Factory de mots de passe

Abstract

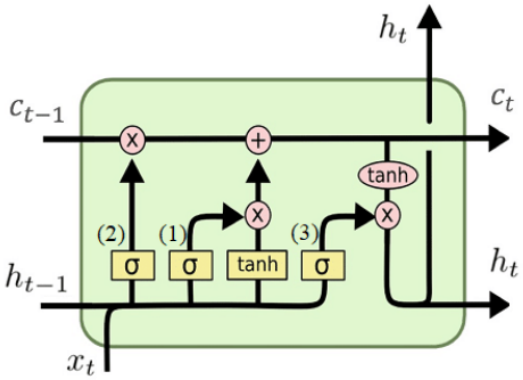
Ce projet se concentre sur la génération de mots de passe en exploitant des modèles d'apprentissage avancés tels que le RNN (Réseau de Neurones Récurrents), la LSTM (Long Short-Term Memory) et la LSTM bidirectionnelle. Initialement basé sur un modèle RNN simple, nous étendons nos efforts en utilisant diverses variantes de LSTM pour explorer les améliorations potentielles dans la génération de mots de passe. Notre approche s'appuie sur l'analyse comparative de l'efficacité de ces modèles, en tenant compte de facteurs tels que la complexité du réseau et le temps d'entraînement. Les résultats préliminaires suggèrent que les LSTM, en particulier la LSTM bidirectionnelle, présentent des performances prometteuses par rapport au RNN simple. Cette recherche vise à contribuer à une meilleure compréhension des modèles d'apprentissage pour la génération de mots de passe, avec des implications potentielles pour renforcer la sécurité des systèmes informatiques.

1. Introduction

Dans le cadre de l'Unité d'Enseignement (UE) sur l'application d'innovation, notre choix s'est porté sur le sujet de la "Factory de Mots de Passe", proposé par M. Morchid et M. Haddad. L'objectif de ce sujet est de nous familiariser avec les différentes phases constitutives de l'application d'innovation en sécurité. En outre, notre mission consiste à concevoir un générateur de mots de passe, formé et évalué à l'aide des jeux de données d'apprentissage et d'évaluation fournis. Ce générateur de mots de passe doit s'inspirer du modèle initial proposé.

Pour proposer des améliorations au modèle, nous devons d'abord le mettre en œuvre et le tester. À cette fin, nous disposons d'un jeu de données supplémentaire que nous utilisons pour entraîner ce modèle, en particulier pour la génération de noms russes. Ce modèle prend la forme d'un réseau de neurones simple (RNN) implémenté avec le package PyTorch.

Les réseaux de neurones récurrents (RNN) sont des types de réseaux de neurones artificiels spécialement conçus pour traiter des données séquentielles ou des séries temporelles. Ils sont fréquemment utilisés dans des domaines tels que la traduction linguistique, le traitement du langage naturel, la reconnaissance vocale et le sous-titrage d'images [1]. Cependant, malgré leur utilité, les RNN présentent certains défis, notamment le problème du "vanishing gradient". Ce problème survient lors de l'entraînement du réseau, où les gradients associés aux poids des couches précédentes diminuent exponentiellement, entraînant des difficultés dans la rétropropagation du gradient. Le "vanishing gradient" peut compromettre les performances des RNN, en limitant leur capacité à apprendre des dépendances à long terme dans les données séquentielles. Cette limitation est cruciale dans des applications telles que la modélisation de séquences temporelles où la compréhension des dépendances à long terme



Long Short-Term Memory, Schéma d'une cellule mémoire d'un LSTM.

est essentielle. Ainsi, malgré leurs avantages, la gestion du problème du "vanishing gradient" demeure un défi crucial dans l'optimisation des performances des RNN.

Ce problème a conduit au développement des réseaux LSTM, qui utilisent des mécanismes de régulation pour éviter le vanishing gradient et capturer efficacement les dépendances à long terme dans les séquences.

Les LSTM introduisent une nouvelle unité appelée cellule mémoire, qui permet au réseau de stocker et d'accéder à des informations sur une période étendue. La cellule mémoire d'un LSTM est composée de plusieurs portes : une porte d'entrée, une porte de sortie et une porte d'oubli. Ces portes régulent le flux d'informations à l'intérieur de la cellule mémoire, permettant ainsi de contrôler les informations à retenir et celles à oublier. Cela donne aux LSTM la capacité de mémoriser des informations importantes sur de longues séquences et d'ignorer les éléments moins pertinents [2].

