Polytechnique de Montréal

INF8225

Intelligence artificielle : techniques probabilistes et d'apprentissage

TP4
Correction d'erreurs d'orthographes dans la langue anglaise par apprentissage profond.

Philippe Marcotte
MATRICULE 1736842
Olivier St-Amour
MATRICULE 1733229

TRAVAIL PRÉSENTÉ
à Christopher Pal

Date de remise
18 avril 2017

Table des matières

Introduction

Le traitement du langage naturel par intelligence artificielle est un domaine sur lequel de la recherche est effectuée depuis les années 1950. Étant donné la complexité des règles grammaticales et syntaxiques des différents langages, les avancées récentes en apprentissage profond ont permis de développer de nouvelles techniques.

Plus précisément, l'utilisation des réseaux de neurones récurrents est particulièrement populaire et efficace dans ce domaine. En effet, le langage naturel étant contextuel, les résultats de tâches comme la prédiction de mots ou la correction d'erreurs sont dépendants des données observées antérieurement.

L'une des tâches qui nous intéresse particulièrement est la correction d'erreurs d'orthographes dans une phrase écrite dans la langue anglaise. Pour ce faire, nous nous inspirons d'une expérimentation qui semble avoir produit de bons résultats dans la prédiction de mots. (Kim, Jernite, Sontag, & Rush, 2015) Nous pensons que ce modèle est intéressant et s'appliquerait également dans la correction d'erreurs d'orthographes étant la contrainte de temps imposée pour ce projet. Nous émettons l'hypothèse qu'il suffirait de modifier le prétraitement des données en entrées et cibles de ce modèle pour qu'il s'applique à notre étude de cas.

Résumé des travaux récents

D'autres expérimentations récentes dans le domaine du traitement du langage naturel semblent donner de bons résultats. Notamment, les réseaux récurrents semblent être ce qui offre les meilleures performances d'après la littérature scientifique récente (Chelba, Mikolov, Schuster, Ge, & Brants, 2013). Parmis les réseaux récurrents, le *Long Short Term Memory* a la réputation d'être efficace pour les modèles de langage naturel (Sundermeyer, Schlüter, & Ney, 2012). En effet, leur mémoire à long terme ne souffre pas des mêmes problèmes que celle des réseaux récurrents traditionnels (Hochreiter & Scmidhuber, 1997). De plus, on observe également l'utilisation fréquente de l'encodage des mots (Mikolov, Chen, Corrado, & Dean, 2013) ou des caractères dans le but d'extraire des caractéristiques supplémentaires des données textuelles. Le concept de n-gram en est un autre très utilisé pour analyser les mots dans une phrase. Enfin, l'approche proposée, par le modèle reproduit dans ce projet, effectue un encodage des caractères grâce à une couche convolutionnelle dont les filtres servent à extraire des caractéristiques pour différent n-gram de caractères.

Approche théorique

L'architecture de notre modèle est basée sur celle utilisée par Yoon Kim et al. (2015) dans leurs expérimentations sur la prédiction de mots. Cette architecture est composée d'une couche d'encodage des caractères, une couche de convolution, une couche de type *Highway* et une couche de type LSTM avant d'appliquer un *softmax*.

Encodage par caractères

Contrairement à d'autres domaines comme la reconnaissance d'images, les données brutes provenant d'un texte, soit les chaînes de caractères, comportent très souvent peu de similarités malgré les fortes associations qui peuvent y être contenues. Par exemple, des mots comme "oiseau" et "chat" sont très différents en ce qui a trait aux arrangements de lettres qui constituent les mots, cependant tous deux sont entre autres des espèces animales. Il est donc raisonnable de penser qu'ils vont être utilisés dans des contextes similaires. Ce genre d'information n'est pas communiquée efficacement dans le lexique d'un langage, c'est pourquoi l'utilisation récente de l'encodage des mots dans les réseaux de neurones pour le traitement du langage naturel est un autre outil très souvent utilisé.

L'encodage d'un mot (Word Embedding) consiste donc à transformer sa représentation comme étant un index dans un vocabulaire, représenté soit par un chiffre ou un vecteur *one-hot* à un vecteur ayant une taille arbitraire dont chaque composante représente le poids d'une caractéristique identifiée. Les mots à proximité dans cette représentation vectorielle partagent une certaine relation sémantique.

Étant donné que nous cherchons à corriger des erreurs d'orthographe, il peut être difficile d'encoder des mots qui ne se retrouvent pas dans le vocabulaire initial lors de l'apprentissage en raison de l'injection d'erreurs lors du prétraitement des données. Nous utilisons donc une approche similaire et plus récente dans notre modèle, il s'agit d'effectuer l'encodage au niveau des caractères (Character Embedding). Un mot est donc représenté sous forme matricielle où chaque colonne correspond à l'encodage sous forme de vecteur d'un caractère du mot.

Couche convolutionnelle

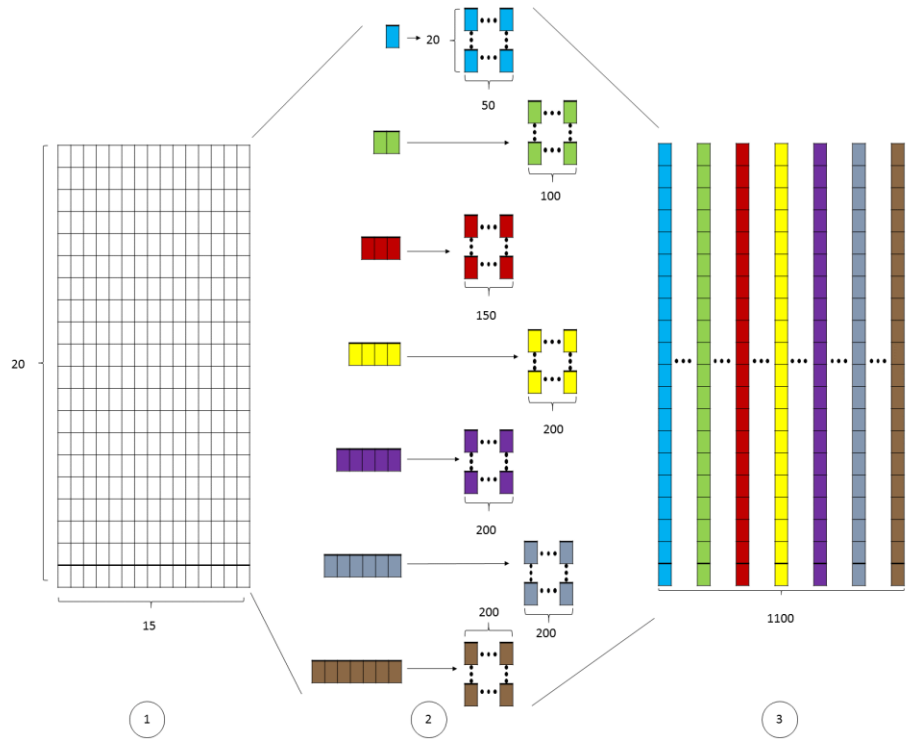

Figure 1 - Couche convolutionnelle utilisée dans le modèle

Suite à l'encodage au niveau des caractères, la représentation matricielle du mot est envoyée à une couche où une convolution est appliquée entre cette matrice et plusieurs filtres de taille arbitraire. La taille variable des filtres permet d'ajuster le nombre de caractères qui sont considérés pour chaque application du filtre sur le vecteur représentant le mot. L'intention derrière ces filtres est d'extraire des caractéristiques des mots. Une couche de *max-pooling over time* (Kim, Jernite, Sontag, & Rush, 2015) est appliqué sur le résultat de la convolution pour obtenir l'information la plus importante des caractéristiques filtrées en trouvant la valeur la plus grande du tenseur. L'intuition est que cette couche va permettre de reconnaître un mot malgré les erreurs comme un caractère en moins, de trop ou à la mauvaise place.

