Second School on Data Science and Machine Learning

Generative models

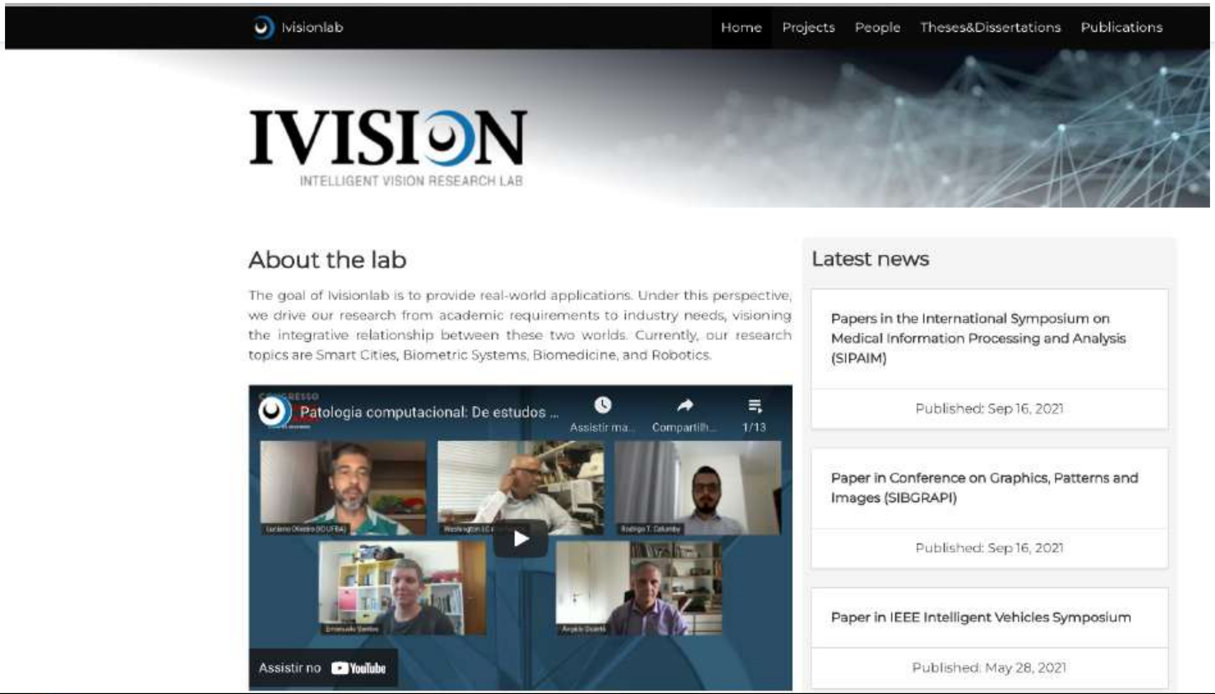
Luciano Oliveira

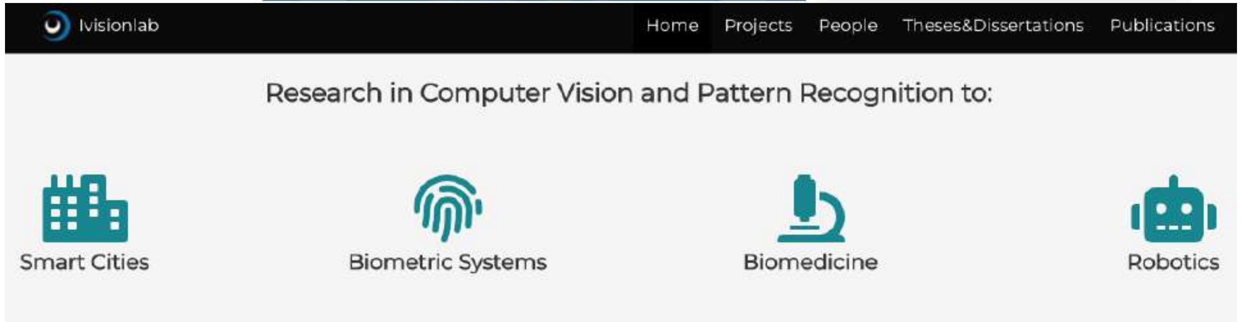
Universidade Federal da Bahia (UFBA)

Universidade Federal da Bahia

IVISION

INTELLIGENT VISION RESEARCH LAB





Agenda

- (1) Generative models at a glance
- (1) Mathematical foundations of generative models
- (1) Warming up with Latent Dirichlet Allocation
- Deep generative models
  - (2) Restricted Boltzmann Machines
  - (2) Deep Belief Networks
  - (3) Autoencoders
  - (3) Generative Adversarial Networks
  - (4) ChatGPT
- (4) Conclusions

(n), where n is one of the 4 parts of the course today

Summary

Generative models at a glance

Mathematical foundations of generative models

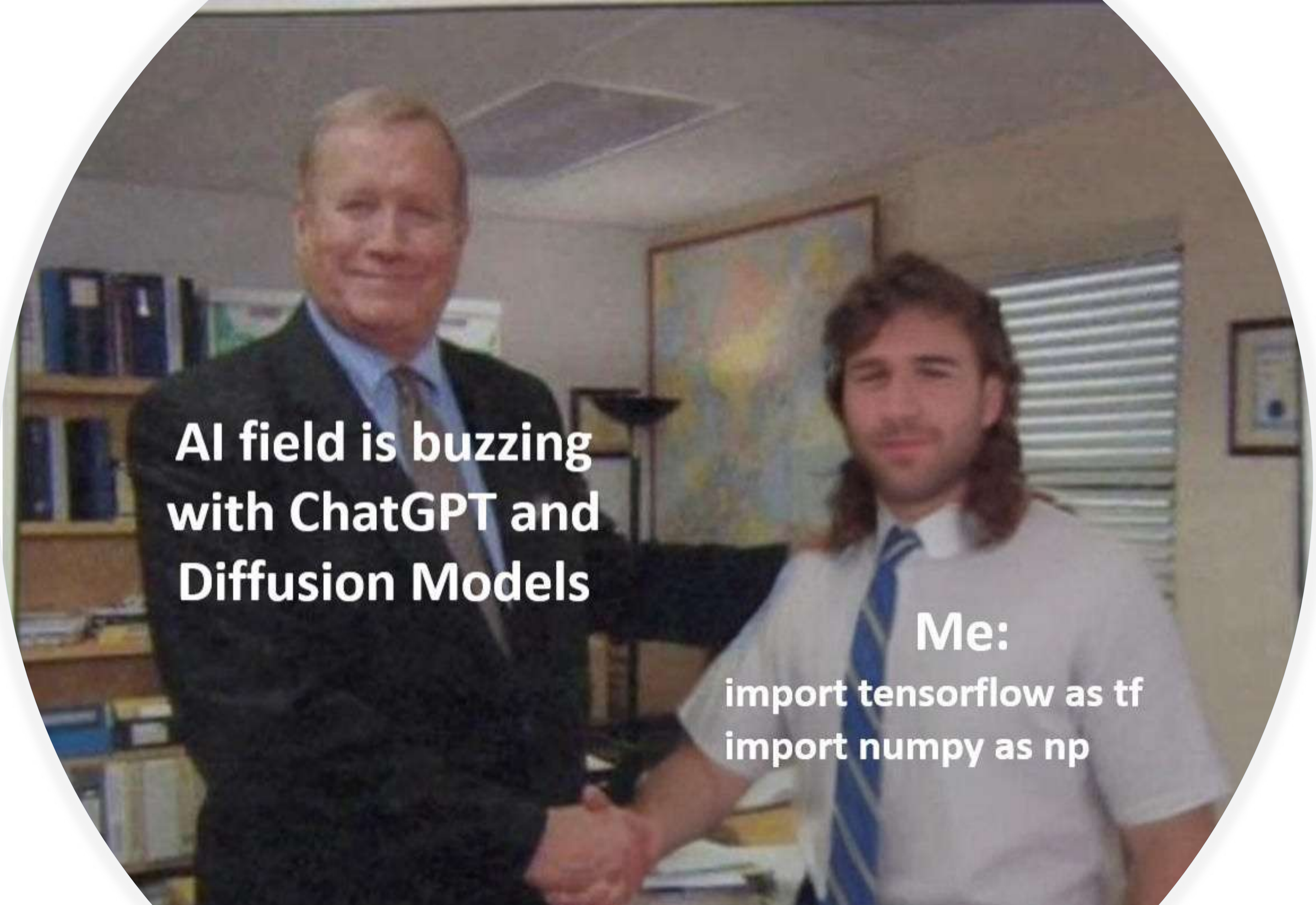
Warming up with LDA

Deep generative models

Conclusions

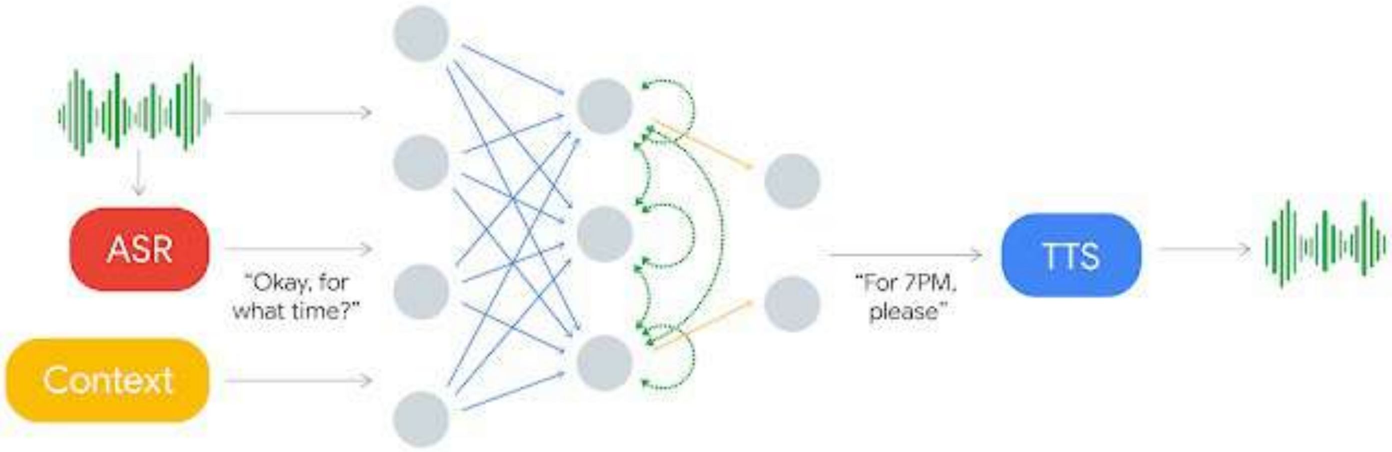
- Modelling the probability distribution of a generative model is not an easy task, while requiring:
  - large computational resources
  - a lot of patience to efficiently modelling the generative side
- Understanding the fundamentals of each tecnique is of underlying importance to make it work, but not only.. **It is necessary a lot of patience.**

Generative models at a glance

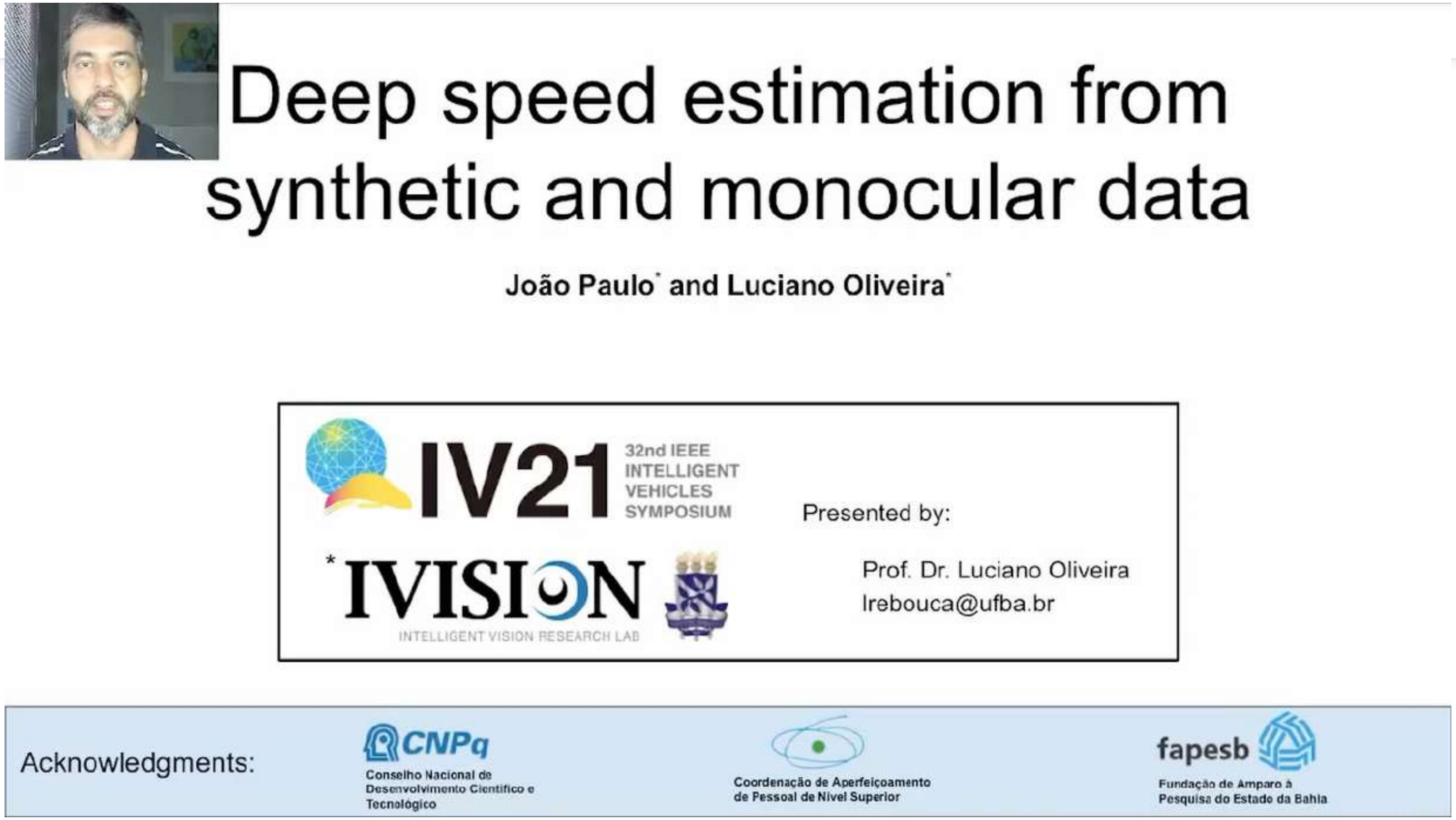

AI field is buzzing
with ChatGPT and
Diffusion Models
Me:
import tensorflow as tf
import numpy as np

Google Duplex

- Natural chating with humans
  - Conducting natural conversations
  - Fully autonomous
  - Synchronization
  - Interrupt control
- Recurrent networks



Voice cloning



Automatic staning

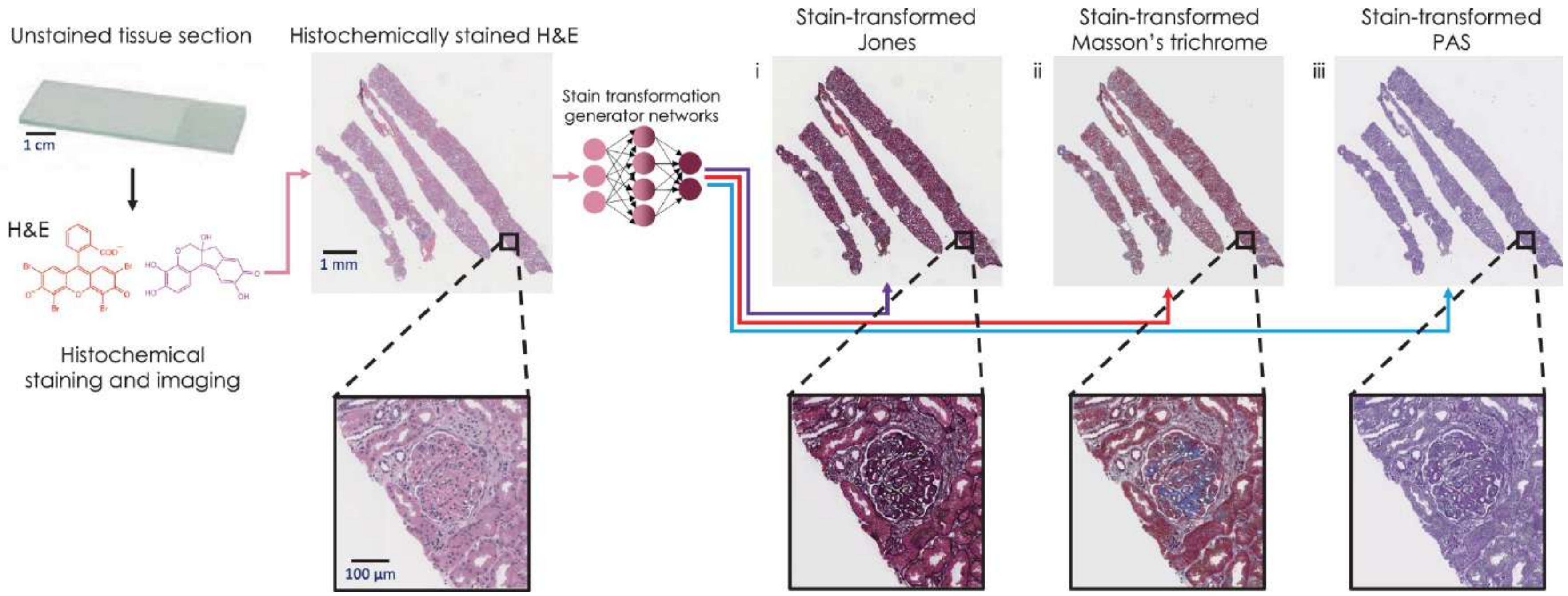
Article | Open access | Published: 12 August 2021

Deep learning-based transformation of H&E stained tissues into special stains

Kevin de Haan, Yijie Zhang, Jonathan E. Zuckerman, Tairan Liu, Anthony E. Sisk, Miguel F. P. Diaz, Kuang-Yu Jen, Alexander Nobori, Sofia Liou, Sarah Zhang, Rana Riahi, Yair Rivenson, W. Dean Wallace & Aydogan Ozcan

Nature Communications **12**, Article number: 4884 (2021) | Cite this article

- Stanning of slide windows from H&E
- Use of GANs to generate slides with different staing patterns



Leonardo.ai

A scene in a 3D disney/pixar style containing a york shire dog dressing a red and black shirt

Generative models

Definition: Machine learning models that learn to generate new data samples similar to the training data

Challenges for generative models

Complex data:

- which requires very large models to capture all nuances of the features and distributions

Models:

- which require powerful processing resources

- are difficulty to assess performance

- require complex control to generate data diversity

Mathematical foundations of generative models

Probability and machine learning

Probability rules

P() = 1/6 = 16,67% → 

$$\begin{cases} 0 \leq P(A) \leq 1 \\ P(\emptyset) = 0 \\ P(\Omega) = 1 \end{cases}$$

… in the context of machine learning

Labels: Y=y

Features: $X=\{x_1, x_2, \ldots x_n\}$

$P(Y, X) = P(y, x_1, x_2 \ldots x_n)$

P(Y|X= ) = ? or P(X=  |Y) = ?

P(Y|X= ) = ? P(X=  |Y) = ?

Bayes theorem

$$P(Y|X) = \frac{\overbrace{P(Y)}^{\text{prior}}\,\overbrace{P(X|Y)}^{\text{likelihood}}}{\underbrace{P(X)}_{\text{marginal}}} = \frac{P(X,Y)}{P(X)}$$

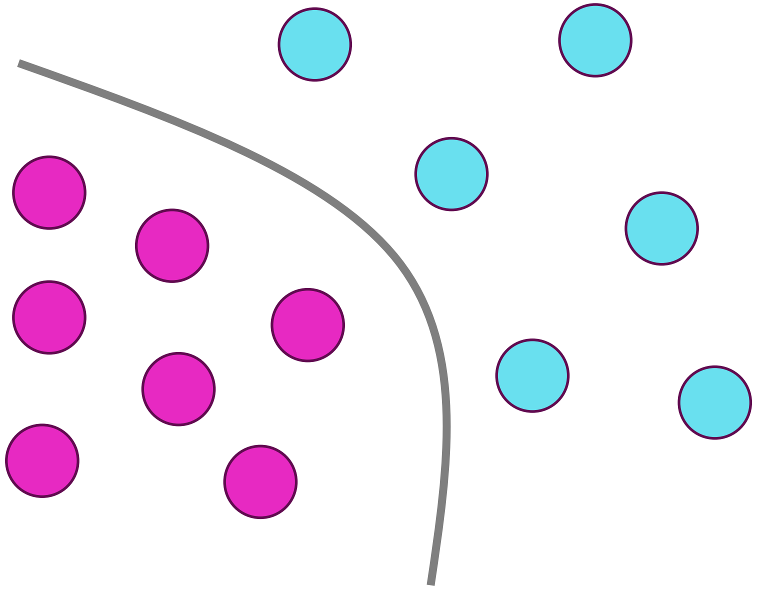
$$\text{posterior} = \frac{\text{prior x likelihood}}{\text{evidence}}$$

Bayes theorem

$$P(y \mid \underbrace{x_1, x_2, x_3, \ldots}_{\text{features}}) = \frac{P(y)P(x_1|y)P(x_2|y)P(x_3|y)\ldots}{P(x_1)P(x_2)P(x_3)\ldots}$$

labels

Discriminative models

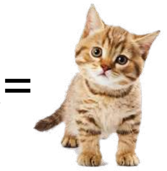


f: separation
hyperplane

f: $X \rightarrow Y$ or $P(Y \mid X)$ or

P(label | features)

P(Y|X= )

P(Y|X= )

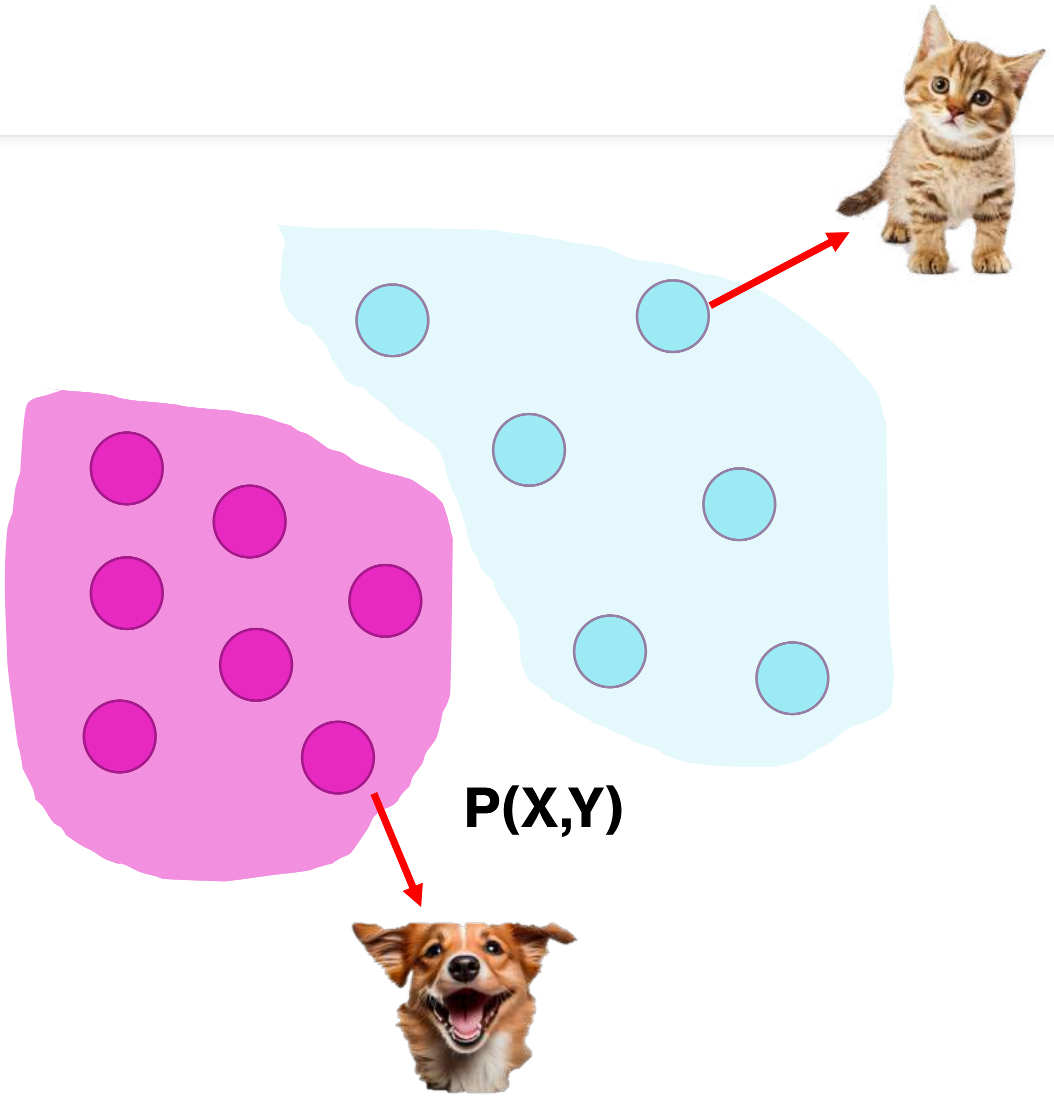
Posterior is learnt or modeled from data (*conditional models*):

