Generative Artificial Intelligence

Stable diffusion – Generative Adversarial Network – Variational Autoencoder.

Generative AI | Iheb BOUARICHE | 19-07-2023

Generative AI:

Generative AI, short for Generative Artificial Intelligence, refers to a branch of artificial intelligence that focuses on creating systems and models capable of generating new and original content. Unlike traditional AI models that are primarily used for classification or prediction tasks, generative AI aims to produce novel and creative outputs such as images, music, text, and even realistic simulations.

At the core of generative AI lies the concept of generative models. These models are designed to learn the underlying patterns and structure of a given dataset, enabling them to generate new samples that resemble the original data. They accomplish this by learning from vast amounts of training data and extracting meaningful features, allowing them to generate new content that exhibits similar characteristics.

Autoencoder:

An autoencoder is a type of neural network architecture that is used for unsupervised learning and dimensionality reduction. It is designed to encode and decode input data, aiming to reconstruct the original input as accurately as possible. Autoencoders are composed of two main components: an encoder and a decoder.

Encoder:

The encoder takes an input and maps it to a lower-dimensional representation, often referred to as a "latent space" or "encoding." This encoding captures the essential features and patterns of the input data in a compressed form. The dimensionality of the encoding is typically smaller than the dimensionality of the input data, allowing for efficient representation.

Decoder:

The decoder takes the encoded representation from the encoder and attempts to reconstruct the original input data. It maps the encoded representation back to the original input space, aiming to produce an output that closely resembles the input data. The decoder essentially learns to generate output samples based on the learned encoding.

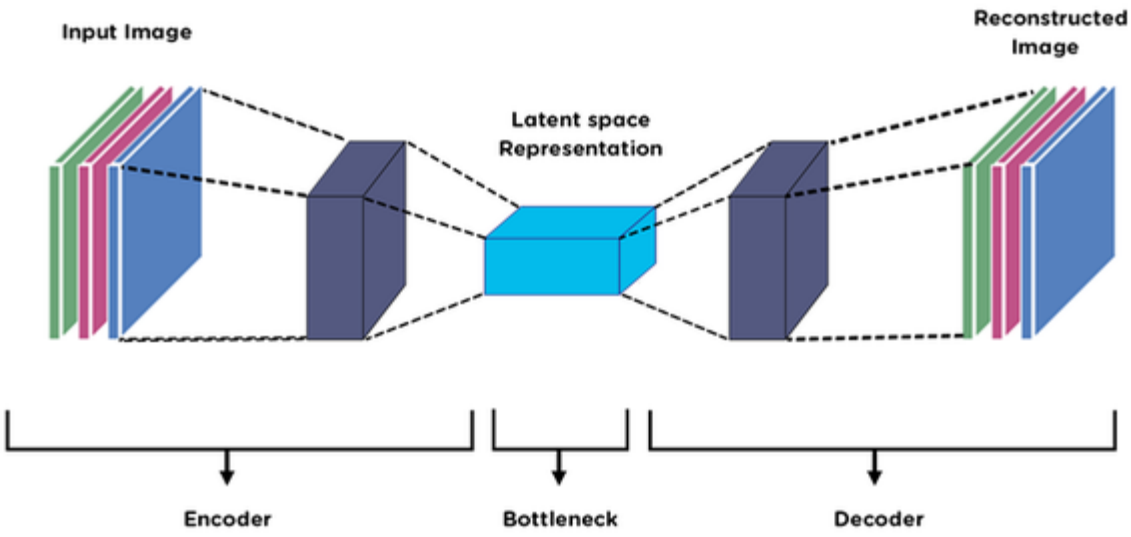


Figure 01: Autoencoder architecture.

Autoencoder Downside:

One downside is that traditional autoencoders solely focus on reconstructing the input data without considering the distribution of the latent space. As a result, they may produce latent representations that are not well-structured or easily interpretable. Additionally, traditional autoencoders lack a probabilistic framework, making them less suitable for tasks such as generating new data samples. To address these limitation, Variational autoencoder was introduced.

Variational autoencoder:

VAEs incorporate probabilistic concepts into the autoencoder framework, enabling them to model the underlying distribution of the latent space. By doing so, VAEs offer a more robust and flexible approach to data generation and latent space exploration.

In a Variational Autoencoder, the encoder does not directly output a point in the latent space. Instead, it produces a probability distribution over the latent variables. This distribution is typically modeled as a multivariate Gaussian with a mean and variance. The mean and variance are learned by the encoder based on the input data.

