INTRODUCTION Générale

Dans le monde de l’internet, on trouve différents types de données sur des produits différents : vidéos, photos, messages, annonces…etc. Ces données à leurs tours donnent des informations sur les prix, nombre d’achat, nombre de vue, nombre de lien et l’évolution de marché de ce produit en général. Avec l’essor des réseaux Internet et Wi-Fi, des smartphones, des objets connectés et des réseaux sociaux, de plus en plus de données de formes variées sont générées. En parallèle, le développement d’outils de stockage et d’analyse – notamment les algorithmes de prédictions – ainsi que de nouveaux outils de visualisation permettent la valorisation de ces données non structurées, variées et en très grande quantité : c’est une bombe technologique que l’on nomme l’intelligence artificielle (AI).

Les applications du AI sont nombreuses « pour optimiser la logistique, le marketing via la personnalisation, le développement et le suivi de produits plus proches des besoins des clients, comme le suivi de marché des Smartphone, AI est donc source d’innovations dans tous les secteurs d’activité, mais aussi de performance et de productivité. L’objectif de ce mémoire est de présenter les démarches des processus de prédiction et d’analyse de données dans le service de marketing, en montrant que les algorithmes des AI jouent le rôle de faciliter la recherche aux clients, car les informations deviennent rapides et dans un seul clic d’un part, et d’autre part les entreprises peuvent s’avoir la tendance de vente de leurs produits grâce aux informations requise « le nombre de vue, la note donnée par ces clients » pour arriver à une bonne prédiction par la suite.

L’évolution du Système informatique amène les entreprises à traiter de plus en plus de données issues de sources toujours plus variées. Ainsi les entreprises cherchent toujours la bonne décision pour régler les problèmes de production et de management dans les conditions les plus optimales.

Chapitre1 : Pôles de l’intelligence artificielle.

1. **Introduction :**

Le cerveau humain, oui vous l’avez bien lu. L’intuition et la motivation derrière l’IA viennent du cerveau humain. C’est l’outil le plus puissant donné à l’humanité. L’homme est l’espèce la plus intelligente de la planète Terre. Et la sauce secrète derrière ce cerveau puissant est Néocortex. Néocortex vous donne le pouvoir de penser, d’agir, de fonctionner, de vous souvenir, etc. , c’est un mythe commun que les baleines sont plus intelligentes que les humains, ce qui n’est pas vrai puisque les baleines ont de 4 à 5 couches dans leur néocortex qui sont plus petites que les humains qui ont six couches dans leur néocortex et sont plus importantes que les baleines.  
  
Votre cerveau est câblé de telle sorte que la mémoire néocorticale est responsable de stocker non pas des images, mais des modèles (à la fois des modèles temporels et spatiaux). Lorsque ces modèles ou ces souvenirs sont rappelés, où deux domaines s’introduisent, Machine Learning et Deep Learning.

1. **Introduction au machine Learning**
   1. **Définition**

Machine Learning est une technique de programmation informatique qui utilise des probabilités statistiques pour donner aux ordinateurs la capacité d’apprendre par eux-mêmes sans programmation explicite. Pour son objectif de base, le machine learning « apprend à apprendre » aux ordinateurs – et par la suite, à agir et réagir – comme le font les humains, en améliorant leur mode d’apprentissage et leurs connaissances de façon autonome sur la durée. L’objectif ultime serait que les ordinateurs agissent et réagissent sans être explicitement programmés pour ces actions et réactions. Le Machine Learning utilise des programmes de développement qui s’ajustent chaque fois qu’ils sont exposés à différents types de données en entrée.

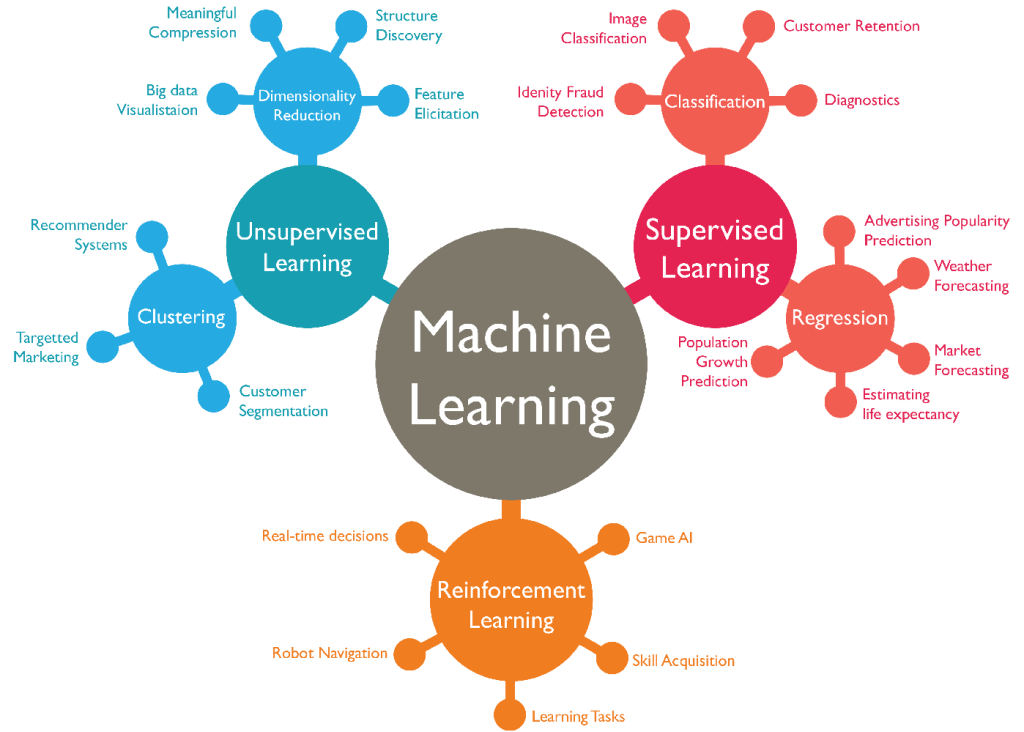
La clé du [machine Learning](https://www.talend.com/fr/blog/2017/06/26/what-everyone-should-know-about-machine-learning/) réside dans l’entrée de volumes considérables de données dans l’ordinateur-étudiant. Pour apprendre, la machine a besoin de consommer des [big data](https://www.talend.com/fr/resources/future-big-data/).

## Les catégories de machine Learning

L'apprentissage automatique n'est pas une nouvelle technologie. Le psychologue américain Frank Rosenblatt (Frank Rosenblatt) a inventé le premier réseau neuronal artificiel en 1958, appelé le «perceptron».

Au départ, Perceptron doit être une machine, pas un algorithme. En 1960, il a été utilisé dans le développement de la machine d'imagerie "Mark 1 Perceptron". Mark 1 Perceptron est le premier ordinateur qui utilise des réseaux de neurones artificiels (ANN) pour simuler la pensée humaine et l'apprentissage par essais et erreurs.

