

## Deep Learning: advanced models Generative models, Transformers, GNNs

Nicolas Keriven  
CNRS, IRISA, Rennes

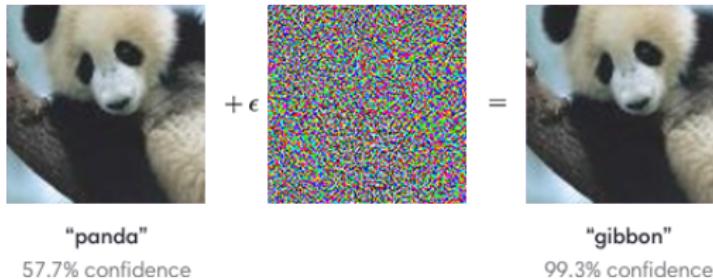
(material from Florent Chatelain, Olivier Michel)

ENSTA 2023

## Pros and cons of deep learning

Deep Neural Nets are :

- ▶ **Data-hungry** : they *will not* work with only a few data
- ▶ **Power-hungry** : computationally expensive to train
- ▶ **Difficult to interpret** : hard to get confidence intervals, generalization guarantees...
- ▶ **Unstable** : a micro-change in the data can fool them. Such change almost never happens in practice, but can be engineered by a malicious party !



- ▶ A **bazooka**, when sometimes all you need to do is kill a fly ! In many real-life scenarii, linear models, PCA,  $k$ -means... work amazingly well

## Pros and cons of deep learning

But ! Deep Neural Nets :

- ▶ are easy to implement with modern libraries, with **automatic differentiation** and **GPU compatibility completely transparent** but there are many tricks to train them ! Can be frustrating...
- ▶ work **mysteriously well** for problems we thought out-of-reach for decades !
- ▶ are **amazingly flexible** ! You can plug “anything anywhere”, and it will almost always “work”. Huge playground ! This can be dangerous ! Always be aware of what you do, or bugs will become impossible to fix.
- ▶ Today, we are going to see a few variants of DNNs, without going into details. Many resources are available online.

## Table of Contents

### Generative Models

Generative Adversarial Networks

Variational Auto-encoder

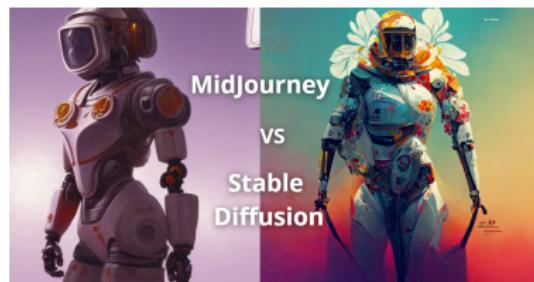
Diffusion models

Attention, Transformers

Conclusions and open questions

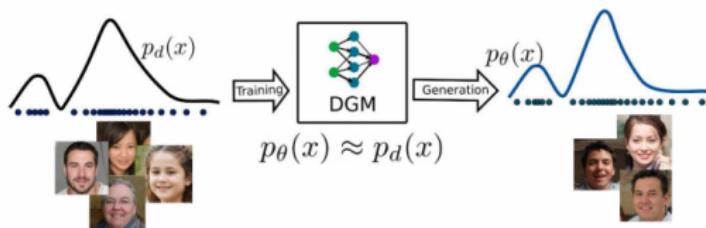
## Generative model

- ▶ Goal : generate data resembling other data.
- ▶ Extremely hot topic right now ! Text generation, image generation...



## Basic generative model : density estimation !

- ▶ Most basic problem : observe samples from a distribution, generate **new** samples not observed before
- ▶ Just a *density estimation* problem ! But the probability distribution is **far too complicated** to be modelled explicitly (eg images)



- ▶ Basic idea : take a very simple distribution in a **latent space**, eg  $z \sim \mathcal{N}(0, Id)$ , train a **complicated** function  $f_\theta$  such that  $f_\theta(z)$  are good samples.

