

# Introduction à la notion de « plongement de mot » (*word embedding*)

Thierry Poibeau  
thierry.poibeau@ens.psl.eu

Master Humanités numériques, PSL, 2025

# Principales sources de ce support

- Sur les modèles plus anciens d'analyse distributionnelle
  - Distributional Semantic Models, S. Evert
  - [https://esslli2016.unibz.it/wp-content/uploads/2015/10/dsm\\_tutorial\\_part1.slides.pdf](https://esslli2016.unibz.it/wp-content/uploads/2015/10/dsm_tutorial_part1.slides.pdf)
- Sur BERT
  - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Bidirectional Encoder Representations from Transformers) Jacob Devlin
  - <https://nlp.stanford.edu/seminar/details/jdevlin.pdf> (contient beaucoup plus d'information que dans ces transparents)

# Plan

- Analyse distributionnelle
- Plongements de mots
- Visualisation de plongements de mots
- Bert
- Jeux de données disponibles
- Evaluation
- Conclusion

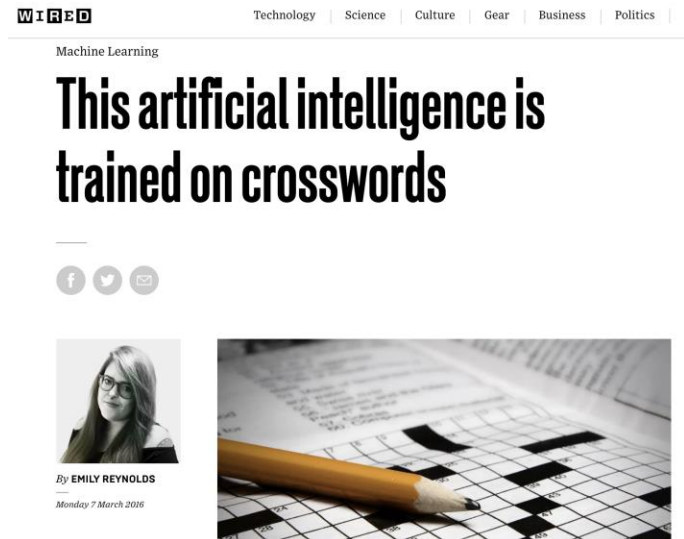
# Analyse distributionnelle et modèles vectoriels

# L'hypothèse distributionnelle

- Hypothèse distributionnelle : mots apparaissant dans contextes similaires = signification similaire (Harris 1954)
  - Difference of meaning correlates with difference of distribution —(Zellig Harris 1954)
  - “Die Bedeutung eines Wortes liegt in seinem Gebrauch.” — (Wittgenstein)
  - “You shall know a word by the company it keeps” — (Firth, 1957)
  - “What people know when they say that they know a word is not how to recite its dictionary definition – they know how to use it (...) in everyday discourse.” (Miller 1986)

# Exemples

- Retrouver un mot par son contexte (cad par des mots qui le caractérisent, cf. mots croisés)
  - Concombre, sauce, pizza, ketchup ⇒ **tomate**
  - Parole, morceau, musique, populaire ⇒ **chanson**



# Représentation vectorielle

- On peut représenter le sens d'un mot par les mots apparaissant dans son contexte proche

	song	cucumber	meal	black
tomato	0	6	5	0
book	2	0	2	3
pizza	0	2	4	1

- Chaque mot est associé à un vecteur de dimension  $|V|$  (la taille du vocabulaire)
- On voit que *tomato* est assez proche de *pizza* mais pas de *book* (d'après les valeurs dans les colonnes)

# Représentation vectorielle

- On s'attend à ce que des mots sémantiquement similaires aient des vecteurs similaires
- Etant donné la représentation vectorielle de deux mots donnés, on peut déterminer leur similarité (cf. *infra*)
- La même représentation peut être utilisée à différents niveaux : mot, syntagme, document, *etc.*



# Prétraitements

- Un simple comptage est inefficace
  - Les mots les plus fréquents sont peu informatifs
- D'autres techniques sont généralement utilisées
  - Suppression des mots sémantiquement vides
  - Pondération tf.idf
  - Information mutuelle ( $PMI = p(x,y) / p(x)p(y)$ )
  - Cf. cours précédents

# « Densifier » la représentation

- Deux problèmes avec une représentation vectorielle « brute »
  - Les vecteurs sont trop grands (par défaut, taille vecteur = taille du vocabulaire à modéliser)
  - Ils sont « creux » (la plupart des valeurs = 0)
- Deux solutions possibles
  - Réduire la taille des vecteurs (il y a des algorithmes pour ça, par ex. SVD)
  - « Apprendre » directement des vecteurs de taille réduite  $\Rightarrow$  word embedding)

Plongements de mots  
(= *Word embeddings*)

# Plongements de mots (*Word embeddings*)

- Chaque mot est représenté par un vecteur de faible dimension (typiquement, 300 ou 500 valeurs)
- Tous les mots sont modélisés dans un même espace sémantique
- Les mots sémantiquement similaires ont normalement des vecteurs similaires (= leur vecteurs sont proches dans l'espace sémantique)
- Les plongements de mots sont le « cœur » de beaucoup d'applications de TAL actuellement

# « Vectoriser » le sens des mots

- Encodage du sens des mots dans des vecteurs de taille fixe (typiquement, dimension 300 à 500). Chaque case représente un ensemble de contextes, c'est-à-dire une notion sémantique latente



Source : Blog d'Adrian Acolier,  
*The morning paper* :  
<https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/>.