En raison de la prolifération des bibliothèques et des cadres de code open source et de l'augmentation de plusieurs milliards de dollars de la puissance de traitement informatique entre 1956 et 2018, l'apprentissage automatique est de plus en plus utilisé. Aujourd'hui, l'apprentissage automatique est partout: de la négociation d'actions à la protection contre les logiciels malveillants en passant par la personnalisation marketing. Peu importe sa simplicité ou sa complexité, l'apprentissage automatique peut être divisé en trois catégories:



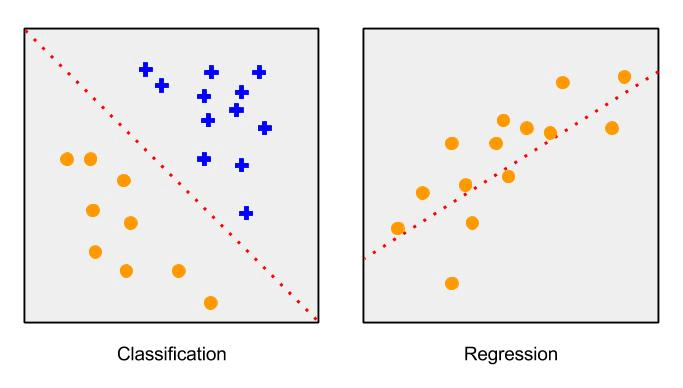
### **1. Machine Learning avec supervision**

L'apprentissage automatique supervisé est une technique basique mais rigoureuse. L'opérateur fournit à l'ordinateur des exemples des entrées et des sorties requises, puis l'ordinateur trouve des solutions basées sur ces entrées pour obtenir ces sorties. Le but est de faire apprendre à l'ordinateur les règles générales de mappage d'entrée et de sortie.

L'apprentissage automatique supervisé peut être utilisé pour faire des prédictions sur des données non disponibles ou des données futures (on parle de «modélisation prédictive»). L'algorithme tente de développer une fonction qui peut prédire avec précision la sortie des variables d'entrée - par exemple, en prédisant la valeur des attributs d'entrée (sortie), tels que le nombre de pièces, l'année de construction, la superficie du terrain, l'emplacement, etc.

L'apprentissage automatique sous supervision peut être divisé en deux types :

* **Classification** – La variable de sortie est une catégorie.
* **Régression** – La variable de sortie est une valeur spécifique.



Les principaux algorithmes d'apprentissage automatique supervisé sont: forêt aléatoire, arbre de décision, méthode du k-plus proche voisin (k-NN), régression linéaire, classification de base naïve, machine à vecteurs de support (SVM), régression logistique et gradient d'accélération.

### **3. Machine Learning par renforcement**

Dans l'apprentissage automatique gratuit, l'algorithme est laissé à son propre appareil pour déterminer la structure de l'entrée (l'algorithme n'a pas d'étiquette). Cette méthode peut être un but en soi (découvrir la structure enfouie dans les données), ou elle peut être un moyen d'atteindre un but. Cette méthode est également appelée «apprentissage fonctionnel».

L'algorithme de reconnaissance faciale prédictive de Facebook, qui peut reconnaître les personnes sur les photos publiées par les utilisateurs, est un exemple d'apprentissage automatique gratuit.

Il existe deux types d'apprentissage automatique gratuit:

* Clustering - L'objectif est de trouver des clusters dans les données. Vous pouvez également diviser la base de données en différents groupes, puis les regrouper en fonction du comportement des clients.

• Association - L'objectif est de déterminer les règles qui définiront de grands ensembles de données.

Les principaux algorithmes de l'apprentissage automatique gratuit sont: K ressources, regroupement / regroupement hiérarchique et réduction de la dimensionnalité.

C -Domaine d’application:

Les voitures autonomes sont un bon exemple d'apprentissage automatique. Une voiture indépendante est équipée de plusieurs caméras, de plusieurs radars et d'un capteur lidar. Ces différents appareils offrent les fonctions suivantes :

• Utilisez le GPS pour déterminer en continu et avec précision l'emplacement de la voiture.

• Analysez la section de la route devant la voiture.

• Détecte les objets mobiles ou stationnaires à l'arrière ou sur le côté de la voiture.

Ces informations sont traitées en continu par un ordinateur central également installé dans la voiture. L'ordinateur collecte et analyse en permanence de grandes quantités de données et classe les données et le réseau neuronal dans le cerveau humain. Pour diriger la voiture, l'ordinateur prend des milliers de décisions par seconde en fonction de probabilités et d'observations mathématiques : comment tourner la roue, quand freiner, accélérer, changer de vitesse, etc.

1. **La conformité de l’apprentissage profond** **aux complexités**

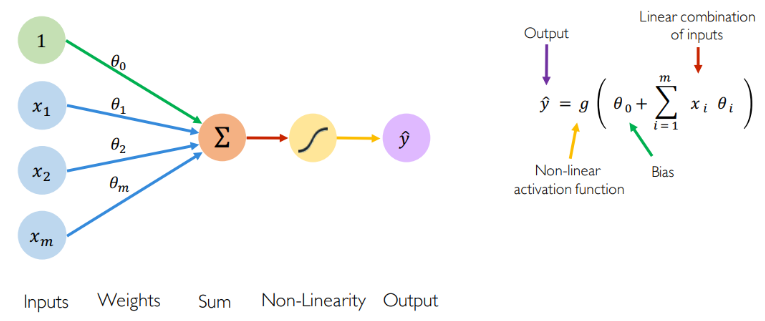
Les modèles traditionnels d'apprentissage automatique ont été très puissants dans le traitement des données structurées et sont largement utilisés par les entreprises pour les scores de crédit, les prévisions de revenus, le positionnement des consommateurs cibles, etc.

Le succès de ces modèles dépend beaucoup de la performance de la phase d’ingénierie : plus nous sommes proches de l'entreprise qui extrait des connaissances pertinentes à partir de données structurées, plus le modèle est solide.

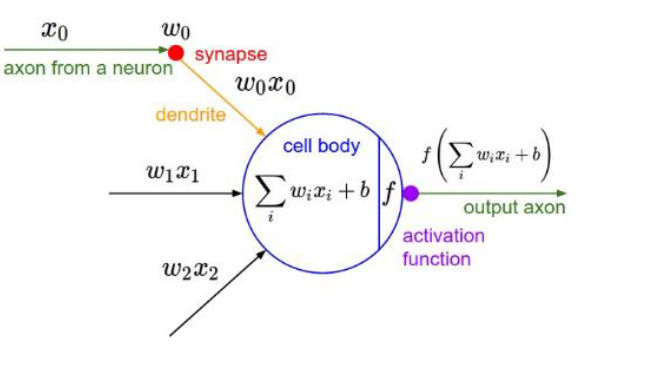
Lorsqu'il s'agit de données non structurées (images, texte, voix, vidéo), les fonctions artisanales prennent du temps, sont délicates et ne peuvent pas être étendues dans la pratique. C'est pourquoi les réseaux de neurones sont de plus en plus populaires, car ils peuvent découvrir automatiquement les représentations nécessaires pour détecter ou classer les données brutes. Il remplace l'ingénierie des fonctions manuelles

1. **La base d’architecture – Perceptron.**

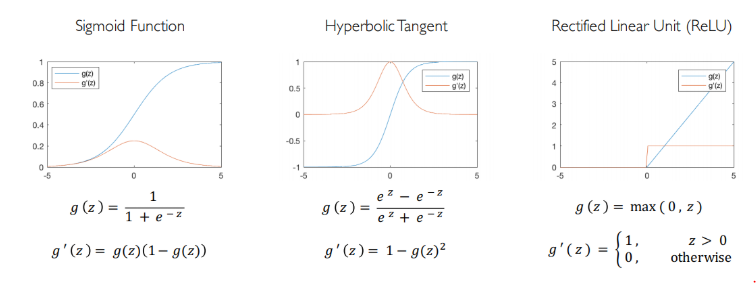
L’élément fondamental de l’apprentissage profond (Deep Learning) est le Perceptron qui est un neurone unique dans un réseau neuronal.  
  