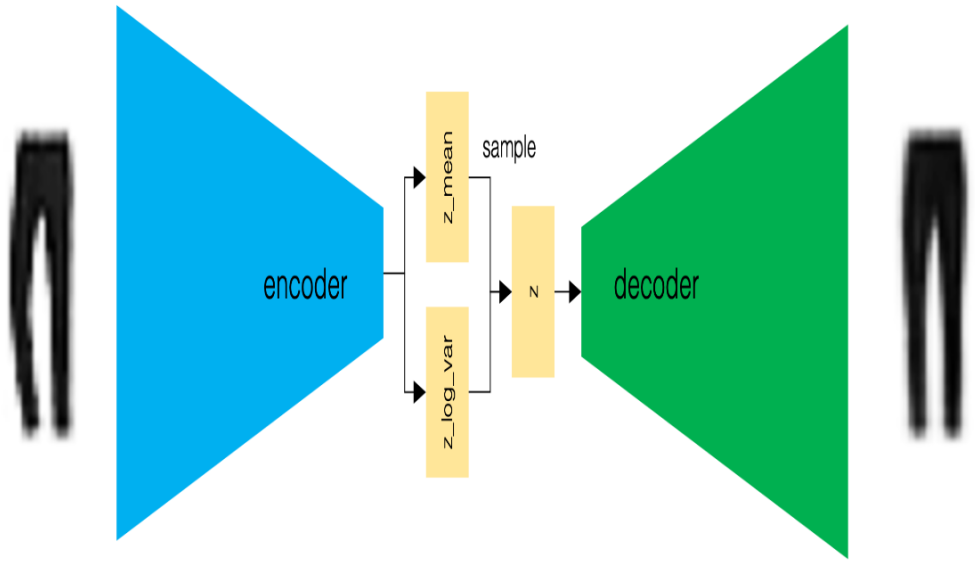


Figure 02: Variational autoencoder architecture

During training, the VAE introduces a regularization term that encourages the latent distribution to match a prior distribution, usually a standard Gaussian. This regularization term is known as the Kullback-Leibler (KL) divergence and ensures that the latent space is well-structured and follows a known distribution.

> "KL divergence is a way of measuring how much one probability distribution differs from another. In a VAE, we want to measure how much our normal distribution with parameters `z_mean` and `z_log_var` differs from a standard normal distribution."

To generate new samples, VAEs utilize the decoder in conjunction with sampling from the learned latent distribution. By sampling from the latent space, VAEs can generate diverse and realistic outputs. This capability makes VAEs particularly valuable for tasks like image generation, where they can produce new and unique visual samples.

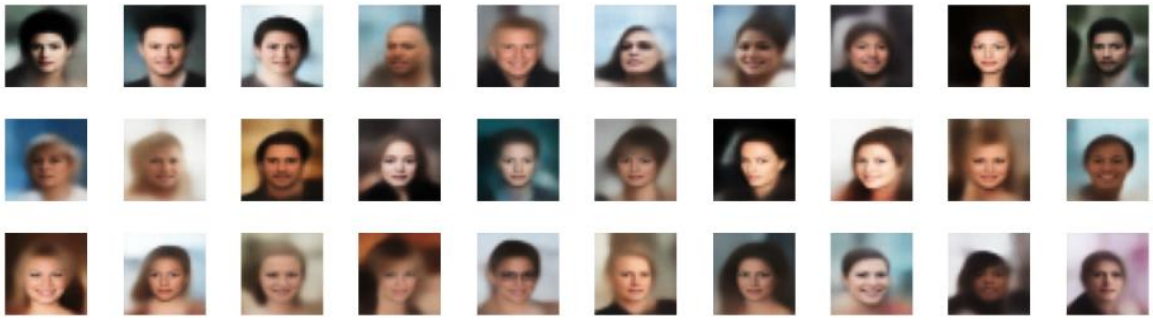


Figure 03: Generated images with Variational autoencoder.

Generative adversarial network:

A Generative Adversarial Network (GAN) is a type of machine learning model that consists of two neural networks, a generator and a discriminator.

- Generator:

The generator network in a GAN is responsible for creating new samples, such as images, sounds, or text, that resemble the training data it was provided. The generator takes random input, often referred to as noise, and tries to generate realistic outputs. Initially, the generator's outputs are random and of low quality.

- Discriminator:

The discriminator network acts as a critic and tries to distinguish between the real samples from the training data and the fake samples generated by the generator. The discriminator is trained on a dataset containing real examples and is updated to become better at differentiating between real and fake samples.

The training process:

The training process of GANs involves a competition between the generator and the discriminator. The generator aims to generate realistic samples that can fool the discriminator, while the discriminator aims to correctly identify the real samples. The generator and discriminator are trained in alternating cycles, with each iteration updating their respective parameters.

Through this adversarial process, the generator gradually improves its ability to generate realistic samples, while the discriminator improves its ability to differentiate between real and fake samples. Ideally, this leads to a point where the generated samples are indistinguishable from real samples, as judged by the discriminator.