## Table of Contents

### Generative Models

  Generative Adversarial Networks

  Variational Auto-encoder

  Diffusion models

### Attention, Transformers

### Conclusions and open questions

## Generative Adversarial Networks (GANs)

- ▶ Q : how to train a Neural Network  $f_\theta$  such that  $f_\theta(z)$  produces "realistic images" different from the training set ? Images are already hard to understand !
- ▶ Breakthrough idea : only Neural Networks are capable of automatically understanding images... well, let's train **two CNNs**! One to produce the images, one to criticize them, and make them compete against each other! **Generative Adversarial Networks** (Goodfellow et al. 2014)



a)



b)



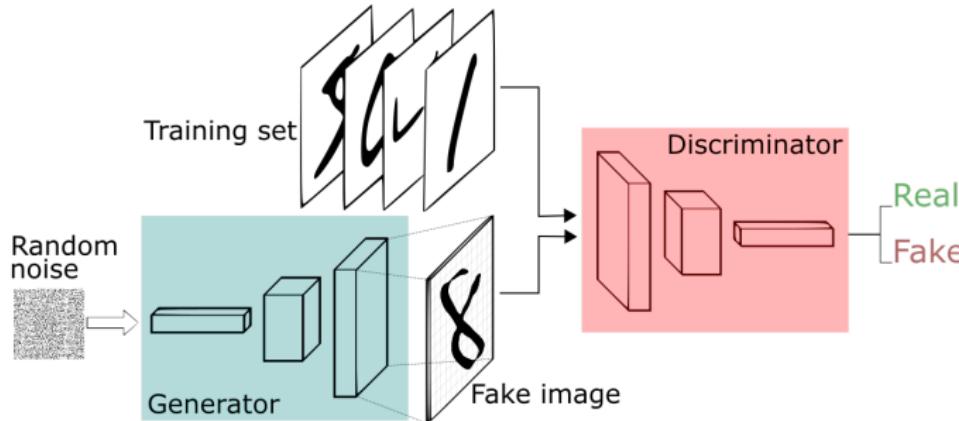
c)



d)

## GAN description

- ▶ Train a **generator** network  $G_{\theta_g}$  such that  $G_{\theta_g}(z)$ , with  $z$  Gaussian, are good samples
- ▶ Train a **discriminator** network  $D_{\theta_d}$  to classify the samples as being from the training set or from the generator
- ▶ Make them “compete” : the discriminator will become better at **classifying the fake samples**, the generator will become better at **fooling the discriminator**



## GAN training

- ▶ The original GAN paper presents the problem as a **two-players minimax game**, based on the logistic regression loss :

$$\min_{\theta_g} \max_{\theta_d} \mathbb{E}_{x \sim p_{data}} [\log D_{\theta_d}(x)] + \mathbb{E}_{z \sim \mathcal{N}} [\log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

- ▶  $D(x)$  represent the probability that  $x$  is a true sample.  $D$  wants to **maximize the log-likelihood**, ie, outputting 1 for  $x_i$  and 0 for  $G(z)$ .
- ▶  $G$  want to **minimize the log-likelihood**, ie, make  $G(z)$  ressemble the distribution of the data

---

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,  $k$ , is a hyperparameter. We used  $k = 1$ , the least expensive option, in our experiments.

---

for number of training iterations do

  for  $k$  steps do

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{data}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right].$$

  end for

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)}))).$$

end for

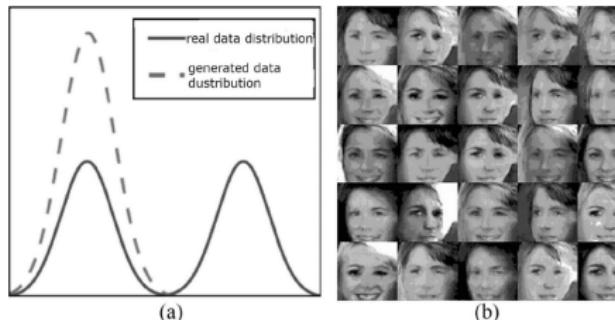
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---

Original algo : alternate  $k$  steps of gradient ascent for  $D$  with one step of gradient descent for  $G$

## GAN issues

- ▶ Hard to train : minimax problems are typically hard !
  - ▶ they can loop
  - ▶ saddle points are unstable
  - ▶ convergence is (very) slow, etc.
- ▶ Data-hungry : GANs require a **lot** of data to be trained properly
- ▶ Mode collapse : the generator produces limited diversity (eg, for a dataset with cats and dogs, produces only dogs)



## Sota GANs

There are many (many !) tricks and variants to improve GANs. Mostly, **bigger models with more data seem to be always better!** (BigGAN below). Seemed a bit depressing for a time...