- Logistic regression
- Redes neurais
- SVM
- CRFs
- Random forest

Generative models

- Rewrite Bayes:
  - **P(X|Y=y) = P(X,Y) / P(Y)**
- We use P(X,Y) *to sample* new data:
  - *Tuples* (X,Y)
  - *Inputs* using Y
- Examples:
  - Naïve Bayes
  - GMM
  - HMM
  - VAE
  - GAN

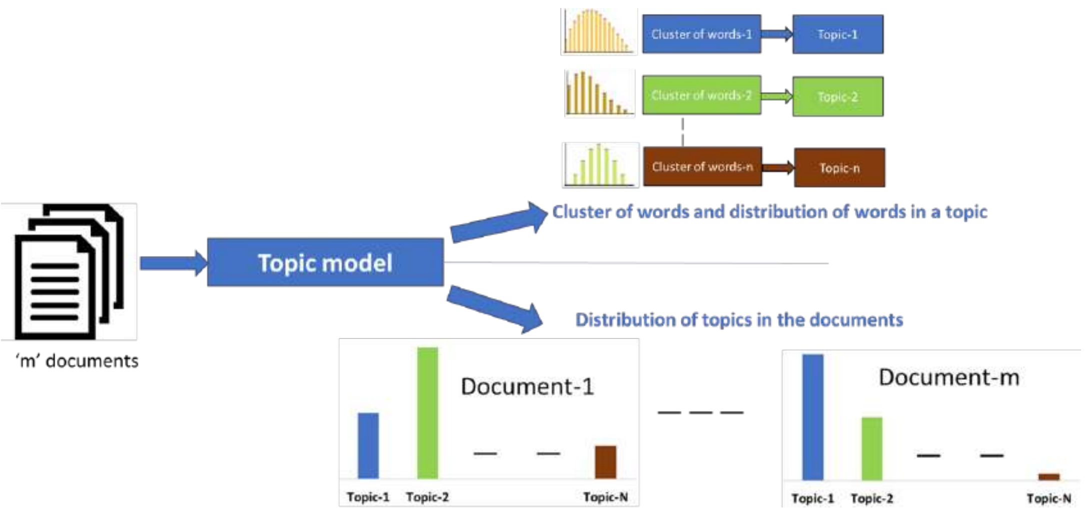
Depend on a latent variable

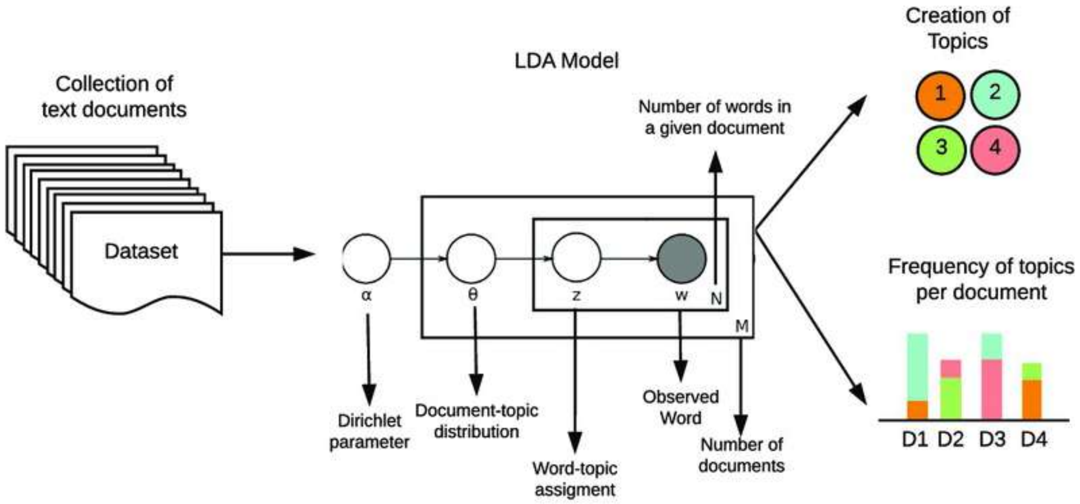


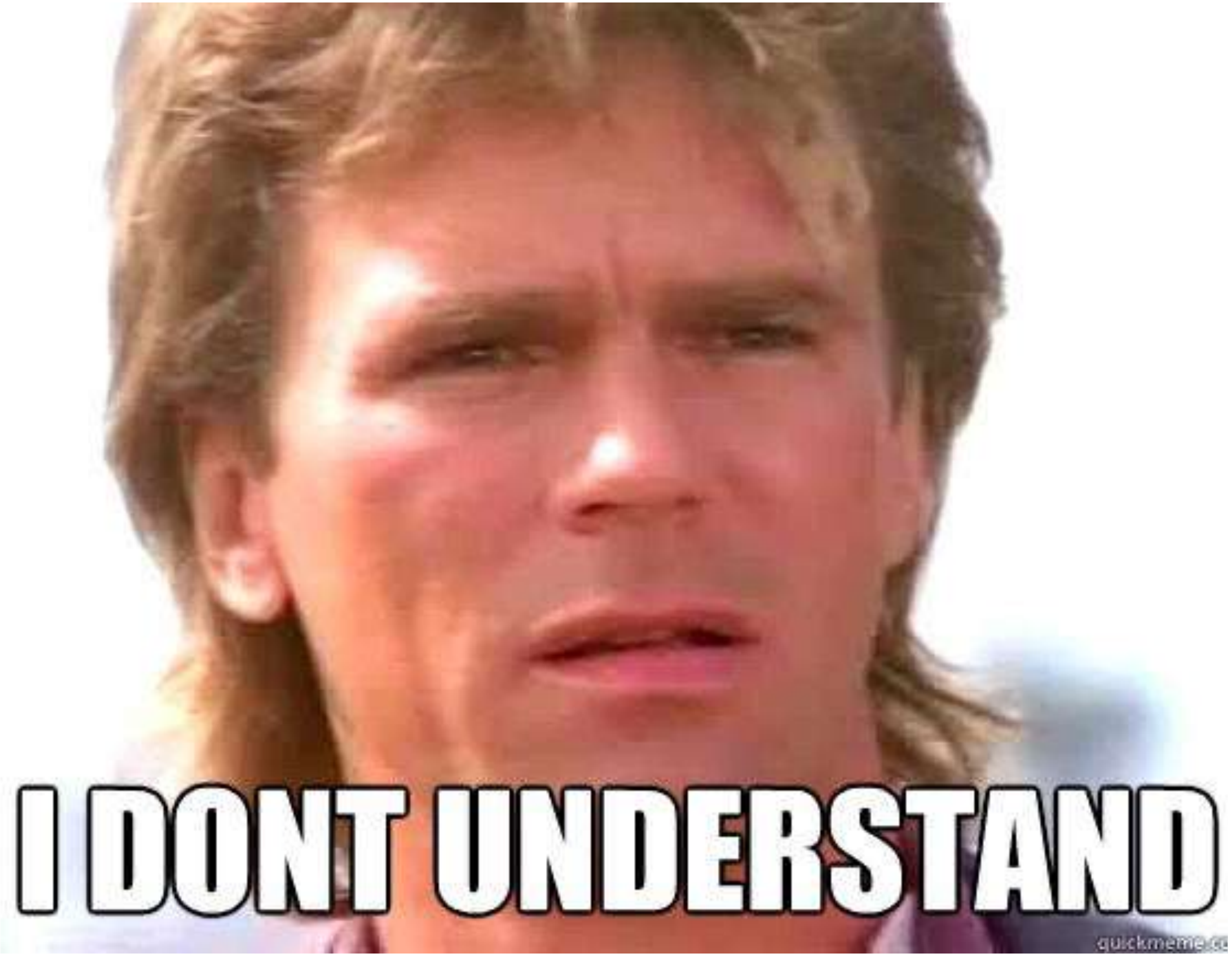
Warming up with LDA

Latent Dirichlet Allocation

- Method for Unsupervised Topic Modeling
- Bayesian network based on Dirichlet distribution:
  - Observable Variables: Words
  - Latent variables: topics
- Goals are:
  - Discovering topics in a corpus of words
  - The proportion of these topics in each document
- Assuming that:
  - Each document is a mixture of latent topics
  - Each topic is a distribution over words






I DONT UNDERSTAND
quickmeme.co

Latent Dirichlet Allocation

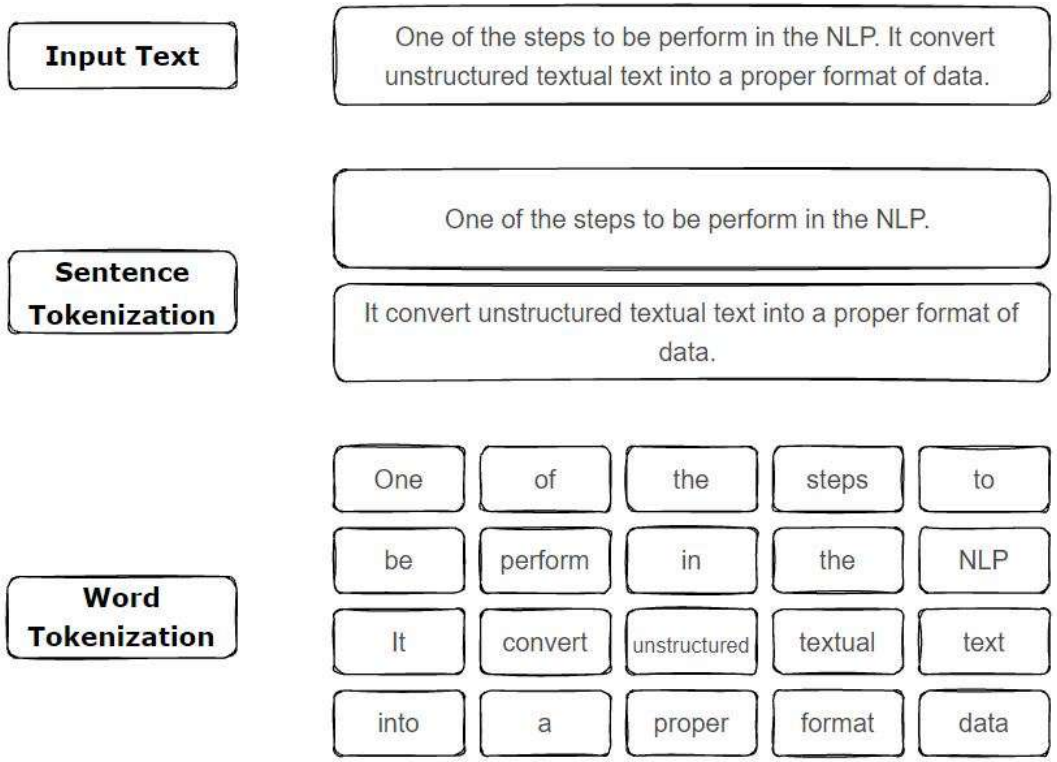
- Consider that we have 5 documents
  - Each containing the words on the right side
- We have to figure out the words in different topics, with their respective probabilities
- Each document is a bag of words:
  - Order of the words and grammar rules are NOT important
  - We have to do some pre-processing
- We have to know beforehand how many (T) topics we want

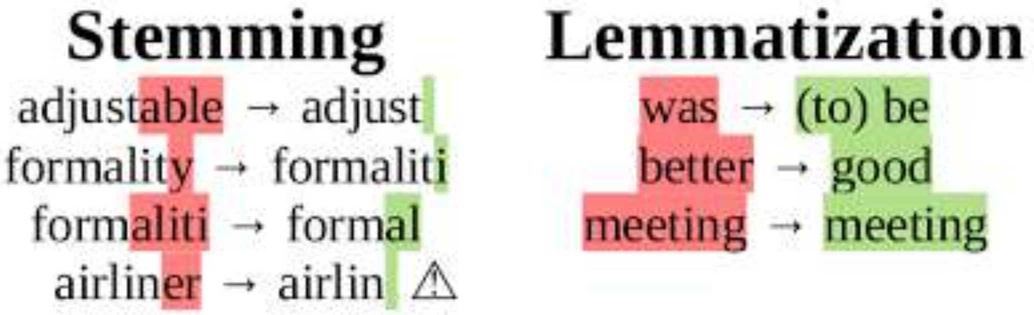
```
Doc1: word1, word3, word5, word45, word11, word 62, word88 ...
Doc2: word9, word77, word31, word58, word83, word 92, word49 ...
Doc3: word44, word18, word52, word36, word64, word 11, word20 ...
Doc4: word85, word62, word19, word4, word30, word 94, word67 ...
Doc5: word19, word53, word74, word79, word45, word 39, word54 ...
```

	Word1	word2	word3	word4	.....
Topic1	0.01	0.23	0.19	0.03	
Topic2	0.21	0.07	0.48	0.02	
Topic3	0.53	0.01	0.17	0.04	

Latent Dirichlet Allocation

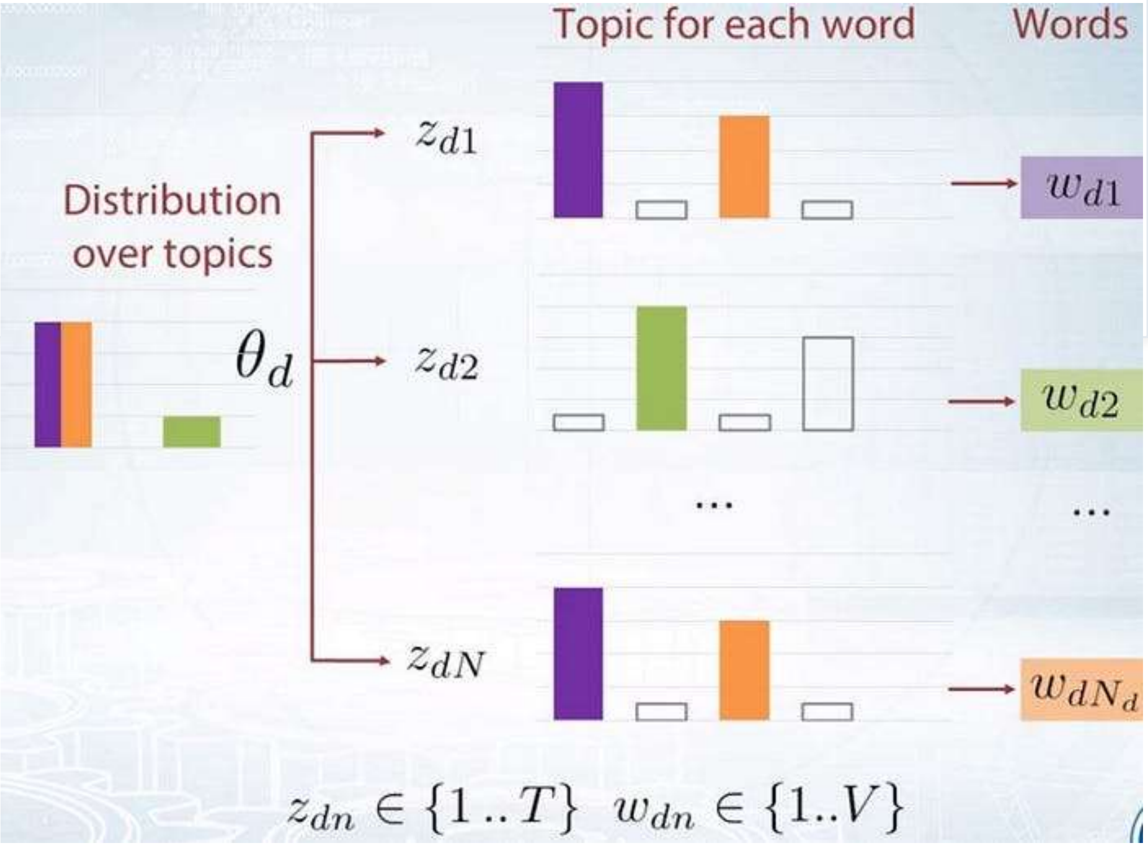
- Pre-processing
  - **Tokenization**
  - **Lemmatization**: Words in the third person are changed to the first person, and verbs in the past and future are changed to the present.
  - **Stemming**: Words are reduced to their root form.

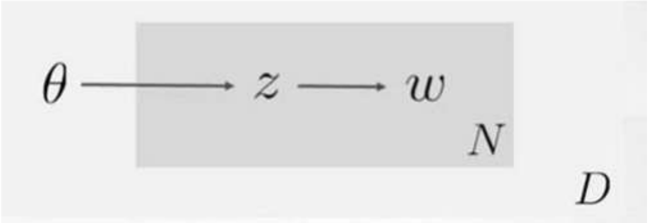




Latent Dirichlet Allocation

- Distribution of:
  - documents over topics – $p(z_{dn} \mid \theta_d)$
  - topics over words – $p(w_{dn} \mid z_{dn})$
- Notation:
  - **D**: total number of documents
  - **θ_d**: distribution for the d-th document.
  - **T**: total number of topics
  - **Z_{d1}**: probability distribution of words over the first topic
  - **W_{dn}**: Probability of a word in the distribution of the n-th topic
  - **V**: Total number of words in a corpus.





$$p(W, Z, \Theta) = \prod_{d=1}^{D} p(\theta_d) \prod_{n=1}^{N_d} p(z_{dn} \mid \theta_d) p(w_{dn} \mid z_{dn})$$

Latent Dirichlet Allocation

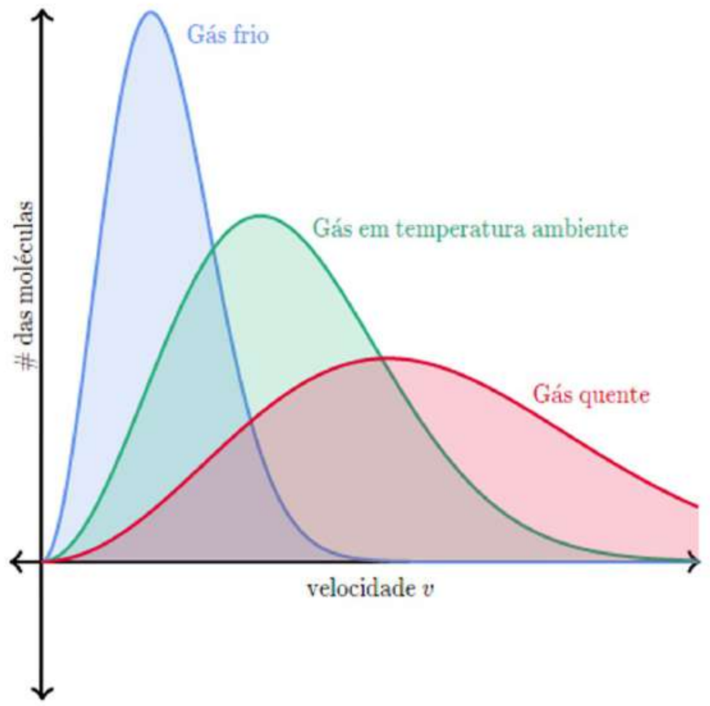
- The generative process works as follows:
  - For each word W in the document, do:
    - choose a topic Z_d of the distribution θ_d
    - choose a word W from a topic Z_d
- Assume that:
  - each document is generated regardless of the others
  - the same set of topics is used for all documents
- LDA allows to generate latent variables (topics) that ultimately generate the observable data (words)

Deep generative models

Boltzmann Machine

- Boltzmann established the concepts of statistical physics.
- The Boltzmann distribution describes the probability of a gas molecule being found in a particular energy state.

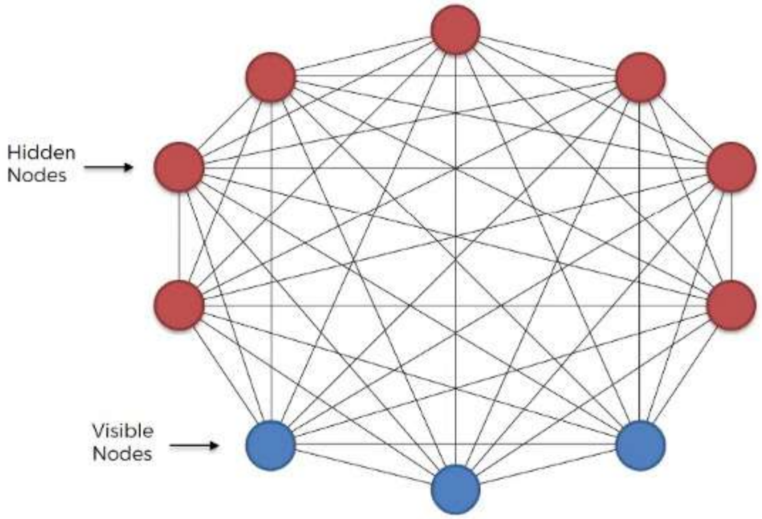
$$\frac{N_i}{N} = \frac{g_i e^{-\frac{E_i}{KT}}}{Z(T)}$$



Onde:

K: Boltzmann constant
T: temperature
g_i: number of states having energy **E_i**
N: total number of particles
Z(T): $\sum_i \; g_i e^{-\frac{E_i}{KT}}$

Boltzmann Machine



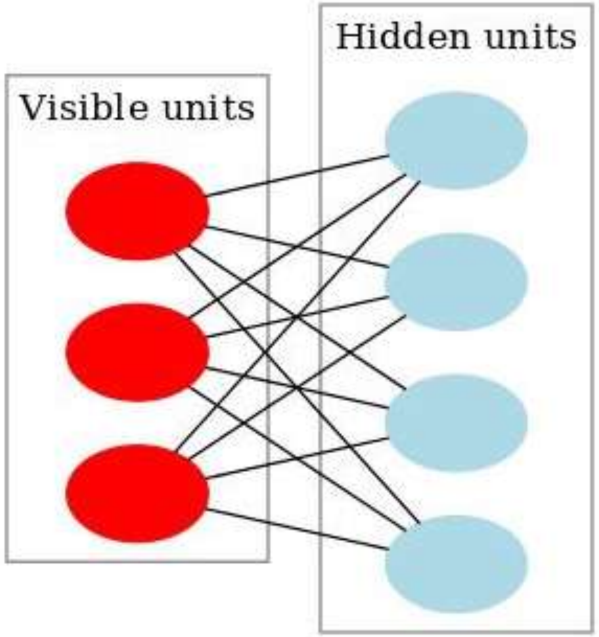
- They are probabilistic, unsupervised models based on energy.
- For each configuration of the system, a value of energy is assigned along with an associated probability:
  - **Low energy** ↓ represents **high probability** ↑
  - **High energy** ↑ represents **low probability** ↓

- By sampling, each neuron is either activated or deactivated with a certain probability.
- After training, convergence is achieved to a stable state, represented by the minimum of the energy function.
- The **hidden** neurons work to learn the latent states of the joint distribution function **P(v, h)**

Restricted Boltzmann Machine

Smolensky (1986) proposes the Restricted Boltzmann Machine under the name **Harmonium**

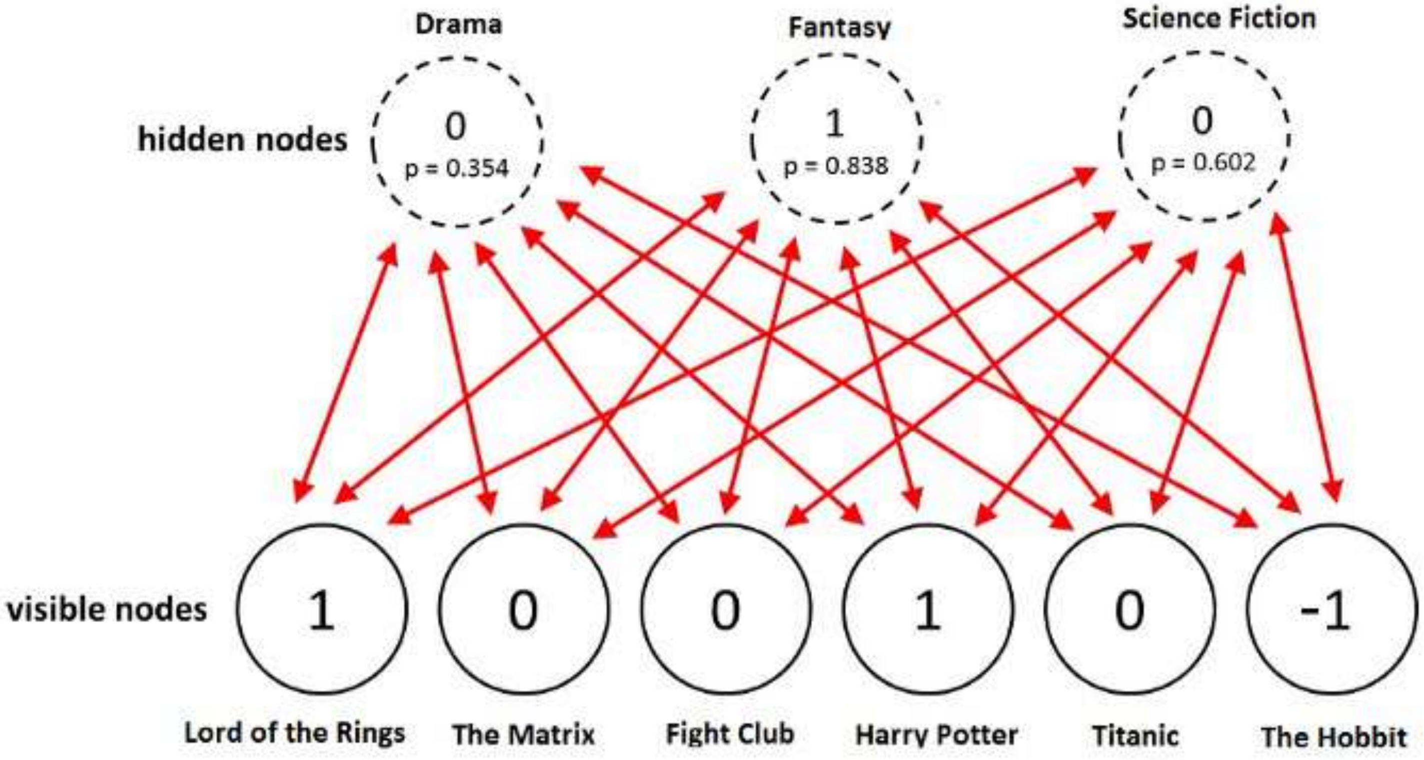
Hinton (2002) proposes a learning model



Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Colorado Univ at Boulder Dept of Computer Science.

