Generating text with diffusion models

William Dalheim
Computer Science / 2022
williad@stud.ntnu.no

Abstract

Diffusion models are currently the state-of-the-art in computer vision for generating images conditioned on text. Using text embeddings stationed in the continuous domain, one can leverage the flexibility diffusion models provide by performing diffusion and denoising on text. Related work shows that such an application is possible with some modifications to the base framework. This report presents several experiments where two different architectures are used. While the results from these are not convincing, they still demonstrate that the trained model has some understanding of the text corpus. With more time spent on implementation, one can manage to reproduce previous work in this area.

1 Introduction

Diffusion models have shown great performance in image synthesis the past years and even beaten Generative Adversarial Networks (GANs) in conditional image generation like text-to-image tasks (Dhariwal and Nichol, 2021). They are flexible models for synthesizing data with methods like in-painting, image-to-image and interpolating between real images. The motivation behind this project is to experiment with such methods on text instead of images.

In 2017, the field of Natural Language Processing (NLP) was introduced to Transformers, which utilizes the self-attention mechanism (Vaswani et al., 2017). Since then, there have been immense improvements for tasks like language modelling and text classification, where GPT-3 (Brown et al., 2020) is a fitting example. Specifically in language modelling with Transformers, the model is autoregressive because of its decoder. Combining self-attention with diffusion models, one can train a model that generates text in a non-autoregressive way, which also understands the full semantics of the text.

The report begins by reviewing the theory behind this project in section 2, before section 3 where previous work in the field is presented. Following, section 4 and 5 provides details around implementation and the experiments performed. In section 6 the results are analyzed before section 7 finishes up with some closing words. The code and details about hyperparameters are available at github.com/willdalh/text-diffusion.

2 Background

This chapter provides the reader with the necessary knowledge in order to understand how results presented in this report are obtained. It starts with diffusion models and proceeds to explain text embeddings before combining these.

2.1 Diffusion models

Diffusion models are likelihood-based generative models where the latent inputs have the same dimensionality as the final synthesized samples (Ho et al., 2020). Diffusion models perform gradual transformations between a Gaussian distribution $\mathcal{N}(0, I)$ and the data distribution $q(x_0)$. This transformation takes place in two processes called the forward- and reverse process. Both processes are represented as Markov chains and done in an iterative fashion with the number of timesteps T usually being in the thousands. In practice, the forward process corrupts a dataset instance x_0 with Gaussian noise ϵ and the model trains by predicting how much noise was added. When performing inference, a random Gaussian vector x_T is iteratively denoised through the reverse process to obtain an instance x_0 that fits into the original data distribution.

2.1.1 Forward process (diffusion)

The forward process is a fixed Markov Chain where a variance schedule is used to determine how much noise should be added each timestep. This schedule is defined as $\beta_1, ..., \beta_T$, and increases linearly. Ho et al. (2020) proposes $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$. The distribution for a Markov transition can be seen in equation 1.

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{1}$$

To compute x_t from x_{t-1}, one can use a reparameterization of equation 1 where $\epsilon \sim \mathcal{N}(0, I)$ and define $\alpha_t = 1 - \beta_t$. The reparameterization is shown in equation 2.

$$x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon \tag{2}$$

In fact, computing x_t at an arbitrary timestep t can be performed in closed form with only one expression, shown in equation 3. The intuition is that the sum of two standardized Gaussians is still a standardized Gaussian. Utilizing this knowledge brings the forward process down from $\mathcal{O}(t)$ to $\mathcal{O}(1)$. We define the cumulative products of α as $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon \tag{3}$$

The idea is that with a significant number of time steps T, x_T will converge to an isotropic Gaussian. This means that all information is lost and it is impossible to determine which variable is sampled directly between x_T and a $y \sim \mathcal{N}(0, I)$ where $x_T, y \in \mathbb{R}^d$ for some arbitrary dimension d.