Les LSTM bidirectionnels ont émergé comme une évolution naturelle pour étendre les capacités des LSTM. En permettant au modèle de traiter l'information dans les deux sens d'une séquence temporelle, les LSTM bidirectionnels ont introduit une approche novatrice dans le domaine des réseaux de neurones récurrents (RNN). En effet, un LSTM bidirectionnel est un type particulier de RNN qui excelle dans le traitement de données séquentielles en capturant simultanément les informations passées et futures de la séquence d'entrée. Cette méthode s'appuie sur la combinaison de la puissance inhérente aux LSTM avec la capacité de traitement bidirectionnel, résultant en une meilleure compréhension contextuelle et des relations plus riches entre les éléments de la séquence. À la différence des RNN traditionnels qui traitent les séquences dans une seule direction, les Bi-LSTM traitent la séquence dans les deux sens simultanément, grâce à deux couches LSTM distinctes : l'une gérant la séquence en avant,

tandis que l'autre en arrière. Chaque couche conserve ses propres états cachés et cellules de mémoire, renforçant ainsi la capacité du modèle à saisir des dépendances complexes dans les données séquentielles [3].

Dans le cadre de cet article, nous procédons à une analyse approfondie du corpus d'entraînement, détaillant également la représentation vectorielle spécifiquement employée pour les données en entrée de nos modèles. Enfin, pour conclure, nous examinons et discutons les résultats obtenus au cours de nos expérimentations.

2. Methodologie

2.1. Analyse du corpus

Pour former et évaluer notre réseau de neurones, nous disposons d'une archive comprenant deux fichiers, à savoir Ashly-madesson.txt, utilisé pour l'entraînement du modèle, et eval.txt, réservé à son évaluation.

Notre corpus d'entrainement contient **375853** mot de passe. Les longueurs des mots de passe varient, allant d'un seul caractère à une taille maximale de 110 caractères. L'analyse de la distribution des tailles de mots de passe révèle que la majorité d'entre eux ont une longueur inférieure à 20 caractères. Plus précisément, 99% des mots de passe observés ont une longueur comprise entre 4 et 14 caractères. En outre, les mots de passe plus longs sont moins susceptibles d'être sélectionnés en raison de la difficulté relative à les mémoriser. Ainsi, pour l'entraînement du modèle (avoir un apprentissage plus performant) , nous faisons le choix de nous concentrer sur ces 99% de mots de passe.

Nous avons procédé à la catégorisation des mots de passe en fonction de leur longueur, calculant ainsi le nombre de mots de passe correspondant à chaque taille. Cette démarche vise à obtenir une vue d'ensemble des données, facilitant ainsi leur utilisation dans le processus d'apprentissage. En analysant la distribution des longueurs de mots de passe, nous avons pu mieux appréhender la variabilité de ces données, ce qui contribue à une meilleure préparation et compréhension du modèle d'apprentissage.

En plus de la catégorisation par longueur, nous avons également segmenté les mots de passe en plusieurs sous-catégories, notamment ceux composés uniquement de chiffres, ceux ne contenant que des lettres, ceux constitués uniquement de caractères spéciaux, ainsi que ceux intégrant une combinaison de mots et de caractères. Cette approche a pour objectif de générer une diversité de mots de passe au sein de notre ensemble de données d'apprentissage, en prenant en compte différentes configurations possibles. Ainsi, en considérant ces différentes catégories, nous cherchons à enrichir la variété des motifs et structures que notre modèle pourrait apprendre.

Nous avons effectué une analyse visant à identifier les mots de passe redondants au sein de notre corpus. Notablement, nous n'avons constaté aucune occurrence de nombres redondants. Cette absence de redondance numérique dans notre ensemble de données renforce la diversité des mots de passe, ce qui sera un élément favorable pour l'apprentissage de notre modèle.