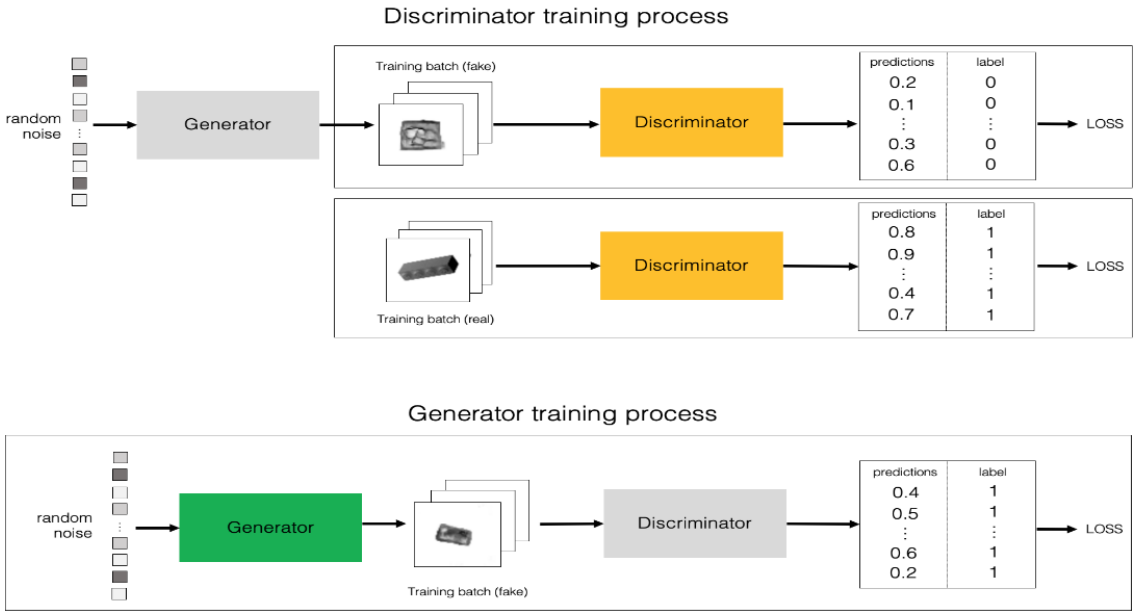


Figure 04: The training process of a GAN.

Problems with GANs: is the Mode collapse, it occurs when the generator produces limited or repetitive outputs, failing to capture the full diversity of the training data. The generator collapses multiple modes into a single mode. mode collapse can occur when the generator and discriminator networks are not properly balanced in terms of their capacities or when the training data itself is highly unbalanced. If the generator network is not sufficiently powerful or if the discriminator is too dominant, the generator may struggle to capture the full complexity of the data distribution. To address this problem, we have Wasserstein GAN - Gradient Penalty (WGAN-GP).

- Wasserstein GAN - Gradient Penalty (WGAN-GP):

Wasserstein GAN with Gradient Penalty (WGAN-GP) is an improved version of the original Wasserstein GAN (WGAN). The original WGAN addressed some of the problems of the traditional GANs, such as mode collapse and instability during training, by introducing the Wasserstein distance as a more stable metric for measuring the distance between the real and generated data distributions.

However, the original WGAN had a limitation in that it required weight clipping to enforce the Lipschitz constraint on the discriminator. Weight clipping could lead to optimization issues and make training difficult. To overcome this limitation, the WGAN-GP introduced a gradient penalty term, which alleviates the need for weight clipping and improves the stability of the training process.

The key idea of WGAN-GP is to add a penalty term to the Wasserstein loss that encourages the gradient of the discriminator's output with respect to its input to have a norm close to 1. This penalty term is computed on random points along the straight lines connecting the real and generated samples. The Wasserstein distance is then minimized along with this gradient penalty.

- Conditional GAN:

Conditional GAN allows the generator to generate data conditioned on specific input information. The key idea behind a conditional GAN is that both the generator and the discriminator take additional conditional information, typically represented as extra input vectors, alongside the random noise and real data samples. This conditional information acts as a "label" or "class" that

guides the generator to produce data that belongs to a specific category or exhibits certain attributes.

Figure 05: Generated images with GAN.

Diffusion Models:

In the rapidly evolving field of generative modeling, diffusion models have emerged as a groundbreaking technique alongside GANs, dominating image generation benchmarks and gaining popularity among practitioners. Inspired by thermodynamic diffusion, the breakthrough came with the Denoising Diffusion Probabilistic Model (DDPM) in 2020, rivaling GANs' performance.

- Denoising Diffusion Probabilistic Model (DDPM):

DDPM is a generative model that aims to generate high-quality data samples by denoising corrupted versions of the data. It was introduced in 2020. The model is built upon the idea of diffusion processes, where data is sequentially corrupted with increasing noise levels, and the model's task is to recover the original clean data by denoising each intermediate noisy version. It operates in a probabilistic framework, allowing it to capture uncertainty in the data and generate diverse samples.

The training of DDPM involves iteratively sampling corrupted versions of the training data and optimizing the model to denoise these corrupted samples effectively. The key components of the model are the diffusion process and the diffusion-based generator.

The diffusion process starts from a fixed noise distribution (e.g., Gaussian) and gradually introduces noise into the data. At each step, the noise level increases, and the data becomes more corrupted. The process is defined as follows:

$$x_{t+1} = (1 - \sqrt{1 - \beta_t}) * x_t + \sqrt{1 - (1 - \beta_t)} * \eta_t$$

Here, "***x_t"*** is the data at time step t, "***beta_t"*** is a schedule that controls the noise level at each step, and "***eta_t"*** is a sample from the noise distribution at time step t.

