Application Log

Hanzhang Yin

October 3, 2024

1 Application Log 1

[1] **chung2024diffusionposteriorsamplinggeneral**.

Background Summary

Note that Diffusion models have recently emerged as effective generative solvers for inverse problems, used in high-quality image reconstruction (i.e. image super-resolution tasks). This work extends diffusion models to solve general noisy (non)linear inverse problems by approximating posterior sampling. This results in a more effective generative path integrating noise statistics like Gaussian and Poisson. The approach efficiently handles complex issues, such as Fourier phase retrieval and non-uniform deblurring, surpassing previous methods in noisy settings.

Mathematical Content

MACHINE LEARNING MODEL (SCORE-BASED DIFFUSION MODELS): Diffusion models define the generative process as the reverse of the noising process. It Contains the forward and reverse (backward) diffusion process (NOTE: diffusion model aims to solve an ill-posed Inverse Problems):

- In the forward diffusion process, a data glitch is gradually perturbed by adding Gaussian noise over a sequence of time slices. This process transforms the original data distribution to a Gaussian-like distribution.

  $$x_t = \sqrt{\bar{a}(t)}x_0 + \sqrt{1-\bar{a}(t)}z, z \sim N(0, I)$$

  (NOTE: x_0 is the original data, x_t is the perturbed data at time t, and z is the Gaussian noise. The term $\bar{a}(t)$ controls the amount of noise added at each time slice.)

- In the reverse diffusion process, the goal of the scored-based diffusion model is to learn how to reverse the forward diffusion process to generate new data samples from noise. This reverse diffusion process is an SDE or a discrete-time Markov chain. The model learns the score function at each time step:

  $$\nabla_{x_t} \log p(x_t) \approx s^*_\theta(x_t, t)$$

APPROXIMATION OF THE LIKELIHOOD: The approximation technique here utilizes "Bayesian Net‘' and "Monte-Carlo" random process. Noting that approximations are made to simplify calculations:

$$p(y|x_t) \approx p(y|\hat{x}_0), \quad \text{where} \quad \hat{x}_0 := E[x_0|x_t]$$

The expectation $\hat{x}_0$ is the proxy for most likely original data given the noisy observation.
In further mathematical proof, the diffusion model applied "Jensen's Inequality" and "Jensen gap" to quantify the approximation value for validity and higher accuracy.
OPTIMIZATION ALGORITHM: For score-based model training, the article is aimed at minimizing a score-matching loss function that encourages the NNs to learn the true score function $\nabla_{x_t} \log p(x_t)$ of the data distribution at each time step.

$$\theta^* = \arg\min_{\theta} E_{t,x(t),x(0)} \left[\| s_\theta(x(t), t) - \nabla_{x_t} \log p(x(t)|x(0)) \|_2^2 \right]$$

The article aims to minimize the expectation of differences based on the random variables on the set parameters θ: t (time from a certain distribution), $x(t)$ (The noisy data sample at time t, generated by the forward diffusion process), $x(0)$ (The original clean data sample at time before any noise is added).
The differences is calculated between $s_\theta(x(t), t)$, the neural network parameterized by θ to approximate the score function (i.e. "true distribution"); and $\nabla_{x_t} \log p(x(t)|x(0))$, the score function aims to approximate)

Suggested In-Course Beneficial Mathematical Content

Partial Derivatives, Differential Equation, $\cdots$: Might be beneficial for diffusion model and optimization algorithm understanding.
Matrix Operations, Eigenvalues, and Eigenvectors, Taylor Expansions, gra: Might be useful to understand the TensorFlow after linear transformation of the data and analyzing the convergence path for the iterative model behind the suggested score-based diffusion model.
Gaussian and Poisson Distribution, Posterior Distribution: Might be crucial to understanding such probabilistic reasoning and proofs behind the diffusion model.
Monte Carlo Methods: Might be important to understand the sampling configurations on the dataset (e.g. ImageNet). In some respect, they can be combined with "Molecular Dynamics" simulations to enhance sampling efficiency.

2 Application Log 2

[2] zhang2023addingconditionalcontroltexttoimage.

Background Summary

ControlNet introduces a novel neural network architecture that integrates spatial conditioning controls into existing large-scale text-to-image diffusion models. Utilizing a robust backbone built on pretrained encoding layers and innovative "zero convolutions," ControlNet enhances model adaptability to various conditioning controls like edges and depth, tested across datasets of varying sizes. The results demonstrate ControlNet's potential to broaden the application scope of image diffusion models through precise control and fine-tuning capabilities.