- Calcul sur les vecteurs :  $King - Man + Woman = Queen$

# Word2Vec

- Modèle ayant popularisé la notion de word embedding en TAL
- Modèle relativement simple et efficace
  - Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
  - Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (NIPS 2013).

# Word2Vec

- Hypothèse : les mots sémantiquement similaires ont un contexte sémantiquement similaire
  - $X \simeq Y \Rightarrow \text{Contexte}(X) \simeq \text{Contexte}(Y)$
- Modèle de calcul efficace pour obtenir des plongements de mots
  - Simplifier la notion de contexte
  - Repérer les contextes similaires
  - Ajuster leur représentation (poids associé à chaque mot) au fur et à mesure de l'apprentissage (*backpropagation*)
  - Le contenu d'une dimension est arbitraire, mais peut correspondre à une notion sémantique

# Paramétrage

- Analyse sac de mots ou souvent stratégie plus élaborée, de type *skipgram* par ex.
- Autres paramètres
  - Augmenter le poids des mots les plus proches du mot cible dans le contexte
  - Supprimer (ou baisser le poids des) mots vides et/ou très fréquents
- Prise en compte des mots composés
  - Pas possible par défaut avec Word2vec (*New* et *York* dans *New York* sont deux mots simples)
  - Solution : prétraitement pour identifier les mots composés les plus fréquents (par information mutuelle, PMI, par ex.) alors considérés comme un seul token



# Package Word2vec

- On trouve facilement des Jupyter notebook autour de Word2vec
- Le code est aussi disponible ici : (répertoire de Mikolov) :  
<https://github.com/tmikolov/word2vec>
  - Récupérer le corpus par défaut (wget <http://mattmahoney.net/dc/text8.zip>) ou n'importe quel corpus

# Les paramètres (ou hyperparamètres) jouent un rôle important

- Taille de la fenêtre prise en compte pour le calcul

Word: **walk**

Window size = 3

Word	Cosine distance
go	0.488083
snipe	0.464912
shoot	0.456677
fly	0.449722
sit	0.449678
pass	0.442459
climbs	0.440931
walked	0.436502
ride	0.434034
stumble	0.426750
bounce	0.425577
travelling	0.419419
walking	0.412107
walks	0.410949
trot	0.410418
leaping	0.406744
sneak	0.401918
climb	0.399793
move	0.396715
wait	0.394463
going	0.391639
shouted	0.388382
roam	0.388073
thrown	0.384087
get	0.383894

Window size = 30

Word	Cosine distance
walking	0.486317
walked	0.430764
walks	0.406772
stairs	0.401518
go	0.399274
sidewalk	0.385786
stand	0.380480
cortege	0.371033
wheelchair	0.362877
strapped	0.360179
hollywood	0.356544
carousel	0.356187
grabs	0.356007
swim	0.355027
breathe	0.354314
tripped	0.352899
cheer	0.352477
moving	0.350943
inductees	0.347791
walkway	0.347164
shout	0.346229
pounding	0.340554
blvd	0.339121
crowd	0.338731
levada	0.334899

# Les paramètres (ou hyperparamètres) jouent un rôle important

- Nombre d'itérations

Word: **walk**

No. of iterations = 1

Word	Cosine distance
walking	0.851438
walks	0.846485
bat	0.843796
ride	0.830734
crowd	0.821692
quiet	0.812538
spot	0.802777
steal	0.787917
door	0.787571
doors	0.786485
bed	0.773686
dinner	0.772160
shadow	0.769573
luck	0.768221
baby	0.767862
shoot	0.765968
walked	0.765739
sitting	0.765394
shirt	0.759116
rides	0.759047
watching	0.755140
watch	0.750808
gehrig	0.741494
shoots	0.740971
looking	0.740904

No. of iterations = 100

Word	Cosine distance
walked	0.483473
ride	0.470925
walks	0.470889
stand	0.449993
walking	0.449071
go	0.430172
shoot	0.421110
get	0.404258
move	0.403757
live	0.403347
fly	0.400929
climbs	0.396346
throw	0.391768
climb	0.384038
wiggle	0.380892
thrown	0.380426
pull	0.375478
goes	0.375406
moving	0.374447
pass	0.372463
conversing	0.364413
sit	0.362765
crowd	0.361651
kiss	0.359883
stay	0.357015

# Les paramètres (ou hyperparamètres) jouent un rôle important

- Taille des vecteurs de représentation utilisés

Word: **walk**

No. of dimensions = 5

Word	Cosine distance
catcher	0.998074
shirt	0.996589
lechuck	0.995313
bullseye	0.994644
bowler	0.994381
punter	0.993154
lovell	0.992815
heels	0.992255
whip	0.992085
outfit	0.992047
tore	0.991924
steals	0.991524
guybrush	0.991166
gigs	0.990291
hanging	0.990201
burns	0.990043
backing	0.989966
orser	0.989960
torch	0.989747
beat	0.989435
showdown	0.989381
feat	0.989242
cheers	0.988951
clad	0.988646
lunch	0.988326