La figure présentée ci-haut démontre en trois étapes l'application de la couche convolutionnelle avec les hyper-paramètres utilisés dans l'article sur lequel nous nous basons (Kim, Jernite, Sontag, & Rush, 2015). En premier lieu, celle-ci reçoit en entrée un lot de 20 mots convertis en leur encodage de caractères. L'encodage est fait sur 15 dimensions. En deuxième lieu, 7 filtres sont appliqués sur le lot. Ils sont tous d'une hauteur de 1 et d'une de largeur de [1, 2, 3, 4, 5, 6, 7] respectivement. Lors de cette deuxième

étape, un biais est ajouté et une fonction tan est appliqué sur la convolution pour obtenir des caractéristiques de taille [50, 100, 150, 200, 200, 200, 200] respectivement. Enfin, le résultat de la couche *max-pooling over time* pour chaque caractéristique est concaténé pour obtenir une matrice de 20 de hauteurs et 1100 de largeur. 1100 étant la somme de la taille de chaque caractéristique.

Couche *Highway*

Le modèle sur lequel nous nous basons pour effectuer nos expériences donne de meilleurs résultats en utilisant une couche *Highway*. Il s'agit d'une couche de type perceptron appliquant une transformation affine puis une non-linéarité entre la sortie de l'encodage par convolution des mots et la couche longue mémoire à court terme pour obtenir un nouvel ensemble de caractéristiques. Contrairement à un réseau *Feed-Forward* traditionnel, une couche *Highway* applique le calcul suivant à chaque couche :

$$\boldsymbol{z} = \boldsymbol{t} \odot \boldsymbol{g}(\boldsymbol{W_H y} + \boldsymbol{b}_{\mathrm{H}}) + (\mathbf{1} - \boldsymbol{t}) \odot \boldsymbol{y}$$ (Kim, Jernite, Sontag, & Rush, 2015)

où $g$est une fonction non-linéaire, $t = \sigma(W_T y + b_T)$ est la *transform gate* et $(1 - $ est la *carry gate*. Ces portes servent à déterminer si l'entrée devrait passer au travers de la couche comme dans un perceptron ordinaire ou si elle devrait aller directement à la sortie.

Longue mémoire à court terme

Finalement, une couche récurrente (RNN) est utilisée pour interpréter le contexte qui se retrouve dans un ensemble de mots. Un réseau récurrent est un type de réseau fonctionnant avec une boucle où une partie de l'information persiste et est transmise au cours des différentes itérations. Cette boucle est déroulée de façon à ce qu'elle puisse s'intégrer avec le reste du réseau.

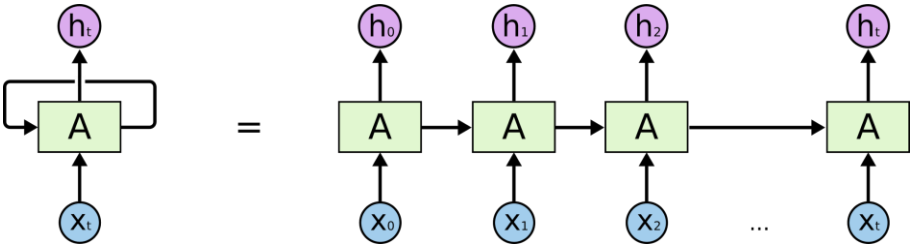

Figure 2 - Un réseau récurrent déroulé (Olah, 2015)

Un type particulier de réseau récurrent est utilisé dans notre modèle, il s'agit d'une *Long Short Term Memory* (LSTM) (Hochreiter & Scmidhuber, 1997). Ce type de réseau comporte notamment plusieurs

couches cachées permettant de retirer l'information provenant d'un état précédent considéré inutile et ajouter l'information provenant de l'entrée considérée pertinente à sauvegarder dans l'état courant. La LSTM est particulièrement intéressante à utiliser dans le traitement du langage naturel étant donné les changements de contexte fréquents dans le texte.

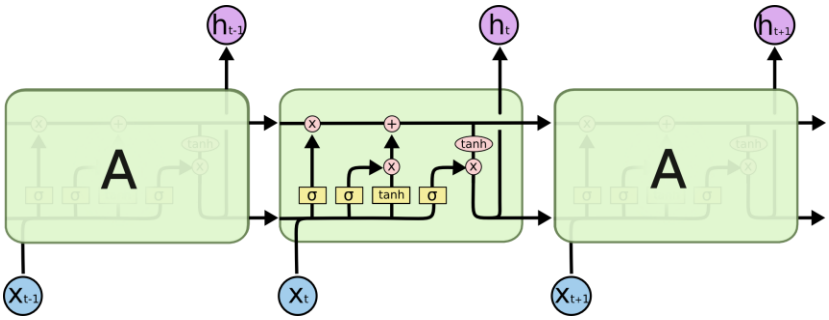

Figure 3 -Une LSTM déroulée (Olah, 2015)

De plus, la LSTM résout le problème du *vanishing gradients* (Bengio, Simard, et Frasconi 1994) lors de l'optimisation en incorporant un vecteur de mémoire additif Ct. Le gradient peut cependant toujours exploser, le problème est contourné en limitant la valeur maximale du gradient.

Entrainement et optimisation

La métrique utilisée pour l'évaluation de la performance de notre modèle est la perplexité, tel que généralement utilisé dans la modélisation du langage naturel.

$$loss = -\frac{1}{N}\sum_{i=1}^{N} ln\, p_{target|input}$$

$$perplexity = e^{loss} = e^{-\frac{1}{N}\sum_{i=1}^{N} ln\, p_{target}}$$

Notre modèle est entraîné en utilisant la descente du gradient stochastique tronquée où la propagation du gradient est limitée à 35 étapes. Le taux d'apprentissage est initialisé à 1.0 et est réduit de moitié à chaque époque si la perplexité ne s'est pas améliorée. Un dropout est utilisé pour la régularisation (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) avec une probabilité de 0.5. Les paramètres du modèle sont initialisés selon une distribution uniforme portant de -0.05 à 0.05.

Expériences

Données d'entraînement

Nous avons réutilisé la banque de mots English Penn Treebank (Marcus, Marcinkiewicz, & Santorini, 1993) et non, le *1 Billion Words Benchmark* de Google (Chelba, Mikolov, Schuster, Ge, & Brants, 2013). En effet, le premier offrait des données ayant subi un prétraitement plus en profondeur. Ce prétraitement consistait, entre autres, à remplacer tous les mots occurrents qu'une seule fois par le symbole <unk>. Aussi, tous les nombres plus gros que des années sont remplacés par la lettre N. Mise à part ce dernier symbole, il n'y a aucune lettre majuscule. Il n'y a aucun symbole diacritique. Les données contiennent un total de 1085779 mots.

