

Méthode d'Arora pour l'obtention de sentence embedding

Paul MINCHELLA
minchellapaul@gmail.com



$$x \in \mathbb{R}^N \xrightarrow{\boxed{\text{model } \mu}} \mu(x) = \hat{y}^{\text{pred}} \in \mathcal{Y}$$

Pour optimiser l'apprentissage du modèle μ , on calcule une fonction perte bien définie : $\ell(\hat{y}^{\text{pred}})$ qu'on suppose **convexe** et qu'on cherche à **minimiser** le plus possible.

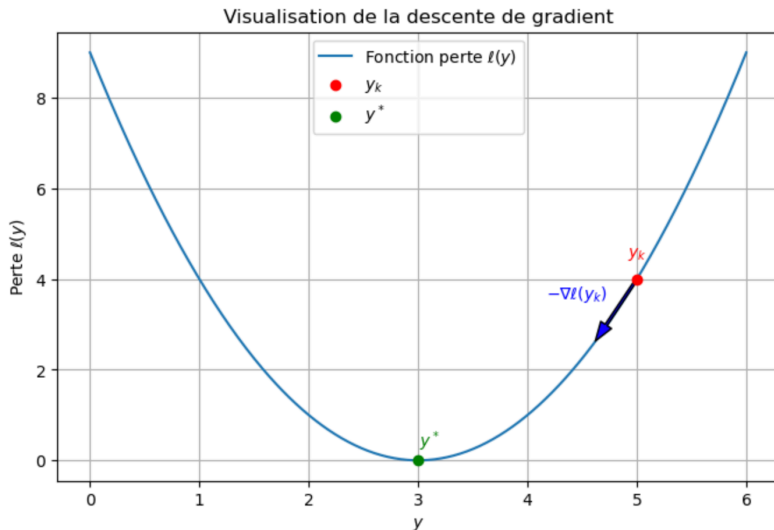
Algorithm 1 Descente de gradient

Input : Point initial y_0 , tolérance d'écart $\varepsilon \geq 0$, pas de descente $\alpha > 0$.

Tant que : $\|\nabla \ell(y_k)\| > \varepsilon$

$$y_{k+1} \leftarrow y_k - \alpha \nabla \ell(y_k)$$

Output : Valeur de convergence y^* où $\|\nabla \ell(y^*)\| \leq \varepsilon$, représentant un minimum approximatif de ℓ .



L'un des objectifs fondamentaux du NLP est de **vectoriser** les mots, et par extension les phrases. Pour ce faire, l'apprentissage des *embeddings* de mots s'effectue schématiquement comme suit (cas Word2Vec) : ([Mikolov et al. 2013](#))

- 1 **Tokenizer** la phrase. Chaque token a son **token id**. On note l'ensemble du vocabulaire \mathcal{V} et son cardinal $|\mathcal{V}|$.
- 2 On considère une grande matrice $W = [w]_{w \in \mathcal{V}} \in \mathbb{R}^{N \times |\mathcal{V}|}$ dont **chaque colonne représente l'embedding** d'un token. Elle est initialisée aléatoirement. N désigne la dimension souhaitée de l'espace pour les *word embeddings*.
- 3 Le corpus d'apprentissage permet l'**estimation de la fréquence d'apparition** de mot contextuel c étant donné un mot central w , qu'on note $p(c | w)$ (méthode *Skip-Gram*).

- 4 La probabilité d'apparition du mot c compte tenu de w se modélise par un modèle **softmax** :

$$\mathbb{P}(c \mid w) = \frac{\exp(\langle v_w, v_c \rangle)}{\sum_{w' \in \mathcal{V}} \exp(\langle v_w, v_{w'} \rangle)} \quad (1)$$

où $\langle v_w, v_c \rangle$ est le produit scalaire des vecteurs d'embeddings pour w et c .

- 5 On parcourt le Neural Network, et on calcule la perte qu'on va minimiser itérativement par descente de gradient couplé à la backpropagation.
On cherchera à **maximiser (??) par backpropagation** (McGonagle, Shaikouski, Williams, et al. 2024) et descente de gradient stochastique, itérées un certain nombre de fois, jusqu'à l'obtention de *word embeddings* $(w)_{w \in \mathcal{V}}$ finaux.

Disposant des *word embeddings* $(v_w)_{w \in \mathcal{V}}$, comment "**mieux apprendre**" les vecteurs représentatifs des phrases (dits **sentence embeddings**), afin d'**améliorer nos modèles** ?

