Projet TSA : Estimation du prix d'une crypto-monnaie à l'aide de réseaux de neurones récurrents

DAVID Maxence

12 juin 2020

Introduction

Depuis plusieurs années, le domaine des crypto-monnaies prend de plus en plus d'ampleur dans la finance. A partir de 2017 les prix ont explosé, notamment celui du bitcoin qui a atteint les 20 000 euros. Il est important d'essayer de prédire le prix de ces nouvelles monnaies. Cependant nous devons noter que la prédiction de crypto-monnaie diffère des prédictions boursières classiques car ils dépendent de nombreux autres facteurs tels que la concurrence même entre les monnaies, mais aussi l'évolution de la block-chain qui influe grandement sur les prix des monnaies. Bien d'autres facteurs rentrent en compte dans l'évolution du prix de ces monnaies. Nous devons noter que le prix des crypto-monnaies est très volatile, donc trouver un modèle de prédiction fiable n'est pas une chose facile.

Dans ce projet nous allons essayer de créer un modèle de prédiction en utilisant un réseau de neurones récurrents. Plus précisement un réseau "Long Short-term Memory" (LSTM). Ces réseaux offrent une capacité de traitement très adaptée aux données de type séquentielles plus précisément temporelles. Ces réseaux s'appliquent bien à la prédiction du prix d'une crypto-monnaie.

Table des matières

Chapitre 1

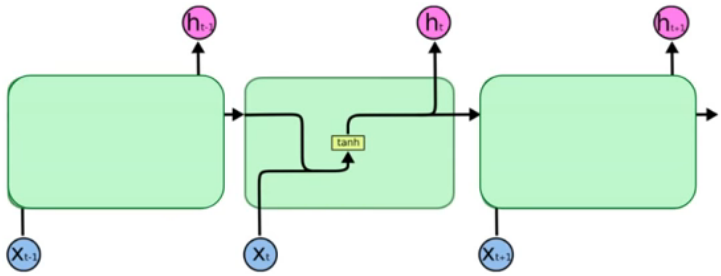
Les réseaux de neurones récurrents, RNN et LSTM

Un réseau de neurones récurrents, est composé de plusieurs couches de neurones appelées "layer", la particularité est que l'information peut se propager dans les deux sens. Ils se rapprochent plus du système neuronal humain. Ces réseaux possèdent des connexions récurrentes, car elles gardent des informations en mémoire : les réseaux récurrents peuvent prendre en compte des états du passé. Ils sont fortement utilisés dans le traitement des séquences temporelles comme lapprentissage et la génération de signaux. Notamment quand les données ne sont pas indépendantes les unes des autres. Néanmoins, lorsque les séquences sont longues (sur de longues durées), cette mémoire à court-terme n'est pas suffisante. En effet, les RNNs classiques ne sont capables de mémoriser que le passé proche. Il y a une perte d'information au bout de 50 itérations environ. C'est pour cela que d'autres méthodes ont été inventées. Notamment les LSTMs (Long Short Term Memory). Les réseaux LSTM sont notamment utilisés dans la reconnaissance de la voix et bien d'autres domaines.

1.1 Structure d'un RNN simple

Un RNN a pour particularité de diposer d'une mémoire à court terme. Ils sont donc utilisés dans le traitement de séquences. Voici un schéma d'une cellule RNN la plus simple possible.

FIGURE 1.1 – Schéma RNN



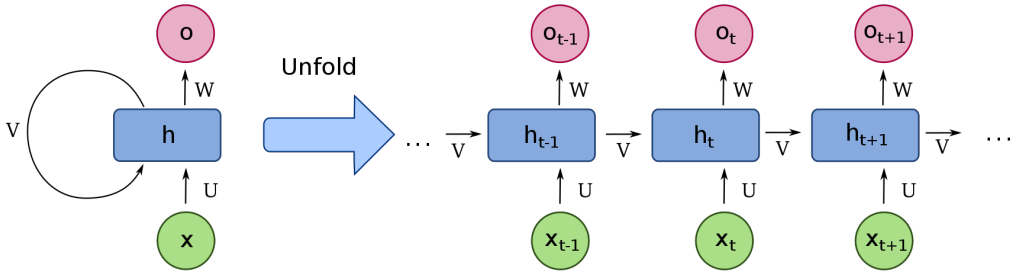
On remarque que la cellule est une suite d'opérations appliquées à un vecteur d'entrée : X_t, mais il est ajouté la sortie de la cellule précedente dans les calculs : h_{t-1}. Nous ne détaillons pas la structure profonde d'une cellule RNN, mais plutôt celle d'un LSTM par la suite (structure plus complexe).