Suite à notre analyse, nous avons pris la décision d'éliminer

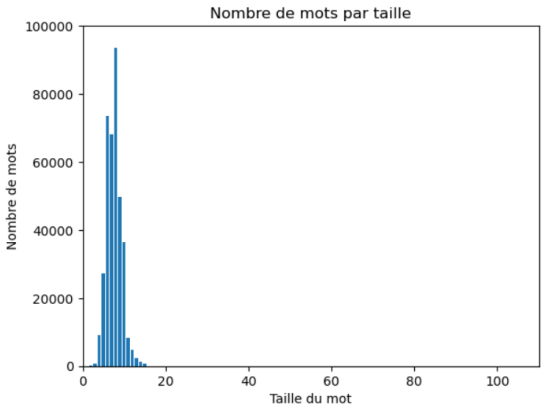


Figure 1: *Longeur du mot de passe.*

Longueur	Nombre de mots
1	60
2	177
3	955
4	9354
5	27372
6	73603
7	68205
8	93440
9	49630

Figure 2: *la catégorisation des mots de passe en fonction de leur longueur, calcule du nombre de mots de passe correspondant 'a chaque taille .*

les mots de passe qui comportent moins de 4 caractères ainsi que ceux dépassant les 20 caractères. Cette sélection a été guidée par la considération du fait que, au-delà de 20 caractères, le nombre de mots de passe disponibles n'est pas suffisamment significatif pour garantir un apprentissage robuste de notre modèle. Cette démarche vise à concentrer notre modèle sur des mots de passe de longueurs plus pertinentes et représentatives.

Suite à la classification des mots de passe en différentes catégories, notre prochaine étape consiste à extraire tous les caractères présents dans notre fichier d'entrée. Cette démarche vise à obtenir une représentation exhaustive de la diversité des caractères utilisés dans les mots de passe, couvrant ainsi les lettres, les chiffres et les caractères spéciaux. L'extraction de cette information constitue une étape cruciale pour la création d'une représentation vectorielle complète, permettant à notre modèle d'apprentissage de saisir les nuances et les variations présentes au sein de l'ensemble de données. Nous avons identifié un total de 94 caractères, incluant des symboles tels que p̄our indiquer le début d'un mot de passe pour représenter un retour à la ligne.

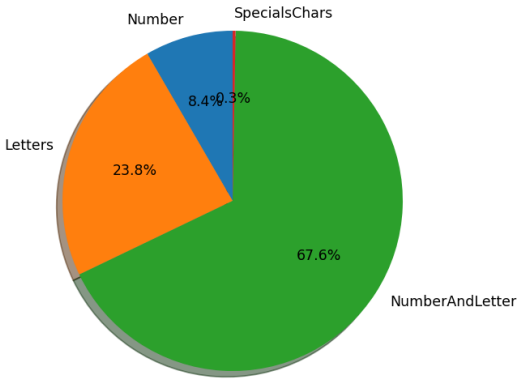


Figure 3: *segmentation des mots de passe en plusieurs sous-catégories: Chiffres, Lettres,caractères spéciaux.*

2.2. Encodage vectoriel

Afin d'adapter nos données au modèle, il est nécessaire de transformer leur représentation vectorielle. En effet, nous disposons d'un grand nombre de mots de passe de tailles variées, tandis que notre modèle peut uniquement traiter des entrées ayant une représentation vectorielle similaire. Pour remédier à cela, nous devons d'abord créer un dictionnaire de caractères auquel nous attribuerons un index (par exemple, [a, 1], [b, 2], etc.), ce qui donne un dictionnaire de taille 92 avec nos données. Ensuite, nous générons notre matrice X train et y train. Chaque mot de passe est ajouté à la première matrice sous forme d'une liste de jetons, où chaque caractère est remplacé par son indice dans le dictionnaire créé précédemment. Par exemple, le mot "Hello" devient la liste suivante : [41, 70, 77, 77, 80]. En ce qui concerne la deuxième matrice y train, elle contient à chaque indice la même liste de jetons décalée vers la droite d'un caractère. De plus, des jetons de début et de fin (e̋t , respectivement) sont ajoutés à nos listes.

