Rapport projet TLNL

Leader : Adrien ZABBAN
Follower : Yanis LABEYRIE

05 novembre 2023

1 Introduction

Le but de ce projet est de coder un modèle de langage basé sur un réseau de neurone multicouche. Ce modèle de langage devra permettre de prédire le mot suivant à partir d'un contexte, qui est un ensemble de mots précédent le mot à prédire.

Pour faire cela, on utilise des embeddings. Le principe est d'associer à chaque mot un vecteur de tels sorte que deux mots similaires ont des vecteurs proches (avec une distance euclidienne), et deux mots totalement différents sont très loin. Cela permet de projeter les mots dans un espace latent pour pouvoir travailler plus efficacement sur ces mots. On note e la dimension de l'espace latent.

Dans un premier temps, on utilisera les embeddings des mots appris par l'algorithme Word2Vect [1] (modèle nommé **FROZEN**). Par la suite, nous tenterons d'améliorer le modèle en lui permettant d'apprendre ses propres embeddings à partir d'embeddings aléatoire (modèle nommé **SCRATCH**), ou directement les embeddings appris de Word2Vect (modèle nommé **ADAPT**).

2 Modèle FROZEN

Nous avons implémenté le perceptron multicouche comme modèle de langage de base. Celui-ci est composé d'une couche d'entrée qui prenant un vecteur de dimension $k \times e$ représentant les embeddings de k mots concaténés. Une couche de neurone cachée dont nous avons choisi de faire varier le nombre de neurone noté h, suivit d'une fonction ReLU [2]. Puis une deuxième couche de neurones suivit d'un softmax retournant un vecteur de taille V, le nombre de mots appris. La Figure 1 représente ce modèle.

2.1 Formalisme mathématiques

En notant, $W \in \mathcal{M}_{k\times e,h_1}(\mathbb{R})$, et $U \in \mathcal{M}_{h,V}(\mathbb{R})$ les matrices de poids, $(b_1, b_2) \in \mathbb{R}^h \times \mathbb{R}^V$ les biais, $X \in \mathbb{R}^{k\times e}$ le vecteur d'entrée, l'équation (1) donne la fonction de sortie $F(X) \in \mathbb{R}^V$ du modèle.

Après plusieurs essais, nous avons choisie de prendre : $k = 3$, $e = 100$, $h_1 = 256$. Et dans nos données d'entraînement, on avait un vocabulaire contenant : $V = 2908$ mots distincts. Avec ces hyperparamètres, nous avons dans ce modèle 824412 paramètres apprenables.

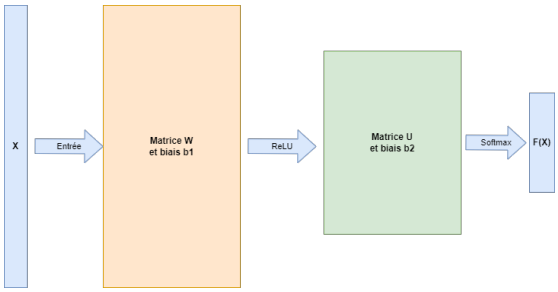


FIGURE 1 – Modèle : **FROZEN**

$$F(x) = softmax(U(ReLU(WX + b_1)) + b_2) \tag{1}$$

Pour entraîner ce modèle, nous avons comparer notre sortie au mot à prédire dans le format One-hot encoding [1], et nous appliquons le calcule de la fonction de coût : cross-entropy. On avons utilisé Adam [3] pour optimiser les paramètres avec un learning rate de 0.01 pour les 5 premières époques et 0.001 pour les suivantes.

2.2 Métriques

Nous avons par ailleurs décidé d'évaluer ce réseau à l'aide de plusieurs métriques comme l'accuracy, la perplexité, le f_1-score et la métrique "top k" [2] (avec $k = 5$), voir [4].

2.3 Résultats

Sur la Figure 2, on voit les courbes d'apprentissage. Les valeurs des métriques sur les données d'entraînement sont en bleue, sur les données de validations sont en orange.