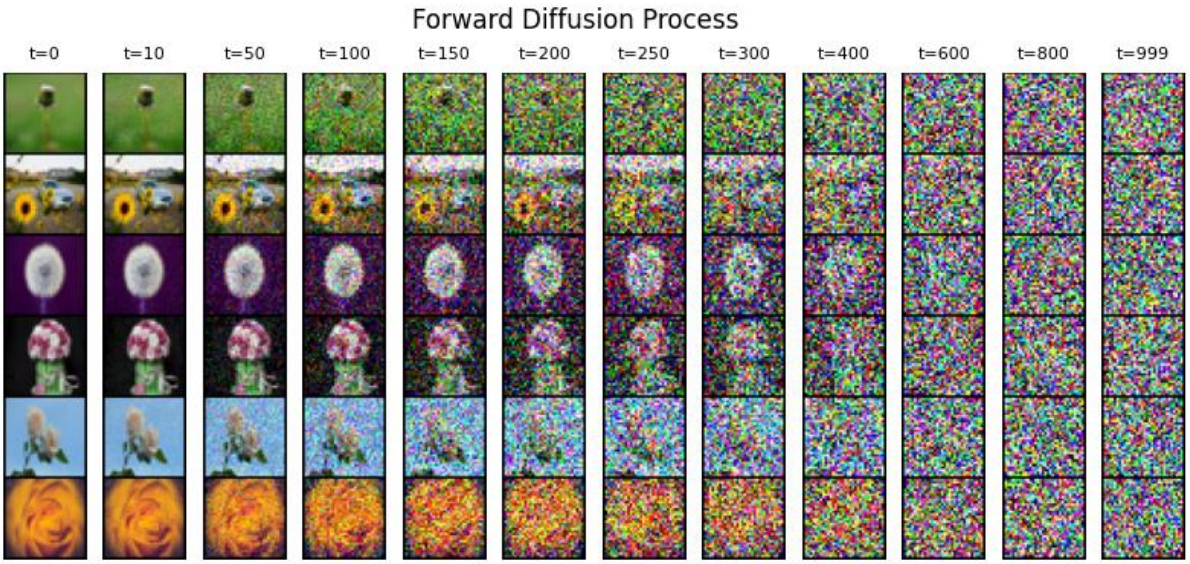


Figure 06: Diffusion process.

Diffusion-Based Generator network takes the corrupted data at each diffusion step and aims to denoise it, producing an estimate of the underlying clean data. The generator is trained to approximate the inverse of the diffusion process, allowing it to denoise the data effectively.

During inference, the model can generate new samples by iteratively denoising a sample from the noise distribution in a reverse manner, going backward through the diffusion steps to produce a realistic sample.

DDPM has shown promising results in generating high-quality samples for a variety of data types, including images and audio. It leverages the idea of diffusion processes and employs neural networks to achieve state-of-the-art performance in generating diverse and realistic data samples.

Keep in mind that research in this field progresses rapidly, and there have been many models and with different neural network architectures that have been released in the last few years (e.g., U-net).

- U-net denoising model architecture:

It contains residual blocks, which are group of layers that contains a skip connection that adds the input to the output. Residual blocks help us to build deeper networks that can learn more complex patterns without suffering as greatly from vanishing gradient and degradation problems.

DownBlock increases the number of channels via block depth of 2, whilst also applying a final AveragePooling2D layer in order to halve the spatial dimensionality of the image.

UpBlock applies an UpSampling2D layer that doubles the spatial dimensionality of the image, through bilinear interpolation. Each successful UpBlock decreases the number of channels via block depth of 2, whilst also concatenating the outputs from the DownBlock through skip connections across the U-Net.

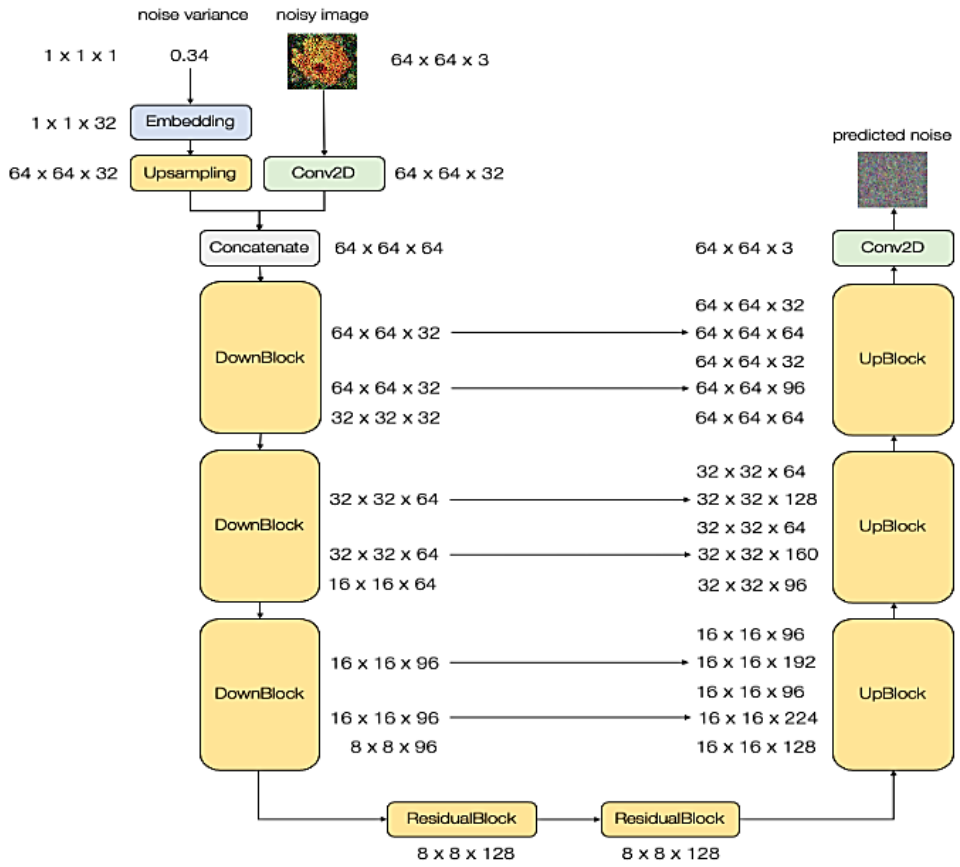


Figure 07: The U-Net architecture.

