

Dealing with large vocabularies

Matthieu Labeau

2017-12-06

Outline

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization

Plan

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization

Modèles de langue

But

Estimer les probabilités **non-nulles** d'une séquence de mots étant donné un vocabulaire \mathcal{V} :

$$P(w_1^L) = P(w_1, w_2, \dots, w_L) = \prod_{i=1}^L P(w_i | w_1^{i-1}), \quad \forall i, w_i \in \mathcal{V}$$

On peut ré-écrire ces probabilités comme celles d'un mot w étant donné un contexte H :

$$P(w_i | w_1^{i-1}) = P(w_i | H_i)$$

Taille du vocabulaire \mathcal{V}

Selon l'application et le langage, sa taille peut être énorme: de **10.000** à **800.000** pour les benchmarks usuels des modèles de langue neuronaux.

Plan

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization

Sortie du réseau: classification

Probabilités de classification

Représentation du contexte H : sortie de la couche cachée: \mathbf{h}

Embedding d'un mot w : \mathbf{w}

Score de sortie: $s(w, H) = \mathbf{w} \cdot \mathbf{h}$

La fonction softmax:

$$P(w|H) = \frac{\exp s(w, H)}{\sum_{w' \in \mathcal{V}} \exp s(w', H)}$$

transforme ces scores en probabilités de classification pour chaque mot de \mathcal{V}

Entraînement

Paramètres

L'ensemble des paramètres du réseau est noté θ

On note la probabilité de classification calculée par le réseau $P_{\theta}(w|H)$

Entropie croisée

On cherche les θ minimisant la log-vraisemblance négative (*negative log-likelihood*):

$$NLL(\theta) = \sum_{w_i, H_i} \log P_{\theta}(w|H)$$

ce qui est aussi l'entropie croisée entre deux distributions P (les données d'entraînement) et P_{θ} (l'estimation):

$$H(P, P_{\theta}) = - \sum_{w \in \mathcal{V}} P(w) \log P_{\theta}(w)$$

Difficulté

Coût du softmax

Dépendance linéaire en la taille du vocabulaire \mathcal{V} , à cause:

- du calcul de la constante de normalisation $\sum_{w \in \mathcal{V}} \exp s_{\theta}(w, H)$
- du calcul de la fonction de score $s_{\theta}(w, H) = \mathbf{w} \cdot \mathbf{h}$ pour chaque mot $w \in \mathcal{V}$

Comment réduire cette complexité ?

Plan

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire**
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization

Shortlists

Solution évidente: réduire la taille du vocabulaire aux s mots les plus fréquents

- On remplace les autres mots par un token spécifique
- Selon l'application, on peut utiliser un modèle moins coûteux pour calculer ces probabilités

Référence: (Schwenk2007)

Softmax hiérarchique

Arbre de calcul de probabilités

On remplace la couche de softmax par un arbre dont les feuilles sont des mots

- Structure d'arbre binaire équilibré: profondeur en $\log_2(|\mathcal{V}|)$
- Chaque node n est représenté par un *embedding* \mathbf{n} - nombre de paramètres conservés
- Probabilité d'un mot : produit de probabilités de décisions binaires

$$P(w|H) = \prod_i P(d_i|n_i, H)$$

ou la décision d_i pour le node n_i est calculée:

$$P(d_i|n_i, H) = \sigma(\mathbf{n}_i \cdot \mathbf{h})$$

- Probabilités binaires, donc normalisées

Référence: (Morin and Bengio2005)

Softmax hiérarchique

Définir la structure de l'arbre

- Utilisation de ressources linguistiques (cluster de mots): baisse de performance (Morin and Bengio2005)
- Stratégie de partitionnement récursive: performance équivalente (Mnih and Hinton2009)
- Codages de Huffman basés sur la fréquence (Mikolov et al.2011), (Zweig and Makarychev2013)

Autres approches similaires

- Short-list + softmax hiérarchique: (Le et al.2011)
- Softmax différencié: (Chen et al.2016)
- Approximation pour GPU: (Grave et al.2017)

Plan

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization

Retour sur la log-vraisemblance négative

Calcul du gradient

Pour comprendre l'impact de la fonction objectif sur l'apprentissage, on examine le gradient - qui détermine les mise à jour:

$$\frac{\partial}{\partial \theta} \log P_{\theta}(w|H) = \frac{\partial}{\partial \theta} s_{\theta}(w, H) - \sum_{w' \in \mathcal{V}} P_{\theta}(w'|H) \frac{\partial}{\partial \theta} s_{\theta}(w', H)$$

Deux termes:

- Ré-enforcement positif pour le mot-cible w
- Ré-enforcement négatif pour tous les autres mots, pondérés par leurs probabilités Ce second terme peut s'écrire

$$\sum_{w' \in \mathcal{V}} P_{\theta}(w'|H) \frac{\partial}{\partial \theta} s_{\theta}(w', H) = \mathbb{E}_{w' \sim P_{\theta}(\bullet|H)} \left[\frac{\partial}{\partial \theta} s_{\theta}(w', H) \right]$$

Approximation par échantillonnage

Faciliter le calcul du second terme

Échantillonnage de Monte-Carlo:

$$\mathbb{E}_{w' \sim P_\theta(\bullet|H)} \left[\frac{\partial}{\partial \theta} s_\theta(w', H) \right] \approx \frac{1}{k} \sum_{i=1}^k \frac{\partial}{\partial \theta} s_\theta(\hat{w}_i, H)$$

avec les k exemples échantillonnés à partir de $P_\theta(\bullet|H)$

→ On veut justement éviter de calculer $P_\theta(w|H)$ pour tous les mots $w \in \mathcal{V}$!

