Factory de mots de passe

Abstract

Dans cet article, nous discutons de l'efficacité pour la génération de mots de passe de différents types de réseaux de neurones. Nous disposons d'un modèle simple de `RNN`, nous nous inspirons de celui-ci pour entraîner d'autres modèles. Nous comparons alors l'efficacité des `LSTM` et de certaines de ses variantes comme les `Bidirectional LSTM`, `Stacked LSTM` et `GRU`. Nous faisons une revue de la littérature puis de l'existant concernant ces modèles. Nous discutons du corpus fourni ainsi que de la représentation vectorielle de nos données pour l'entraînement de nos modèles. On observe que les modèles cités précédemment, à l'exception du `Bidirectional LSTM`, sont supérieurs au `RNN` simple pour la génération de séquences. En moyenne 20% de mots de passe générés par notre meilleur modèle sont présents dans la liste de mot `rockyou`. Le nombre de neurones par couches semble avoir un impact sur nos résultats. Plus ils sont nombreux mieux c'est, au prix d'un temps d'entraînement plus long.

1. Introduction

Dans le cadre de l'UE application d'innovation, nous avons choisi le sujet factory de mots de passe, proposé par `M. Morchid` et `M. Haddad`. L'objectif du sujet est de nous permettre de nous informer sur les différentes phases constituant l'application d'innovation-sécurité. De plus nous devons proposer un générateur de mots de passe entraîné et évalué avec les données d'apprentissage et d'évaluation fournies. Ce générateur de mots de passe doit être inspiré du modèle proposé. Pour proposer une amélioration du modèle, il faut d'abord l'utiliser et expérimenter avec. Nous avions pour ça, un autre jeu de données, afin d'entraîner ce modèle et de générer des noms russes. Ce modèle est un réseau de neurones simple (RNN), écrit avec le package `pytorch`.

Un réseau de neurones récurrents (Recurent Neural Network) est un modèle de réseau de neurones qui prend en compte les entrées précédentes en ajustant les poids avec chaque entrée. On l'utilise pour la prédiction de séquences, où l'ordre est important, comme pour la génération de mots de passe ou de prénoms. Les réseaux de neurones récurrents sont sujets à certains problèmes concernant la capacité de ce dernier de garder en "mémoire" les données entrées. Il s'agit du `vanishing gradient` ainsi que du `exploding gradient`. Le premier intervient quand le gradient devient trop faible, la mise à jour de poids devient alors insignifiante, dans le pire des cas le RNN peut ne plus apprendre. Les informations des premières couches ont alors de moins en moins d'importance, les RNN peuvent être moins efficaces pour traiter les données redondantes ou hétérogènes. Quant à l'explosion du gradient, la mise à jour des poids est bien trop importante à chaque itération. Ce qui résulte en un modèle bien trop instable et donc en des prédictions moins bonnes. Il est donc compliqué d'entraîner un RNN, ces problèmes sont décrits par Y. Bengio et al. [1].

Il existe d'autres types de réseaux comme les LSTM (Long Short-Term Memory), qui ont été développés pour remédier à certaines des limites des réseaux neuronaux récurrents (RNN) traditionnels. Les activations des neurones d'un RNN peuvent se diluer ou être oubliées au fur et à mesure du traitement de la séquence, ce qui rend difficile pour le réseau de conserver une mémoire à long terme de la séquence. Les LSTM résolvent ce problème de "`vanishing gradient`" en introduisant un type spécial de neurone appelé cellule mémoire, qui est capable de maintenir son état à travers plusieurs étapes temporelles. Cela permet aux LSTM d'apprendre à partir de longues séquences de données et de modéliser des dépendances complexes, ce qui en fait un outil plus puissant et plus souple pour la modélisation des données séquentielles que les RNN traditionnels.

Comme les `LSTM` sont capables de maintenir leur état à travers plusieurs étapes temporelles, ils peuvent apprendre à partir des informations redondantes de la séquence et filtrer le bruit, ce qui leur permet de faire des prédictions plus précises que les RNN.

