We run **two** different experiments on GPT-2: Finetune the pre-trained GPT-2 model on the text8 dataset and Train a GPT-2 model with only the text8 dataset.

1. Model Settings: We mostly leverage existing APIs for GPT-2 on Huggingface, and key model settings are described as follows. We adopt default settings for those that are not listed below.
   - Tokenizer: Instead of the word tokenizer commonly used, we build a new character-level tokenizer which is specifically designed for the text8 dataset. We have 27 kinds of character tokens in total, containing characters in alphabet and space character.
   - Pre-trained Model: We adopt the Hugging Face pre-trained weights for GPT2LMHeadModel directly.
   - The total number of trainable parameters is **124,439,808**.
2. Training Configurations:
   - Optimiser: Adam optimiser with initial learning rate equals 1e-5.
   - Loss Criterion: Consistent with the Transformer Baseline method.

     For both settings (using pre-trained GPT-2 or not), we train each model on a single Nvidia Tesla V100 card for 30 epochs. Both models converged by the end of training.

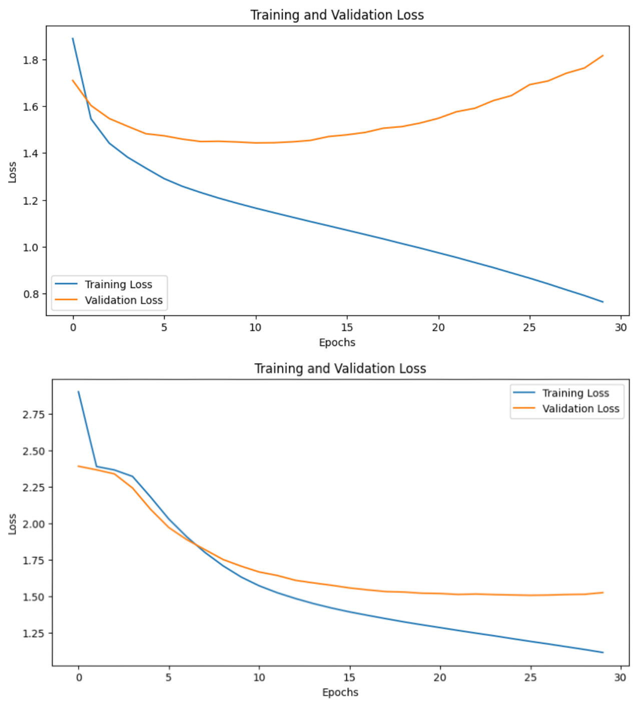

Figure 2: Transformer - Loss Curve (Top: Finetuning pre-trained GPT-2 model on text8; Down: Training GPT-2 model on text8 from scratch)

3. Results: Specifically, we finetune a pretrained GPT-2 model on text8 with the first 1,000,000 characters, and also train a GPT-2 from scratch with the same amount of data. Figure 2 shows the loss curves of both experiments.

In comparison, adopting pre-trained GPT-2 weights increase the speed of convergence significantly (converges within 10 epochs), and gains an optimal validation loss of 1.443. While training GPT-2 from scratch takes longer to converge (takes around 25 epochs), the optimal performance is 1.511 in terms of bpc validation loss, which is degraded compared with the pre-trained model.

Multinomial Diffusion

We employ the Multinomial Diffusion model to generate characters on the text8 dataset.
For model settings, we build the transformer based on the nn.module in PyTorch, which is built totally based on the vanilla transformer. The model parameters are the same as discussed in . The key parameters used in our model are shown below with values and descriptions: We first train the

Field	Value
batch size	32
lr	1e-4
epochs	30
diffusion steps	1000
scheduler	ExponentialLR
multiplicative factor	0.99

Table 2: Diffusion - Parameters

Multinomial Diffusion model for 30 epochs, achieving training bpc 4.06397 and validation bpc 4.05751. The results are apparently lower than that of Transformer baselines. Therefore, we extend the training epochs to 1000, whose learning curves are as shown in Figure 3 and 4. It takes around 235 minutes to train a total of 1000 epochs. A training bpc 3.12599 and validation bpc 3.17823 are achieved. To

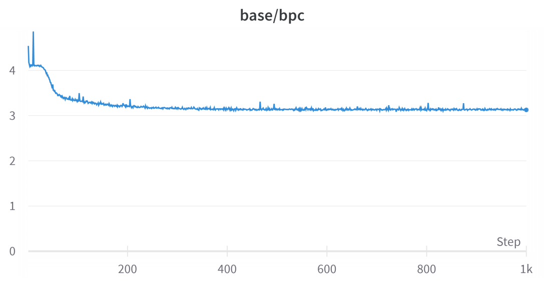

Figure 3: Training Curve
.

step further, we explore the effect of data magnitude. We, therefore, employ 90 million data for training. The learning curves are as indicated in Figure 5 and 6. However, due to the training costs, we only train for 25 epochs, achieving a training bpc of 2.05 and validation bpc of 2.028. The whole training process takes less than 17 hours.