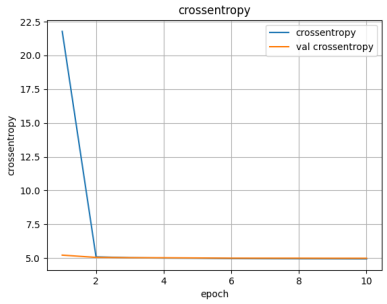
On constate d'après les courbes que le modèle apprend très vite (les courbes atteignent un plateau en quasiment 2 epochs). Ceci est du au fait que les embeddings du modèle sont déjà appris. On observe par ailleurs que le modèle n'overfit pas car il n'y a pas de décalage important entres les valeurs de métrique d'entraînement et de validation à la fin de l'entraînement. Le modèle se stabilise à la fin de l'entraînement avec les métriques de validation présenté dans la Table 1.

1. Le format One-hot encoding est un encodage des indices en un vecteur d'une taille du nombre d'indice possible tel que ce vecteur possède des 0 partout sauf à l'indice en question qui possède un 1.

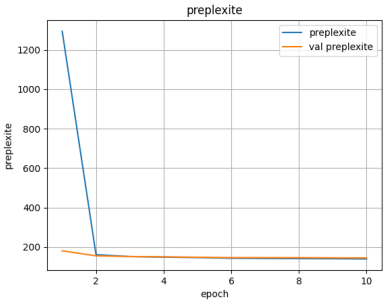
2. La métrique "top-k" évalue un modèle en comptant combien de prédictions correctes il fait parmi les k premières prédictions les plus probables. Attention ici k n'est pas le nombre de mots dans le contexte !

métriques	accuracy	top k	perplexité	f_1 score
modèle **FROZEN**	0.19	0.34	144	9.6e-4

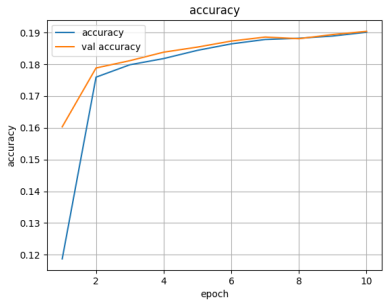
TABLE 1 – Meilleures métriques de validation obtenues en fin d'apprentissage du modèle **FROZEN**.



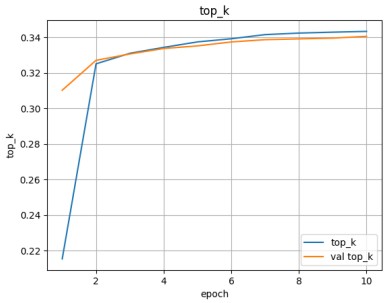
(a) Cross Entropy Loss (Lower is Better)



(b) perplexité (Lower is Better)



(c) Accuracy (Higher is Better)



(d) top k, avec k=5 (Higher is Better)

FIGURE 2 – Entraînement du modèle **FROZEN**

3 Apprentissage automatique des embeddings par le modèle de langage : modèle SCRATCH

3.1 Description

Cette première piste que nous avons exploré consiste à se dire que notre modèle d'embeddings pré-entraîné n'est pas forcément pertinent pour la tâche de prédiction du mot suivant d'une phrase. L'idée est donc de considérer que la matrice d'embeddings peut-être apprise par le modèle de langage et que la matrice apprise sera plus pertinente qu'une matrice apprise séparément pour résoudre la tâche. On va donc considérer que les paramètres de la matrice sont des paramètres apprenables du modèle de langage et donc étendre jusqu'à eux l'algorithme de rétro-propagation du gradient.

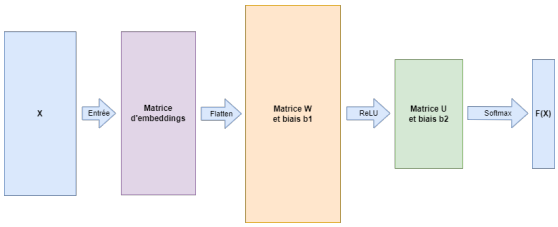


FIGURE 3 – Modèle : **SCRATCH** avec l'apprentissage des embeddings

modèle	accuracy	top k	perplexité	f_1 score
FROZEN	0.19	0.34	144	9.6e-4
SCRATCH	0.195	0.333	280.37	1.27e-3

TABLE 2 – Meilleures métriques de validation obtenues en fin d'apprentissage des modèles **FROZEN** et **SCRATCH**.