Avec un ensemble fini d’entrées m (par ex. mots m ou pixels m), on multiplie chaque entrée par un poids (thêta 1 à thêta m) puis on additionne la combinaison pondérée d’entrées, on ajoute un biais et on les passe finalement par une fonction d’activation non linéaire. Cela produit la sortie suivante :



* Comme la structure montre, les entrées représentées par un vecteur de données (x.…xm) seront passées par tant d’opération pour produire une sortie qu’on appelle une décision.



* Le biais thêta 0 permet d’ajouter une autre dimension à l'espace d'entrée. Par conséquent, si l'entrée est entièrement à zéro, la fonction de déclenchement fournira toujours une sortie. Il fait partie de la sortie et n'a rien à voir avec l'entrée.
* Le but de la fonction d'activation est d'introduire la non-linéarité dans le réseau. En fait, les fonctions d'activation linéaires produisent des décisions linéaires, quelle que soit la distribution des entrées. La non-linéarité nous donne un meilleur accès aux fonctions complexes indépendantes. Voici quelques exemples de fonctions d’activation courantes :



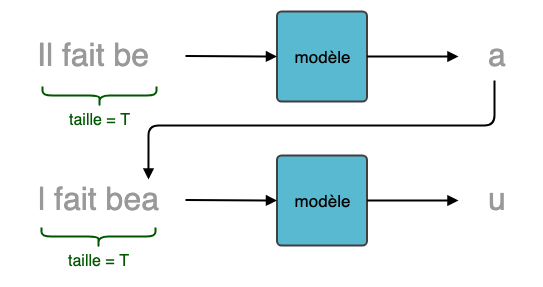
Les réseaux neuronaux profonds ne sont rien de plus qu’un empilement de multiples perceptrons (couches cachées) pour produire une sortie.

Dans notre projet on va traiter algorithme d’apprentissage profond s’appellent « Recurrent neurale network (RNN) »

1. Récurrent neural network.

Les réseaux de neurones constituent aujourd’hui l’état de l’art pour diverses tâches d’apprentissage automatique. Ils sont très largement utilisés par exemple dans les domaines de la vision par ordinateur (classification d’images, détection d’objets, segmentation…) et du traitement automatique du langage (traduction automatique, reconnaissance vocale, modèles de langage…). Cette famille de modèles, particulièrement adaptée aux données séquentielles, nous a permis de générer automatiquement, caractère par caractère, du texte compréhensible à partir d’une séquence initiale. Nous vous avions alors promis de vous expliquer plus en détail le fonctionnement des modèles en question. Le but de ce nouvel article est de décortiquer deux variantes de RNN pour en expliquer le fonctionnement interne et démythifier leur aspect « boîte noire ». Il nous servira de fil rouge tout au long de cette explication.

On va procéder par un problème simple de génération de texte pour mieux simplifier l’algorithme et arrêter sur les points d’inflexion de cette méthode. Afin de générer du texte, nous partons d’une séquence de caractères de taille fixe pour prédire le caractère suivant.

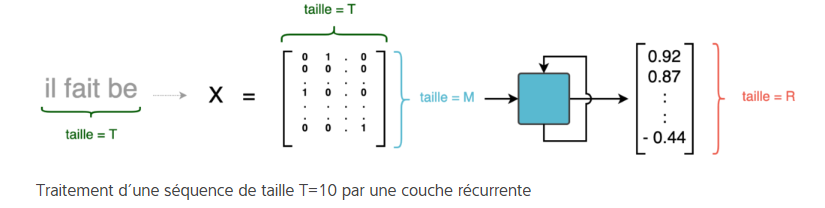


Le pré-traitement détaillé dans l’article précédent permet de représenter numériquement le texte brut, le transformant en une matrice dont les dimensions sont **nb échantillons**\***taille séquence**\***nb variables**  où :

* **Nb échantillons**= nombre de séquences d’entraînement (taille du dataset)
* **Taille séquence**= nombre (fixe) de caractères par séquence (dimension temporelle). Dans l’article précédent, nous avions choisi d’entraîner le modèle sur des séquences de taille . Le choix de est empirique et il y a un compromis à trouver : si T est petit, le modèle aura la « mémoire courte ». Si T est grand, l’entraînement sera très long et pas toujours efficace. Pour une meilleure lisibilité, nous choisirons plutôt =10 dans les illustrations suivantes.
* **Nb variables**= taille de la représentation vectorielle de chaque caractère. Nous avons choisi de limiter notre vocabulaire aux 26 lettres de l’alphabet + 4 caractères spéciaux. Chaque caractère est donc représenté par un vecteur de taille grâce au [one-hot-encoding](https://fr.wikipedia.org/wiki/Encodage_one-hot).

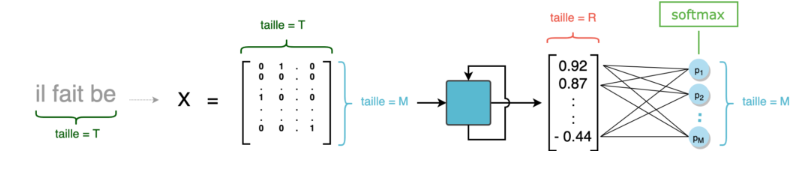
En résumé, le modèle prend en entrée un tableau de séquences, chacune de longueurcaractères**,**où chaque caractère est un vecteur numérique de taille.

Une couche de type RNN prend en entrée une séquence de ce tableau (donc une matrice de taille ) et retourne en sortie un vecteur de taille qui est une représentation compressée de la totalité de la séquence de caractères. Encore une fois, **R** est un hyperparamètre à choisir judicieusement. Pour l’exemple, on peut choisir . C’est en quelque sorte l’équivalent du nombre de neurones dans une couche dense (ou « [fully-connected](http://cs231n.github.io/neural-networks-1/#nn)« ).



La génération du symbole suivant implique la sélection du symbole le plus significatif parmi symboles possibles. Par conséquent, notre modèle devrait produire une sortie de taille . Ensuite, étant donné le symbole T ci-dessus, ce vecteur de sortie est associé à la probabilité de chaque symbole, qui est le symbole suivant dans la séquence.