But nowadays, they have been mostly replaced with more modern approaches like diffusion models.

## Table of Contents

### Generative Models

  Generative Adversarial Networks

  Variational Auto-encoder

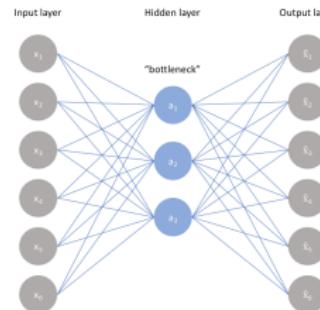
  Diffusion models

### Attention, Transformers

### Conclusions and open questions

## Auto-Encoder

- ▶ We have seen Auto-Encoder :
  - ▶ Encoder : images → compressed **latent space**
  - ▶ Decoder : latent space → image



- ▶ Idea : by working directly in the latent space, the decoder is a generative model !
- ▶ **Difficulty** : sampling a random vector in the latent space might produce garbage. The latent space is **highly irregular** !

## Variation Auto-Encoder (VAE) Kingma and Welling, 2014

<https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

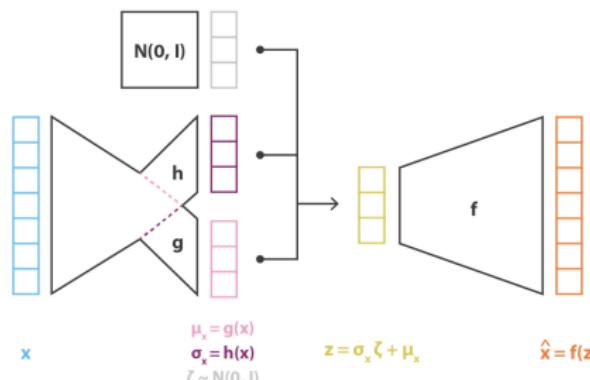
- ▶ Idea : **regularize** the distribution of the encodings, such that the latent space is smoother, and new sample can be produced
- ▶ To do so :
  - ▶ encode inputs as distributions, and feed sample from these distributions to the decoder.
  - ▶ Add a regularizing term (Kullback-Leibler divergence) on these distributions.



## Variation Auto-Encoder (VAE) Kingma and Welling, 2014

<https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>

- ▶ Idea : **regularize** the distribution of the encodings, such that the latent space is smoother, and new sample can be produced
- ▶ To do so :
  - ▶ encode inputs as distributions, and feed sample from these distributions to the decoder.
  - ▶ Add a regularizing term (Kullback-Leibler divergence) on these distributions.



---


$$\text{loss} = C \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = C \|x - f(z)\|^2 + \text{KL}[N(g(x), h(x)), N(0, I)]$$

## Variation Auto-Encoder (VAE) Kingma and Welling, 2014

- Desired properties : continuity (two close points in the latent space should give close outputs) and completeness (sampling any point should give a reasonable result)
- Connection with Variational Inference (hence the name)
- Easier to train than GANs, but output