Nous avons mis en place deux dictionnaires essentiels dans notre processus, à savoir "charindices" et "indiceschar". Ces dictionnaires revêtent une importance cruciale puisqu'ils permettent d'établir une correspondance bidirectionnelle entre les caractères de nos données et leurs indices respectifs. Cette correspondance joue un rôle fondamental lors de l'encodage des données, facilitant ainsi leur utilisation au sein d'un modèle de réseau de neurones. La création de ces dictionnaires constitue une étape clé pour garantir une représentation cohérente des caractères dans le contexte de l'apprentissage profond, où la transformation de l'information en indices numériques est une nécessité.

Nous avons effectué la conversion des séquences de caractères présentes dans les données initiales en séquences d'indices correspondants. Cette transformation a été réalisée en utilisant le dictionnaire charindices que nous avions préalablement établi. Ainsi, chaque caractère a été substitué par son indice associé, permettant ainsi de représenter les données de manière numérique.

Nous avons ensuite instancié un TensorFlow Ragged Tensor (Xtrain) à partir de la liste d'inputs et (ytrain) à partir de la liste d'outputs. Un tensor ragged est un type particulier de tensor qui a la capacité de représenter des données avec des dimensions variables. Ensuite, pour adapter ces données à un format adapté à l'entraînement d'un modèle de réseau de neurones, nous avons procédé à une conversion en représentation one-hot. Il est important de noter que l'utilisation de la fonction tf.ragged.constant suivie de tf.onehot constitue une approche efficace pour traiter des séquences de longueurs variables tout en effectuant la conversion one-hot. Ces opérations sont essentielles pour préparer nos données d'entraînement, les rendant ainsi compatibles avec les attentes d'un modèle de réseau de neurones qui requiert des entrées one-hot encodées.

L'idée centrale de l'encodage one-hot est de convertir une variable catégorielle en un vecteur binaire, où chaque catégorie est représentée par une valeur binaire distincte (0 ou 1). La longueur du vecteur est égale au nombre total de catégories possibles. Pour une catégorie donnée, toutes les positions du vecteur sont initialisées à zéro, sauf celle correspondant à la catégorie, qui est mise à un.

2.3. Création des modèles

Nous avons alors procédé à la création des modèles, suivi du lancement de leur apprentissage. Les apprentissages que nous avions effectués ont été basé sur les parametres suivants :

- Batch size = 32;
- Number of units = 128 ;
- epochs = 20.

Une fois l'entraînement terminé, les poids et les paramètres appris ont été sauvegardés dans un fichier dédié. Cette étape de stockage garantit la préservation des connaissances acquises par le modèle lors de l'apprentissage, permettant ainsi une utilisation ultérieure sans avoir à recommencer le processus d'entraînement depuis le début. Cette sauvegarde peut se faire dans différents formats, tels que des fichiers HDF5, des fichiers de poids, ou d'autres formats compatibles avec le framework ou la bibliothèque de deep learning utilisée. En résumé, cette phase de création, d'apprentissage et de sauvegarde des modèles assure une mise en œuvre efficace et réutilisable des connaissances acquises par le modèle pour des tâches futures.

2.4. Résultats

2.4.1. RNN

Lors de nos tentatives initiales d'utiliser un réseau de neurones récurrents (RNN) pour générer des mots de passe, nous avons constaté que le modèle n'a réussi à générer qu'un nombre restreint de mots de passe, représentant approximativement 0.1% du total, même après avoir effectué des essais sur plusieurs epochs, variant de 5 à 20. Malgré ces itérations, le modèle a continué à présenter une performance insatisfaisante, se traduisant par un loss relativement élevé. Face à cette limitation, nous avons décidé d'explorer une approche plus avancée en passant à un modèle LSTM (Long Short-Term Memory).

Epochs	Pourcentage dans eval
5	0
10	0.05
15	0.1
20	0.1

2.4.2. LSTM

Dans la phase d'expérimentation avec le modèle LSTM, nous avons réalisé des ajustements significatifs qui ont considérablement amélioré ses performances. En effectuant des essais sur 20 et 30 epochs, tout en maintenant rigoureusement les paramètres, notamment une taille de batch de 32 et un nombre d'unités de 128, nous avons atteint des résultats prometteurs. Le modèle a démontré une remarquable capacité à minimiser le loss, parvenant à une valeur impressionnante d'environ 0.43. Ce faible loss constitue un indicateur clair de l'efficacité du modèle à assimiler les schémas complexes des données d'entraînement et à générer des mots de passe de manière précise.