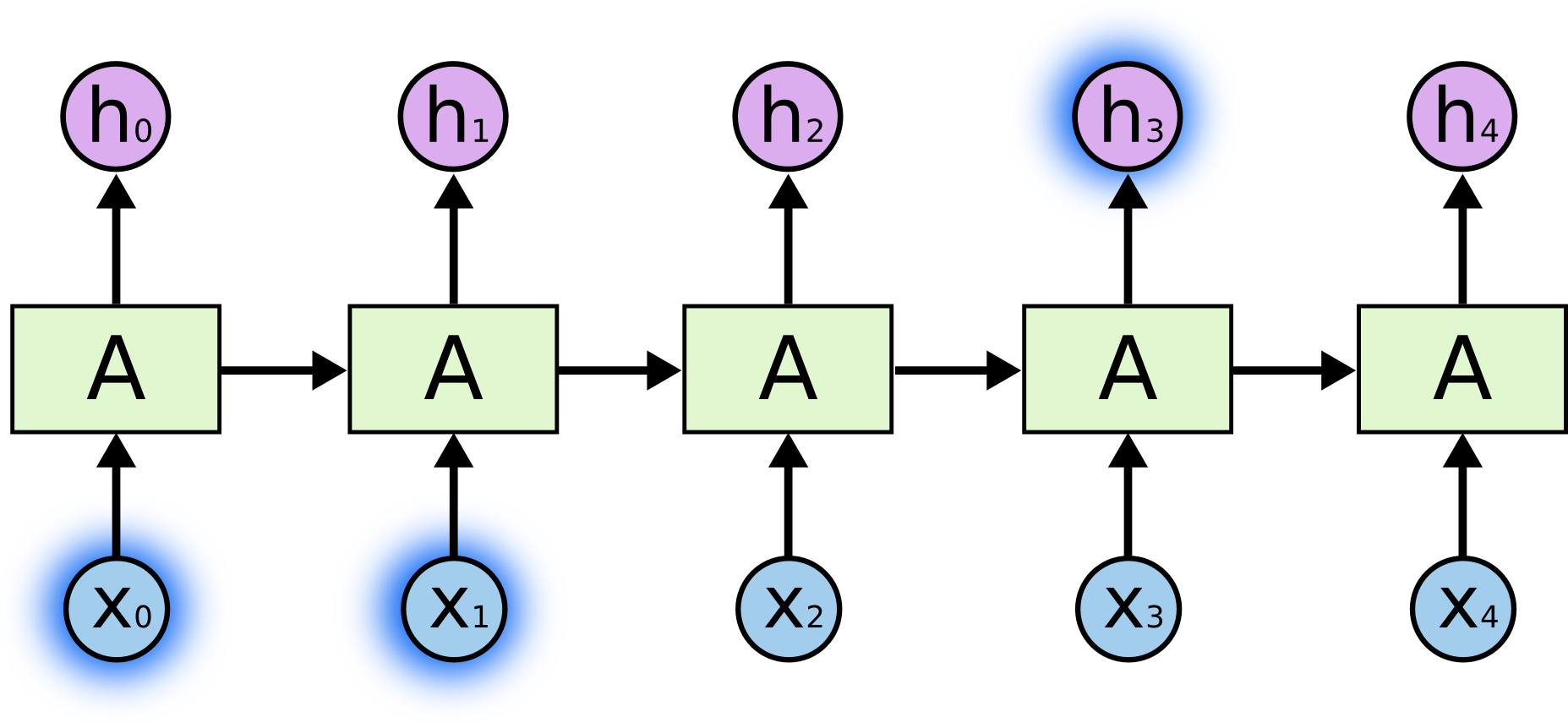
Pour obtenir cette distribution de probabilité, nous empilons une couche de neurones "classiques" (également appelée couche dense) de taille après la couche RNN et utilisons la fonction d'activation softmax. Prenons l'exemple suivant : la séquence d'entrée est « terminée », la probabilité de sortie la plus élevée peut être la lettre « a » pour générer l'expression « ensoleillé »



-Prédiction de la distribution de probabilité du caractère suivant à partir d’une séquence initiale

Détaillons maintenant ce qui se passe à l’intérieur-même de la couche RNN. Une couche RNN est une succession de T cellules. Chaque cellule a deux entrées :

* L’élément de la séquence lui correspondant (en version one-hot) : la --ème cellule est associée au --ème caractère de la séquence
* le vecteur en sortie de la cellule précédente. La première cellule, qui n’a pas d’antécédent, prend alors un vecteur initialisé aléatoirement



Une couche , prenant en entrée des séquences de 10 caractères

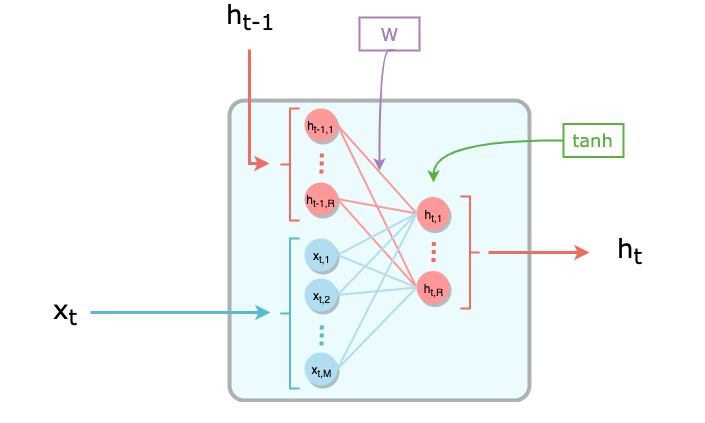
Dans une couche , on parcourt donc successivement les entrées  à . À l’instant , la --ème cellule combine l’entrée courante  avec la prédiction au pas précédent pour calculer une sortie de taille .

Le dernier vecteur calculé   (qui est de taille R) est la sortie finale de la couche RNN. Une couche RNN définit donc une relation de récurrence de la forme :

)

### Comportement d’une cellule

La t-ème cellule n’est rien d’autre qu’une couche dense de taille **R**, dont l’entrée est la concaténation de  (de taille ) et  (de taille **R**). La fonction d’activation classique utilisée pour les cellules du RNN est la tangente hyperbolique . Plusieurs propriétés intéressantes expliquent ce choix, notamment une [convergence plus rapide](https://www.quora.com/In-an-LSTM-unit-what-is-the-reason-behind-the-use-of-a-tanh-activation) que la sigmoïde.



Zoom sur une cellule RNN

La formule devient donc :

)

Où  et  sont les poids appris par le modèle :

* est une matrice de taille
* un vecteur de taille

Il est à noter que les poids sont partagés entre toutes les cellules d’une couche RNN. Autrement dit, c’est exactement la même fonction (avec les mêmes poids) qui est appliquée à chaque pas de temps . Cela permet à la fois de modéliser toute la séquence de façon homogène et d’éviter de multiplier le nombre de poids à apprendre. En effet, si on apprenait une matrice de poids différente pour chacune des  cellules, il faudrait alors apprendre  matrices (,…, ) au lieu d’une seule.

Selon les implémentations, il est possible que la matrice W soit séparée en deux parties :

* (Correspondant aux connexions entre  et )
* (Correspondant aux connexions entre  et ).