## Table of Contents

### Generative Models

  Generative Adversarial Networks

  Variational Auto-encoder

  Diffusion models

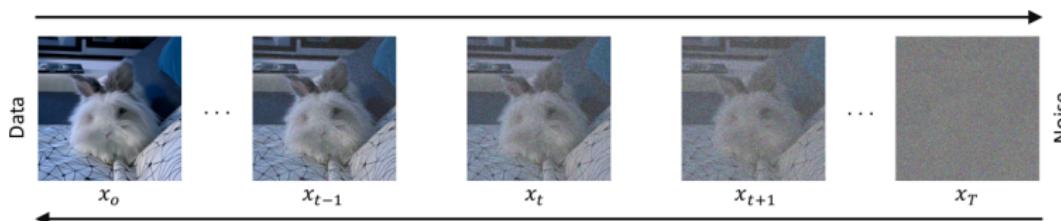
### Attention, Transformers

### Conclusions and open questions

## Diffusion models

<https://lilianweng.github.io/posts/2021-07-11-diffusion-models/>

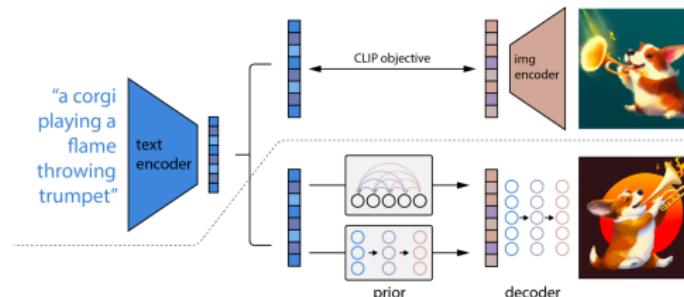
- ▶ Diffusion models have taken the generative world by storm and start to replace GANs and VAEs nowadays (Stable Diffusion, Midjourney, DALL-E, Imagen...), due to their performance and flexibility
- ▶ Diffusion models were *not* invented as generative models, and the idea seems a bit ludicrous at first glance !
- ▶ Idea (roughly) : train a model to **denoise** an image, at many different noise levels. Then, start with **random noise** (!), and denoise it until an image is produced !
- ▶ Interpretation : we learn to “reverse” the diffusion process that gradually transforms an image into pure noise



## Diffusion models : pros

### ► Pro : Flexibility

- ▶ denoising can happen **in the latent space**; somehow, learn to “project” random noise to a point in the latent space that makes sense
- ▶ possible to incorporate additional information/embedding at various stages before **conditional** diffusion happens : basis for all modern **text-to-image** systems ! (DALL-E, Stable Diffusion, Midjourney...)



- ▶ Pro : **Tractability and performance**. Every step is analytically tractable, thus improving stability and interpretability. Images are generally more realistic than GANs or VAEs
- ▶ Con : **slow sampling**. The sampling process is slower, we must perform many steps of denoising.

## Table of Contents

### Generative Models

Generative Adversarial Networks

Variational Auto-encoder

Diffusion models

### Attention, Transformers

Conclusions and open questions

## Transformers : what

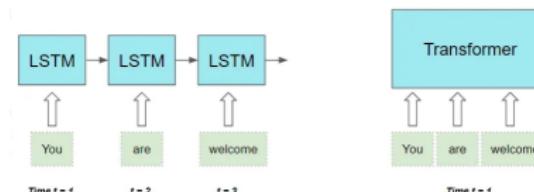
[https:](https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452)

//towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

- ▶ **Transformers** (Vaswani et al.), introduced by Google in 2017, have revolutionized Natural Language Processing (NLP)
- ▶ Documents classification, translation, text generation, text-to-image, image-to-text, chatbots...
- ▶ Well-known models, mostly from two companies :
  - ▶ Google : BERT, and successors : RoBERTa, ALBERT... (for French language : CamemBERT model!). Also T5, PaLM, etc.
  - ▶ OpenAI : GPT, and successors : GPT-2, GPT-3, ChatGPT...

## Transformers : how ?

- ▶ Unlike RNNs, Transformers almost totally discard the “sequential” nature of texts : everything depends on everything !



- ▶ far, far more parameters ! (usually in the billions) : huge training data, pre-trained components. Modern Transformers can almost only be trained by companies with massive funding and resources. Ethical problems...

The key to Transformers is **attention** : a (deceptively) simple mechanism to **model dependencies** between words/objects



## Attention : equations

- ▶ In NLP, words (or subwords) are *token*, embedded using a [tokenizer](#).
- ▶ Given a token  $e$ , and [context tokens](#)  $e_t$ , [attention coefficients](#) are computed to relate  $e$  to the context
- ▶ It can be [self-attention](#) (a word within its sentence), or [attention to a target context](#)
- ▶ The key operation is [softmax](#)  
 $a_k : x \mapsto \frac{e^{x_k}}{\sum_i e^{x_i}}$ . Attention is nothing else than common linear operations with learned parameters, [with proper normalization](#).

---

### Algorithm 3: Basic single-query attention.

---

**Input:**  $e \in \mathbb{R}^{d_{\text{in}}}$ , vector representation of the current token

**Input:**  $e_t \in \mathbb{R}^{d_{\text{in}}}$ , vector representations of context tokens  $t \in [T]$ .

**Output:**  $\tilde{v} \in \mathbb{R}^{d_{\text{out}}}$ , vector representation of the token and context combined.

**Parameters:**  $W_q, W_k \in \mathbb{R}^{d_{\text{attn}} \times d_{\text{in}}}$ ,  $b_q, b_k \in \mathbb{R}^{d_{\text{attn}}}$ , the query and key linear projections.