Vocabulaires

Le prétraitement des données d'entraînement consistait à élaborer le vocabulaire de caractères ainsi que celui des mots. Le symbole <unk> est remplacé par un caractère non-utilisé "|" dans ce cas-ci. De plus, chaque mot est transformé en vecteur de caractère. Chaque vecteur commence par "{" et termine par "}". Cela est nécessaire, car tous les mots vont être ajusté pour être de la même longueur. Il faut donc une manière de montrer le début et la fin d'un mot pour que le modèle n'apprenne pas que tous les mots sont de la même longueur. Pour égaliser la longueur de tous les mots, nous trouvons le mot le plus long et ajoutons des 0 à la fin de tous les autres jusqu'à obtenir la même longueur. Ce traitement est nécessaire, car les couches LSTM ne supportent pas des entrées de tailles variables. Le résultat du prétraitement fût le suivant : 51 caractères et 10000 mots uniques dont le plus long était de 21 caractères.

Modèles expérimentés

Nous avons expérimenté avec quatre modèles. Nous voulions comparer les résultats entre des modèles ayant de l'injection d'erreurs et des modèles sans injection. Nous voulions aussi voir la différence entre des modèles avec une couche *Highway* et sans celle-ci. Nos quatres modèles étaient donc les suivants :

- Injection d'erreurs et couche *Highway*
- Injection d'erreurs sans couche *Highway*
- Couche *Highway* sans injection d'erreurs
- Ni couche *Highway*, ni injection d'erreurs

Chaque modèle applique une détérioration du taux d'apprentissage lorsque la perte de l'ensemble de

validation ne s'est pas améliorée comparativement à la meilleure perte rencontrée depuis le début. La détérioration consiste à diviser le taux par deux. Un entraînement se termine lorsque le taux d'apprentissage devient plus petit que 1×10^{-5}.

Injection d'erreurs

L'injection s'effectuait en probabilité que sur un mot sur quatre. Un mot pouvait être corrompu de quatre manière différentes. Soit une lettre serait retirée, soit une lettre serait ajoutée, soit une lettre serait transposée avec sa voisine immédiate ou soit une lettre serait remplacée par une autre. Chaque type d'injection avait aussi une chance sur quatre d'être appliquée. Cependant, un mot ne pouvait être corrompu qu'une seule fois.

Hyper-paramètres

L'intention de l'expérimentation étant de comparer l'injection d'erreurs et la couche *Highway*, nous avons repris les hyper-paramètres proposés dans l'article de (Kim, Jernite, Sontag, & Rush, 2015). Ceux-ci ont été démontrés comme étant de bons hyper-paramètres pour un modèle de langue. Les hyper-paramètres communs aux 4 expériences étaient les suivants :

Taux d'apprentissage : 1.0

Détérioration du taux d'apprentissage : 0.5

Taille du vecteur d'encodage des caractères : 15

Nombre de filtres de convolutions : 7

Largeur des filtres de convolutions : [1, 2, 3, 4, 5, 6, 7]

Taille des caractéristiques sortant des filtres : [50, 100, 150, 200, 200, 200, 200]

Nombre de couches LSTM : 2

Nombre de cellules par LSTM : 35

Taille de l'état caché des LSTM : 650

Probabilité de dropout à la sortie des LSTM : 0.5

Taille d'un lot d'entrées : 20

Initialisations des poids : [-0.05, 0.05]

Normalisation du gradient : 5.0

Résultats

Ensemble	Perte	Perplexité
Entraînement	0,0847	1,088
Validation	0,1274	1,136
Test - Sans injection d'erreurs	0,0948	1,099
Test - Avec injection d'erreurs	0,9825	2,671

Tableau 1 - Résultats du modèle ni couche *Highway*, ni injection d'erreurs

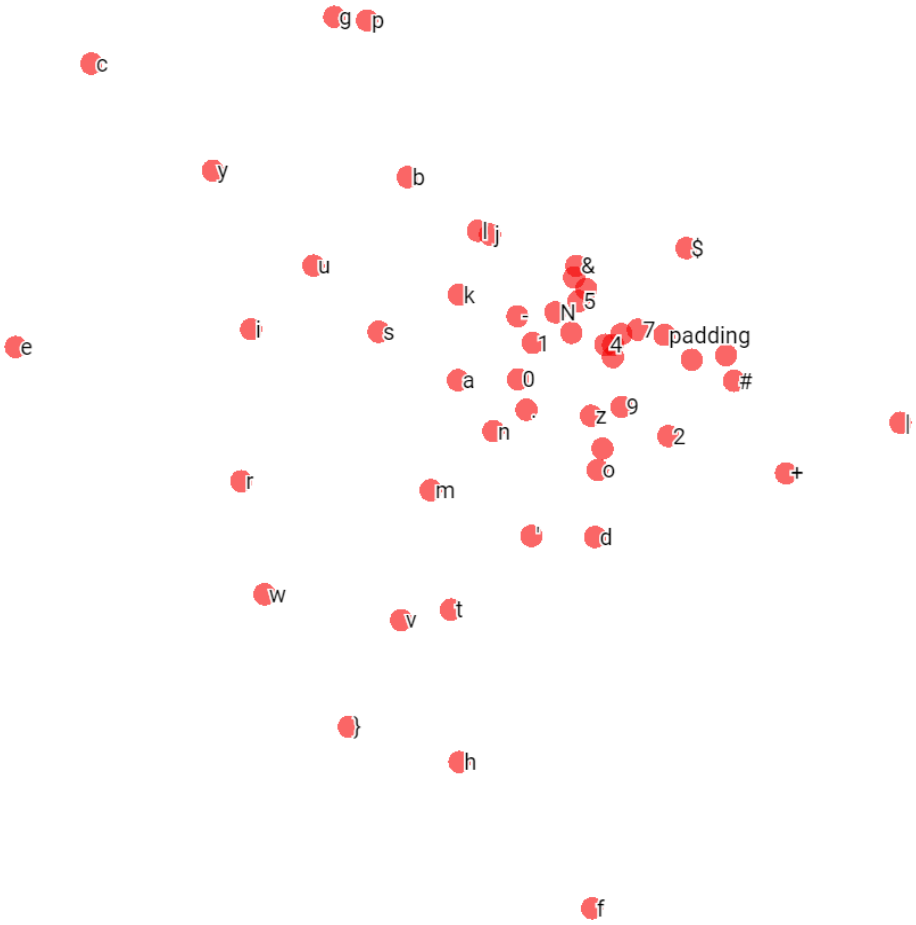

Figure 4 - Projection en 2D des encodages des caractères en 15 dimensions de l'expérimentation sans couche *Highway* et sans injection d'erreurs par ACP. Les deux axes sélectionnés représentent 45.7% de la variation totale.