Mathematical Content

The mathematical formulation of ControlNet modifies the Stable Diffusion (SD) model to incorporate additional conditions for controlling the image generation process. The key concepts and steps are outlined as follows:

Stable Diffusion Model without ControlNet

The original image generation process using the Stable Diffusion model can be expressed as:

$$\hat{x} = f_\theta(x, t, c)$$

where: $\hat{x}$ is the generated image; f_θ is the Stable Diffusion model with parameters θ;

x is the input image; t represents the time step; c is the conditional information (e.g., text prompt).

Stable Diffusion Model with ControlNet

After integrating ControlNet, the image generation process becomes:

$$\hat{x} = f_\theta(x, t, c, c') + \mathcal{T}_\phi(c'')$$

where: c' is the additional control condition (e.g., edge map, depth map), $\mathcal{T}_\phi$ represents the zero convolution modules with parameters ϕ; c'' is the transformed control condition in the latent space.
The error approximation we've been taught in class might help quantify the sensitivity of the model and hence help us to do such accurate model evaluations.

Zero Convolution Modules Initialization

Zero convolution modules are initialized with:

$$w_{\text{zero}} = 0, \quad b_{\text{zero}} = 0$$

where: w_{zero} and b_{zero} are the weights and biases of the zero convolution layers, both set to zero initially.
The convolution operation for a general convolutional layer can be represented as:

$$y_{i,j,k} = \sum_{m=1}^{M} \sum_{n=1}^{N} \sum_{c=1}^{C} x_{i+m,j+n,c} \cdot w_{m,n,c,k} + b_k$$

where: $y_{i,j,k}$ is the output feature map at spatial location (i, j) and channel k; $x_{i+m,j+n,c}$ is the input feature map at spatial location $(i+m, j+n)$ and channel c; $w_{m,n,c,k}$ are the weights of the convolutional filter of size $M \times N$; where c is the input channel and k is the output channel; b_k is the bias term for the output channel k, $M \times N$ is the size of the convolutional kernel (filter); C is the number of input channels.

Gradient Descent and Weight Update

During training, the gradient descent updates the weights, ensuring they become non-zero:

$$\text{If } w_{\text{zero}} = 0, \text{ and } \frac{\partial \mathcal{L}}{\partial w_{\text{zero}}} \neq 0, \text{ then after one step, } w_{\text{zero}} \neq 0$$

where:

- $\mathcal{L}$ is the loss function being minimized.
- $\frac{\partial \mathcal{L}}{\partial w_{\text{zero}}}$ is the gradient of the loss with respect to the zero convolution weights.

Joint Modeling of ControlNet

ControlNet jointly models the image and pixel-level conditions by modifying the loss function:

$$\mathcal{L}_\theta(x, t, c, c') = E_{x \sim p(x)} \left[\| f_\theta(x, t, c, c') + \mathcal{T}_\phi(c'') - x \|^2 \right]$$

where:

- E denotes the expected value over the data distribution $p(x)$.
- The loss function $\mathcal{L}_\theta$ aims to minimize the difference between the generated image and the target image.

Here, Taylor expansion might help approximate how small the weight changes affect the loss. As an example, we can apply the Hessian matrix of second derivatives to retrieve the following formula:

$$\mathcal{L}(\theta + \delta\theta) \approx \mathcal{L}(\theta) + \nabla_\theta \mathcal{L} \cdot \delta\theta + \frac{1}{2}\delta\theta^\top H \delta\theta$$

3 Application Log 3

[3] tian2024visualautoregressivemodelingscalable.

Background Summary

Visual AutoRegressive modeling (VAR) introduces a new paradigm for image generation by redefining autoregressive learning as a "coarse-to-fine" strategy, termed "next-scale prediction" or "next-resolution prediction," which departs from the traditional raster-scan "next-token prediction." This approach enables AR transformers to efficiently learn visual distributions and generalize effectively, allowing GPT-style AR models to surpass diffusion transformers for the first time. Empirical results demonstrate that VAR outperforms Diffusion Transformers (DiT) across multiple dimensions, including image quality, inference speed, data efficiency, and scalability. Additionally, VAR shows strong zero-shot generalization capabilities in tasks like image in-painting, out-painting, and editing, reflecting its ability to capture two essential properties of LLMs.