**Parameters:**  $W_v \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ ,  $b_v \in \mathbb{R}^{d_{\text{out}}}$ , the value linear projection.

---

```

1  $q \leftarrow W_q e + b_q$ 
2  $\forall t : k_t \leftarrow W_k e_t + b_k$ 
3  $\forall t : v_t \leftarrow W_v e_t + b_v$ 
4  $\forall t : \alpha_t = \frac{\exp(q^\top k_t / \sqrt{d_{\text{attn}}})}{\sum_u \exp(q^\top k_u / \sqrt{d_{\text{attn}}})}$ 
5 return  $\tilde{v} = \sum_{t=1}^T \alpha_t v_t$ 

```

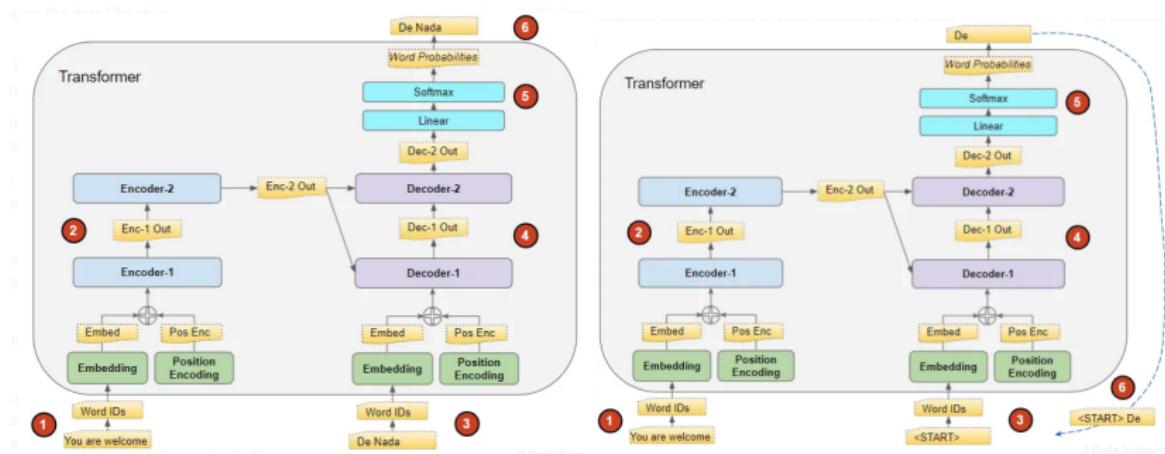
---

# Transformer : example of a (high-level) seq2seq architecture

<https://towardsdatascience.com/transf...>

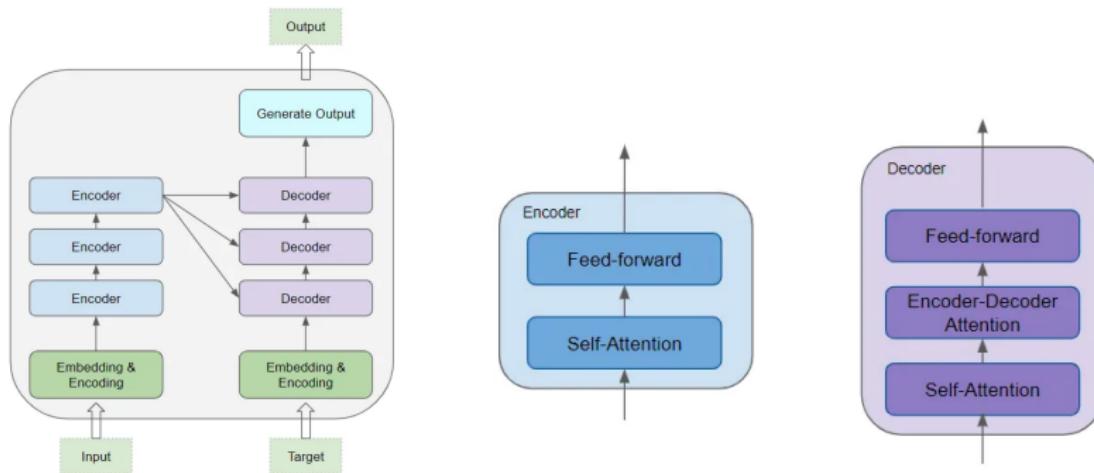
[//towardsdatascience.com/transf... explained-visual...-part-1-overview-of-fun...-95a6dd460452](https://towardsdatascience.com/transf...)