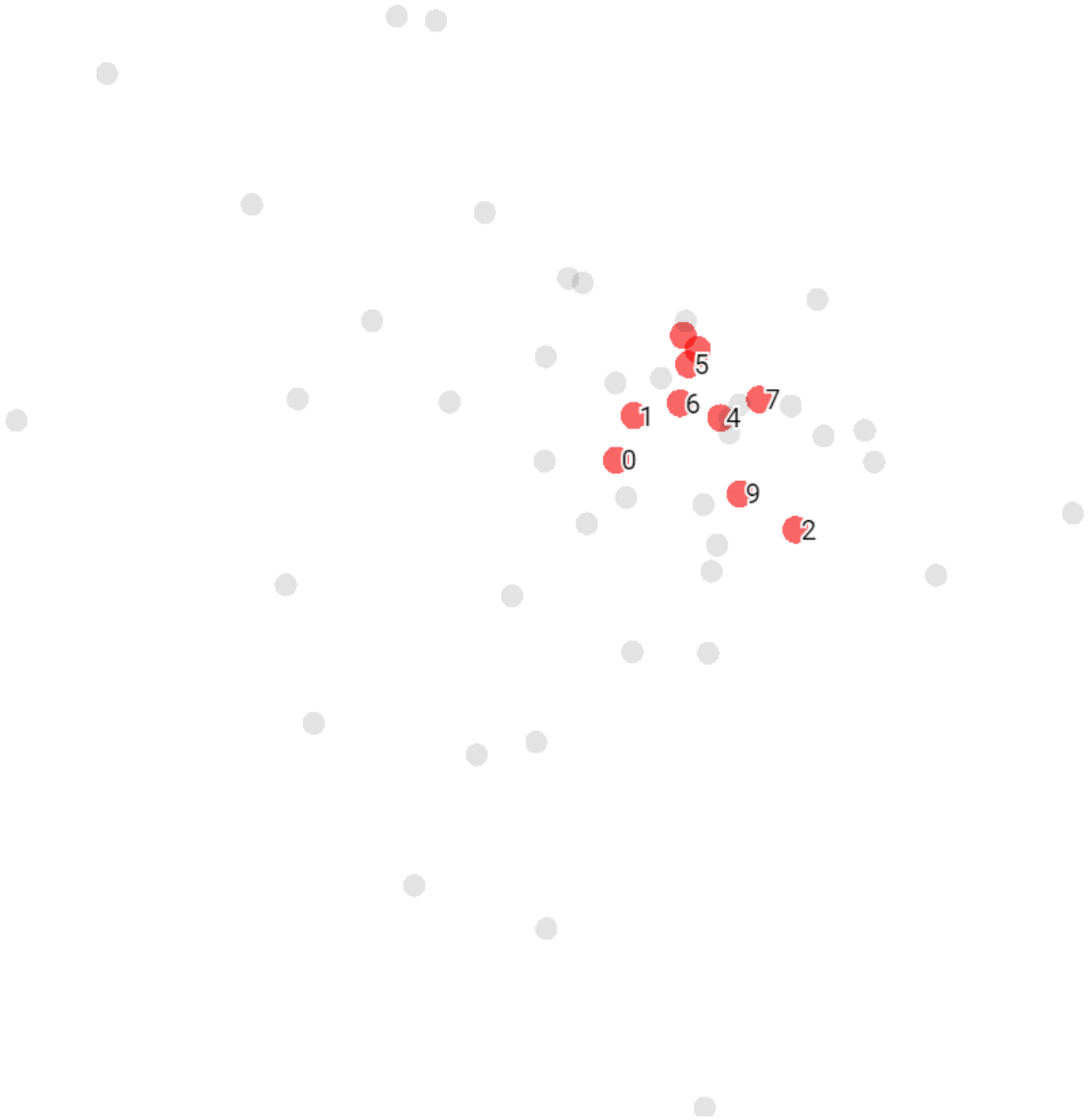

Figure 5 - Projection en 2D des encodages des nombres de l'espace original de 15 dimensions de l'expérimentation sans couche *Highway* et sans injection d'erreurs par ACP. Les deux axes sélectionnés représentent 45.7% de la variation totale.

Ensemble	Perte	Perplexité
Entraînement	0,2088	1,232
Validation	0,2389	1,270
Test - Sans injection d'erreurs	0,1035	1,109
Test - Avec injection d'erreurs	0,2146	1,239

Tableau 2 – Résultats du modèle injection d'erreurs et sans couche Highway

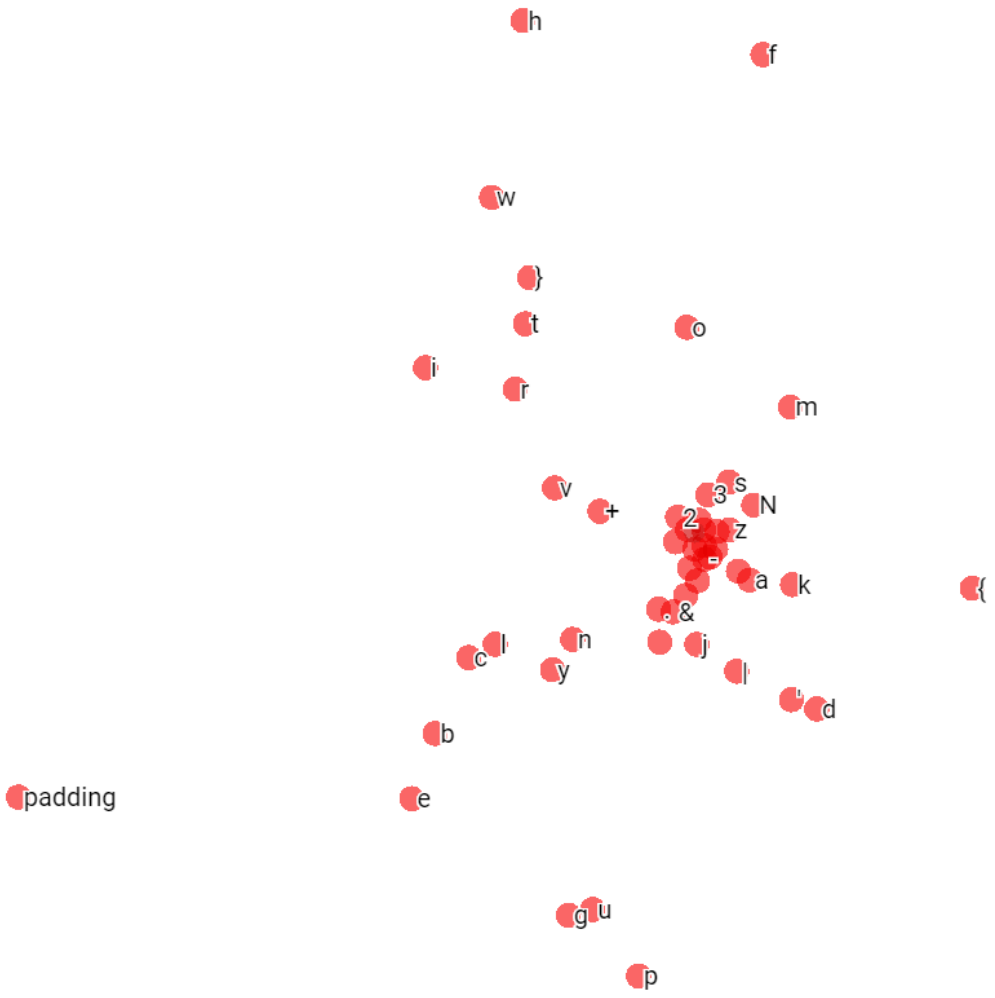

Figure 6 - Projection en 2D des encodages des caractères en 15 dimensions de l'expérimentation sans couche *Highway* et avec injection d'erreurs par ACP. Les deux axes sélectionnés représentent 41.6% de la variation totale.