L'un des principaux inconvénients des réseaux LSTM est que leur apprentissage peut être coûteux en termes de calcul, en particulier pour les très longues séquences de données. Cela s'explique par le fait que les LSTM ont un plus grand nombre de paramètres que les RNN traditionnels et que leur apprentissage nécessite davantage de ressources. En outre, les LSTM peuvent être difficiles à interpréter et à expliquer, en particulier lorsqu'ils sont appliqués à des tâches complexes. Enfin, dans certains cas, des modèles plus simples ou d'autres types de RNN peuvent être plus performants. Il existe cependant plusieurs variantes au LSTM décrites par K. Greff et al. [2], en plus des `Vanilla LSTM`, il existe une variante nommée Gated Recurrent Units (GRU).

Les `GRU` (Gated recurrent units) sont un type de réseau neuronal récurrent (RNN) similaire aux réseaux à mémoire à long terme (LSTM). Elles ont été développées pour remédier à certaines des limites des RNN traditionnels cités précédemment. Les GRU sont capables de surmonter celles-ci en introduisant des "`gate`", qui sont des filtres qui contrôlent le flux d'informations entrant et sortant du réseau. Cela permet aux GRU d'apprendre quelles informations retenir et quelles informations oublier, ce qui les rend plus efficaces pour apprendre à partir de séquences de données.

Parmi les principaux avantages des GRU, leur simplicité, ce qui les rend plus faciles à entraîner et plus rapides à exécuter que les LSTM. Les GRU sont capables de filtrer les informations redondantes et d'apprendre à partir des informations les plus pertinentes ou importantes de la séquence, là où les LSTM retiennent l'information. Ce qui explique en partie l'entraînement plus efficace et donc plus rapide des GRU. Cela permet aux GRU de faire des prédictions plus précises à partir de données redondantes et d'apprendre de longues séquences de données plus efficacement que les RNN traditionnels.

Y. Bengio et al. [3] dans leur article arrivent à la conclusion que les modèles LSTM et GRU étaient supérieurs aux RNN traditionnels pour la modélisation de séquences. Cependant ils ne peuvent pas clairement conclure sur le meilleur des deux `gating units`.

Nous avions donc le choix entre LSTM ou GRU pour améliorer le générateur fourni. Pour notre modèle nous utiliserons la bibliothèque KERAS, pour la simple raison que nous avons déjà utilisé cette bibliothèque python dans une autre UCE et que cette dernière nous est familière. Keras nous propose une solution pour utiliser ces deux `gating unit`. Nous utiliserons donc les deux, ainsi que d'autres alternatives au `Vanilla LSTM` comme les `Stacked LSTM` et les `Bidirectional LSTM` sur 20 époques avant de choisir notre modèle.

Dans cet article, nous analysons le corpus d'entraînement puis nous expliquons quelle représentation vectorielle nous utilisons pour les données en entrée de nos modèles. Avant de conclure, nous analysons les résultats obtenus.

2. Méthodologie

2.1. Analyse du corpus

Pour l'entraînement et l'évaluation de notre réseau de neurones, nous avons à notre disposition une archive contenant deux fichiers, train.txt ainsi que eval.txt, le premier pour l'entraînement de notre modèle et le dernier pour l'évaluation de celui-ci.

Notre corpus d'entraînement contient 375_853 mots de passe. Les tailles des mots de passe sont variables, du plus petit à 1 caractère jusqu'au plus grand avec 110 caractères. La distribution des tailles de mots de passe nous montre que la grande partie de ceux-ci font moins de 20 caractères de long. En effet, 99% des mots de passe ont une longueur comprises entre 3 et 14 caractères. Cette répartition semble cohérente, on l'explique en partie par le fait que la plupart des sites internet demandant des identifiants de connexion requièrent un minimum de caractères. De plus, les mots de passe longs sont moins susceptibles d'être choisi de par la difficulté relative de retenir ces derniers. On décide donc pour l'entraînement du modèle de nous concentrer sur ces 99% de mots de passe, la figure 1 nous montre la nouvelle répartition des tailles.

