Projet – Intelligence Artificielle

Développement d'un algorithme de prédiction des mots sur Python à l'aide du groupe de modèles Word2Vec

BRUNET Nathan – MICHOT Vincent

Sommaire

I - Introduction

Ce projet a pour thématique le traitement automatique des langages, qui apparaît dans beaucoup de domaines : saisie de texte, traduction de texte, prédiction de mots, lecture automatique et transposition numérique d'une écriture à la main… La méthode Word2Vec est une méthode développée par Google (depuis 2013) qui permet, en "vectorisant" les mots, de faire des calculs vectoriels sur ceux-ci afin d'appliquer les connaissances mathématiques à un domaine qui ne pouvait pas en bénéficier auparavant. Cela permet entre autres de manipuler des mots (indirectement avec les vecteurs correspondant) avec des réseaux de neurones informatiques.

II - Objectif du projet

L'objectif de ce projet est de créer un réseau de neurones qui, étant donné un mot au milieu d'une phrase (entrée), nous retournera la probabilité pour chaque mot que l'on aura dans notre dictionnaire de se situer dans une fenêtre d'une certaine taille centrée autour de ce mot (fenêtre de n mots avant et après le mot d'entrée).

III - Méthode générale

Tout d'abord, il faut "vectoriser" les mots de notre dictionnaire pour présenter ces vecteurs au réseau de neurones car celui-ci ne reconnait pas les éléments de type string. Nous sommes en possession d'un dictionnaire de 10 000 mots (les 10 000 plus courants dans un texte de 17 000 000 de mots).

L'idée est de représenter un mot du dictionnaire par un vecteur colonne de taille 10 000*1 qui possède exclusivement des 0, sauf à l'emplacement unique où il apparait dans le dictionnaire : il y aura un 1 à cet indice (ce vecteur sera l'entrée de notre réseau de neurones (input). En sortie, on aura aussi un vecteur colonne de 10 000 composantes, contenant cette fois les probabilités respectives pour chaque mot du dictionnaire d'être dans la fenêtre du mot d'entrée.

a) Structure du réseau de neurones

La première couche de neurones est une couche cachée (car on ne voit pas ce qui en ressort) ou « *hidden layer »,* dont le rôle sera de créer 300 caractéristiques relatives au mot d'entrée (donc la couche contient 300 neurones). Ces caractéristiques vont varier pour chaque mot du dictionnaire : chacun des 10 000 mots du dictionnaire se verra associer ses 300

caractéristiques (300 poids qui définiront celles des 300 caractéristiques qui seront les plus importantes pour établir si le mot est proche du mot d'entrée).

On obtient par ce procédé une matrice de taille 10 000*300 dont chaque ligne correspond à un mot du dictionnaire et contient les 300 caractéristiques associées à ce même mot.

La dernière couche de neurones (qui va donner directement la sortie du réseau) est quant à elle composée de 10 000 neurones, où chacun va donner la probabilité pour un mot du dictionnaire de se situer dans la fenêtre du mot d'entrée. Cette dernière couche de neurones utilise Softmax pour fournir cette probabilité recherchée, ceci à l'aide des caractéristiques du mot.

Dans notre étude, on commence par un texte plus court que celui de 17 000 000 de mots, donc on utilise moins de 300 caractéristiques (ici 2) ; on aura donc 2 neurones sur la couche cachée (choix de 2 neurones expliqué en partie suivante). Le texte étant moins long, le dictionnaire le sera aussi, donc la dernière couche (output layer) sera composée de moins de neurones également.

b) Méthode employée pour le code Python (implémentation)

On part d'un texte brut, avec lequel on crée un dictionnaire, puis on indexe chaque mot du dictionnaire par un entier (Word2Int). On parcourt chaque mot (x) du texte brut et l'on créé toutes les listes de 2 mots contenant x et ses voisins (voisins définis par la taille de la fenêtre).

Soit par exemple le texte brut initial suivant : « il y a deux projets principaux » : les listes de deux mots pour une fenêtre de taille 2 sont : [il,y], [il,a], [y,il], [y,a], [y,deux], [a,il], [a,y], [a,deux], [a,projets]… On a donc chaque mot d'entrée et toutes les sorties voulues (notées « label » dans le code) : cela constitue ainsi deux listes que l'on va entrainer. Ces mots sont indexés par des entiers que l'on va alors vectoriser afin de pouvoir les présenter à un réseau de neurones.