Hinton, G.E. (2002). Training products of experts by minimizing contrastive divergence. Neural computation, 14(8), 1771-1800.

Restricted Boltzmann Machine



Restricted Boltzmann Machine

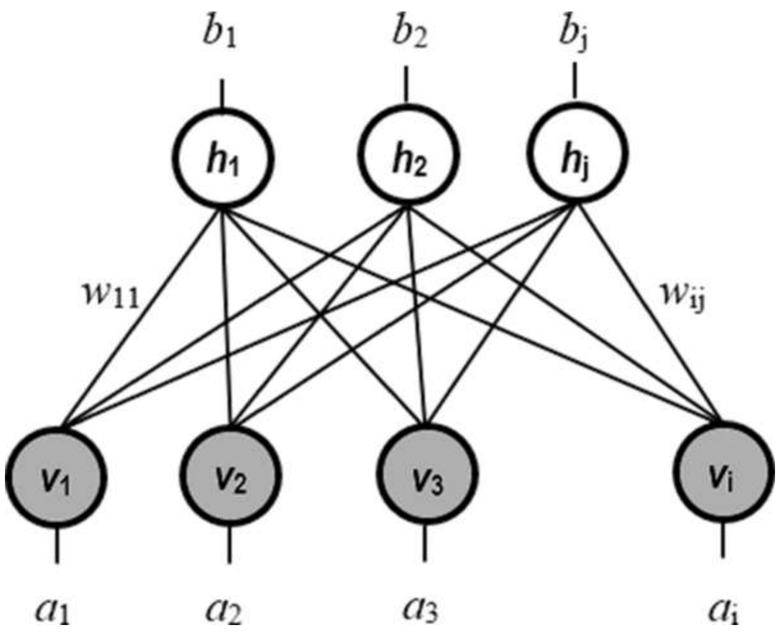
Total energy:

$$E(v,h) = -\sum_{j=1}^{n_h} b_j b_h - \sum_{i=1}^{n_v} a_i v_i - \sum_{i=1}^{n_v}\sum_{j=1}^{n_h} h_j w_{ji} v_i$$

Probability distribution

$$P(v,h) = \frac{e^{-E(v,h)}}{Z}$$

An RBM is totally specified by W, b, a



Restricted Boltzmann Machine

Training:

1. Input a vector **v**
2. Compute **h**
3. Use **h** to generate (samples of) visible states **v'**
4. Use **v'** to generate (samples of) hidden states **h'**
5. Update the parameters **W**, **b** and **c** (where ∊ is the LR):

Gibbs sampling: Sampling unknown parameters from a distribution while fixing the others.

$$\Delta \boldsymbol{W} = \epsilon\ (\boldsymbol{vh} - \boldsymbol{v'h'}) \qquad \Delta \boldsymbol{b} = \epsilon\ (\boldsymbol{v} - \boldsymbol{v'}) \qquad \Delta \boldsymbol{a} = \epsilon\ (\boldsymbol{h} - \boldsymbol{h'})$$

Restricted Boltzmann Machine

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \dots, V_m, H_1, \dots, H_n)$, training batch S

Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \dots, n$, $j = 1, \dots, m$

init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \dots, n$, $j = 1, \dots, m$

forall the $\boldsymbol{v} \in S$ **do**

  $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$

  for $t = 0, \dots, k-1$ **do**

    for $i = 1, \dots, n$ **do** sample $h_i^{(t)} \sim p(h_i \,|\, \boldsymbol{v}^{(t)})$

    for $j = 1, \dots, m$ **do** sample $v_j^{(t+1)} \sim p(v_j \,|\, \boldsymbol{h}^{(t)})$

  for $i = 1, \dots, n$, $j = 1, \dots, m$ **do**

    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)}) \cdot v_j^{(k)}$

    $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$

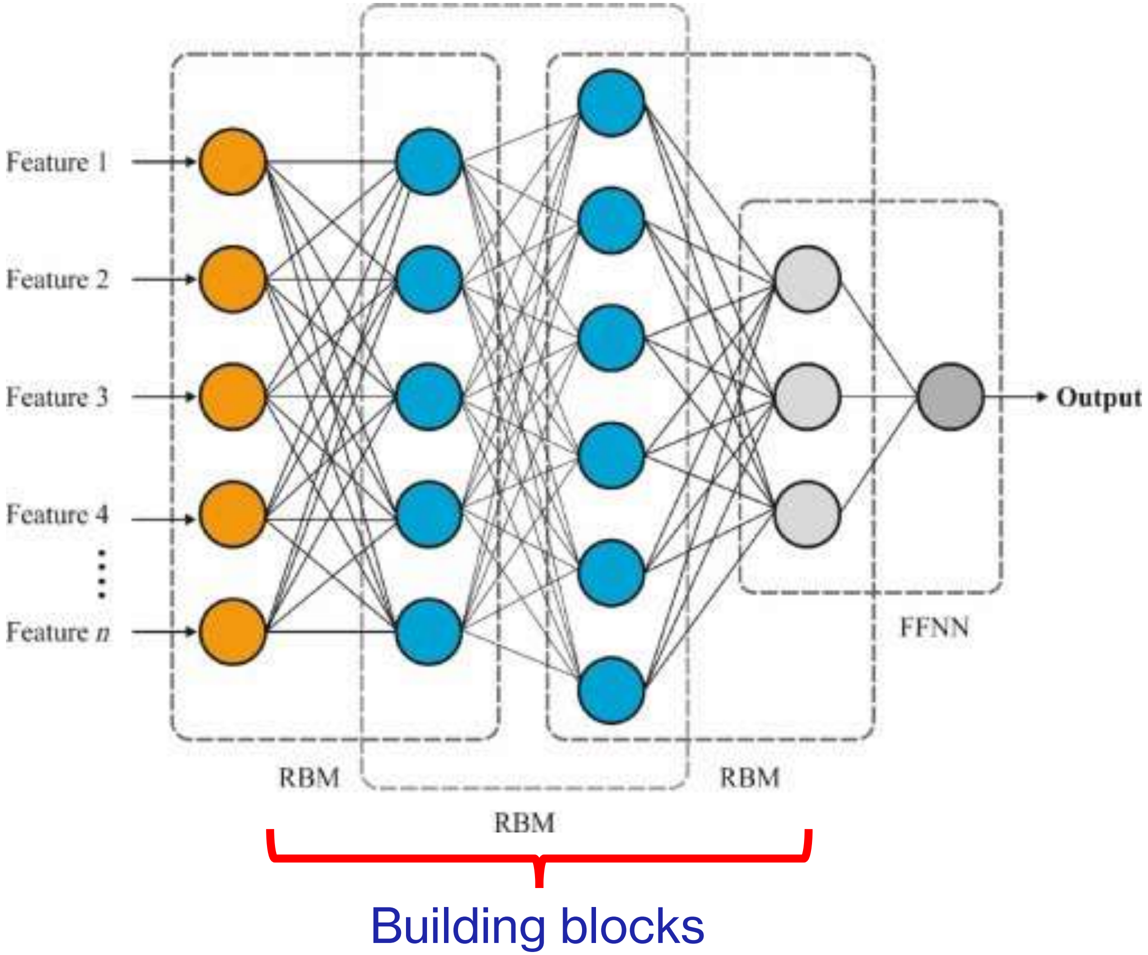
    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \,|\, \boldsymbol{v}^{(0)}) - p(H_i = 1 \,|\, \boldsymbol{v}^{(k)})$

Alternating step of Gibbs Sampling

Kulback-Leibler divergence

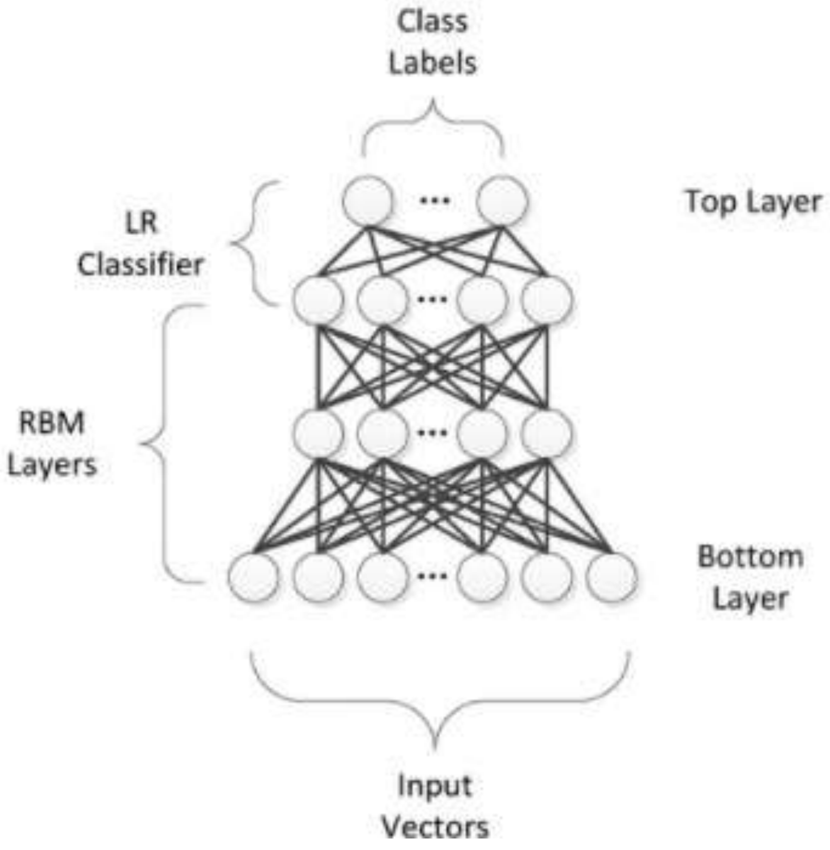
Deep Belief Network

- **Stack of RBMs**: The output of one RBM is taken as input by another RBM
- It is possible to add as many RBMs as you want; however, this can cause:
  - Vanishing gradient
  - Local minima
- **DAG**: directed acyclic graph
- Supervised and unsupervised
- **Training**: Greedy learning algorithm (*layer-by-layer pre-training*)



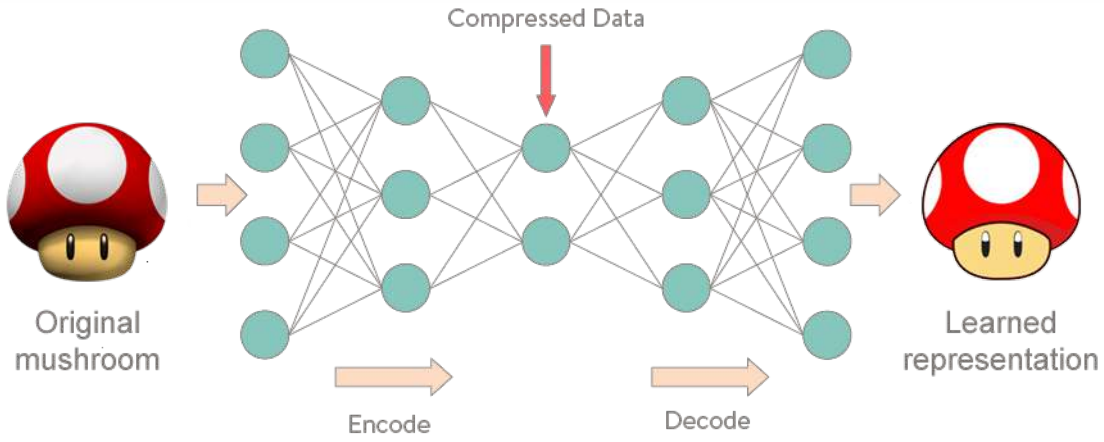
Deep Belief Network

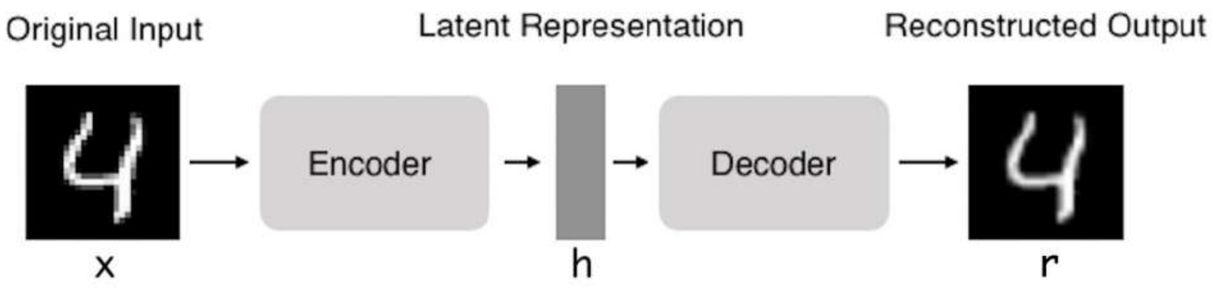
- Pre-processing:
  - Image converted to gray scale
  - Pixel normalization
  - Image resize to a standard size
- Training:
  - **DBN** training using unsupervised learning, layer by layer
  - Use of an **RBM** to train the first layer
  - Using the outputs of the previous layer as inputs for the subsequent layer after pre-training.
- Fine-tunning:
  - Adjust of the **TOP layer of the DBN**, using supervised learning
  - Updating the network weights based on labeled training data using **backpropagation** and **gradient descent.**



Autoencoders

- **NOT** a generative model
- **NOT** supervised
- They are used to learn representations in a latent feature space (**bottleneck**).
- Input is an image, output is the same image.
- **Encoder**: h = f(x); **decoder**: r = g(h)
- Generate **x'** similar to **x**
- **h** has an usefull property**:**
  - **h is incomplete and compressed** – force **h** to capture generic features
- Parameters: Latent neurons, encoder and decoder layers, nodes per layer, loss.

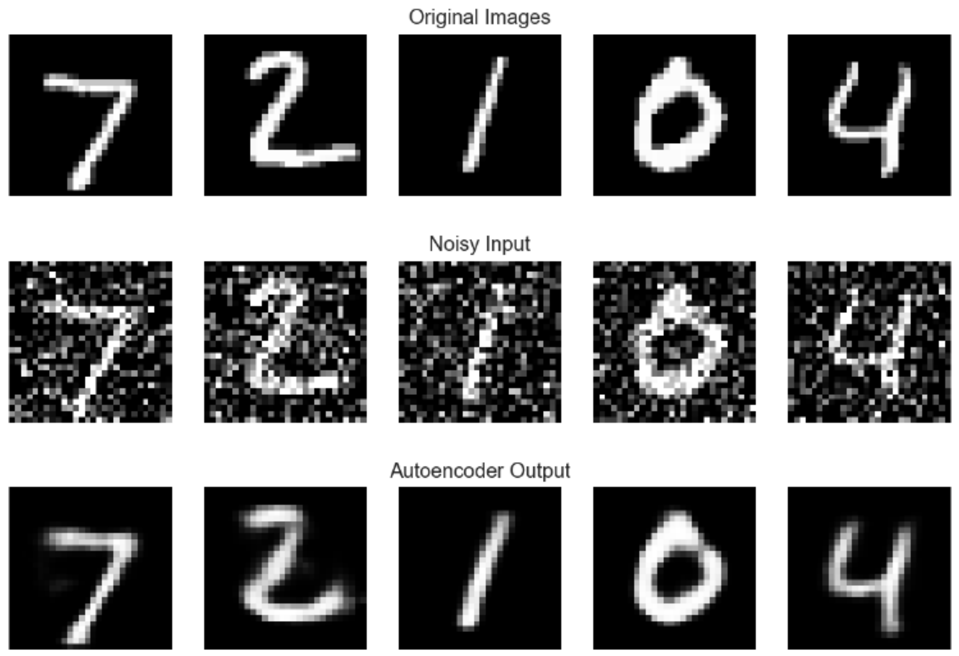




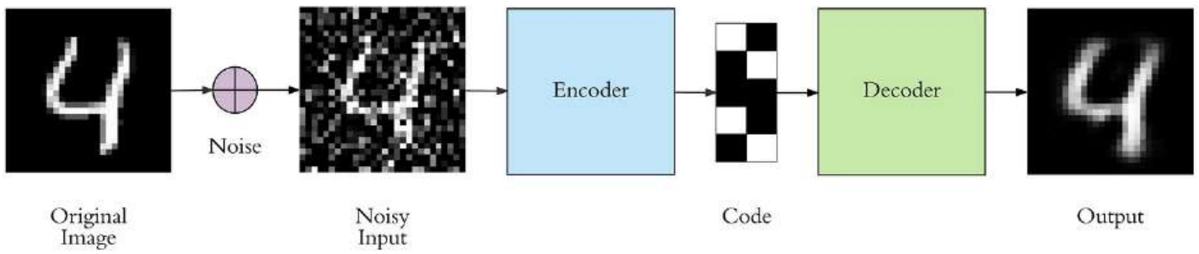
Trained by backpropagation

Autoencoders

Autoencoders can also learn to do denoysing

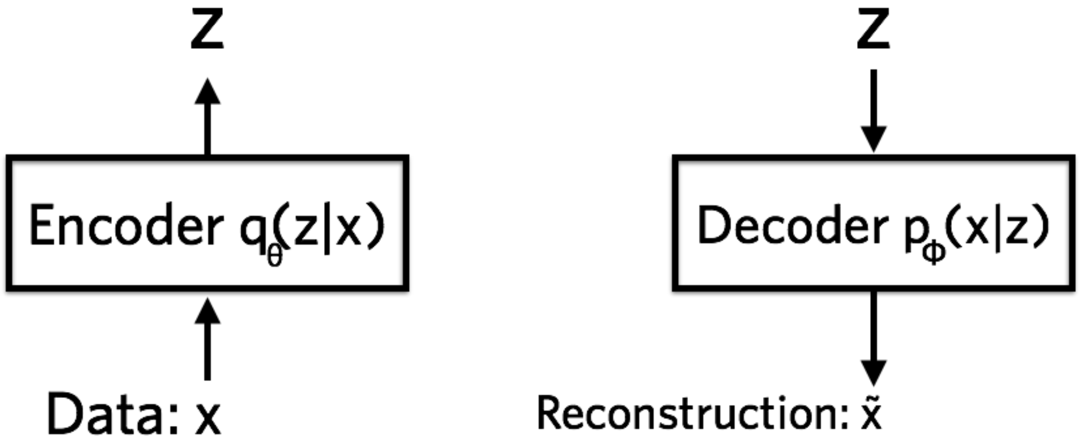


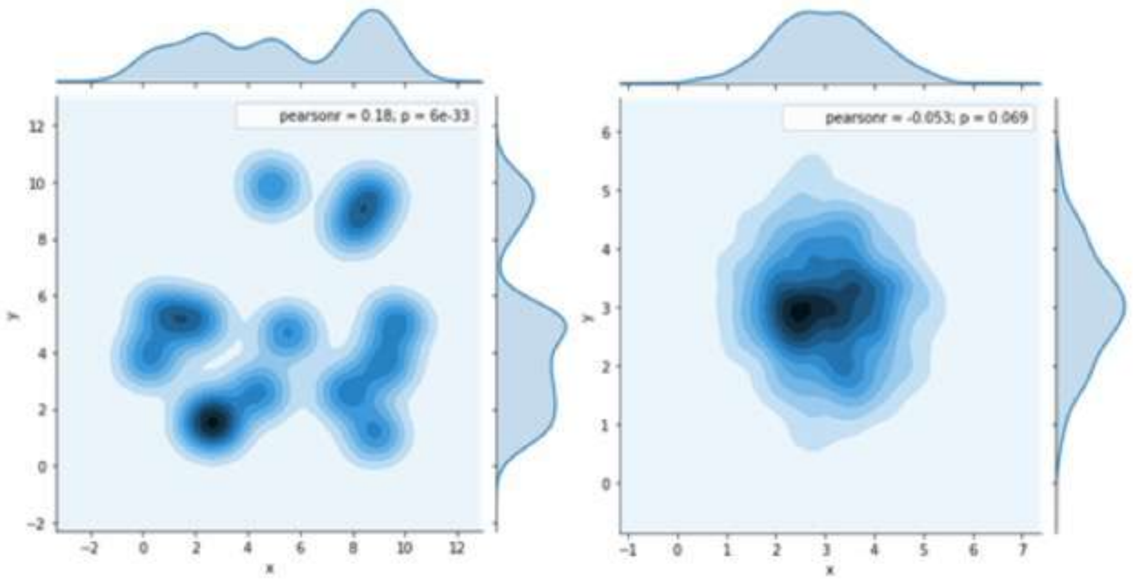
But why learn autoencoders in this course?



Variational autoencoders

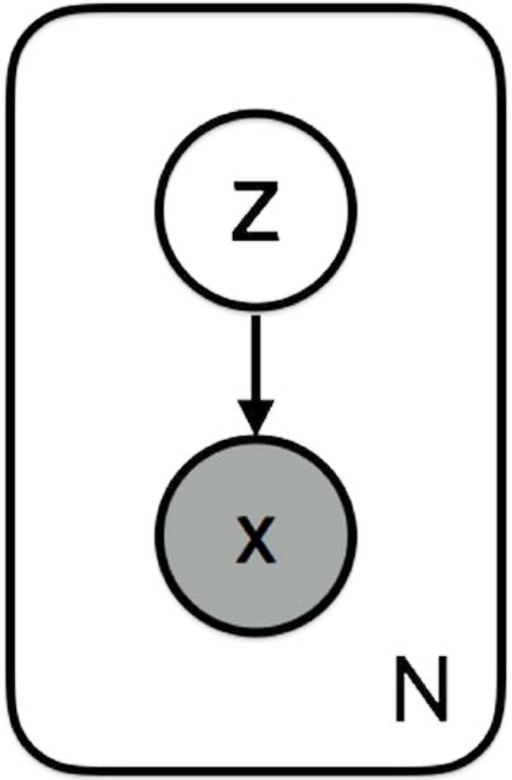
- Generative model, unsupervised, autoencoder similar architecture
- In the **bottleneck**, VAE learns a **posterior**
  - Latent space is stochastic; a Gaussian PDF
  - Sample from **q** to find **z.**
- **Decoder** has weights and biases, whose output allows for data generation
  - It takes the distribution of z, and the output is the parameters of a Gaussian or Bernoulli distribution (if the input is binary) – output between 0 and 1 for each pixel.
  - The loss is comprised of: **log p_ϕ(x | z)** of the reconstruction from **z** and the **KL divergence** between **q** and **p(z)**, where **p** is a Gaussian distribution with zero mean and variance equal to 1





Variational autoencoders

- VAEs learn: $p(x, z) = p(x \mid z)p(z)$
- For each sample, **i**, in the dataset:
  - Find latent variables: $\mathbf{z_i} \sim \mathbf{p(z)}$
  - Find $\mathbf{x_i} \sim \mathbf{p(x \mid z)}$
- The latent variables are found from **p(z)**
- **Model inference will be:**



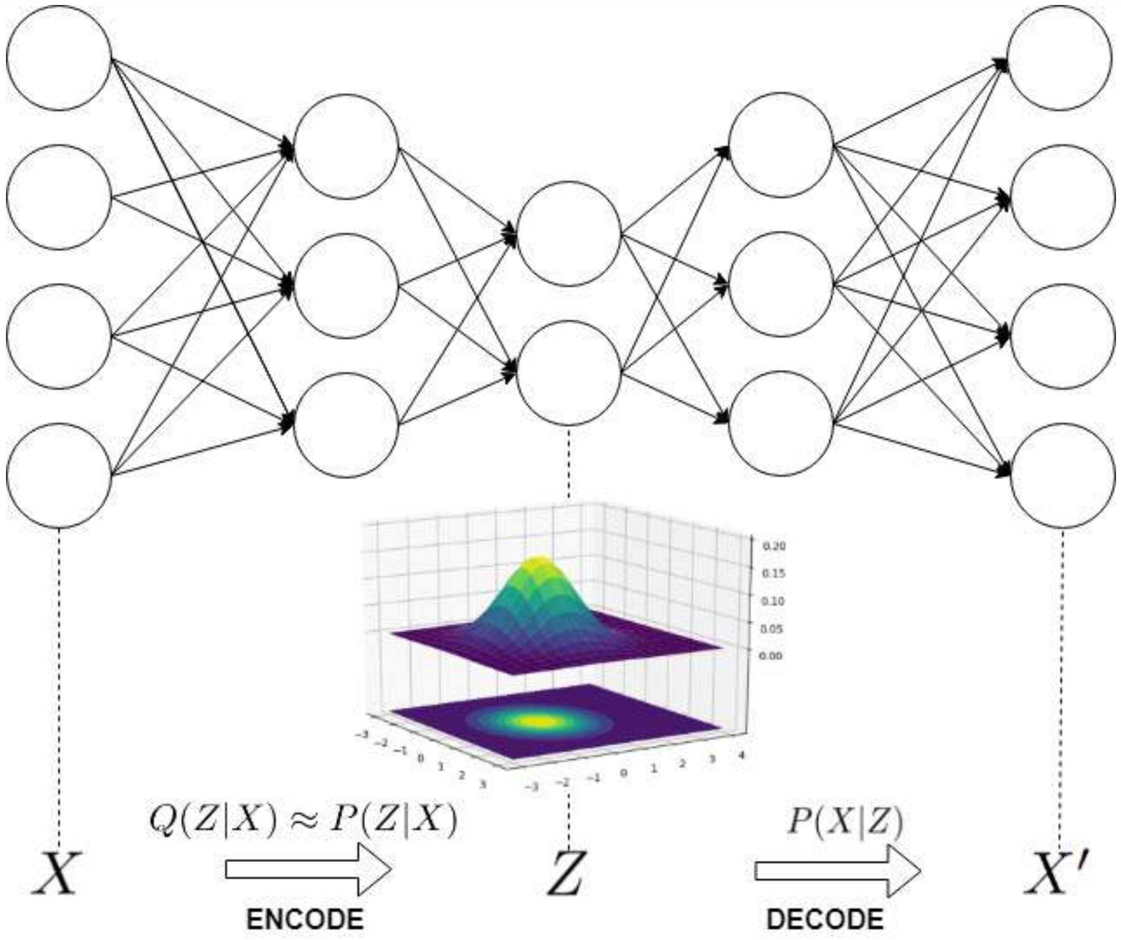
$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$, where: $p(x) = \int p(x \mid z)p(z)dz$