No. of dimensions = 1000

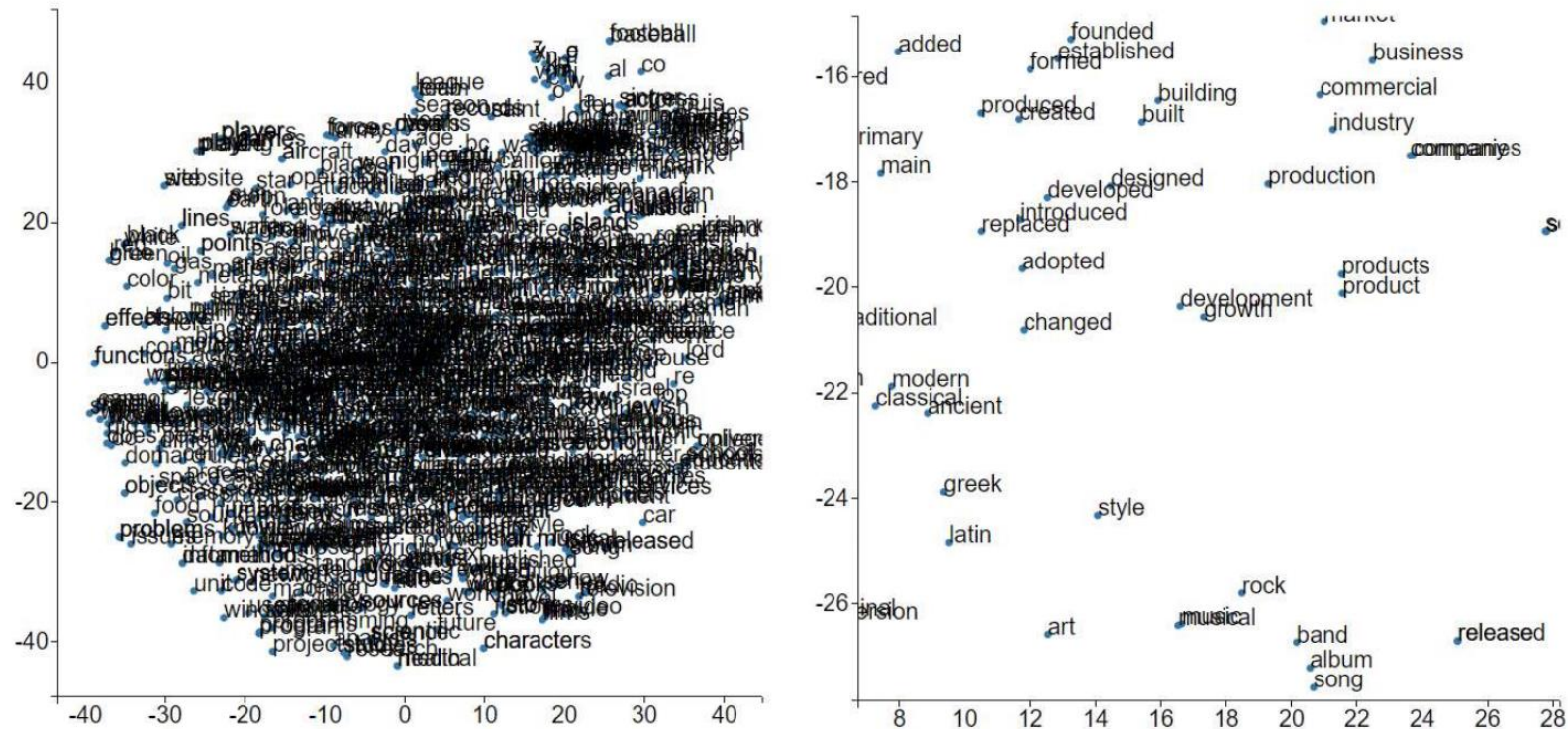
Word	Cosine distance
walks	0.304954
walked	0.303322
snipe	0.287221
walking	0.272690
ride	0.266770
canter	0.251025
bandleaders	0.246454
climbs	0.233725
catapulted	0.230075
climb	0.229263
trot	0.228362
shouted	0.227306
stand	0.223288
seagulls	0.221745
fly	0.216602
fences	0.216366
lifts	0.215402
pray	0.214977
paws	0.214865
bounces	0.214449
shoot	0.213457
grabs	0.212018
walkway	0.211136
swim	0.209120
tumble	0.207765