1.2 Apprentissage des RNNs

Dans un réseau de neurones non récurrent, le méthode utilisée pour ajuster le poids des entrées est la rétropropagation du gradient. On commence donc à la derniere couche, l'une des méthodes qui peut être utilisée est la descente de gradient. On répète cette étape jusqu'à la première couche. Grâce à cette méthode, à chaque itération on ajuste les poids et on apprend au réseau à être plus proche du résultat attendu.

Maintenant dans le cas d'un RNN,une nouvelle difficulté apparaît : la présence de cycles dans le structure du neurone. Cela rend la rétropropagation du gradient décrite auparavant impossible. Mais il existe une alternative, la rétropropagation a travers le temps. Il est donc possible de prendre une approximation du réseau récurrent en le dépliant. Chaque couche représente alors l'état du réseau à un instant T. Dès lors on peut utiliser la rétropropagation du gradient à travers le temps et ajuster les matrices de poids normalement. Voici un schéma du dépliement d'un RNN :

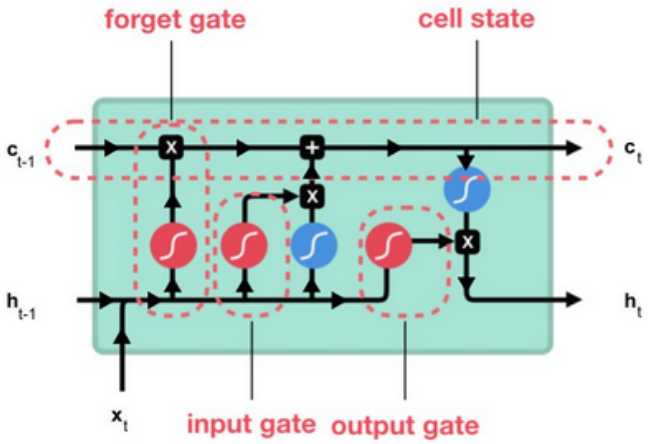
FIGURE 1.2 – Schéma d'un réseau de neurones récurrents. A droite la version dépliée de la structure.



1.3 Structure d'un Long Short Term Memory (LSTM)

Ce type de réseau résout le problème de mémoire des RNNs simples. Ils ont une mémoire dans laquelle il va être possible d'ajouter ou de retirer des informations. Voici un schéma détaillé d'une cellule LSTM :

FIGURE 1.3 – Schéma LSTM



Passons maintenant aux explications du fonctionnement. Comme on le remarque sur le schéma, la cellule du réseau LSTM est un neurone composé de plusieurs actions/opérations efféctuées à l'intérieur. On peut noter qu'il ne peut utiliser que 5 opérations :

1. La fonction sigmoïde
2. La fonction Tanh
3. Multiplication
4. Addition
5. Concaténation

Chacune des portes du réseau est pondérée par une matrice de poids, W_f pour la porte d'oubli,W_i pour la porte d'entrée,W_C pondère les données qui vont se combiner à l'entrée,W_o pondère l'entrée de la porte de sortie.

1.3.1 Forget gate

Cette porte a un rôle de décision, elle sélectionne les informations à garder ou à jeter. Plus précisement l'information de l'état précédent est concaténé avec l'entrée puis on y applique une sigmoïde pour la normaliser entre 0 et 1. Ensuite on décide : si le résultat est proche de 1 on garde l'information, sinon on l'oublie.