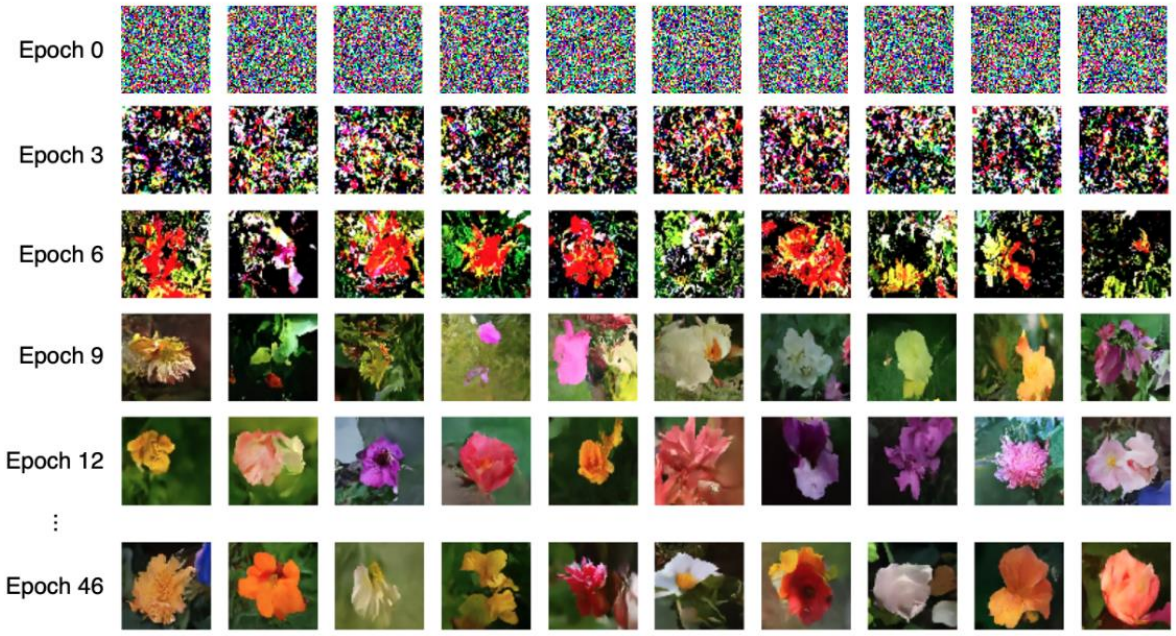


Figure 08: Generated images with Stable Diffusion.

Normalizing flows:

Is a process used in Generative AI for mapping between simple distribution (the base distribution) and complex distribution (the target distribution) by using neural network. The main objective of Normalizing Flows is to learn a sequence of invertible transformations, where each transformation takes a sample from the base distribution and maps it to a sample in the target distribution. By learning this mapping, the Normalizing Flow model gains the ability to generate new samples that resemble the target complex distribution. Moreover, since the transformations are invertible, we can also perform inference by mapping data points from the target distribution back to the base distribution.

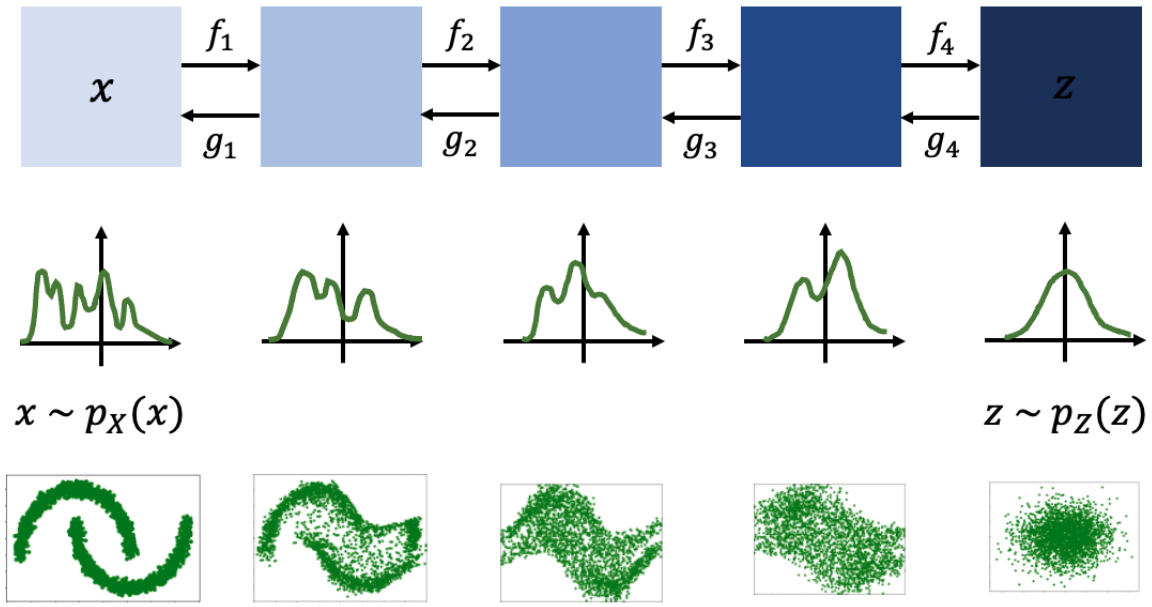


Figure 09: Normalizing Flow process.