2.1.2 Reverse Process (denoising)

The previous section demonstrates that determining x_t from x_{t-1} is computationally feasible. In contrast, determining x_{t-1} from x_t is an intractable analytical process as it will be conditioned on the whole dataset. The reverse transitions are therefore approximated with a Neural Network p_θ as shown in equation 4.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \beta_t I) \tag{4}$$

Through a derivation out of scope for this report by Ho et al. (2020), it is made clear that $\mu_\theta(x_t, t)$ predicts $\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$. As x_t is used as input, a parameterization is chosen to model the noise ϵ instead. To approximate x_{t-1} from x_t we use equation 5 using a parametrized noise ϵ_θ taking the current latent vector x_t and the timestep t as input.

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sqrt{\beta_t}z \tag{5}$$

where the last term, $\sqrt{\beta_t}z$, is some noise correction and $z \sim \mathcal{N}(0, I)$ when $t > 1$, else it is 0. Sampling from the model is done by starting with a $x_T \sim \mathcal{N}(0, I)$, and iteratively denoising it with equation 5 until x_0 is obtained.

2.1.3 Loss function

The intention is to minimize the divergence between the dataset distribution and the distribution that the reverse process produces as t reaches 0. This boils down to minimizing the difference in noise that the network predicts and the actual noise that was added (Ho et al., 2020). The loss function is the mean squared error.

$$\| \underbrace{\epsilon}_{\text{actual noise}} - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon}_{x_t \text{ from equation 3}}, t) \|^2 \tag{6}$$

2.2 Vector Representations of Text

Natural language is discrete data where each word can be seen as a category. Diffusion models as described in section 2.1 are designed for continuous data as small adjustments are made to the features each timestep. Therefore an imminent next step is to convert text into a continuous representation. These representations are called word embeddings, and can generally be separated into word-specific and contextualized embeddings. The first maps each word to a specific vector, while the latter computes a vector conditioning on the words surrounding the target word. In this project, only word-specific embeddings are used.

For a text sequence $w = (w_1, ..., w_N)$, w_i is a one-hot encoding in $\mathbb{R}^{|V|}$ where V is a vocabulary that manages all unique words in the text corpus. For each unique token there exists an embedding $e \in \mathbb{R}^d$. All embeddings are stored inside an embedding matrix $E \in \mathbb{R}^{|V| \times d}$ (Strudel et al., 2022).

2.3 Diffusion models for text generation

We perform the diffusion and denoising in the embedding space of textual data. The forward process starts by computing the initial diffusion variable $x_0 = Ew$. The general framework for diffusion models gives us the loss L_{noise} as defined in equation 6. The next step is to perform a transformation back to the one-hot encodings from the embedding space. This is done using a decoder formulated as a matrix $D \in \mathbb{R}^{|V| \times d}$. Reconstructing one-hot-encodings using the decoder is a task that naturally brings up a cross entropy loss L_{recon} between w and Dx_0. Learning is then comprised of minimizing $L_{\text{total}} = L_{\text{noise}} + L_{\text{recon}}$.

2.3.1 Self-conditioning

Diffusion models as described in section 2.1 works great out of the box for computer vision tasks like generating images. (Li et al., 2022) explains that the intermediate latents during denoising are unstable and not centered around the original embeddings. Instead of directly learning the noise to be removed, they learn an estimate of x_0 which they scale using the constants from the variance schedule in order to compute x_{t-1}. (Strudel et al., 2022) uses a method called self-conditioning to solve the same issue. During sampling, two calls of the diffusion model takes place, and the second call conditions on the output of the first call. This way, the model learns to generate text that closer resembles the text corpus.

3 Related Work

One of the first articles about generating text with an autoregressive model using neural networks was by Sutskever et al. (2011), where Recurrent Neural Networks (RNNs) were used. Fast forward, the preferred method today is Transformers (Vaswani et al., 2017) which solve many problems present in RNNs and LSTMs.

Diffusion models were first proposed by Sohl-Dickstein et al. (2015) as a way of combining Machine Learning and Non-equilibrium Thermodynamics for data generation. The theory and results was further improved by Ho et al. (2020) where they showed that diffusion models could perform similarly to GANs. Only a year later, Dhariwal and Nichol (2021) stated that their diffusion model had beaten GANs completely with superior quality in their samples.

Text is discrete data, but the state-of-the-art diffusion models operate in the continuous domain. However, the first implementation of diffusion models (Sohl-Dickstein et al., 2015) experimented modelling discrete data where binary heartbeat sequences were generated. Modelling text with diffusion is an area much less explored than computer vision. Li et al. (2022) sets some foundations while Strudel et al. (2022) improves upon these.

