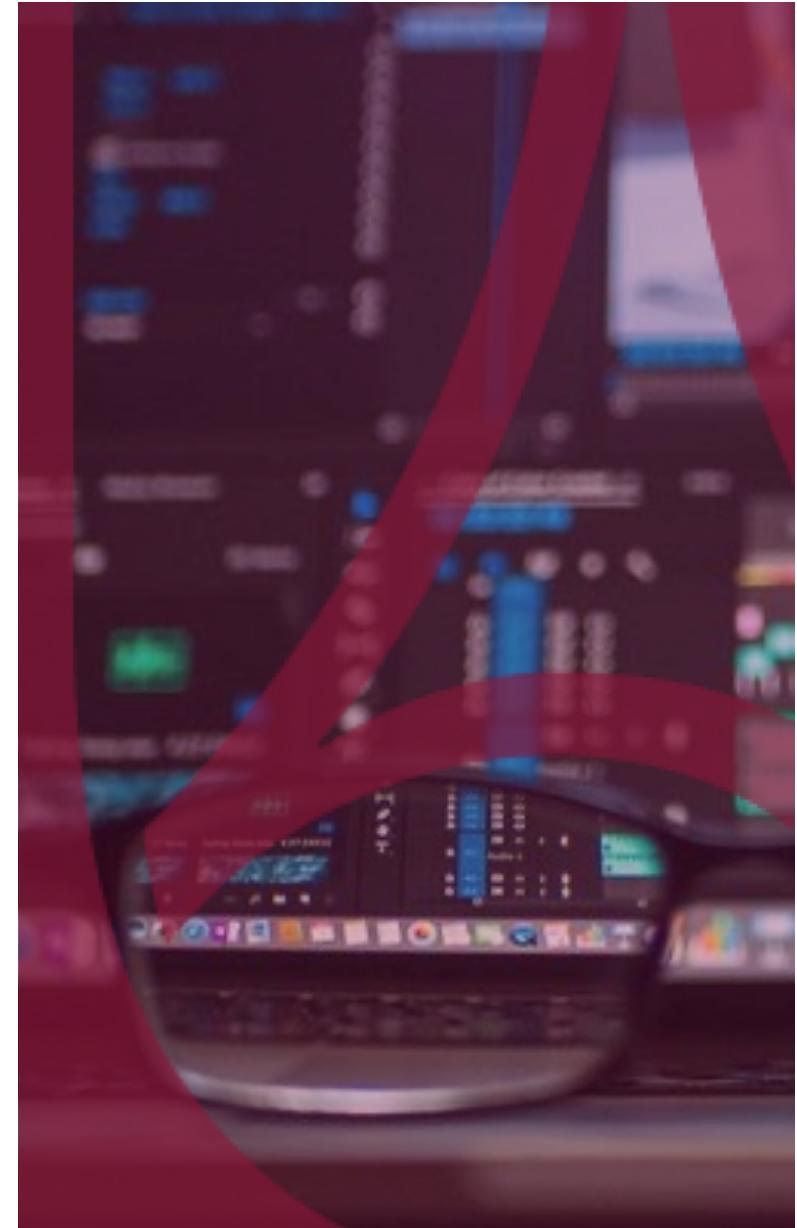


Traitement Automatique de la Langue Naturelle

M2 Data Science • Année 2020 / 2021

Marie CANDITO - Antoine SIMOULIN



Ressources



<https://moodle.u-paris.fr/course/view.php?id=11048>



<https://github.com/AntoineSimoulin/m2-data-sciences>



m2midsunivdeparis.slack.com

Icon made by [Freepik](#), [Becris](#), [Smashicons](#), [Pixel perfect](#), [srip](#), [Adib](#), [Flat Icons](#), [Vitaly Gorbachev](#), [Becris](#) from www.flaticon.com

2

Séance #9

Objectif



Les **modèles de langue** (en anglais language model : LM) cherchent à estimer la **probabilité des séquences de mots**. Ce type de modèles ont plusieurs **applications** très concrètes :