Cette difficulté à retenir des mots de passe explique aussi en partie le fait que la grande majorité suivent un pattern. La plupart de ces mots de passe correspondent à des mots ou des suites de mots accompagnés de chiffres. De ce fait beaucoup de ces mots de passe sont simples. Comme on peut l'apercevoir sur la figure 1, la majorité des mots de passe, peut importe leurs longueurs, ne contiennent pas de caractères spéciaux. En effet les caractères spéciaux représentent seulement 0.085% des caractères présents dans le corpus d'entraînement alors qu'ils représentent 32% de notre dictionnaire. Nous décidons quand même de garder ces caractères, nous pensons qu'il est préférable qu'un caractère ait une possibilité quasi nulle d'être prédit que de le supprimer. Cette présence de caractères spéciaux est même assez élevée par rapport à d'autres liste de mot comme rockyou.txt où seulement 0.013% des caractères du fichier sont dit spéciaux.

Table 1: *Notre dictionnaire*

Type	**Symbole**
Lettres	abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
Chiffres	0123456789
Caractère spé.	! ” # $ % & ’ ( ) * + - . / : ; < ? @ [\] ^ _ ‘ | ∼
Token de début	\t
Token de fin	\n

2.2. Représentation vectorielle

Pour que nos données soient utilisable par le modèle, il faut modifier la représentation vectorielle de ces dernières. En effet nous disposons de beaucoup mots de passe de différentes tailles, alors que notre modèle ne peut traiter que des entrées avec une représentation vectorielle similaire. Nous devons donc dans un premier temps générer notre dictionnaire de caractère auquel nous assignerons un index ([a, 1], [b, 2], etc..) , avec ces données le dictionnaire est de taille 92. Ensuite nos générons notre matrice X_train et y_train. Chaque mot de passe est ajouté à cette première matrice sous la forme d'une liste de token, chaque caractère est remplacé par son indice dans le dictionnaire généré précédemment, le mot ”Hello” devient alors la liste suivante: [41, 70, 77, 77, 80]. Quant à la seconde matrice y_train, elle contient à chaque indice la même liste de token décalée par la droite d'un caractère. De plus on ajoute des tokens de début et de fin à nos listes, respectivement \t et \n. Le mot ”Hello” dans nos matrices prend donc la forme suivante:

- X_train[0]: [0, 41, 70, 77, 77, 80]
- y_train[0]: [41, 70, 77, 77, 80, 1]

Avec les caractère associé a chaque indice:

- X_train[0]: [\t, H, e, l, l, o]
- y_train[0]: [H, e, l, l, o, \n]

L'entraînement permet donc de prédire le prochain caractère en fonction du caractère précèdent.Avec l'exemple ci-dessus, quand on donne ”\t” au modèle, on l'entraîne à générer un ”H”, avec ”H” on l'entraîne à générer un ”e” et ainsi de suite. Finalement on encode à chaud (One Hot Encoding) chaque mot de passe. La représentation vectorielle finale de X_train et y_train est la suivante: [374863, None, 94]. La première dimension pour les 374_863 mots de passe. La deuxième, None, car ces mots de passe sont de tailles variables et que nous utilisons des raggedTensor, dans les faits cette dimension prend des valeurs entre 3 et 14. La dernière, 94, quatre-vingt-treize 0 et un seul 1, où l'indice du 1 correspond à l'indice du caractère dans notre dictionnaire (One Hot Encoding). En donnant un caractère au modèle, on souhaite obtenir une liste de probabilités du prochain caractère.