- 1 Les *word embeddings*
- 2 Méthode d'Arora
- 3 Application avec Python
- 4 Conclusion

1 Les *word embeddings*

2 Méthode d'Arora

3 Application avec Python

4 Conclusion

Notre ensemble de mots w issus du vocabulaire \mathcal{V} , de cardinal J , est traité comme suit :

Étape 1/3 : Tokenization

On tokenize les mots, obtenant ainsi un ensemble $(t_w)_{w \in \mathcal{V}}$.

Étape 2/3 : Initialisation des *word embeddings*

Chaque vecteur d'embedding v_w associé à t_w doit *in fine* représenter numériquement le mot dans un espace vectoriel de dimension N .

Remarque 1 : Dans ce contexte, on peut dire que deux mots ont un sens sémantique proche si le produit scalaire de leurs vecteurs d'embedding est élevé, relativement aux normes de ces vecteurs. On appelle *cosinus similarity* la fonction qui propose un score de similarité entre deux mots w et z comme suit :

$$\text{cos-similarity}(w, z) = \frac{\langle w, z \rangle}{\|w\| \cdot \|z\|}$$

Remarque 2 : On dispose d'une base de données (corpus textuels). Étant donné un mot de contexte c , on compte la fréquence d'apparition du mot w qui apparaît dans une courte fenêtre de mots autour de c . Notons cela $p(w | c)$.

Remarque 3 : On désigne par $\mathbb{P}(w | c)$ la probabilité d'apparition de w étant donné c . Plus w et c sont proches sémantiquement, plus $\langle w, c \rangle$ doit être élevé. Une façon de rendre ce nombre nécessairement positif est de le composer avec l'exponentielle. Ainsi, $\exp(\langle w, c \rangle)$ intervient dans l'expression de $\mathbb{P}(w | c)$.

On veut obtenir un nombre $\mathbb{P}(w | c) \in [0, 1]$ et qui vérifie $\sum_{w' \in \mathcal{V}} \mathbb{P}(w' | c) = 1$. Un modèle souvent proposé et adopté en NLP est le **softmax**, qui consiste simplement à diviser (normaliser) $\exp(\langle w, c \rangle)$ par la somme des possibles $\sum_{w' \in \mathcal{V}} \exp(\langle w', c \rangle)$:

$$\mathbb{P}(w | c) = \frac{\exp(\langle v_w, v_c \rangle)}{\sum_{w' \in \mathcal{V}} \exp(\langle v_w, v_{w'} \rangle)} \quad (2)$$

Étape 3/3 : Apprentissage des *word embeddings*

Pour un mot w , son vecteur représentatif $v_w \in \mathbb{R}^N$. On assemble les vecteurs d'embedding de tous les mots $w \in \mathcal{V}$ dans une matrice notée \mathcal{W} écrite comme :

$$\mathcal{W}_{\text{embd}} = \begin{bmatrix} v_{1,1} & \cdots & v_{1,J} \\ v_{2,1} & \cdots & v_{2,J} \\ \vdots & \ddots & \vdots \\ v_{N,1} & \cdots & v_{N,J} \end{bmatrix}$$

Comment obtenir la matrice finale¹ $\mathcal{W}_{\text{embd}}$?

¹C'est-à-dire après apprentissage via un réseau de neurones.

Plus de détails : [Ruder 2016](#)

- 1 On part d'une matrice aléatoire $\mathcal{W}_{\text{embd}}^0$ (on peut imaginer que chaque $v_{i,j} \sim \mathcal{N}(0, 1)$).
- 2 En utilisant (2), on compare $\mathbb{P}(w \mid w')$ avec la fréquence estimée $p(w \mid w')$ depuis notre base de donnée et on calcule la perte ℓ qu'on cherche à minimiser par descente de gradient (et donc, par backpropagation). À chaque étape k , on dispose donc d'une matrice $\mathcal{W}_{\text{embd}}^k$ mieux représentative que la précédente.

Remarque : La perte mesurée est la **cross-entropy**

$$\ell(w, w') = - \sum_{(w, w') \in \mathcal{D}} p(w \mid w') \log \mathbb{P}(w \mid w')$$

- 3 Après un certain nombre d'itérations et/ou une perte ℓ suffisamment petite (en dessous d'un seuil), on récupère en dernière couche du réseau de neurone (de taille N) les vecteurs d'embedding finaux $(v_w)_{w \in \mathcal{V}}$.