Variational autoencoders

- Reparametrization trick:

$$S_i \sim \mathcal{N}(0,1), i \in 0, \ldots, n$$
$$Z_{sampled,i} = \mu_i + (S_i \odot \sigma_i), i \in 0, \ldots, n$$

- Sampling from mean and standard deviation vector, instead of from the latente variables

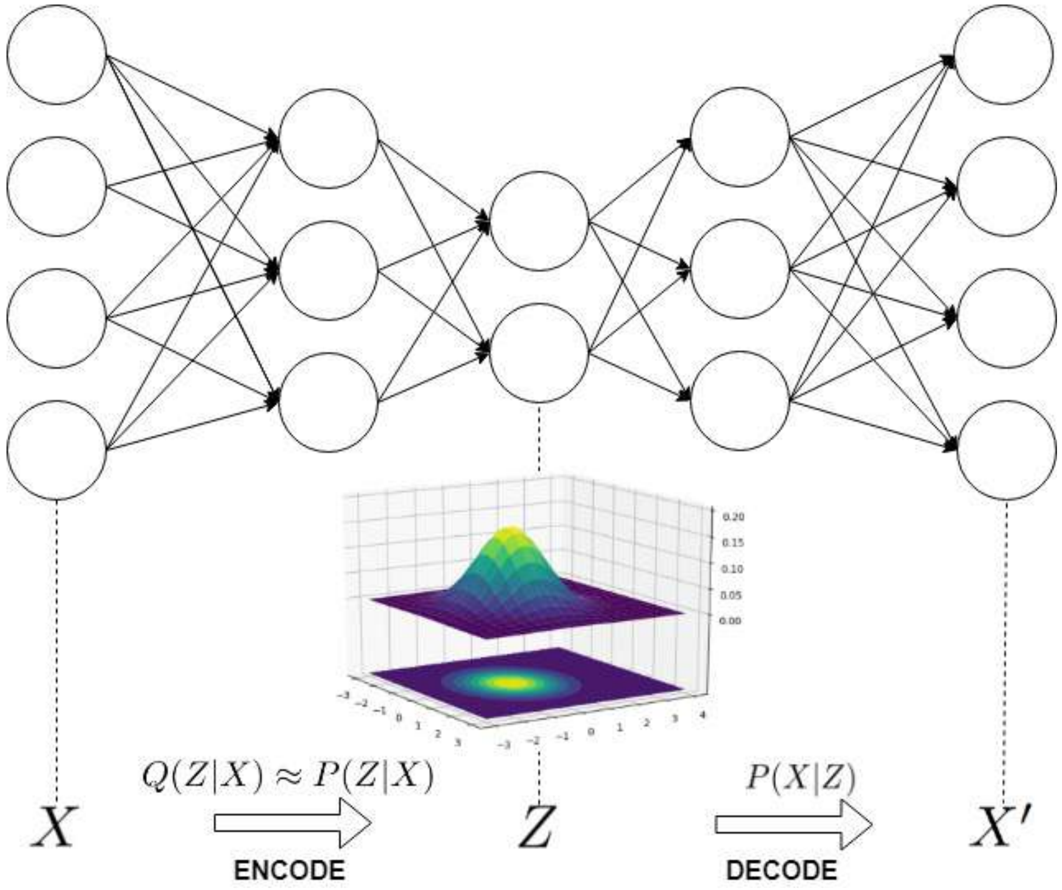


Variational autoencoders

- As p(x) is costly, the posterior is approximated to a family of distributions λ : **q_λ(z|x)**
  - For example, if **q** is Gaussian, so

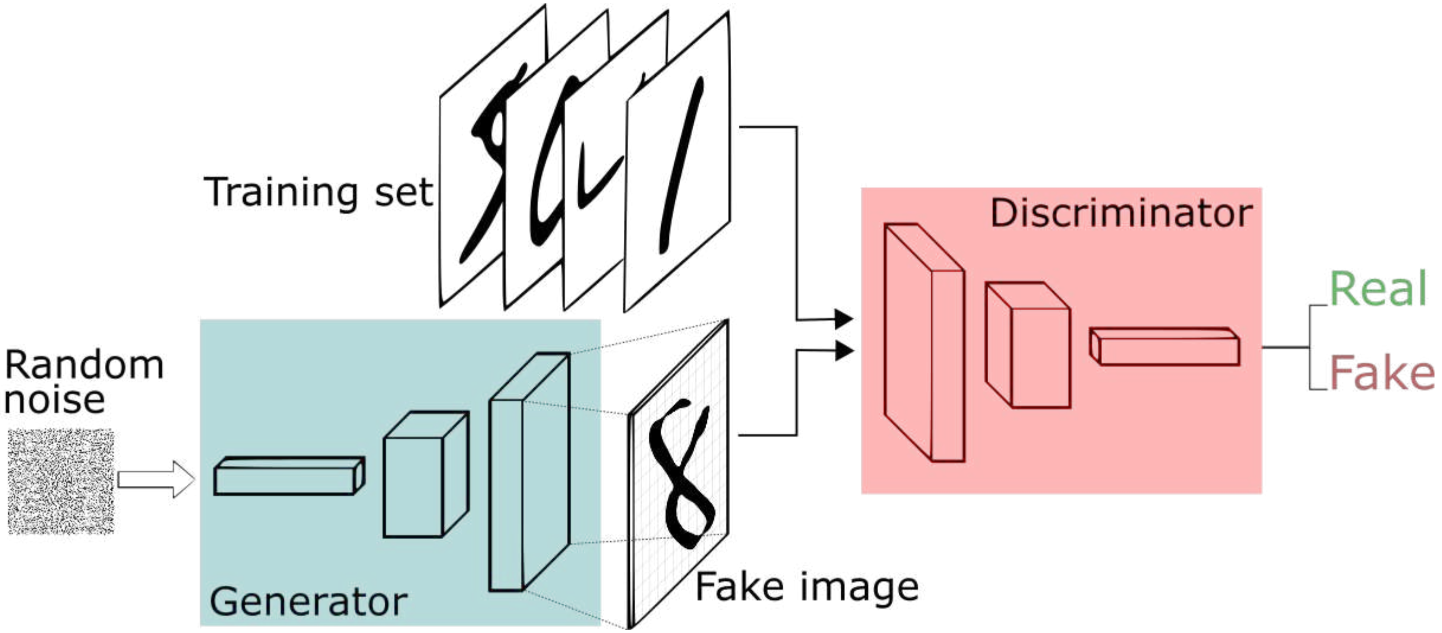
$$\lambda_{x_i} = (\mu_{x_i}, \sigma^2_{x_i})).$$

- We use KL divergence to know how much **q** is approximated of **p.**
  - We should use an algorithm to compute **KL divergence** in a tractable way: minimizing KL means maximizing the **Evidence Lower Bound (ELBO)** to compute the **posterior**.
  - We use gradient ascendent in ELBO over the parameters of each distribution **p** and **q**



GANs

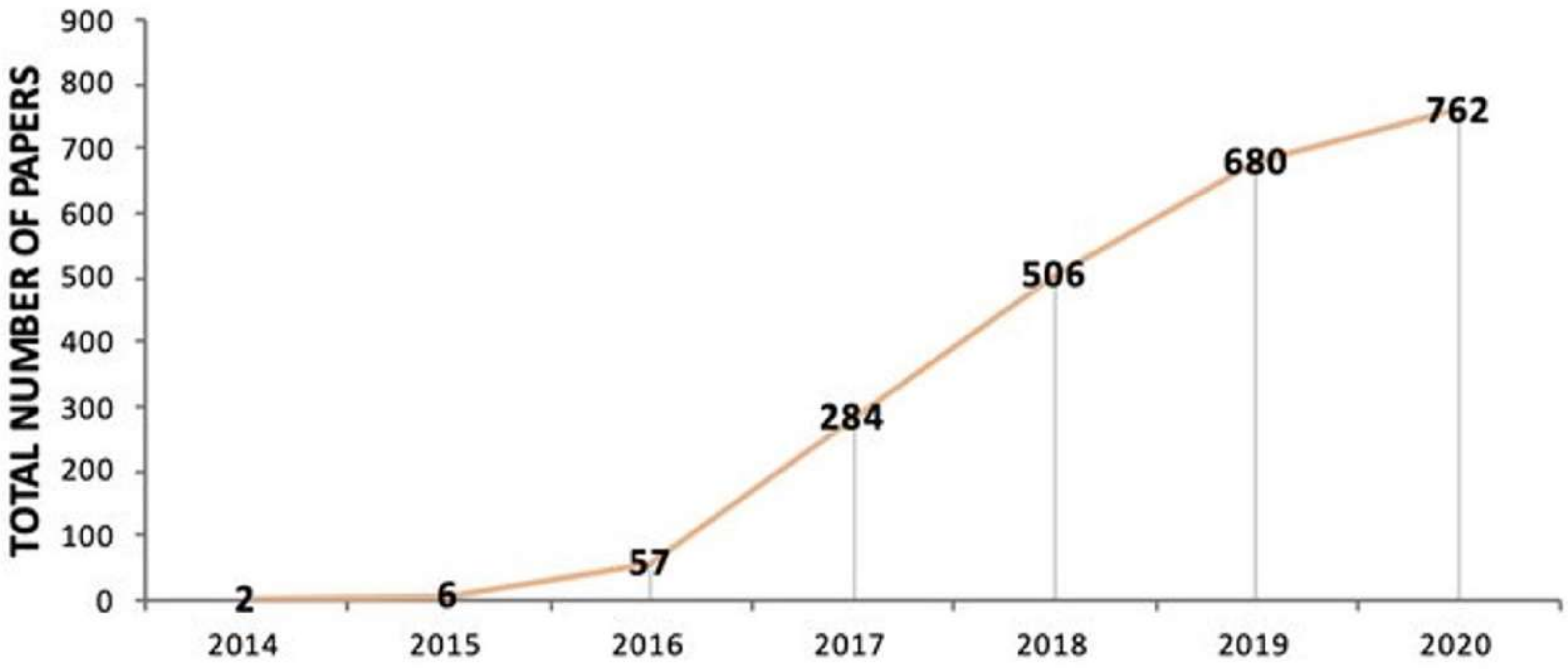
Generative Adversarial Networks



Imagine as:

Generator – counterfeiter
Discriminator - policeman

GANs’ zoo

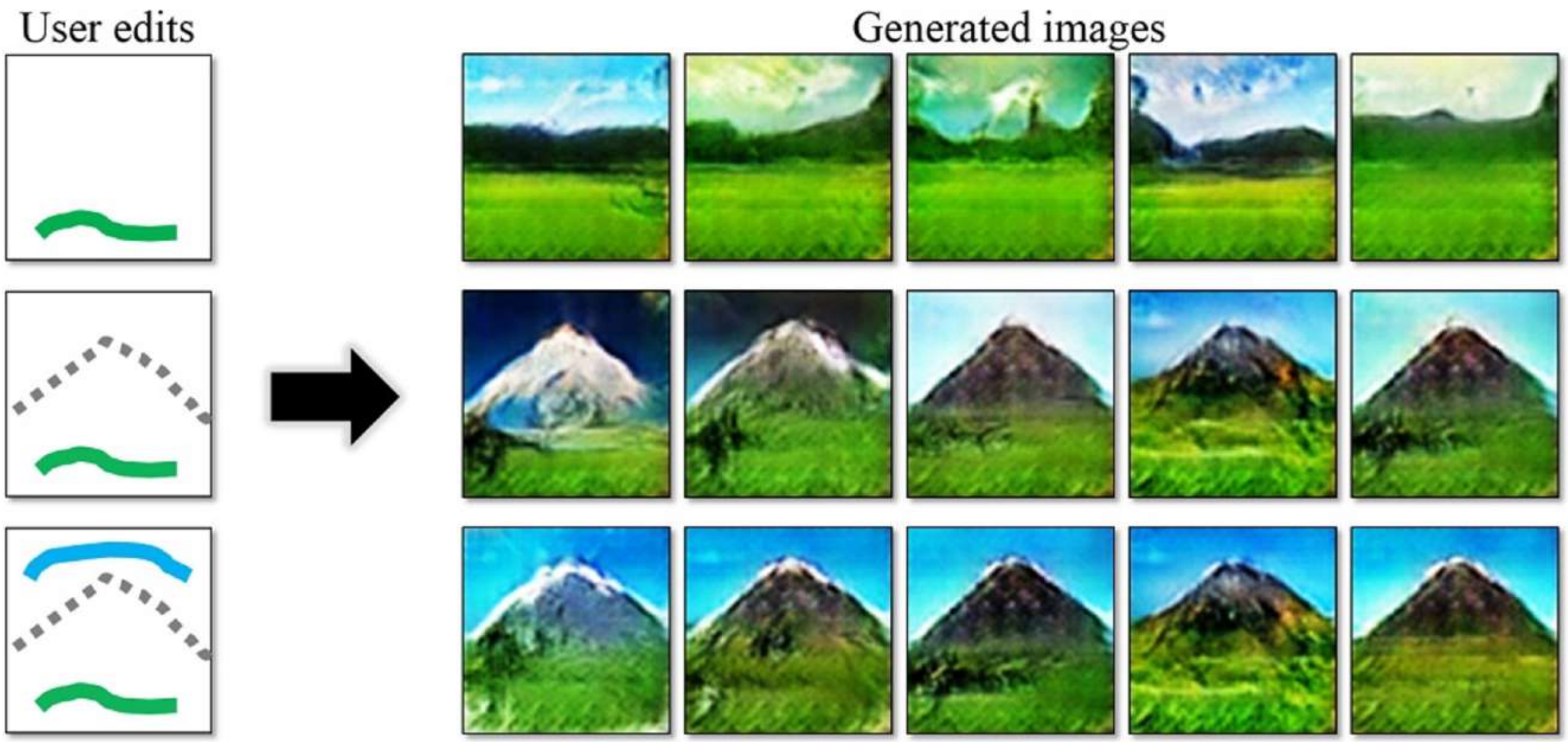


https://github.com/hindupuravinash/the-gan-zoo

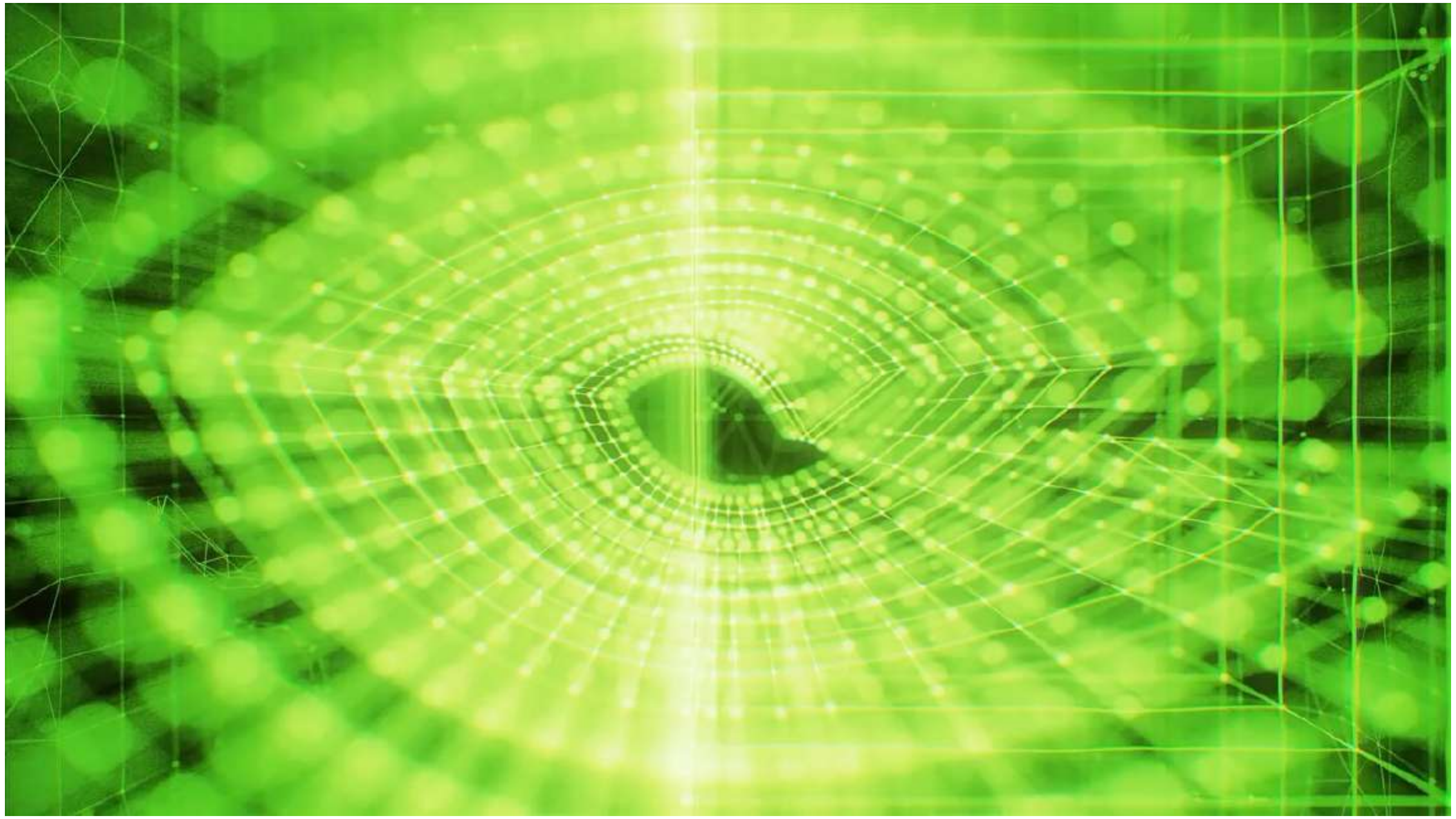
GANs

- Unsupervised generative models
- They are in an architecture that “resembles" supervised learning.
  - **Generator (G)**: fed by random noise (Gaussiano/Uniform); try to generate “fake news”
  - **Discriminator (D)**: tries to discriminate what is real from fake of the Generator; trained by backprop
  - **Generator** and **discriminator** are trained based on **adversarial** process

GANs

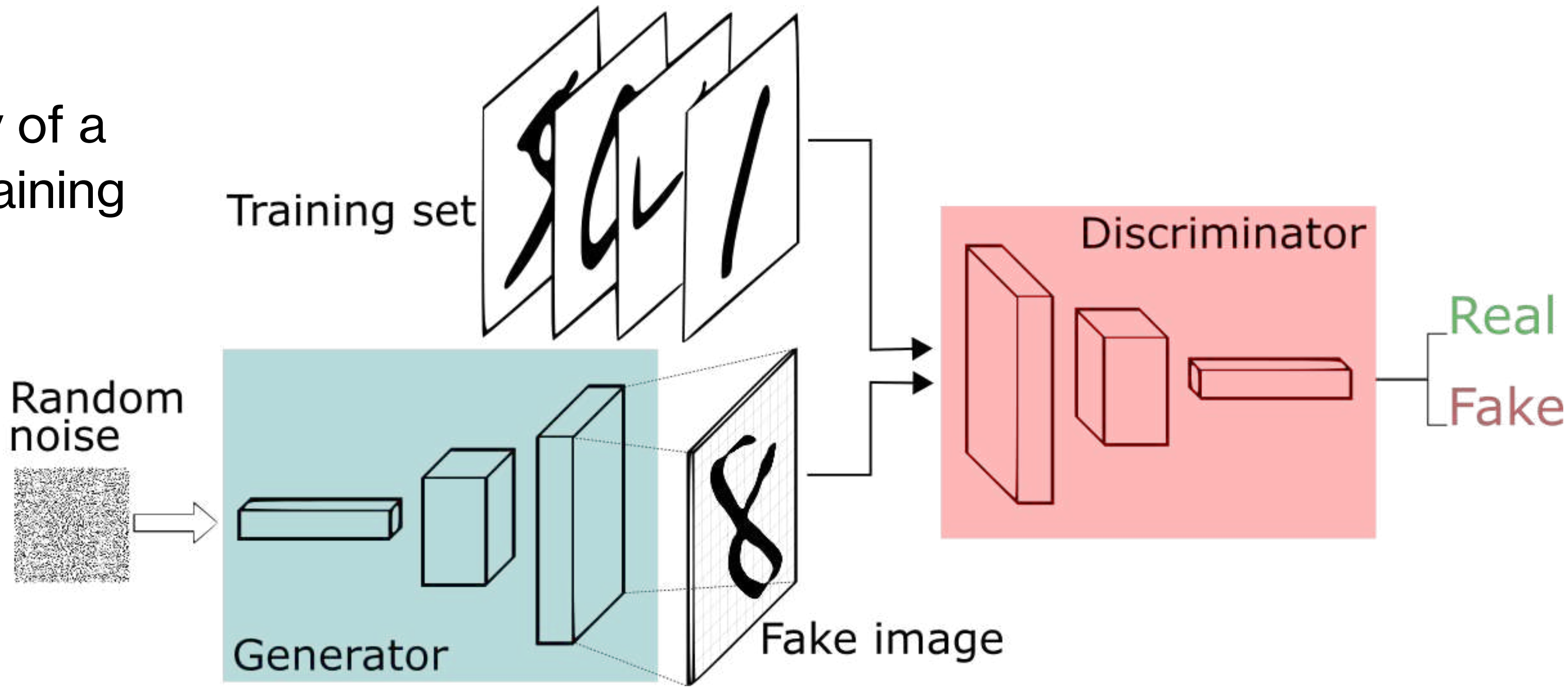


GANs



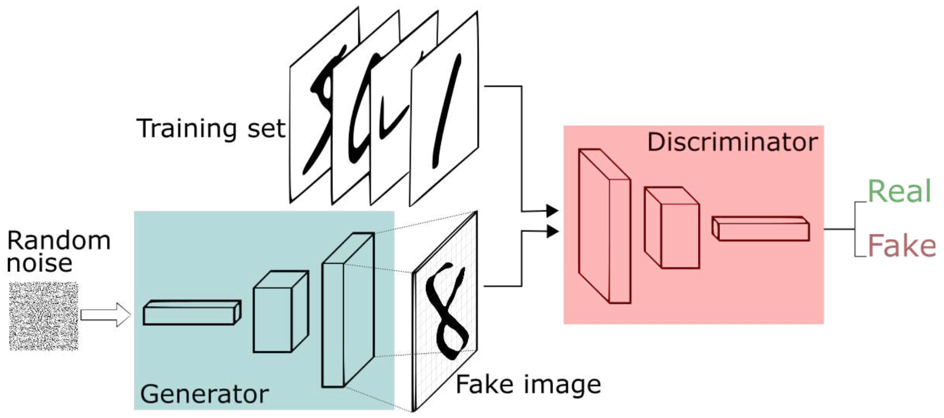
GANs

- **G** needs to capture the distribution of the data.
- **D** Estimates the probability of a sample coming from the training data or from **G**.



GANs

- Work as a as a zero-sum game:
  - If **D** successfully determines what is real or fake, it is rewarded, and there is no need to change the training parameters.
  - In this case, **G** is penalized with updates to its parameters.
- Without limits, **G** generates perfect examples, and **D** guesses correctly only 50% of the time.



GANs

$$\underbrace{\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]}_{\text{Adaptive Loss}}$$

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

Generator: $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$

IMPORTANT: Discriminator and Generator are trained independently!!!!