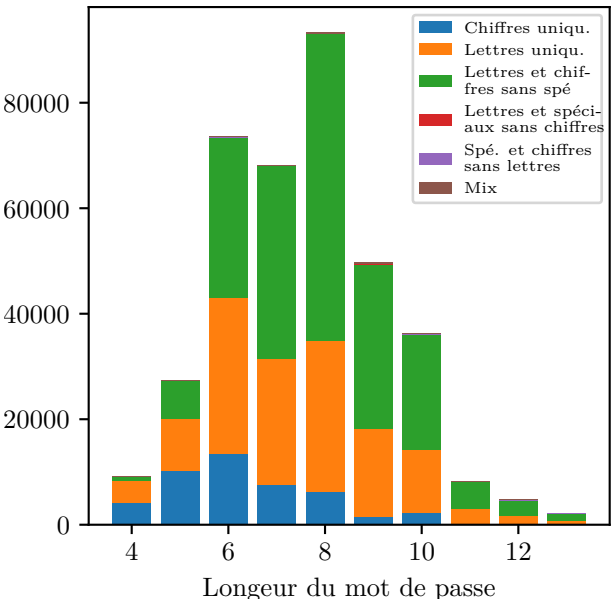

Figure 1: *Histogramme, visualisation des types de mots de passe pour chaque longueur de mots de passe. Avec les 375_853 mots de passe de notre corpus d'entraînement.*

Table 2: *Précision des modèles, pour 2000 mots de passe générés à l'époque 20 de leur entraînement.* `unit` *ou neurone, représente la forme de la sortie (outpus shape) et donc le nombre de neurones de la couche. 21.23% signifie que 427 des 2000 mots générés sont présents dans le fichier rockyout.txt.*

Modèle	**unit**	**train.txt**	**rockyout.txt**
RNN	90	0%	8.45%
GRU	128	6.8%	19.45%
LSTM	256	8.05%	21.35%
Stacked LSTM	128	6.7%	19.8%
Bi-LSTM	128	0%	0%

3. Résultats

Pour vérifier que les mots de passe générés correspondaient à des mots de passe qui auraient pu être utilisés par des vraies personnes, nous avons décidé de confronter ces derniers à la liste de mots de passe rockyou.txt. Cette liste contient des mots de passe rendus public, le fichier que nous avons utilisé en contenait 14 millions. Nous avons donc décidé de générer 2000 mots de passe à intervalles réguliers de 5 époques pendant l'entraînement de nos modèles pour juger de la précision de ce dernier. La table 2 présente nos résultats dans la 3ème colonne. On s'aperçoit que le modèle à la troisième ligne (LSTM) obtient les meilleurs résultats.

On observe avec la table 2 que le modèle RNN de base obtient les moins bon résultats parmi les modèles qui n'ont pas eu de problèmes à l'entraînement. Le modèle Bi-LSTM n'en fait pas partie, dès les premières époques les mots générés étaient vides, il n'est pas évident d'en trouver les causes, la représentation vectorielle des données en entrée n'était probablement pas la bonne. Nous avons par la suite entraîné un autre modèle avec une couche GRU avec pour forme de sortie: (None, None, 128). Nous avons entrainé un autre modèle avec cette fois deux couches LSTM à la suite, avec la même forme de sortie que le modèle précèdent. On remarque alors que nous obtenons des résultats similaires. Le point commun de ces deux modèles c'est la forme de leurs sorties, alors nous avons entraîné un autre modèle avec une forme de sortie différente dans l'espoir d'améliorer nos précédents résultats. Nous utilisons une simple couche LSTM pour notre dernier modèle avec cette fois une forme de sortie multiplié par deux: (None, None, 256). La proportion de "véritable" mots de passe générés passe de 19% à 21%, c'est une amélioration de plus ou moins 2% et donc une différence de 40 mots de passe. Le nombre de neurones par couche (unit) semble affecter positivement nos résultats plus ils sont nombreux. Il est tentant d'essayer d'entraîner un modèle avec des couches possédant plus de neurones, passer de 128 à 256 représente une différence d'une minute pour l'entraînement d'une couche. Cependant passer de 256 neurones à 1024 représente une différence de plus d'une heure pour une seule époque. On passe alors de plus ou moins 6h d'entraînement à plus de 81h avec 1024 neurones par couche. Tous nos modèles ont une couche `Dense` pour obtenir une sortie que nous pouvons interpréter et donc générer nos caractères selon les probabilités obtenues. De plus on utilise la fonction CategoricalCrossentropy de Keras pour calculer la loss à chaque époque, nos faisons ce choix car nous disposons de représentation "one hot" de notre dictionnaire. Pour compiler nos modèles, on utilise l'optimiseur `Adam`, après des tests sur des jeux de données différents, l'optimiseur Adam était le plus performant. Nous avons donc décider d'entraîner notre meilleur modèle sur 50 autres époques pour peut être améliorer ces résultats.