Mathematical Content

Next-Token Prediction (Traditional Approach): In traditional autoregressive modeling, the probability of generating a sequence of tokens $x = (x_1, x_2, \ldots, x_T)$ is factored based on the probability of each token given its prefix:

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, x_2, \ldots, x_{t-1}).$$

Here, x_t represents a token from a vocabulary of size V.
Tokenization: To apply autoregressive modeling to images, the images are tokenized into discrete tokens using a quantized autoencoder:
In the VAR model, tokenization involves converting continuous image feature maps into discrete tokens, which can be optimized using linear algebra techniques like LU-factorization and Gaussian elimination. LU-factorization efficiently handles matrix operations by decomposing them into simpler triangular matrices, speeding up computations needed for transforming features into tokens. Gaussian elimination helps maintain numerical stability during these transformations, preventing instabilities that could degrade performance.
- **Quantized Autoencoder:**
The image feature map $f \in R^{h \times w \times c}$ is converted into discrete tokens $q \in [V]^{h \times w}$:

$$f = \mathcal{E}(\text{im}), \quad q = Q(f),$$

where $\mathcal{E}(\cdot)$ is an encoder, and $Q(\cdot)$ is a quantizer. The quantization process $q = Q(f)$ involves mapping each feature vector $f^{(i,j)}$ to the closest index $q^{(i,j)}$ in the codebook Z:

$$q^{(i,j)} = \arg\min_{v \in [V]} \|\text{lookup}(Z, v) - f^{(i,j)}\|_2.$$

The reconstructed image $\hat{\text{im}}$ from quantized tokens is:

$$\hat{f} = \text{lookup}(Z, q), \quad \hat{\text{im}} = D(\hat{f}),$$

where $D(\cdot)$ is a decoder.

Loss Function:

The compound loss function combines perceptual loss $\mathcal{L}_P$, adversarial loss $\mathcal{L}_G$, and codebook loss:

$$\mathcal{L} = \|\text{im} - \hat{\text{im}}\|_2^2 + \lambda_P \mathcal{L}_P(\hat{\text{im}}) + \lambda_G \mathcal{L}_G(\hat{\text{im}}).$$

*Similarly, as in application log 2, Taylor expansion might help to approximate how small the changes in the weights affect the loss; we can apply the Hessian matrix of second derivatives to retrieve the following formula:

$$\mathcal{L}(\theta) \approx \mathcal{L}(\theta_0) + \nabla_\theta \mathcal{L}(\theta_0) \cdot (\theta - \theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta_0)(\theta - \theta_0),$$

The error $R_n(\theta)$ in the Taylor approximation after truncating at the second-order term can be bounded as follows:

$$|R_2(\theta)| \leq \frac{M}{6}\|\theta - \theta_0\|^3,$$

where M is an upper bound on the norm of the third derivative of $\mathcal{L}(\theta)$ within the neighborhood of θ_0, and $\|\theta - \theta_0\|$ is the distance between the current and initial parameters.

Next-Scale Prediction (VAR Approach):

VAR shifts from "next-token prediction" to "next-scale prediction." Here, an image is represented by a sequence of multi-scale token maps $R = (r_1, r_2, \ldots, r_K)$, where each token map r_k corresponds to a resolution scale.

The autoregressive likelihood for next-scale prediction is:

$$p(r_1, r_2, \ldots, r_K) = \prod_{k=1}^{K} p(r_k \mid r_1, r_2, \ldots, r_{k-1}).$$

At each scale k, the token map r_k is generated conditioned on its prefix $r_{<k}$, maintaining spatial consistency across scales.

*Linear algebra and error analysis are crucial to optimizing performance in here. LU-factorization and Gaussian elimination enable efficient and stable computations of transformations and interpolations across scales, reducing the computational cost during iterative predictions in the following Encoder and Muti-scale image reconstruction.

Multi-Scale Quantization:

VAR employs a multi-scale quantization strategy where images are encoded into multiple discrete token maps at varying resolutions using a shared codebook Z. Each token map r_k is generated by:

$$r_k = Q(f_k), \quad f_k = \mathcal{E}_k(\text{im}),$$

where $\mathcal{E}_k$ is an encoder for scale k.