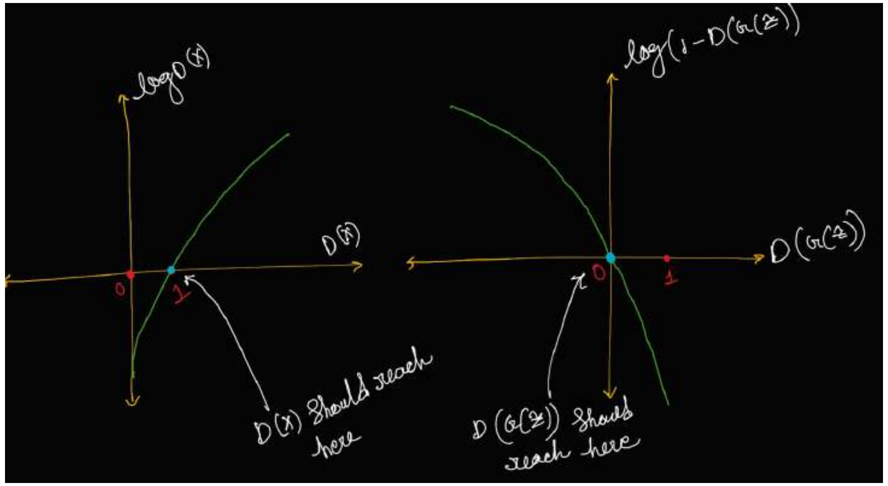
GANs

$$\underset{G}{Min}\ \underset{D}{Max}\left[\underbrace{\mathbb{E}_{(X\sim P(X))}\left[\log D(X)\right]}_{\text{Real samples}} + \underbrace{\mathbb{E}_{(Z\sim P(Z))}\left[\log\left(1 - D(G(Z))\right)\right]}_{\text{Generated sample}}\right]$$

Nash equilibrium:

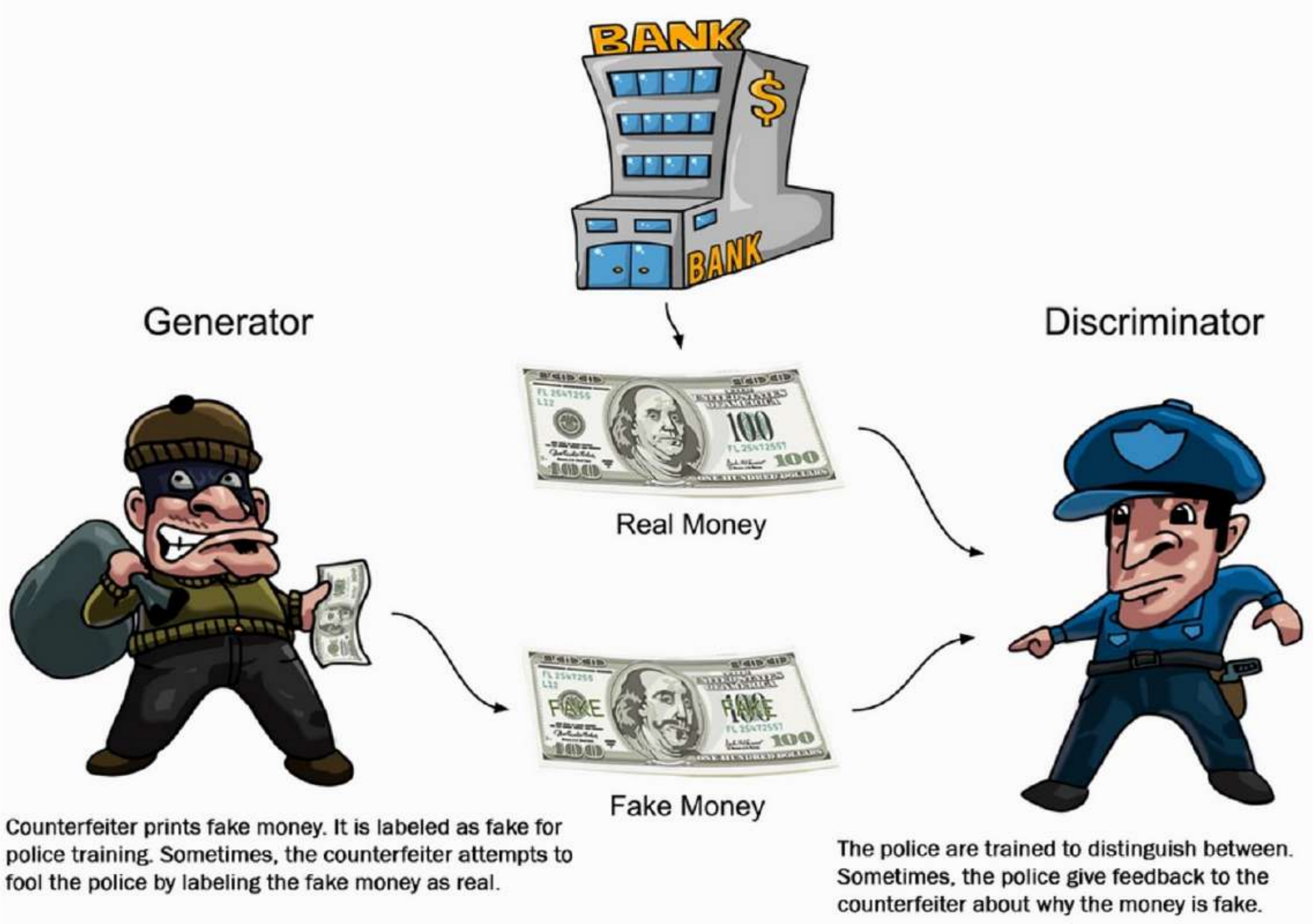
$P_{\text{data}}(x) = P_{\text{gen}}(x) \forall x$

$D(x) = ½ \ \forall x$

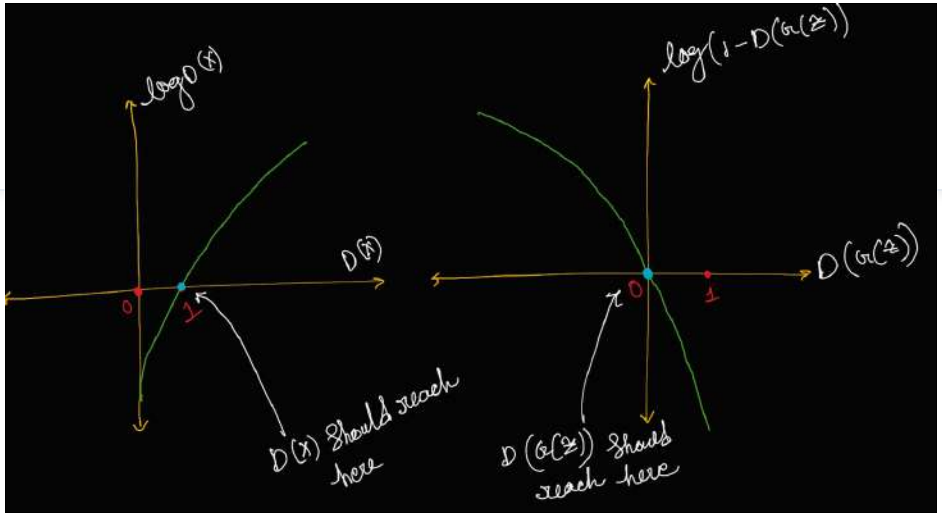


GANs

- Training:
  - D and G compete against each other.
  - Training steps alternate between D and G.
  - Mini-batch stochastic gradient descent/ascent is used.



GANs: Training



for number of training iterations **do**

for k steps **do**

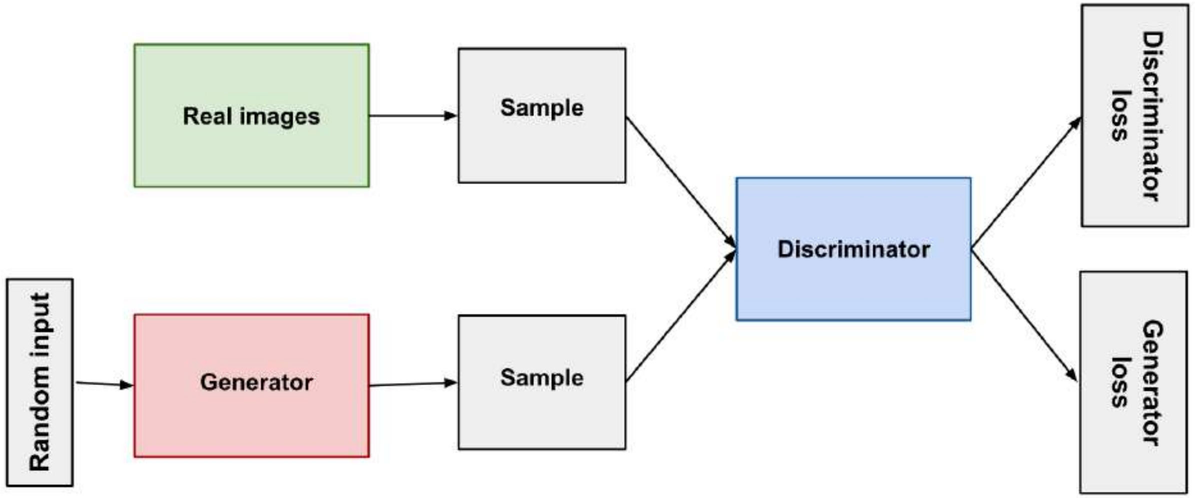
- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

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

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



Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, Generative Adversarial Nets, NIPS, 2014

ChatGPT


ME: FIND 10 SCHOLARLY ARTICLES
CHATGPT: MAKE THEM UP

NLP: The main task

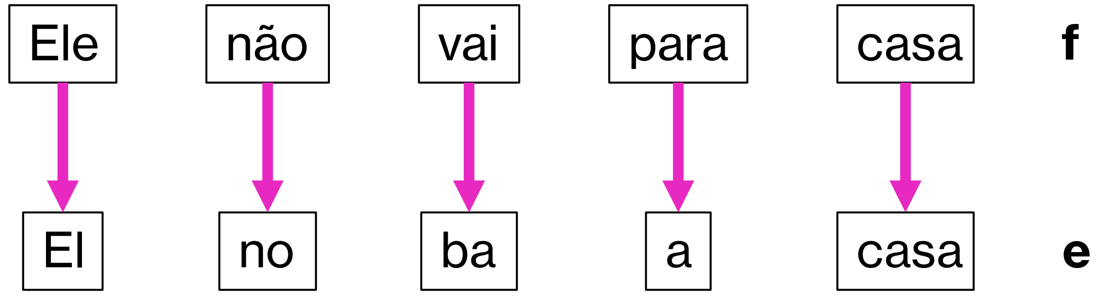
- Machine translation of 2 sequences
- Model for **decoding**:

  $P(e \mid f)$

- Find the translation with highest probability:

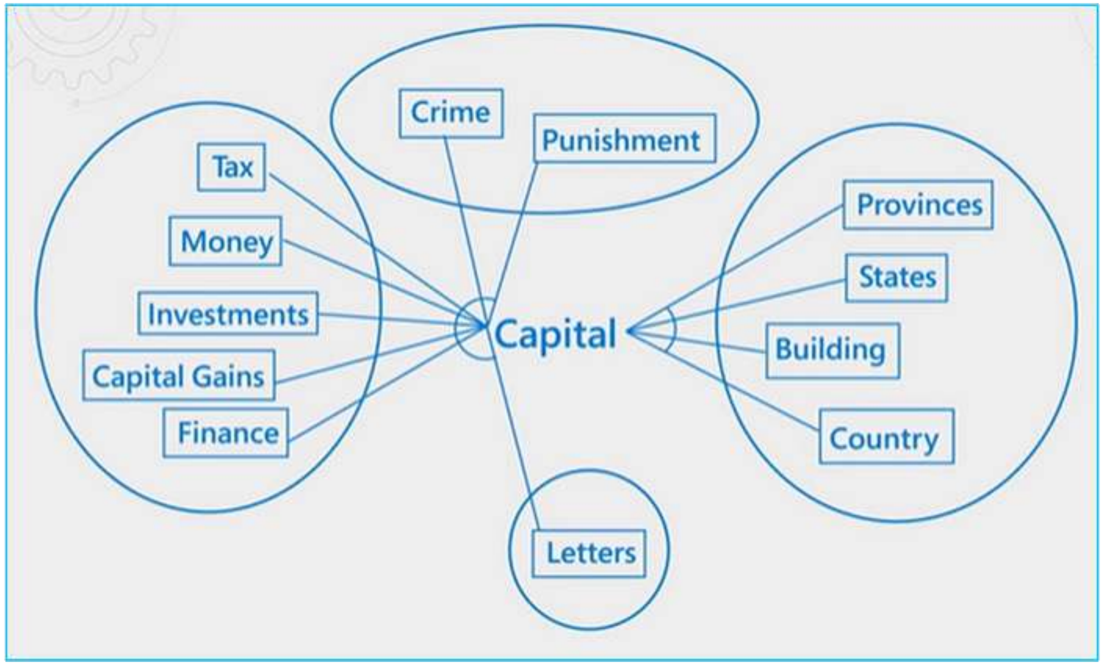
  $$e_{best} = \text{argmax}_e \, P(e \mid f)$$

Example:



NLP: The main task

- Two types of error:
  - the most probable translation is bad -> **fix the model**
  - search does not find the most probably translation -> **fix the search**
- Decoding is evaluated by search error, not quality of translations (although these are often correlated)
- Inherent problems: complexity (NP-complete), alignment / reordering, context



Timeline of NLP

1950 → **1960** → **1980** → **1990** → **>2013**

1950
- Turing test
- Georgetown-IBM experiment
- Rules-based methods

1960
- ELIZA
- ALPAC report and First AI Winter

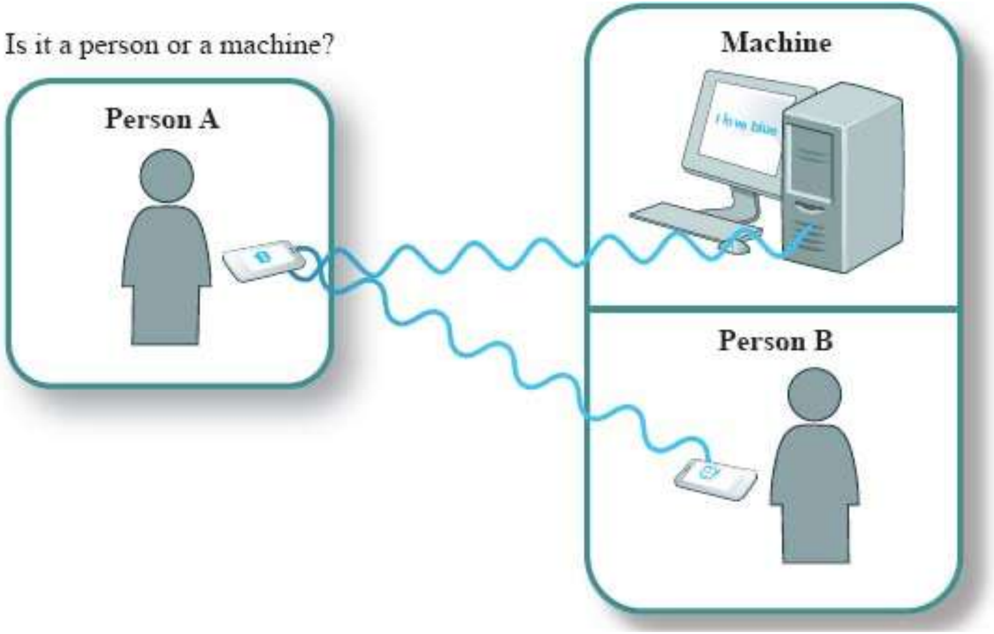
1980
- Statistical models (TF-IDF)
- Expert systems (e.g. MYCIN)

1990
- Probabilistic graphical models (especially HMM)
- RNN

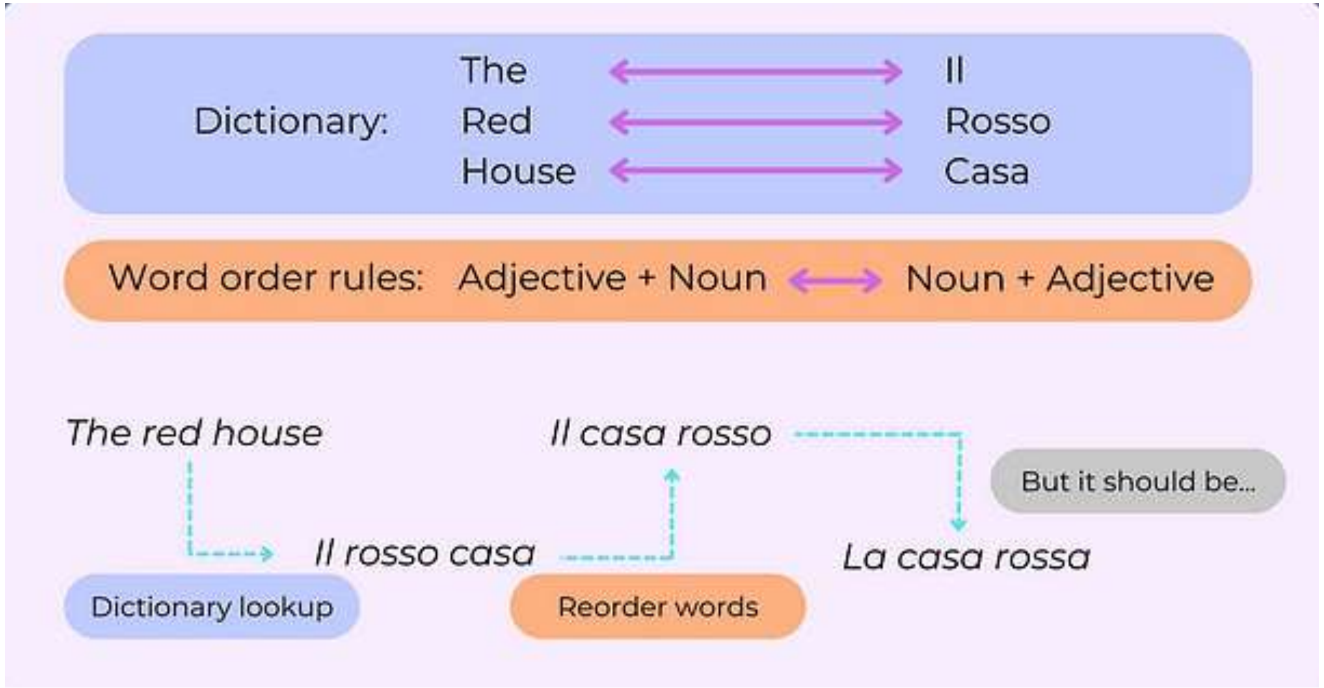
>2013
- Word2vec
- Recursive neural tensor networks (RNTNs)
- …
- CNN
- LSTM
- LLMs (GPT, BERT)

Timeline of NLP: early models

Turing test

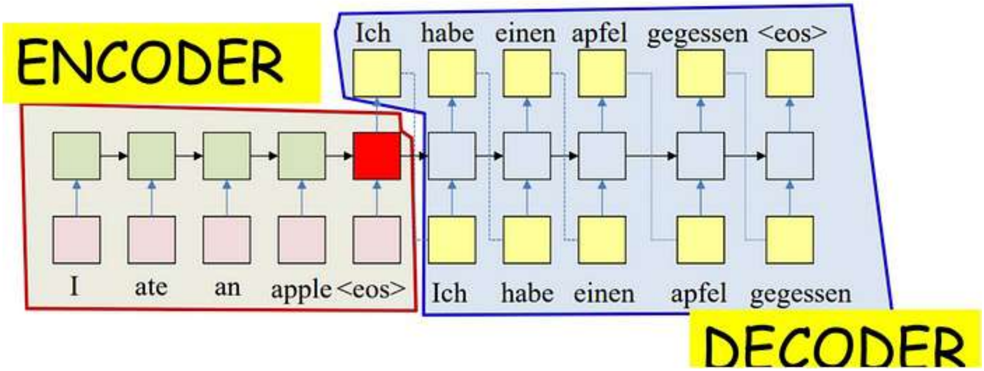


Rule-based systems

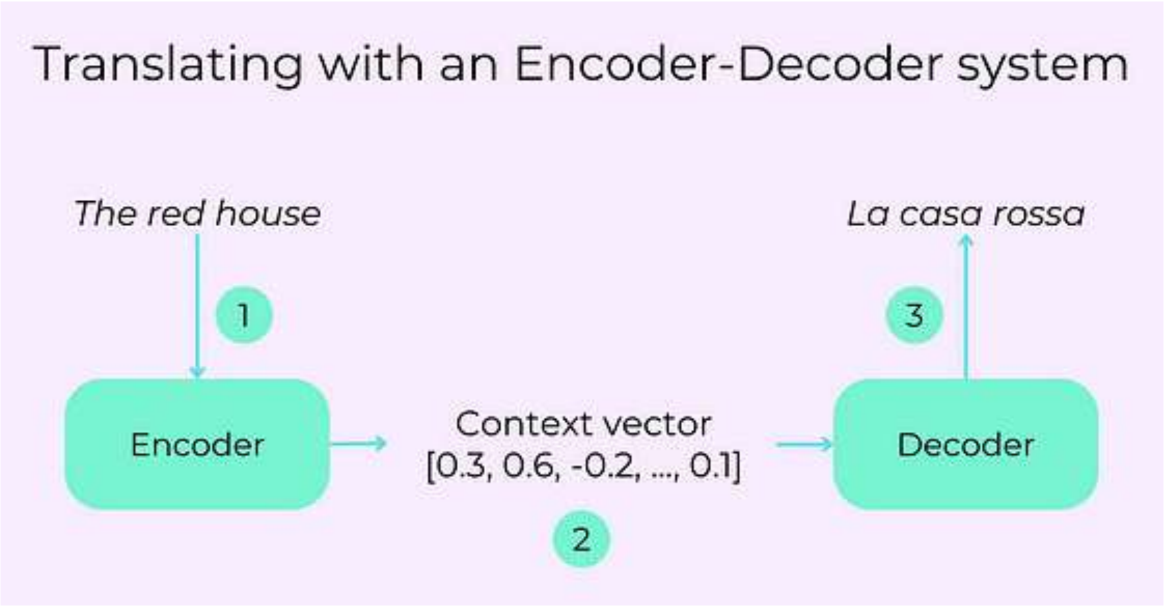


Timeline of NLP: current models

- From 1990: sequence to sequence (seq2seq) probabilistic or neural network models
- From 2013: deep learning models applied on the **encoder** and **decoder**



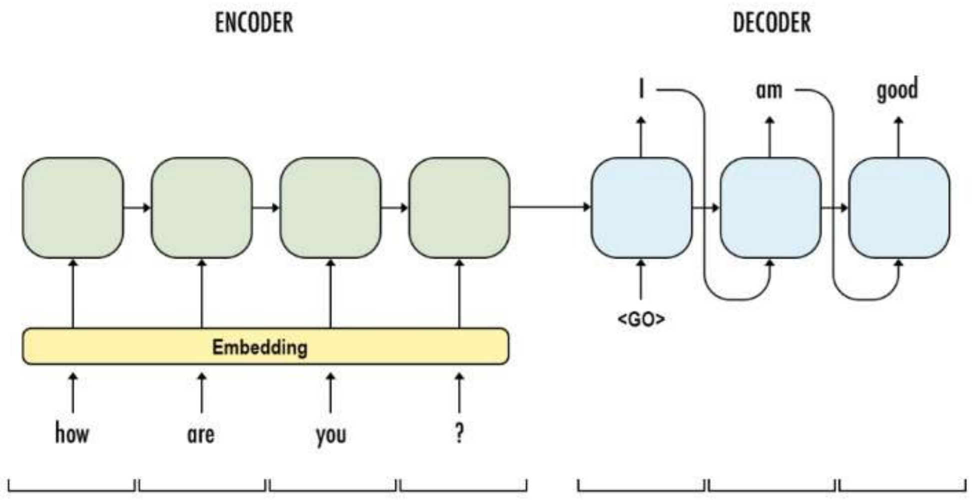
Encoder-decoder architecture

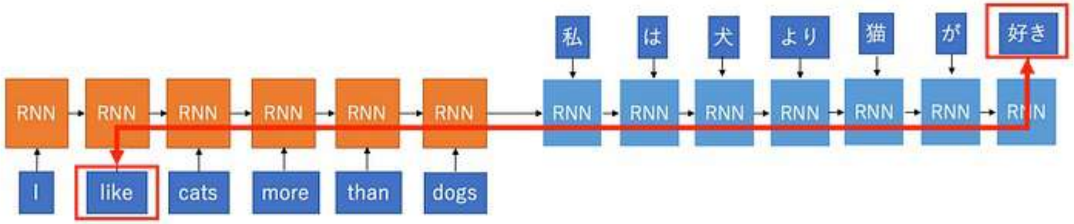


Pathway to ChatGPT

- RNN (e.g. LSTM and GRU)
  - Encoder: in charge of outputing a context vector (final hidden state)
  - Decoder: outputs a different sequence (translation, question-answering, summarization, etc)

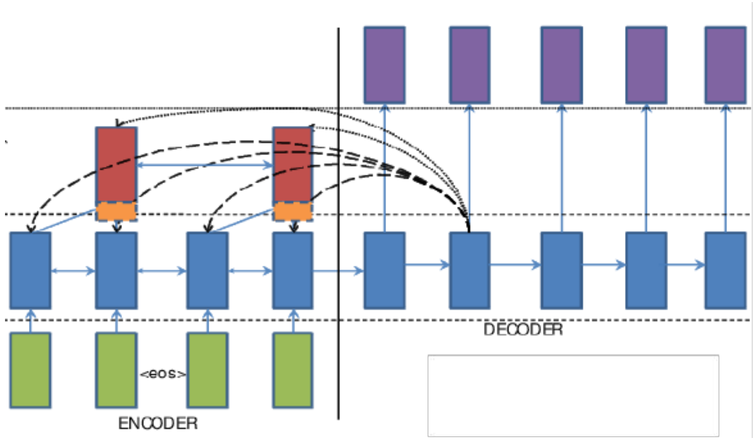
- Drawbacks:
  - Performance drops drastically for longer sentences since embeddings (signals) get diluted as they pass through the network





