Generative models

Luciano Oliveira
Universidade Federal da Bahia (UFBA)
Universidade Federal da Bahia
About the lab

The goal of IVisionLab is to provide real-world applications. Under this perspective, we drive our research from academic requirements to industry needs, visualizing the integrative relationship between these two worlds. Currently, our research topics are Smart Cities, Biometric Systems, Biomedicine, and Robotics.

Latest news

- **Papers in the International Symposium on Medical Information Processing and Analysis (I3MIA)**
  Published: Sep 16, 2021
- **Paper in Conference on Graphics, Patterns and Images (GPGI)**
  Published: Sep 14, 2021
- **Paper in IEEE Intelligent Vehicles Symposium**
  Published: May 28, 2021

Research in Computer Vision and Pattern Recognition to:

- Smart Cities
- Biometric Systems
- Biomedicine
- Robotics
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 model the generative state.
- Understanding the fundamentals of each technique 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 chatting with humans
  - Conducting natural conversations
  - Fully autonomous
  - Synchronization
  - Interrupt control
- Recurrent networks
Voice cloning

Deep speed estimation from synthetic and monocular data

João Paulo and Luciano Oliveira

Presented by:
Prof. Dr. Luciano Oliveira
lrebouca@ufba.br

Acknowledgments:
Automatic staining

- Staining of slide windows from H&E
- Use of GANs to generate slides with different staining patterns
A scene in a 3D disney/pixar style containing a yorkshire 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

\[ P(\text{dice}) = \frac{1}{6} = 16.67\% \]

\[ 0 \leq P(A) \leq 1 \]
\[ P(\emptyset) = 0 \]
\[ P(\Omega) = 1 \]

... 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= \(\text{cat}) = ? \]
P(X= \(\text{cat}) |Y) = ? \]
\[ or \]
P(Y|X= \(\text{dog}) = ? \]
P(X= \(\text{dog}) |Y) = ? \]
Bayes theorem

\[
P(Y|X) = \frac{P(Y)P(X|Y)}{P(X)} = \frac{P(X,Y)}{P(X)}
\]

posterior = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}
Bayes theorem

\[ P(y \mid x_1, x_2, x_3, \ldots) = \frac{P(y)P(x_1 \mid y)P(x_2 \mid y)P(x_3 \mid y)\ldots}{P(x_1)P(x_2)P(x_3)\ldots} \]
Discriminative models

\[ f: X \rightarrow Y \text{ or } P(Y \mid X) \text{ or } P(\text{label} \mid \text{features}) \]

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
    \[ \text{Depend on a latent variable} \]
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
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
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 – \( p(z_{dn} | \theta_d) \)
  - topics over words – \( p(w_{dn} | z_{dn}) \)

- Notation:
  - \( D \): total number of documents
  - \( \theta_d \): 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.

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

---


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 \( \mathbf{v} \)
2. Compute \( \mathbf{h} \)
3. Use \( \mathbf{h} \) to generate (samples of) visible states \( \mathbf{v}' \)
4. Use \( \mathbf{v}' \) to generate (samples of) hidden states \( \mathbf{h}' \)
5. Update the parameters \( W, b \) and \( c \) (where \( \varepsilon \) is the LR):

\[
\Delta W = \varepsilon (\mathbf{vh} - \mathbf{v}'h') \quad \Delta b = \varepsilon (\mathbf{v} - \mathbf{v}') \quad \Delta a = \varepsilon (\mathbf{h} - \mathbf{h}')
\]

\textbf{Gibbs sampling}: Sampling unknown parameters from a distribution while fixing the others.
Algorithm 1. \( k \)-step contrastive divergence

Input: RBM \((V_1, \ldots, V_n, H_1, \ldots, H_n)\), training batch \(S\)

Output: gradient approximation \(\Delta w_{ij}, \Delta b_j\) and \(\Delta c_i\) for \(i = 1, \ldots, n, j = 1, \ldots, m\)