Visualiser le contenu d'un *Word  
Embedding* obtenu avec Word2vec

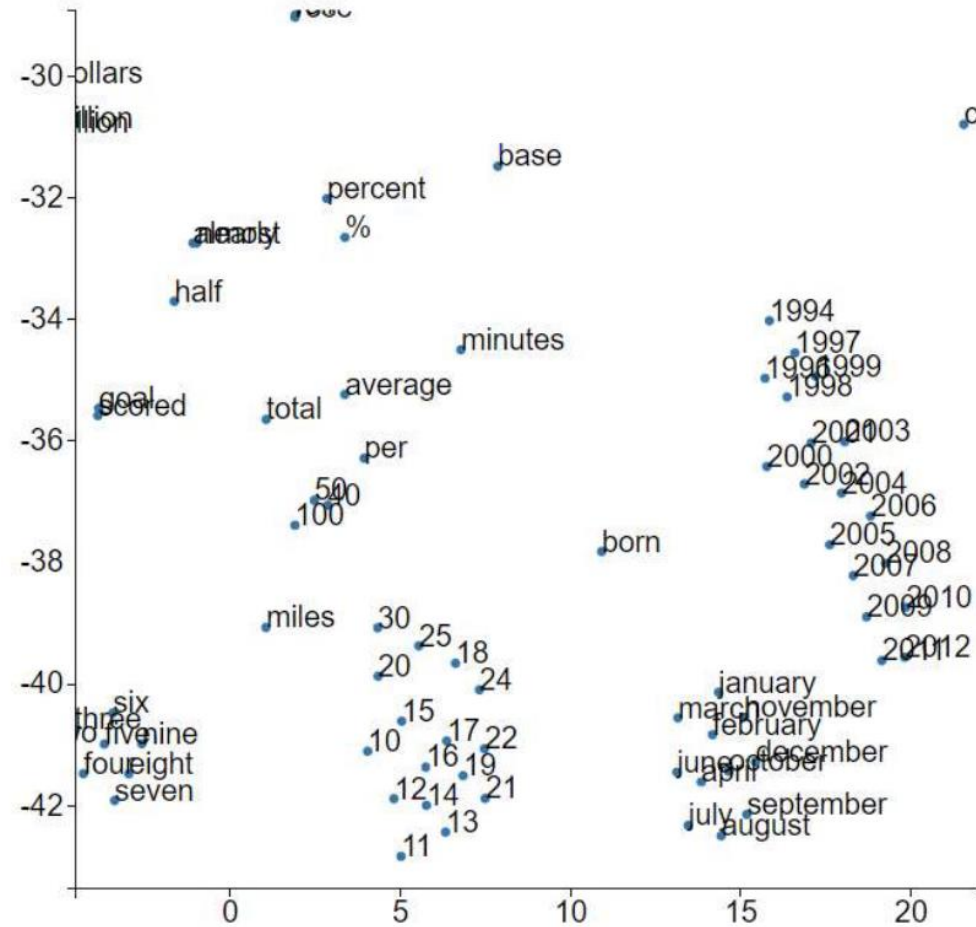
# Visualiser les word embeddings

- Comment « voir » le contenu d'un Word embedding ?
  - Impossible / difficile directement (vecteur = juste un ensemble de chiffres. Pas de proximité entre mots immédiatement lisible)
  - Nécessité de recourir à des outils (même si, en pratique, on va rarement voir le contenu d'un word embedding)
- *Visualizing Data using t-SNE*, Maaten and Hinton, 2008

# Visualisation : Word2Vec



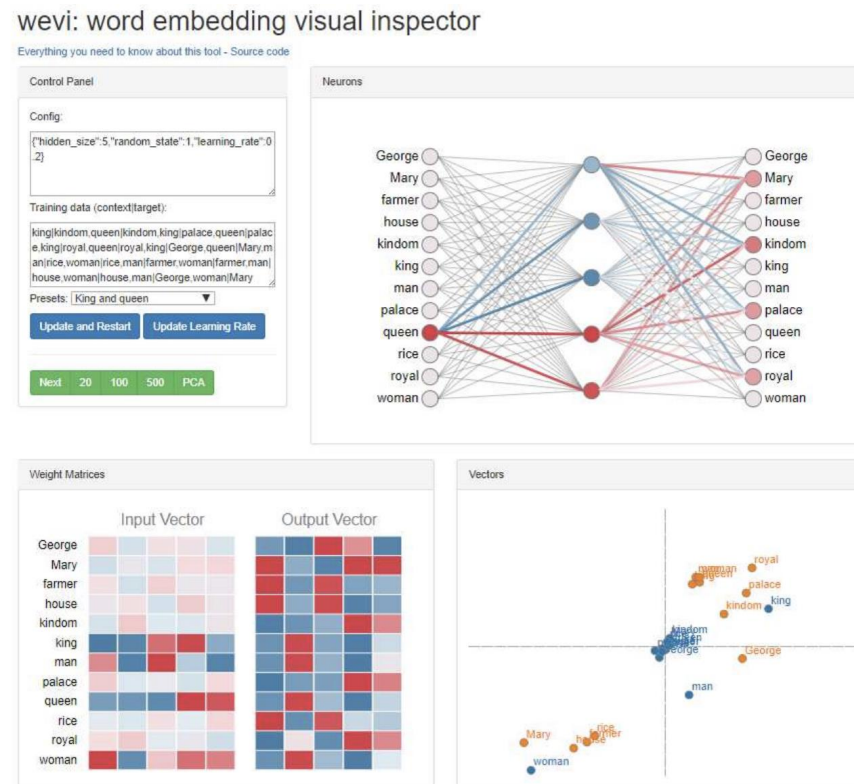
# Visualisation : Glove





# Un outil plus spécialisé : Wevi

- Wevi permet d'observer le contenu d'un Word embedding : <https://ronxin.github.io/wevi/>



