Title: Latent diffusion model

URL: https://en.wikipedia.org/wiki/Latent_diffusion_model

PageID: 77815635

Categories: Category:2021 software, Category:Artificial intelligence art, Category:Deep learning, Category:Generative artificial intelligence, Category:Image processing, Category:Text-to-image generation, Category:Unsupervised learning

-----

The Latent Diffusion Model (LDM) [1] is a diffusion model architecture developed by the CompVis (Computer Vision & Learning) [2] group at LMU Munich . [3]

Introduced in 2015, diffusion models (DMs) are trained with the objective of removing successive applications of noise (commonly Gaussian) on training images. The LDM is an improvement on standard DM by performing diffusion modeling in a latent space , and by allowing self-attention and cross-attention conditioning.

LDMs are widely used in practical diffusion models. For instance, Stable Diffusion versions 1.1 to 2.1 were based on the LDM architecture. [4]

Version history

Diffusion models were introduced in 2015 as a method to learn a model that can sample from a highly complex probability distribution. They used techniques from non-equilibrium thermodynamics , especially diffusion . [5] It was accompanied by a software implementation in Theano . [6]

A 2019 paper proposed the noise conditional score network (NCSN) or score-matching with Langevin dynamics (SMLD). [7] The paper was accompanied by a software package written in PyTorch release on GitHub. [8]

A 2020 paper [9] proposed the Denoising Diffusion Probabilistic Model (DDPM) , which improves upon the previous method by variational inference . The paper was accompanied by a software package written in TensorFlow release on GitHub. [10] It was reimplemented in PyTorch by lucidrains. [11] [12]

On December 20, 2021, the LDM paper was published on arXiv, [13] and both Stable Diffusion [14] and LDM [15] repositories were published on GitHub. However, they remained roughly the same. Substantial information concerning Stable Diffusion v1 was only added to GitHub on August 10, 2022. [16]

All of Stable Diffusion (SD) versions 1.1 to XL were particular instantiations of the LDM architecture.

SD 1.1 to 1.4 were released by CompVis in August 2022. There is no "version 1.0". SD 1.1 was a LDM trained on the laion2B-en dataset. SD 1.1 was finetuned to 1.2 on more aesthetic images. SD 1.2 was finetuned to 1.3, 1.4 and 1.5, with 10% of text-conditioning dropped, to improve classifier-free guidance. [17] [18] SD 1.5 was released by RunwayML in October 2022. [18]

Architecture

While the LDM can work for generating arbitrary data conditional on arbitrary data, for concreteness, we describe its operation in conditional text-to-image generation.

LDM consists of a variational autoencoder (VAE), a modified U-Net , and a text encoder.

The VAE encoder compresses the image from pixel space to a smaller dimensional latent space , capturing a more fundamental semantic meaning of the image. Gaussian noise is iteratively applied to the compressed latent representation during forward diffusion. The U-Net block, composed of a ResNet backbone, denoises the output from forward diffusion backwards to obtain a latent representation. Finally, the VAE decoder generates the final image by converting the representation back into pixel space. [4]

The denoising step can be conditioned on a string of text, an image, or another modality. The encoded conditioning data is exposed to denoising U-Nets via a cross-attention mechanism . [4] For conditioning on text, the fixed, a pretrained CLIP ViT-L/14 text encoder is used to transform text prompts to an embedding space. [3]

Variational Autoencoder

To compress the image data, a variational autoencoder (VAE) is first trained on a dataset of images. The encoder part of the VAE takes an image as input and outputs a lower-dimensional latent representation of the image. This latent representation is then used as input to the U-Net. Once the model is trained, the encoder is used to encode images into latent representations, and the decoder is used to decode latent representations back into images.

Let the encoder and the decoder of the VAE be E , D {\displaystyle E,D} .

To encode an RGB image, its three channels are divided by the maximum value, resulting in a tensor x {\displaystyle x} of shape (3 , 512 , 512) {\displaystyle (3,512,512)} with all entries within range [0 , 1] {\displaystyle [0,1]} . The encoded vector is 0.18215 × E (2 x – 1) {\displaystyle 0.18215\times E(2x-1)} , with shape (4 , 64 , 64) {\displaystyle (4,64,64)} , where 0.18215 is a hyperparameter, which the original authors picked to roughly whiten the encoded vector to roughly unit variance. Conversely, given a latent tensor y {\displaystyle y} , the decoded image is (D (y / 0.18125) + 1) / 2 {\displaystyle (D(y/0.18125)+1)/2} , then clipped to the range [0 , 1] {\displaystyle [0,1]} . [19] [20]

In the implemented version, [3] : ldm/models/autoencoder.py the encoder is a convolutional neural network (CNN) with a single self-attention mechanism near the end. It takes a tensor of shape (3 , H , W) {\displaystyle (3,H,W)} and outputs a tensor of shape (8 , H / 8 , W / 8) {\displaystyle (8,H/8,W/8)} , being the concatenation of the predicted mean and variance of the latent vector, each of shape (4 , H / 8 , W / 8) {\displaystyle (4,H/8,W/8)} . The variance is used in training, but after training, usually only the mean is taken, with the variance discarded.