1. init \(\Delta w_{ij} = \Delta b_j = \Delta c_i = 0\) for \(i = 1, \ldots, n, j = 1, \ldots, m\)

2. for all the \(v \in S\) do
   3. \(v^{(0)} \leftarrow v\)
   4. for \(t = 0, \ldots, k - 1\) do
      5. for \(i = 1, \ldots, n\) do sample \(h_i^{(t)} \sim p(h_i | v^{(t)})\)
      6. for \(j = 1, \ldots, m\) do sample \(v_j^{(t+1)} \sim p(v_j | h^{(t)})\)
      7. for \(i = 1, \ldots, n, j = 1, \ldots, m\) do
         8. \(\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 | v^{(0)}.v_j^{(0)}) - p(H_i = 1 | v^{(k)}.v_j^{(k)})\)
         9. \(\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}\)
        10. \(\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 | v^{(0)}) - p(H_i = 1 | v^{(k)})\)

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-tuning:
  - 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 useful 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: $\log p_{\phi}(x | z)$ 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: \( z_i \sim p(z) \)
  - Find \( x_i \sim p(x \mid z) \)
- The latent variables are found from \( p(z) \)
- Model inference will be:

\[
p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)} = \frac{\int p(x \mid z)p(z)dz}{p(x)}
\]
Variational autoencoders

- Reparametrization trick:

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

- Sampling from mean and standard deviation vector, instead of from the latent variables
Variational autoencoders

- As \( p(x) \) is costly, the posterior is approximated to a family of distributions \( \lambda : q_\lambda(z|x) \)
  - 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 ascent 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 (Gaussian/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

User edits

Generated images
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 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

**Adaptive Loss**

**Discriminator:**

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

**Generator:**

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

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

\[
\min_G \max_D \left[ \mathbb{E}_{(X \sim p(X))} \left[ \log D(X) \right] + \mathbb{E}_{(Z \sim p(Z))} \left[ \log (1 - D(G(Z))) \right] \right]
\]

Nash equilibrium:

\[p_{\text{data}}(x) = p_{\text{gen}}(x) \forall x\]

\[D(x) = \frac{1}{2} \forall x\]
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

\begin{center}
\begin{tikzpicture}
\node (input) at (0,0) {Random Input};
\node (generator) at (3,0) {Generator};
\node (discriminator) at (6,0) {Discriminator};
\node (sample) at (6,-2) {Sample};
\node (sample2) at (3,-2) {Sample};
\node (real_images) at (0,-2) {Real Images};
\draw[->] (input) -- (generator);
\draw[->] (generator) -- (sample);
\draw[->] (generator) -- (sample2);
\draw[->] (sample2) -- (discriminator);
\draw[->] (real_images) -- (discriminator);
\end{tikzpicture}
\end{center}

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:
  \[
  \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D \left( G \left( z^{(i)} \right) \right) \right).
  \]
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:
  \[ e_{\text{best}} = \arg\max_e P(e \mid f) \]

Example:

```
Ele  não  vai  para  casa
El   no   ba   a    casa
```
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
- 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

Is it a person or a machine?

Person A

Person B

Machine

Rule-based systems

Dictionary:
- The
- Red
- House
- Il
- Rosso
- Casa

Word order rules:
- Adjective + Noun <-> Noun + Adjective

The red house
- Il rosso casa
- La casa rossa

Il casa rosso

Dictionary lookup
- Reorder words

But it should be...
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

Translating with an Encoder-Decoder system

The red house

Encoder

Context vector [0.3, 0.6, -0.2, ..., 0.1]

Decoder

La casa rossa
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
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 states into a single context vector? More problems: Memory (RNNs require 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

What you will really find inside...
Tensor is all we need!
Self-attention
Self-attention