4 Architecture

When tackling the task of performing diffusion on text, I came up with my own architecture which is almost identical to the one used by Li et al. (2022). The architecture is explained in section 2.3, and will for the remainder of this report be referred to as Text-Model. In this architecture I chose to omit the timestep as input for the diffusion model. My reasoning is that in a previous implementation of diffusion models I have worked on, I found that for relatively simple distributions (like MNIST) the model performed well without any knowledge of the timestep [1]. The diffusion model itself, meaning the architecture after the embedder and before the decoder, is a Transformer Encoder of multiple layers (Vaswani et al., 2017).

A second architecture that was used in this project was one proposed by Strudel et al. (2022). Proceeding, this architecture will be referred to as Strudel-Model. This architecture takes the timestep into account using a sinusoidal encoding typically used internally in Transformer models (Vaswani et al., 2017). The Strudel-Model employs a bottleneck through a linear layer on the embeddings. This means that $d_{\text{model}} < d$, where d_{model} is the dimensionality of embeddings used by the Transformer Encoder, and d is the size of the original embeddings. The output is then linearly projected back to vectors of size d. The Strudel-Model is only used for some of the experiments.

5 Experiments and Results

Prior to starting this project, my idea was to perform experiments after training a diffusion model. These experiments would involve exploring the flexibility that diffusion models pertains in tasks like masking and interpolating between text sequences. Extending this, I wanted to also train a conditional diffusion model where the generation is conditioned on some class label like mood or subject. With such a model, one could experiment with generating text that fit multiple conditions for interesting results.

Unfortunately, I did not manage to train any models that converged towards solving the objective I had in mind. Therefore, my experiments revolve around adjusting hyperparameters and training algorithms to find a model that generates text similar to the text corpus distribution. For all experiments, I present generated text sampled from an epoch where L_{total} had converged.

In most experiments, I use the WikiText-2 [2] dataset. The exception is in section 5.2 where I trained on a dataset of jokes [3]. Strudel et al. (2022) mentions initializing the decoder matrix D as E^T. Some of my experiments do this, while some do not. The reason I have not explored this further, is because it only helped during the first epochs. The reconstruction loss L_{recon} would converge anyways.

5.1 Training embeddings

In my first implementation, all parameters used in the model were being trained, importantly the embedding as well. Interestingly, the trained embeddings became similar across all dimensions for each word with a threshold of about 0.01. A sample is shown in table 1.

Text-Model at epoch 4100	the of of of the the of of of of the the the the of the of of of the of the of the of the of of the of the the

Table 1: Sample from a model that trained its embeddings

5.2 Using frozen Gaussian embeddings

The previous experiment shows that training the embeddings does not achieve convincing results. Strudel et al. (2022) state that the model can converge using random frozen Gaussian embeddings, which is what this experiment attempts. A sample is shown in table 2.

Text-Model at epoch 9320	ours slice taste an minutes paint ladder 246 your down joke task harm t synonym platypus fourth apart townspeople cheep blind innuendo bananas table soviet-era situation welcome* baghdad who creating eastern nose-feratu

Table 2: Sample from a model where frozen random Gaussian embeddings were used

[1]Link to repository
[2]Link to WikiText-2 dataset
[3]Link to jokes-dataset.

5.3 Using pre-trained GloVe embeddings

Strudel et al. (2022) also discusses the use of pre-trained embeddings. In their experiments, they fully train a BERT model before utilizing it for embeddings in the diffusion training. Instead of training them separately myself, I found a set of pre-trained GloVe embeddings on Kaggle [4]. For this experiment I used one providing vectors of length 100 for each word. Given the text corpus I was training on, I only loaded the relevant words, and initialized embeddings for missings words to random Gaussians. Samples are shown in table 3.