- GLOW :

GLOW was one of the first models to demonstrate the ability of normalizing flows to generate high quality samples and produce a meaningful latent space that can be traversed to manipulate samples.

Figure 10: Generated images with GLOW.

Energy-based models:

Energy-based models (EBMs) are a class of generative models used in artificial intelligence, specifically in the field of generative modeling. The fundamental idea behind energy-based models is to define an energy function that measures the compatibility or quality of a given input data point. The model's goal is to learn an energy function that assigns low energy to real data points and high energy to data points that do not belong to the true data distribution.

In the context of generative AI, the main objective of energy-based models is to generate new data samples that resemble the training data. This is typically achieved through the process of sampling from the model's learned energy function. The lower the energy assigned to a generated sample, the more likely it is to be similar to real data.

Energy-based models can be trained in a few different ways, but one common approach is to use contrastive divergence or Markov Chain Monte Carlo (MCMC) methods. Contrastive divergence is often used in training Restricted Boltzmann Machines (RBMs), which are a type of energy-based model.

Langevin dynamics is a technique used to sample from deep EBMs. It achieves this by updating the input data (image) in small steps, following the gradient of the energy function (score) with respect to the input, while keeping the model parameters fixed. This process gradually transforms random noise into plausible observations by exploring low-energy regions of the model's distribution.

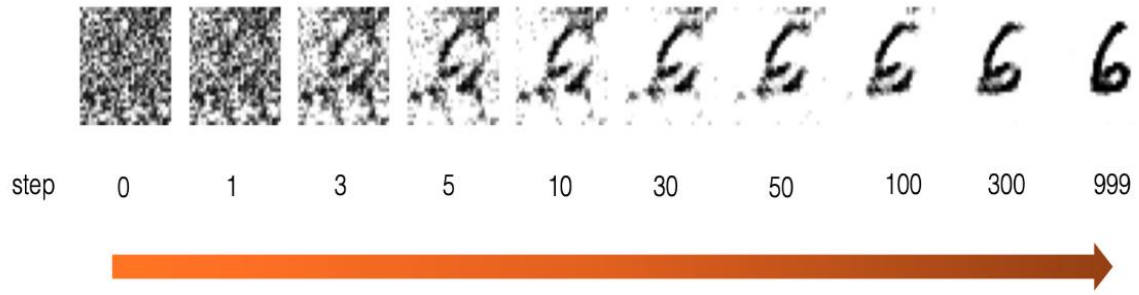


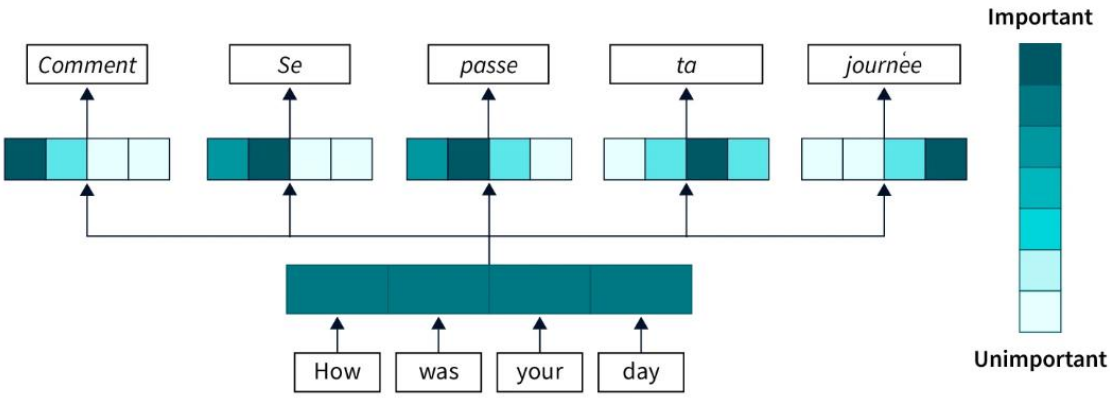
Figure 11: Generated images by EBM.

Transformer for texts generation:

a. Attention mechanism:

An attention mechanism is a crucial component in many machine learning models, particularly in natural language processing. It enables models to focus on specific parts of the input data by assigning varying weights or importance. It is commonly employed in sequence-to-sequence models, such as recurrent neural networks (RNNs).

The attention mechanism calculates attention weights for each element in the input sequence, indicating their relative importance. These attention weights are then used as inputs instead of the output of the last hidden layer in the RNN. By doing this, the vector can capture all the sequence information, representing the focused parts. This context vector, combined with other information, helps generate the output or make predictions.



In the context of the attention mechanism, RNNs are used to process the input sequence and generate a sequence of hidden states. These hidden states capture the contextual information at each step of the input sequence. The attention mechanism then calculates attention weights based on these hidden states, indicating the relative importance of each input element. These attention weights are used to create a weighted context vector that represents the focused parts of the input sequence. This context vector, along with other information, is then used to generate the output or make predictions. By utilizing RNNs within the attention mechanism, the model can effectively capture the sequential dependencies and contextual information present in the input sequence.

