Using Diffusion Models to Generate Counterfactual Objects

CS 726 Project - TEAM WHITE
Guide : SUNITA SARAWAGI

Shiv Kiran Bagathi - 200050019
Sasank - 200050044
Saikiran Botla - 200050023
Tanuja - 200050029

Introduction:
Diffusion Models are generative models, meaning they are used to generate data similar to the data on which they are trained. Fundamentally, Diffusion Models work by destroying training data through the successive addition of Gaussian noise and then learning to recover the data by reversing this noising process. After training, we can use the Diffusion Model to generate data by passing randomly sampled noise through the learned denoising process.

The diffusion Model consists of a forward process (or diffusion process), in which a datum (generally an image) is progressively noised, and a reverse process (or reverse diffusion process), in which noise is transformed back into a sample from the target distribution.
The equation looks like

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

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

Training. With sufficient data and model capacity, the following training procedure ensures that the optimal solution to ∇x log pt(x) can be found by training θ to approximate ∇x log pt(xt | x0). The training procedure can be formalized as

$$\theta^* = \arg\min_\theta \mathbb{E}_{t,\mathbf{x}_0\sim p_{\text{data}},\epsilon\sim\mathcal{N}(0,\mathrm{I})}\left[(1-\alpha_t)\left\|\boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon, t) - \epsilon\right\|_2^2\right].$$

Inference. Once the model θ is learned using the above equation, generating samples consists in starting from xT ~ N (0,I) and iteratively sampling from the reverse Markov chain following:

$$\mathbf{x}(t-1) = \frac{1}{\sqrt{1-\beta_t}}\left[\mathbf{x}_t + \beta_t\, \boldsymbol{\epsilon}_{\theta^*}(\mathbf{x}_t, t)\right] + \sqrt{\beta_t}\mathbf{z}, \quad t = T\cdots 0\ ,\ \mathbf{z}\sim\mathcal{N}(\mathbf{0},\mathrm{I}).$$

Diff-SCM: Unifying Diffusion Processes and Causal Models:

SCMs have been associated with ordinary and stochastic differential equations and other types of dynamical systems. In these cases, differential equations are useful for modeling time-dependent problems such as chemical kinetics or mass-spring systems. From the energy-based models perspective. and denoising score models into a framework based on SDEs. SDEs are used for formalizing a diffusion process in a continuous manner where a model is learned to reverse the SDE to generate images.
we unify the SDE framework with causal models.

The diffusion forces the exogenous noise u (j) corresponding to a variable x (j) of interest to be independent of other u (i) , ∀ i 6= j, following the constraints from Eq. 4. The Brownian motion (diffusion) leads to a Gaussian distribution, which can be seen as a prior. Analogously, the

original joint distribution entailed by the SCM pG(X) diffuses to independent Gaussian distributions equivalent to p(U). As such, the time-dependent joint distribution p(Xt), $\forall$t $\in$ [0, T] have as bounds p(XT) = p(U) and p(x0) = pG(X). Note that p(Xt) refers to time-dependent distribution over all causal variables x (k) .

$$d\mathbf{x}^{(k)} = -\frac{1}{2}\beta_t \mathbf{x}^{(k)} dt + \sqrt{\beta_t}\, d\mathbf{w}, \quad \forall k \in [1, K],$$
$$\text{where } p(\mathbf{x}_0^{(k)}) = \prod_{j=k}^{K} p(\mathbf{x}^{(j)} \mid \mathbf{pa}^{(j)}) \text{ and } p(\mathbf{x}_T^{(k)}) = p(\mathbf{u}^{(k)}).$$

The generative process is the solution of the reverse-time SDE from Eq. 6 in time. This process is done by iteratively updating the exogenous noise x (k) T = u (k) with the gradient of the data distribution w.r.t. the input variable ∇x (k) t log p(x (k) t), until it becomes x (k) 0 = x (k) with:

$$d\mathbf{x}^{(k)} = \left[-\frac{1}{2}\beta_t + \beta_t \nabla_{\mathbf{x}_t^{(k)}} \log p(\mathbf{x}_t^{(k)}) \right] dt + \sqrt{\beta_t}\bar{\mathbf{w}}.$$

Interventions as anti-causal gradient updates:
We consider the SCM G and a variable x (j) $\in$ an(k) . The effect observed on x (k) caused by an intervention on x (j) , pG(x (k) | do(x (j) = x (j))), is equivalent to solving a reverse-diffusion process for x (k) t . Since the sampling process involves taking into account the distribution entailed by G, it is guided by the gradient of an anti-causal predictor w.r.t. the effect when the cause is assigned a specific value:

$$\nabla_{x_t^{(k)}} p_{\mathfrak{G}-}(\mathbf{x}^{(j)} = x^{(j)} \mid x_t^{(k)}).$$

Counterfactual Estimation with Diff-SCM:
The first step for estimating a counterfactual is the abduction of exogenous noise. the value of a causal variable depends on its parents and respective exogenous noise. From a deep learning perspective, one might consider the exogenous u (k) an inferred latent variable. The prior p(u (k)) of u (k) in Diff-SCM is a Gaussian.