Text-Model at epoch 2100	21 october 21 october 19 4 21 19 53 21 4 21 19 october 19 october 19 19 21 19 19 october 37 2009 21 february october 21 october 19 21 21
Strudel-Model at epoch 2100	february 26 26 the 27 , 1926 , the 26 26 2013 26 , 13 , , the 26 the , 26 february , , , 27 26 the the 28 19

Table 3: Samples from models where pre-trained GloVe embeddings were used

5.4 Using self-conditioning with GloVe embeddings

Self-conditioning as mentioned in section 2.3.1 was tried with both the Text- and Strudel model. Samples are shown in table 4.

Text-Model at epoch 3340	1844 21 13 19 19 1852 21 july 36 19 19 19 1921 19 19 19 five 19 19 21 2009 19 1837 1902 19 19 19 19 42 1919 14 eight
Strudel-Model at epoch 3340	not . not . , , not not however not well the . . , , not not not not the , . . . , not , , . , .

Table 4: Samples from models trained with self-conditioning using pre-trained GloVe embeddings

6 Evaluation and Discussion

From the results in section 5, we can see that none of the models managed to learn the intended tasks. Still, some of them reached an optimum as the generated samples seem consistent. An example of consistent output is from section 5.1 where arguably some of the most common English words are produced. In the same experiment, the trained embeddings became the similar across all dimensions for each word. These results indicate that the model does not rely on the coherence of words, but instead learns embeddings to separate them. One can argue that this demonstrates that the model has some understanding of the text corpus, even though it is in a limited way. This is in contrast to using frozen Gaussian embeddings in section 5.2, where the least convincing results emerge. The generated text appears to be sampled randomly.

In three out of four experiments in section 5.3 and 5.4 where GloVe is used, the models seem to produce only tokens that has some connection to dates and time. A possible explanation for this is that all Wikipedia articles usually mentions dates and numbers, meaning it is a group of tokens that appears frequently when training. It is also the case that these tokens have similar embeddings. When the reverse process moves towards the part of the embedding space where many of these are close together, the stochasticity in equation 5 makes the decoder choose randomly between these.

In general, there seems to be no major improvement with self-conditioning, as the differences in results between experiment 5.4 and the others are not significant to state the opposite. This indicates that my implementation had some errors that caused the objective to be defined the wrong way, as Strudel et al. (2022) managed to produce convincing results.

Diffusion models are great at modelling the distribution of a dataset, which does not mean it is great at reconstructing the dataset itself. This is not a problem when generating images, as they will have combinations of features that make them believable. Combining features of different text embeddings

[4]Link to GloVe embeddings on Kaggle

and then having the decoder interpret these becomes a difficult task. The decoder will then be met with embeddings not trained on. I believe this is where my implementation failed, as many of the samples seem to be random words. Strudel et al. (2022) designed their architecture to predict μ instead of ϵ. Ho et al. (2020) states that these are identical, only different parameterizations. I had some difficulties when translating the self-conditioning algorithm to the ϵ-case, and I am certain the error lies here.

7 Conclusion and Future Work

For this project, I had many interesting ideas on how to creatively explore diffusion models for text generation. To fulfill these, a working diffusion model is unquestionably needed. This area is unexplored, and it was therefore difficult to find good resources and clues for how to improve my implementation. With more time, I would try more algorithms and spend more resources on getting the models to function as intended. With a working diffusion model however, I believe there are many interesting tasks one could explore like generating variations of sentences and interpolating between text with different meanings.

(Li et al., 2022) and (Strudel et al., 2022) both show that generating text with diffusion is a task that can be solved. My results does not show the opposite, but instead show that the models are capable of generating text with some form of coherence. The performance of diffusion models in computer vision sets an inspiring path for text modelling as well. I believe we will see more attempts in this space with further improvements and new creative ways of generating text.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021. URL https://arxiv.org/abs/2105.05233.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020. URL https://arxiv.org/abs/2006.11239.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation, 2022. URL https://arxiv.org/abs/2205.14217.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015. URL https://arxiv.org/abs/1503.03585.

Robin Strudel, Corentin Tallec, Florent Altché, Yilun Du, Yaroslav Ganin, Arthur Mensch, Will Grathwohl, Nikolay Savinov, Sander Dieleman, Laurent Sifre, and Rémi Leblond. Self-conditioned embedding diffusion for text generation, 2022. URL https://arxiv.org/abs/2211.04236.

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1017–1024, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.