3.2 Mise en œuvre

Il a fallut adapter un peu notre code car avant pour avoir l'embedding d'un mot i, on utilisé $E[i]$, où $E \in \mathcal{M}_{V,e}(\mathbb{R})$ est la matrice d'embedding. Cependant, cette opération n'est pas dérivable, donc il a fallut transformer le mot i en un vecteur $x \in \mathbb{R}^V$ one-hot, et faire $x \times E$ pour obtenir l'embedding du mots i, qui cette fois ci, est une opération dérivable. Une fois cette adaptation faite, on a rajouté cette opération dans notre modèle et nous obtenons donc un nouveau modèle illustré par la Figure 3, et donné par l'équation (2).

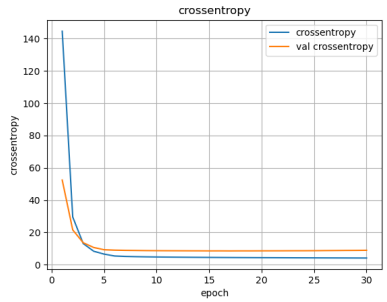
$$F(x) = softmax(U(ReLU(W\tilde{X} + b_1)) + b_2) \quad (2)$$
$$\text{où } \tilde{X} = flatten(XE) \quad \text{et} \quad X \in \mathcal{M}_{k,V}(\mathbb{R})$$

3.3 Résultats

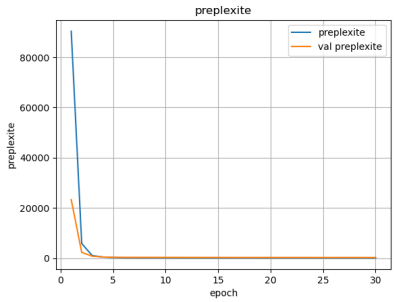
On remarque (Figure 4) tout d'abord que l'apprentissage à la fois du modèle et des embeddings est plus long car l'ensemble (modèle + matrice d'embedding) possède un plus grand nombre de paramètres (1115212 contre 824412 pour le modèle **FROZEN**) qui nécessitent donc plus d'epochs que **FROZEN** pour être actualisés.

On obtient en terme de métriques de validation, des résultats semblables au modèle précèdent. Cela s'explique par le fait que, bien qu'on apprenne par rétro-propagation les embeddings et que donc on s'attend à une meilleur qualité de résultat, cela est compensé par le fait que l'apprentissage se fait à partir de 0 et on commence donc l'apprentissage avec des résultats aléatoires.

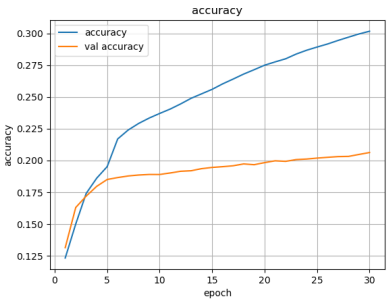
A la fin de l'apprentissage, nous obtenons les métriques de validation suivantes, comme on peut le voir dans le tableau 2.



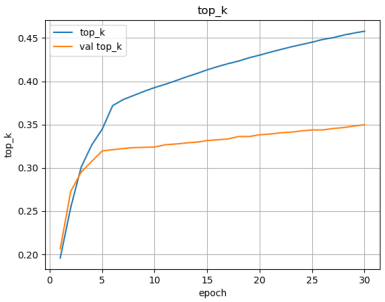
(a) Cross Entropy Loss (Lower is Better)



(b) perplexité (Lower is Better)



(c) Accuracy (Higher is Better)



(d) top k, avec k=5 (Higher is Better)

FIGURE 4 – Entraînement du modèle **SCRATCH**

4 Apprentissage des embeddings à partir de l'algorithme Word2Vect : modèle ADAPT.

4.1 Description

Dans cette partie, nous allons tenter d'améliorer la performance et surtout la vitesse d'apprentissage. Au lieu, d'initialiser la matrice d'embedding aléatoirement comme dans la partie précédente, nous allons initialiser cette matrice à partir des embeddings déjà appris avec l'algorithme Word2Vect. En procédant comme cela, nous allons pouvoir continuer d'apprendre ces embeddings pour coller au mieux à notre tâche de prédiction.