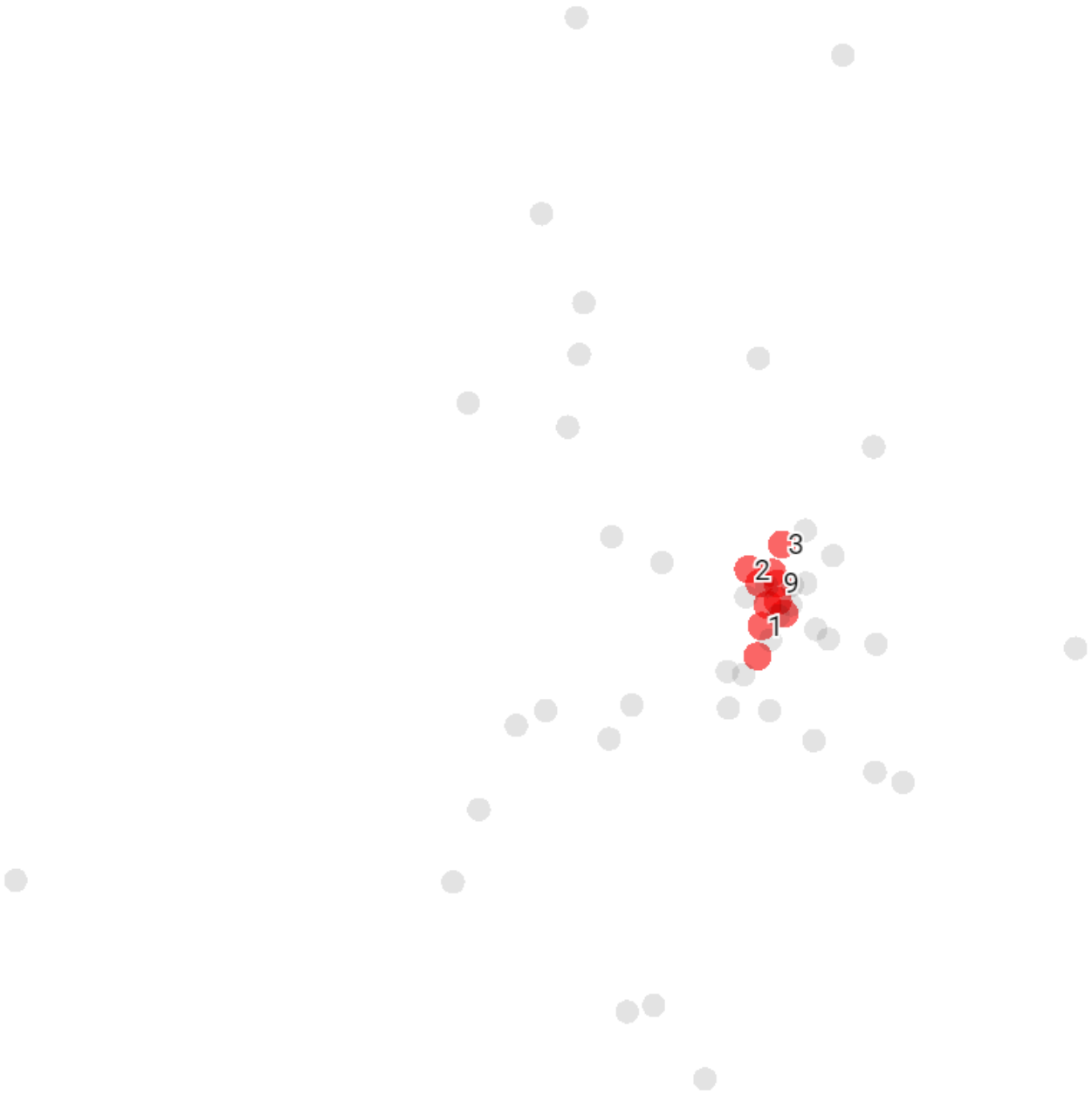

Figure 7 - Projection en 2D des encodages des nombres de l'espace original de 15 dimensions de l'expérimentation sans couche *Highway* et avec injection d'erreurs par ACP. Les deux axes sélectionnés représentent 41.6% de la variation totale.

Ensemble	Perte	Perplexité
Entraînement	0,0904	1,095
Validation	0,1485	1,160
Test - Sans injection d'erreurs	0,1115	1,118
Test - Avec injection d'erreurs	1,0922	2,981

Tableau 3 – Résultats du modèle sans injection d'erreurs et 2 couches Highway

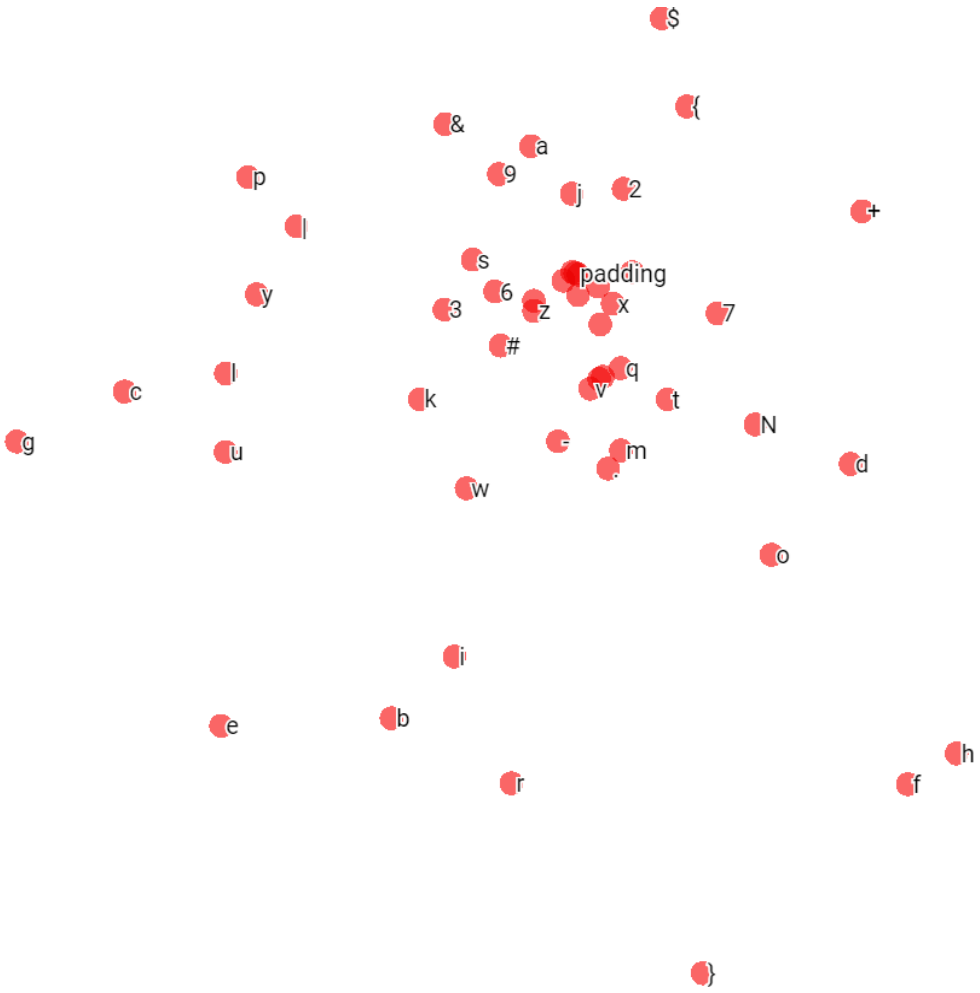

Figure 8 - Projection en 2D des encodages des caractères en 15 dimensions de l'expérimentation avec 2 couches *Highway* et sans injection d'erreurs par ACP. Les deux axes sélectionnés représentent 48.2% de la variation totale.