Les calculs réalisés :

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

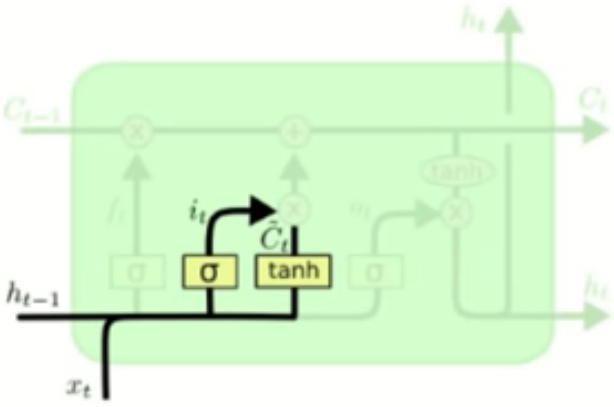
Puis on le multiplie par l'état précédent :

$$F_g = f_t * C_{t-1}$$

1.3.2 Input gate

C'est la porte d'entrée, elle a pour but d'extraire l'information des données courantes. Elle est résponsable de l'ajout d'information à la mémoire. Il y a une application en parallèle d'une sigmoïde aux données concaténées, ainsi que d'une fonction tanh.

FIGURE 1.4 – Schéma Input Gate



Voici les opérations :

$$\bar{C_t} = tanh(W_c * [h_{t-1}, x_t] + b_c)$$

Cette opération a pour but de proposer des informations à mettre dans la mémoire. Elle normalise les vecteurs entre $[-1, 1]$

$$i_t = sigma(W_i * [h_{t-1}, x_t] + b_i)$$

Cette opération a pour but de définir la quantité des nouvelles informations contenues dans C_t que l'on doit ajouter à la mémoire. Elle revoit un vecteur où les valeurs proches de 0 signifient que l'information n'est pas importante. A l'inverse les valeurs proches de 1 sont importantes pour la prédiction. En faisant le produit des deux, les informations non importantes seront remplacées par 0 et on ne garde que les informations importantes.
On a donc :

$$C_{new} = \bar{C_t} * i_t$$

1.3.3 Cell state

C'est l'état de la cellule avant d'aborder la sortie. Le calcul est simple :

$$C_t = F_g * C_{new}$$

1.3.4 Output gate

C'est la derniere étape d'un neurone LSTM, elle calcule et décide le prochain état caché h_t. Voici les calcules réalisés :

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$$

C'est la sélection des informations dont on a besoin dans la mémoire.
Et finalement :

$$h_t = o_t * tanh(C_t)$$

On a donc vu comment un LSTM peut avoir une mémoire, un état de la cellule, elle peut oublier des informations, en ajouter et grâce aux informations de la mémoire, définir celles qui ont de l'importance ou non. Ces cellules sont utilisées pour de la génération de texte, de dessins, mais aussi dans notre cas la prédicition du prix d'une crypto-monnaie.

Chapitre 2

Application à la prédiction du cours d'une crypto-monnaie

La prédiction du prix d'une crypto-monnaie se rapproche grandement de la prédiction de cours de la bourse. Les paramètres qui influent sur les variations sont très nombreux. De plus il est à noter que le crypto-monnaie est beaucoup plus volatile que la bourse. On peut donc se demander quels outils existent déjà pour la prédiction dans ce domaine. Ce sont des devises totalement virtuelles qui ne sont pas soumises à des banques mais à des technologies de transfert totalement décentralisées et en ligne par le réseau de la block-chain. Apparues en 2017, avec le bitcoin, les cours ont connu un pic à 20 000 $, à ce moment les investiseurs se sont alors intéressés à ces monnaies.

C'est donc ici qu'intervient la prédiction des cours. De nombreuses techniques sont possibles. On peut notamment utilser la technique " Rolling Linear Regression " qui est un régression qui prédit $X(t+1)$ en utilisant $x(t), x(t-1), x(t-2), ...$ Cepandant cette technique n'est pas très précise et les résultats sont peu satisfaisants. Il y a aussi des techniques d'apprentissage utilisant des arbres de décision, des adaboosts ou encore des Forêts aléatoires. Dans notre cas nous allons utiliser la prédiction avec un réseau de neurones récurrents de type LSTM, permettant d'oublier des informations non pertinentes.