# wevi: word embedding visual inspector

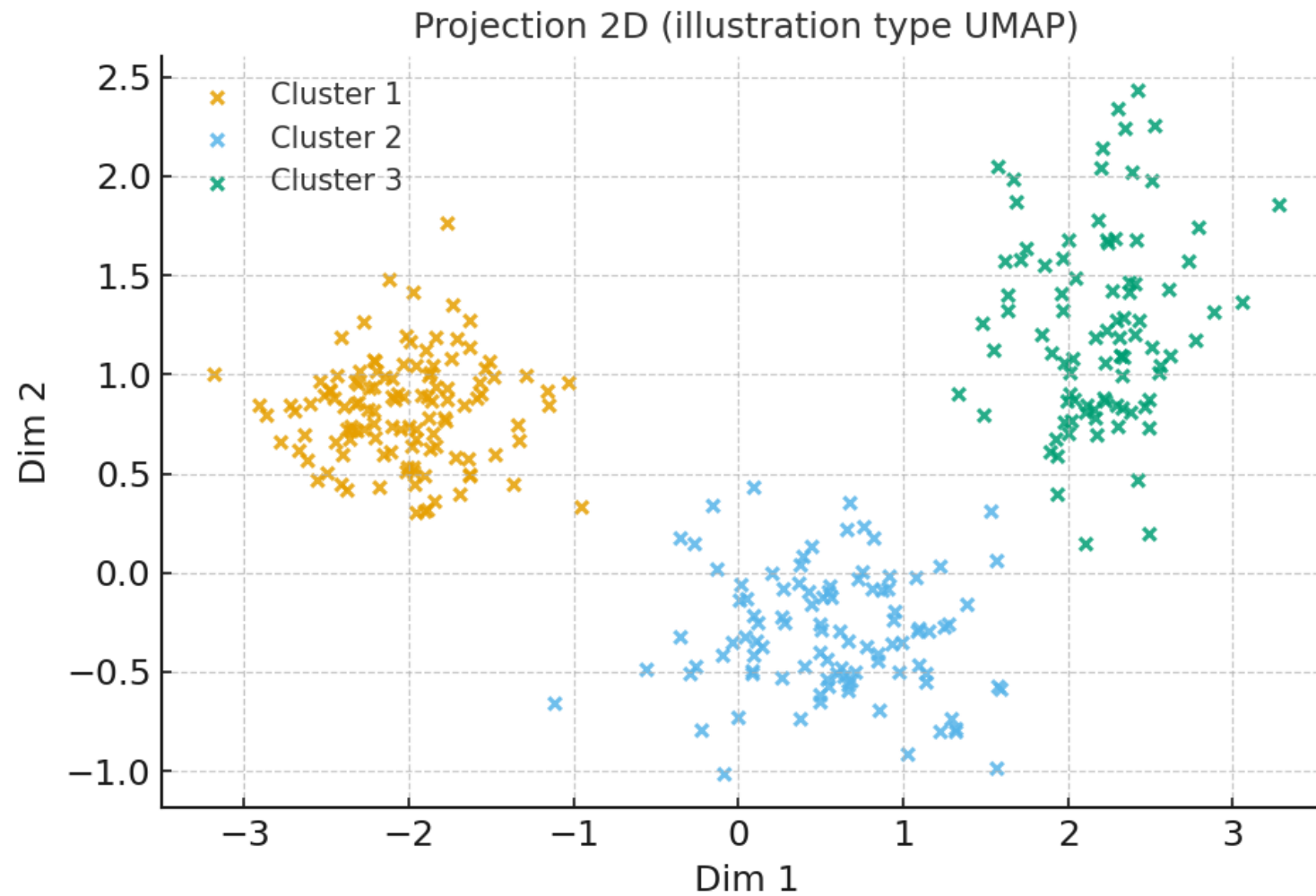
Everything you need to know about this tool - [Source code](#)



# UMAP

- UMAP (Uniform Manifold Approximation and Projection) est une méthode de réduction de dimension
- Conserve la structure locale des données en les projetant dans un espace 2D/3D
- Modélise les voisinages (graphes de k plus proches voisins) et préserve ces liens
- Avantages : rapide, non linéaire, souvent plus lisible que t-SNE pour les grands jeux de données

# Exemple de représentation avec UMAP



# UMAP vs PCA / t-SNE

- PCA : linéaire, très rapide, conserve surtout la variance globale.
- t-SNE : non linéaire, bon pour les structures locales, mais plus lent et paramètres sensibles.
- UMAP : non linéaire, conserve bien les voisinages, plus stable et scalable que t-SNE en pratique.

# Paramètres UMAP essentiels

- `n_neighbors` : taille du voisinage local (plus grand → structure globale ; plus petit → local)
- `min_dist` : densité des amas (plus petit → points serrés ; plus grand → plus espacés)
- `metric` : mesure de distance (euclidean, cosine, etc.)
- Astuce : tester rapidement quelques valeurs (p. ex. `n_neighbors=5, 15, 50` ; `min_dist=0.1, 0.5`).

BERT

# BERT

- BERT: *Bidirectional Encoder Representations from Transformers*
  - Modèle mis au point par Google en 2018
  - Modèle très populaire = a permis des améliorations importantes en termes de performances pour la plupart des tâches de TAL
  - Aujourd'hui utilisé dans de très nombreuses applications de TAL
  - Existe pour l'anglais à l'origine, puis d'autres langues + version multilingue
- Des modèles plus récents (Mistral, Llama) sont devenus plus populaires suite à ChatGPT, mais Bert reste d'actualité pour l'analyse (robuste, plus léger, disponible en français...)



# BERT

- Points clés
  - BERT peut considérer tous les mots de la phrase comme contexte alors que les modèles précédents se limitaient généralement à une fenêtre autour du mot à modéliser
  - BERT peut distinguer les différents sens d'un mot (jusqu'à un certain point)

# BERT en pratique

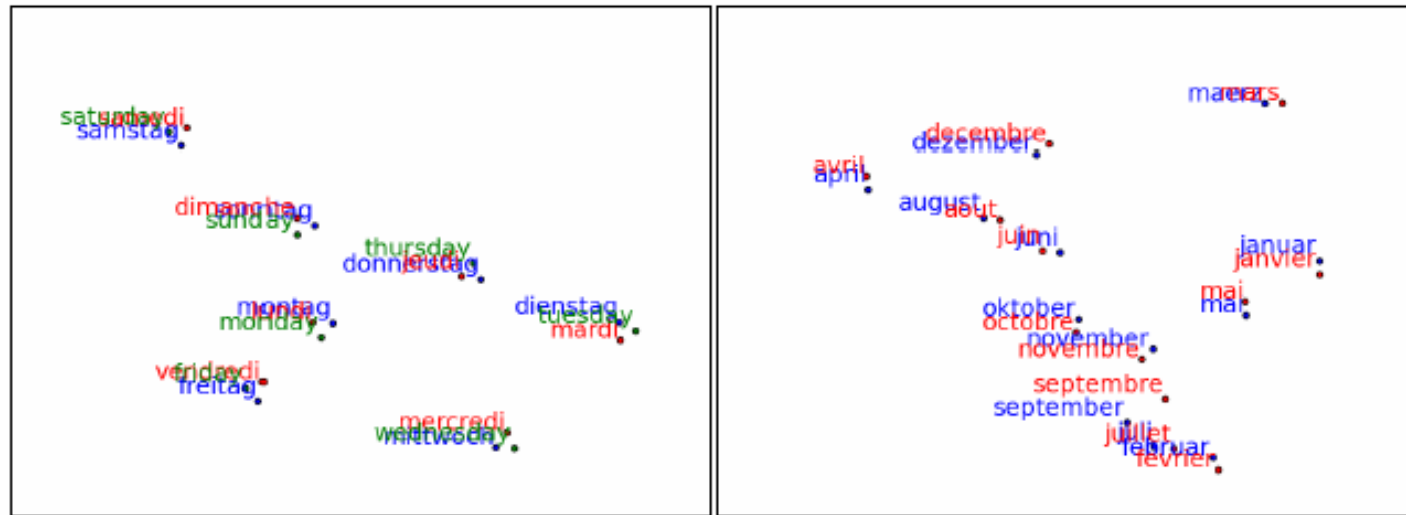
- Reprend l'idée du « cloze test » de Word2Vec
  - « Masquage » de certains mots
  - Tests pour essayer de les « identifier » dynamiquement d'après leur contexte
  - Jusqu'à obtenir des représentations fiables
- Utilise ensuite la notion de « transformeur » (le T de Bert, GPT, ...) pour calculer une représentation dynamique en fonction du contexte
- Voir des présentations en ligne pour les détails techniques
  - Par exemple : <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>