- ▶ Training : takes a query  $x$ , a target  $y$ , produces an output
- ▶ Inference : takes a query  $x$ , start with an empty target, generate words one by one, by feeding the sequence generated until now as the target ("next word prediction")
- ▶ Unlike RNNs/LSTMs, the next word generated directly depends on everything else (through attention and self-attention), not just by recurrent mechanisms



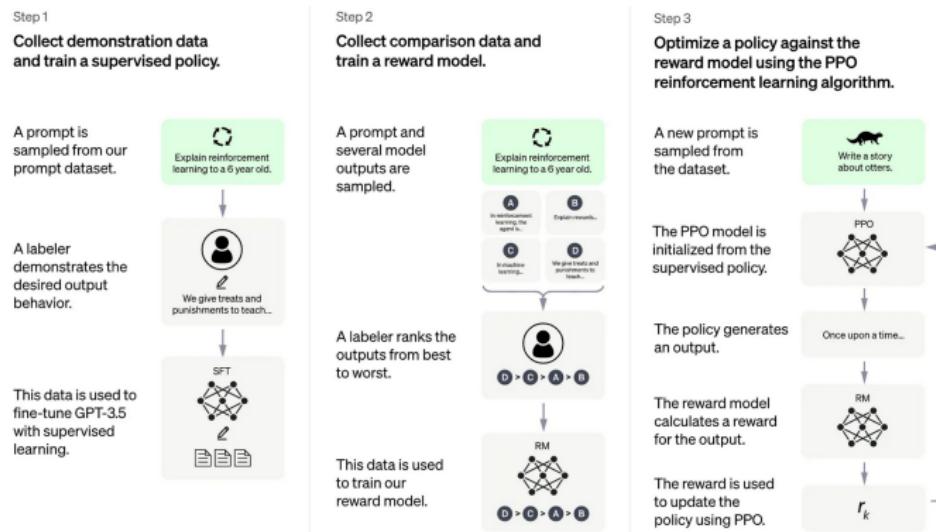
## Transformer : example of a (high-level) seq2seq architecture

- ▶ Familiar concepts : encoding/decoding, latent space... but with attention layers !



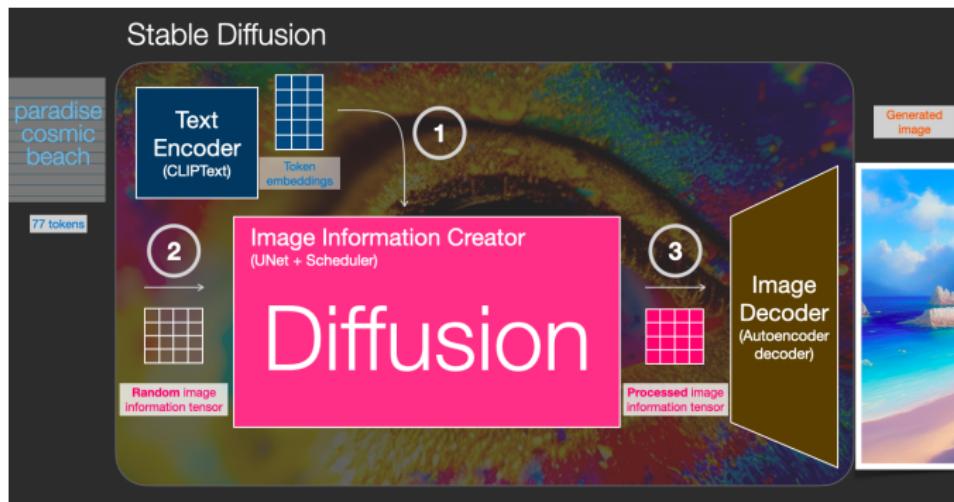
# Transformers + RL

- ▶ Combined with RL strategies (action/rewards), Transformers become incredible chatbots.
- ▶ ChatGPT is based on the GPT3 model (Generative Pre-trained Transformer), trained with complex RL strategies and **HUGE** amount of data.