2.1 Présentation du jeu de données

Le jeu de données est un jeu extrait du site de traiding de crypto-monnaie : Cryptocompare (`https://www.cryptocompare.com/`)

Ce site enregistre les différentes variations des prix et de nombreux autres indicateurs pour un grand nombre de monnaies. Il permet notamment de récupérer ces données en temps réel pour en réaliser un traitement

Pour ce projet nous avons choisi de recupérer les données chaque jour sur les 2000 derniers jours. A noter que ce projet est générique, si vous souhaitez faire de la prédiction sur le prix par heure ou sur une autre monnaie il vous suffit de changer le endpoint et le request de cette partie du code :

```
1       endpoint = 'https://min-api.cryptocompare.com/data/histoday'
2       res = requests.get(endpoint + '?fsym=BTC&tsym=USD&limit=2000')
3       hist = pd.DataFrame(json.loads(res.content)['Data'])
```

Vous pouvez aussi changer le nombre de data utilisé.

Voici une description des différentes variables :

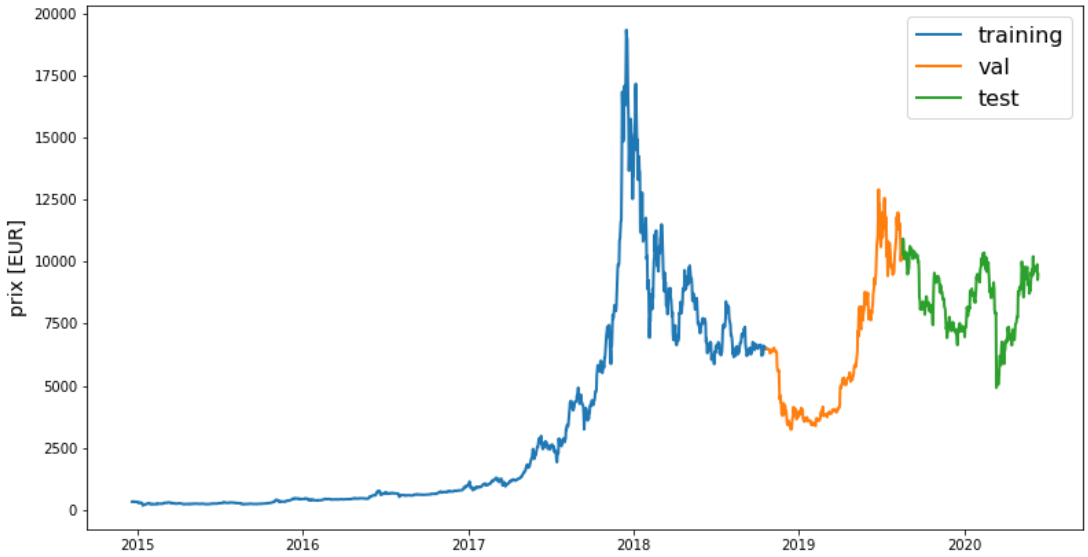
Variable	Déscription	Unité
Time	L'horodatage Unix de la donnée.	float
High	prix le plus élevé de la monnaiependant cette période.	float
Low	prix le plus bas de la monnaie pendant cette période.	float
Open	prix de la monnaies à l'ouverture de cette période.	float
Volumefrom	montant total de la devise échangée dans la devise de cotation .	float
Volumeto	montant total de la deviseéchangée dans la devise de cotation.	float
Close	Le prix de la monnaies demandée à la fin de cette période.	float

Nous allons essayer de faire des predictions du prix de la crypto-monnaie en fonction de 5 paramètres au cours du temps.

2.2 Pré-traitement

Voici une representation graphique du prix au cours du temps : les données sont séparées en 3 zones. Une zone d'apprentisage, une zone de validation et une zone de test.