# BERT en pratique

- Le modèle original de Google
  - Entraîné sur Wikipedia (2.5 milliards de mots) + un corpus issu de Google Books
  - Grand à l'époque, très petit aujourd'hui
- On dispose de corpus d'entraînement de milliards de mots pour de nombreuses langues (cf. corpus Common Crawl)
  - Débauche de calcul et d'énergie !
  - Heureusement, on peut utiliser des modèles disponibles (pré-entraînés)

# Plongements de mots multilingues

- Utile pour la traduction ou l'analyse multilingue
  - Les différentes langues partagent le même espace sémantique
  - Les mots de même sens occupent des positions rapprochées dans l'espace




Source : Hermann and Blunsom, 2013. Cf. <https://arxiv.org/abs/1312.6173>

# Pourquoi le succès de Bert ?

- Bert encode de très nombreuses informations (au-delà de la simple proximité sémantique)
  - Éléments de contexte local et moins local
  - Analyse syntaxique et sémantique
  - Entités nommées, etc.
- Mais il est très difficile de savoir ce qui est encodé exactement dans ce type de modèle
  - Question de recherche active
  - Limite de ces modèles (ils sont très puissants mais ont un côté « boîte noire » difficile à contourner)

# BERT vs GPT

- Objectif : prédire les mots manquants dans une phrase incomplète.
  - Exemple : « Le chat \_\_\_\_ sur le canapé » → BERT prédit : “dort”.
- Entraînement bidirectionnel : le modèle lit à gauche et à droite du mot masqué → il prend en compte le contexte complet d’une phrase.
- Utilisation principale : Analyse de texte, classification, recherche d’information, reconnaissance d’entités, etc.
- Type d’architecture : encodeur (Transformer encoder).
-  BERT ne génère pas de texte (au sens propre) : il comprend et classe, mais ne produit pas de phrases complètes.

# GPT vs BERT

- Objectif : prédire le mot suivant dans une séquence de texte
  - Exemple : « Le chat dort sur » → GPT prédit : “le canapé”.
- Entraînement unidirectionnel : le modèle ne regarde que vers la gauche
- GPT = Generative Pretrained Transformer (OpenAI, depuis 2018), Llama, Gemini, Claude, Deepseek, Mistral...
- Utilisation principale : Rédaction, dialogue, résumé, traduction, raisonnement, code, etc.
- Type d'architecture : décodeur (Transformer decoder).
- GPT est un modèle génératif : il produit du texte mot par mot, de façon cohérente (y compris des annotations)

# Succès des modèles GPT

- Les modèles de type GPT sont dits génératifs :
  - Leur atout principal : ils fonctionnent sans entraînement spécifique.
  - Il suffit d'un prompt (instruction en langage naturel) pour obtenir une réponse.
- Cela les rend extrêmement polyvalents
  - Ils peuvent s'adapter à de nombreuses tâches : écriture, analyse, aide à la recherche, prototypage, etc.
- Mais : leurs sorties dépendent d'un apprentissage massif, souvent opaque, et d'une logique de probabilités plutôt que de vérification
  - Ils sont puissants, mais parfois peu fiables ou peu reproductibles