On observe avec la figure 2 que la loss sur 70 époques converge vers 2,35. Pendant l'entraînement nous avons généré 2000 mots de passe toutes les 5 époques, pour mesurer son efficacité. La figure 3 nous permet de nous rendre compte de l'efficacité du modèle. On observe que plus ou moins 20% des mots de passe générés sont des mots de passe qui ont déjà été utilisés par des utilisateurs, ces mots de passe font partie de la liste de mots rockyou.txt. De plus la proportion de ces deniers présents dans notre corpus d'entraînement reste assez faible, ce qui montre que notre modèle n'a pas surappris, en effet on constate que plus ou moins 6.5% de ces mots générés sont présents dans le fichier train.txt.

La figure 1 montre la répartition des types de mots de passe de notre corpus d'entraînement. La figure 4 montre cette même répartition mais sur les 2000 mots de passe générés par notre modèle à la 70ème époque. On observe une très forte similitude entre ces deux graphiques à barres. C'est encore une fois un bon indicateur sur l'efficacité du modèle qui plus est sur un faible nombre de mots générés. Les proportions de types de mots de passe est similaire, pratiquement identique, pour chaque longueur de mots. Avec "seulement" 6 à 7% des mots générés présents dans le corpus d'entraînement, il est facile d'imaginer et d'extrapoler sur la pertinence des mots de passe prédits par le modèle. Le fait qu'en moyenne 20% de ces derniers soient présents dans la liste