Prediction under Intervention. Once the abduction of exogenous noise u (k) is done for a given factual observation x (k) F , counterfactual estimation consists in applying an intervention in the reverse diffusion process with the gradients of an anti-causal predictor.
Controlling the Intervention. There are three main factors contributing for the counterfactual estimation in Alg. 1: (i) The inferred u (k) keeps the information about the factual observation; (ii)

∇x (k) t log pφ(x (j) CF | x (k) t) guide the intervention towards the desired counterfactual class; and (iii) θ(x (k) t , t) forces the estimation to belong to the data distribution.

Algorithm 1 Inference of **counterfactual** for a variable $\mathbf{x}^{(k)}$ from an intervention on $\mathbf{x}^{(j)} \in \mathbf{an}^{(k)}$

Models: trained diffusion model $\boldsymbol{\epsilon}_\theta$ and anti-causal predictor $p_\phi(x^{(j)} \mid x_t^{(k)})$

Input : factual variable $x_{0,\mathrm{F}}^{(k)}$, target intervention $x_{0,\mathrm{CF}}^{(j)}$, scale s

Output: counterfactual $x_{0,\mathrm{CF}}^{(k)}$

Abduction of Exogenous Noise – Recovering $u^{(k)}$ from $x_{0,\mathbf{F}}^{(k)}$

for $t \leftarrow 0$ **to** T **do**

$$x_{t+1,\mathrm{F}}^{(k)} \leftarrow \sqrt{\alpha_{t+1}} \left(\frac{x_{t,\mathrm{F}}^{(k)} - \sqrt{1-\alpha_t}\, \boldsymbol{\epsilon}_\theta(x_{t,\mathrm{F}}^{(k)}, t)}{\sqrt{\alpha_t}} \right) + \sqrt{\alpha_{t+1}}\, \boldsymbol{\epsilon}_\theta(x_{t,\mathrm{F}}^{(k)}, t)$$

end

$$u^{(k)} = x_{T,\mathrm{F}}^{(k)} = x_T^{(k)}$$

Generation under Intervention

for $t \leftarrow T$ **to** 0 **do**

$$\epsilon \leftarrow \boldsymbol{\epsilon}_\theta(x_t^{(k)}, t) - s\sqrt{1-\alpha_t}\, \nabla_{x_t^{(k)}} \log p_\phi(x_{0,\mathrm{CF}}^{(j)} \mid x_t^{(k)})$$

$$x_{t-1}^{(k)} \leftarrow \sqrt{\alpha_{t-1}} \left(\frac{x_t^{(k)} - \sqrt{1-\alpha_t}\, \epsilon}{\sqrt{\alpha_t}} \right) + \sqrt{\alpha_{t-1}}\, \epsilon$$

end

$$x_{0,\mathrm{CF}}^{(k)} = x_0^{(k)}$$

Implementation and Code

/training/
main_diffusion_train.py : Here we train the main diffusion process loop based on the score diffusion model.
anticausal_classifier_train.py : Here we train the Anti Causal Predictor function.
Both the Neural Network in score based diffusion model and classifier used for Anti Causal Predictor function are based on Unet model.

/configs/
mnist_configs.py : Contain all the parameters and hyperparameters defined in the model. Also contains the causal and endogenous nodes defined.

/models/
Defines architecture of each step in the model

/sampling/
Has the ddim event loop of forward and reverse process.

Important Steps in Architecture

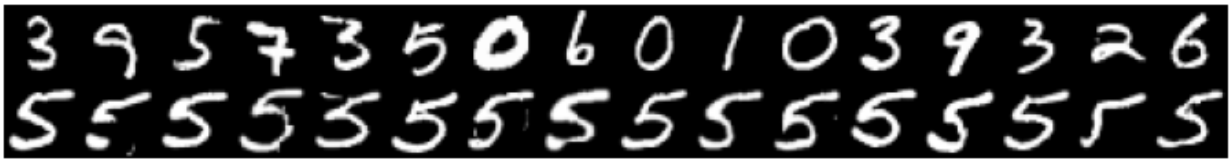
```
def get_models_from_config(config):
    diffusion = create_gaussian_diffusion(config)
    model = create_score_model(config)
    model.load_state_dict(
        dist_util.load_state_dict(config.sampling.model_path, map_location=dist_util.dev())
    )
    model.to(dist_util.dev())
    if config.score_model.use_fp16:
        model.convert_to_fp16()
    model.eval()

    if config.sampling.classifier_scale != 0:
        classifier = create_anti_causal_predictor(config)
        classifier.load_state_dict(
            dist_util.load_state_dict(config.sampling.classifier_path, map_location=dist_util.dev())
        )
        classifier.to(dist_util.dev())
        if config.classifier.classifier_use_fp16:
            classifier.convert_to_fp16()
        classifier.eval()
    else:
        classifier = None
    return classifier, diffusion, model
```

– wrapper function to generate score_model, anti causal predictor and forward diffusion step.

```
def get_models_functions(config, model, anti_causal_predictor):
  ## return cond_fn, model_fn,
```

Counterfactual generated on change in causal variable



Top is input dataset and bottom is counterfactuals generated.

References :

This model is heavily based on Diffusion Causal Models for Counterfactual Estimation by Pedro Sanchez, Sotirios A. Tsaftaris https://arxiv.org/pdf/2202.10166.pdf

Denoising Diffusion Probabilistic Models Jonathan Ho, Ajay Jain, Pieter Abbeel https://arxiv.org/pdf/2006.11239

A great intuition on Diffusion Models by Computerphile
https://www.youtube.com/watch?v=1CIpzeNxIhU

Thank You.