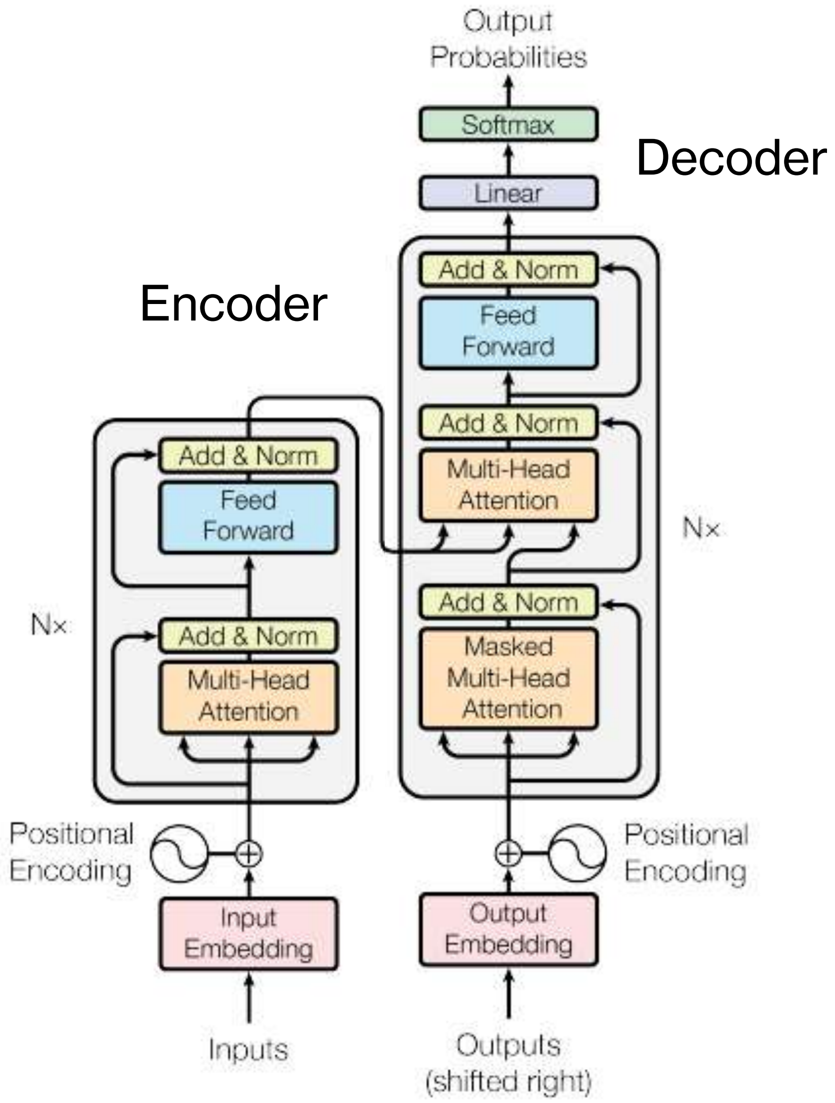
Pathway to ChatGPT

- The previous problem can be solved by skip connections
  - feed every hidden state of the encoder into every input of the decoder
- This creates another problem:
  - how to combine multiple hidden state into a single context vector?
- More problems: Memory (RNNs requires a lot of memory) and context (RNN only looks at the tokens to the left)



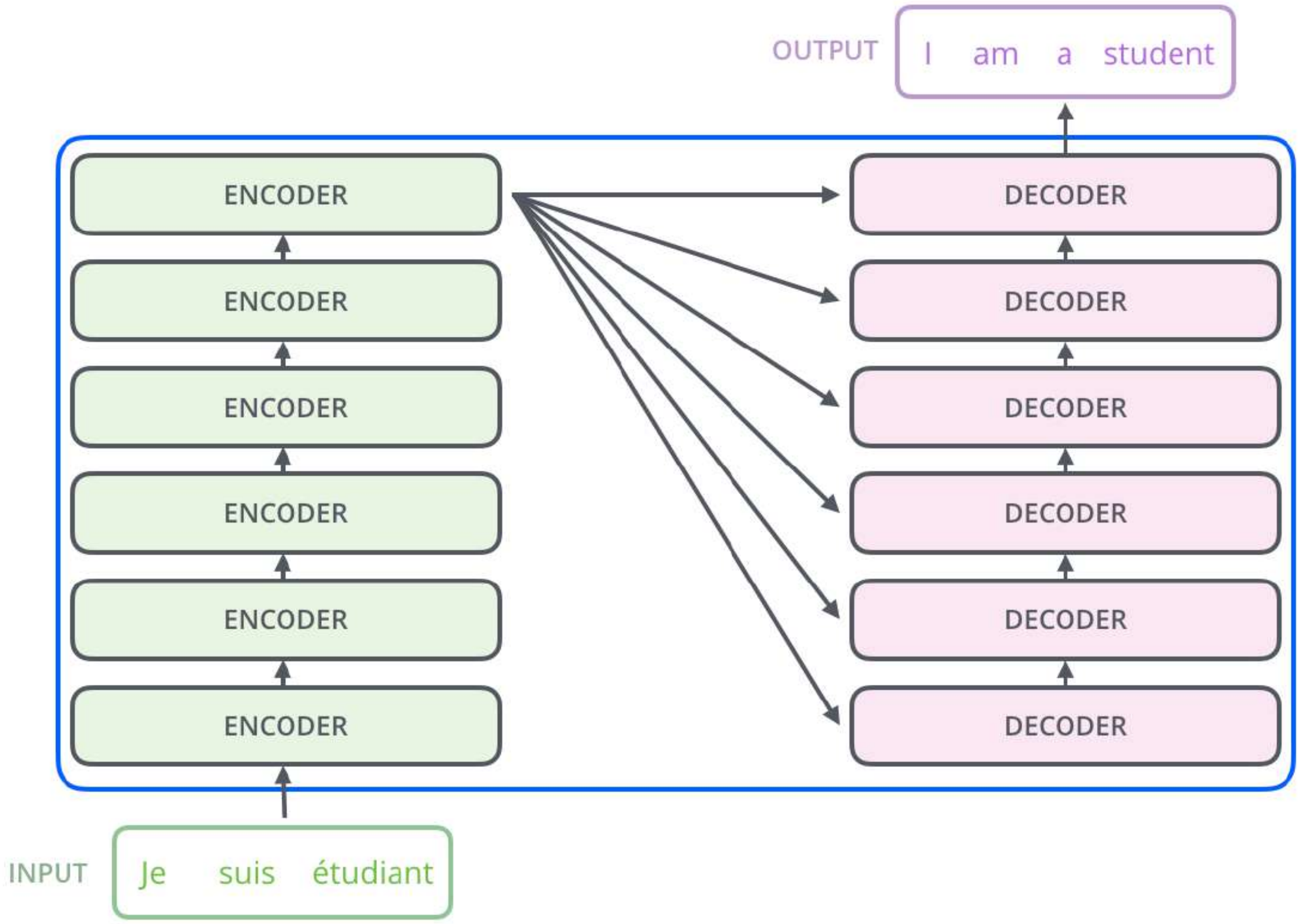
Inside Transformer

- Why do we need Transformer?
  - In RNN-based networks, the decoder only **access the last hidden state** and it will lose relevant information
  - Attention can solve the last problem, but… RNNs **treat one element at a time**

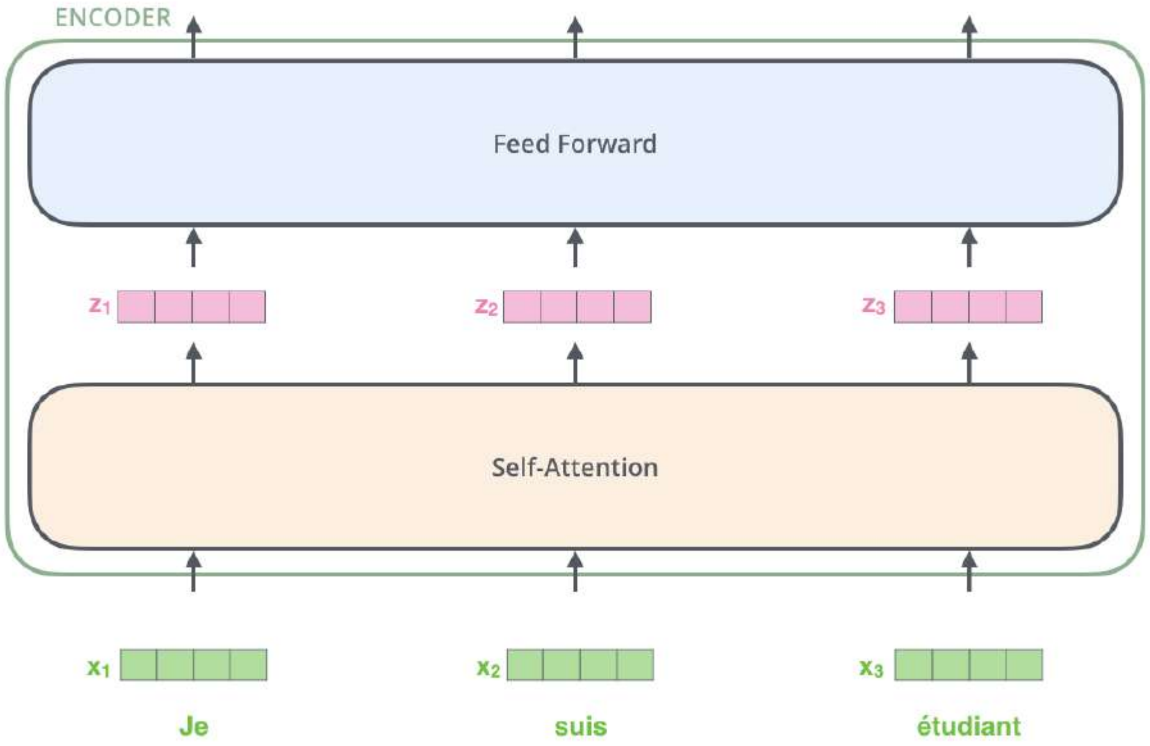


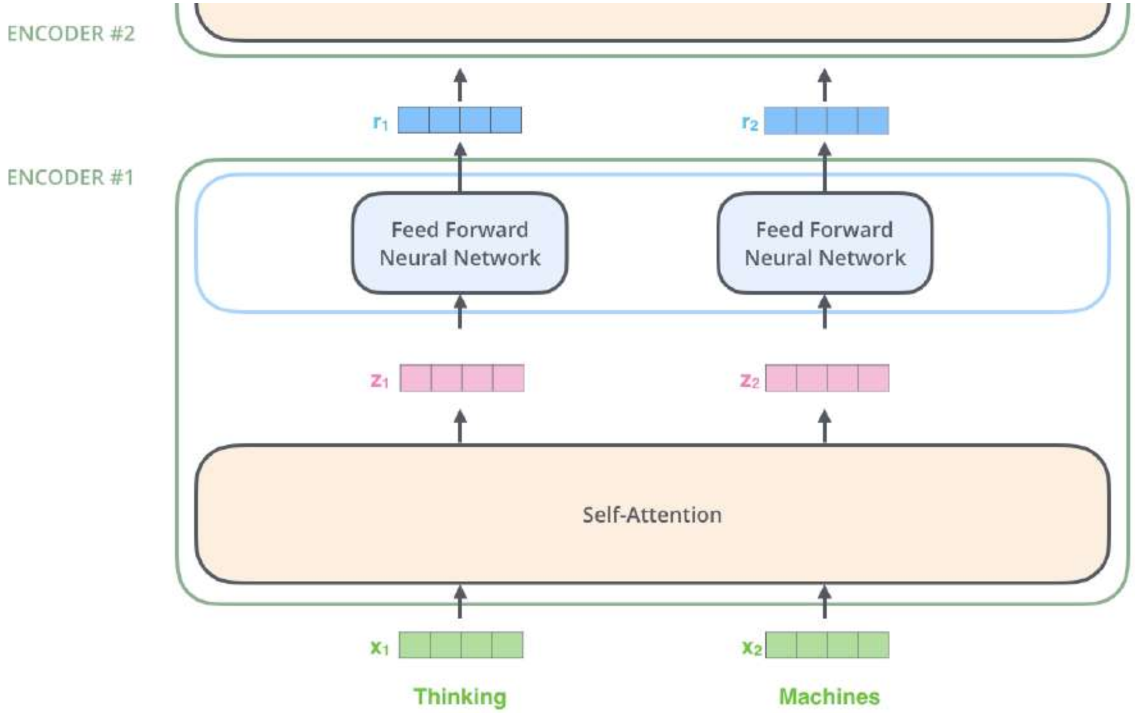
[1] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia, "Attention is all you need" , 2017.

What you will really find inside...

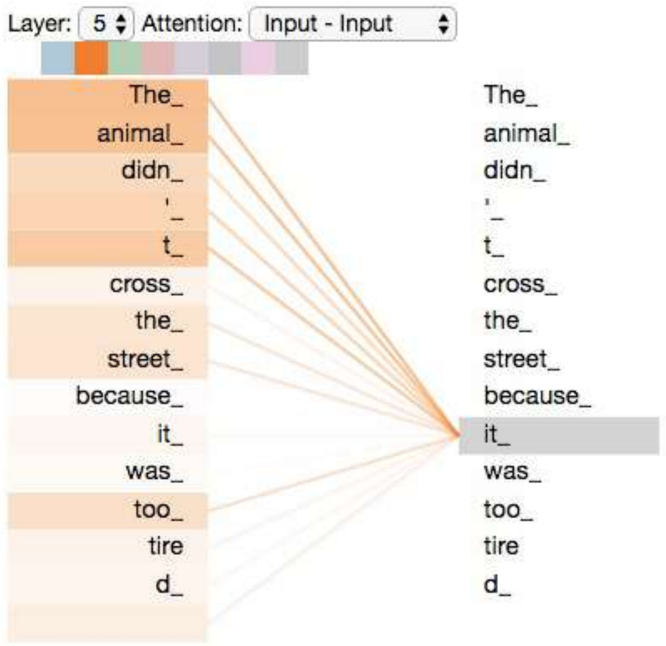


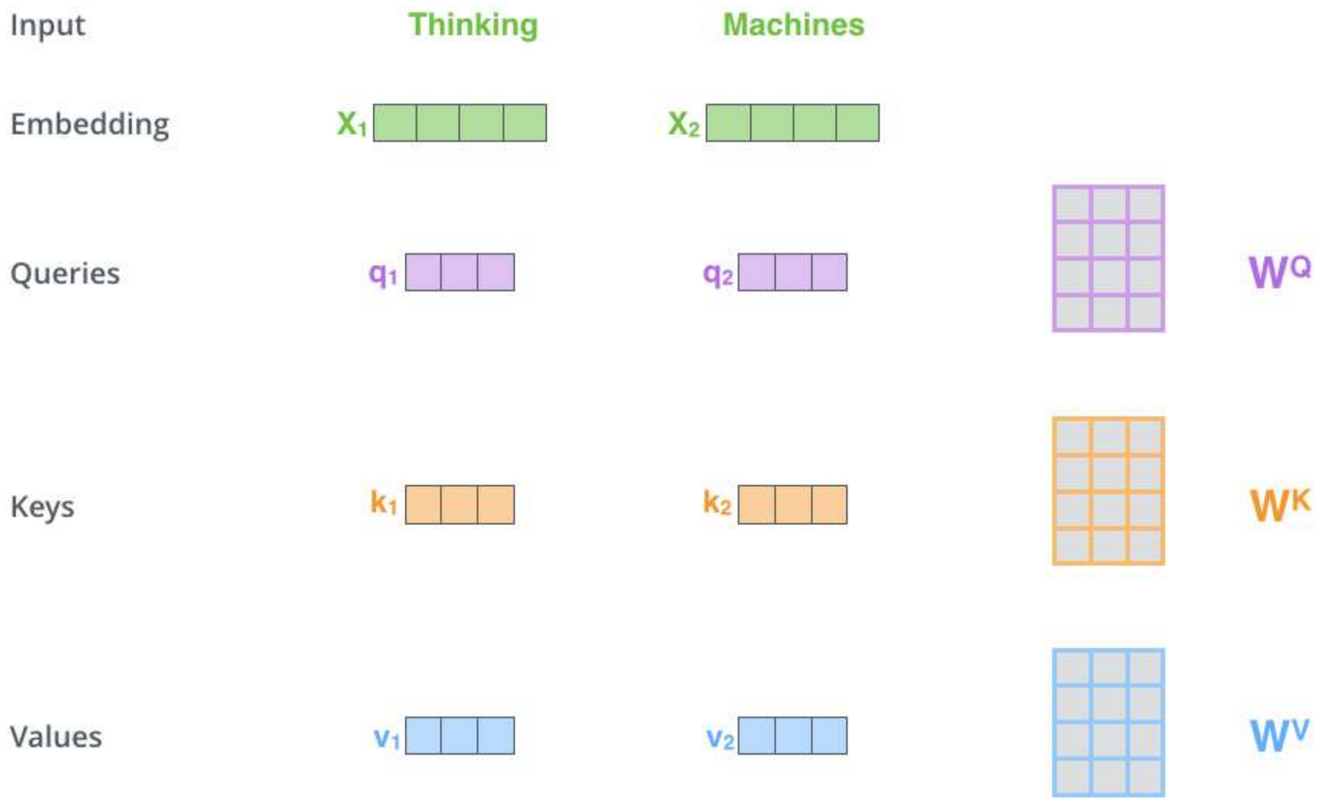
Tensor is all we need!