Attention mechanisms improve model performance by capturing all the relevant information in the input sequence, thereby leading to better results in tasks such as machine translation, text summarization, and sentiment analysis. One limitation of this mechanism is that it cannot be parallelized efficiently due to the recurrent neural network.

"This approach continues to have an important limitation; each sequence must be treated one element at a time. Both the encoder and the decoder have to wait till the completion of t-1 steps to process thet-th step. So, when dealing with huge corpus it is very time consuming and computationally inefficient."

-Eduardo Muñoz- Towards Data Science

b. What's Transformer?

The transformer is currently regarded as the most powerful deep learning model architecture and has gained widespread usage in the field of artificial intelligence, particularly in natural language processing (NLP) tasks. It was introduced in a groundbreaking paper titled "Attention Is All You Need," published by Vaswani et al. in 2017.

The transformer architecture revolutionized NLP by addressing the limitations of previous recurrent neural network (RNN) and convolutional neural network (CNN) models. It achieves this by relying on a mechanism called self-attention, which allows the model to weigh the importance of different words or tokens in a sequence when processing each individual word.

The transformer model consists of an encoder and a decoder. The encoder processes the input sequence, such as a sentence, and extracts its contextual representation, while the decoder generates an output sequence based on the encoder's representation and an attention mechanism. Each encoder and decoder layer in the transformer contains multiple self-attention heads and feed-forward neural networks, enabling the model to capture complex dependencies and long-range dependencies in the input sequence.

One of the key advantages of the transformer architecture is its ability to parallelize computations, making it more efficient to train and allowing for faster inference times. This parallelization is possible because the self-attention mechanism allows the model to process all input tokens in parallel, unlike the sequential nature of RNNs.

Transformers have been successfully applied to a wide range of NLP tasks, such as machine translation, text summarization, sentiment analysis, question answering, and language generation. Moreover, the transformer architecture has also been adapted to other domains, including computer vision (e.g., vision transformers), audio processing, reinforcement learning and others.

The Transformer model has significantly advanced the state-of-the-art in NLP and has become a fundamental building block in many AI systems due to its ability to capture complex dependencies and improve performance on various tasks.

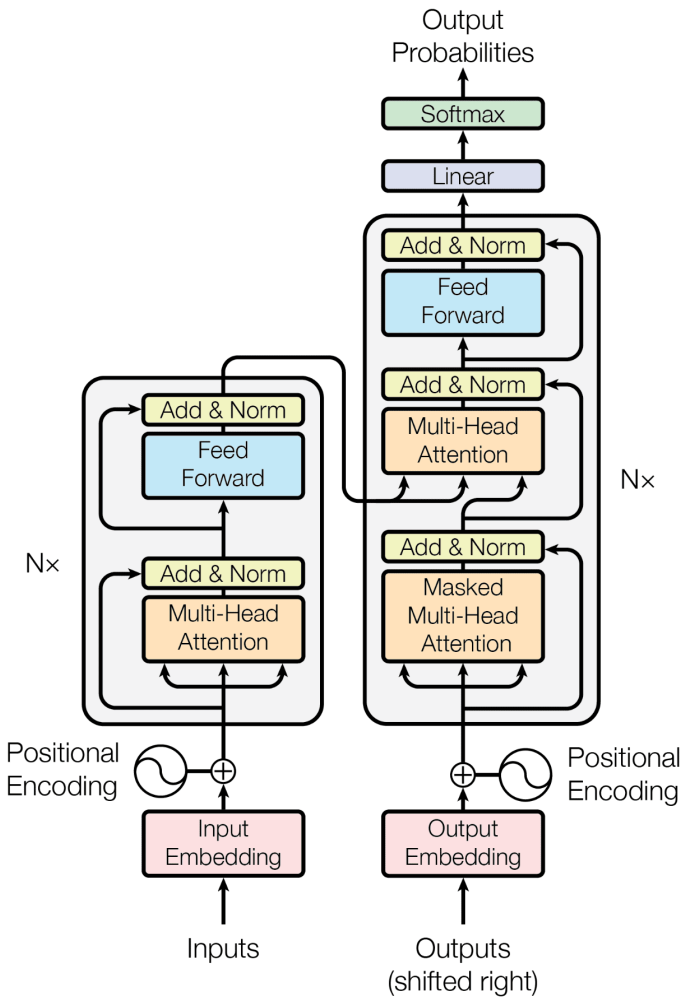


Figure 12: The Transformer architecture

We will describe the components of this model and analyze their operations in the following papers.

c. Self-Attention mechanism:

The core element in transformers is the self-attention mechanism, which empowers the model to concentrate on relevant parts of the input by calculating weights for each position in the sequence and considering the relationships between all positions.

In natural language processing (NLP), the input consists of vectors that represent each word in a sequence. Self-attention operates on these vectors and computes a weighted sum of the input vectors. This computation is achieved by utilizing three trainable vectors known as Keys, Queries, and Values.

d. Key and Query and Value:

"Keys," "queries," and "values" describe the three linear transformations that occur during the calculation of attention weights. These transformations are typically applied to input sequences or feature maps to capture relationships and compute the importance of different elements within the sequence.