Plus intrigant encore, lors de la comparaison du fichier généré avec le fichier RockYou, nous avons constaté une ressemblance substantielle atteignant 21%, soulignant la capacité du modèle à reproduire des motifs présents dans des ensembles de données réels. De manière similaire, une correspondance de 20% a été observée avec le fichier Ashley Madison, renforçant davantage la validité et la polyvalence du modèle dans la création de mots de passe réalistes.

Ces résultats encourageants laissent entrevoir la possibilité d'améliorations continues en augmentant davantage le nombre d'époques d'entraînement. En prolongeant cette exploration, il est probable d'obtenir des résultats encore meilleurs, renforçant ainsi la capacité du modèle LSTM à générer des mots de passe diversifiés et conformes aux modèles observés dans nos données.

2.4.3. LSTM-Bidirectionnel

L'expérience avec le modèle LSTM bidirectionnel a conduit à la génération de fichiers de mots de passe ne contenant que des espaces. Cette observation suggère deux possibilités : soit la configuration du modèle LSTM n'était pas optimale, soit des critères supplémentaires, tels qu'une quantité substantielle de données, sont nécessaires pour garantir son efficacité.

Dans le premier scénario, il pourrait être nécessaire de revoir la configuration du modèle LSTM bidirectionnel, en ajustant des paramètres tels que le nombre d'unités, la profondeur du réseau, ou les hyperparamètres liés à l'apprentissage. Un réexamen minutieux de ces paramètres pourrait permettre d'optimiser le modèle et d'améliorer sa capacité à générer des mots de passe diversifiés.

Dans le deuxième scénario, l'efficacité du modèle LSTM bidirectionnel pourrait dépendre d'une quantité de données plus importante pour apprendre les motifs et la complexité des mots de passe. Il serait alors envisageable d'augmenter la taille du jeu de données d'entraînement afin d'améliorer la représentativité des exemples fournis au modèle.

En somme, cette situation souligne la nécessité d'explorer différentes configurations et conditions d'entraînement pour garantir le succès du modèle LSTM bidirectionnel dans la génération de mots de passe diversifiés et significatifs.

3. Conclusions

L'exploitation de réseaux de neurones récurrents se présente comme une solution prometteuse pour générer un dictionnaire de grande envergure. Des résultats satisfaisants ont été obtenus en utilisant un modèle relativement simple avec une quantité de données limitée. En investissant des ressources supplémentaires, il serait envisageable de réentraîner ce modèle avec une quantité plus importante de données, telle que la liste de mots Rockyou, ou en ajustant la taille des lots de données, ce qui pourrait potentiellement améliorer les performances. Néanmoins, il convient de souligner que les mots de passe tendent généralement à suivre un schéma de construction relativement simplifié pour un système informatique, soulevant ainsi des interrogations quant à la sécurité sous-jacente des mots de passe. La plupart des mots de passe observés se composent de combinaisons de mots et de chiffres. Par conséquent, il est fortement recommandé de complexifier autant que possible les mots de passe, en incorporant des caractères spéciaux, des majuscules, des minuscules, des chiffres, et en évitant les mots de passe trop courts, afin de mieux se prémunir contre les attaques par dictionnaire.

Enfin, bien que le mot de passe demeure une solution relativement efficace et simple pour sécuriser ses données, d'autres dispositifs s'avèrent désormais impératifs pour renforcer la sécurité des données. Parmi ces dispositifs, on peut citer l'authentification à deux facteurs et l'utilisation de clés de chiffrement. Ces mesures complémentaires contribuent à ériger une défense plus robuste contre les menaces émergentes, assurant ainsi une protection renforcée dans le paysage numérique en constante mutation.

4. References

[1] Tech. Rep. [Online]. Available: https://www.ibm.com/fr-fr/topics/recurrent-neural-networks

[2] Tech. Rep. [Online]. Available: https://datascientest.com/long-short-term-memory-tout-savoir

[3] Tech. Rep. [Online]. Available: https://ichi.pro/fr/prediction-de-sequence-avec-le-modele-lstm-bidirectionnel-17354776589143