Reconnaissance de la parole

Estimer la phrase la plus probable
 $P(\text{« Tel est pris qui croyait prendre »}) > P(\text{« Tel les prix qui croyait prendre »})$



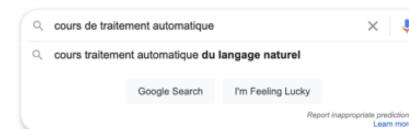
Correction des fautes d'orthographe

$P(\text{« Voici notre carte »}) > P(\text{« Voici notre tarte »})$



Auto compléction de phrase

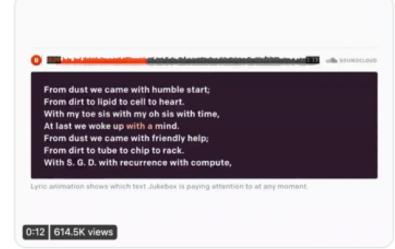
Google



Génération de texte



OpenAI @OpenAI - Apr 30
Introducing Jukebox, a neural net that generates music, including rudimentary singing, as raw audio in a variety of genres and artist styles. We're releasing a tool for everyone to explore the generated samples, as well as the model and code: openai.com/blog/jukebox/



Notations et définitions



On note \mathbf{W} la phrase $(w_1, w_2 \dots w_n)$.



L'objectif du modèle de langue est d'estimer : $\mathbb{P}(\mathbf{W}) = \mathbb{P}(w_1, w_2 \dots w_n)$

Un objectif relié est d'estimer la probabilité du mot suivant : $\mathbb{P}(w_n | w_1, w_2 \dots w_{n-1})$

On note N la taille du corpus qui désigne le nombre de mots dans le corpus, ici 6.

Modèle Unigram



Le modèle le plus simple est le modèle **Unigram**, on suppose que les probabilités de chaque mots sont indépendantes. La probabilité de la phrase est donnée par :

$$\mathbb{P}(\mathbf{W}) = \mathbb{P}(w_1, w_2 \dots w_n) = \prod_{i=1}^n \mathbb{P}(w_i)$$

En pratique, on peut estimer la probabilité de chaque Unigram w en fonction de sa fréquence d'apparition $C(w)$ telle que $\mathbb{P}(w) = \frac{C(w)}{N}$. Avec N la taille du corpus.

Dans notre exemple, on a $N = 6$ et donc $P(\text{« croyait »}) = 1/6$.

Tel est pris qui croyait prendre



Modèle Bigram

On peut envisager un modèle plus complexe telle que **chaque mot dépend du précédent** (une modélisation qui ressemble aux chaînes de Markov). Ce modèle s'appelle le modèle **Bigram**. On a alors $\mathbb{P}(w_n | w_1, w_2 \dots w_{n-1}) \approx \mathbb{P}(w_n | w_{n-1})$. La probabilité de la phrase est donc donnée par :

$$\mathbb{P}(\mathbf{W}) = \mathbb{P}(w_1, w_2 \dots w_n) = \prod_{i=1}^n \mathbb{P}(w_i | w_{i-1})$$

En pratique, on peut estimer la probabilité de chaque Bigram (w_1, w_2) en fonction de leurs fréquences d'apparitions $C(w_1, w_2)$ telle que $\mathbb{P}(w_2 | w_1) = \frac{C(w_1, w_2)}{C(w_1)}$.

Dans le modèle trigram, on modélise la probabilité de $\mathbb{P}(w_3 | w_2, w_1)$ en fonction de leurs fréquences d'apparitions $C(w_1, w_2, w_3)$ telle que $\mathbb{P}(w_3 | w_2, w_1) = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)}$.

Dans notre exemple, on aurait $P(\text{« croyait »} | \text{« qui »}) = 1 / 1 = 1$. En effet dans notre corpus exemple, le mot « qui » est toujours suivi du mot « croyait ».