1 Les *word embeddings*

2 Méthode d'Arora

3 Application avec Python

4 Conclusion

Arora, Liang, and Ma 2017

- ⇒ Méthode non supervisée pour **générer des sentence embeddings v_s** à partir d'une **moyenne pondérée** des word embeddings v_w .
- ⇒ Introduction d'une pondération appelée *Smooth Inverse Frequency* (**SIF**) pour diminuer l'influence des mots très fréquents.
- ⇒ Le vecteur v_s offre une représentation numérique (dite "vectorisée") du texte.
- ⇒ Surpasse LSTM et RNN en son temps.

L'idée de départ réside dans la variable latente du modèle génératif de texte :

- vu comme un processus dynamique (qui dépend de la position t dans la phrase).
- conduit par un vecteur représentatif du discours à t , noté $c_t \in \mathbb{R}^N$.
- Pour chaque mot w , on dispose de son vecteur d'embedding v_w , indépendant du temps t .

Modèle Log-Linéaire [Mnih and Hinton 2008](#):

$$\mathbb{P}(w \text{ émis à } t \mid c_t) \propto \exp \left(\langle c_t, v_w \rangle \right) \quad (3)$$

équation qui approxime plutôt bien Word2Vec & Glove d'après [Arora et al. 2016](#).

- Estimation de v_s par estimateur de *Maximum a posteriori* pour une phrase s donnée :

$$v_s^{\text{MAP}} \in \operatorname{argmax}_{v_s} \left\{ \mathbb{P}(v_s | s) \right\} \equiv \operatorname{argmax}_{v_s} \left\{ \log \mathbb{P}(v_s) + \sum_{w \in s} \log \mathbb{P}(w | v_s) \right\} \quad (4)$$

$MLE \equiv MAP$: Dans notre cas, l'estimateur *MAP* est identique à celui du maximum de vraisemblance car on suppose la distribution des v_s uniforme [Arora, Liang, and Ma 2017](#); [Arora et al. 2016](#).

- Une très bonne explication du *MAP* : [Tsun 2021](#).

- $\forall t, c_t \equiv c_s$: le vecteur de discours **reste le même le long de la phrase**. On "lisse" ce vecteur selon les contextes en considérant :

$$\tilde{c}_s = \beta c_0 + (1 - \beta) c_s \quad (5)$$

- Ajout de "termes de lissage" : prendre en compte les mots qui apparaissent hors contexte via c_0 .

Le modèle ainsi proposé :

$$\mathbb{P}(w \text{ émis dans } s \mid c_s) = \alpha p(w) + (1 - \alpha) \frac{\exp(\langle \tilde{c}_s, v_w \rangle)}{Z_{\tilde{c}_s}} \quad (6)$$

où :

- $p(w)$: fréquence d'apparition du mot w dans la phrase s
- α, β : hyperparamètres barycentriques
- $Z_{\tilde{c}_s} = \sum_{w \in \mathcal{V}} \exp(\langle \tilde{c}_s, v_w \rangle)$ supposé constant dans toutes les directions vues la dispersion uniforme supposée de v_w . On le notera Z .

(6) quantifie la probabilité d'apparition d'un mot w dans la phrase s , étant donné son contexte c_s , comme étant un **barycentre** (équilibre) entre sa **fréquence d'apparition** $p(w)$ et une **probabilité softmax** justifiant son apparition **selon le contexte** lissé.

On note f_w la log-vraisemblance de (6), pour une phrase s donnée :

$$f_w(\tilde{c}_s) = \log \left[\alpha p(w) + (1 - \alpha) \frac{\exp(\langle v_w, \tilde{c}_s \rangle)}{Z} \right]$$

Son gradient s'exprime par

$$\nabla f_w(\tilde{c}_s) = \frac{1}{\alpha p(w) + (1 - \alpha) \exp(\langle v_w, \tilde{c}_s \rangle) / Z} \left(\frac{1 - \alpha}{Z} \exp(\langle v_w, \tilde{c}_s \rangle) v_w \right)$$

Par développement de Taylor à l'ordre 1, on obtient :

$$f_w(\tilde{c}_s) \simeq f_w(0) + \nabla f_w(0)^\top \tilde{c}_s$$

$$f_w(\tilde{c}_s) \simeq \text{cst} + \frac{a}{p(w) + a} \langle v_w, \tilde{c}_s \rangle$$

où $a = \frac{1-\alpha}{\alpha Z}$.