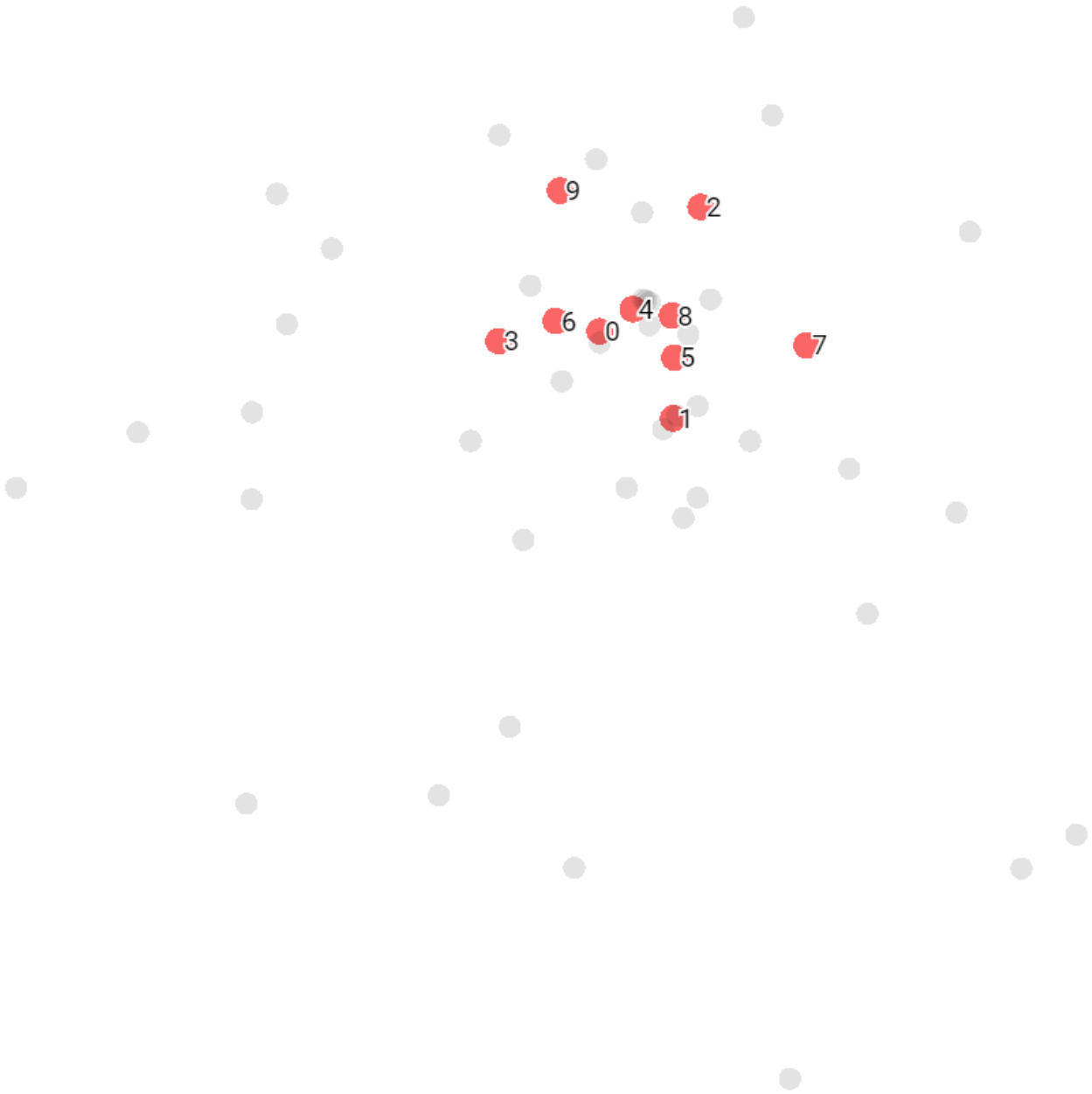

Figure 9 - Projection en 2D des encodages des nombres de l'espace original de 15 dimensions de l'expérimentation avec 2 couches *Highway* et sans injection d'erreurs par ACP. Les deux axes sélectionnés représentent 48.2% de la variation totale.

Ensemble	Perte	Perplexité
Entraînement	0,1984	1,219
Validation	0,2564	1,292
Test - Sans injection d'erreurs	0,1110	1,117
Test - Avec injection d'erreurs	0,2133	1,237