### Empiler des couches récurrentes

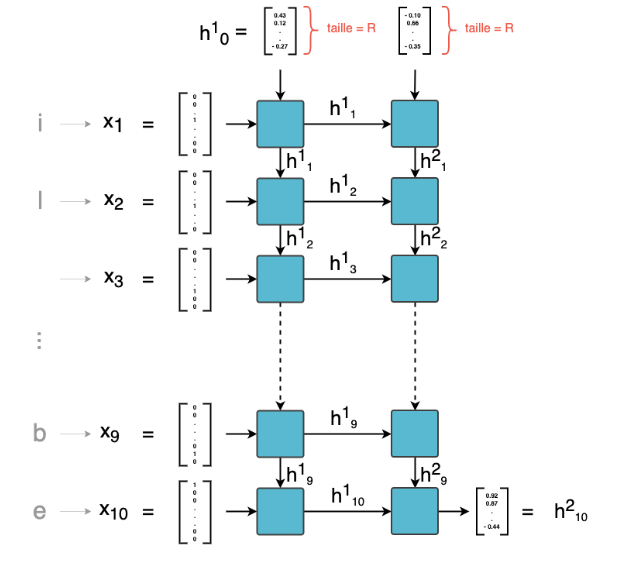
Généralement, une seule couche RNN n'est pas suffisante pour capturer toutes les informations contenues dans la séquence. Par conséquent, les couches de type RNN peuvent être empilées pour avoir un réseau plus profond, qui fonctionne généralement en couches denses ou alambiquées. De cette manière, des informations plus complexes peuvent être extraites de l'entrée pour mieux modéliser nos données.

Dans la plupart des cas, 2 à 5 répétitions sont suffisantes, sinon la fusion devient difficile. Cette profondeur "peu profonde" est en fait trompeuse, car la structure de la couche RNN est un peu profonde, car la couche est étirée longtemps à plusieurs reprises.

Essayons de construire un réseau avec 2 couches répétitives. La première couche RNN renvoie un seul vecteur de taille . La deuxième couche se répète et nécessite une séquence de vecteurs d'entrée au lieu d'un seul vecteur.

Alors, comment remplir cette condition ?

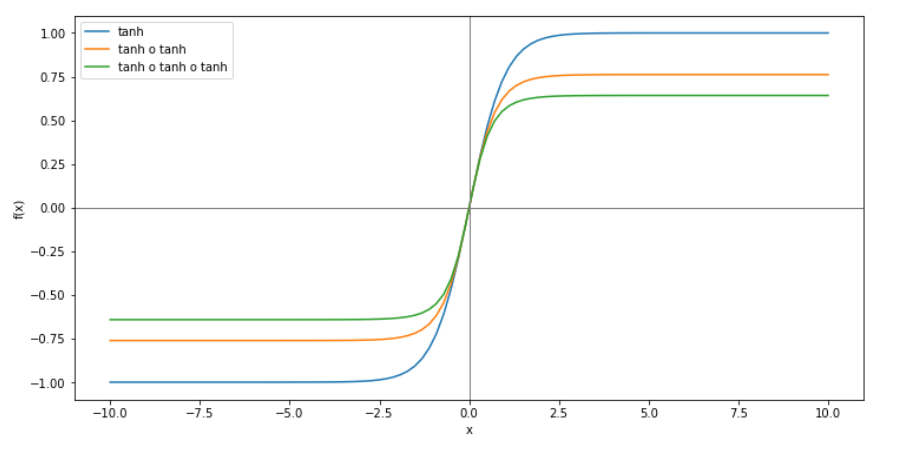
La solution comprend la séquence de sortie intermédiaire , , ..., de la première couche comme entrée de la deuxième couche, pas seulement la sortie finale. Le schéma de notre modèle est alors le suivant :



### Limites des RNN simples

Bien qu'ils soient conçus spécifiquement pour le traitement de séquences, les RNN simples présentent encore certaines limitations.

La couche RNN du modèle à court terme est une séquence de cellules, et chaque cellule reçoit la représentation du symbole actuel et la sortie de la cellule précédente en entrée (voir la figure de la section 2.A). Ces données d'entrée sont converties même par la fonction clé hyperbolique. La tangente hyperbolique a tendance à écraser les valeurs qu’elle prend en entrée. En effet, elle est définie sur l’espace des réels mais ses valeurs de sortie sont dans l’intervalle ]-1, 1[. Ainsi, après un premier passage par la tangente hyperbolique, on obtient une valeur entre -1 et 1. Ensuite, comme (1) = 0.76 et (-1) = -0.76, un deuxième passage par la fonction résulte en un intervalle de valeurs encore plus réduit, et ainsi de suite.



et ses compositions

Les informations de sont passées à travers la clé hyperboliquefois, tandis que les informations de ne subissent cette conversion qu'une seule fois. La matrice de poids peut compenser cet effet d'écrasement par des poids plus lourds associés à . Cependant, puisque cette matrice est partagée par toutes les unités de la couche, toutes les sorties moyennes sont pondérées avec le même poids. Par conséquent, intuitivement, l'influence de sur la prédiction sera inférieure à . Par conséquent, même si est essentiel pour la tâche de prédiction, une transmission continue via la clé hyperbolique réduira mécaniquement son impact.

Prenons un exemple pour comprendre que cela peut poser des problèmes. Supposons que notre chaîne d'entrée soit "notifiée". Pour prédire la prochaine lettre du verbe correspondant, vous devez connaître le sujet. Cependant, ces informations se trouvent au début de la séquence. Le modèle peut avoir du mal à distinguer cette séquence d'une séquence qui ressemble à "vous avertissez". C'est pourquoi vous pouvez attribuer une probabilité élevée à "s" et faire des erreurs. Ce problème devient très inconfortable lors de l'utilisation de chaînes plus longues, ce qui rend difficile pour les RNN de base de générer un texte cohérent au fil du temps. Modèles difficiles à former

L’une des principales difficultés pour entraîner des réseaux de type RNN est le problème du [vanishing gradient](http://proceedings.mlr.press/v28/pascanu13.pdf). Pour comprendre ce problème, faisons tout d’abord quelques rappels au sujet de l’entraînement d’un réseau de neurones. Un réseau de neurones est un enchaînement de fonctions simples, prenant en entrée les données de notre problème . Dans notre exemple de génération de texte, il s’agit des  séquences de taille .

Cet enchaînement de fonctions a comme paramètres des poids . L’apprentissage consiste à ajuster les poids  afin de minimiser une erreur, donnée par une fonction de coût . Cette erreur mesure l’écart entre les labels réels  (la « vérité », les caractères que le modèle est censé prédire) et les labels prédits  (les caractères prédits par le modèle). Elle est mesurée sur une partie de nos données (Xtest, ytest).