Probabilité d'une phrase



On peut bien sûr continuer de raffiner le modèle et considérer plus généralement les **n-grams** qui introduisent des **dépendances d'un mots avec les n précédents**. D'après la définition des probabilités conditionnelles, on a $\mathbb{P}(B|A) = \frac{\mathbb{P}(A,B)}{\mathbb{P}(A)}$. En appliquant la règle de composition on a donc pour une phrase (w_1, w_2, w_3, w_4) :

$$\mathbb{P}(w_1, w_2, w_3, w_4) = \mathbb{P}(w_4|w_1, w_2, w_3) \times \mathbb{P}(w_3|w_1, w_2) \times \mathbb{P}(w_2|w_1) \times \mathbb{P}(w_1)$$

Toujours dans notre exemple, on aurait $P(\text{« Tel est pris qui croyait prendre »}) = P(\text{« Tel »}) \times P(\text{« est »} | \text{« Tel »}) \times P(\text{« pris »} | \text{« Tel est »}) \times \dots \times P(\text{« prendre »} | \text{« Tel est pris qui croyait »})$. Mais ce type d'expression est généralement difficile à estimer car on aurait :

$$P(\text{« prendre »} | \text{« Tel est pris qui croyait »}) = C(\text{« Tel est pris qui croyait prendre »}) / C(\text{« Tel est pris qui croyait »})$$

En pratique ces expressions vont rarement apparaître dans le corpus. Certaines phrases n'apparaitront même pas du tout. Si on se limite à un modèle bigram où les mots dépendent du précédent. La probabilité de la phrase sera donc donnée par $P(\text{« Tel est pris qui croyait prendre »}) = P(\text{« Tel »}) \times P(\text{« est »} | \text{« Tel »}) \times P(\text{« pris »} | \text{« est »}) \times \dots \times P(\text{« prendre »} | \text{« croyait »})$

Apprentissage



En pratique, on va apprendre le modèle de langue sur d'importants corpus. On se donne le corpus exemple suivant :

« Je suis en vacances »

« Je vais partir à la réunion »

« Je suis en réunion »

« Je vais partir en vacances »

On peut construire la **matrice de co-occurrence** des mots dans le corpus (<s> et </s> désignent les caractères de début et fin de phrase) et en déduire les **probabilités de transition** dans le modèle bi-gram.

	réunion	vacances	à	suis	en	je	partir	vais	la	<e>	<unk>
(<s>)	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0
(vais,)	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0
(réunion,)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
(la,)	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(vacances,)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
(à,)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
(suis,)	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(en,)	1.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
(je,)	0.0	0.0	0.0	2.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
(partir,)	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

matrice de co-occurrence

	réunion	vacances	à	suis	en	je	par
(<s>)	0.066667	0.066667	0.066667	0.066667	0.066667	0.333333	0.0666
(vais,)	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	0.2307
(réunion,)	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	0.0769
(la,)	0.166667	0.083333	0.083333	0.083333	0.083333	0.083333	0.0833
(vacances,)	0.076923	0.076923	0.076923	0.076923	0.076923	0.076923	0.0769
(à,)	0.083333	0.083333	0.083333	0.083333	0.083333	0.083333	0.0833
(suis,)	0.076923	0.076923	0.076923	0.076923	0.230769	0.076923	0.0769
(en,)	0.142857	0.214286	0.071429	0.071429	0.071429	0.071429	0.0714
(je,)	0.066667	0.066667	0.066667	0.200000	0.066667	0.066667	0.0666
(partir,)	0.076923	0.076923	0.153846	0.076923	0.153846	0.076923	0.0769

matrice de probabilités

Perplexité



On va ensuite **évaluer notre modèle** sur un corpus de test. Pour cela on se donne une **métrique d'évaluation des performances**.