Tableau 4 – Résultats du modèle avec injection d'erreurs et 2 couches Highway

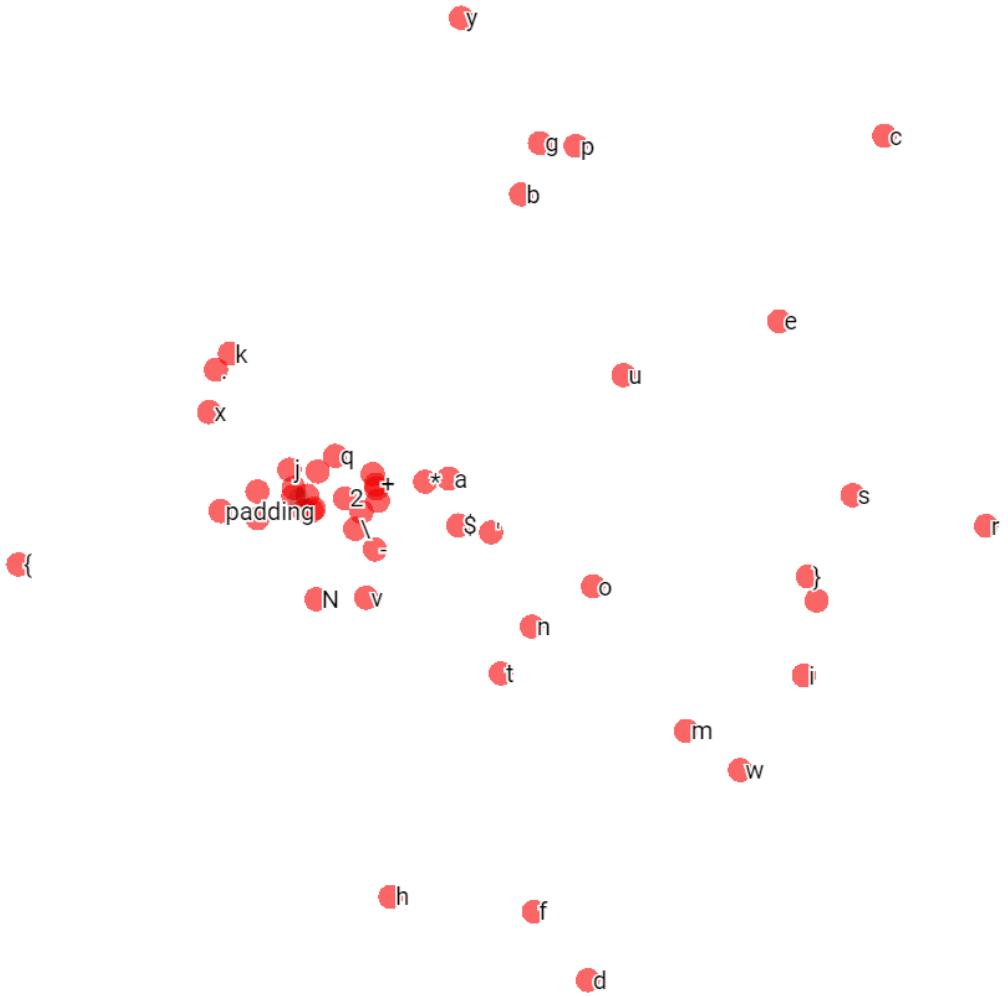

Figure 10 - Projection en 2D des encodages des caractères en 15 dimensions de l'expérimentation avec 2 couches *Highway* et avec injection d'erreurs par ACP. Les deux axes sélectionnés représentent 36.8% de la variation totale.

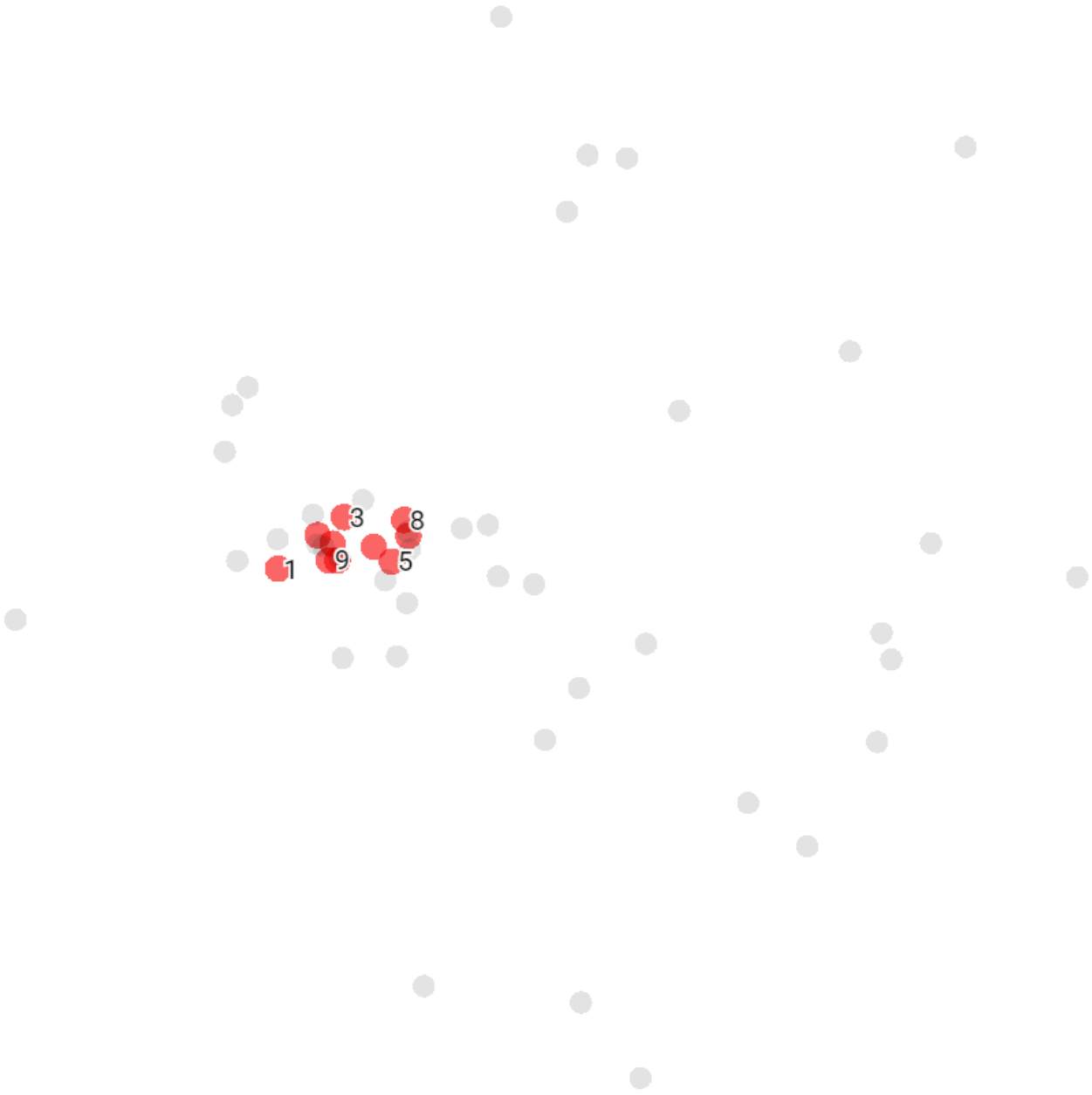

Figure 11 - Projection en 2D des encodages des nombres de l'espace original de 15 dimensions de l'expérimentation avec 2 couches *Highway* et avec injection d'erreurs par ACP. Les deux axes sélectionnés représentent 36.8% de la variation totale.

Discussion

Le but étant de trouver le meilleur modèle, il est important de comparer les résultats des tests de chaque modèle. En ce qui concerne le test sans injection d'erreurs, le modèle le plus performant est celui sans couche *Highway* et sans injection d'erreurs. Pour le test avec injection d'erreurs, il s'agit du modèle sans couche *Highway* et avec injection d'erreurs. Cependant, si l'on regarde l'écart de performance dans pour les catégories où ces deux modèles ne sont pas les plus performants, soit avec injection d'erreurs pour le premier modèle et sans injection d'erreurs pour le deuxième, c'est le deuxième qui a la plus petite différence. De ce fait, il serait débatable que le deuxième modèle est donc le meilleur en général pour deux raisons. Premièrement, il est meilleur dans la catégorie correspondante au but de l'expérience qui est de détecter les erreurs. Deuxièmement, l'intention derrière le test sans injection d'erreurs est de vérifier à quel point les modèles déclenchent de faux positif, c'est-à-dire qu'ils détectent une erreur alors qu'il n'y en a pas. Le premier modèle n'ayant jamais vu d'erreur auparavant, il est naturel qu'il déclenche moins de faux positifs. Il est en quelque sort moins sensibles à celles-ci. Alors, le fait que le deuxième est une performance presque aussi bonne démontre qu'il ne détecte pratiquement pas de faux positifs alors qu'il est plus sensible aux erreurs. Il remplit donc mieux la tâche demandée.