L’entraînement a lieu comme il suit :

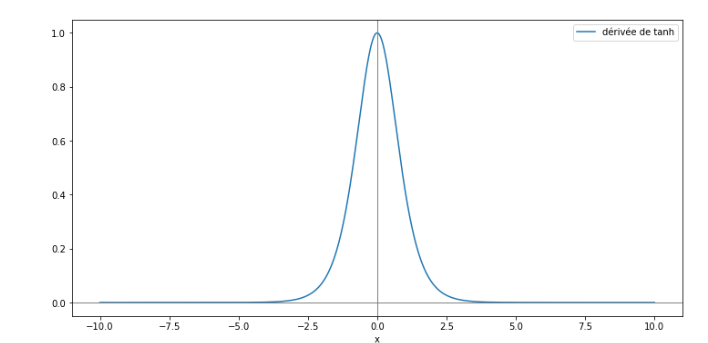
1. Les poids  sont initialisés aléatoirement
2. On fait passer  par notre modèle pour obtenir une prédiction train. On calcule ensuite la valeur de la fonction de coût  (,  train)
3. On calcule le [gradient](https://fr.wikipedia.org/wiki/Gradient) de cette fonction L par rapport aux paramètres
4. On cherche à minimiser notre fonction de coût en mettant à jour les paramètres  Pour cela, on effectue une [descente de gradient](https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e) grâce au gradient calculé à l’étape précédente :

Un réseau de neurones est une suite de fonctions (couches) appliquées successivement. Calculer son gradient revient donc à [dériver des fonctions composées](https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_d%C3%A9rivation_des_fonctions_compos%C3%A9es). Schématiquement, cela revient à calculer la dérivée d’une fonction composée comme une multiplication de dérivées partielles des fonctions qui la composent.

Ainsi, si notre modèle est composé de trois couches et est représenté par la fonction , son gradient par rapport à sera

Dans le cas d’une couche RNN, on a :

Le gradient de  par rapport à  est donc proportionnel à la multiplication de dérivées de la fonction tangente hyperbolique. Or, la dérivée de cette fonction se situe dans l’intervalle [0, 1] et prend presque sûrement des valeurs dans [0, 1[.



La courbe de la dérivée de la

Or, plus on multiplie des valeurs entre 0 et 1 entre elles, plus le résultat se rapproche de 0.  prend donc des valeurs très petites lorsque les séquences sont longues. La mise à jour des paramètres devient donc très lente et l’entraînement du modèle est mis à mal.

.

Chapitre2 :

* 1. LSTM : RNN améliorer.

Plusieurs variantes de la norme RNN semblent avoir résolu les problèmes ci-dessus. Nous décrirons ici LSTM pour la mémoire à long terme. Ce type de RNN est largement utilisé dans le traitement du langage naturel.

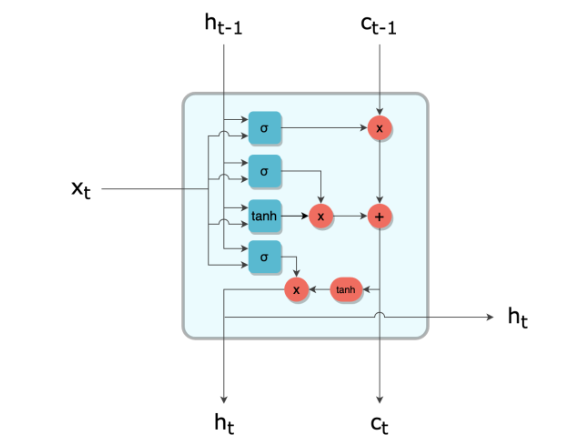
L'idée derrière le choix d'une architecture de réseau neuronal est de diviser le signal entre les signaux importants à court terme et l'état de l'unité qui sera interprété à long terme à travers des états cachés (similaire à la sortie d'une unité RNN). Allons. Par conséquent, la fonction générale du LSTM peut être résumée en 3 étapes :

1. Détecter les informations pertinentes dans le passé et extraire les informations pertinentes de l'état de la cellule à travers la porte d'oubli.

2. À partir de l'entrée actuelle, sélectionnez l'entrée qui vous est étroitement liée dans la porte principale depuis longtemps. Ceux-ci seront ajoutés à l'état du lecteur en tant que mémoire à long terme.

3. Sélectionnez les informations importantes à court terme du nouvel état du disque pour générer l'état masqué suivant via la sortie.

Regardons de plus près. Suivez la même convention que le schéma simplifié de l'unité RNN, on peut représenter une cellule LSTM de la façon suivante :



Schema de RNN améliorée

Comme le RNN, le LSTM définit donc une relation de récurrence, mais utilise une variable supplémentaire qui est le **cell state c**:

L’information transite d’une cellule à la suivante par deux canaux,  et . À l’instant , ces deux canaux se mettent à jour par l’interaction entre leurs valeurs précédentes et ainsi que l’élément courant de la séquence .

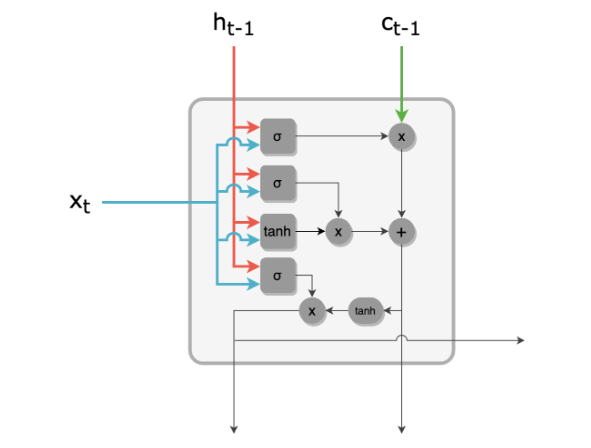
Dans le paragraphe suivant, nous allons essayer de comprendre pas à pas les opérations réalisées par une cellule LSTM. Les schémas suivants sont inspirés de [l’excellent article de C.](http://colah.github.io/posts/2015-08-Understanding-LSTMs/)Olah.

### La cellule LSTM pas à pas

1 – La ème cellule LSTM prend 3 vecteurs en entrée :

* L’élément courant de la séquence  : représentation vectorielle du ème caractère (vecteur de taille )
* Le hidden state de la cellule précédente(vecteur de taille )
* Le cell state de la cellule précédente (vecteur de taille ).

C’est ce dernier vecteur que nous allons suivre particulièrement. Il s’agit d’une route privilégiée de transmission d’information sur la séquence. Pour éviter le problème du vanishing gradient, le cell state est en effet mis à jour de façon **additive** à chaque étape, sans passer par une activation.



Entrées d’une cellule LSTM: input, hidden state, cell state

|  |  |
| --- | --- |
| FIG-1 | FIG-2 |
| FIG-3 | FIG-4 |
| FIG-5 | FIG-6 |
| FIG-7 | |
| FIG-8 | |

1 – La **forget gate** est une couche dense de taille  avec une activation **sigmoïde**. À partir de  et , cette forget gate produit un vecteur  de taille , dont les valeurs sont comprises entre 0 et 1.

2 – La forget gate agit comme un filtre pour « oublier » certaines informations du **cell state**. En effet, on effectue une [multiplication terme à terme](https://fr.wikipedia.org/wiki/Produit_matriciel_de_Hadamard) entre  et , ce qui a tendance à annuler les composantes de  dont les homologues côté  sont proches de 0. On obtient alors un cell state *filtré*, toujours de taille .