Efficiency and Parallelization: VAR allows for parallel token generation at each scale, reducing computational complexity. The model uses block-wise causal attention to ensure each token's dependency is confined to its respective prefix.

Algorithms

Algorithm 1 Multi-scale VQVAE Encoding

Inputs: raw image im
Hyperparameters: steps K, resolutions $(h_k, w_k)_{k=1}^{K}$
$f = \mathcal{E}(\text{im}), R = [\,]$
for $k = 1, \ldots, K$ **do**
    $r_k = Q(\text{interpolate}(f, h_k, w_k))$
    $R = \text{queue\_push}(R, r_k)$
    $z_k = \text{lookup}(Z, r_k)$
    $z_k = \text{interpolate}(z_k, h_k, w_k)$
    $f = f - \phi_k(z_k)$
end for
Return: multi-scale tokens R

Algorithm 2 Multi-scale VQVAE Reconstruction

Inputs: multi-scale token maps R
Hyperparameters: steps K, resolutions $(h_k, w_k)_{k=1}^{K}$
$f = 0$
for $k = 1, \ldots, K$ **do**
    $r_k = \text{queue\_pop}(R)$
    $z_k = \text{lookup}(Z, r_k)$
    $z_k = \text{interpolate}(z_k, h_k, w_k)$
    $\hat{f} = f + \phi_k(z_k)$
    $\hat{\text{im}} = D(\hat{f})$
end for
Return: reconstructed image $\hat{\text{im}}$

- Algorithm 1: Multi-scale VQVAE Encoding describes using a quantization autoencoder encoding an image into multiple scales. It takes a raw image as input and iteratively encodes it at different resolutions to produce multi-scale tokens.
- Algorithm 2: Multi-scale VQVAE Reconstruction reconstructs the image from the multi-scale tokens generated by the encoding algorithm. It utilizes a shared codebook and progressively reconstructs the image at increasing resolutions.

4 Application Log 4

[4] **hu2021loralowrankadaptationlarge**.

Background Summary

The article "LoRA: Low-Rank Adaptation of Large Language Models" introduces a method for efficiently fine-tuning large pre-trained models by freezing the original weights and training only low-rank decomposition matrices inserted into each layer. This reduces the number of trainable parameters by up to 10,000 times and memory requirements by 3 times, without sacrificing performance. LoRA achieves comparable or better results than traditional fine-tuning on several models (RoBERTa, DeBERTa, GPT) and introduces no additional inference latency.

Mathematical Content

LoRA (Low-Rank Adaptation) uses low-rank matrices to efficiently represent weight updates during model adaptation. Instead of updating the full weight matrix, LoRA approximates the updates using a low-rank decomposition. Below is a detailed description of the method:

1. **Original Weight Matrix**:

Given a pre-trained weight matrix $W_0 \in R^{d\times k}$, where d is the input dimension and k is the output dimension of the layer, a typical adaptation would involve learning an update matrix ΔW of the same size, i.e., $\Delta W \in R^{d\times k}$.

2. **Low-Rank Decomposition**:

Instead of learning ΔW directly, LoRA approximates it using a low-rank decomposition:

$$\Delta W = BA,$$

where:

- $B \in R^{d\times r}$,
- $A \in R^{r\times k}$,
- $r \ll \min(d, k)$.

Here, r is a hyperparameter that controls the rank of the decomposition and thus the number of trainable parameters. The effective number of trainable parameters is reduced from $d \times k$ to $r \times (d + k)$.

3. **Modified Forward Pass**:

During the forward pass, for an input $x \in R^d$, the output is computed as:

$$h = W_0x + \Delta Wx = W_0x + B(Ax).$$

This formulation keeps the original weights W_0 frozen and only optimizes the smaller matrices A and B.

4. **Scaling Factor**:

To control the impact of ΔW, a scaling factor α is introduced:

$$h = W_0 x + \frac{\alpha}{r} B(Ax).$$

This scaling ensures that the initialization of ΔW starts with a small magnitude, stabilizing the training.

5. **Parameter Efficiency**:

The number of trainable parameters in LoRA is:

$$r \times (d + k) \quad \text{versus} \quad d \times k.$$

For large models, where d and k are large, choosing a small r significantly reduces the computational cost and memory footprint. For example, setting $r = 4$ for a layer where $d = 12288$ and $k = 4096$ results in a reduction from 50 million parameters to around 66,000 trainable parameters.