Approximation par échantillonnage

Importance Sampling

Par l'intermédiaire d'une distribution \mathcal{Q} à partir de laquelle on peut échantillonner facilement, on peut obtenir une approximation de notre second terme par la méthode de Monte-Carlo:

$$\mathbb{E}_{w' \sim P_\theta(\bullet|H)} \left[\frac{\partial}{\partial \theta} s_\theta(w', H) \right] \approx \frac{1}{k} \sum_{i=1}^k \frac{P_\theta(\hat{w}_i|H)}{\mathcal{Q}(\hat{w}_i)} \frac{\partial}{\partial \theta} s_\theta(\hat{w}_i, H)$$

tant que $\mathcal{Q}(w) > 0$ si $P_\theta(w|H) \frac{\partial}{\partial \theta} s_\theta(w, H) \neq 0$

→ On a toujours besoin de calculer $P_\theta(w|H)$ pour tous les mots $w \in \mathcal{V} \dots$

Approximation par échantillonnage

Importance Sampling: version biaisée

Idée: remplacer la distribution normalisée $P_\theta(\hat{w}_i|H)$ par un poids calculé à l'aide de la distribution non-normalisée et Q

$$P_\theta(\hat{w}_i|H) \approx \frac{r(\hat{w}_i)}{R}$$

ou $r(\hat{w}_i) = \frac{\exp(s_\theta(\hat{w}_i, H))}{Q(\hat{w}_i)}$ et $R = \sum_{i=1}^k r(\hat{w}_i)$

Référence: (Bengio and S  n  cal2003)

Choix de Q

- Adapter Q pendant l'apprentissage: (Bengio and S  n  cal2008)
- Choisir une sous partie "cible" du vocabulaire pour obtenir un softmax simplifi  : (Jean et al.2015)

Autres approches similaires

- Noise-Contrastive Estimation: approximation via une tâche de classification binaire - vrai mots cs bruit (Mnih and Teh2012)
- Negative Sampling: très similaire au NCE (Melamud et al.2017)

Plan

- 1 Introduction
- 2 Modèles neuronaux et fonction softmax
- 3 Réduire la taille du vocabulaire
- 4 Approches basées sur l'échantillonnage
- 5 Self-Normalization**

Contrainte de normalisation

Idée: remplacer la normalisation du softmax par une contrainte additionnelle dans la fonction objectif:

$$NLL_{Self-normalized}(\theta) = \sum_{(w_i, H_i)} \left[s_{\theta}(w_i | H_i) - \alpha \log^2 \sum_{w' \in \mathcal{V}} \exp s(w', H_i) \right]$$

Référence: (Devlin et al.2014)

Infrequent normalization

Amélioration: appliquer cette contrainte sur une fraction des exemples seulement suffit:

$$NLL_{Self-normalized}(\theta) = \sum_{(w_i, H_i)} s_{\theta}(w_i | H_i) - \frac{\alpha}{\gamma} \sum_{H_i \in \mathcal{H}} \log^2 \sum_{w' \in \mathcal{V}} \exp s(w', H_i)$$

ou \mathcal{H} contient une fraction γ des exemples d'entraînement.

Référence: (Andreas et al.2015)



Jacob Andreas, Maxim Rabinovich, Michael I. Jordan, and Dan Klein.

2015.

On the accuracy of self-normalized log-linear models.

In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 1783–1791.



Yoshua Bengio and Jean-Sébastien S  n  cal.

2003.

Quick training of probabilistic neural nets by importance sampling.

In *Proceedings of the conference on Artificial Intelligence and Statistics (AISTATS)*.



Yoshua Bengio and Jean-S  bastien S  n  cal.

2008.

Adaptive importance sampling to accelerate training of a neural probabilistic language model.

IEEE Trans. Neural Networks, 19(4):713–722.



Wenlin Chen, David Grangier, and Michael Auli.

2016.

Strategies for training large vocabulary neural language models.

In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1975–1985, Berlin, Germany, August. Association for Computational Linguistics.



Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul.

2014.

Fast and robust neural network joint models for statistical machine translation.

In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland, June. Association for Computational Linguistics.



Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou.

2017.

Efficient softmax approximation for gpus.
In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 1302–1310.



Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio.

2015.

On using very large target vocabulary for neural machine translation.
In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Beijing, China, July. Association for Computational Linguistics.



Hai-Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and Francois Yvon.

2011.

Structured output layer neural network language model.

In *Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*, pages 5524–5527, Prague, Czech Republic.

 Oren Melamud, Ido Dagan, and Jacob Goldberger.

2017.

A simple language model based on pmi matrix approximations.

In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1861–1866, Copenhagen, Denmark, September. Association for Computational Linguistics.

 Tomas Mikolov, Stefan Kombrink, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur.

2011.

Extensions of recurrent neural network language model.


In *ICASSP*, pages 5528–5531. IEEE.

 Andriy Mnih and Geoffrey E Hinton.

2009.

A scalable hierarchical distributed language model.

In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1081–1088. Curran Associates, Inc.

 Andriy Mnih and Yee Whye Teh.

2012.

A fast and simple algorithm for training neural probabilistic language models.

In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*.



Frederic Morin and Yoshua Bengio.

2005.

Hierarchical probabilistic neural network language model.

In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252. Society for Artificial Intelligence and Statistics.



Holger Schwenk.

2007.

Continuous space language models.

Comput. Speech Lang., 21(3):492–518, July.



Geoffrey Zweig and Konstantin Makarychev.

2013.

Speed regularization and optimality in word classing.

IEEE, January.