The decoder is also a CNN with a single self-attention mechanism near the end. It takes a tensor of shape (4 , H / 8 , W / 8) {\displaystyle (4,H/8,W/8)} and outputs a tensor of shape (3 , H , W) {\displaystyle (3,H,W)} .

U-Net

The U-Net backbone takes the following kinds of inputs:

A latent image array , produced by the VAE encoder. It has dimensions (channel , width , height) {\displaystyle ({\text{channel}},{\text{width}},{\text{height}})} . Typically, (channel , width , height) = (4 , 64 , 64) {\displaystyle ({\text{channel}},{\text{width}},{\text{height}})=(4,64,64)} .

A timestep-embedding vector , which tells the backbone how much noise there is in the image. For example, an embedding of timestep t = 0 {\displaystyle t=0} would indicate that the input image is already noiseless, while t = 100 {\displaystyle t=100} would mean there is much noise.

A modality-embedding vector sequence , which indicates to the backbone about additional conditions for denoising. For example, in text-to-image generation, the text is divided into a sequence of tokens, then encoded by a text encoder, such as a CLIP encoder , before feeding into the backbone. As another example, an input image can be processed by a Vision Transformer into a sequence of vectors, which can then be used to condition the backbone for tasks such as generating an image in the same style.

Each run through the U-Net backbone produces a predicted noise vector. This noise vector is scaled down and subtracted away from the latent image array, resulting in a slightly less noisy latent image. The denoising is repeated according to a denoising schedule ("noise schedule"), and the output of the last step is processed by the VAE decoder into a finished image.

Similar to the standard U-Net , the U-Net backbone used in the SD 1.5 is essentially composed of down-scaling layers followed by up-scaling layers. However, the U-Net backbone has additional modules to allow for it to handle the embedding. As an illustration, we describe a single down-scaling layer in the backbone:

The latent array and the time-embedding are processed by a ResBlock : The latent array is processed by a convolutional layer . The time-embedding vector is processed by a one-layered feedforward network , then added to the previous array (broadcast over all pixels). This is processed by another convolutional layer, then another time-embedding.

The latent array is processed by a convolutional layer .

The time-embedding vector is processed by a one-layered feedforward network , then added to the previous array (broadcast over all pixels).

This is processed by another convolutional layer, then another time-embedding.

The latent array and the embedding vector sequence are processed by a SpatialTransformer , which is essentially a standard pre-LN Transformer decoder without causal masking. In the cross-attentional blocks, the latent array itself serves as the query sequence, one query-vector per pixel. For example, if, at this layer in the U-Net, the latent array has dimensions (128 , 32 , 32) {\displaystyle (128,32,32)} , then the query sequence has 1024 {\displaystyle 1024} vectors, each of which has 128 {\displaystyle 128} dimensions. The embedding vector sequence serves as both the key sequence and as the value sequence. When no embedding vector sequence is input, a cross-attentional block defaults to self-attention, with the latent array serving as the query, key, and value. [21] : line 251

In the cross-attentional blocks, the latent array itself serves as the query sequence, one query-vector per pixel. For example, if, at this layer in the U-Net, the latent array has dimensions (128 , 32 , 32) {\displaystyle (128,32,32)} , then the query sequence has 1024 {\displaystyle 1024} vectors, each of which has 128 {\displaystyle 128} dimensions. The embedding vector sequence serves as both the key sequence and as the value sequence.

When no embedding vector sequence is input, a cross-attentional block defaults to self-attention, with the latent array serving as the query, key, and value. [21] : line 251

In pseudocode,

The detailed architecture may be found in. [22] [23]

Training and inference

The LDM is trained by using a Markov chain to gradually add noise to the training images. The model is then trained to reverse this process, starting with a noisy image and gradually removing the noise until it recovers the original image.

More specifically, the training process can be described as follows:

Forward diffusion process: Given a real image x 0 {\displaystyle x_{0}} , a sequence of latent variables x 1 : T {\displaystyle x_{1:T}} are generated by gradually adding Gaussian noise to the image, according to a pre-determined "noise schedule".

Reverse diffusion process: Starting from a Gaussian noise sample x T {\displaystyle x_{T}} , the model learns to predict the noise added at each step, in order to reverse the diffusion process and obtain a reconstruction of the original image x 0 {\displaystyle x_{0}} .

The model is trained to minimize the difference between the predicted noise and the actual noise added at each step. This is typically done using a mean squared error (MSE) loss function.

Once the model is trained, it can be used to generate new images by simply running the reverse diffusion process starting from a random noise sample. The model gradually removes the noise from the sample, guided by the learned noise distribution, until it generates a final image.

See the diffusion model page for details.