4.2 Mise en œuvre

La mise en œuvre de cette partie n'a pas été très compliqué à implémenter car il nous a suffit de récupérer la matrice d'embedding apprise et de la copier dans le modèles lors de son initialisation.

4.3 Résultats

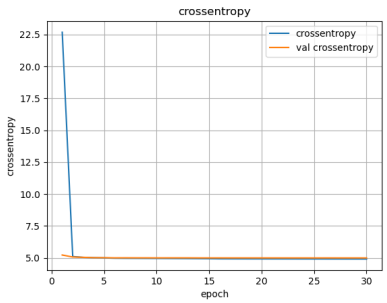
On constate immédiatement que l'apprentissage est bien plus rapide (voir Figure 5) dans le cas où on initialise les embeddings avec word2vec par rapport au cas où on les initialise aléatoirement. Cela s'explique par le fait que contrai-

modèles	cross entropie	accuracy	top k	f_1 score	perplexité
FROZEN	4.89	0.20	0.35	$9.42e-4$	131
SCRATCH	8.05	0.20	0.35	$1.16e-3$	247
ADAPT	4.90	0.20	0.35	$8.86e-4$	130

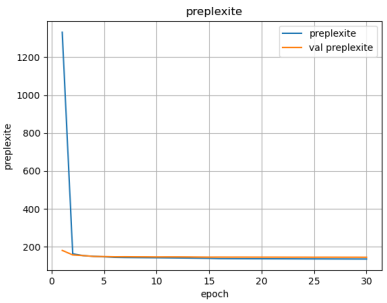
TABLE 3 – Résultat des différents modèles sur la base de données de teste.

rement au modèle précédent, le modèle a déjà des embeddings valides aux début de l'apprentissage qu'il peut par la suite perfectionner.

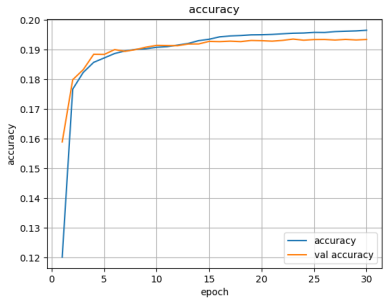
Notre hypothèse était que initialiser les embeddings avec word2vec puis les modifier au cours de l'apprentissage permettrait d'obtenir de meilleurs performances que celles du modèle où les embeddings sont gelés pendant l'apprentissage.



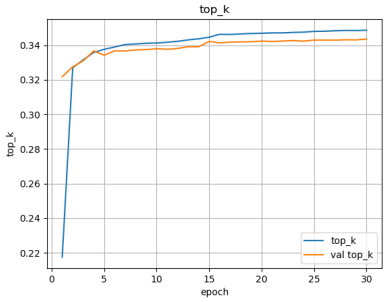
(a) Cross Entropy Loss (Lower is Better)



(b) perplexité (Lower is Better)



(c) Accuracy (Higher is Better)



(d) top k, avec k=5 (Higher is Better)

FIGURE 5 – Entraînement du modèle **ADAPT**

5 Conclusions et perspectives

Références

[1] T. Mikolov, K. Chen, G. Corrado et J. Dean, « Efficient estimation of word representations in vector space, » *arXiv preprint arXiv :1301.3781*, 2013.

[2] A. F. Agarap, « Deep Learning using Rectified Linear Units (ReLU), » *CoRR*, t. abs/1803.08375, 2018. arXiv : 1803.08375. adresse : http://arxiv.org/abs/1803.08375.

[3] D. P. Kingma et J. Ba, « Adam : A method for stochastic optimization, » *arXiv preprint arXiv :1412.6980*, 2014.

[4] L. Liu, T. G. Dietterich, N. Li et Z. Zhou, « Transductive Optimization of Top k Precision, » *CoRR*, t. abs/1510.05976, 2015. arXiv : 1510.05976. adresse : http://arxiv.org/abs/1510.05976.