Enfin, la représentation en 2D des encodages de caractères ne sert qu'à donner une intuition quant à la manière que les modèles les représentent. En effet, une représentation sur deux axes d'un espace qui en contient originalement 15 ne peut pas être complètement fidèle. Cependant, on peut voir qu'il y a certaines continuités entre les différents encodages. Par exemple, les figures 5, 7, 9 et 11 semblent indiquer que les nombres sont toujours regrouper plus ou moins ensemble. Certains modèles, comme ceux avec injections d'erreurs, semblent les regrouper de manière plus serrée que les modèles sans injection d'erreurs. De plus, le symbole N, qui représente les nombres plus grands que des années, et $ ne sont jamais bien loin. Cela pourrait s'expliquer par le fait que N représente souvent des montants d'argent. Par contre, il est difficile de comprendre quels liens sont représentés par la position des lettres qui changent de manière assez drastique d'un modèle à l'autre.

Finalement, on peut voir que de manière générale, les expérimentations effectuées avec une couche de type *Highway* performent moins bien avec une perplexité et une perte plus élevée sur un ensemble de données de test. Nous émettons l'hypothèse que cela est dû à un surentraînement du modèle.

Analyse critique

Afin de réaliser ce projet, nous avons eu à appliquer certains concepts d'intelligence artificielle que nous avons vu en classe de concert avec la librairie TensorFlow. Nous avons choisi d'utiliser cette librairie dès le début et nous avons décidé de réaliser un projet portant sur le traitement du langage naturel étant donné que le prétraitement et l'étiquetage des données semble être plus facile à appliquer sur ce genre de données ce qui nous permet de passer plus de temps à explorer des facettes du projet qui nous semble plus intéressantes comme l'architecture du réseau de neurones profond.

Il aurait probablement été plus sage de ne pas se restreindre à une seule librairie d'intelligence artificielle en début de projet et de plutôt rechercher la librairie qui soit la mieux adaptée pour résoudre notre problème.

Nous avons commencé par explorer la documentation et les tutoriels qui étaient offerts par TensorFlow. De plus, nous avons recherché les modèles généralement utilisés pour effectuer de la correction d'erreurs ou du traitement de langage naturel plus général. Nous nous sommes rendus compte que l'encodage des mots et l'utilisation de réseaux récurrents étaient très populaires dans ce domaine. Cependant, il était difficile pour nous de voir comment effectuer de la correction d'orthographe en encodant les mots étant donné que cela impliquerait de maintenir un vocabulaire très large des mots incorrectement écrits. Nous avons donc décidé de procéder par encodage des caractères qui est une pratique relativement récente comparativement à l'encodage des mots.

Toutefois, étant donné que la pratique de l'encodage par caractère est relativement récente, nous avons eu de la difficulté à trouver de l'information. En effet, intégrer l'encodage avec une couche récurrente n'était pas tout à fait évident pour nous. Aussi, comment le résultat de cette couche serait interprété pour identifier les mots correctement écrits était une question qui est resté longtemps sans réponse.

Au cours de notre recherche portant sur l'architecture de notre réseau, nous avons trouvé un modèle utilisé pour effectuer de la prédiction de mots. Celui-ci correspondait avec le modèle que nous avions en tête pour effectuer de la correction d'orthographe. Nous avons donc décidé d'opter pour un modèle déjà éprouvé. Il nous semblait plus intéressant de comprendre le fonctionnement d'un modèle fonctionnel et de changer son application, que de faire le nôtre à partir de rien. Il aurait sans doute été intéressant pour nous de tenter de créer notre propre modèle par essaies et erreurs. Le modèle final aurait probablement été bien moins élaboré. Cependant, le processus nous aurait amené d'autres acquis que nous n'avons pas nécessairement acquis en prenant un modèle déjà fait.

Conclusion

Dans le cadre de ce projet, nous avons exploré le domaine du traitement du langage naturel par l'intelligence artificielle. Plus précisément, nous avons tenté de réutiliser un modèle fonctionnel servant à la prédiction de mots, et l'appliquer à la correction d'erreurs d'orthographes.

Au cours de nos expériences, nous avons comparé les résultats obtenus en utilisant des couches de type *Highway* et en appliquant notre injection d'erreurs. Nous pouvons en conclure que le modèle semble mieux s'adapter lors des erreurs sont incorporés dans le texte, ce qui n'est pas étonnant étant donné qu'il s'agit d'une forme d'augmentation des données. Notre modèle semble également moins bien performer avec les couches *Highway* sur un ensemble de données de test. Nous émettons l'hypothèse que cela est dû à un surentraînement.

Finalement, ce projet nous a permis de mieux comprendre le fonctionnement des réseaux de neurones ainsi que le rôle de chacun des types de couche que nous avons utilisés.

Références

Bengio, Y., Simard, P., & Frasconi, P. (1994, mars). Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions On Neural Networks, 5*(2), 157-166. Récupéré sur http://www-dsi.ing.unifi.it/~paolo/ps/tnn-94-gradient.pdf

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., & Brants, T. (2013). *One Billion Word Benchmark for Measuring Progress in.* Récupéré sur arxiv: https://arxiv.org/pdf/1312.3005.pdf

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. Récupéré sur https://arxiv.org/pdf/1103.0398.pdf

Google. (s.d.). *Vector Representations of Words*. Récupéré sur TensorFlow: https://www.tensorflow.org/tutorials/word2vec

Hochreiter, S., & Scmidhuber, J. (1997, novembre 15). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780. doi:10.1162/neco.1997.9.8.1735

Kim, Y., Jernite, Y., Sontag, D., & Rush, A. (2015, août 26). *Character-Aware Neural Language Models.* Récupéré sur arxiv: https://arxiv.org/abs/1508.06615

Kroutikov, M. (n.d.). tf-lstm-char-cnn. Récupéré de https://github.com/mkroutikov/tf-lstm-char-cnn

Lei, T., Barzilay, R., & Jaakola, T. (2015). Molding CNNs for text: non-linear, non-consecutive convolutions. Récupéré sur https://arxiv.org/abs/1508.04112

Marcus, M. P., Marcinkiewicz, M., & Santorini, B. (1993, juin 02). Building a large annotated corpus of English: the penn treebank. *Computational Linguistics - Special issue on using large corpora: II, 19*(2), 313-330. Récupéré sur http://dl.acm.org/citation.cfm?id=972475

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Récupéré sur https://arxiv.org/abs/1301.3781

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic Regularities in Continuous Space Word Representations. Dans *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013).* Association for Computational Linguistics. Récupéré sur https://www.microsoft.com/en-us/research/publication/linguistic-regularities-in-continuous-space-word-representations/

Moris, C. (2016, septembre 21). *Dissecting Google's Billion Word Language Model Part 1: Character Embeddings.* Récupéré sur colin_moris: http://colinmorris.github.io/blog/1b-words-char-embeddings

Olah, C. (2015, août 27). *Understanding LSTM Networks.* Récupéré sur colah's blog: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014, juillet). Dropout: A Simple Way to Prevent Neural Networks from. (Y. Bengio, Éd.) *Journal of Machine Learning Research*, 1929-1958. Récupéré sur https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM Neural Networks for Language Modeling. *Human Language Technology and Pattern Recognition*. Récupéré sur http://www.quaero.org/media/files/bibliographie/sundermeyer_lstm_neural_interspeech2012.pdf

Turian, J., Ratinov, L., & Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. Récupéré sur http://www.aclweb.org/anthology/P10-1040