Pour chaque mot, on créée un vecteur colonne de la taille du dictionnaire, contenant uniquement des zéro, sauf un 1 à l'indice du mot (défini par Word2Int). On créé dès lors notre réseau de neurones avec la première couche qui est la couche cachée : elle contient 2 neurones ce qui donnera 2 poids, soit un par caractéristique. On pourra assimiler ces deux poids à des coordonnées de points.

La deuxième couche de neurones a une fonction d'activation (Softmax) et va réaliser la prédiction (détermination des poids pour chaque mot du dictionnaire) : en sortie on aura donc, pour chaque mot, des coordonnées. Plus deux mots seront proches, plus ils auront de probabilité d'être dans la même

fenêtre. On calcule donc la distance de chaque mot au mot d'entrée dont on veut connaître le voisin le plus probable, et on divise cette distance par la somme de toutes les distances afin d'obtenir une probabilité. Le mot ayant la probabilité la plus faible et différente de 0 sera celui retenu.

IV - Résultats obtenus et analyse

On a effectué différents tests pour ce code (variables : longueur du texte, taille de la fenêtre). Ces tests ont pour but de mesurer la répétabilité du code et l'efficacité de l'entrainement.

a) Test 1

Un premier test avec peu de mots dans le texte brut a permis de visualiser sur un graphe 2D la répartition des mots les uns par rapport aux autres. En lançant plusieurs fois ce code, on observe que l'allure de la répartition ne varie pas beaucoup. Le mot proposé est cohérent avec le mot d'entrée et on arrive à vérifier que pour 73% des essais, le mot retenu en sortie appartient à la fenêtre (de taille 2 ou 3) du mot d'entrée. Quand on augmente la taille de la fenêtre (5 à 7 mots autour du mot d'entrée) on a un pourcentage moyen de 78%. Cela constitue une légère hausse, guère significative cependant, au vu de la taille restreinte du texte brut.

b) Test 2

Le deuxième test a été fait avec un texte beaucoup plus long : 0.1% du texte mis à disposition par Google (txt8). Impossible ici de visualiser nettement les mots les uns par rapport aux autres sur un graphe 2D. On observe pour ce code que pour une fenêtre de 3 mots on tombe à 64% d'efficacité. De plus le temps de calcul est très long.

c) Améliorations de la méthode

Pour améliorer ce code, on peut omettre la représentation 2D de la répartition des mots et mettre plus de neurones dans la couche cachée (hidden layer) ce qui donnera plus de poids (caractéristiques) qui permettront d'être plus discriminant sur les mots à choisir. On pourrait, en ajoutant des neurones, encore calculer une distance entre chaque mot. On pourrait tester une fonction d'activation différente pour voir si elle convient mieux à la prédiction.

d) Limites de l'exercice

Dans un premier temps, nous n'avons réussi, avec la puissance de calcul des ordinateurs mis à disposition, à effectuer le traitement que d'une maigre

partie du texte de 17 millions de mots d'origine (approximativement 0.1% de celui-ci). Conséquemment, nous n'avons pu obtenir un dictionnaire extrêmement étendu, voire très peu en réalité, étant constitué de quelque 1000 mots, ce qui n'était pas vraiment satisfaisant pour l'apprentissage qui allait suivre. Avec si peu de mots comparativement au texte initial, cela était très long, et afin d'obtenir le dictionnaire de 10 000 mots comme initialement demandé avec les ordinateurs en présence, cela eût pris beaucoup trop de temps et présenté peu d'intérêt. Nous avons donc opté pour une diminution drastique du texte brut initial.

De plus, il était possible d'aboutir à des incohérences dans les associations proposées au moment de l'entraînement, dans la mesure où la ponctuation n'était pas prise en compte. En effet, le premier mot d'une phrase pouvait parfois être associé avec le dernier de la phrase précédente (suite de mots insensée), tout ceci étant également dû aux incohérences inhérentes au texte d'origine, avec les conséquences évoquées précédemment lors de l'entraînement.

V - Conclusion

Le code retenu est à même d'établir un dictionnaire et d'effectuer une tâche de suggestion de mots, mais la puissance des ordinateurs à disposition le rendait dans notre contexte d'étude assez peu intéressant à utiliser finalement, c'est-à-dire pour le traitement de textes beaucoup plus grands que ceux avec lesquels nous avons procédé. Il aurait donc été profitable de tester le code élaboré sur une machine de puissance supérieure, afin d'obtenir des résultats plus probants dans le cadre de l'approche Word2Vec de la suggestion automatique.