3 – L’**input gate** produit un filtre  de valeurs comprises entre 0 et 1 et de taille , à partir de  et , de façon similaire à la forget gate.

4 – En parallèle, un vecteur  de taille est créé par une couche .  est le vecteur candidat pour **mettre à jour** le cell state.

5 – Le candidat  est *filtré* par l’input gate via une multiplication terme à terme. On obtient le vecteur de mise à jour du cell state.

6 – Le cell state filtré (obtenu à l’étape 3) est mis à jour grâce au vecteur candidat filtré (obtenu à l’étape 6). La mise à jour est une simple addition terme à terme de ces deux vecteurs. On obtient alors le **nouveau cell state** .

7 – De façon analogue à  et , l’output gate produit un filtre  de valeurs entre 0 et 1, et de taille .

8 – Les valeurs du **nouveau cell state**  sont ramenées à l’intervalle ]-1, 1[ par une activation . Un filtrage par l’output gate  est ensuite effectué pour enfin obtenir la sortie .

### **Quelques astuces pour une meilleure performance**

Des recherches sur les architectures RNN pour les tâches de traduction ont démontré que le sens de lecture classique des séquences n’est pas optimal. En effet, ils ont montré empiriquement que lire les séquences à l’envers améliore significativement la performance du modèle dans ce genre de problèmes. La combinaison des deux sens de lecture (bidirectionnel RNN) donne [des résultats encore meilleurs](https://arxiv.org/pdf/1703.03906.pdf). Ce dernier résultat est prévisible, vu qu’une représentation bidirectionnelle permet au modèle d’accorder autant d’importance aux premiers caractères de la séquence qu’aux derniers. La version unidirectionnelle, quant à elle, « dilue » mécaniquement les premiers caractères, même si ce phénomène est moins prononcé dans un LSTM que dans un RNN simple.

Chapitre3 : La réalisation du projet.

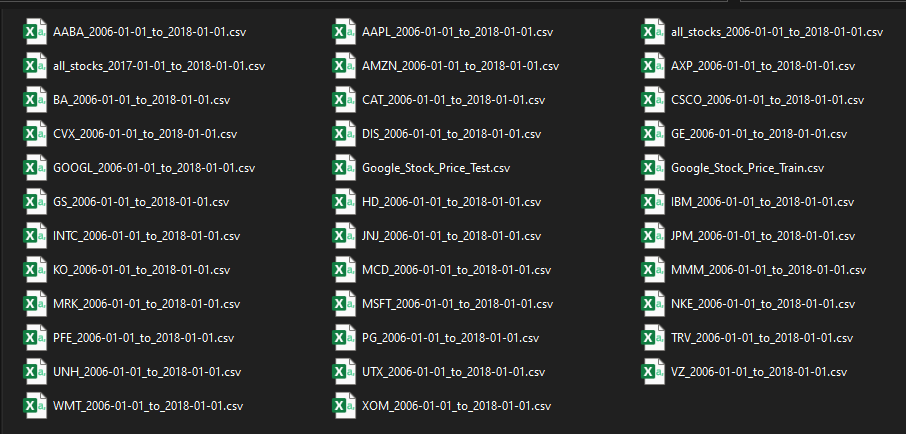
* 1. Présentation de projet.

Tout entreprise a besoin d’une étude sur son niveau de stock d’une manière périodique, la connaissance de cette information a un impact sur toutes les composantes de la société dès le service de production vers le service de distribution. Ainsi d’après ses études on pourra toujours faire une estimation approximative sur les niveaux de stock et sur les bénéfices et aussi une définition large de stock de sureté.

Aujourd’hui, les grandes entreprises ont une base de données suffisantes sur les années précédentes, les achats, les dates et les niveaux de stock. Dans ce chapitre, On a pour objective la prédiction de stock des grandes entreprises comme Apple, IBM, Samsung … et faire une étude complète et une comparaison entre les moyens de prédictions par un réseaux de neurones récurrentes améliorés (LSTM) avec plusieurs model et faire une description bien détaillée sur les l’ensemble des résultats obtenues.

Notre modèle se base sur une base de données des plusieurs entreprises sur la valeur de stock d’après 2006 jusqu’à la fin de 2017. Notre objective est de prédire les niveaux de stock et leurs valeurs à l’année 2017 en se basant sur ce qu’ils sont précédés et donner une analyse sur le comportement de stocks durant cette période (2017).

La base de données de notre étude se trouve sur la plateforme Kaggle et représenté comme suivant :



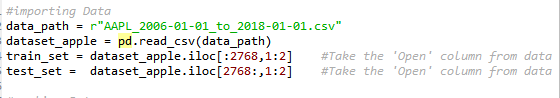
* 1. Les étapes de réalisation du projet.

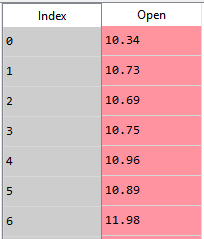
1-Préparation de la base de données.

Avant tout processus d’analyse, il faut faire une collection très sélective de la donnée et la nettoyer d’une manière adéquate à leurs utilisations. Dans cette étude on se basant sur les données de l’entreprise Apple «AAPL\_2006-01-01\_to-2018-01-01». Dans un premier temps, il faut séparer les données entre deux parties, la première concernant les données d’alimentation qui pourra aider notre model à entrainer, on l’appelle « training set » est doté de 2006 vers la fin de 2016, l’autre s’appelle « test set » c’est-à-dire les données réelles de l’année d’étude 2017.



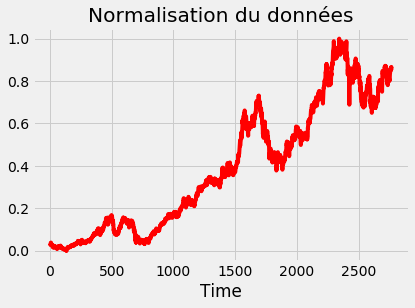
Dans notre donnée, on s’intéresse à « Open » colonne comme suit :







Après cette sélection, ne reste que normaliser les valeurs entre 0 et 1 à l’aide de la fonction MaxMinScaler :



Après la normalisation on arrive au choix de « time step ». Cette information nous permet d’entamer le processus d’entrainement. En fait, le model correspond à une boite noire et donc le nombre des inputs correspond au nombre de donnée dans le cas usuel, et dons plusieurs model on une base de données de test pour améliorer la performance du model. Dans notre cas le nombre des inputs nommé « time step ». Si par exemple notre donnée se constitue de 2768 lignes, on peut prendre 60 lignes pour prédire la 61émé ligne et ainsi de suite.

La dernière étape dans la présentation de la donnée et de faire une transformation matricielle correspond au « time step » convenable à notre étude pour un meilleur performance, cette transformation apparut comme suit :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | … | 79 |
| 0 | 0.0232156 | 0.0262847 | 0.0259699 | 0.0264421 | … | 0.0180216 |
| 1 | 0.0262847 | 0.0259699 | 0.0264421 | 0.0280948 | … | 0.0198316 |
| 2 | 0.0259699 | 0.0264421 | 0.0280948 | 0.0275439 | … | 0.0214055 |
| … | … | … | … | … | … | … |

Dans cette présentation, on choisit « time step » égale à 80 avec un seul indicateur c’est-à-dire la première colonne « Open ». Finalement notre préparation de données et terminé pour entamer l’étape de construction de la boite noire où notre data sera entrainer.