# Intérêt de BERT pour les SHS

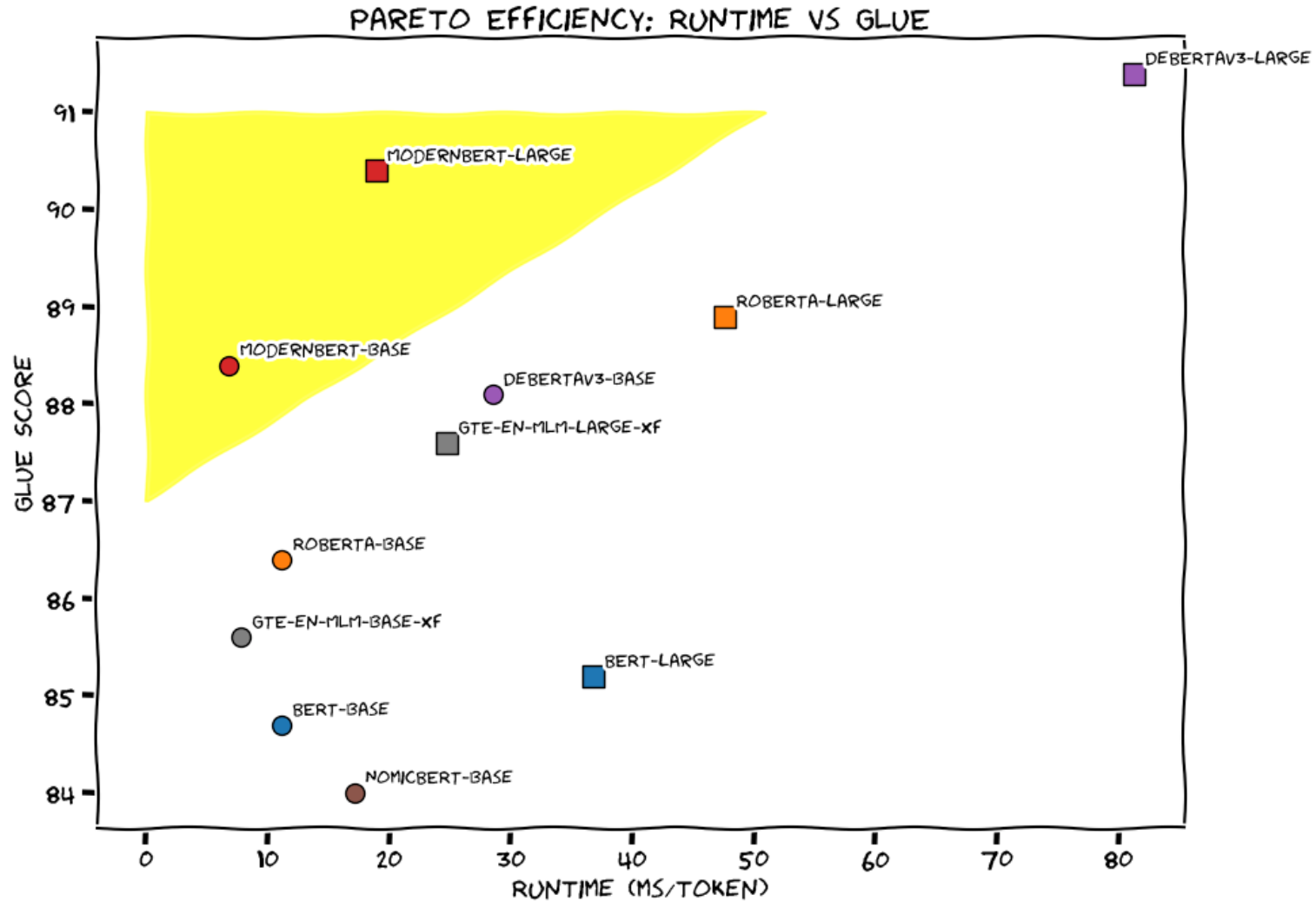
- BERT n'est pas génératif : il « comprend » le texte, mais ne le produit pas
  - Il sert à analyser, classier, comparer, extraire des entités, etc.
- De nombreux modèles dérivés (CamemBERT, FlauBERT, etc.) ont été entraînés sur des corpus francophones ou spécialisés
  - Voir aussi ModernBERT
  - <https://huggingface.co/blog/modernbert>
- Il offre une meilleure interprétabilité et adaptabilité à la recherche académique.

Jeux de données disponibles

# BERT

- BERT
  - Pour l'anglais, puis de nombreuses autres langues
    - <https://github.com/google-research/bert>
  - Pour le français
    - Flaubert <https://github.com/getalp/Flaubert>
    - CamemBert : <https://camembert-model.fr/>
  - Modèle multilingue (de Google)
    - <https://github.com/google-research/bert/blob/master/multilingual.md>
- En pratique, les modèles sont intégrés à des plateformes comme Spacy ou HuggingFace qui en facilitent l'accès (qq lignes de code)

# ModernBERT



# Coût environnemental

- BERT : environ 110 à 340 millions de paramètres.
  - Entraînement initial sur 4 jours avec 64 TPU v2. Idéal pour les tâches de classification et d'analyse linguistique. Peu coûteux en calcul, et facilement réentraînable localement.
- GPT (ex. GPT-3) : 175 milliards de paramètres.
  - Entraînement estimé à environ  $3 \times 10^{23}$  FLOPs, soit plusieurs millions de dollars de coût matériel. Nécessite des grappes massives de GPU A100.
- L'écart de coût est d'un facteur de 1 000 à 10 000, selon la taille et l'usage (pour des différences de performance parfois modestes)
  - Les modèles GPT consomment plusieurs MWh d'électricité par phase d'entraînement et restent très onéreux à exécuter en production.

Evaluation, limites et remarques finales

# Capacité des modèles à repérer des relations sémantiques


- Similarité  $\approx$  « corrélation » entre mots ?
  - walk – walking, walk – run, walk – stroll
  - Germany – Berlin, Germany – England
  - dog – cat, dog – Labrador, dog – leash
- Modèles capables de retrouver des régularités linguistiques, voire des connaissances de sens commun
  - Cf. Mikolov (calculs évidemment sur les vecteurs et non sur les mots)
  - “king” - “man” + “woman” = “queen” (en fait un peu bidon...)
  - “mice” - “mouse” + “door” = “doors” (sg. / pl.)