FIGURE 2.1 – Graphique des données dans le temps



Les données ne peuvent pas être utilisées sans les arranger. C'est pour cela que nous réalisons une phase de normalisation très basique. Dans un second temps nous créons notre vecteur de features en créant des groupes de 10 valeurs. Cela permet de donner en entrée du réseau des séquences et pas seulement une valeur à un instant t. Enfin nous adaptons les targets et les normalisons pour qu'elles correspondent au vecteur des features et aient toutes la même base. Voici le code des trois fonctions, de normalisation et de pré-traitement utilisé dans notre cas :

```
1     #normalisation par rapport à la base 0
2     #permet de mettre toute les valeurs sur une base commune.
3     def normalise(df):
4         return df / df.iloc[0] - 1
```

```
    #Extraction des données par fenêtres de taille "window_len", pour un
    apprentissage par groupe.
    def extract_window_data(df, window_len):
        window_data = []
        for indice in range(len(df) - window_len):
            temp = df[indice:(indice + window_len)].copy()
            temp = normalise(temp)
            #copi des valeur de temp dans windows_data
            window_data.append(temp.values)
        return np.array(window_data)
```

```
    #Préparation des données pour l'apprentissage
    #Séparation en 3 ensembles, préparation des vecteurs de features et targets
    def prepare_data(df, target_col, window_len=10, val_size=0.2,test_size=0.5):
        train_data, test_data = train_test_split(df, test_size=val_size)
        val_data, test_data =train_test_split(test_data, test_size=test_size)
        X_train = extract_window_data(train_data, window_len)
        X_val = extract_window_data(val_data, window_len)
        X_test = extract_window_data(test_data, window_len)
        y_train = train_data[target_col][window_len:].values/ train_data[
    target_col][:-window_len].values - 1
        y_val = val_data[target_col][window_len:].values/ val_data[target_col
    ][:-window_len].values - 1
        y_test = test_data[target_col][window_len:].values/ test_data[target_col
    ][:-window_len].values - 1

        return train_data,val_data, test_data, X_train,X_val, X_test, y_train,
    y_val, y_test
```

2.3 Implémentation

Pour l'implémentation nous avons fait le choix de prendre le langage de programmation Python. Nous utilisons notamment le framework Tensorflow dans la version 2.0 ainsi que keras qui est une bibliothèque qui permet d'intéragir facilement avec les algorithmes de réseaux de neurones profonds et de machine learning.

Le réseau créé est séquentiel, il est composé d'une première couche de neurones LSTM avec un dropout qui permet de désactiver un pourcentage de neurones à chaque epoch pour éviter le surapprentissage. Enfin cette couche est suivie de neuronnes plus simples à propagation vers l'avant. Voici le code de constuction du modèle :

```
    #Construction du model avec la bibliothéque KERAS
    #La fonction est générique et ne fixe que la structure du réseau
    #Le nombre de neurones, le droupout, ect restent paramétrables.
    def build_lstm_model(input_data, output_size, neurons, activ_func, dropout,
    loss, optimizer):
        model = Sequential()
        model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.
    shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))
        model.compile(loss=loss, optimizer=optimizer)
        return model
```

On a donc les caractéristiques suivantes :

FIGURE 2.2 – Caractéristiques du réseau

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 20)	2160
dropout_3 (Dropout)	(None, 20)	0
dense_3 (Dense)	(None, 1)	21
activation_3 (Activation)	(None, 1)	0

Ce réseau n'est pas très complexe, il n'est pas composé de beaucoup de layers. Cependant il utilise des cellules complexes, il est donc efficace.

2.4 Choix des hyper-paramètres et apprentissage

En réalisant plusieurs tests, les hyper-paramètres suivants nous ont permis d'avoir de très bons résultats :

```
#Hyper-paramètres utilisés pour la prédiction
activ_func = 'linear'
lstm_neurons = 20
epochs = 20
batch_size = 32
loss = 'mse'
dropout = 0.5
optimizer = 'adam'
```

Nous avons donc choisi de réaliser un apprentissage en 20 epochs, cela semble suffisant pour résoudre le problème de prédiction. Il y a donc 20 neurones LSTM, avec un dropout a 50% à chaque epoch. Pour le calcul de la perte on utilise le paramètre 'mse' de keras, c'est l'erreur quadratique moyenne. Le calcul est le suivant :