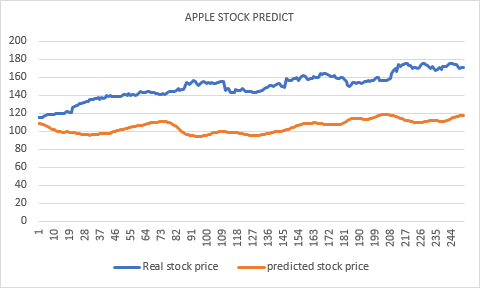
Dans cette étape, On va construire trois boites noires de même type et de différents paramètres, autrement dit, trois RNN améliorées avec un changement radical concernant les fondements du model. Les paramètres qu’on pourra jouer avec et les suivants :

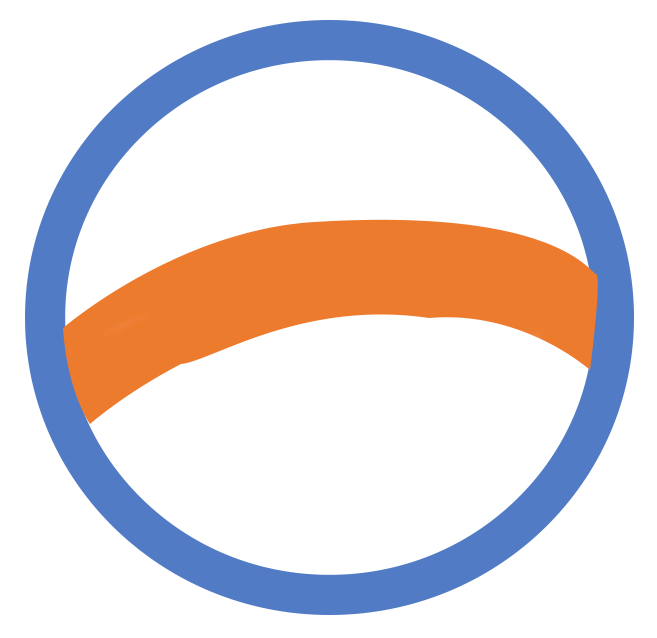
* Time step
* RNN vs RNN avec LSTM
* Les couches invisibles (Hidden layers)
* Dropout.

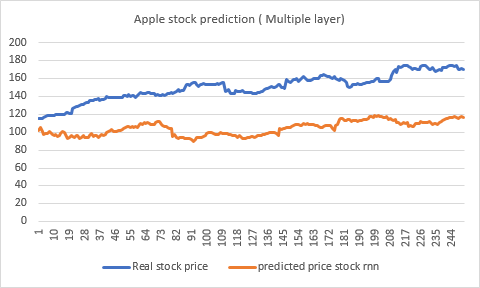
Finalement nos données sont prêtes après un dimensionnement rapide.

* 1. Représentation et analyse du résultat

Avant tous annonce de résultat, Il faut bien tester les paramètres non théoriques pour tirer la meilleure, et bien évident le paramètre est time step où les expériences prouvent que ce paramètre peut agir fortement sur notre résultat. Les expériences regardent que dans les tests les plus élevés, ce paramètre sera entre 60-100 et dans les moyennes teste ça dépend de la nature de la donnée, dans cette étude on fixe ce paramétré à 80, autrement dit, on va insérer 80 entrées dans cette boite noire. Les figures suivantes nous donnent une idée générale sur les observations préliminaires des modèles utilisées :







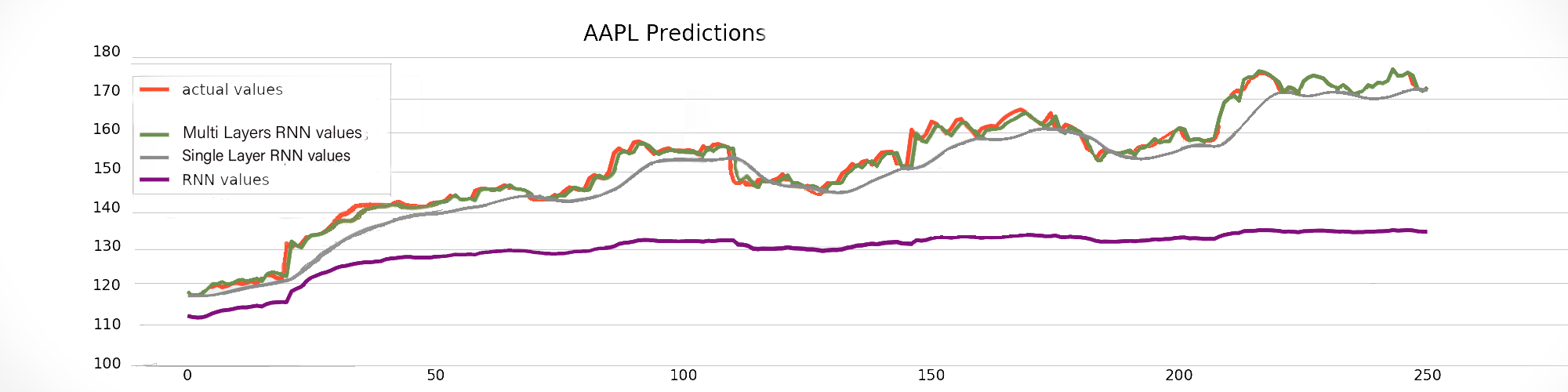


Ce graphe est construit par un LSTM « Stacked Layers » c’est-à-dire plusieurs LSTM l’un après l’autre, dans ce résultat, on va faire un raisonnement sur la linéarité du model, en répondant sur la question est ce que le modèle réagit sur tout changement d’état ?

Comme aperçut dans le graphe\* le model suit tous les changements d’états de stock résumé dans les segmentations faite dans le graphe\*, Ainsi il reste stable dans les segments linéaires même si les prédictions sont fortement différentes.

La chose qu’on peut sortir de la deuxième figure que tant de couches tant de précision au terme d’oscillation de model et de variation entre toutes les composantes de LSTMs couches.

Dans le cas où les couches sont parfaitement entrainées et les paramètres sont justement configurées, notre prédiction est la suivante :



Le résultat de notre modèle est résumé au-dessus, nous donne une comparaison entre les trois modèles testés :

* Dans le cas où le modèle prend plusieurs LSTM couches comme base de sa construction, on peut observer la similarité de la prédiction avec les valeurs actuelles, cette représentation donne une vue extrêmement globale sur le stock dans l’année suivantes et aussi nous permet de faire des décisions sensibles et serrées.
* Le second graphe grise a une réaction quasiment lente au changement de linéarité de graphe actuelle, mais il nous donne une approximation abordable et tolérable.
* Un RNN simple prouve que tant que le temps augmente, le modèle s’éloigné de la courbe actuelle, ceci est une conséquence des phénomènes cités au-dessus.

Conclusion générale :