Here's a breakdown of each component:

1- Keys: are used to encode information about the elements in the input sequence. They are derived from the input sequence by applying a linear transformation to it. The purpose of keys is to represent the elements in a way that captures their characteristics and allows the model to learn their relationships with other elements in the sequence.
2- Queries: Queries are derived from the input sequence as well, but they are used to obtain the attention weights by comparing them with the keys. Queries, similar to keys, undergo a linear transformation to represent the elements in a suitable format for comparison.
3- Values: Values are also derived from the input sequence, and they contain the actual information or features associated with each element. They are transformed linearly to obtain the value representations.

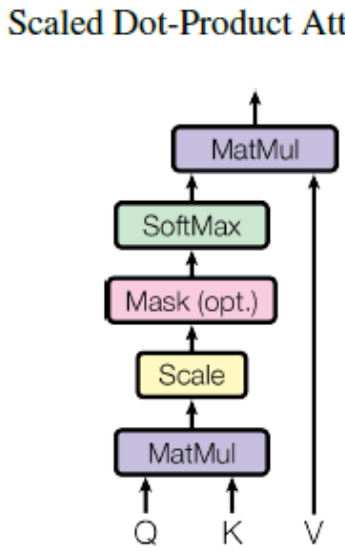
Once the keys, queries, and values are obtained, the attention mechanism calculates the attention weights by measuring the similarity between the queries and keys. This similarity score is used to determine the importance or relevance of each element in the sequence to the others. Finally, the attention weights are used to weigh the corresponding values, generating the context vector or weighted sum of values, which represents the output of the self-attention mechanism.

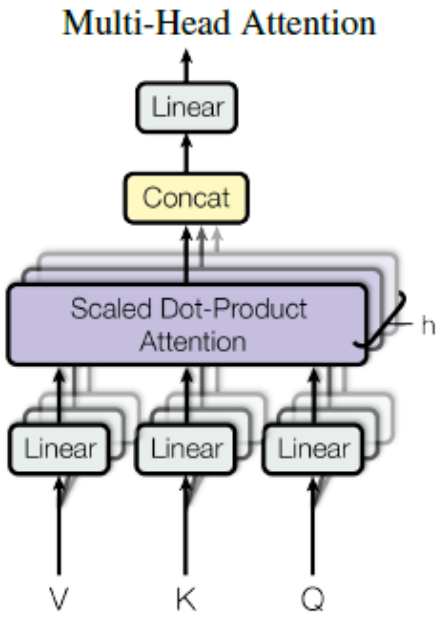
Mathematically, one head of self-attention mechanism computes its outputs by first transforming the input sequence into three matrices: the key matrix (K), the query matrix (Q), and the value matrix (V). These transformations are obtained by multiplying the input sequence by learnable weight matrices. The attention weights are then calculated by taking the dot product between each query vector and the corresponding key vectors, scaled by a factor of the square root of the dimensionality of the key vectors (in order to have a stable gradient). This similarity score is further passed through a SoftMax function to obtain normalized attention weights. The attention weights are then multiplied element-wise with the value vectors to obtain the weighted value vectors. Finally, these weighted value vectors are summed together to form the output of the self-attention mechanism, which represents the context vector capturing the important relationships between the elements in the input sequence.

e. Multi-head self-attention mechanism:

Multi-head attention is an extension of the self-attention mechanism that introduces multiple parallel self-attention layers, each called a "head." Each head has its own set of learnable weight matrices for keys, queries, and values, enabling it to attend to different information and capture diverse patterns in the input sequence.

After obtaining the outputs from the multi-head attention mechanism, concatenating them and feeding the concatenated outputs into a feed-forward neural network helps capture complex relationships and dependencies among elements in the input sequence. This process enhances the representation and expressive power of the self-attention mechanism.





f. Encoder:

The Transformer encoder is responsible for processing the input sequence. It consists of multiple layers, each containing two sub-components: multi-head self-attention mechanism and position-wise feed-forward neural networks (FFNNs). The input sequence is transformed into a sequence of context-aware representations through these layers. Each layer in the encoder attends to the previous layer's representations, allowing the model to capture dependencies and relationships within the input sequence effectively.

g. Decoder:

The Transformer decoder, on the other hand, generates the output sequence based on the encoded representations. It also consists of multiple layers, each containing three sub-components: masked multi-head self-attention mechanism, encoder-decoder attention mechanism, and position-wise FFNNs. The decoder attends to both the encoded representations from the encoder and the previously generated tokens in an autoregressive manner to generate the next token in the output sequence.

The encoder-decoder attention mechanism allows the decoder to focus on relevant parts of the input sequence while generating the output. The masked self-attention mechanism ensures that the decoder attends only to the previously generated tokens and prevents it from attending to future tokens during the generation process.

There are other variations of the Transformer architecture that utilize either only an encoder or a decoder, depending on the specific task.

- Text generation:

Generative models for text (e.g., ChatGPT) are decoder only models and they are trained on big corpus of data from internet in order to predict the next word based on the previous words. Then the next step is fine tuning the model with a process based on human feedback (e.g., Reinforcement Learning with Human Feedback).

Transformer for image generation:

By using the transformer, we can covert textual data to vectors (e.g., encoder), and this vector can be fed to another model that can generate images (e.g., decoder). There are different methods and architectures to map between text and images. DALL-E is a prominent example of such architectures that have been used for this purpose.

Figure 13: Generated images with DALLE 2.0.

Conclusion:

We presented in this report the most powerful methods used in generating AI, and note that this is a rapidly developing field due to the substantial amount of research on it. Therefore, you will find numerous different architectures and methods, but all of them are based on the methods presented in this report.