Ainsi :

$$v_s \in \operatorname{argmax} \left\{ \sum_{w \in S} f_w(\tilde{c}_s) \right\} \propto \operatorname{argmax}_{\|\tilde{c}_s\|=1} \left\{ \sum_{w \in S} f_w(\tilde{c}_s) \right\} = \boxed{\sum_{w \in S} \frac{a}{p(w) + a} v_w} \quad 12$$

¹ La proportionnalité est valable car on linéarise f_w par Taylor.

² $\max_{c \mid \|c\|=1} \{K + \langle c, g \rangle\} = g/\|g\|$ pour toute constante K

On considérera alors:

$$v_s = \frac{1}{|s|} \sum_{w \in s} \frac{a}{p(w) + a} v_w \quad (7)$$

- Il s'agit simplement d'une moyenne pondérée (ramenée au mot) de chaque *word embedding*, où une apparition fréquente du mot w est pénalisée.
- a est un hyperparamètre (on pourra prendre $a = 0.001$).

1 Les *word embeddings*

2 Méthode d'Arora

3 Application avec Python

4 Conclusion

1. Importer un dataset où la ou les variables explicatives sont des textes (+ stats descriptives).
2. Extraire les *word embeddings* (contenus dans un dictionnaire) via Word2Vec, Glove, fast-text, BERT, ... pour chaque observation i .
Quelle est la taille des vecteurs d'embedding ?
3. Calculer la sentence embeddings v_s^i pour chaque observation i via Arora.
4. Effectuer une régression linéaire / logistique / RandomForest avec comme variable d'entrée les $(v_s^i)_i$.
5. Conclusion sur le modèle ? Score (RMSE ou Precision ou autre), intervalle de confiance...

1 Les *word embeddings*

2 Méthode d'Arora

3 Application avec Python




4 Conclusion

Les idées clés à retenir...

- [Arora, Liang, and Ma 2017](#) offre une méthode non supervisée pour **générer des sentence embeddings** v_s à partir d'une **moyenne pondérée** des word embeddings v_w .
 \Rightarrow À partir de word embeddings (transformers, Word2Vec, Glove, fasttext, ...), établir (7).
- Les *sentence embeddings* $(v_s^i)_s$ ainsi obtenus servent de variables d'entrées à notre modèle pour prédire $\hat{y}^{\text{pred}} \in \mathcal{Y}$:

$$\text{phrase} \xrightarrow[\text{word.embd}]{\text{extract.}} (w_1, \dots, w_J) \in \mathbb{R}^{N \times J} \xrightarrow{\text{Arora}} v_s \in \mathbb{R}^N \xrightarrow{\boxed{\text{model}}} \hat{y}^{\text{pred}} \in \mathcal{Y}$$

- Utilisation du package Python SIF_embedding [Liang and Parisi 2017](#).

-  Arora, Sanjeev, Yingyu Liang, and Tengyu Ma (2017). “A Simple but Tough-to-Beat Baseline for Sentence Embeddings”. In: *International Conference on Learning Representations (ICLR)*. Published as a conference paper at ICLR 2017.
-  Arora, Sanjeev et al. (2016). “A Latent Variable Model Approach to PMI-based Word Embeddings”. In: *Transactions of the Association for Computational Linguistics*.
-  Liang, Yingyu and Loreto Parisi (2017). *SIF - Smooth Inverse Frequency implementation*. <https://github.com/PrincetonML/SIF>.
-  McGonagle, John, George Shaikouski, Christopher Williams, et al. (2024). *Backpropagation*. Brilliant.org Wiki. Accessed: 2024-10-13. URL: <https://brilliant.org/wiki/backpropagation/>.
-  Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: *arXiv preprint arXiv:1310.4546*.



Mnih, Andriy and Geoffrey Hinton (2008). "A Scalable Hierarchical Distributed Language Model". In: *Proceedings of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*. URL: https://www.cs.toronto.edu/~amnih/papers/hlbl_final.pdf.



Ruder, Sebastian (2016). *On word embeddings - Part 2: Approximating the Softmax*. <https://ruder.io/word-embeddings-softmax/>.



Tsun, Alex (2021). *Maximum A Posteriori Estimation*. https://web.stanford.edu/class/archive/cs/cs109/cs109.1218/files/student_drive/7.5.pdf. From "Probability & Statistics with Applications to Computing".