L'objectif du modèle est de prédire le jeu de test. Il doit donc attribuer des probabilités élevées au phrases du jeu de test, puisqu'il s'agit de phrases correctes. La perplexité correspond à la probabilité inverse du jeu de test, normalisée par le nombre de mots :

$$\text{PP}(\mathbf{W}) = \mathbb{P}(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n)^{-\frac{1}{n}} = \frac{1}{\sqrt[n]{\mathbb{P}(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n)}}$$

Si on applique la règle de composition des probabilité (chain rule) :

$$\text{PP}(\mathbf{W}) = \frac{1}{\sqrt[n]{\prod_{i=1}^n \mathbb{P}(\mathbf{w}_i | \mathbf{w}_{i-1}, \dots, \mathbf{w}_1)}}$$

Dans le cas du modèle Bi-gram, on a donc

$$\text{PP}(\mathbf{W}) = \frac{1}{\sqrt[n]{\prod_{i=1}^n \mathbb{P}(\mathbf{w}_i | \mathbf{w}_{i-1})}}$$

Perplexité (2/2)



Imaginons que l'on attribue la probabilité 0.9 au jeu de test et que celui-ci comporte 100 mots. La perplexité vaut alors:

$$PP(W) = 0.9^{\frac{1}{100}} = 1.00105416$$

Si on lui avait attribué la probabilité 10^{-250} (très faible), on aurait:

$$PP(W) = 10^{-250}^{\frac{1}{100}} \approx 316$$

En pratique on cherche donc à obtenir la perplexité la plus **faible**. A titre indicatif, on donne souvent l'exemple du livre Speech and Language Processing [1] entraîné sur un échantillon de 38M de mots du Wall Street journal et testé sur un échantillon de 1.5M mots.

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

En pratique, les probabilités conditionnelles peuvent devenir très petites, pour s'assurer plus de stabilité numérique, on utilise généralement la log perplexité. On mesure donc :

$$\log \mathbb{P}\mathbb{P}(W) = \log \left(\prod_{i=1}^n \mathbb{P}(w_i | w_{i-1})^{\frac{1}{n}} \right) = -\frac{1}{n} \sum_{i=1}^n \log \mathbb{P}(w_i | w_{i-1})$$

[1] Dan Jurafsky and James H. Martin : **Speech and Language Processing (3rd ed. draft)**, <https://web.stanford.edu/~jurafsky/slp3/>



Génération de texte

On peut très simplement utiliser les modèles précédents pour générer du texte suivant l'algorithme ci-dessous :

- 1 Sélectionner le token de début de phrase <s>
- 2 Générer le bigram suivant à partir du mot précédent
- 3 Continuer à générer des tokens tant que le token </s> n'a pas été généré.



Limites des modèles statistiques



En pratique il existe de nombreuses limites aux modèles de types n-gram.



- En particulier, les bigrams ou trigrams sont des modèles trop simples qui ne peuvent pas modéliser de longues dépendances entre les mots de la phrase.



- Si l'on cherche à augmenter le nombre de n-gram, ce type de modèle devient rapidement difficile à évaluer d'un point de vue opérationnel.



- La performance des modèles augmente avec la taille du corpus, mais il en est de même pour la puissance de calcul. Ici encore il devient rapidement difficile d'augmenter significativement la taille des corpus

Modèles de Deep Learning



Les modèles de Deep Learning permettent une modélisation beaucoup plus fine. En effet, leur méthode d'entraînement par back-propagation permet d'apprendre sur des **quantités beaucoup plus importantes de données** et en faisant moins de concessions sur la modélisation. En particulier, on peut considérer l'ensemble des tokens précédents tels que :

$$\mathbb{P}(\mathbf{W}) = \mathbb{P}(w_1, w_2 \dots w_n) = \prod_{i=1}^n \mathbb{P}(w_i | w_{i-1} \dots w_1)$$

On représente chacun des tokens par son embedding puis l'utilise comme features pour un réseau de neurones. Le réseau doit prédire le mot suivant. A chaque itération on prend le mot courant et l'état caché précédent comme input. On introduit ainsi une dépendance entre tous les mots précédents. Il existe de multiples variantes d'architectures de réseaux de neurones. Nous allons travailler en exercice sur le modèle GPT-2.