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prdicition(x))^2$$

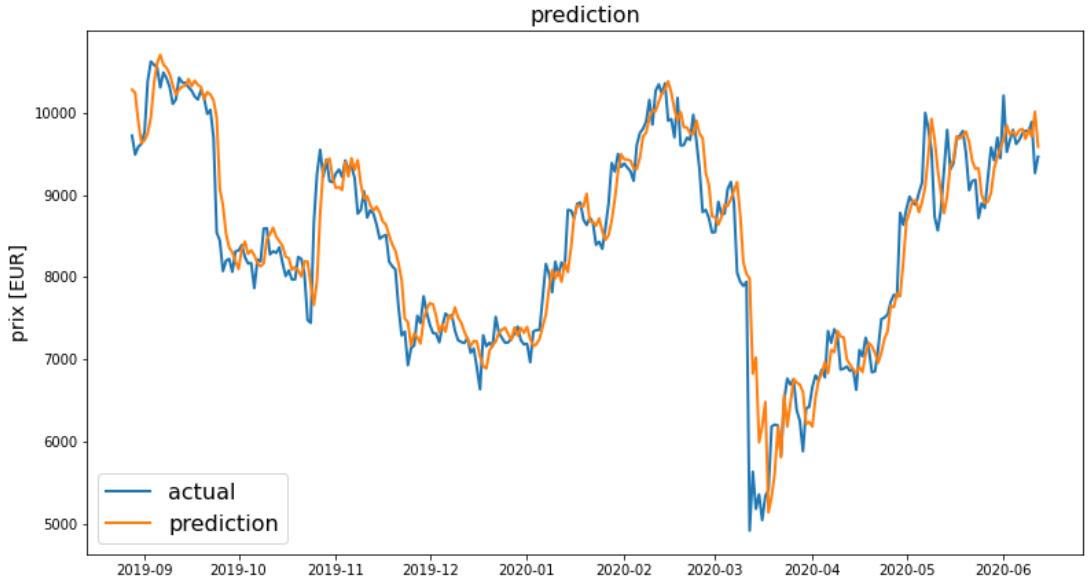
Cette fonction de perte est adaptée aux problèmes de prédiction. Enfin, l'optimiseur Adam semble être le plus utilisé pour ce type de probème, c'est une méthode de descente de gradient basée sur une estimation adaptative des moments du premier et du second ordre. Il sert à ajuster les poids du réseau pendant l'apprentissage. Voici donc le code d'apprentissage :

```
#Constuction et entrainement (.fit) du model
model = build_lstm_model(X_train, output_size=1, neurons=lstm_neurons,
activ_func=activ_func,dropout=dropout, loss=loss,optimizer=optimizer)
fit = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
verbose=1, shuffle=True, validation_data = (X_val,y_val))
```

2.5 Résultats et interprétations

Notre réseau est maintenant entraîné à la prédiction du prix d'une crypto-monnaie. On lance donc notre test sur les données réservées à cet effet. En calculant l'erreur moyenne absolue on trouve environ 3,3% d'erreur. Cela nous paraît être un résultat très satisfaisant. Nous pouvons illustrer ce résultat dans un graphique montrant les prédictions et les valeurs réelles.

FIGURE 2.3 – Résultat de la prédiction sur la zone de test



Ce visuel nous montre que le réseau se comporte très bien sur des données qu'il n'a jamais vues auparavant. Les courbes ne se superposent pas, ce qui est totalement normal, mais sont très proches. Malgré un décalage récurrent des prédictions sur la droite, on peut donc voir que la prédiction est un peu en retard. Si une grosse chute du prix se prduit brusquement le réseau ne le prédira sûrement pas. On peut donc en conclure que malgré la complexité des données et le fait qu'elles dépendent de nombreux facteurs, un réseau de neurones bien entrainé peut réusir la prédiction dans le futur de façon plus que convenable. Cela montre la puissance des RNNs et plus précisement des LSTMs, on peut se demander si un résultat similaire aurait pu être obtenu avec un perceptron multicouches ou un autre moyen de prédiction.

Nous aurions pu choisir des données par heure pour avoir une prédiction sûrement plus précise au cours d'une journée.

Conclusion

Pour conclure ce projet fut très enrichissant pour moi. Il portait sur deux domaines qui m'intéressent énormément : les data-science et le monde de la finance. J'ai donc pu découvrir, et apprendre par mes propres moyens le fonctionnement d'outils de machine learning très intéressants. Ce projet m'a permis de comprendre que le domaine des data est celui qui m'attire le plus dans la formation proposé dans le département ITI de l'INSA. J'ai donc hâte de suivre ces cours en 4eme année.