Comparisons

As indicated in Table 3, the pre-trained GPT-2 achieves the best overall performance among all settings. The multinomial Diffusion model, when trained using the same number of epochs and data, shows an obvious disadvantage compared to autoregressive models. However, the experimental results indicate that extending epochs and training data can largely improve performance. For GPT-2, it is not surprising that the pre-trained version demonstrates superior performance among all models. However, that even non-pretrained GPT-2 achieves quite a solid score convinces us of the importance of parameter quantity.

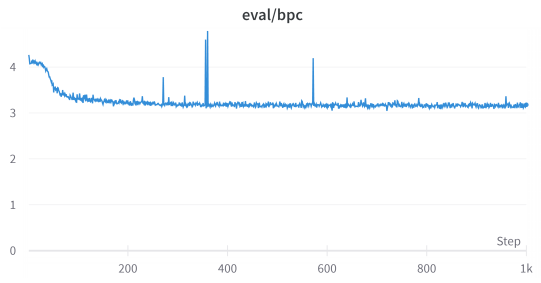

Figure 4: Validating Curve
.

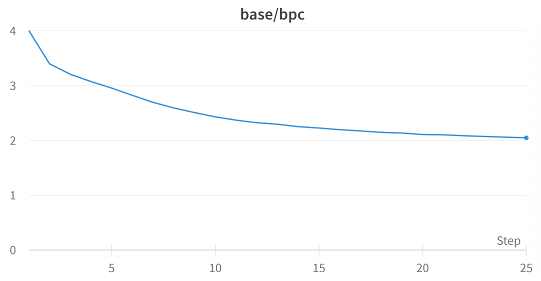

Figure 5: Training Curve
.

Analysis

In this section, we perform a qualitative analysis of the text generation results of the models. A set of example outputs are given below. Since diffusion models do not require an input sequence to start generation, we first generate a 100-character long text using the diffusion model, and then utilise the beginning part of the text as input to autoregressive models to generate text of the same length:

- Transformer Baseline: **that the role of** *ase t cat in sin tend come cor at as tinialof atisin icon anghe tiong tharase s icounte alllichr tom*
- GPT-2 Trained from Scratch: **that the role of** the *movie that it is a moral and the soul and the soul and the us open australia as a fertile of th*
- GPT-2 Finetuned: **that the role of** *the state is the state which is considered to be a more popular than the state which is the state w*
- Multinomial Diffusion: **that the role of** *tellings not be required also action characters passe d on constitution*

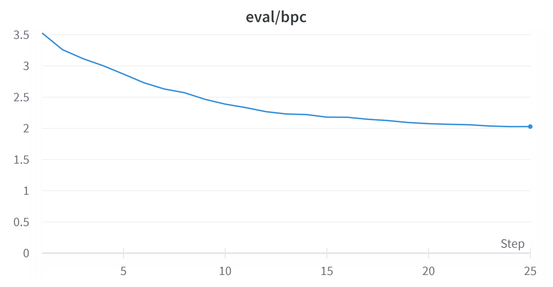

Figure 6: Validating Curve

.

	Epochs	Data	Train BPC	Val BPC
Transformer	30	0.1M	2.358	2.376
Transformer	30	1M	2.371	2.360
GPT-2	30	1M	1.211	1.511
GPT-2 p.t.	30	1M	1.164	1.443
Multi.Diff.	30	1M	4.064	4.058
Multi.Diff.	1000	1M	3.126	3.178
Multi.Diff	25	90M	2.050	2.028

Table 3: Comparative Results

ahmad a nob

Note that all the 3 auto-regressive models were trained with 1M text8 data, while the diffusion model was fully trained with 90M text8 data. In terms of word level, the baseline transformer model struggles to generate valid English words, the Multinomial Diffusion model managed to predict most of the words right, whereas GPT-2 models generate every word correctly. In terms of sentence level, the diffusion model and non-pretrained GPT-2 model contain many grammar mistakes while the pre-trained GPT model only shows few. From these results, we argue that only by fully trained on large enough data can the Multinomial Diffusion model outperform the transformer baseline and be comparable with GPT-2 models. This is because auto-regressive models have higher efficiency on text generation tasks. Furthermore, the pre-trained GPT-2 model gains the best text generation quality, due to abundant prior knowledge acquired from pre-trained weights.

Discussion

In this section, we mainly discuss the problems and futures of diffusion models on generation tasks in natural language processing.