Input
Embedding
Queries
Keys
Values
Score

Thinking
$x_1$
$q_1$
$k_1$
$v_1$
$q_1 \cdot k_1 = 112$

Machines
$x_2$
$q_2$
$k_2$
$v_2$
$q_1 \cdot k_2 = 96$
Self-attention

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 \sqrt{k}$

Softmax

Thinking

$\begin{align*}
q_1 & \cdot k_1 = 112 \\
v_1 & = 14 \\
q_1 & = 0.88
\end{align*}$

Machines

$\begin{align*}
q_1 & \cdot k_2 = 96 \\
v_2 & = 12 \\
q_1 & = 0.12
\end{align*}$
Self-attention

Input
Embedding
Queries
Keys
Values
Score
Divide by $8 (\sqrt{d_k})$
Softmax
Softmax X
Value
Sum
Self-attention

\[
\begin{array}{c}
X \\
X \times W^Q = Q \\
X \times W^K = K \\
X \times W^V = V \\
Q \times K^T / \sqrt{d_k} \\
\text{softmax} \\
= Z
\end{array}
\]
Multi-head self-attention

Thinking Machines

Calculating attention separately in eight different attention heads

\[
\begin{align*}
Q_0 & \quad W_0^Q_{\text{ATTENTION HEAD #0}} & Q_1 & \quad W_1^Q_{\text{ATTENTION HEAD #1}} \\
K_0 & \quad W_0^K_{\text{ATTENTION HEAD #0}} & K_1 & \quad W_1^K_{\text{ATTENTION HEAD #1}} \\
V_0 & \quad W_0^V_{\text{ATTENTION HEAD #0}} & V_1 & \quad W_1^V_{\text{ATTENTION HEAD #1}} \\
\end{align*}
\]
Multi-head self-attention

1) Concatenate all the attention heads

2) Multiply with a weight matrix \( W^o \) that was trained jointly with the model

3) The result would be the \( Z \) matrix that captures information from all the attention heads. We can send this forward to the FFNN
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

ENCODERS

EMBEDDING WITH TIME SIGNAL

EMBEDDINGS

INPUT: Je suis étudiant

PREVIOUS OUTPUTS

DECODERS

Linear + Softmax
Which word in our vocabulary is associated with this index?

Get the index of the cell with the highest value (argmax)

```
log_probs
0 1 2 3 4 5
```

```
logits
0 1 2 3 4 5
```

Decoder stack output
Toy example

Target Model Outputs

Output Vocabulary: a am l thanks student <eos>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>am</th>
<th>l</th>
<th>thanks</th>
<th>student</th>
<th>&lt;eos&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos. 1</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>pos. 2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>pos. 3</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>pos. 4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>pos. 5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Ground truth

Trained Model Outputs

Output Vocabulary: a am l thanks student <eos>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>am</th>
<th>l</th>
<th>thanks</th>
<th>student</th>
<th>&lt;eos&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos. 1</td>
<td>0.01</td>
<td>0.02</td>
<td>0.93</td>
<td>0.01</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>pos. 2</td>
<td>0.01</td>
<td>0.8</td>
<td>0.1</td>
<td>0.05</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td>pos. 3</td>
<td>0.99</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.002</td>
<td>0.001</td>
</tr>
<tr>
<td>pos. 4</td>
<td>0.001</td>
<td>0.002</td>
<td>0.001</td>
<td>0.02</td>
<td>0.94</td>
<td>0.01</td>
</tr>
<tr>
<td>pos. 5</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.98</td>
</tr>
</tbody>
</table>

Trained model output
### Generative Pre-trained Transformer

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

\[
L_2(C) = \sum_{x,y} \log P(y|x_1, \ldots, x_n) \quad L_3(C) = L_2(C) + \lambda L_1(C)
\]
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 $P(\text{output} \mid \text{input})$
• GPT-2 use the same unsupervised mode, but as $P(\text{output} \mid \text{input}, \text{task})$ – 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)
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


- 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!!!!
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 technique 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