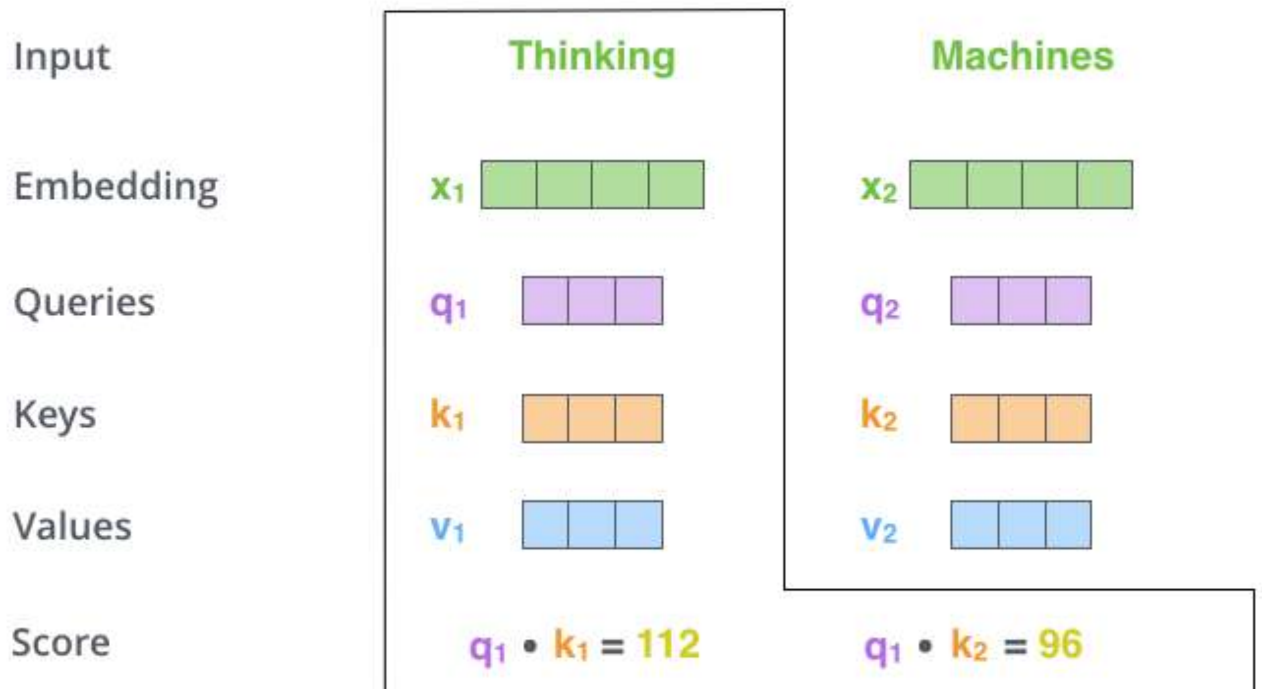


Self-attention

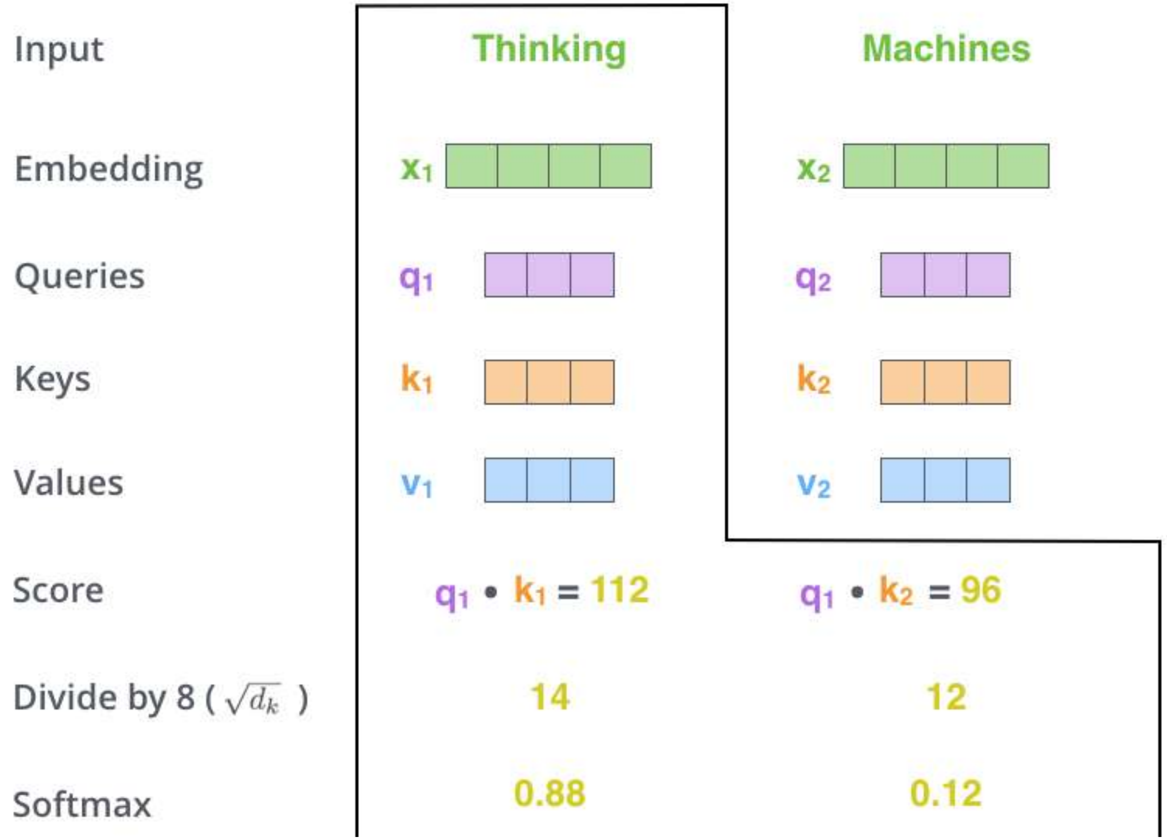




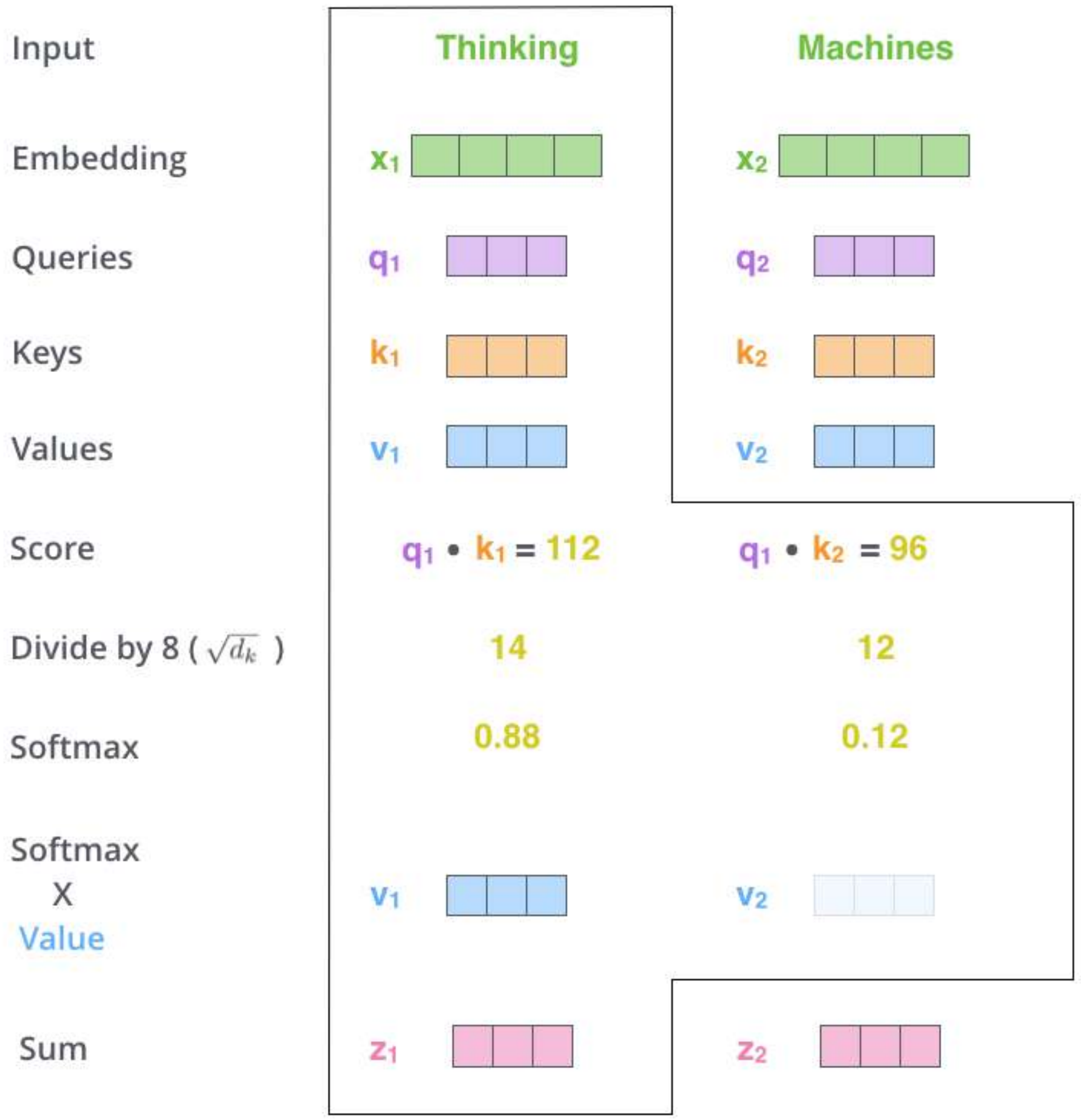
Self-attention



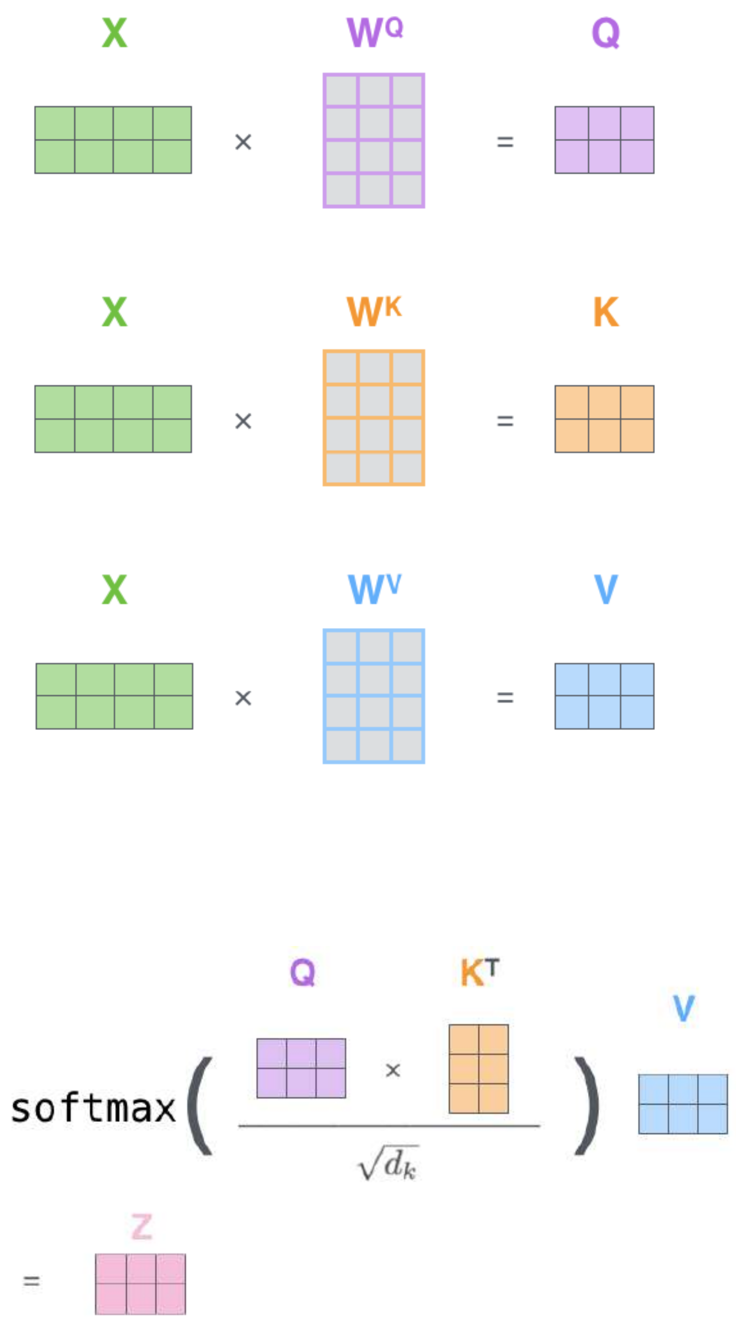
Self-attention



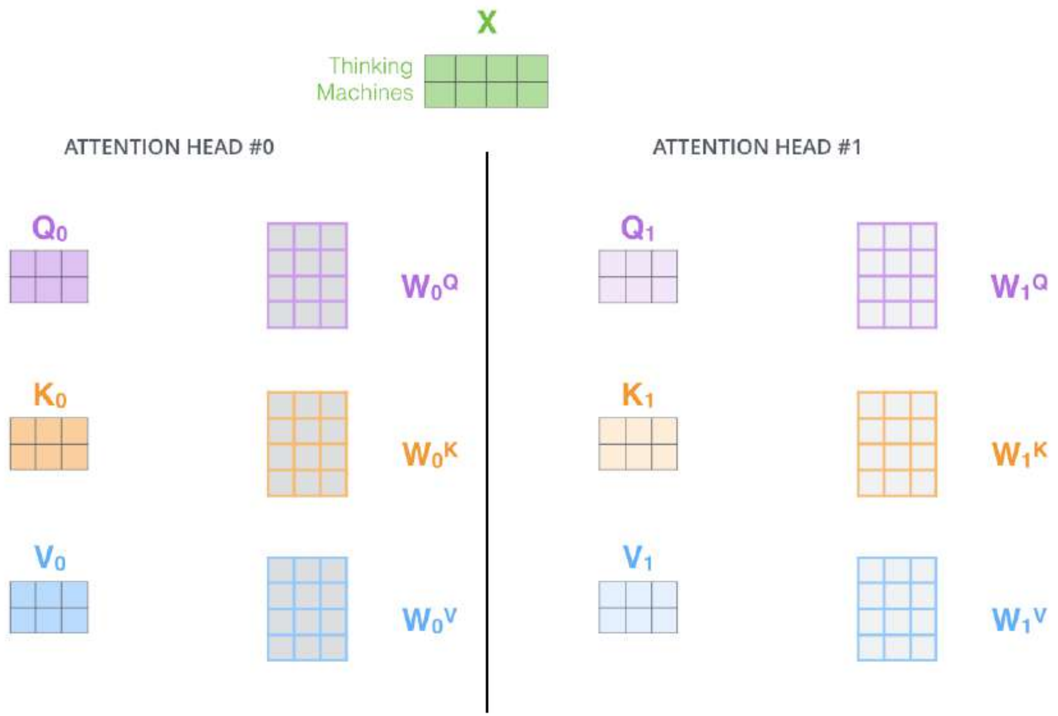
Self-attention

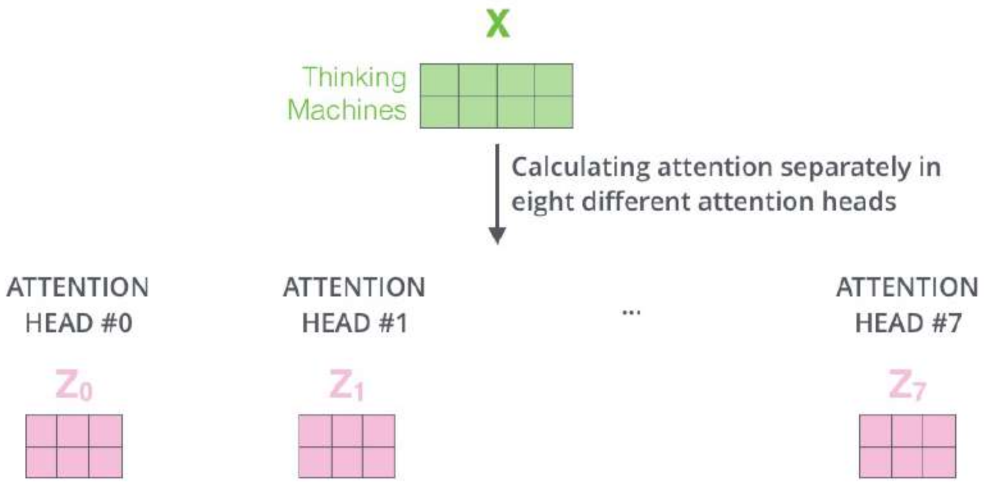


Self-attention

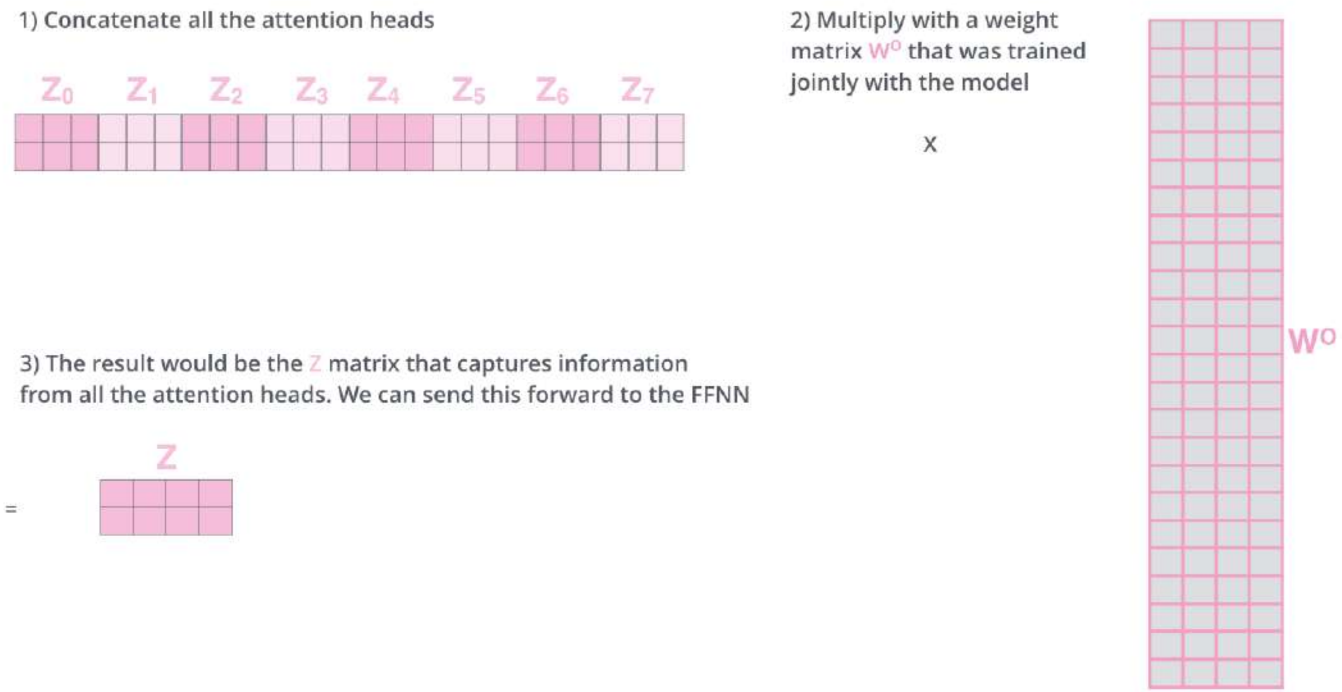


Multi-head self-attention

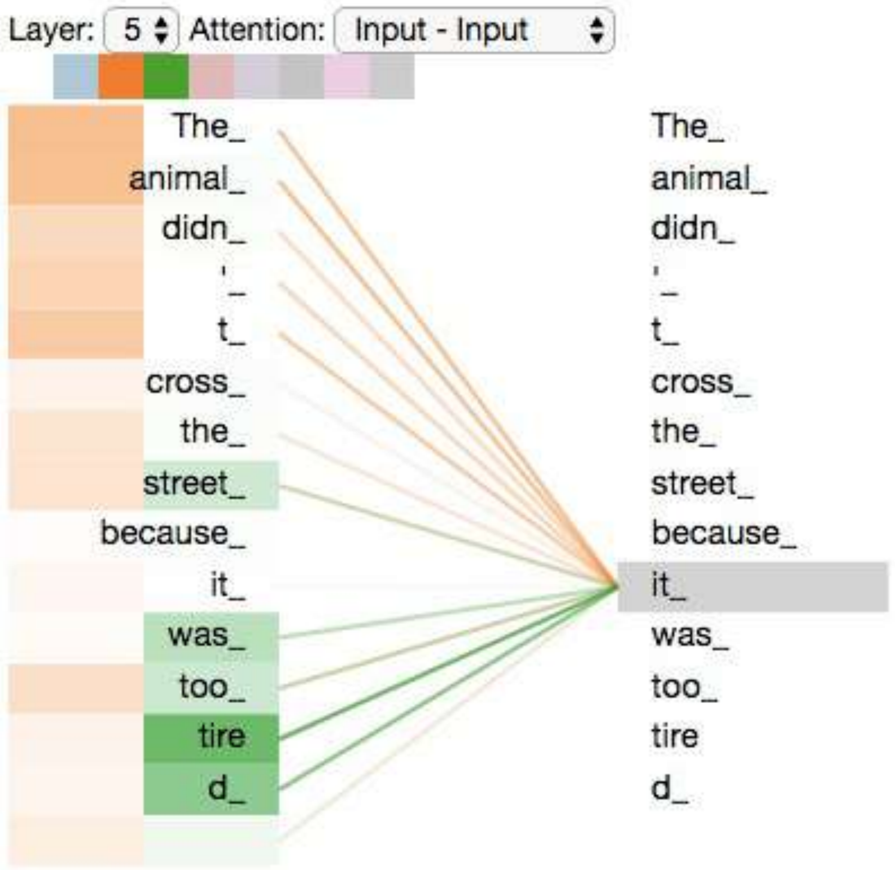




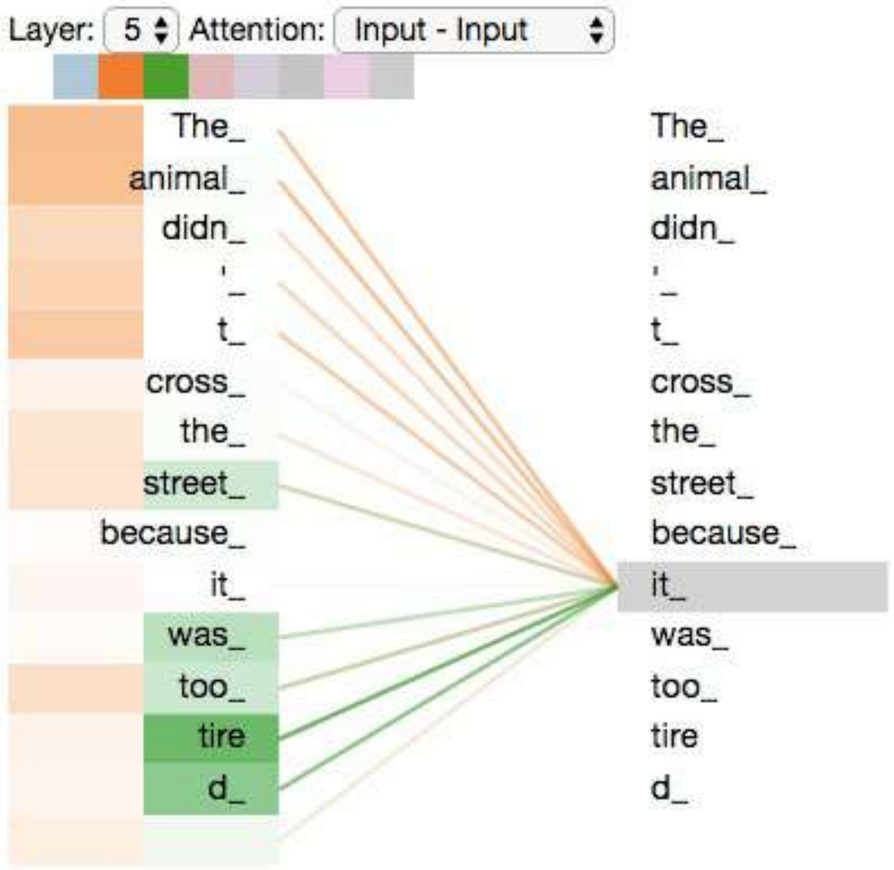
Multi-head self-attention



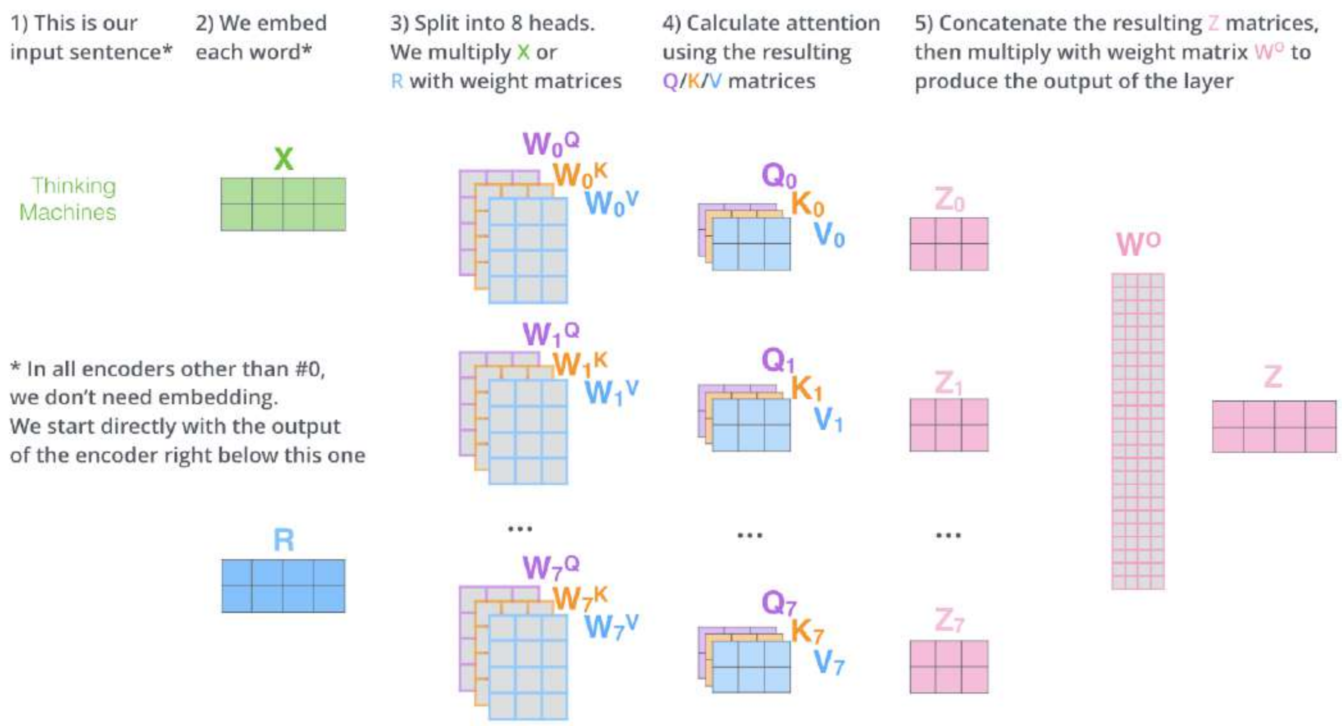
Self-attention



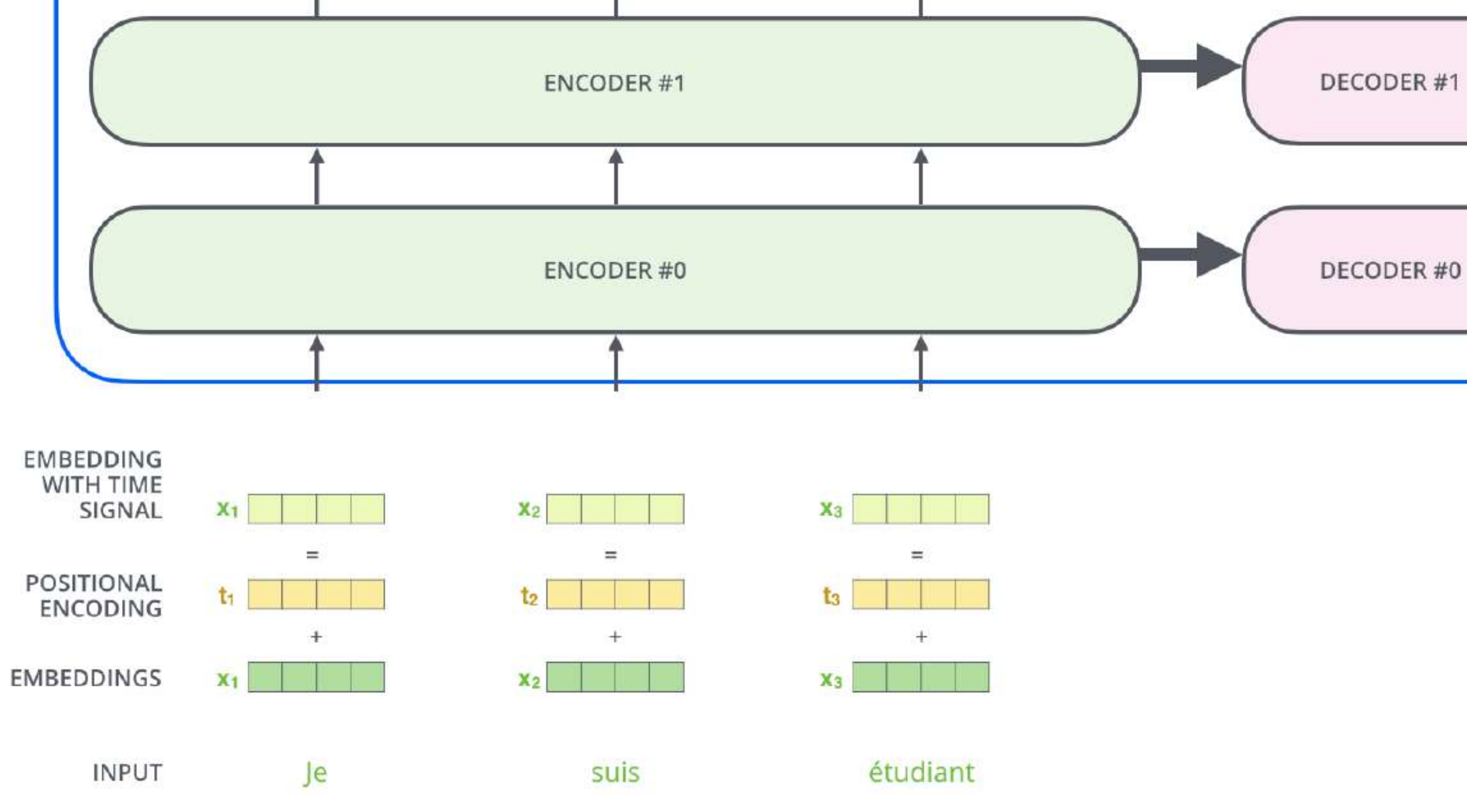
Multi-head self-attention

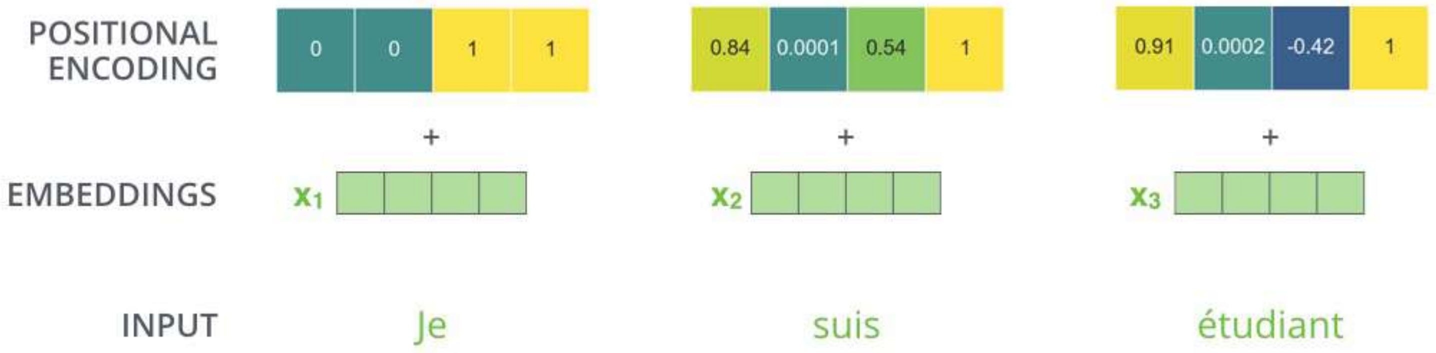


All the steps till now

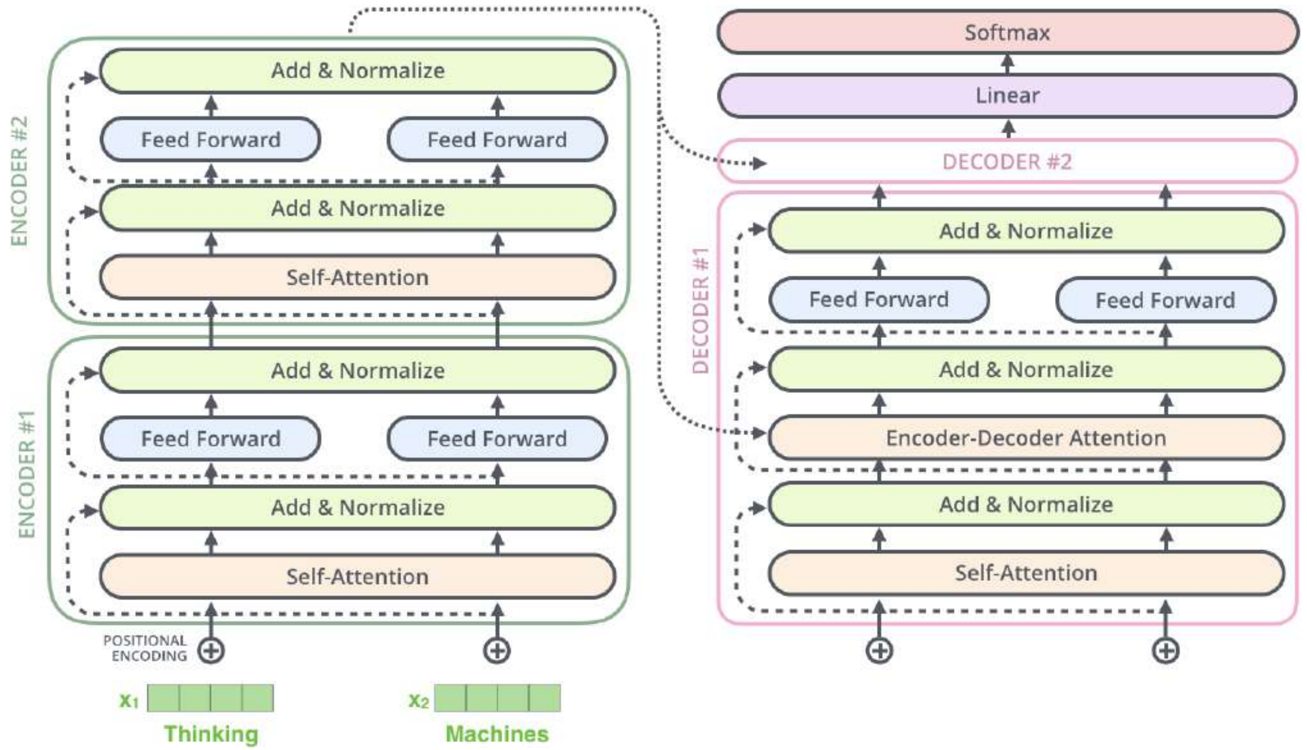


Positional encoding

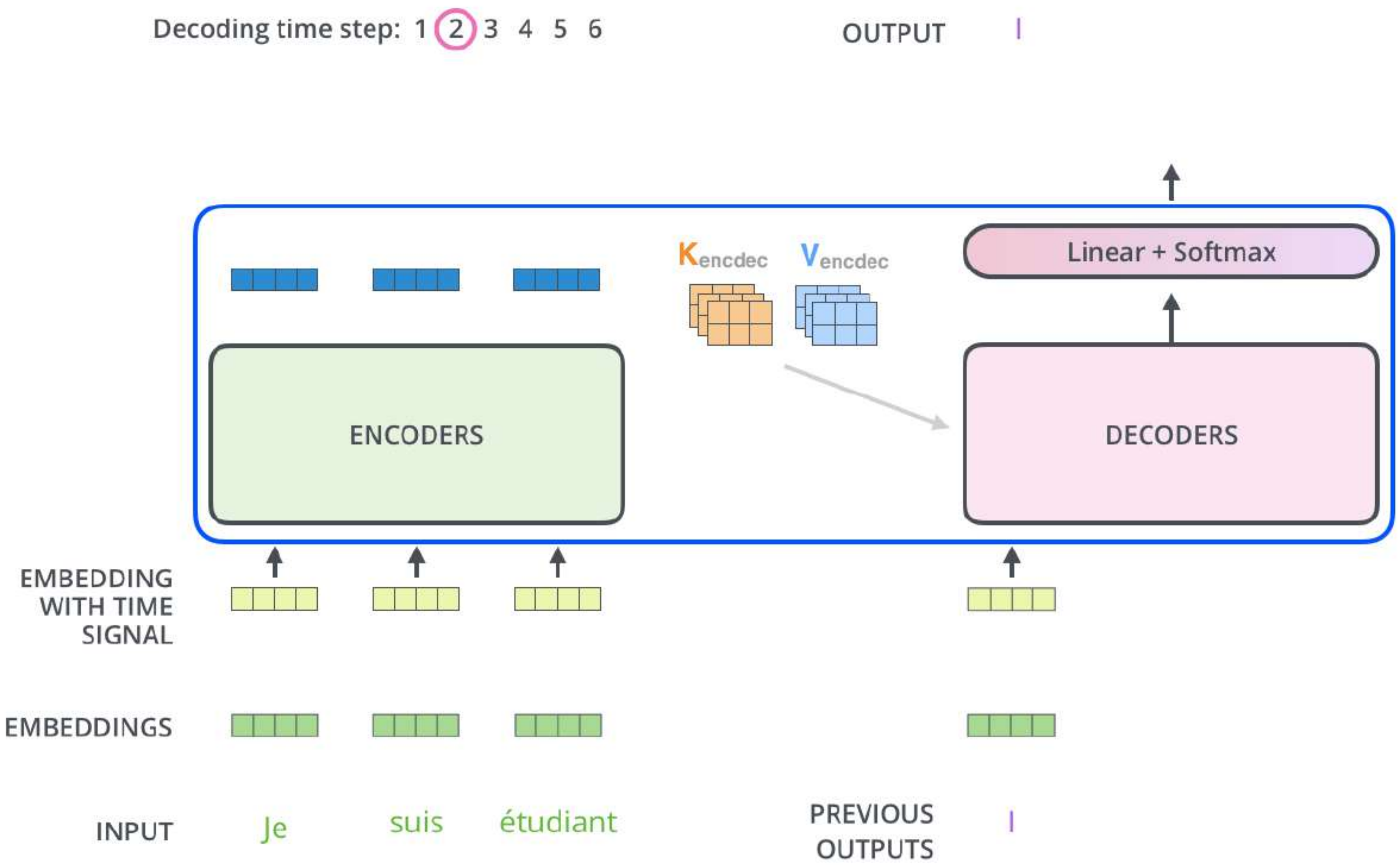




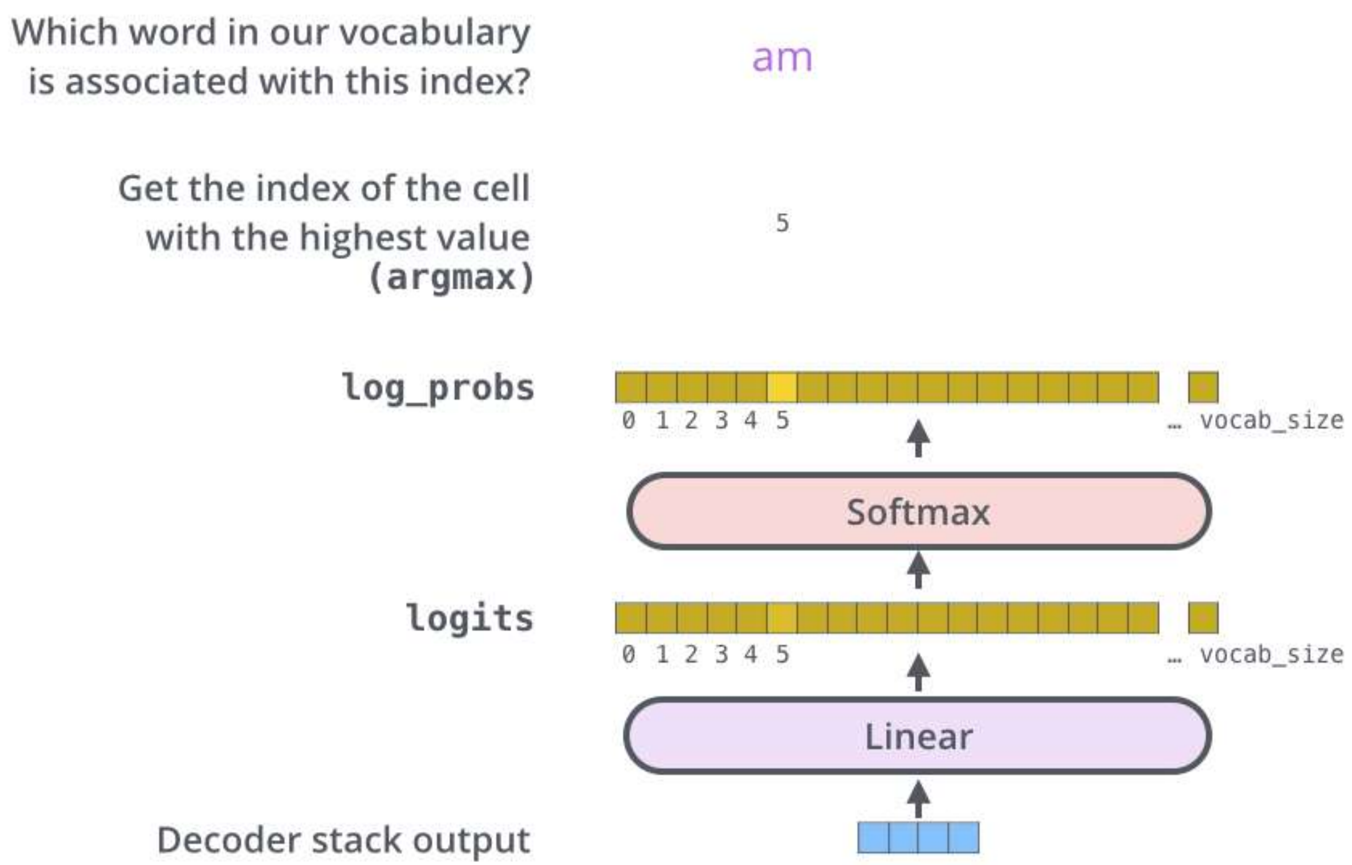
Residuals, FFN, Add&Normalize



Decoder


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
K encdec
V encdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING
WITH TIME
SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS
OUTPUTS
I

Output



Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output

Toy example

Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>

Ground truth

Trained Model Outputs

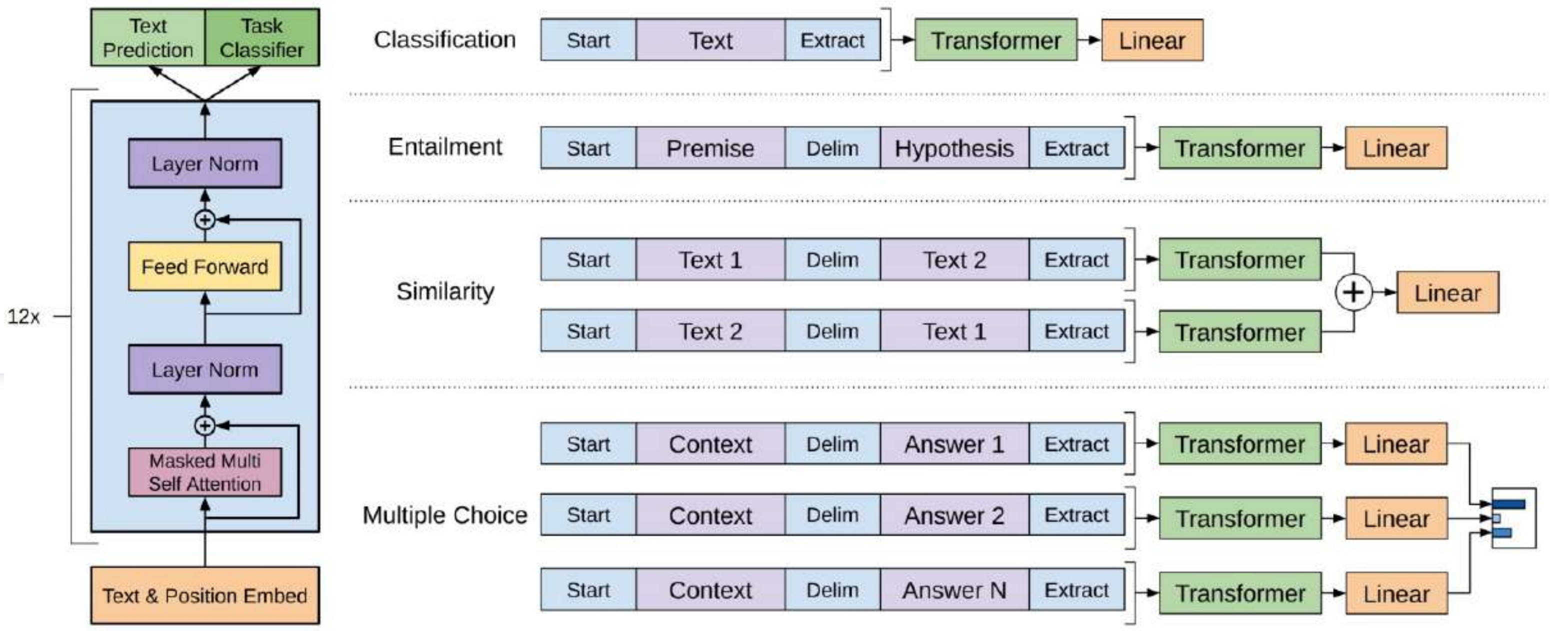
Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98
	a	am	I	thanks	student	<eos>

Trained model output

Generative Pre-trained Transformer

Version	Architecture	Parameter count	Training data	Release date
GPT-1	12-level, 12-headed Transformer decoder (no encoder), followed by linear-softmax.	117 million	**BookCorpus**: 4.5 GB of text, from 7000 unpublished books of various genres.	June 11, 2018
GPT-2	GPT-1, with modified normalization	1.5 billion	**WebText**: 40 GB of text, 8 million documents, from 45 million webpages upvoted on Reddit.	February 14, 2019
GPT-3	GPT-2, with modification to allow larger scaling	175 billion	570 GB plaintext, 0.4 trillion tokens. Mostly **CommonCrawl**, **WebText**, **English Wikipedia**, and two books corpora (**Books1** and **Books2**).	June 11, 2020

GPT-1



- **Semi-supervised learning**: **unsupervised** pre-training followed by **supervised** fine-tuned models – that's why the name **generative pre-training**
- Only uses the decoder part of the transformer
- Supervised fine-tuning was achieved by adding a linear and a softmax layer to the transformer model to get the task labels for downstream tasks.

Unsupervised learning:

$$L_1(T) = \sum_i \log P(t_i | t_{i-k}, \ldots, t_{i-1}; \theta)$$

Supervised fine-tuning:

$$L_2(C) = \sum_{x,y} \log P(y | x_1, \ldots, x_n) \qquad L_3(C) = L_2(C) + \lambda L_1(C)$$

GPT-1

- **Unsupervised learning:**
- Model used 768-dimensional state for encoding tokens into word embeddings. Position embeddings were also learnt during training.
- 12 layered model was used with 12 attention heads in each self-attention layer.
- Adam optimizer was used with learning rate of 2.5e-4.
- Attention, residual and embedding dropouts were used for regularization, with dropout rate of 0.1.
- GELU was used as activation function.
- The model was trained for 100 epochs on mini-batches of size 64 and sequence length of 512.
- The model had 117M parameters in total.

- **Supervised fine-tuning:**
- Supervised fine-tuning took as few as 3 epochs for most of the downstream tasks.
- Most of the hyper parameters from unsupervised pre-training were used for fine-tuning

- **GPT-1 performed better than specifically trained supervised state-of-the-art models in 9 out of 12 tasks**

GPT-2

- GPT-1 train the language model as P(output | input)