# Evaluation

- Evaluer la qualité d'un plongement de mots est un problème ouvert
  - Qu'est-ce qui est encodé ?
  - Syntaxe ? (ou juste du par cœur ?)
  - Sémantique ? (oui, mais sous quelle forme ?)
- En pratique, utilisation de benchmarks avec des ensembles de tests (les modèles sont multitâches)
  - Glue, SuperGLue, etc. (<https://super.gluebenchmark.com/>)



# BERT pour une langue à morphologie riche

 Cornell University

We are the Simons Foundation

arXiv.org > cs > arXiv:1912.07076

Search...  
Help | Advanced

Computer Science > Computation and Language

[Submitted on 15 Dec 2019]

**Multilingual is not enough: BERT for Finnish**

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, Sampo Pyysalo

Deep learning-based language models pretrained on large unannotated text corpora have been demonstrated to allow efficient transfer learning for natural language processing, with recent approaches such as the transformer-based BERT model advancing the state of the art across a variety of tasks. While most work on these models has focused on high-resource languages, in particular English, a number of recent efforts have introduced multilingual models that can be fine-tuned to address tasks in a large number of different languages. However, we still lack a thorough understanding of the capabilities of these models, in particular for lower-resourced languages. In this paper, we focus on Finnish and thoroughly evaluate the multilingual BERT model on a range of tasks, comparing it with a new Finnish BERT model trained from scratch. The new language-specific model is shown to systematically and clearly outperform the multilingual. While the multilingual model largely fails to reach the performance of previously proposed methods, the custom Finnish BERT model establishes new state-of-the-art results on all corpora for all reference tasks: part-of-speech tagging, named entity recognition, and dependency parsing. We release the model and all related resources created for this study with open licenses at [this https URL](#).

Subjects: **Computation and Language (cs.CL)**

Cite as: [arXiv:1912.07076 \[cs.CL\]](#)  
(or [arXiv:1912.07076v1 \[cs.CL\]](#) for this version)

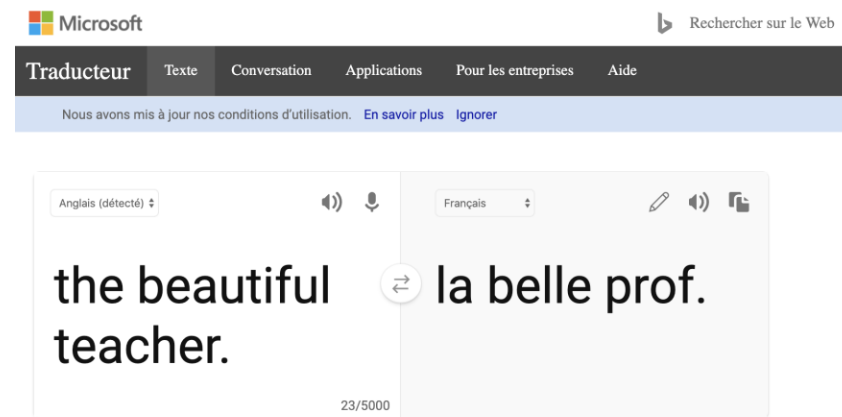
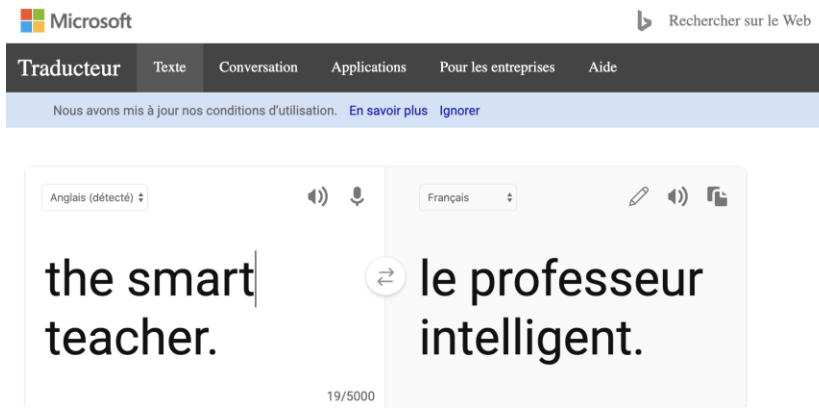
**Submission history**

From: Sampo Pyysalo [[view email](#)]  
[v1] Sun, 15 Dec 2019 17:50:56 UTC (148 KB)

# Limites et évolutions

- Défauts de Word2vec
  - Une seule représentation par mot (même si mot ambigu)
  - Pas de traitement de la morphologie
- BERT est plus précis
  - Adaptable à différentes tâches (entités nommées, analyse de sentiments) et domaines
  - Possibilités de le réentraîner
- Modèles plus récents (de type GPT) plus performants
  - Modèles génératifs (Mistral, Llama...)
  - Production d'annotation vue comme une tâche de génération

# Biais et questions éthiques



- *Beautiful teacher* = traduction au féminin !
- *Smart teacher* = traduction au masculin !

# Biais et questions éthiques

- Les plongements de mots reflètent les biais des corpus sur lesquels ils ont été appris !
  - How to make a racist AI without really trying <https://gist.github.com/rspeer/ef750e7e407e04894cb3b78a82d66aed> (Blog, Robyn Speer 13/07/2017)
  - Lipstick on a Pig: Debiasing Methods Cover up Systematic Gender Biases in Word Embeddings But do not Remove Them. <http://u.cs.biu.ac.il/~gonenhi/> (Hila Gonen and Yoav Goldberg, NAACL 2019)

Conclusion

# Conclusion

- Plongements de mots
  - Un composant essentiel du TAL aujourd'hui
  - Modèles puissants, faciles à intégrer
  - Similarité entre mots, mais aussi bien d'autres types de connaissances
- Limites
  - Modèles (relativement) coûteux à entraîner
  - Aspect boîte noire (pas d'accès direct, difficulté d'interprétation)
  - De nombreux biais, coût environnemental des techniques liées