## Transformers + diffusion models

- ▶ Transformers are great at computing **embeddings** for text, that can be used for other tasks. The BERT model is only an embedding model !
- ▶ These embeddings can be fed to other generative models. For instance, **all** the modern text-to-images models (Stable Diffusion, Midjourney, Imagen, DALL-E 2...) are based on **Transformers + diffusion models**.
- ▶ All these components are **pretrained** ! And then fine-tuned once put together.



## Table of Contents

### Generative Models

Generative Adversarial Networks

Variational Auto-encoder

Diffusion models

### Attention, Transformers

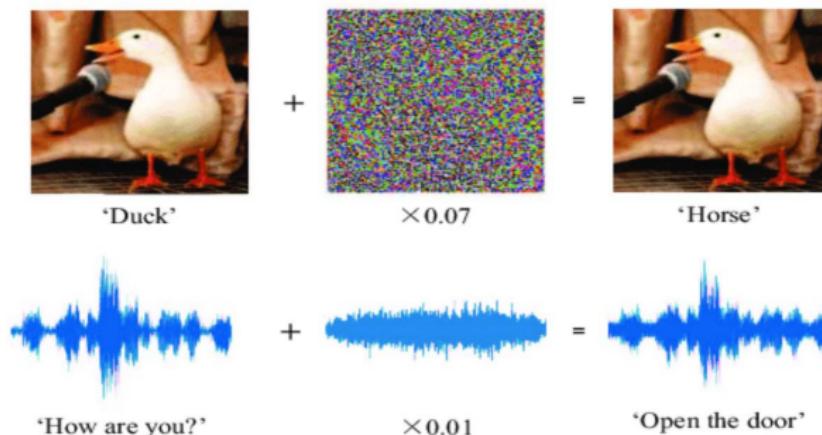
### Conclusions and open questions

## Conclusion

- ▶ Deep learning is still mysterious and frustrating for many reasons
- ▶ They work incredibly well in some situations...
- ▶ ...but don't kill flies with bazookas, and always remember the basic principles ! (which are useful in deep learning too !)
- ▶ Modern libraries make them easy to build, easy to train...
- ▶ ... but they are useful for non-deep stuff too !

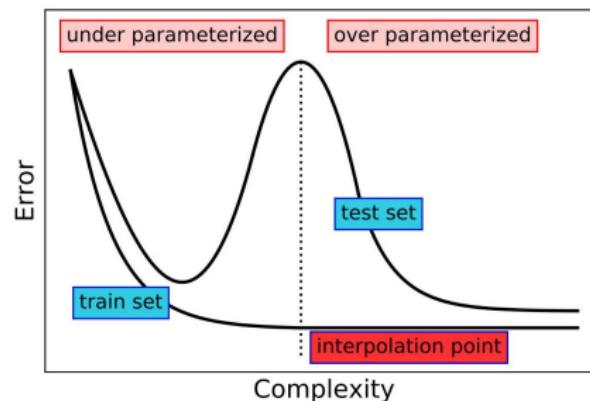
## Hot question : adversarial attack

Once trained, a deep NN usually can be fooled easily, by **designing** an attack specifically for this NN. This unstability seems to be an integral part of high-performance NNs !



## Hot question : generalization and double descent

- ▶ Modern Deep NN are **wildly overparametrized**, and **do interpolate the training data !!**
- ▶ But they still generatlize well !
- ▶ There is a **double descent phenomenon** : by increasing the complexity of the model past the interpolation point, among *all* interpolating models, it seems DNN+SGD favors models that **generalize well**. Still very much an open topic.



## Hot topic : privacy

- ▶ Since DNNs interpolate, and thus generally memorize, training data, **data privacy** is very much a problem
- ▶ Ensuring privacy seems to almost always come at the price of performance
- ▶ There is a link with stability and robustness as well, see eg  
[https://www.youtube.com/watch?v=3hig\\_oEz8Rg](https://www.youtube.com/watch?v=3hig_oEz8Rg)

## Hot topic : GNN, Geometric Deep Learning

- ▶ Graph Neural Networks (GNNs) work on **irregular data** that lives on network : molecules, proteins, social networks, computer networks, meshes, etc. They use “convolution” in this irregular space
- ▶ Most importantly, they are **permutation-invariant** : their output are left unchanged by a relabelling of nodes of the graph (graph isomorphism)
- ▶ They gave rise to **geometric deep learning**, which is the study of **symmetries and invariances** in DNNs : translation for CNN, permutation for GNN, but also rotation, gauge, etc. Much inspired by physics !