- GPT-2 use the same unsupervised mode, but as P(output | input , task) – this is called **task conditioning** where the model is expected to produce different outputs for the same input and different tasks
- Task conditioning forms the ground for **zero-shot task transfer**
- **Zero-shot learning** is a special case of zero shot task transfer where no examples are provided at all
  - The model understands the task based on the given instruction
  - Input is given in a format to help the model understand the nature of the task
- Data sets: **Reddit**, **WebText** and all **Wikipedia** articles
- 1.5 billions of parameters, 50,257 tokens, larger batch size (512)

GPT-3

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

- 175 billion parameters
- Learning objectives and concepts:
  - **In-context learning**: When presented with few examples (or a description of what it needs to do), the language models matches the pattern of the examples with what it had learnt in past for similar data and uses that knowledge to perform the tasks
  - **Few-shot, one-shot and zero-shot setting**: specialized case of zero-shot task transfer
- Data set: Common Crawl, WebText2, Books1, Books2 and Wikipedia
- 96 layers and 96 attention heads for each layer
- Context window size was increased from 1024 for GPT-2 to 2048 tokens for GPT-3
- Accomplish tasks for what it was not trained (generate SQL comands, comprehension reads, etc)

GPT-4

- It is able to provide image synthesis but not image generation
- In the side example, one can realize that this new feature is not working appropriately, yet!!!!

LU

given this image here https://img.freepik.com/free-vector/celebratory-balloons-isolated-background_1284-52282.jpg?w=360 , write a caption

Based on the image you provided, here is a possible caption:

"A festive arrangement of colorful balloons isolated on a white background, creating a joyful and celebratory atmosphere."

LU

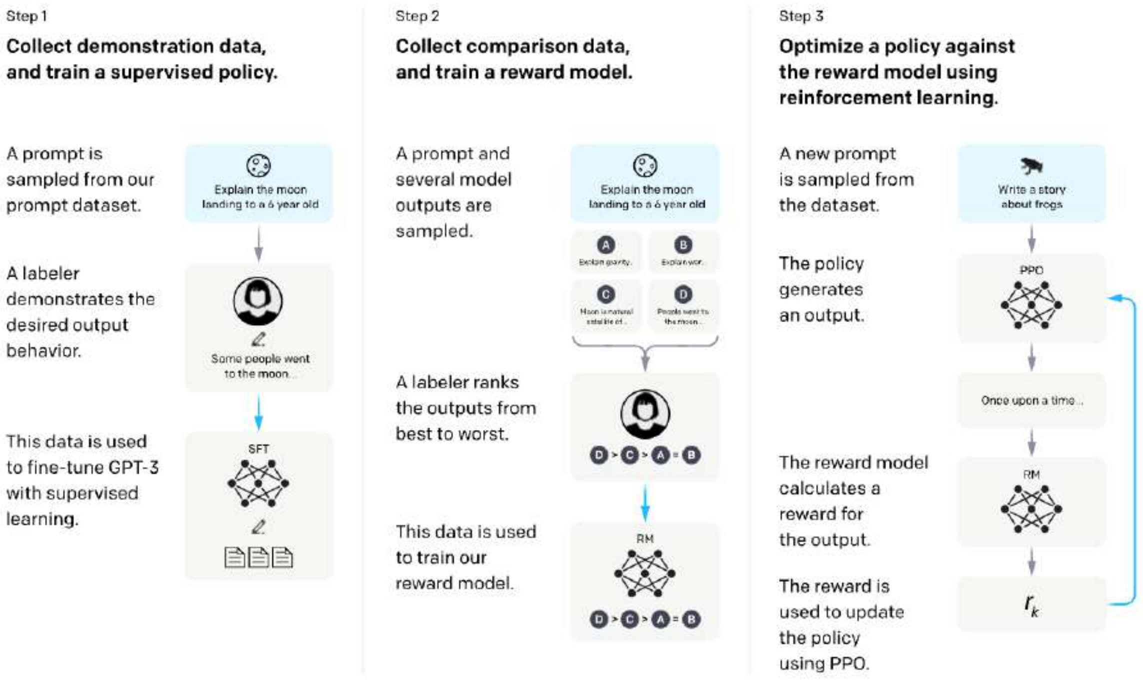
how many balloons in that image?

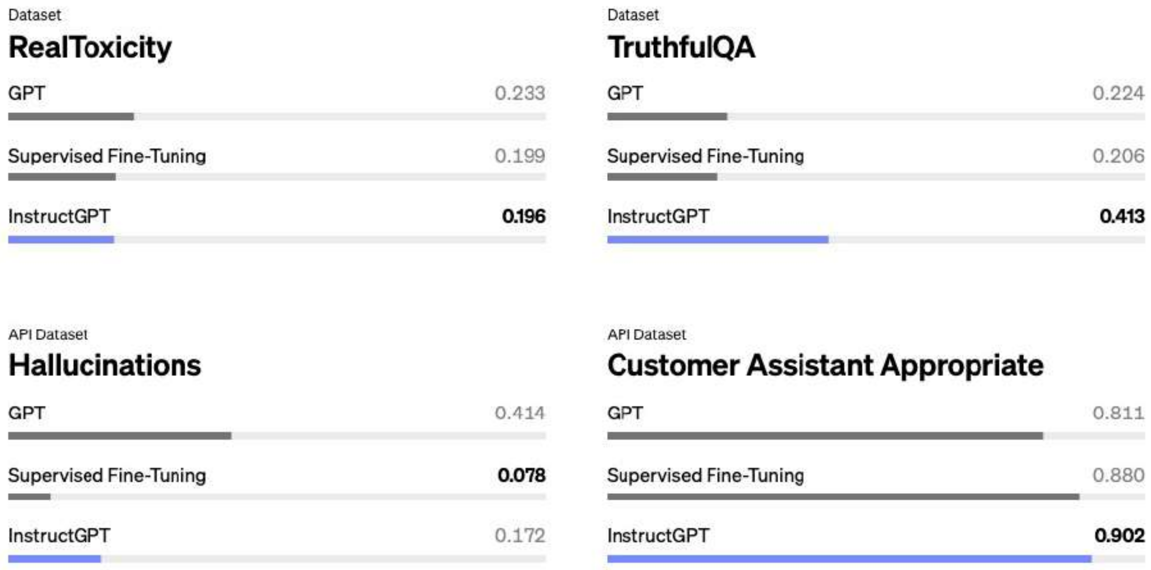
As an AI language model, I do not have the ability to directly analyze images or count the number of balloons in the image you provided. However, based on a visual inspection of the image, it appears that there are around 14 balloons in the arrangement. Please note that this is just an estimation based on a visual assessment and may not be entirely accurate.

14

ChatGPT

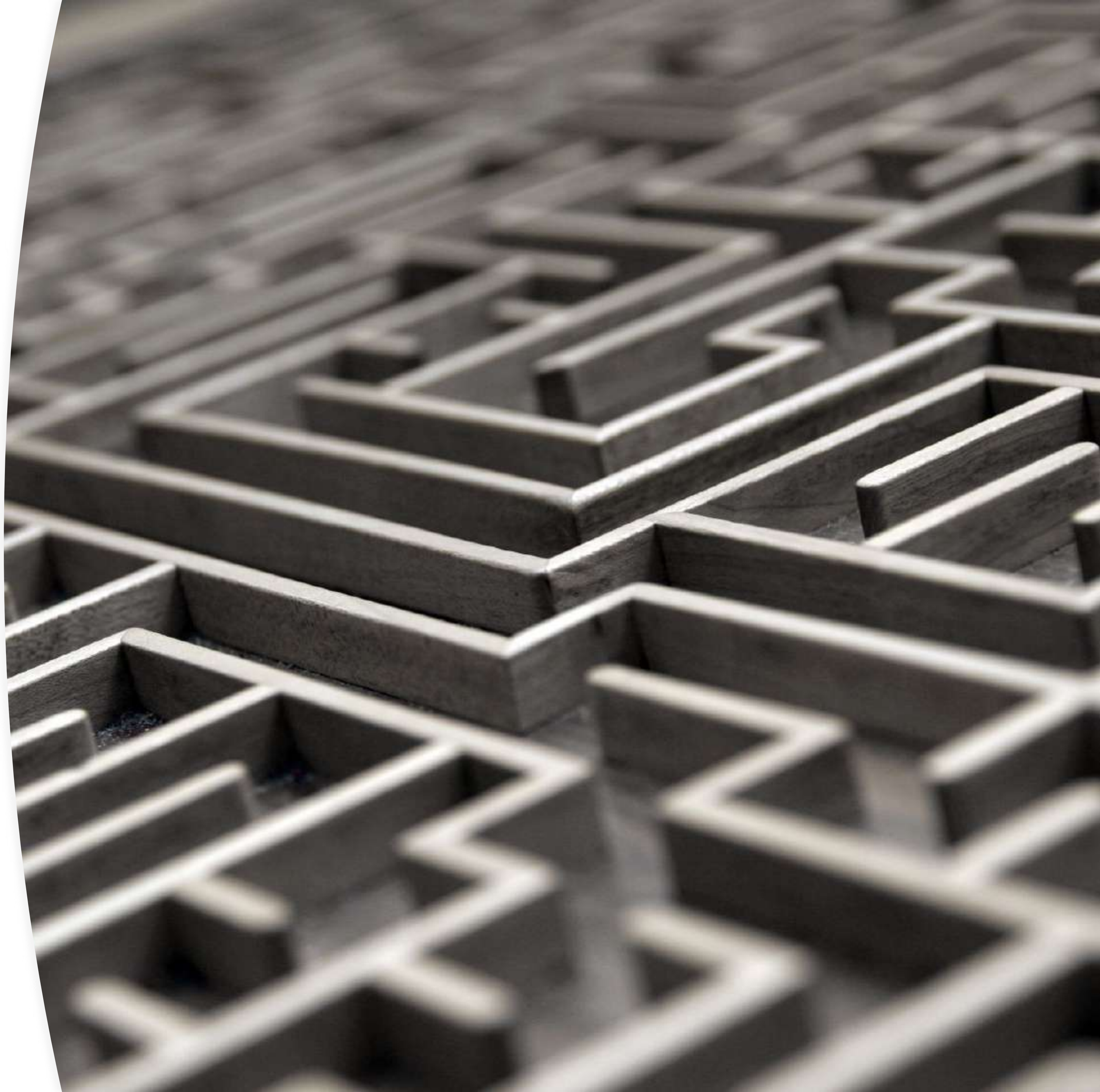
- Trained with **Reinforcement Learning from Human Feedback** (RLHF), based on Proximal Policy Optimization (PPO).
- Use **InstructGPT** to follow instructions
- ChatGPT and GPT-3.5 were trained on an **Azure AI** supercomputing infrastructure.





Some thoughts

- AI applied on text finally started achieving maturity to deal with big data
- Problems yet to solve are **toxicity** and **hallucination**
- If someone knows how to guide ChatGPT to answer the questions, it can make a surprising job. So, we must think about it as a **must-guided AI tool**
    - So, questions about oneself is useless. So do not try to make a guess about the potential of this tool making this kind of question
- ChatGPT is a bullshitter. It’s not a liar because to be a liar, you must know the truth and intend to mislead. ChatGPT is indifferent to the truth



Conclusions

- Modelling the probability distribution of a generative model is not an easy task, while requiring:
  - large computational resources
  - a lot of patience to efficiently modelling the generative side
- Understanding the fundamentals of each tecnique is of underlying importance to make it work, but not only… **It is necessary a lot of patience.**

Thank you

lrebouca@ufba.br
@ivisionlab
http://ivisionlab.ufba.br