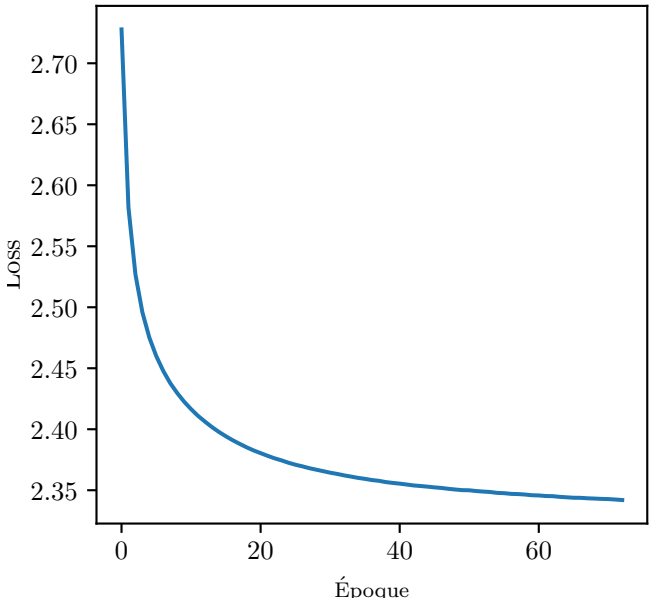

Figure 2: *Visualisation de la métrique Loss sur 70 époques.*

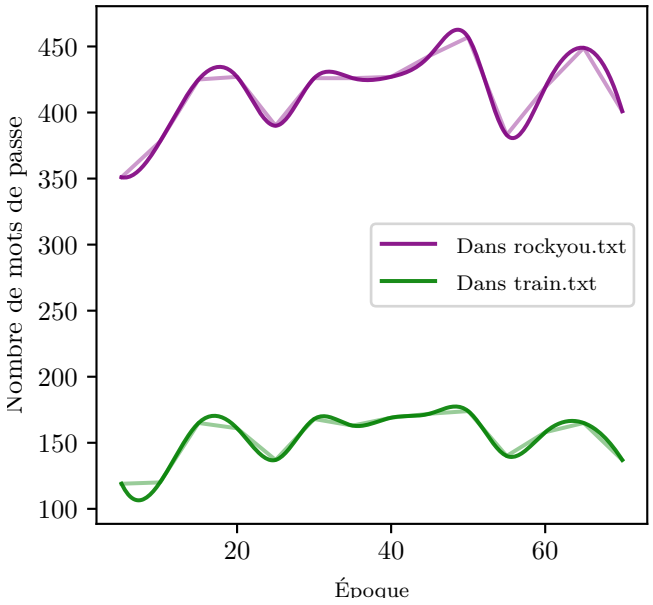

Figure 3: *Nombre de mots de passe dans rockyou.txt et train.txt. Avec 2000 mots de passe générés toutes les 5 époques pendant l'entraînement du modèle 3 (LSTM).*

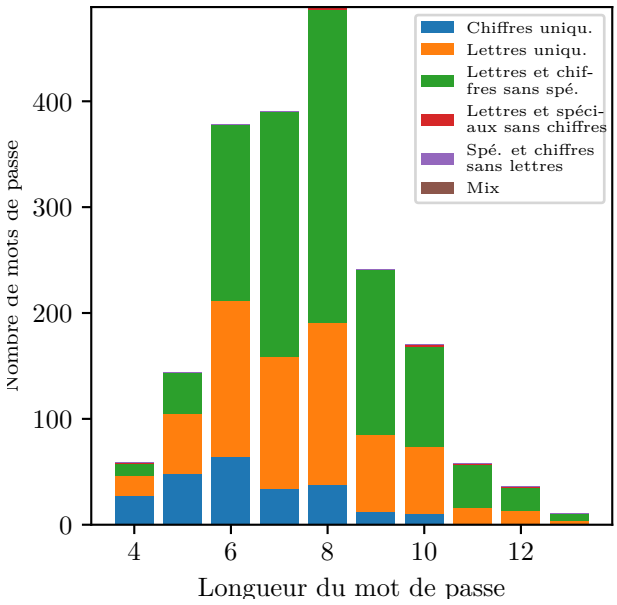

Figure 4: *Histogramme, visualisation des types de mots de passe pour chaque longueur de mots de passe. Avec 2000 mots générés à l'époque 70, avec le modèle 3 (LSTM).*

de mot rockyou appuie l'affirmation précédente, on rappelle que le modèle n'a jamais eu accès à ces données pendant son entraînement.

4. Conclusion

Il semble que l'utilisation de réseaux de neurones récurrents soit une bonne solution pour générer un dictionnaire de grande taille. Avec relativement peu de données et un modèle assez simple, nous obtenons des résultats satisfaisants. En disposant de ressources supplémentaires, nous pourrions entraîner de nouveau ce modèle avec une plus grande quantité de données, comme avec la liste de mots Rockyou par exemple, ou en augmentant la taille des batchs de données, ce qui pourrait améliorer les résultats. Cependant, il est intéressant de noter que les mots de passes suivent en règle générale un pattern de construction relativement simple pour un système informatique à reproduire, et ainsi remettre en cause la sécurité du mot de passe. La plupart des mots de passe observés affichent des combinaisons de mots et de chiffres. C'est pourquoi il est recommandé de complexifier le plus possible le mot de passe, en utilisant des caractères spéciaux, des majuscules, des minuscules, des chiffres, et en évitant les mots de passe trop courts, le tout en suivant aucun pattern pour se protéger au mieux des attaques par dictionnaire. Enfin, le mot de passe est une solution relativement efficace et simple à mettre en oeuvre pour sécuriser ses données, mais aujourd'hui d'autres dispositifs peuvent et doivent être utilisés pour renforcer la sécurité de ses données, comme l'authentification à deux facteurs, ou encore l'utilisation de clés de chiffrement.

5. References

[1] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," arXiv:1211.5063, 2013.

[2] K. Greff, R. K. Srivastava, J; Koutník, B. R. Steunebrink and J. Schmidhuber, "LSTM: A Search Space Odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, Volume: 28, Issue: 10, Oct. 2017 Pages: 2222 - 2232.

[3] J. C. Caglar, G. K. Cho and Y. Bengio "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling " , arXiv:1412.3555, Dec. 2014.