Denoising diffusion generative models have achieved remarkable generation quality in computer vision (Croitoru et al. 2023). However, in this report, we empirically study the performance of a diffusion model on a benchmark dataset, which forms a conclusion that the diffusion-based generative model performs worse than mainstream autoregressive models. This drives us to ask a question: why is the diffusion model better at image generation than text generation?

We notice an inherent difference between the image domain and the text domain. Although images are represented as discrete pixels, the theoretical modeling of images is still based on a continuous domain. In image generation, the diffusion process can be thought of as a sequence of blurring operations, where each iteration produces a smoother image with less noise. This approach can lead to high-quality and realistic images.

In comparison, texts are naturally represented in discrete, which poses challenges to the noise addition process. As such, a mandatory step to generate text using the diffusion model is transforming discrete text into continuous representation. As shown in (Hoogeboom et al. 2021), the text data are lifted to multinomial distribution at the character level. In (Li et al. 2022), the word embeddings are trained in an end-to-end fashion to project original discrete text data onto a continuous space. However, the noise added is still continuous Gaussian in both above papers that operate on hidden states. The noising and denoising are too indirect to efficiently corrupt the data since it is neither controllable to disturb original data nor explainable and perceivable for humans.

Therefore, we argue that the added noise and denoising must fit the inherent generation processes for different data types. For language generation, an intuitive approach is an editing-based perturbation (or noise addition). In (Reid, Hellendoorn, and Neubig 2022), an edit-based corruption and reconstruction approach is proposed to accommodate text data. The texts are perturbed using operations including insert, delete, replace, and keep. During the reconstruction steps, the model first decides which operation to use and then determines the specific token to modify. The empirical results on benchmark tasks including machine translation, summarisation, and style transfer have demonstrated comparable performance to previous methods and autoregressive transformers.

Conclusion

In this report, we conduct an empirical study on the diffusion-based text generation model on the text8 dataset. Autoregressive models including a vanilla Transformer as the baseline and a (pre-trained) GPT-2 are compared. The experimental results show that diffusion generative models achieve poorer performance than autoregressive models, however, when given a large amount of training data, the quality of text generated by diffusion generative models is higher than simple autoregressive models but can't rival the text generated by a pre-trained large scale transformer.

In the future, considering the fabulous results shown by the pre-trained large-scale transformer GPT-2, we want to explore whether the performance or how much the performance will improve if we apply pre-training to or use extra training data on the multinomial diffusion model. Moreover, because the current denoising method is continuous, and texts are naturally represented in discrete, we can study designing a discrete denoising method to the diffusion model to explore if the change of denoising method can improve the performance of diffusion model on the text generation task.

Contribution

Chen Lei (22.5%): Conducted experiments on GPT-2 models together with Kaiyu; Wrote the Introduction chapter, and the Diffusion section of the Methods chapter of the report;

Li Jiayi (25%): Conducted experiments on the transformer; Wrote the Transformer section in the Methods chapter, and the Transformer section of the Experiments chapter of the report;

Li Kaiyu (22.5%): Conducted experiments on GPT-2 models together with Lei; Wrote the GPT-2 section and Analysis section of the Experiments chapter of the report;

Wan Zhang (30%): Conducted experiments on the Multinomial Diffusion model; Diffusion Experiments Section of and Discussion chapter of the report; Recorded the presentation video and made the PPT for it.

References

Austin, J.; Johnson, D. D.; Ho, J.; Tarlow, D.; and van den Berg, R. 2021. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34: 17981–17993.

Croitoru, F.-A.; Hondru, V.; Ionescu, R. T.; and Shah, M. 2023. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Fedus, W.; Goodfellow, I.; and Dai, A. M. 2018. Maskgan: better text generation via filling in the_. *arXiv preprint arXiv:1801.07736*.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33: 6840–6851.

Hoogeboom, E.; Nielsen, D.; Jaini, P.; Forré, P.; and Welling, M. 2021. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in Neural Information Processing Systems*, 34: 12454–12465.

Li, X.; Thickstun, J.; Gulrajani, I.; Liang, P. S.; and Hashimoto, T. B. 2022. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35: 4328–4343.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8): 9.

Reid, M.; Hellendoorn, V. J.; and Neubig, G. 2022. Diffuser: Discrete diffusion via edit-based reconstruction. *arXiv preprint arXiv:2210.16886*.

Savinov, N.; Chung, J.; Binkowski, M.; Elsen, E.; and Oord, A. v. d. 2021. Step-unrolled denoising autoencoders for text generation. *arXiv preprint arXiv:2112.06749*.

Sohl-Dickstein, J.; Weiss, E.; Maheswaranathan, N.; and Ganguli, S. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2256–2265. PMLR.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is All You Need. *arXiv preprint arXiv:1706.03762*.