See also

Diffusion model

Generative adversarial network

Variational autoencoder

Stable Diffusion

References

Further reading

Wang, Phil (2024-09-07). "lucidrains/denoising-diffusion-pytorch" . GitHub . Retrieved 2024-09-07 .

"The Annotated Diffusion Model" . huggingface.co . Retrieved 2024-09-07 .

"U-Net for Stable Diffusion" . U-Net for Stable Diffusion . Retrieved 2024-08-31 .

"Transformer for Stable Diffusion U-Net" . Transformer for Stable Diffusion U-Net . Retrieved 2024-09-07 .

History timeline

timeline

Companies

Projects

Parameter Hyperparameter

Hyperparameter

Loss functions

Regression Bias–variance tradeoff Double descent Overfitting

Bias–variance tradeoff

Double descent

Overfitting

Clustering

Gradient descent SGD Quasi-Newton method Conjugate gradient method

SGD

Quasi-Newton method

Conjugate gradient method

Backpropagation

Attention

Convolution

Normalization Batchnorm

Batchnorm

Activation Softmax Sigmoid Rectifier

Softmax

Sigmoid

Rectifier

Gating

Weight initialization

Regularization

Datasets Augmentation

Augmentation

Prompt engineering

Reinforcement learning Q-learning SARSA Imitation Policy gradient

Q-learning

SARSA

Imitation

Policy gradient

Diffusion

Latent diffusion model

Autoregression

Adversary

RAG

Uncanny valley

RLHF

Self-supervised learning

Reflection

Recursive self-improvement

Hallucination

Word embedding

Vibe coding

Machine learning In-context learning

In-context learning

Artificial neural network Deep learning

Deep learning

Language model Large language model NMT

Large language model

NMT

Reasoning language model

Model Context Protocol

Intelligent agent

Artificial human companion

Humanity's Last Exam

Artificial general intelligence (AGI)

AlexNet

WaveNet

Human image synthesis

HWR

OCR

Computer vision

Speech synthesis 15.ai ElevenLabs

15.ai

ElevenLabs

Speech recognition Whisper

Whisper

Facial recognition

AlphaFold

Text-to-image models Aurora DALL-E Firefly Flux Ideogram Imagen Midjourney Recraft Stable Diffusion

Aurora

DALL-E

Firefly

Flux

Ideogram

Imagen

Midjourney

Recraft

Stable Diffusion

Text-to-video models Dream Machine Runway Gen Hailuo AI Kling Sora Veo

Dream Machine

Runway Gen

Hailuo AI

Kling

Sora

Veo

Music generation Riffusion Suno AI Udio

Riffusion

Suno AI

Udio

Word2vec

Seq2seq

GloVe

BERT

T5

Llama

Chinchilla AI

PaLM

GPT 1 2 3 J ChatGPT 4 4o o1 o3 4.5 4.1 o4-mini 5

1

2

3

J

ChatGPT

4

4o

o1

o3

4.5

4.1

o4-mini

5

Claude

Gemini Gemini (language model) Gemma

Gemini (language model)

Gemma

Grok

LaMDA

BLOOM

DBRX

Project Debater

IBM Watson

IBM Watsonx

Granite

PanGu-Σ

DeepSeek

Qwen

AlphaGo

AlphaZero

OpenAI Five

Self-driving car

MuZero

Action selection AutoGPT

AutoGPT

Robot control

Alan Turing

Warren Sturgis McCulloch

Walter Pitts

John von Neumann

Claude Shannon

Shun'ichi Amari

Kunihiko Fukushima

Takeo Kanade

Marvin Minsky

John McCarthy

Nathaniel Rochester

Allen Newell

Cliff Shaw

Herbert A. Simon

Oliver Selfridge

Frank Rosenblatt

Bernard Widrow

Joseph Weizenbaum

Seymour Papert

Seppo Linnainmaa

Paul Werbos

Geoffrey Hinton

John Hopfield

Jürgen Schmidhuber

Yann LeCun

Yoshua Bengio

Lotfi A. Zadeh

Stephen Grossberg

Alex Graves

James Goodnight

Andrew Ng

Fei-Fei Li

Alex Krizhevsky

Ilya Sutskever

Oriol Vinyals

Quoc V. Le

Ian Goodfellow

Demis Hassabis

David Silver

Andrej Karpathy

Ashish Vaswani

Noam Shazeer

Aidan Gomez

John Schulman

Mustafa Suleyman

Jan Leike

Daniel Kokotajlo

François Chollet

Neural Turing machine

Differentiable neural computer

Transformer Vision transformer (ViT)

Vision transformer (ViT)

Recurrent neural network (RNN)

Long short-term memory (LSTM)

Gated recurrent unit (GRU)

Echo state network

Multilayer perceptron (MLP)

Convolutional neural network (CNN)

Residual neural network (RNN)

Highway network

Mamba

Autoencoder

Variational autoencoder (VAE)

Generative adversarial network (GAN)

Graph neural network (GNN)

Category